Capstone Project PGP-DSBA

Notes 2

Isha Shukla

24 August 2024

Contents

SL No.	Title	Page No.
1	Logistic Regression	3
2	LDA	10
3	KNN	16
4	Gaussian Naive Bayes	27
5	Random Forest	29
6	Bagging with Random Forest	32
7	Gradient Boost	36
8	XGBoost	40
9	Ada-Boost	44
10	SVM	51

1. Model Building

In this stage of the capstone project, we will proceed to build and evaluate various machine learning models using the cleaned and preprocessed data from the earlier stages. We will tune the models to optimize their performance and assess their accuracy using multiple metrics, including Accuracy, F1 Score, Recall, Precision, ROC curve, AUC score, Confusion matrix, and Classification report. Our goal is to select a model that strikes a balance between underfitting and overfitting, while achieving the highest accuracy possible.

Splitting Data into Train and Test dataset

Data is split into training and testing sets in a 70:30 ratio, adhering to standard market practice. Various models are built and trained using the training data, and their performance is evaluated by measuring their accuracy on the testing data.

Shape of Train and Test dataset

```
X_train (7882, 17)
X_test (3378, 17)
y_train (7882,)
y_test (3378,)
```

Dimension of Train and Test dataset

1) Logistic Regression

i. Logistic Regression with default parameters

After splitting the data into training and testing sets, a logistic regression model is fitted to the training dataset. Predictions are then made on both the training and testing datasets using the same model. The model is configured

to run for a maximum of 1,000 iterations, ensuring convergence. The 'lbfgs' solver is selected as the optimization algorithm, which is suitable for smaller datasets and supports multiclass classification. Furthermore, a random state of 42 is set to guarantee reproducibility, enabling consistent model behavior across different runs.

```
Accuracy of training set: 0.8869576249682821
Confusion matrix of training set: [[6363  192]
 [ 699  628]]
Training ROC-AUC Score: 0.8757512371407206
```

Accuracy, Confusion Matrix, ROC-AUC Score - Train

```
Test Accuracy: 0.8931320307874482
Test Confusion Matrix:
 [[2732   77]
 [ 284  285]]
Test ROC-AUC Score: 0.8717666851652454
```

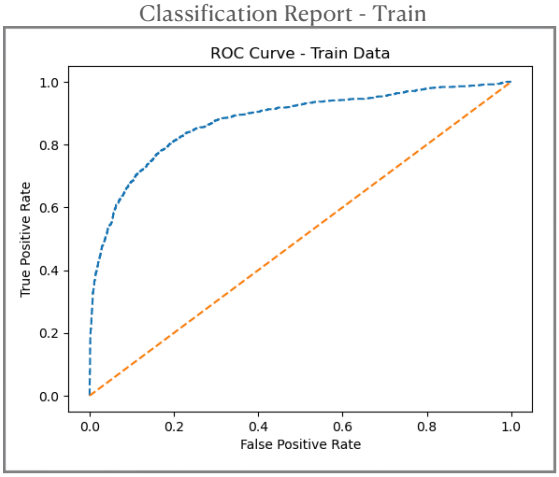
Accuracy, Confusion Matrix, ROC-AUC Score - Test

```
Classification report for train dataset
              precision    recall  f1-score   support

           0       0.90      0.97      0.93      6555
           1       0.77      0.47      0.59      1327

    accuracy                           0.89      7882
   macro avg       0.83      0.72      0.76      7882
weighted avg       0.88      0.89      0.88      7882
```

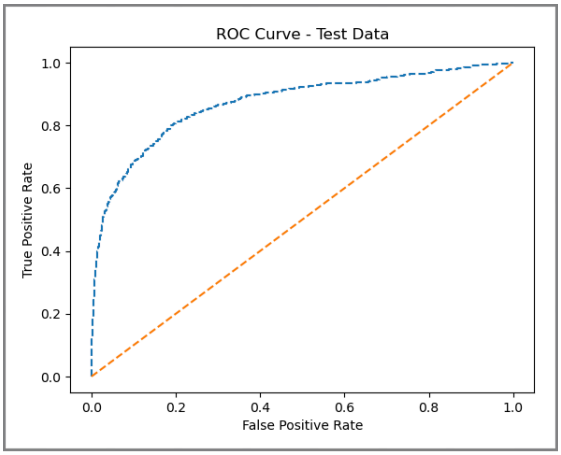
Classification Report - Train

	precision	recall	f1-score	support
0	0.91	0.97	0.94	2809
1	0.79	0.50	0.61	569
accuracy			0.89	3378
macro avg	0.85	0.74	0.78	3378
weighted avg	0.89	0.89	0.88	3378

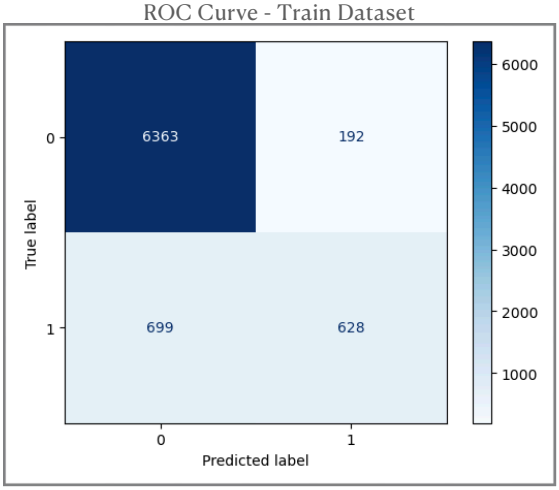
Classification Report - Test



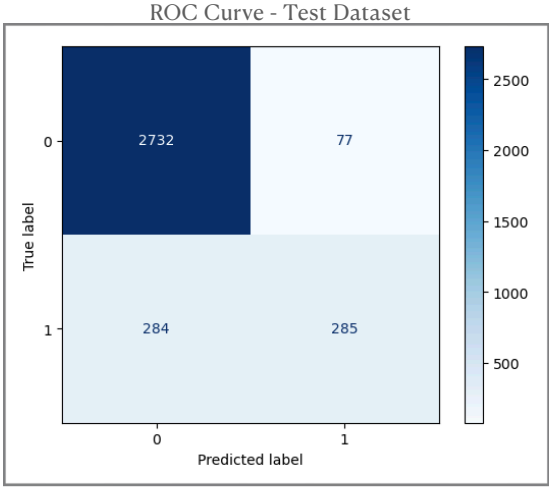
ROC Curve - Train Dataset



ROC Curve - Test Dataset



Confusion Matrix -Training Performance



Confusion Matrix -Testing Performance

Cross-validation is a technique used to assess the performance and generalizability of a model by partitioning the data into multiple subsets, or "folds." Cross validation with default parameters with 5 fold for both training and testing.

```
Cross-Validation Score (Training Set): [0.88142042 0.88395688 0.88705584 0.88642132 0.89403553]
Cross-Validation Score (Testing Set): [0.8816568  0.87721893 0.88905325 0.88148148 0.89481481]
```

Cross validation score for Logistic Regression with default parameters

The model's performance is relatively stable, with minimal variation in scores across different 5 folds, suggesting that the model is robust and not sensitive to specific subsets of the data. Additionally, the scores are relatively high, indicating that the model is able to accurately predict the target variable for a significant proportion of instances. Furthermore, the similarity in scores between the training and testing sets suggests that the model is not overfitting to the training data, and is able to generalize well to new, unseen data. Notably, the model's performance on the testing set is slightly lower than on the training set, which is expected, but the difference is relatively small, indicating that the model is not overfitting to the training data.

ii. Logistic Regression using GridSearchCV

1. **Logistic Regression**: A classification algorithm that models the probability of a binary outcome based on one or more features. It includes hyperparameters like C (inverse of regularization strength), solver (optimization algorithm), and penalty (type of regularization).
2. **GridSearchCV:** An exhaustive search method provided by scikit-learn that tries all possible combinations of the hyperparameters in the specified grid. It uses cross-validation to evaluate each combination, ultimately selecting the one that performs the best.

