CSE 587: DATA INTENSIVE COMPUTING

PROJECT PHASE 2

Introduction:

- Analyse the bank loan lending data and predict the chances of customers defaulting on their loans.
- Identify the loan applicants who are at risk of defaulting on their loans so that the banks can see to that the applications of these applicants are carefully monitored before approving loans.
- Identify the loan applicants who can repay the loan and hence banks can gain profit.

Features of the dataset:

loan_amnt float64
term int8
int_rate float64
installment float64
grade int8
emp_length int8
home_ownership int8
annual_inc float64
verification_status int8
purpose int8
addr_state int8
dti float64
open_acc float64
pub_rec float64
revol_bal float64
revol_util float64
total_acc float64
pub_rec_bankruptcies float64

Target Feature: Loan Status – Loan status is the target feature of the dataset that gives the final information regarding the repayment status of the loan.

The above problem is a classification problem, and several Machine Learning algorithms are applied to get the results. Furthermore, this problem is a binary classification problem where we are trying to predict the repayment capacity of the loan applicants. The output of the applied machine learning algorithms is either 1 or 0 which denotes the status i.e fully paid or charged off. If the outcome of the ML algorithm is

- 1 – then the loan is likely to be repaid
- 0 – then the loan is likely to be defaulted on.

Pre-Processing before applying ML Algorithms:

Though most of the data cleaning and preprocessing was done as part of phase 1, before fitting the data to the machine learning algorithms, min max scaling was performed to avoid bias. The feature **revol_bal** has uneven data which can negatively affect the outcome of the ML algorithms and hence it is scaled such that it equally contributes to the model fitting and model prediction. Min Max scaling is a technique where all the values of that feature are scaled in such a way that the minimum value is 0 and the maximum value is 1.

```
In [135]: columns = ['revol_bal']
          for column in columns:
              max = df_default_loan_copy[column].max()
              min = df_default_loan_copy[column].min()
              print("max of column", column, "is", max)
              print("min of column", column, "is", min)
              df_default_loan_copy[column] = (df_default_loan_copy[column] - min) / (max - min)

          max of column revol_bal is 1207359.0
          min of column revol_bal is 0.0
```

MACHINE LEARNING ALGORITHMS:

1. ## DECISION TREES:

Decision Tree is a classification algorithm where the data is organized in a tree like structure, and it splits the input data into classes. Here in our data set we want the outcome to belong to one of the classes i.e., 1 or 0. The decision tree continuously narrows down the feature space and applies rules at each split until the input data is classified into required outcomes. Decision trees are used because they perform well for all types of data with various kinds of datatypes. In our dataset we have various kinds of datatypes for various features and hence decision tree is used. Also, decision trees consider each feature at a time while making decisions instead of considering all the features at once and hence contribution by each feature is equal.

Train-Test Split- Before fitting the model, the input data which has 39455 is split into training and testing data where 70% training data is used for fitting the model and the rest 30% is used for testing the accuracy of the model. The more the training data, the better the model fits and better the predictions and hence 70% of the input data is used for training the model.

A Decision Tree classifier has parameters like

criterion/loss function - The function to measure the quality of a split.

max_depth - The maximum depth of the tree

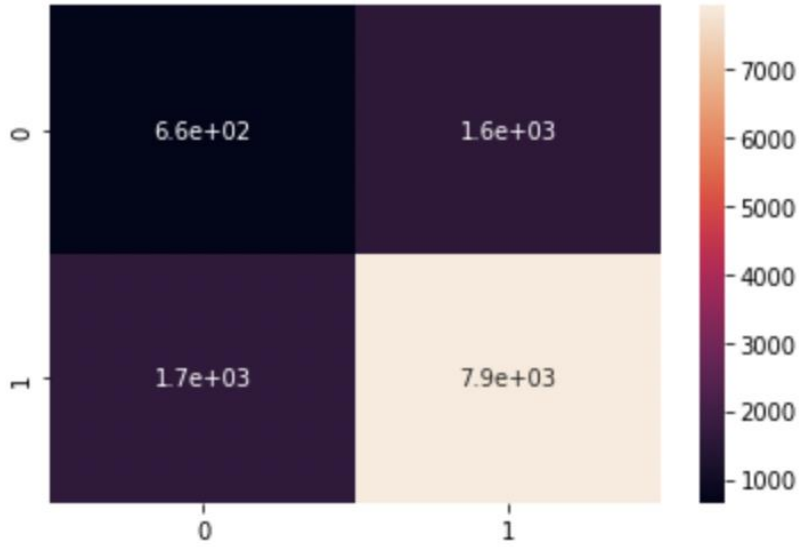
splitter - The strategy used to choose the split at each node

A Model is fit with default parameters of the Decision Tree classifier where criterion = gini and splitter = best. Training and testing accuracy is achieved as below

```
Test accuracy 0.7247613415561376
Train accuracy 1.0
```

Since training accuracy is more than the testing accuracy, the model is overfitted.
Confusion matrix is calculated for this model and a heatmap is plotted for the same.

```
[[ 656 1584]
 [1674 7923]]

<AxesSubplot:>
```



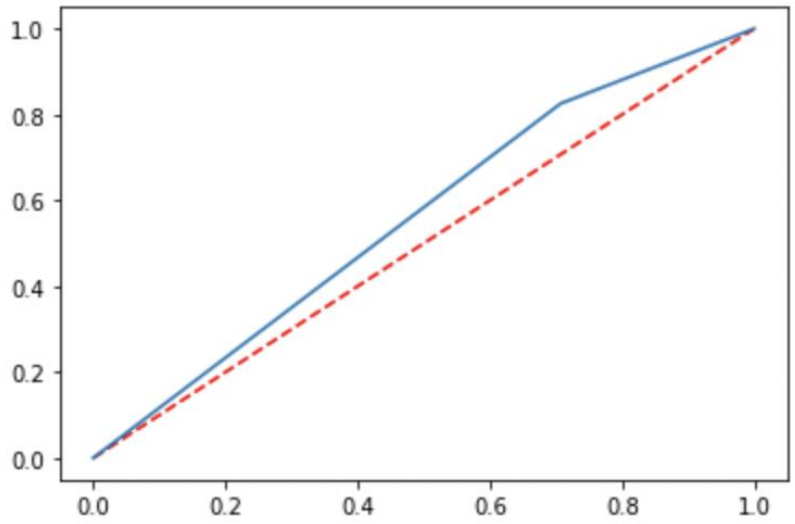
The model has high true positives and almost an equal number of false positives and false negativities.

The classification report for this model is

	precision	recall	f1-score	support
0	0.28	0.29	0.29	2240
1	0.83	0.83	0.83	9597
accuracy			0.72	11837
macro avg	0.56	0.56	0.56	11837
weighted avg	0.73	0.72	0.73	11837

This model has good precision and recall for class 1 - the predictions of candidates repaying the loan. However, the model has low precision and recall for the class 0 - predictions of loan defaulters.

ROC-AUC is another method of predicting the performance of the model where ROC stands for Receiver Operator Characteristics and AUC stands for Area under the curve. This measure is used to find the model is capable of distinguishing between the different outcomes of the data. It is plotted against false positive rate and true positive rate at various thresholds. ROC-AUC curve is plotted and the outcome is as follows.



The Area under the curve is very low and roc-auc score for this model is 55.921

Decision Tree Hypertuning:
GridSearchCV is used to find the best params for training the model and the best params are chosen among {'max_depth': np.arange(1,25),'criterion':['gini','entropy'],'splitter':['best', 'random']}
After performing the GridSearchCV the best params are found which are
{'criterion': 'gini', 'max_depth': 6, 'splitter': 'best'}.

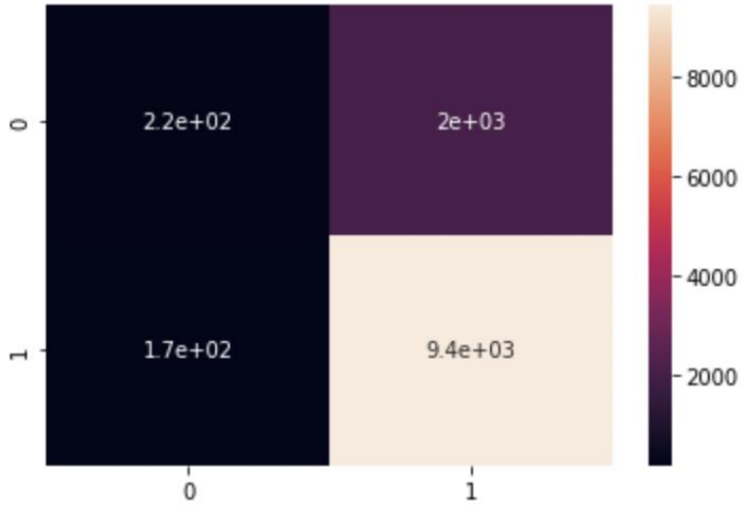
Decision Tree is again trained with these best params, and accuracy is calculated.

```
decision_tree_hp=DecisionTreeClassifier(max_depth=6,criterion='gini',random_state=10,splitter='best')
decision_tree_hp.fit(x_train,y_train)
test_predict=decision_tree_hp.predict(x_test)
train_accuracy=decision_tree_hp.score(x_train,y_train)
test_accuracy=decision_tree_hp.score(x_test,y_test)
print ('Test accuracy',test_accuracy)
print ('Train accuracy',train_accuracy)
```

```
Test accuracy 0.814902424600828
Train accuracy 0.8263089289593744
```

A significant change in testing accuracy is observed.
Confusion Matrix for this hyper tuned model also has significant changes. There is an increase in the number of true positives and decrease in the number of false negatives.

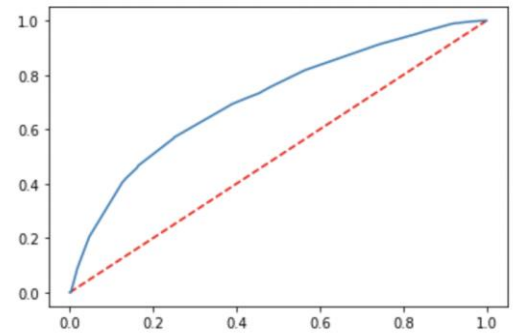
```
[[ 218 2022]
 [ 169 9428]]

<AxesSubplot:>
```



If we look at the classification report for this hypertuned model we can see that there is an increase in the precision and recall for the class 0.

	precision	recall	f1-score	support
0	0.56	0.10	0.17	2240
1	0.82	0.98	0.90	9597
accuracy			0.81	11837
macro avg	0.69	0.54	0.53	11837
ighted avg	0.77	0.81	0.76	11837

If we plot the ROC AUC curve, we can see that the area under the graph has increased



The ROC- Auc score is 71.2 which has also significantly increased which means that it is able to distinguish between classes.

2. ## LOGISTIC REGRESSION:

Logistic Regression is commonly used for solving binary classification problems. Since loan default prediction is a binary classification problem, logistic regression is used as it provides efficient results. Logistic regression works on the principle of finding the relationship between dependent variable and independent variable. Here, the dependent variable is the **loan_status** and the independent variables are the set of features. Logistic regression is also used for huge data and since our dataset contains large number of rows, this algorithm is used.

Train Test Split – 45% is used as test data and 55% of training data is used to fit the model.

Logistic regression has following parameters:
max_iter- Maximum number of iterations taken for the solvers to converge.
Solver - Algorithm to use in the optimization problem.
random_state – Used to shuffle the data.
Penalty – form of regularization
fit_intercept – specifies if constant should be added to decision function

A logistic regression model is fitted, and the results are observed using the parameters below.

```
logistic_reg=LogisticRegression(max_iter=10000,
                                solver='saga',
                                random_state=1234,C=10000,
                                penalty='l2',fit_intercept=True,intercept_scaling=1)
logistic_reg.fit(x_train,y_train)
test_score=logistic_reg.score(x_test,y_test)
print("Test Accuracy",test_score)
train_score=logistic_reg.score(x_train,y_train)
print("Train Accuracy",train_score)
```

```
Test Accuracy 0.8094621233455365
Train Accuracy 0.817926267281106
```

Here, the solver 'Saga' is used because it performs faster for large datasets and since our dataset is large enough, this solver is used. Penalty l2, which is l2 regularization is used for this solver. The max_iterations is set to 10000 as the dataset is huge and it will not allow the algorithm to converge faster.

The model is slightly overfit and hence the model is hyper tuned with other parameters.

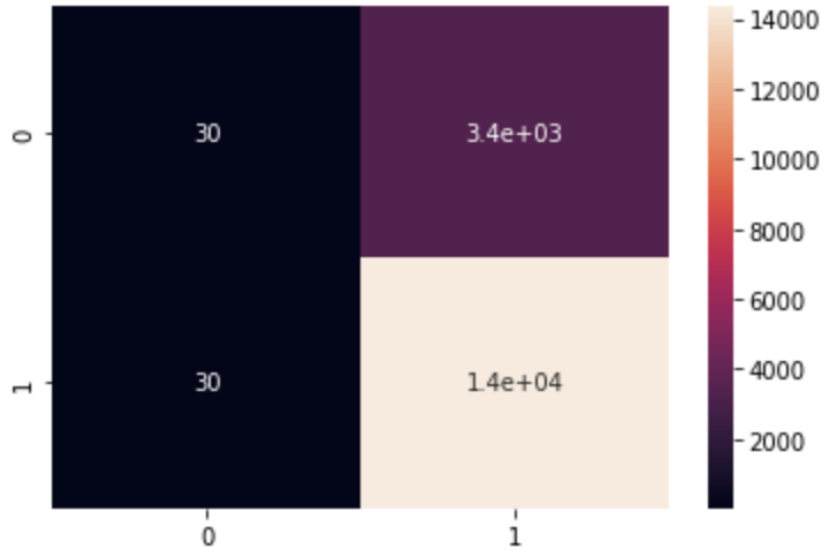
```
logistic_reg=LogisticRegression(max_iter=1000,
                                solver='lbfgs',
                                random_state=1234,C=10000,
                                penalty='l2',fit_intercept=True,intercept_scaling=1)
logistic_reg.fit(x_train,y_train)
test_score=logistic_reg.score(x_test,y_test)
print("Test Accuracy",test_score)
train_score=logistic_reg.score(x_train,y_train)
print("Train Accuracy",train_score)
```

```
Test Accuracy 0.8094621233455365
Train Accuracy 0.8172350230414747
```

Here, the model is trained with the solver 'lbfgs' which performs well with binary classifications. We can see some change in the accuracies.

Confusion matrix and classification report for this model is below

```
[[   30  3353]
 [   30 14342]]
```



	precision	recall	f1-score	support
0	0.50	0.01	0.02	3383
1	0.81	1.00	0.89	14372
accuracy			0.81	17755
macro avg	0.66	0.50	0.46	17755
weighted avg	0.75	0.81	0.73	17755

The model is able to predict a good number of candidates who can repay the loan but performs badly for the prediction of defaulters.

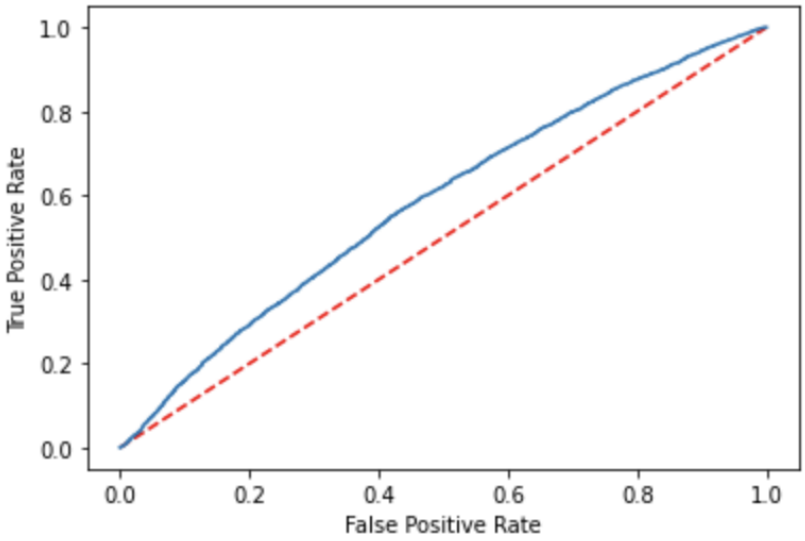
Sensitivity and Specificity is calculated for the model.

```
true_positive=confusion_matrix_logistic[1,1] #predicted positive actual positive
true_negative=confusion_matrix_logistic[0,0] #predicted negative actual negative
false_positive=confusion_matrix_logistic[0,1] #predicted positive actual negative
false_negative=confusion_matrix_logistic[1,0] #predicted negative actual positive
specificity=true_negative/(false_positive+true_negative)
sensitivity=true_positive/(true_positive+false_negative)
print("Specificity",specificity)
print("Sensitivity",sensitivity)
```

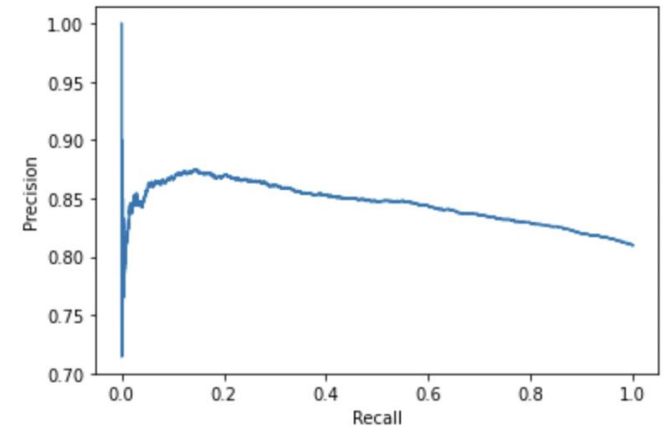
```
Specificity 0.008867868755542418
Sensitivity 0.9979126078485945
```

It is observed that the model is more sensitive than specific. It means that it can predict the positive values more accurately than the negative ones.

ROC-AUC curve is plotted for class 1, and results are observed as below

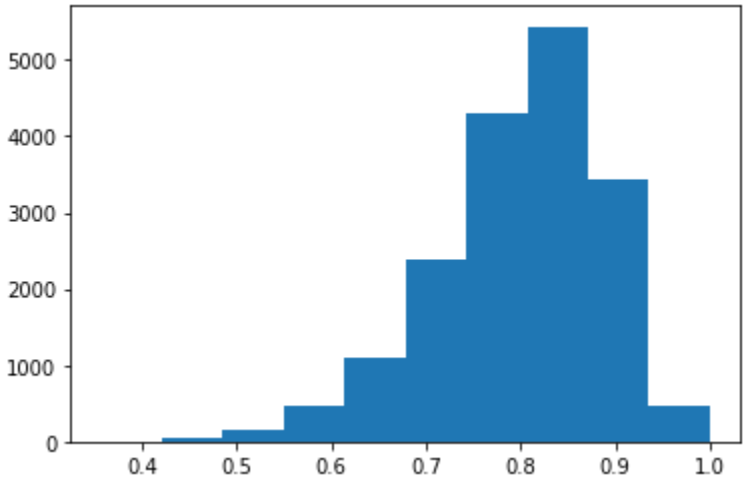


The area under the curve is low which means that the model is unable to classify the 2 classes correctly. The model is incorrectly predicting the outcome of the data for some thresholds.
The ROC AUC score for this model is - 58.43.
Precision Recall Curve is plotted for class 1.

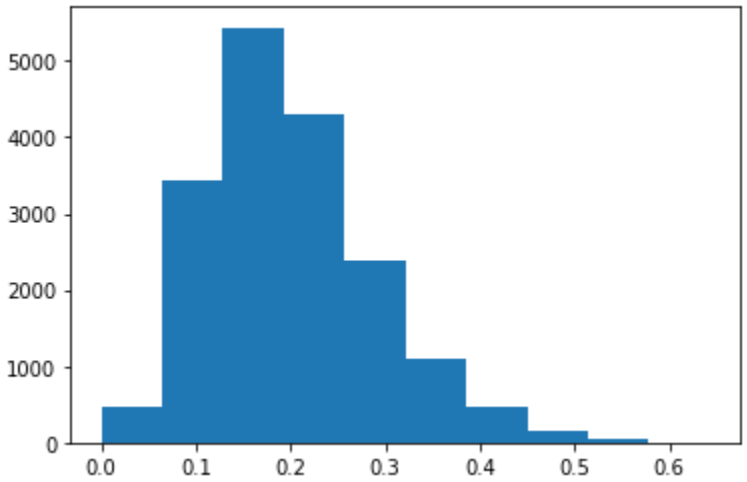


The probability distributions of the class 1. Here we can see that the greatest number of rows have the

probability distribution 0.6-0.9



The probability distributions of the class 0. Here we can see that the greatest number of rows have the probability distribution 0.1-0.3.



The given dataset is imbalanced, which means that there are a greater number of samples for the candidates who have paid the loan and there are only a few records of the candidates who have defaulted on the loan.
Sampling technique is used to overcome the imbalance dataset.

Random sampling technique is used where the samples with low number are duplicated randomly. Here, we around 7000 samples for the candidates who have defaulted on the loan and 32000 samples for the candidates who have repaid the loan. Random sampling is used where 15000 samples are generated from 7000 samples of candidates who have defaulted on the loan.

```
oversample = df_default_loan[df_default_loan['loan_status'] == 0].sample(15000, replace=True)
df_oversample = pd.concat([oversample, df_default_loan[df_default_loan['loan_status'] == 1]], axis=0)
print("value counts after oversampling",df_oversample['loan_status'].value_counts())
```

```
value counts after oversampling 1    32121
0    15000
Name: loan_status, dtype: int64
```

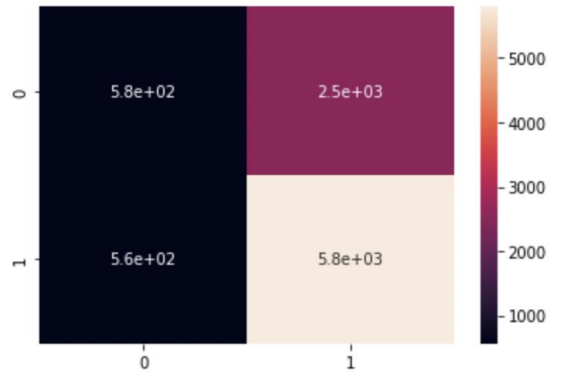
After resampling the dataset, logistic regression is performed, and the following results are observed:

```
logistic_reg=LogisticRegression(max_iter=1000,
                                solver='lbfgs',
                                random_state=1234,C=10000,
                                penalty='l2',fit_intercept=True,intercept_scaling=1)
logistic_reg.fit(x_train,y_train)
test_score=logistic_reg.score(x_test,y_test)
print("Test Accuracy",test_score)
train_score=logistic_reg.score(x_train,y_train)
print("Train Accuracy",train_score)
```

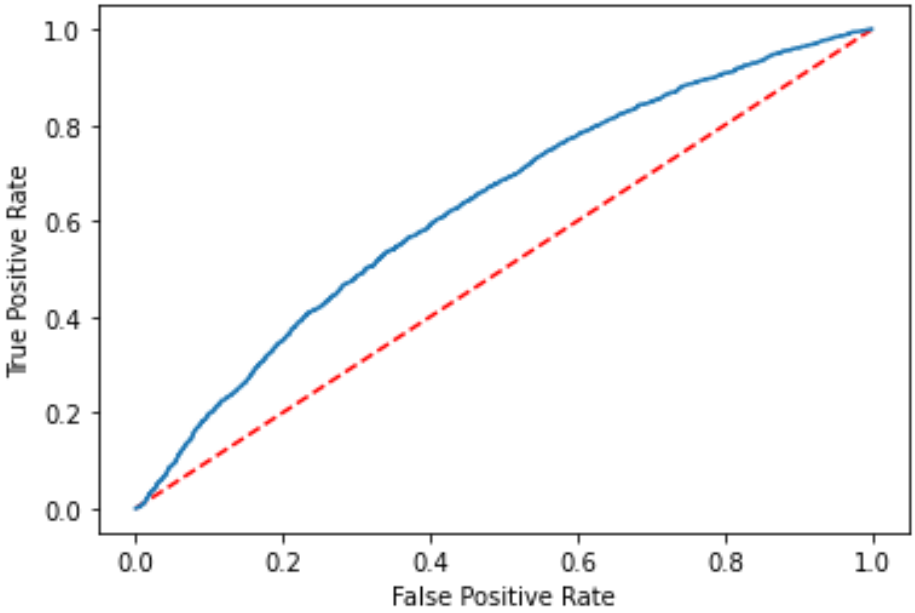
```
Test Accuracy 0.676498673740053
Train Accuracy 0.6840513582342954
```

The overall accuracy of the model is decreased, but it is observed that it is predicting well for negative class.

```
[[ 578 2489]
 [ 560 5798]]
```

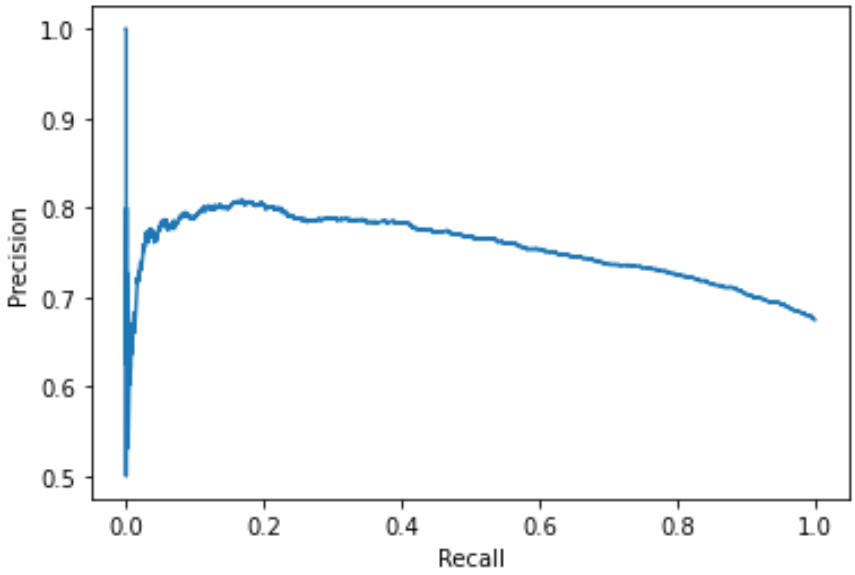


Here we can observe that the model is able to predict 578 true negative values unlike the earlier model which was able to predict only 30.



The area under the curve is also increased and the ROC-AUC accuracy is 63.16 which is greater than the previous model.

The Precision Recall graph for class 1 for the oversampled model is. Precision is almost constant.



3. NAIVE BAYES

Naïve Bayes is a classification algorithm that is based on the bayes theorem. It is known that naïve bayes algorithms perform faster and are more efficient on large datasets. Additionally, naïve bayes algorithms require only less training data for getting results. In naïve bayes, there is an independence of features which means that correlations need not exist between features in the dataset. In our loan default dataset, the correlation among various features is very less and hence this algorithm is used to get better results. Naïve bayes is highly scalable and can perform well with large data.

Train-Test split – since naïve bayes requires less training data, we are splitting it into 45-55 ratio where 45% of input data is test data and 55% of input data is training data.

Gaussian Naïve Bayes is used to fit the model with parameters
priors- Prior probabilities of the classes
var_smoothing - Portion of the largest variance of all features that is added to variances for calculation stability.

Model is fit with default values of the Gaussian Naïve Bayes:

```
class NaiveBayes:
    def __init__(self):
        self.naive_bayes=GaussianNB()
    def fit(self,x_train,y_train):
        self.naive_bayes.fit(x_train,y_train)
    def predict(self,x_test):
        return self.naive_bayes.predict(x_test)
    def score(self,x_test,y_test):
        return self.naive_bayes.score(x_test,y_test)
    def predict_proba(self,x_test):
        return self.naive_bayes.predict_proba(x_test)
```

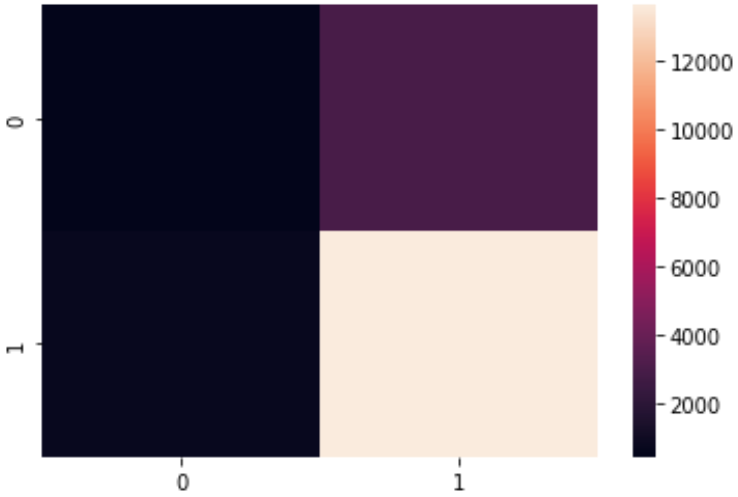
```
naive_bayes=NaiveBayes()
naive_bayes.fit(x_train,y_train)
test_predict_naive_bayes=naive_bayes.predict(x_test)
print(naive_bayes.score(x_test,y_test))
```

```
0.792114897212053
```

The accuracy of the model is 79% which is almost equal to both the above algorithms.