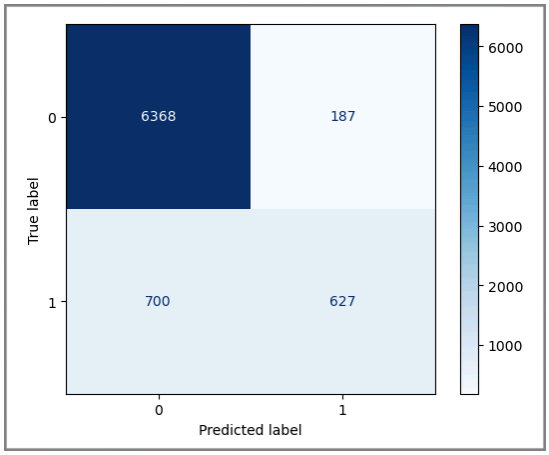
	Accuracy	Recall	Precision	F1
0	0.887211	0.470987	0.769704	0.584385

Logistic Regression using GridSearchCV Train Performance

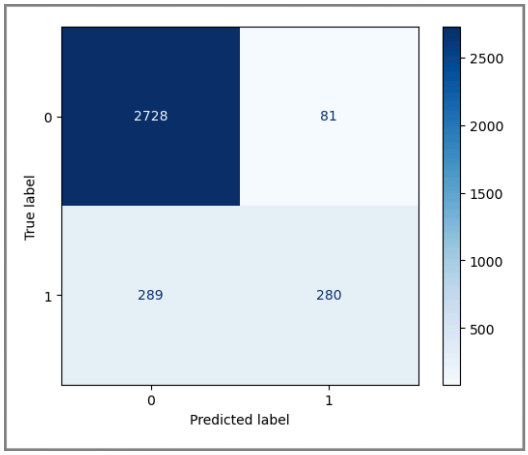
Testing Performance:

	Accuracy	Recall	Precision	F1
0	0.890468	0.492091	0.775623	0.602151

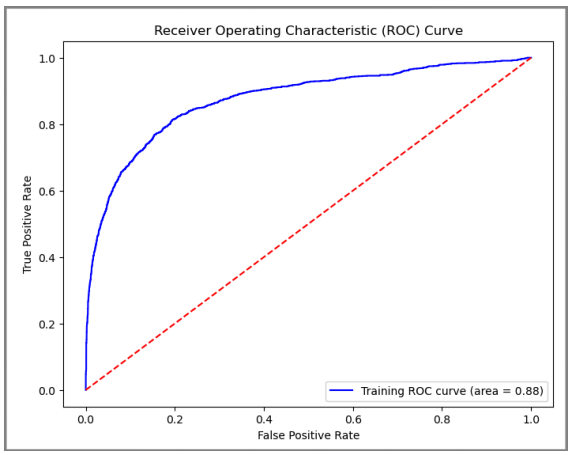
Logistic Regression using GridSearchCV - Testing Performance



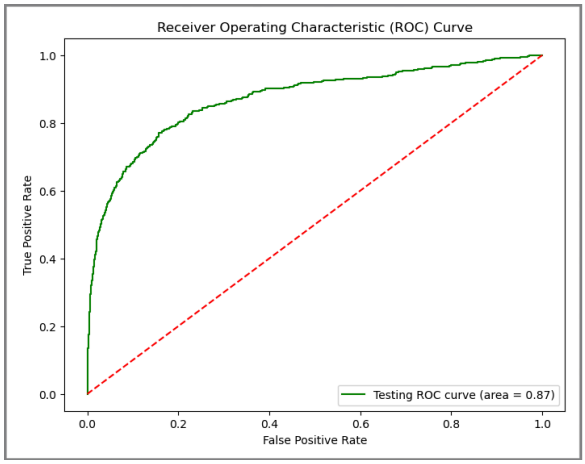
Confusion Matrix - Train



Confusion Matrix - Test



ROC Curve - Train Data



ROC Curve - Test Data

```
cross validation score for training dataset
[0.88649334 0.884591   0.88959391 0.88705584 0.8927665 ]
```

```
cross validation score for testing dataset
[0.87721893 0.88017751 0.8816568  0.86962963 0.89777778]
```

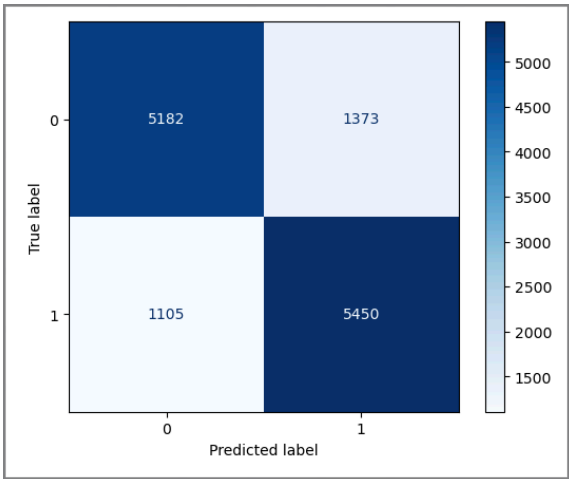
The Logistic Regression model tuned with GridSearchCV shows strong and consistent performance, with training cross-validation scores ranging from

0.884 to 0.893 and testing scores between 0.870 and 0.898. This close alignment between training and testing scores indicates that the model generalizes well, effectively capturing underlying patterns without significant overfitting. The results suggest that the selected hyperparameters are well-suited for the task, leading to robust predictive capabilities across both datasets.

iii. Logistic Regression using SMOTE

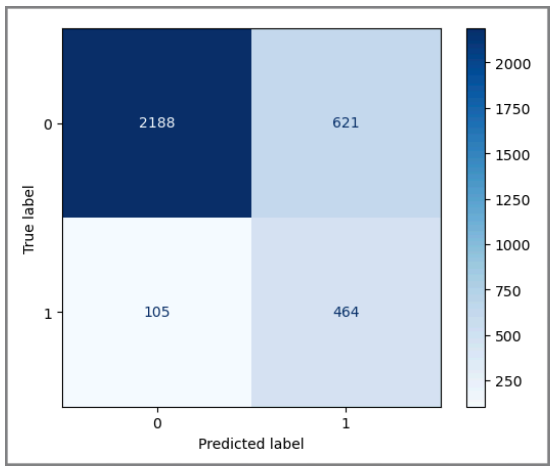
In our previous analysis, we observed that the data is imbalanced. We applied the SMOTE technique to balance the dataset and then built a model on this balanced data to assess any significant improvements in accuracy for both training and testing sets. However, after evaluating the model, we found that the accuracy improvements were not substantial.

```
Resampled Training Accuracy: 0.8109839816933638
Resampled Training Confusion Matrix:
 [[5182 1373]
 [1105 5450]]
Resampled Training ROC-AUC Score: 0.8815377015815832
```

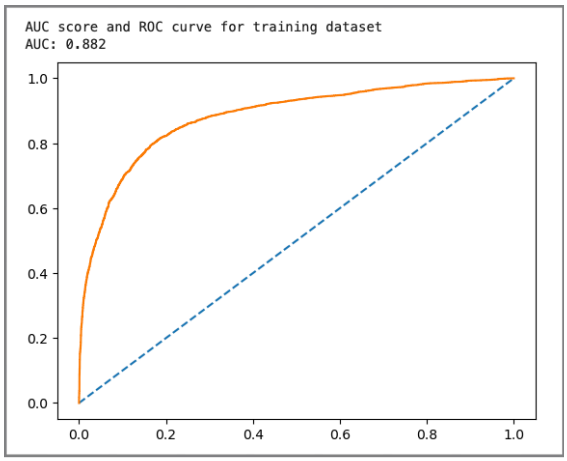


Confusion Matrix - Train Dataset(SMOTE)

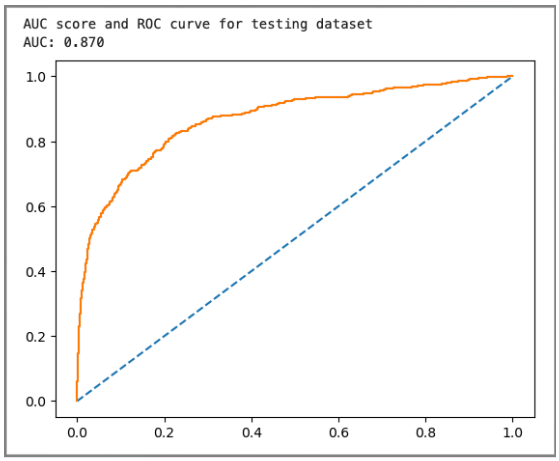
```
Test Accuracy: 0.7850799289520426
Test Confusion Matrix:
 [[2188  621]
 [ 105  464]]
Test ROC-AUC Score: 0.8697176599694304
```



Confusion Matrix - Test Dataset(SMOTE)



ROC Curve & AUC Scores From Logistic Regression with SMOTE - Train Set



ROC Curve & AUC Scores From Logistic Regression with SMOTE - Train Set

Training set classification report:

	precision	recall	f1-score	support
0	0.82	0.79	0.81	6555
1	0.80	0.83	0.81	6555
accuracy			0.81	13110
macro avg	0.81	0.81	0.81	13110
weighted avg	0.81	0.81	0.81	13110

Testing set classification report using SMOTE:

	precision	recall	f1-score	support
0	0.95	0.78	0.86	2809
1	0.43	0.82	0.56	569
accuracy			0.79	3378
macro avg	0.69	0.80	0.71	3378
weighted avg	0.87	0.79	0.81	3378

Classification Report From Logistic Regression with SMOTE

Cross validation score from Logistic Regression using SMOTE

```
cross validation score for balanced training dataset
array([0.79595728, 0.81617086, 0.81083143, 0.80968726, 0.81884058])
```

```
cross validation score for testing dataset
array([0.8816568 , 0.87721893, 0.88905325, 0.88148148, 0.89481481])
```

The logistic regression model appears to be benefiting from the balanced training dataset, with a notable improvement in performance on the testing dataset. This suggests that the model is able to learn more effectively from the balanced data and generalize well to new, unseen data. The high scores on the testing dataset indicate that the model is able to accurately predict the target variable for a significant proportion of instances.

Conclusion from Logistic Regression

Upon comparison, it is evident that the models with default parameters and with GridSearch CV outperform the model trained on the balanced dataset on both training and testing sets. This implies that the default parameters with GridSearch CV are better suited for this dataset, rendering dataset balancing unnecessary in this case. However, it is important to note that the balanced dataset model may still hold value if the primary objective is to enhance performance on the minority class.

2. Linear Discriminant Analysis Model (LDA)

i. LDA with default parameters

Using the same training and testing data split, we are now building a Linear Discriminant Analysis (LDA) model with default parameters to determine if it can outperform the Logistic Regression model. This will help us select the best model for further predictions.

```
Accuracy score of training dataset: 0.8808678000507485
```

```
Accuracy score of testing dataset: 0.8854351687388987
```

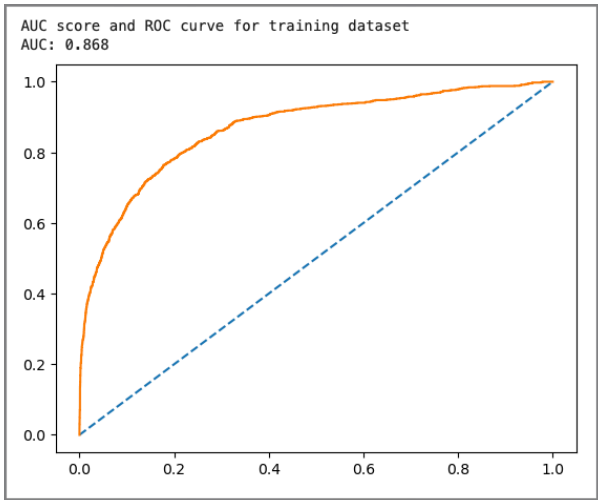
Accuracy from LDA

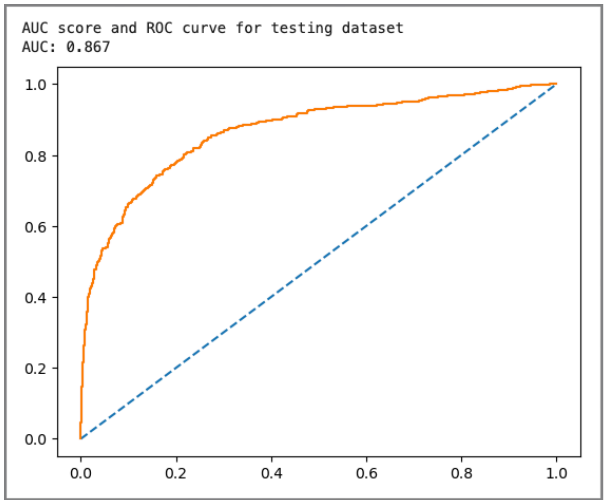
```
Confusion matrix of training dataset
array([[6385,  170],
       [ 769,  558]])
```

```
Confusion matrix of testing dataset
array([[2742,   67],
       [ 320,  249]])
```

```
Classification Report of the training data:

              precision    recall  f1-score   support

           0       0.89      0.97      0.93      6555
           1       0.77      0.42      0.54      1327

    accuracy                           0.88      7882
   macro avg       0.83      0.70      0.74      7882
weighted avg       0.87      0.88      0.87      7882


Classification Report of the test data:

              precision    recall  f1-score   support

           0       0.90      0.98      0.93      2809
           1       0.79      0.44      0.56       569

    accuracy                           0.89      3378
   macro avg       0.84      0.71      0.75      3378
weighted avg       0.88      0.89      0.87      3378
```

Classification Report From LDA






```
cross validation score for training dataset
array([0.88212928, 0.8643853 , 0.88324873, 0.87690355, 0.8819797 ,
       0.87817259, 0.87690355, 0.88959391, 0.89467005, 0.87817259])
```

```
cross validation score for testing dataset
array([0.88757396, 0.87278107, 0.88757396, 0.86982249, 0.86094675,
       0.89940828, 0.90532544, 0.86094675, 0.884273  , 0.89614243])
```

Cross Validation for LDA with default parameters

The LDA model with default parameters appears to be performing well on both the training and testing datasets, with high cross-validation scores. This suggests that the model is able to effectively discriminate between the classes and make accurate predictions. The similarity in scores between the training and testing datasets indicates that the model is not overfitting or underfitting the data, and is generalizing well to new data.

ii. LDA using GridSearchCV and best parameter

The Linear Discriminant Analysis (LDA) model is defined with the goal of finding the optimal hyperparameters to improve its performance. The hyperparameter space to search is defined, including the number of components to retain, shrinkage method, and solver. A GridSearchCV object is created to perform an exhaustive search over this space, configured to use

the LDA model, perform 5-fold cross-validation, utilize all available CPU cores, and optimize based on the accuracy metric.