The confusion matrix for this model is

[[428 2955]
[736 13636]]



Here we can see that model is able to predict most of the positive outcomes, i.e., there are a greater number of true positives and almost equal number of true negatives and false negatives.

The classification report for this model:

	precision	recall	f1-score	support
0	0.37	0.13	0.19	3383
1	0.82	0.95	0.88	14372
accuracy			0.79	17755
macro avg	0.59	0.54	0.53	17755
weighted avg	0.74	0.79	0.75	17755

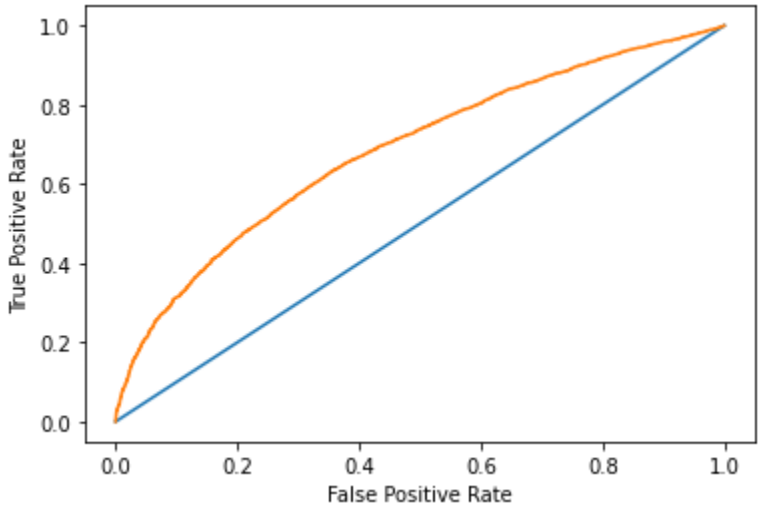
It has higher precision and recall for class 1 label and low precision and recall for class 0 label, which means the model is correctly able to predict the candidates who can repay the loan.

Specificity 0.12651492757907182

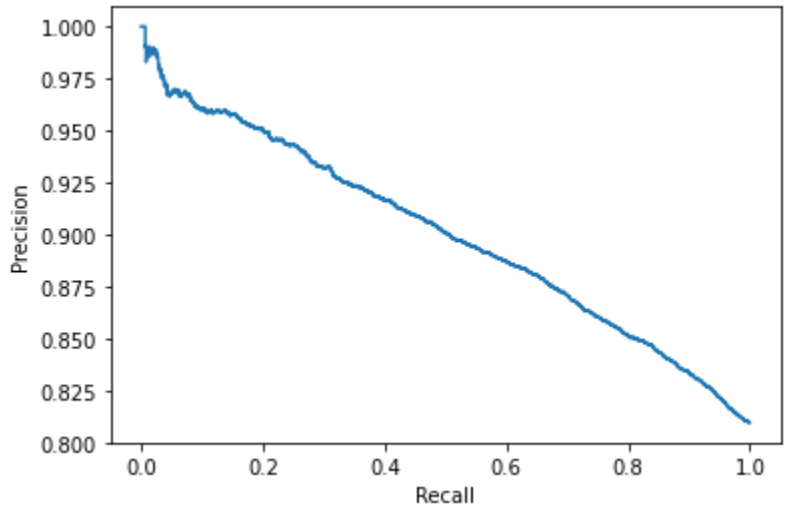
Sensitivity 0.9487893125521848

The model is highly sensitive than specific, it can predict positive results.

ROC-AUC CURVE



The ROC-AUC curve for the positive class label is plotted and it is observed that the area under the curve is more than what was achieved in logistic regression. The ROC-AUC score is 68.5 which is also better than logistic regression.



For each threshold, for the positive class label, the precision recall curve is as above.

Model -2

By increasing the training data samples, it is assumed that the accuracy of the model is also increased.

We have increased the size of the training set and observed the following results.

Training data size - 80% of input data

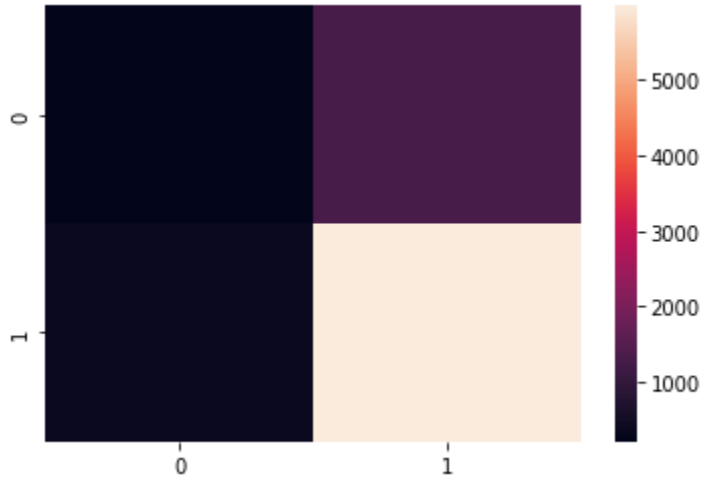
Test data size - 20% input data

x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size = 0.20, random_state = 55,shuffle=True)

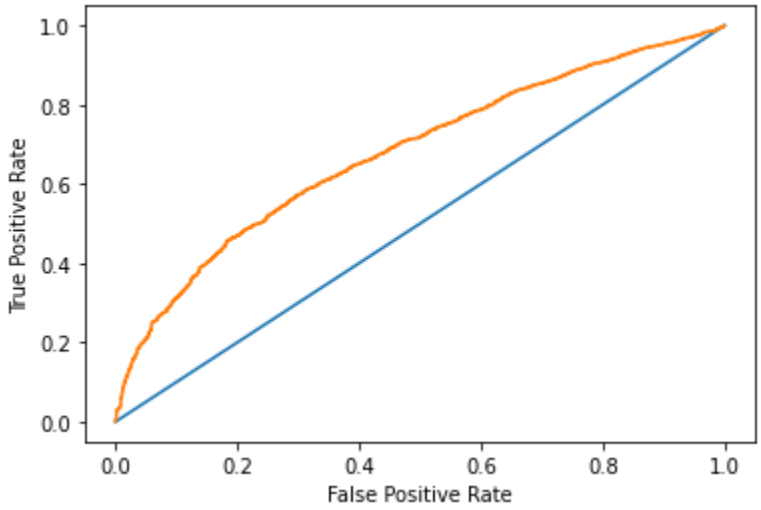
We have observed that the accuracy is 78.48% which is little less than the previous model where the training data size was 55%. It can be inferred that Naive Bayes can perform well with less training data.

The confusion matrix for this model is as below: It is observed that the number of true negatives were decreased when compared to the previous model.

[[214 1308]
[390 5979]]

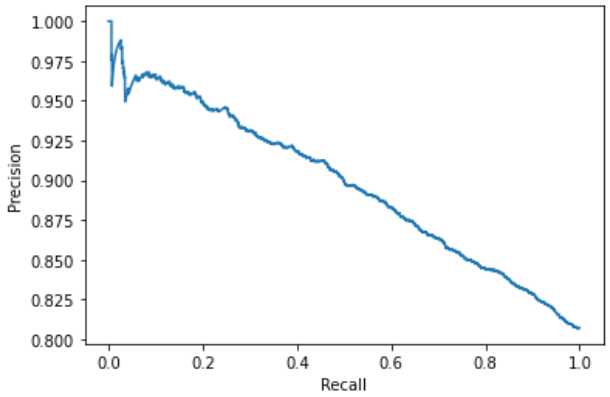


ROC-AUC Curve



ROC-AUC score - 67.82

Precision Recall Curve



Hyper Tuning - Naive Bayes:

GridSearchCV is performed on naive bayes and best params are found

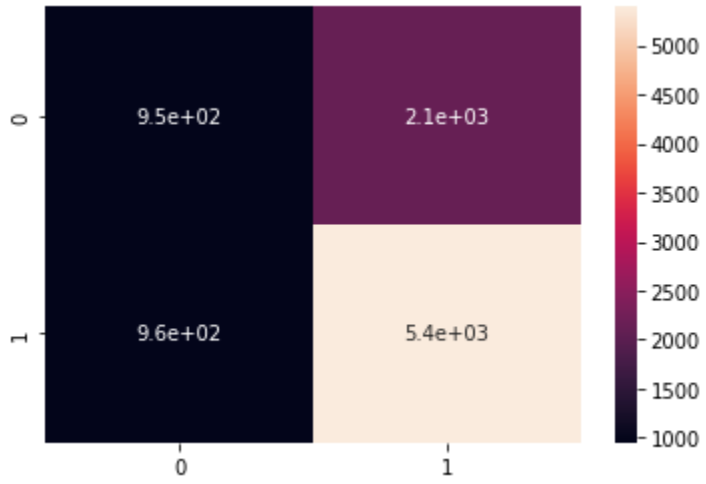
```
naive_bayes=GaussianNB()
hypertune_params={'var_smoothing': np.logspace(0,-9, num=1000)}
grid_search=GridSearchCV(naive_bayes , param_grid=hypertune_params , scoring='roc_auc')
grid_search.fit(x_train,y_train)
print(grid_search.best_params_)
```

```
{'var_smoothing': 1e-09}
```

The accuracy obtained after training the model with above params is 67.32, which is less when compared to a model where default parameters are applied.

The confusion matrix for this model is below. It is observed that the number of true negatives and false negatives have increased in this model.

[[951 2116]
[964 5394]]

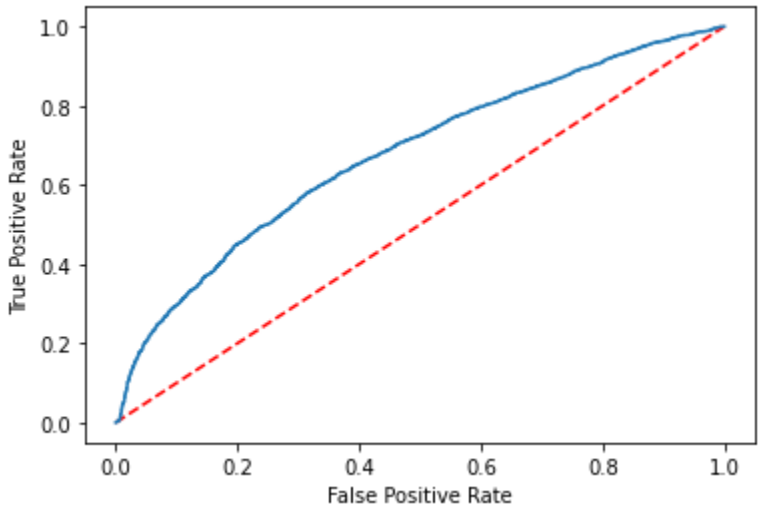


The Classification report:

	precision	recall	f1-score	support
0	0.50	0.31	0.38	3067
1	0.72	0.85	0.78	6358
accuracy			0.67	9425
macro avg	0.61	0.58	0.58	9425
weighted avg	0.65	0.67	0.65	9425

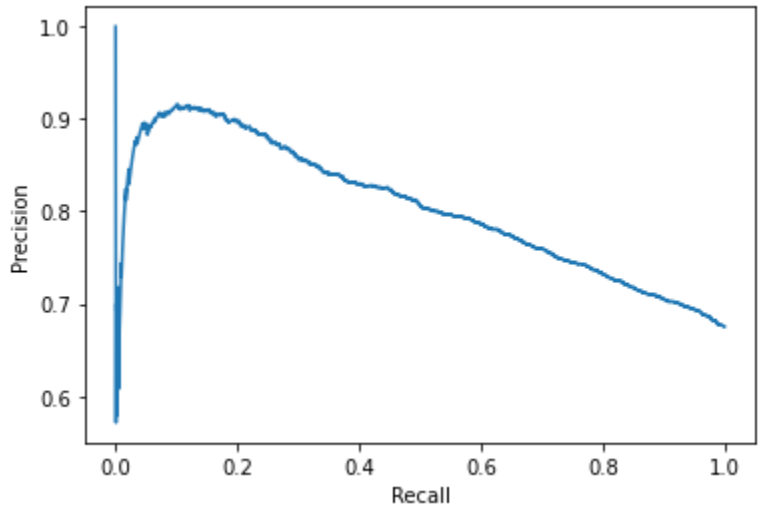
Specificity 0.3100749918487121
Sensitivity 0.8483799937087134

ROC-AUC Curve:



roc_auc_score - 67.52

Precision Recall Curve



4. K NEAREST NEIGHBORS:

The KNN algorithm uses distance metric to classify the data points. KNN Is said to give high accuracy and has a very low error rate. It is also used for huge datasets. KNN is also very easy to implement and interpret the results. KNN also relies very less on the training data. Choosing k is very important in KNN, which decides the classification of particular data point based on the number of nearest data points for that data point. In all the above algorithms we have observed that the roc-auc score was very low. Since KNN gives high accuracy and performs better, this algorithm is used. KNN is used to improve roc-auc score, i.e., to correctly identify the class labels.

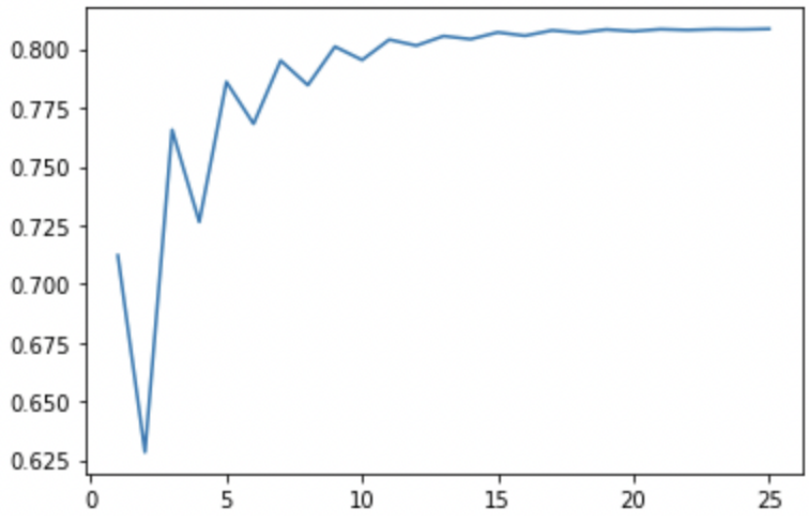
Train-Test split – since knn does not rely on training data for accuracy, 40% is used as testing data and 60% of input data is used for training data.