The hyperparameter space to search is defined, comprising three key parameters: the number of components to retain in the LDA model, which can take on values of 1, 2, 3, 4, or 5; the shrinkage method, which can be either 'auto' or None; and the solver, which can be either 'lsqr' or ‘eigen'.

```
Best LDA model: LinearDiscriminantAnalysis(n_components=1, shrinkage='auto', solver='lsqr')
Best hyperparameters: {'n_components': 1, 'shrinkage': 'auto', 'solver': 'lsqr'}
```

LDA Hyperparameter

```
Train set accuracy: 0.8809946714031972
```

```
Test set accuracy: 0.8854351687388987
```

LDA Train and Test Accuracy

```
Classification report for train dataset
              precision    recall  f1-score   support

           0       0.89      0.97      0.93      6555
           1       0.77      0.42      0.54      1327

    accuracy                           0.88      7882
   macro avg       0.83      0.70      0.74      7882
weighted avg       0.87      0.88      0.87      7882
```

LDA Classification Report - Train Dataset

```
Classification report for test dataset
              precision    recall  f1-score   support

           0       0.90      0.98      0.93      2809
           1       0.79      0.44      0.56       569

    accuracy                           0.89      3378
   macro avg       0.84      0.71      0.75      3378
weighted avg       0.88      0.89      0.87      3378
```

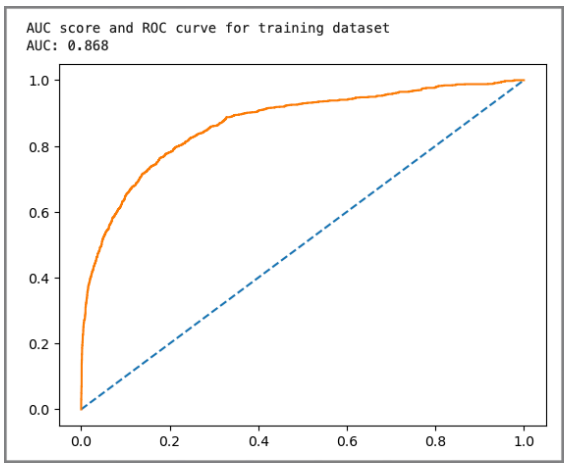
LDA Classification Report - Test Dataset

```
confusuon matrix for training dataset
array([[6387,  168],
       [ 770,  557]])
```

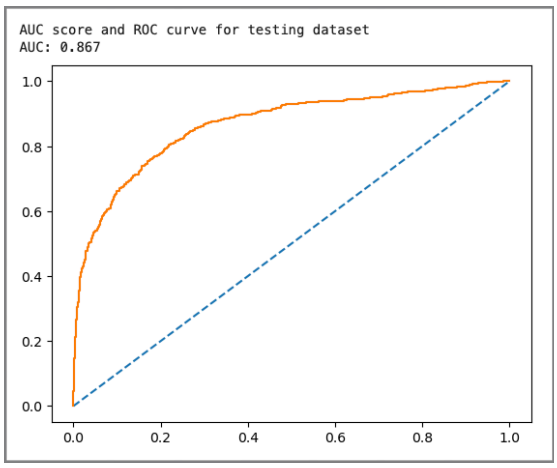
Confusion Matrix for training datasets

```
array([[2743,   66],
       [ 321,  248]])
```

Confusion Matrix for Testing datasets



AUC_ROC Curve for LDA using GridSearchCV - train



AUC_ROC Curve for LDA using GridSearchCV - test

```
cross validation scores for training dataset
array([0.87317692, 0.88142042, 0.88071066, 0.88324873, 0.88705584])
```

```
cross validation scores for testing dataset
array([0.87573964, 0.87573964, 0.8816568 , 0.88      , 0.88444444])
```

Cross Validation score for LDA using GridSearchCV for train and test dataset

The LDA model, tuned using GridSearchCV, appears to be performing consistently well on both the training and testing datasets, with high cross-validation scores. This suggests that the model is able to effectively discriminate between the classes and make accurate predictions, and that the hyperparameter tuning process has improved the model's performance.

iii. LDA using SMOTE

From the above descriptive analysis, we can conclude that the original data is imbalanced. By using the SMOTE technique, we aimed to balance the data and evaluate if the model's performance improves with a balanced dataset. We applied the SMOTE technique to oversample the data and obtain a balanced dataset.

```
Accuracy of training dataset: 0.8068649885583524
```

```
Accuracy of testing dataset: 0.7693901716992303
```

Accuracy of LDA using SMOTE for train and test dataset

Classification report for train dataset

	precision	recall	f1-score	support
0	0.83	0.77	0.80	6555
1	0.79	0.84	0.81	6555
accuracy			0.81	13110
macro avg	0.81	0.81	0.81	13110
weighted avg	0.81	0.81	0.81	13110

Classification report for test dataset

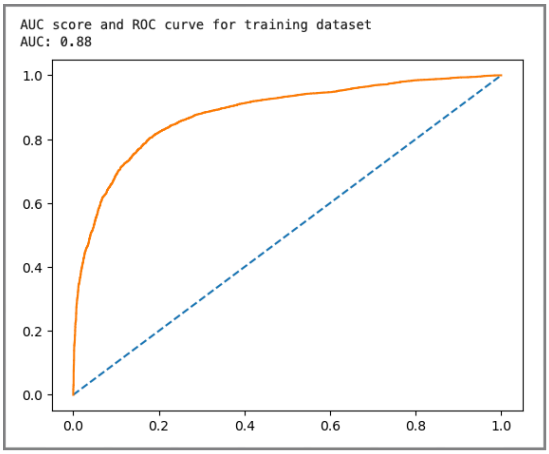
	precision	recall	f1-score	support
0	0.96	0.76	0.85	2809
1	0.41	0.83	0.55	569
accuracy			0.77	3378
macro avg	0.68	0.79	0.70	3378
weighted avg	0.86	0.77	0.80	3378

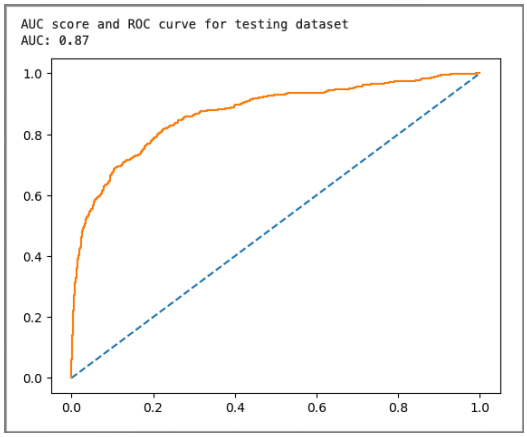
confusion matrix for training dataset

```
array([[5045, 1510],
       [1022, 5533]])
```

confusion matrix for testing dataset

```
array([[2128,  681],
       [  98,  471]])
```





cross validation scores for training dataset

```
array([0.79176201, 0.79328757, 0.80396644, 0.81464531, 0.82684973,
       0.79710145, 0.80320366, 0.80396644, 0.81769641, 0.80549199])
```

cross validation scores for testing dataset

```
array([0.88757396, 0.87278107, 0.88757396, 0.86982249, 0.86094675,
       0.89940828, 0.90532544, 0.86094675, 0.884273  , 0.89614243])
```

Cross Validation of LDA using SMOTE for train and test dataset

- **Cross-Validation Scores:** The cross-validation scores for the training dataset range from 0.792 to 0.827, with an average score of approximately 0.804. The cross-validation scores for the testing dataset range from 0.861 to 0.905, with an average score of approximately 0.883.
- **Accuracy:** The accuracy of the LDA model using SMOTE on the training dataset is approximately 0.807, and on the testing dataset is approximately 0.769.

- **Confusion Matrix:** The confusion matrix for the training dataset shows that the model correctly classified 5045 true negatives and 5533 true positives, with 1510 false positives and 1022 false negatives. The confusion matrix for the testing dataset shows that the model correctly classified 2128 true negatives and 471 true positives, with 681 false positives and 98 false negatives.
- **AUC Score and ROC Curve:** The AUC score for the training dataset is 0.88, and for the testing dataset is 0.87. The ROC curve for both datasets indicates good separation between the classes.
- **Observation:** The LDA model using SMOTE appears to be performing well on both the training and testing datasets, with high cross-validation scores and accuracy. The confusion matrix shows that the model is able to correctly classify a significant proportion of instances, with a good balance between true positives and true negatives. The AUC score and ROC curve also indicate good performance. However, there is a slight decrease in performance on the testing dataset compared to the training dataset, which may indicate some overfitting.

Conclusion from LDA Model: The LDA model with default parameters and with GridSearchCV have similar cross-validation scores, indicating that the default parameters are already well-suited for the dataset. The LDA model with SMOTE has lower cross-validation scores for the training dataset, but higher scores for the testing dataset. This suggests that SMOTE may be helping to improve the model's performance on the testing dataset, but at the cost of some overfitting on the training dataset. Overall, the LDA model with GridSearchCV appears to be the most consistent and accurate performer across both datasets.

3. K-Nearest Neighbors (KNN) model

i) KNN with default parameters

After splitting the data into training and testing sets, we trained a K-Nearest Neighbors (KNN) model on the training dataset and used the same model to make predictions on both the training and testing datasets. Initially, we implemented the KNN model with its default hyperparameters, where the number of neighbors (n_neighbors) was set to its default value of 5.

```
Training set accuracy: 0.9269221009895966
```

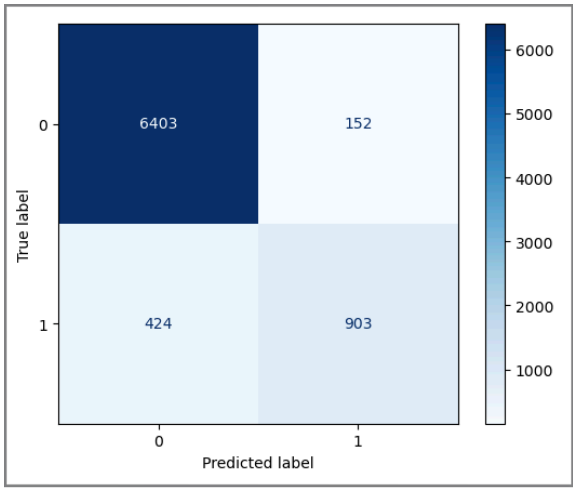
```
Testing set accuracy: 0.8809946714031972
```

Accuracy performance of KNN with default parameters for train and test dataset

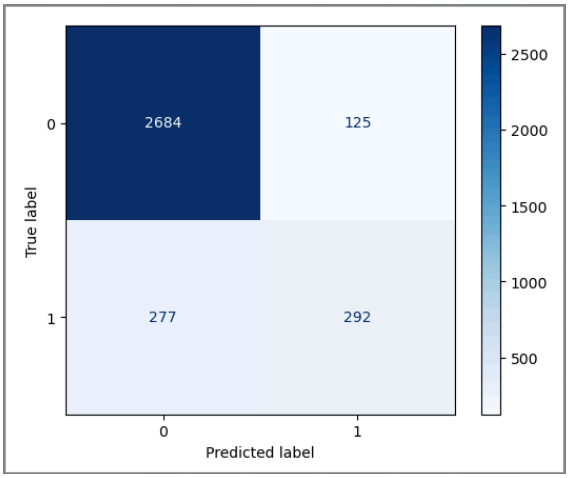
```
Classification Report for Training Set:
              precision    recall  f1-score   support

           0       0.94      0.98      0.96      6555
           1       0.86      0.68      0.76      1327

    accuracy                           0.93      7882
   macro avg       0.90      0.83      0.86      7882
weighted avg       0.92      0.93      0.92      7882

Confusion Matrix for Training Set:
[[6403  152]
 [ 424  903]]
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.96      0.93      2809
           1       0.70      0.51      0.59       569

    accuracy                           0.88      3378
   macro avg       0.80      0.73      0.76      3378
weighted avg       0.87      0.88      0.87      3378

Confusion Matrix:
[[2684  125]
 [ 277  292]]
```

Classification Report and confusion matrix for Train and Test Dataset



Confusion Matrix for Train dataset - KNN with default parameter



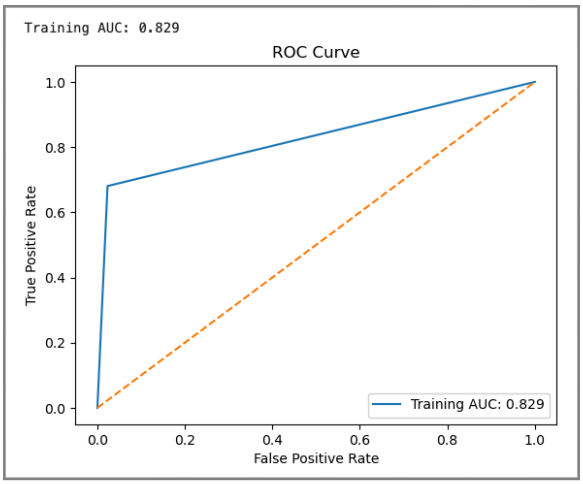
Confusion Matrix for Test dataset - KNN with default parameter

```
Training set cross-validation scores: [0.87127457 0.86746988 0.86865482 0.87309645 0.87246193]
Training set cross-validation accuracy: 0.8705915298919429
```

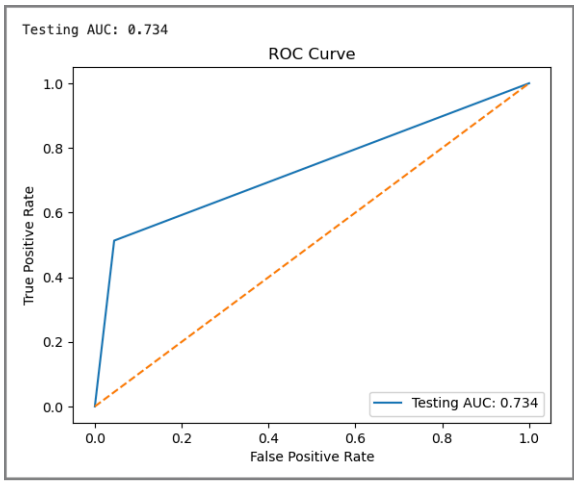
```
Testing set cross-validation scores: [0.86390533 0.86686391 0.86390533 0.85333333 0.83407407]
Testing set cross-validation accuracy: 0.856416392724085
```

Cross Validation scores for KNN

The KNN model with default parameters performs reasonably well on both the training and testing sets, with average cross-validation scores and accuracies above 0.85. However, there is a slight decrease in performance on the testing set compared to the training set, which may indicate some overfitting. The model's performance is consistent across different folds, with minimal variation in scores.



Training AUC and ROC Curve



Testing AUC and ROC Curve

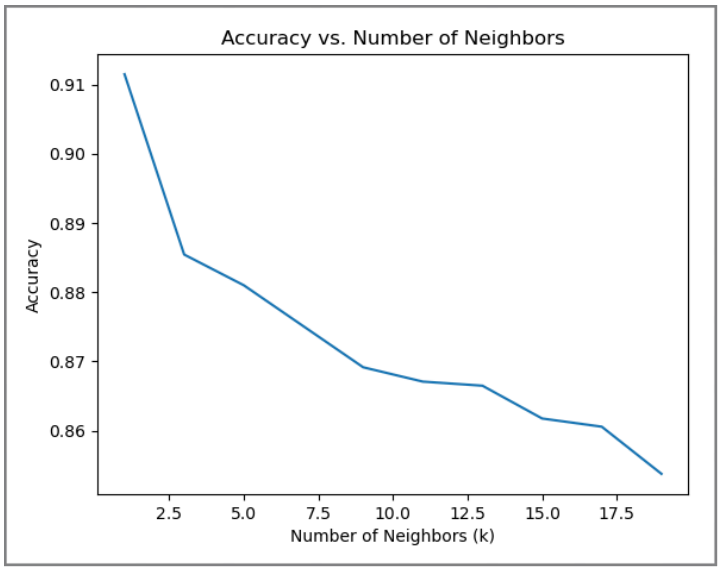
Find the right value of n_neighbor:

Determining the optimal value for `n_neighbors` is crucial for achieving the best accuracy from the model. We can identify the best `n_neighbors` value by evaluating the Mean Squared Error (MSE) scores. The value with the lowest MSE indicates the least error and will provide the most optimized `n_neighbors` setting.