Since choosing k is important in knn, several models are fit over a range of k values and graph is plotted.

```
scores=[]
for k in range(1,26):
    knn=KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train,y_train)
    scores.append(knn.score(x_test,y_test))
```

```
r=range(1,26)
plt.plot(r,scores)
```

```
[<matplotlib.lines.Line2D at 0x164ad0100>]
```

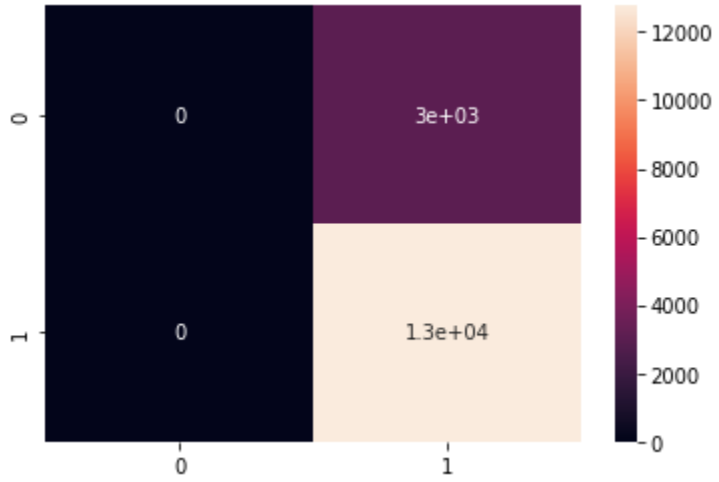


It is observed that with initially with increase in the value of k, the accuracy decreased but for values of k greater than k, the accuracy seemed to steadily increase.

A model is trained with k=100 and the following results are observed.
Test Accuracy - 80.87%
As estimated, the knn model gave a high accuracy.
Confusion Matrix:

[[0 3018]
[0 12764]]

All the positive inputs were predicted correctly, but there are no true negatives predicted. This means that the model can predict the candidates who are able to repay their loan.

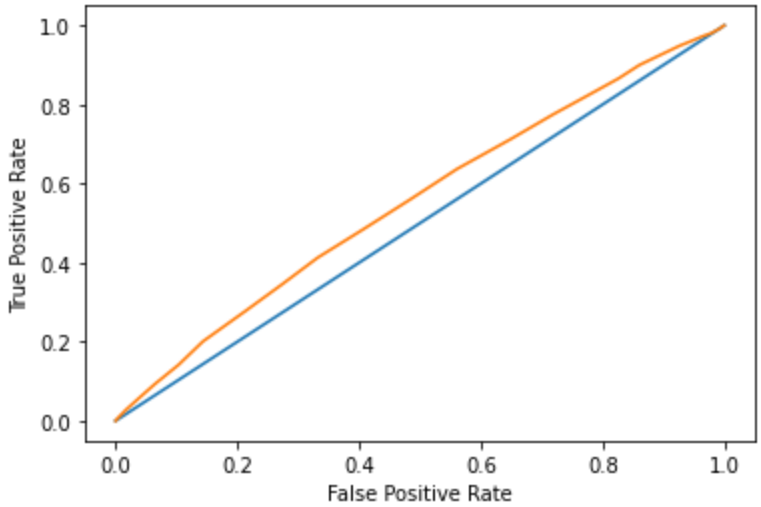


The classification report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	3018
1	0.81	1.00	0.89	12764
accuracy			0.81	15782
macro avg	0.40	0.50	0.45	15782
weighted avg	0.65	0.81	0.72	15782

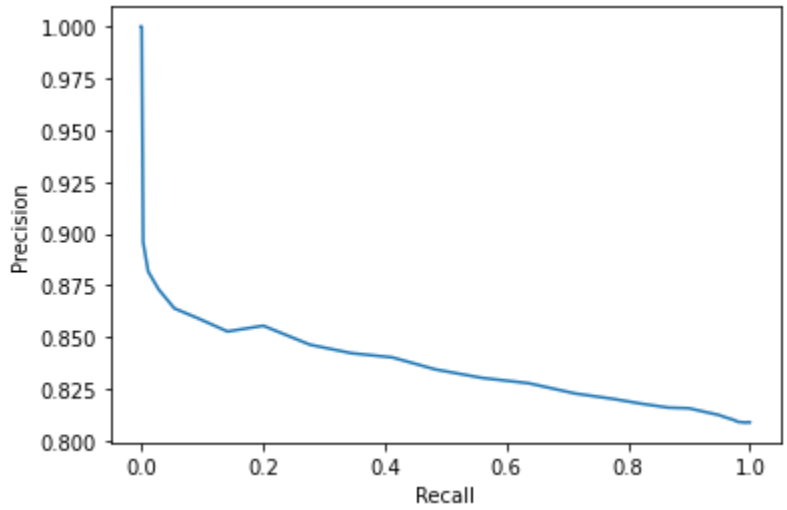
The precision and recall for positive class label is high.

ROC-AUC Curve



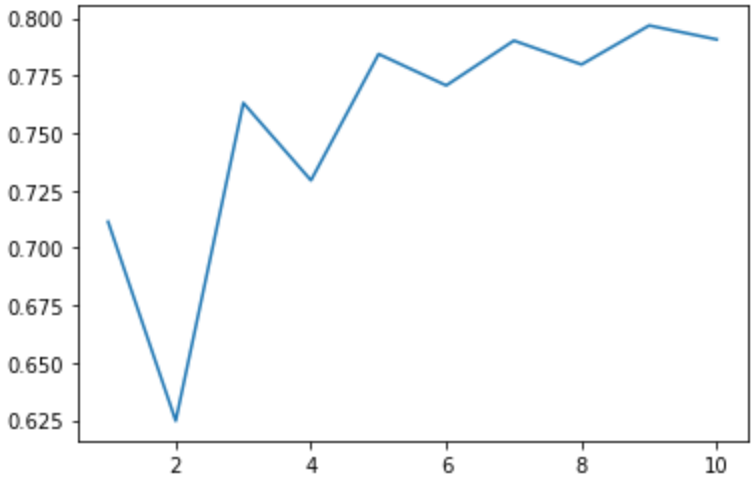
ROC-AUC Score - 55.42 - The area under the curve for ROC-AUC curve is very low.

Precision Recall Curve



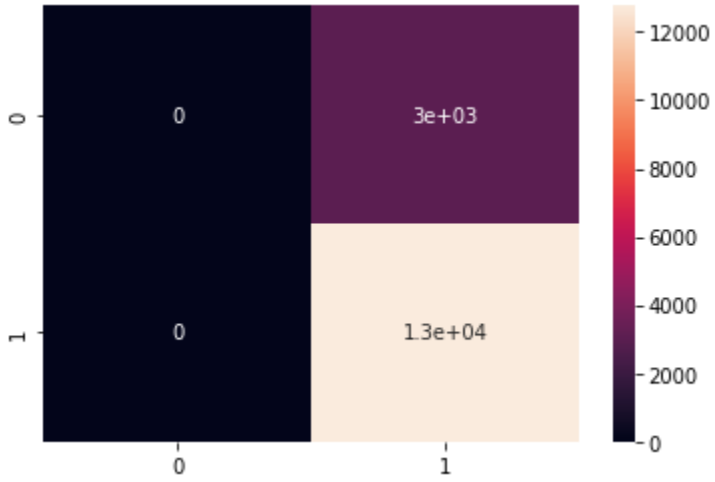
Model -2
Train and Test size is modified, and a model is fitted.
Train size - 90%
Test Size - 10%

Accuracies are plotted for a range of k values(1,11)



The accuracy of the model seems to increase with the increase in k values.
A model is trained with k=50 and following results are observed
KNeighborsClassifier(n_neighbors=50)
Testing Accuracy = 80.71%

Confusion Matrix:

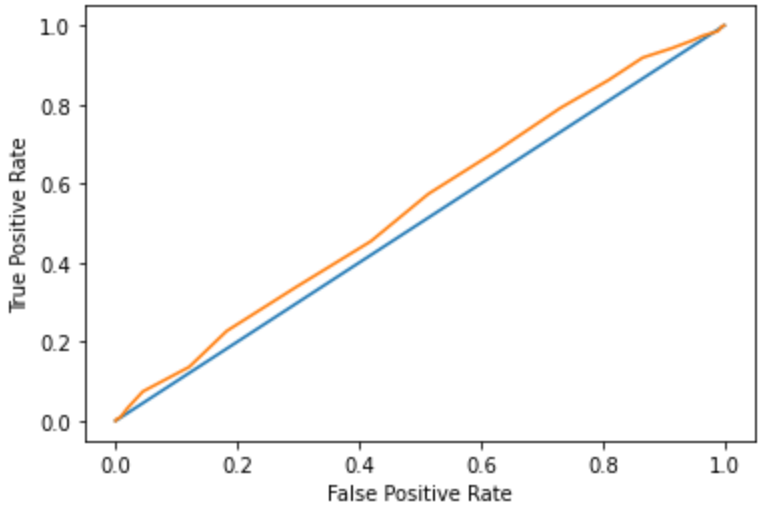


Classification Report:

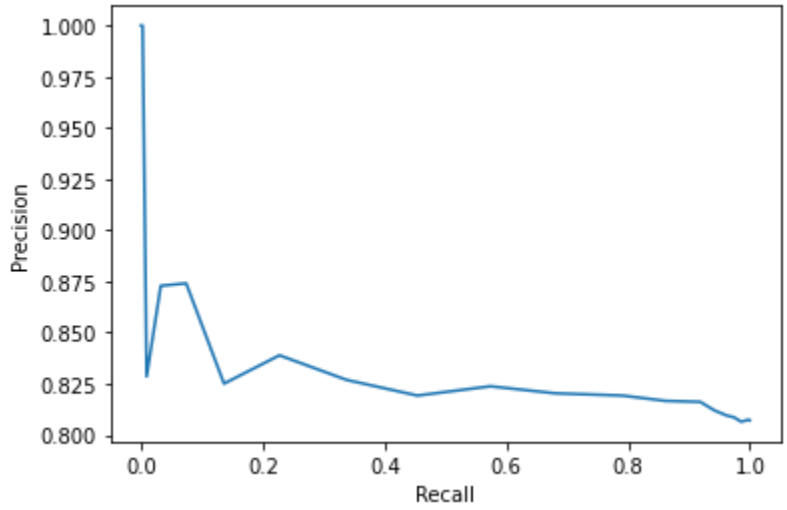
	precision	recall	f1-score	support
0	0.00	0.00	0.00	761
1	0.81	1.00	0.89	3185
accuracy			0.81	3946
macro avg	0.40	0.50	0.45	3946
weighted avg	0.65	0.81	0.72	3946

The observed classification report is the same as the model with 200 neighbors.

ROC-AUC Curve:



Precision Recall Curve:



KNN-Resampling:
Since we know that the model is imbalanced with a smaller number of records for those users who have defaulted on their loans, oversampling of those records is done.

```
oversample = df_default_loan[df_default_loan['loan_status'] == 0].sample(15000, replace=True)
df_oversample = pd.concat([oversample, df_default_loan[df_default_loan['loan_status'] == 1]], axis=0)
print("value counts after oversampling",df_oversample['loan_status'].value_counts())
```

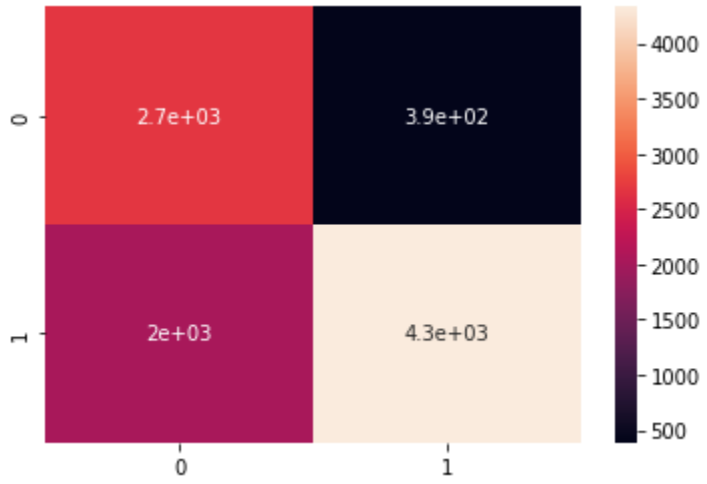
```
value counts after oversampling 1    32121
0    15000
Name: loan_status, dtype: int64
```

After the model is resampled, it is trained with 2 neighbors and the results are as follows. The overall accuracy was decreased but there was a sharp increase in the roc-auc score.

Testing Accuracy - 74.3%

Confusion Matrix:

[[2680 387]
[2029 4329]]



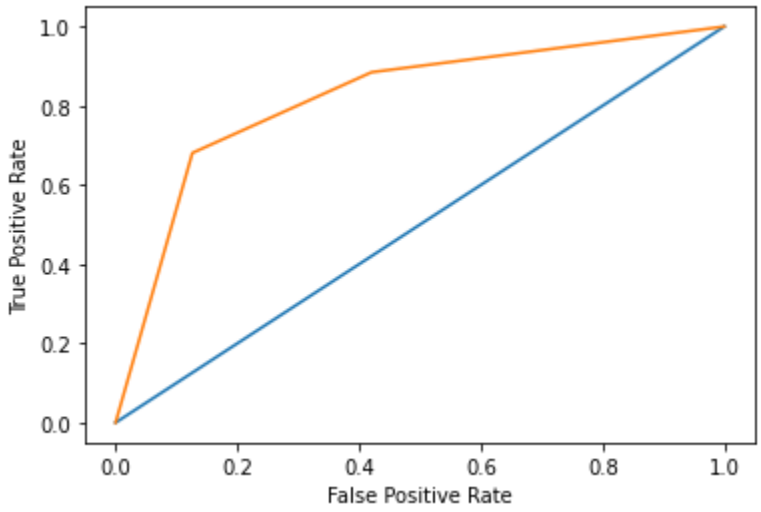
We can see that the number of true negatives has sharply increased unlike other models.

Classification Report:

	precision	recall	f1-score	support
0	0.57	0.87	0.69	3067
1	0.92	0.68	0.78	6358
accuracy			0.74	9425
macro avg	0.74	0.78	0.74	9425
weighted avg	0.80	0.74	0.75	9425

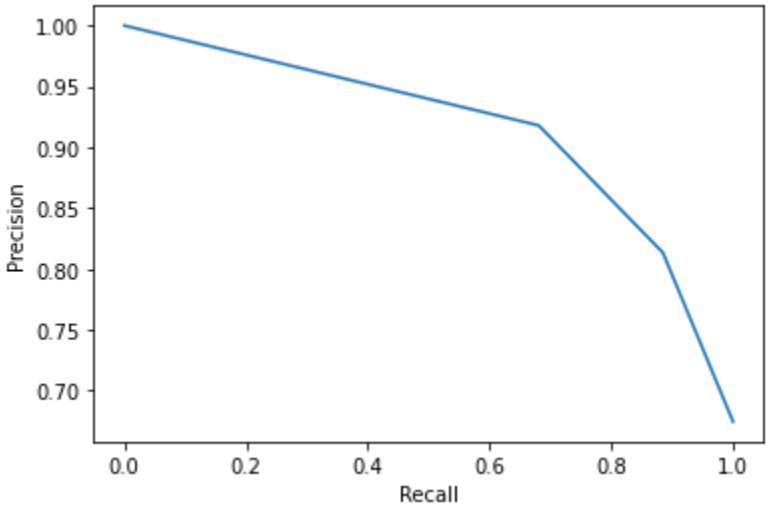
The precision and recall values have also improved for both the class labels.