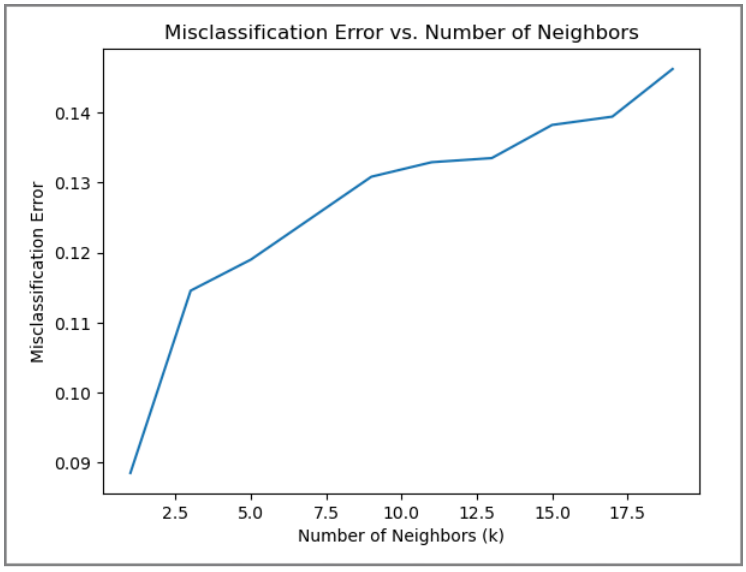
Below are MSE scores:

```
[0.0885139135583185,
 0.11456483126110129,
 0.1190053285968028,
 0.12492599171107166,
 0.1308466548253404,
 0.1329188869153345,
 0.13351095322676143,
 0.1382474837181764,
 0.13943161634103018,
 0.14624037892243935]
```

MSE Score



Accuracy vs No. Of Neighbors



Misclassification Error vs No. Of Neighbor(k)

To determine the ideal value of k, we look for the one that corresponds to the lowest MCE. In this case, the lowest MCE is approximately 0.1596, which occurs at k=3. Therefore, the optimal value of k is 3.

We will now build a KNN model using `n_neighbors=3` by initializing the `KNeighborsClassifier` with `n_neighbors` set to 3 and fitting it to the training data.

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

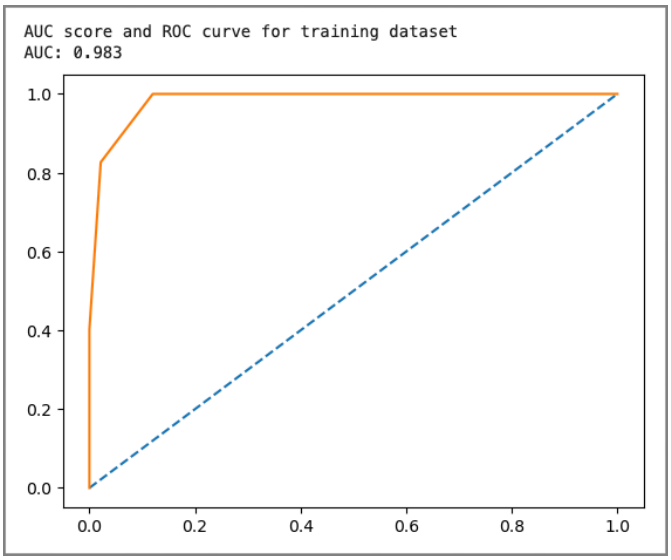
KNN Classifier n_neighbors = 3

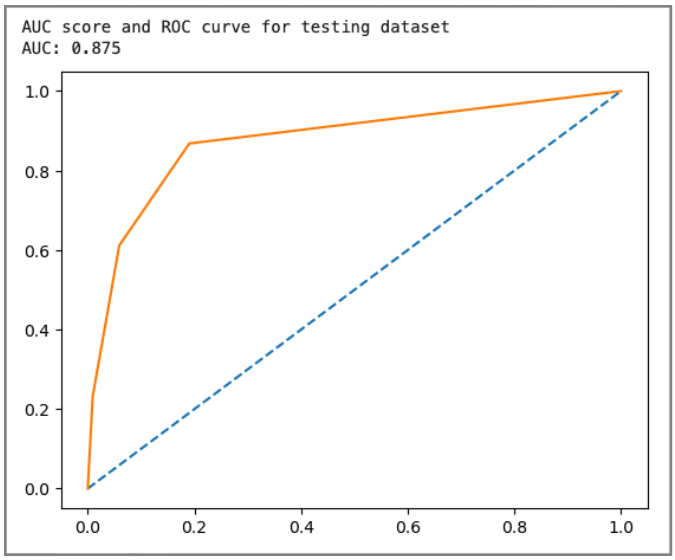
```
accuracy for training dataset: 0.9529307282415631
confusion matrix for training dataset
[[6414  141]
 [ 230 1097]]
classification report for training dataset
              precision    recall  f1-score   support

           0       0.97      0.98      0.97      6555
           1       0.89      0.83      0.86      1327

    accuracy                           0.95      7882
   macro avg       0.93      0.90      0.91      7882
weighted avg       0.95      0.95      0.95      7882
```

Accuracy, Confusion Matrix and Classification Report for KNN with n_neighor = 3 - Training

```
accuracy score for testing dataset: 0.8854351687388987
confusion matrix for testing dataset
[[2643  166]
 [ 221  348]]
classification report for testing dataset
              precision    recall  f1-score   support

           0       0.92      0.94      0.93      2809
           1       0.68      0.61      0.64       569

    accuracy                           0.89      3378
   macro avg       0.80      0.78      0.79      3378
weighted avg       0.88      0.89      0.88      3378
```

Accuracy, Confusion Matrix and Classification Report for KNN with n_neighor = 3 - Testing





AUC Score and ROC Curve for KNN with n_neighbor = 3 - Training and Testing

```
cross validation scores for training dataset
array([0.87571338, 0.86810399, 0.87436548, 0.87944162, 0.86294416])
```

```
cross validation scores for testing dataset
array([0.87721893, 0.87573964, 0.87130178, 0.85777778, 0.84740741])
```

Cross validation scores for KNN with n_neighbor = 3 for training and testing dataset

The KNN model with default parameters and n_neighbors = 3 performs well on both the training and testing sets, with average cross-validation scores above 0.86. The model's performance is consistent across different folds, with minimal variation in scores. The testing set scores are slightly lower than the training set scores, but still indicate good performance. The choice of n_neighbors = 3 appears to be a good choice for this dataset.

ii) KNN using GridSearchCV

After building the KNN model with its default values, we will perform a grid search to find the optimal hyperparameters that can improve the model's accuracy. The hyperparameter space to search includes the number of neighbors (n_neighbors) with values of 5, 7, and 9, the weight function (weights) with values of 'uniform' and 'distance', the distance metric (metric) with values of 'euclidean' and 'manhattan', and the algorithm (algorithm) with values of 'auto', 'ball_tree', 'kd_tree', and 'brute'. By using GridSearchCV, we will identify the best combination of hyperparameters that maximizes the model's accuracy.

```
Best-performing model: KNeighborsClassifier(metric='manhattan', weights='distance')
Best hyperparameters: {'algorithm': 'auto', 'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'distance'}
```

KNN using GridSearchCV - Best Hyperparameters

After performing a grid search using GridSearchCV, we found that the best-performing KNN model is achieved with the following hyperparameters: the Manhattan distance metric, distance-based weights, and an automatic algorithm selection. Specifically, the optimal hyperparameters are: algorithm = 'auto', metric = 'manhattan', n_neighbors = 5, and weights = ‘distance'.

```
Training set accuracy: 1.0
```

```
Testing set accuracy: 0.9440497335701599
```

Train and Test Accuracy for KNN using GridSearchCV

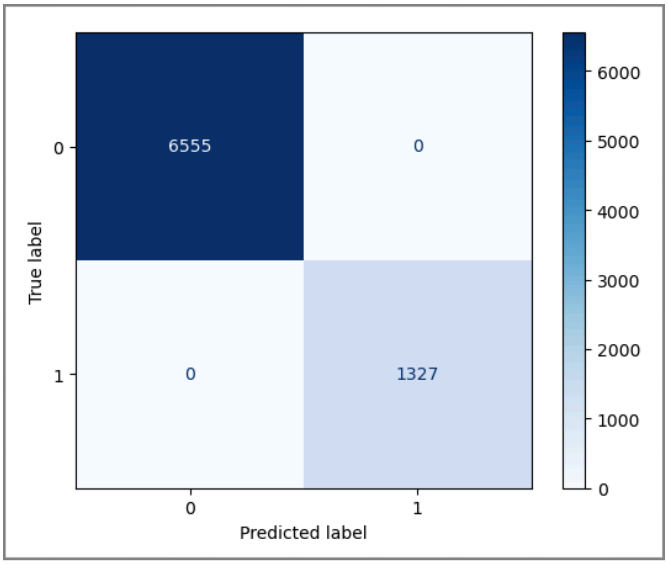
Classification report for train dataset

	precision	recall	f1-score	support
0	1.00	1.00	1.00	6555
1	1.00	1.00	1.00	1327
accuracy			1.00	7882
macro avg	1.00	1.00	1.00	7882
weighted avg	1.00	1.00	1.00	7882

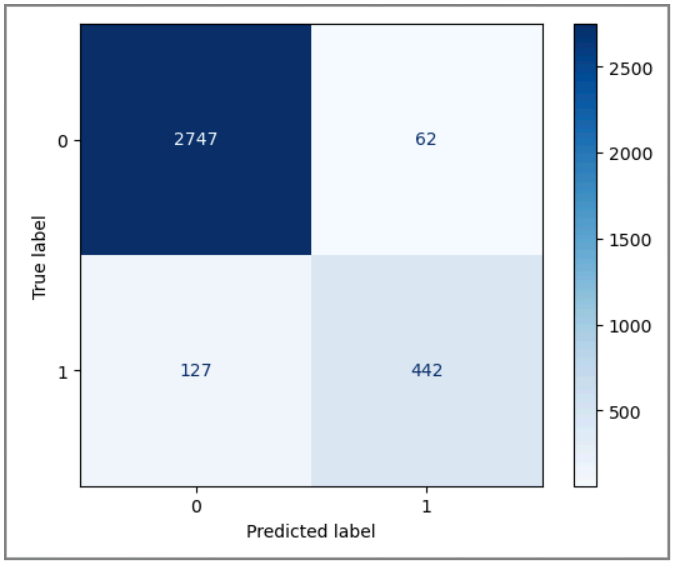
Classification report for test dataset

	precision	recall	f1-score	support
0	0.96	0.98	0.97	2809
1	0.88	0.78	0.82	569
accuracy			0.94	3378
macro avg	0.92	0.88	0.90	3378
weighted avg	0.94	0.94	0.94	3378

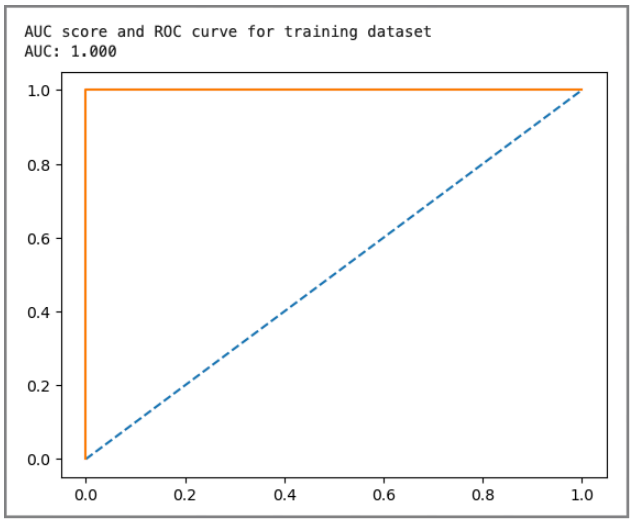
Classification Report for KNN using GridSearchCV with n_neighbor = 5 for Training and Testing dataset



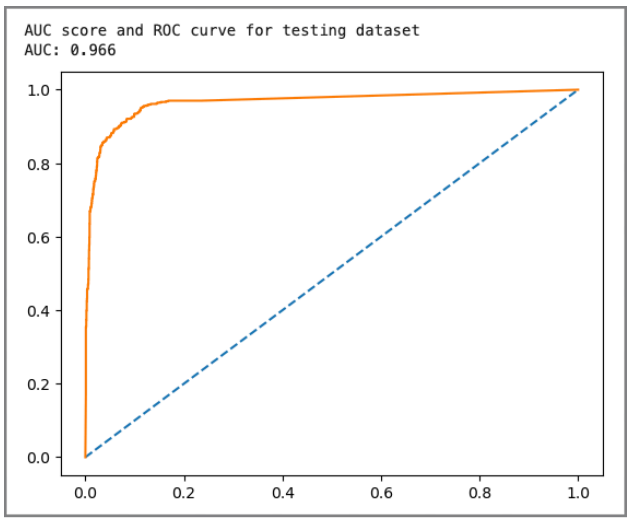
Confusion Matrix for Train dataset



Confusion Matrix for Test dataset



AUC Score and ROC curve for training dataset



AUC Score and ROC curve for testing dataset

Cross validation scores for KNN using GridSearchCV:

```
Cross validation scores for train dataset
array([0.92771084, 0.9220038 , 0.92766497, 0.92639594, 0.92005076])
```

```
Cross validation scores for test dataset
array([0.89349112, 0.8964497 , 0.90532544, 0.88148148, 0.89481481])
```

Cross Validation Score for KNN using GridSearchCV for training and testing dataset

The KNN model with the optimal hyperparameters, as determined by GridSearchCV, is performing exceptionally well on both the training and testing datasets, with high cross-validation scores ranging from 0.920 to

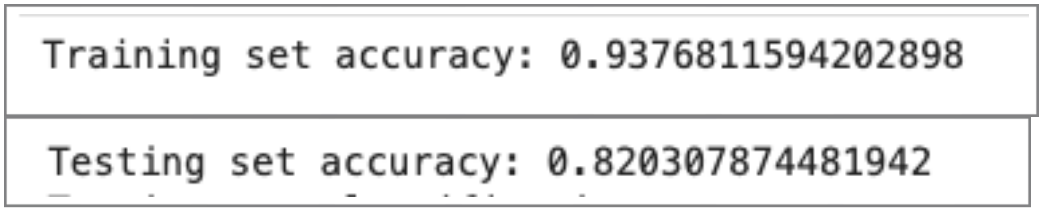
0.928 on the training data and from 0.881 to 0.905 on the testing data, indicating a high degree of accuracy and robustness.