ROC-AUC Curve



ROC-AUC Score - 81.94 - We can see that the area under the curve has increased which means that the model is able to correctly classify the classes

Precision Recall Curve:



The curve performed initially well with a sharp decrease.

KNN MODEL SUMMARY

Neighbors	Accuracy	ROC-AUC Score	Precision	Recall	Resampling	Train-Test%
100	80.87%	55.4	0.81	1.0	False	60-40
50	80.71%	54.13	0.81	1.0	False	90-10
2	74.36%	81.94	0.92	0.68	True	80-20

5. SUPPORT VECTOR MACHINES

Support vector machines are used when there is a clear separation between the classes. Here in the loan default dataset, there is a clear separation between both the outcomes i.e., the candidates who can repay the loan and the candidates who default on the loan. SVM is known to find an optimal boundary between the classes. It can also clearly segregate nonlinear data. SVM is used here to improve accuracy.
Train-test split – 65% train data and 35% test data.

The Linear SVC has the following parameters.
Penalty- regularization parameter
loss – specifies the loss function
C – the strength of regularization is inversely proportional to the value of C
max_iter – maximum iterations to be run

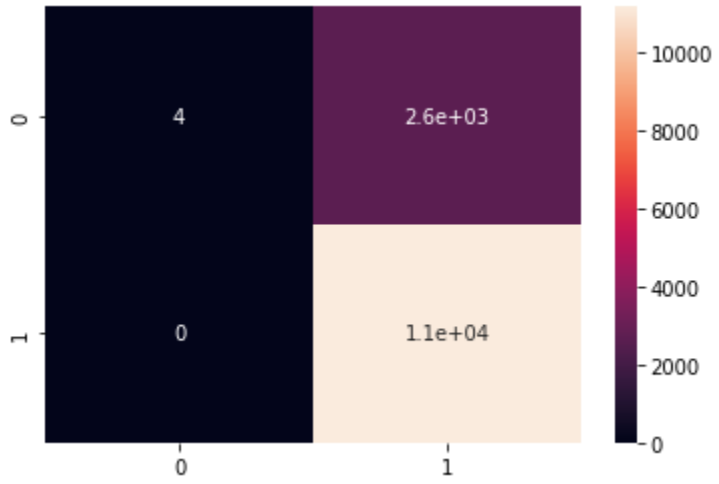
A Linear SVC is fitted, and the following results are observed

```
svm=LinearSVC(C=100,max_iter=1000000)
svm.fit(x_train, y_train)
print("score on test: " + str(svm.score(x_test, y_test)))
print("score on train: "+ str(svm.score(x_train, y_train)))
```

```
score on test: 0.8089065894279508
score on train: 0.8171573406122051
```

Confusion Matrix :

[[4 2639]
[0 11167]]



Classification Report:

	precision	recall	f1-score	support
0	1.00	0.00	0.00	2643
1	0.81	1.00	0.89	11167
accuracy			0.81	13810
macro avg	0.90	0.50	0.45	13810
weighted avg	0.85	0.81	0.72	13810

Specificity 0.0015134317063942 49
Sensitivity 1.0
The model is more sensitive than specific. It is performing better for positive class labels.

MODEL-2
Increasing the training data can increase accuracy. The training data is made up of 90% of the total input data and test size is 10% of total input data.

The following results are observed:

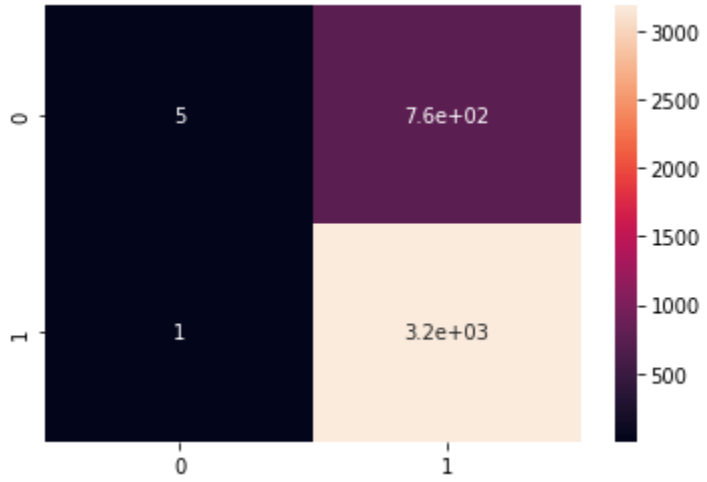
```
svm=LinearSVC(C=0.1,max_iter=100000,loss='hinge')
svm.probability=True
svm.fit(x_train, y_train)
print("score on test: " + str(svm.score(x_test, y_test)))
print("score on train: "+ str(svm.score(x_train, y_train)))
```

```
score on test: 0.808160162189559
score on train: 0.8152581035793742
```

Confusion Matrix: By increasing the training data size, the number of true negatives slightly increased.

[[5 756]

[1 3184]]



Classification Report:

	precision	recall	f1-score	support
0	0.83	0.01	0.01	761
1	0.81	1.00	0.89	3185
accuracy			0.81	3946
macro avg	0.82	0.50	0.45	3946
weighted avg	0.81	0.81	0.72	3946

Specificity 0.006570302233902759

Sensitivity 0.9996860282574568

The model performs well for positive class label.

MODEL-3

Since there was not much change in accuracy by changing the size of training data, now the size of test data and train data is made same.

Train Size - 50% of input data

Test Size - 50% of input data

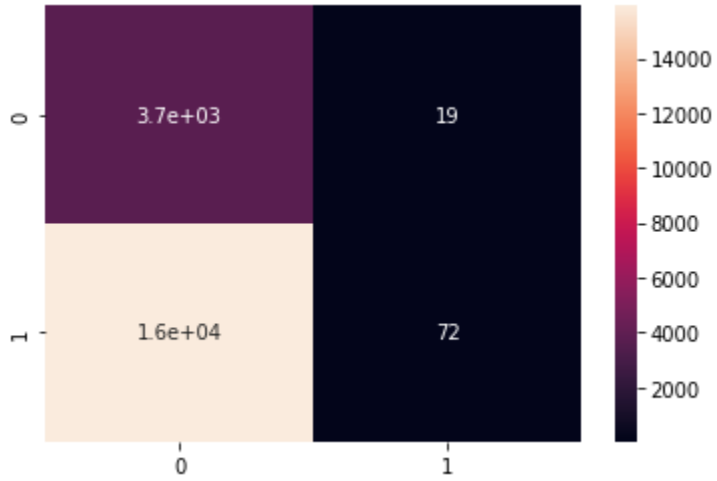
```
svm=LinearSVC(C=1000,max_iter=100000,class_weight='balanced')
svm.fit(x_train, y_train)
print("score on test: " + str(svm.score(x_test, y_test)))
print("score on train: "+ str(svm.score(x_train, y_train)))
```

```
score on test: 0.19251824817518248
score on train: 0.18472144776195062
```

The accuracy of the model drastically decreased.

Confusion Matrix: Number of true negatives increased but the total number of true positives highly decreased.

[[3726 19]
[15911 72]]



Classification Report:

	precision	recall	f1-score	support
0	0.19	0.99	0.32	3745
1	0.79	0.00	0.01	15983
accuracy			0.19	19728
macro avg	0.49	0.50	0.16	19728
weighted avg	0.68	0.19	0.07	19728

Specificity 0.9949265687583445
Sensitivity 0.004504786335481449

Now the model is highly specific, it is performing better for negative class labels. It can predict those candidates who default on their loan.

SVM-Oversampling:
Since the dataset is imbalanced, the class label with few numbers of records is oversampled.

```
oversample = df_default_loan[df_default_loan['loan_status'] == 0].sample(15000, replace=True)
df_oversample = pd.concat([oversample, df_default_loan[df_default_loan['loan_status'] == 1]], axis=0)
print("value counts after oversampling",df_oversample['loan_status'].value_counts())
```

```
value counts after oversampling 1    32121
0    15000
Name: loan_status, dtype: int64
```

After resampling the dataset, the following results were observed.

```
svm=LinearSVC(C=10,max_iter=10000,loss='hinge')
svm.fit(x_train, y_train)
print("score on test: " + str(svm.score(x_test, y_test)))
print("score on train: "+ str(svm.score(x_train, y_train)))
```

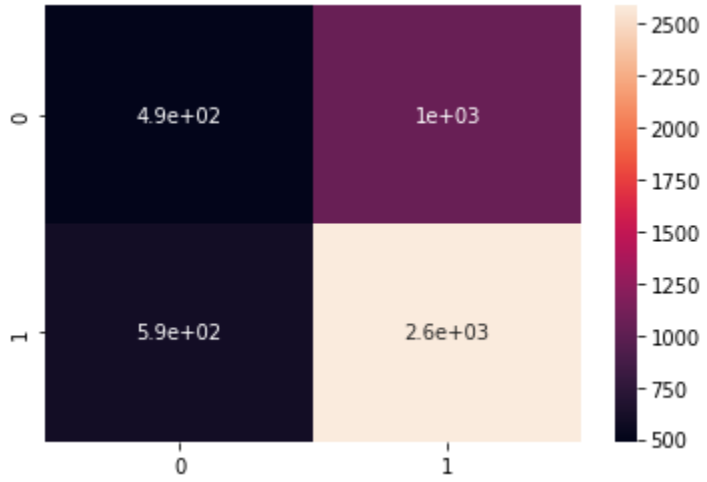
```
score on test: 0.6522384892849565
score on train: 0.6504904734955669
```

The accuracy seemed to decrease.

Confusion Matrix:

[489 1049]

[590 2585]]



Classification Report:

	precision	recall	f1-score	support
0	0.45	0.32	0.37	1538
1	0.71	0.81	0.76	3175
accuracy			0.65	4713
macro avg	0.58	0.57	0.57	4713
weighted avg	0.63	0.65	0.63	4713

Specificity 0.3179453836150845

Sensitivity 0.8141732283464567

SVM- ROC-AUC CURVE:

```
svm_roc = SGDClassifier(max_iter=10000,loss='hinge')
svm_calibrated = CalibratedClassifierCV(svm_roc, cv=5)
svm_calibrated.fit(x_train,y_train)
test_predict=svm_calibrated.predict(x_test)
train_accuracy=svm_calibrated.score(x_train,y_train)
test_accuracy=svm_calibrated.score(x_test,y_test)
print ('Test accuracy',test_accuracy)
print ('Train accuracy',train_accuracy)
```

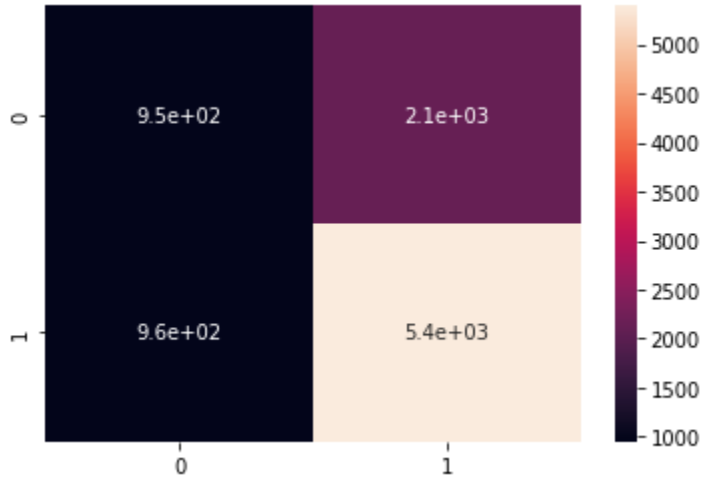
```
Test accuracy 0.8071464774455145
Train accuracy 0.8148919992114675
```

The accuracy seemed to improve.

Confusion Matrix: The number of true negatives and false negatives are equal

[[951 2116]

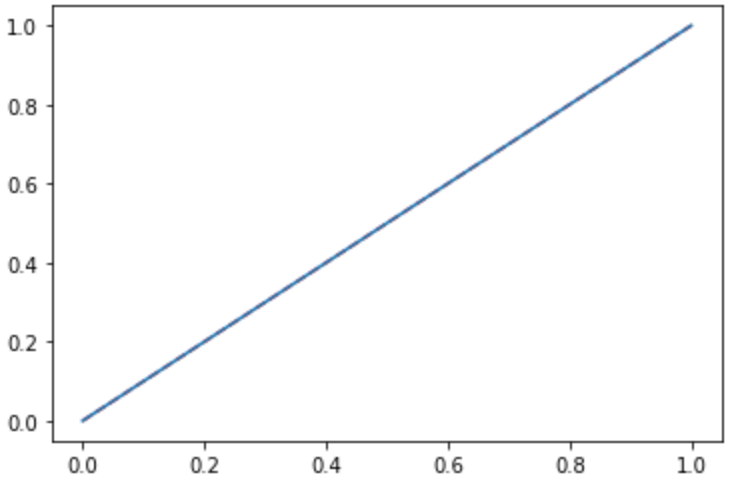
[964 5394]]



Classification report:

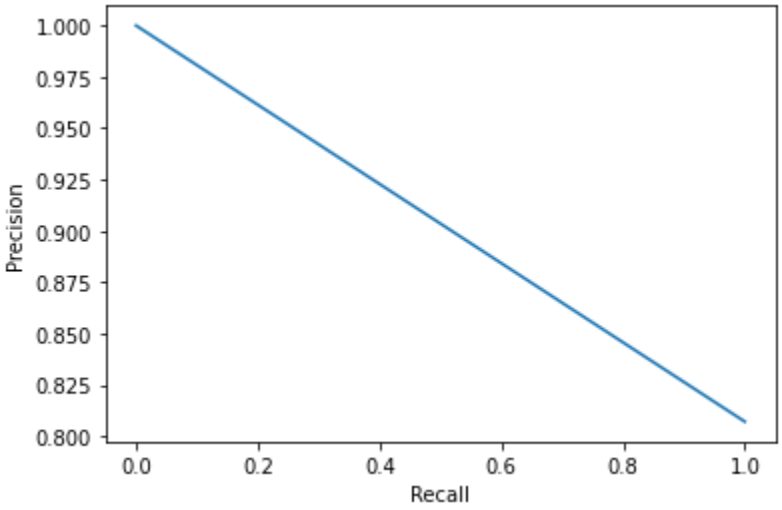
	precision	recall	f1-score	support
0	0.00	0.00	0.00	761
1	0.81	1.00	0.89	3185
accuracy			0.81	3946
macro avg	0.40	0.50	0.45	3946
eighted avg	0.65	0.81	0.72	3946

ROC-AUC Curve:



The model is not able to discriminate between classes.
ROC-AUC Score - 50.0

Precision Recall Curve:



SVM Models Summary:

loss	C	Max_iter	Accuracy	Precision	Recall	Train-Test%
hinge	100	1000000	80.89%	0.81	1.00	65-35
hinge	0.1	100000	80.81%	0.81	1.00	90-10
hinge	1000	100000	19.25%	0.79	0.00	50-50

6. ## XGBOOST:

It is a tree-based algorithm that is used for classification. Since the dataset we have is highly imbalanced, xgboost can be used instead of random sampling the dataset. In gradient boosting, weights are used for sampling, where weights are added to those records that belong to the class with less number of records. Weights are removed for those records that belong to the class with more number of records. Instead of manually sampling the dataset, xgboost can be used to predict the candidates who can repay the loan and the candidates who default on the loan. It also allows faster convergence of the algorithm. It has hyperparameter 'scale_pos_weight' that allows to minimize the class imbalance.

Parameters of the xgboost are as follows:
objective - Specify the learning task and the corresponding learning objective
random_state – Sampling
eval_metric - Evaluation metrics for validation data
eta - Step size shrinkage used in update to prevents overfitting
gamma - Minimum loss reduction required to make a further partition

Train-Test Split – 40% of dataset is test data and 60% is training data.
After applying the xgboost, the following results are observed:

```
xgb_classifier = xgb.XGBClassifier(objective="binary:logistic",
 random_state=42,
 eval_metric="auc",
 eta=0.5,
 gamma=1)
xgb_classifier.fit(x_train,y_train)
test_predict=xgb_classifier.predict(x_test)
train_predict=xgb_classifier.predict(x_train)
print("Train Accuracy:",accuracy_score(y_train,train_predict))
print("Test Accuracy:",accuracy_score(y_test,test_predict))
```

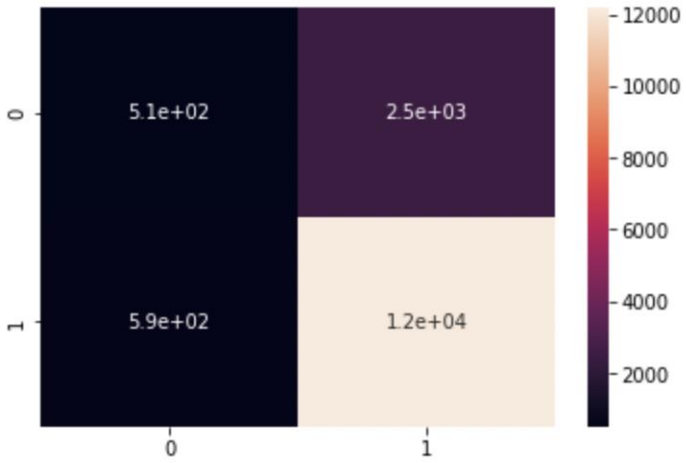
```
Train Accuracy: 0.929159802306425
Test Accuracy: 0.8037637815232543
```

Here the objective function is binary:logistic because we are trying to classify into 2 classes - binary classification.

Confusion Matrix:

```
[[  512  2506]
 [  591 12173]]
```

```
1]: <AxesSubplot:>
```

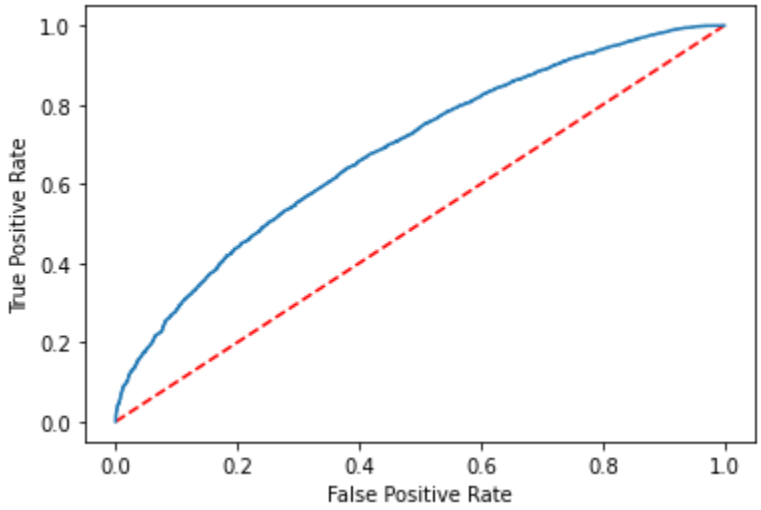


There are more number of true positives than true negatives. The model is able to predict those candidates who can repay the loan.

Classification Report:

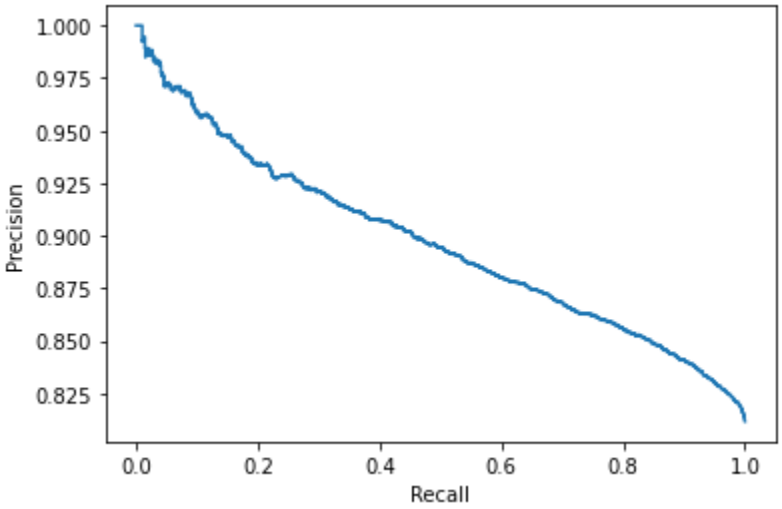
	precision	recall	f1-score	support
0	0.46	0.17	0.25	3018
1	0.83	0.95	0.89	12764
accuracy			0.80	15782
macro avg	0.65	0.56	0.57	15782
weighted avg	0.76	0.80	0.77	15782

ROC-AUC Curve:



The area under the curve is more than SVM.
ROC-AUC Score - 68.54. The ROC-AUC curve is pretty much decent but not better than KNN model.

Precision Recall Curve:



HYPERPARAMETERS TUNING:

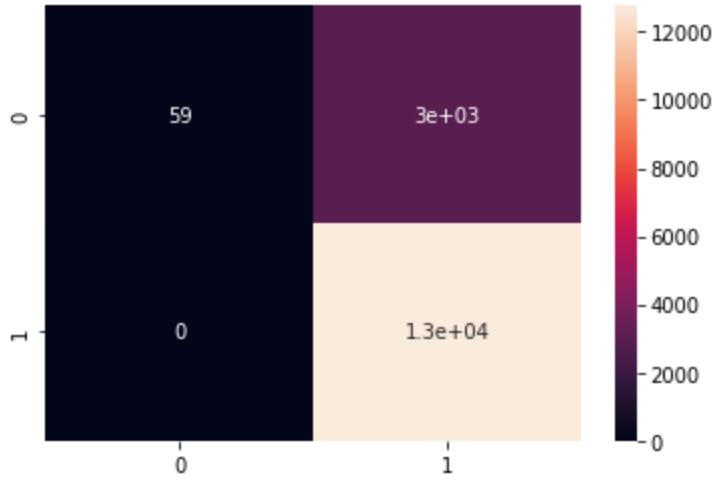
```
xgb_classifier = xgb.XGBClassifier(objective="binary:logistic",
 random_state=1234,
 eval_metric="auc",
 eta=1e-7,
 gamma=1e-5,
scale_pos_weight=200)
xgb_classifier.fit(x_train,y_train)
test_predict=xgb_classifier.predict(x_test)
train_predict=xgb_classifier.predict(x_train)
print("Train Accuracy:",accuracy_score(y_train,train_predict))
print("Test Accuracy:",accuracy_score(y_test,test_predict))
```

```
Train Accuracy: 0.8215266337177375
Test Accuracy: 0.8125079204156634
```

After tuning some of the hyperparameters, the accuracy of the model seemed to increase and is less overfit now. The hyperparameter scale_pos_weight is added to decrease the imbalances in the dataset.

Confusion Matrix: There were no false negatives predicted.

[[59 2959]
[0 12764]]



Classification report:

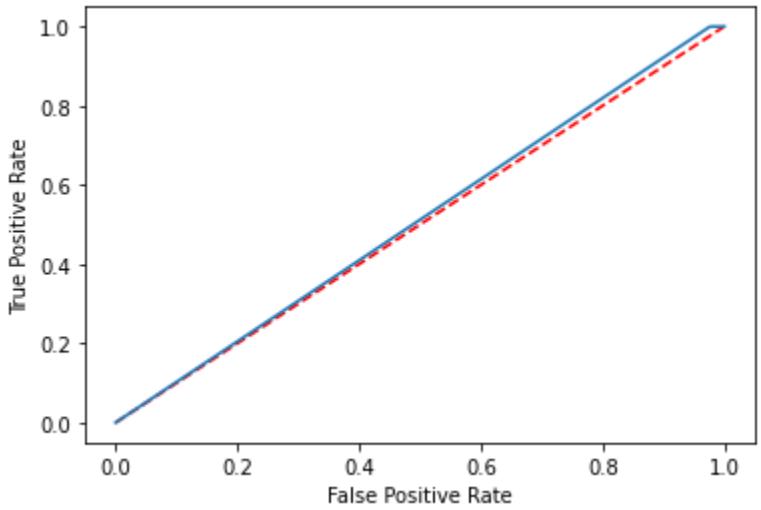
	precision	recall	f1-score	support
0	1.00	0.02	0.04	3018
1	0.81	1.00	0.90	12764
accuracy			0.81	15782
macro avg	0.91	0.51	0.47	15782
weighted avg	0.85	0.81	0.73	15782

Specificity 0.01954937044400265

Sensitivity 1.0

The model is highly sensitive, it is able to predict positive outcomes correctly.

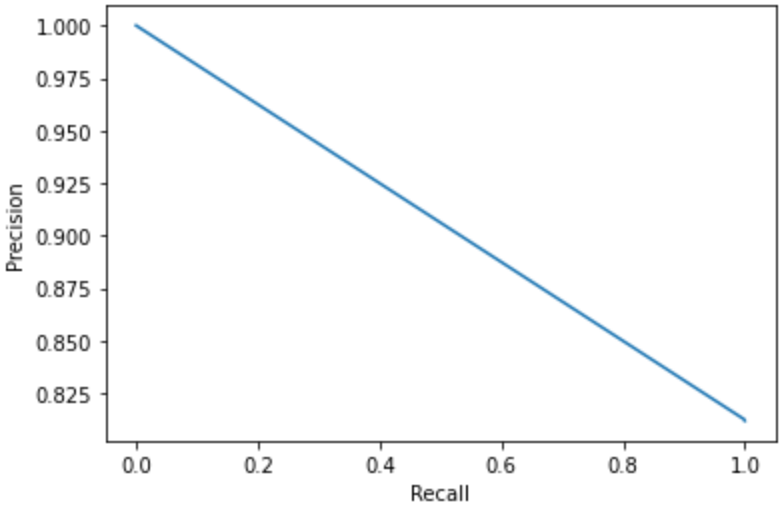
ROC-AUC CURVE:



The area under the curve is very low.

ROC AUC Score – 51.17 - Less than most of the models.

PRECISION RECALL CURVE:



MODEL - 2:

```
xgb_classifier = xgb.XGBClassifier(objective="binary:hinge",
 random_state=42,
 eval_metric="rmse",
 eta=1e-7,
 gamma=1e-5)
xgb_classifier.fit(x_train,y_train)
test_predict=xgb_classifier.predict(x_test)
train_predict=xgb_classifier.predict(x_train)
print("Train Accuracy:",accuracy_score(y_train,train_predict))
print("Test Accuracy:",accuracy_score(y_test,test_predict))
```

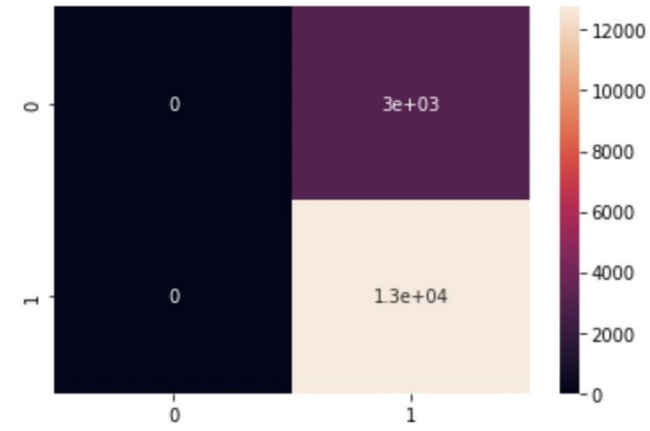
```
Train Accuracy: 0.8176825919824272
Test Accuracy: 0.808769484222532
```

Here binary:hinge objective function is used as it makes predictions 0 or 1 instead of predicting probabilities.

Confusion Matrix:

```
[[    0  3018]
 [    0 12764]]
```

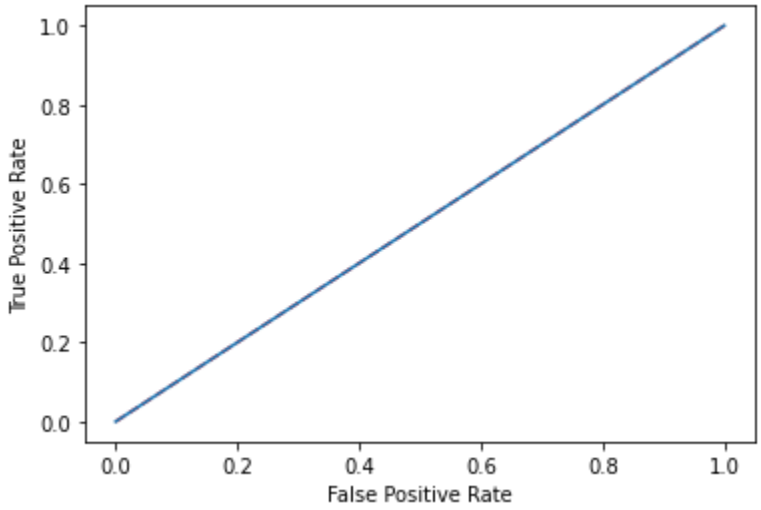
```
<AxesSubplot:>
```



Classification Report:

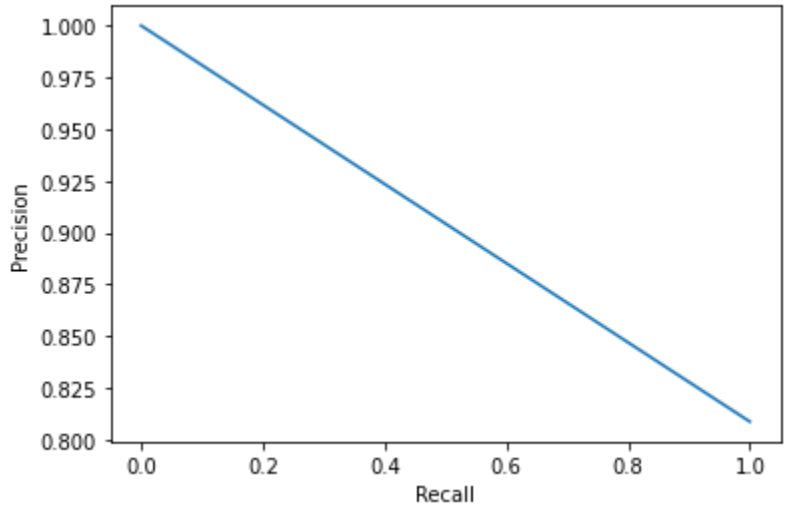
	precision	recall	f1-score	support
0	0.00	0.00	0.00	3018
1	0.81	1.00	0.89	12764
accuracy			0.81	15782
macro avg	0.40	0.50	0.45	15782
weighted avg	0.65	0.81	0.72	15782

ROC-AUC CURVE:



ROC AUC Score : 50.0

PRECISION RECALL CURVE:



Model - 3

Here the sizes of train data and test data are changed to 70% and 30% respectively and following results were observed.

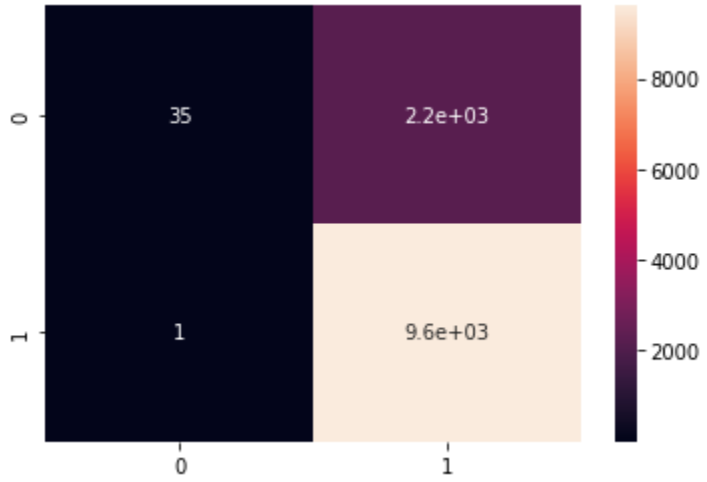
```
xgb_classifier = xgb.XGBClassifier(objective="binary:logistic",
 random_state=1234,
 eval_metric="auc",
 eta=1e-7,
 gamma=1e-5,
scale_pos_weight=200)
xgb_classifier.fit(x_train,y_train)
test_predict=xgb_classifier.predict(x_test)
train_predict=xgb_classifier.predict(x_train)
print("Train Accuracy:",accuracy_score(y_train,train_predict))
print("Test Accuracy:",accuracy_score(y_test,test_predict))
```

```
Train Accuracy: 0.8187776088058513
Test Accuracy: 0.8136352116245671
```

Confusion Matrix:

[[35 2205]

[1 9596]]



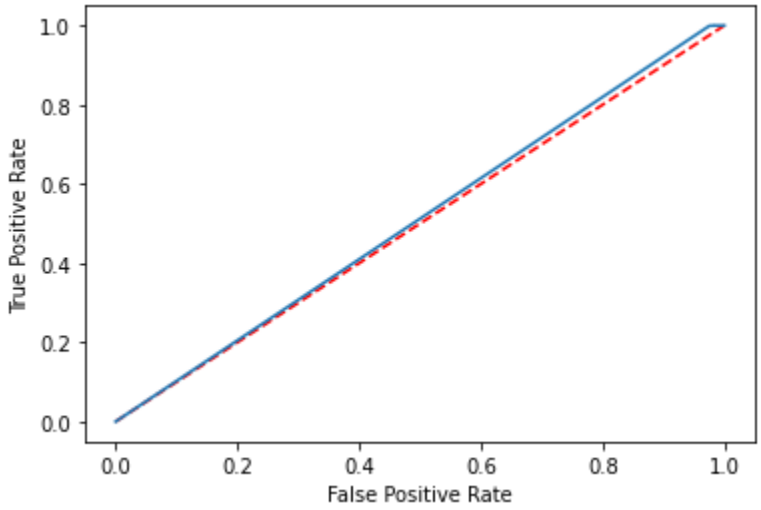
Classification Report:

	precision	recall	f1-score	support
0	0.97	0.02	0.03	2240
1	0.81	1.00	0.90	9597
accuracy			0.81	11837
macro avg	0.89	0.51	0.46	11837
weighted avg	0.84	0.81	0.73	11837

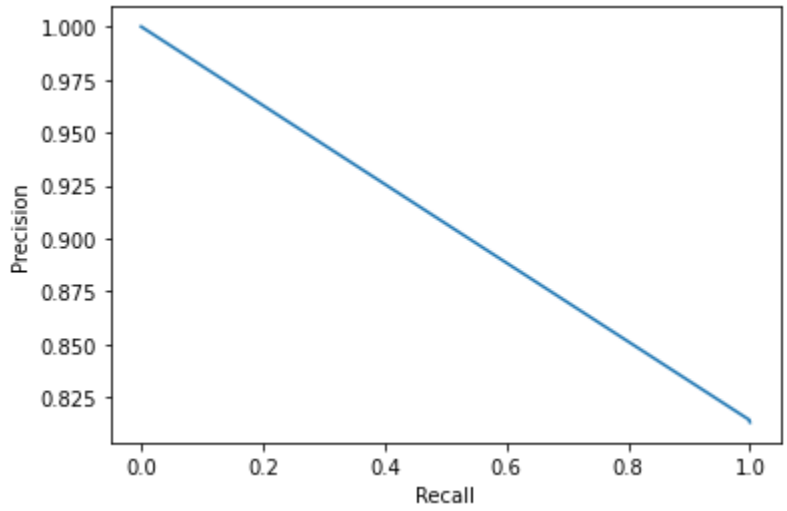
Specificity 0.015625

Sensitivity 0.9998958007710743

ROC-AUC CURVE:



PRECISION RECALL CURVE:



XGBOOST MODELS SUMMARY

Objective	Eval_metric	eta	gamma	Accuracy	ROC-AUC Score	Train-Test %
binary:logistic	auc	0.5	1	80.37%	68.54	60-40
binary:logistic	auc	1e-7	1e-5	81.25%	51.17	60-40
binary:hinge	rmse	1e-7	1e-5	81.76%	50.0	60-40
binary:logistic	auc	1e-7	1e-5	81.36%	51.19	70-30

7. NEURAL NETWORKS

Neural networks are used for efficient classification of binary classification problems. Since we were not able to visualize the internal details of the training for the above algorithms, neural networks can be used for visualization and for also better accuracies. Here, we want to improve the accuracy of the problem, hence we are using neural networks as they are very efficient for large datasets and they can also work with complex and incomplete datasets.

Here we are designing a neural network with 128 input nodes and activation layers as sigmoid and selu. Loss function is binary_crossentropy as we are trying to solve binary classification problem. Optimizer used is Adam as it uses stochastic gradient descent.

Model Summary:

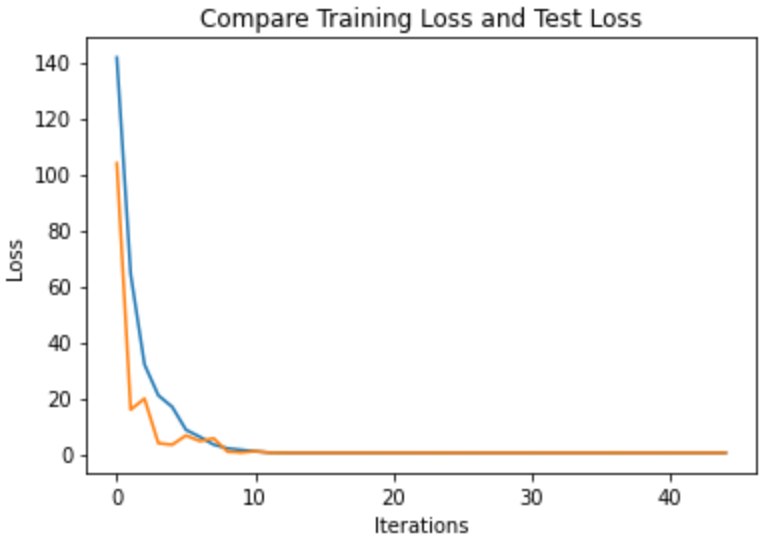
Model: "sequential_1"

Layer (type)	Output Shape	Param #
layer1 (Dense)	(None, 128)	2432
layer2 (Dense)	(None, 128)	16512
outputlayer (Dense)	(None, 1)	129

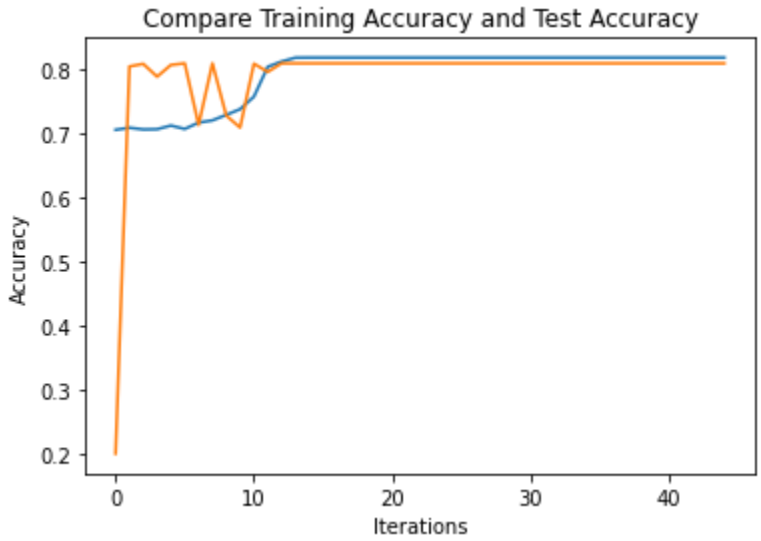
Total params: 19,073
Trainable params: 19,073
Non-trainable params: 0

Accuracy of the model : 80.88

Graph comparing Training Loss and Testing loss:



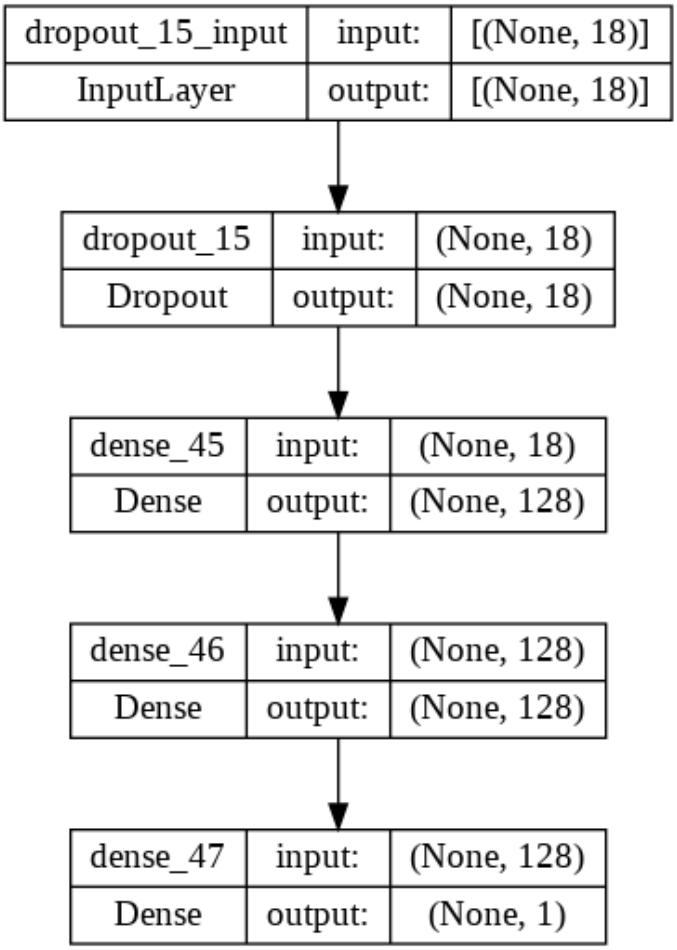
Graph comparing Training Accuracy and Test Accuracy:



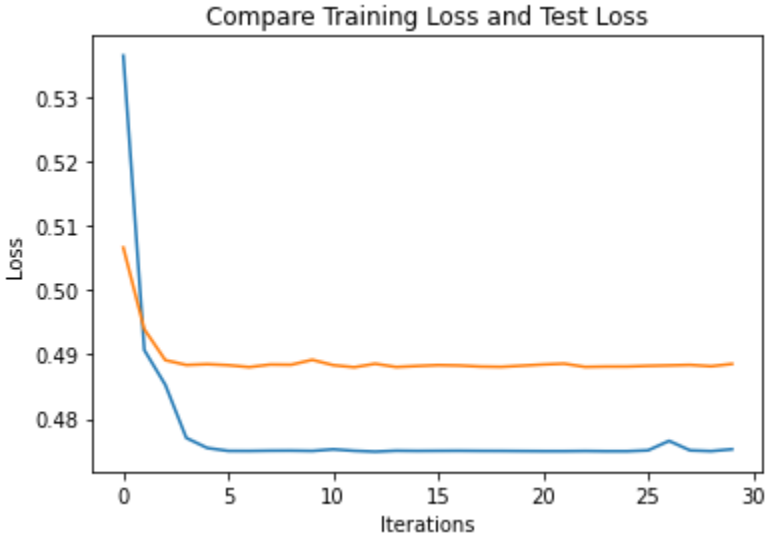
Tuning of Hypermeters:

MODEL-1

Activation function - relu,relu,sigmoid
Loss – binary_crossentropy
Optimizer – nadam
Accuracy - 80.8%
Model Summary:

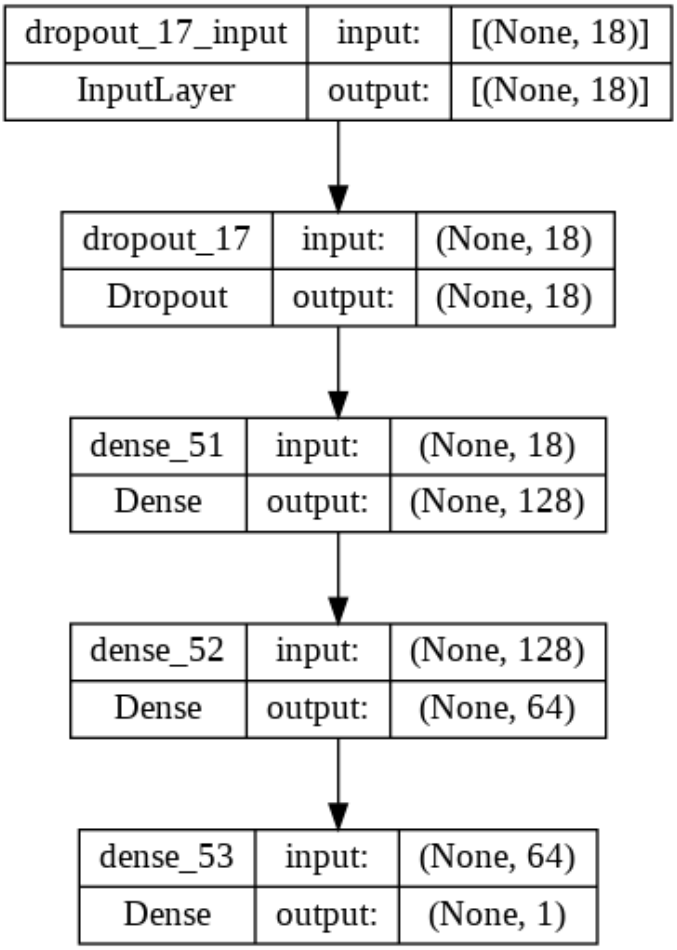


Graph comparing Training Loss and Testing loss:

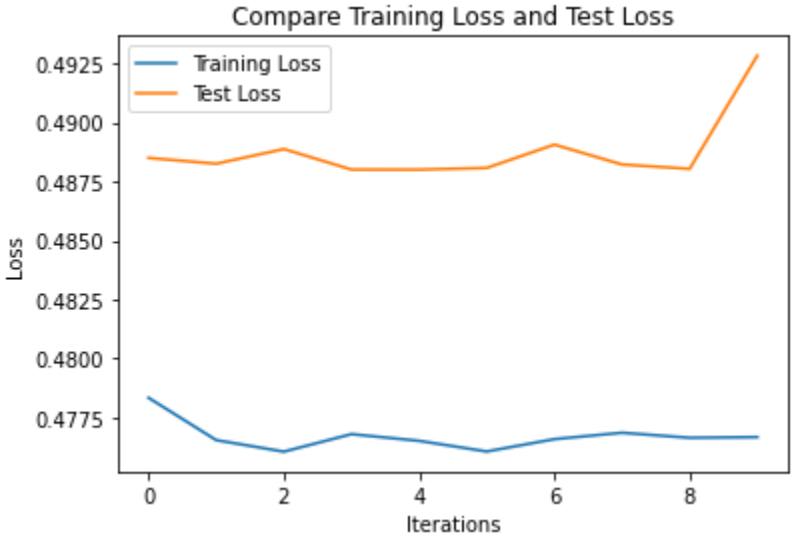


MODEL-1

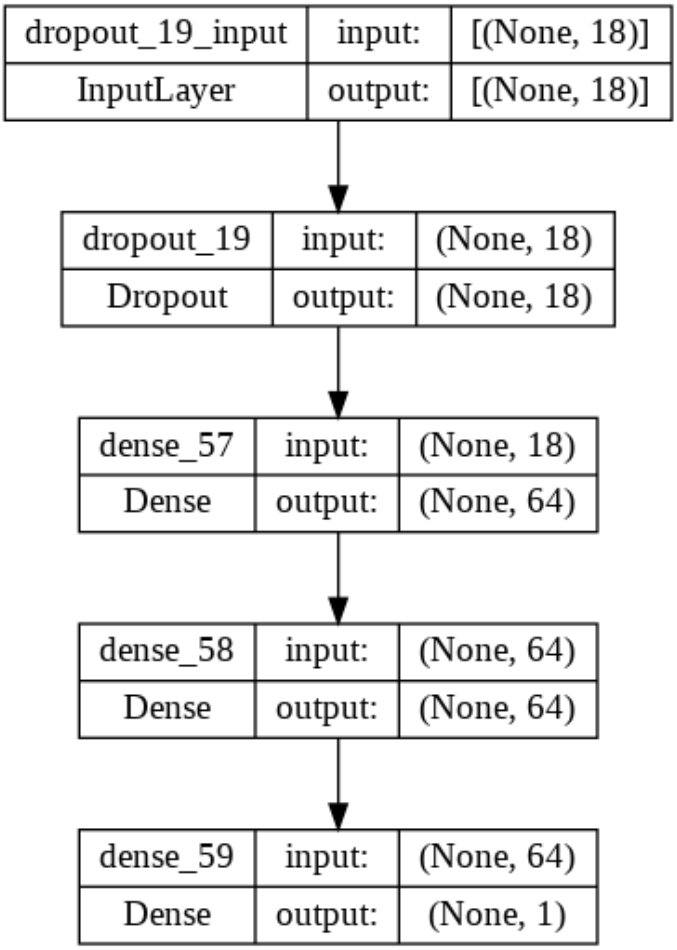
Activation function - tanh,selu,sigmoid
Loss – binary_crossentropy
Optimizer – aadam
Accuracy - 80.88%
Model Summary:



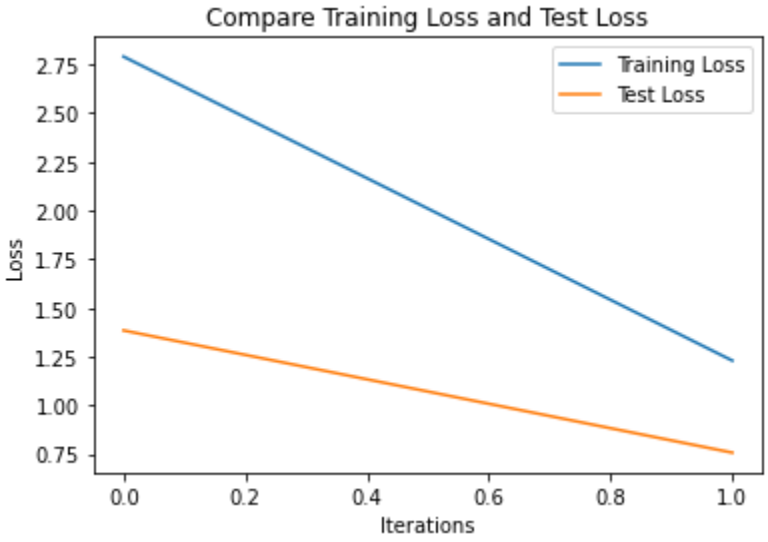
Graph comparing Training Loss and Testing loss:



Model - 3 Early Stopping:
Activation function - selu,relu,sigmoid
Loss – binary_crossentropy
Optimizer – adam
Accuracy - 80.8%
Model Summary:

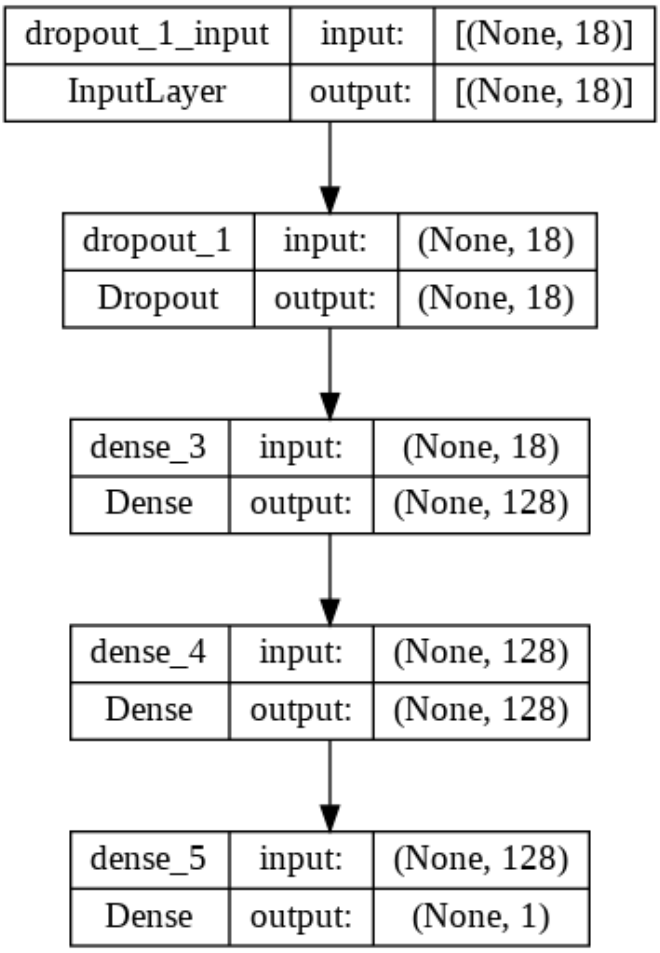


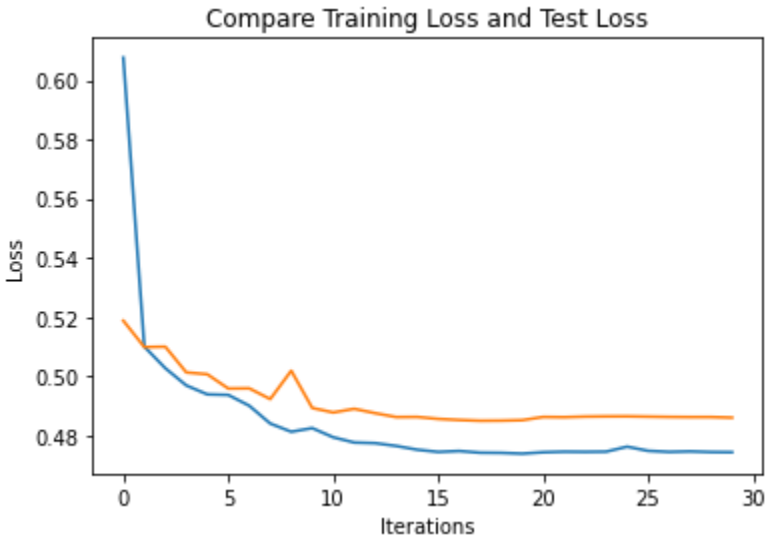
Graph comparing Training Loss and Testing loss:



MODEL – 4
Train data - 50%
Test data - 50%

Activation function - relu,relu,sigmoid
Loss – binary_crossentropy
Optimizer – adam
Accuracy - 81.02%
Model Summary:





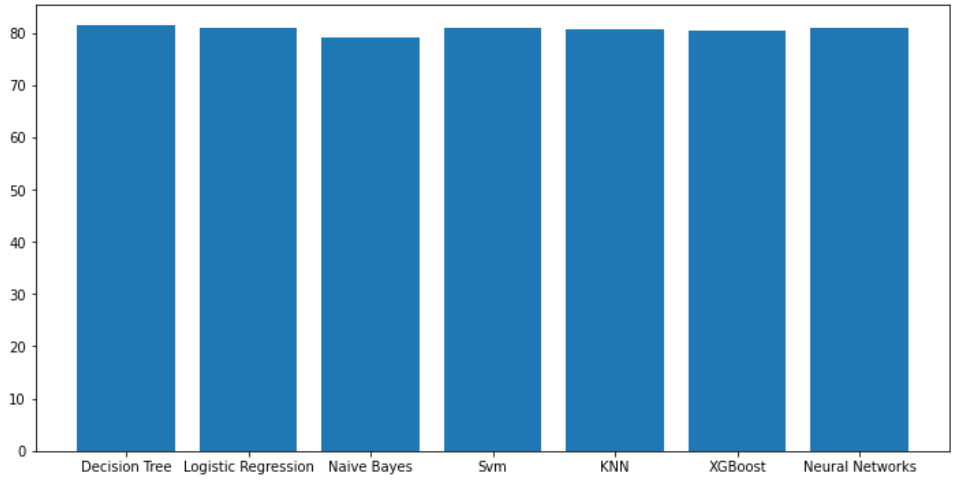
Neural Networks Models Summary:

Activation	Loss	optimizer	Accuracy
Selu,selu,sigmoid	binary_crossentropy	adam	80.8%
relu,relu,sigmoid	binary_crossentropy	nadam	80.8%
Tanh,relu,sigmoid	binary_crossentropy	adam	80.8%
relu,relu,sigmoid	binary_crossentropy	adagrad	80.8%
Selu,relu,sigmoid	binary_crossentropy	adam	80.8% - early stopping
relu,relu,sigmoid	binary_crossentropy	nadam	81.02% - train data- 50%

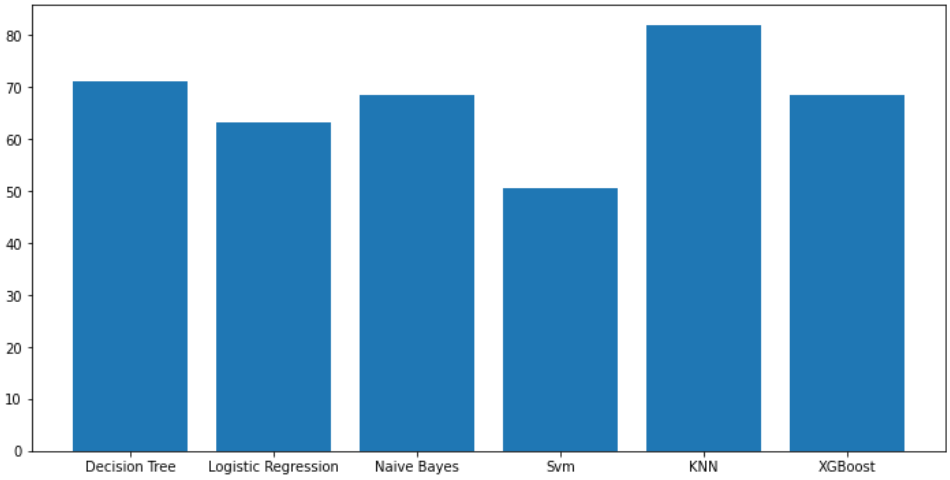
It is observed that though the accuracy of all the models is same but there is a significant change in the training loss and testing loss while fitting and training the model.

SUMMARY OF ALL THE MODELS:

Graph comparing the accuracies of all the above models.



Graph Comparing the ROC-AUC Score of all the above models:



REFERENCES:

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
https://xgboost.readthedocs.io/en/stable/parameter.html

https://keras.io/guides/sequential_model/

https://keras.io/api/layers/activations/
https://numpy.org/doc/stable/reference/generated/numpy.logspace.html

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

https://www.tensorflow.org/

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html

https://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

https://keras.io/api/callbacks/early_stopping/

https://keras.io/api/utils/model_plotting_utils/

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

TEAM MEMBERS

NAME	UBIT#	UBID
Srujana Golconda	50442205	sgolcond@buffalo.edu
Mahalakshmi Chintala	50442308	Mchintal@buffalo.edu