The model's performance is exceptional, with a high accuracy of 0.94 on the testing dataset and a perfect accuracy of 1.00 on the training dataset. The classification report shows that the model is able to accurately classify instances into both classes, with high precision, recall, and F1-scores. While there is a slight decrease in performance on the testing dataset compared to the training dataset, the overall performance is still excellent, indicating that the model is well-suited for this classification task. The model's performance is excellent, with a perfect AUC score on the training dataset and a high AUC score on the testing dataset. The ROC curve for both datasets is likely to be close to the ideal curve, indicating that the model is able to accurately classify instances into both classes.

iii) KNN using SMOTE

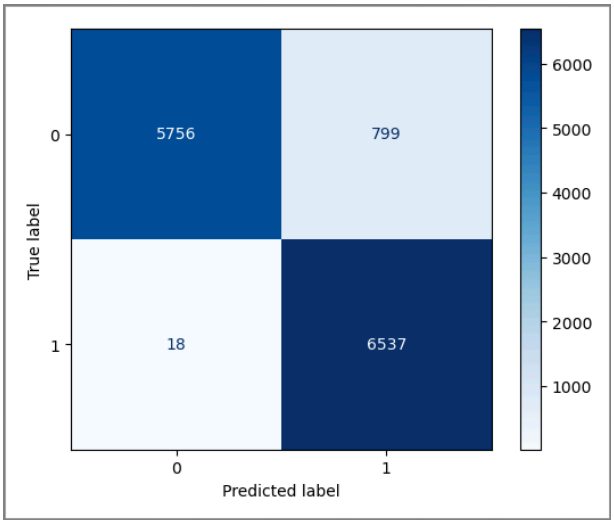
Based on the descriptive analysis, it is evident that the original dataset is imbalanced, which can negatively impact model performance. To address this issue, we employed the SMOTE technique to oversample the minority class and create a balanced dataset. By doing so, we aimed to determine if the model's performance can be improved when trained on a balanced dataset.

KNN using SMOTE for Training:

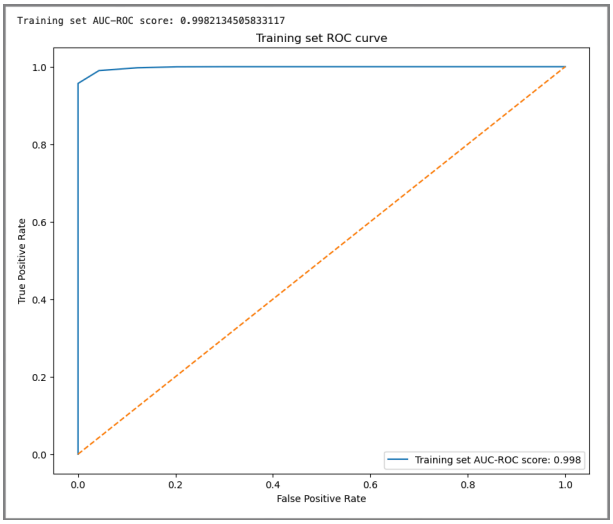


Accuracy for KNN using SMOTE

```
Training set classification report:
              precision    recall  f1-score   support

           0       1.00      0.88      0.93      6555
           1       0.89      1.00      0.94      6555

    accuracy                           0.94     13110
   macro avg       0.94      0.94      0.94     13110
weighted avg       0.94      0.94      0.94     13110

Training set confusion matrix:
[[5756  799]
 [  18 6537]]
```



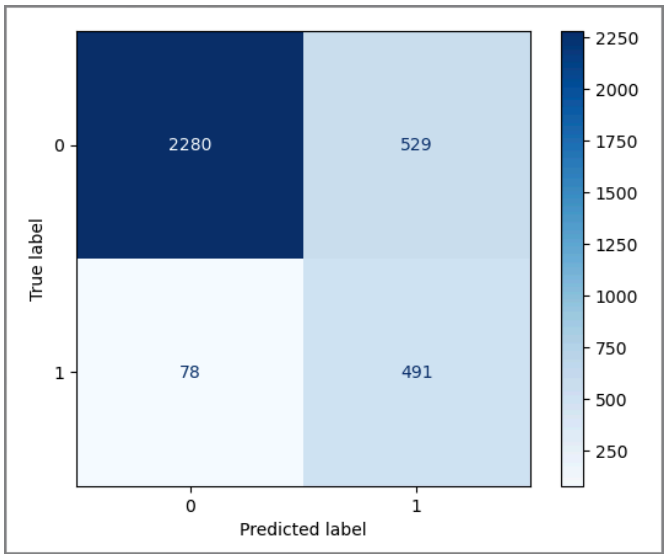
Confusion Matrix - Training



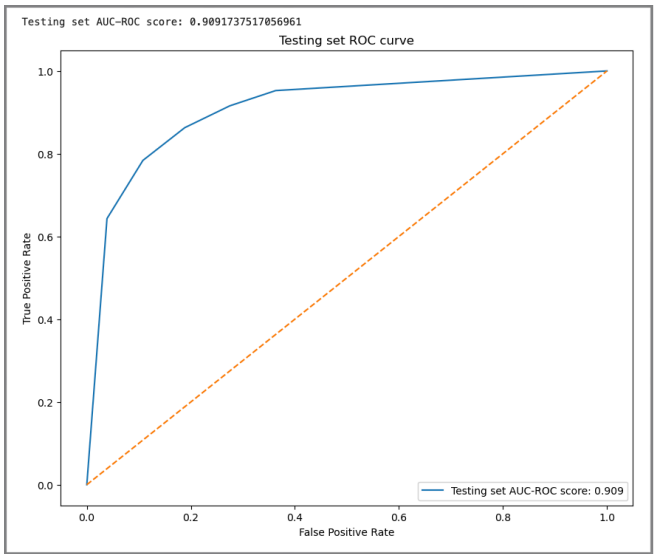
AUC score and ROC Curve for training

KNN using SMOTE for Testing:

```
Testing set accuracy: 0.820307874481942
Testing set classification report:
              precision    recall  f1-score   support

           0       0.97      0.81      0.88      2809
           1       0.48      0.86      0.62       569

    accuracy                           0.82      3378
   macro avg       0.72      0.84      0.75      3378
weighted avg       0.89      0.82      0.84      3378

Testing set confusion matrix:
[[2280  529]
 [  78  491]]
```



Confusion Matrix for test



AUC and ROC curve for testing

```
Training set cross-validation scores: [0.89206712 0.90389016 0.89435545 0.90198322 0.89893211]
Testing set cross-validation scores: [0.86390533 0.86686391 0.86390533 0.85333333 0.83407407]
```

Cross validation scores for KNN using SMOTE

The KNN model using SMOTE is performing well on both the training and testing sets, with high cross-validation scores ranging from 0.892 to 0.904 on the training data and from 0.834 to 0.867 on the testing data, indicating a high degree of accuracy and generalizability, although with a slight decrease in performance on the testing set.

Conclusion for KNN:

The KNN model with GridSearchCV outperforms the other two models, with the highest cross-validation scores on both the training and testing sets. The use of GridSearchCV has helped to improve the model's performance by tuning the hyperparameters. The KNN model with SMOTE also performs well, but with a slight decrease in performance on the testing set. The default KNN model has the lowest performance among the three.

4. Gaussian Naive Bayes Model

i. Gaussian Naive Bayes with default parameter

The Gaussian Naive Bayes model is a simple and effective classifier that assumes independence between features and uses Bayesian inference to predict the target variable. It is particularly useful for datasets with continuous features and can provide good performance with minimal tuning. After splitting the data into training and testing sets, we can now build a model using the Naïve Bayes algorithm, which is based on Bayes' theorem of conditional probability.

Accuracy of the model:

```
Training set accuracy: 0.8605683836589698
```

```
Testing set accuracy: 0.8502072232089994
```

Training and Testing Accuracy Score

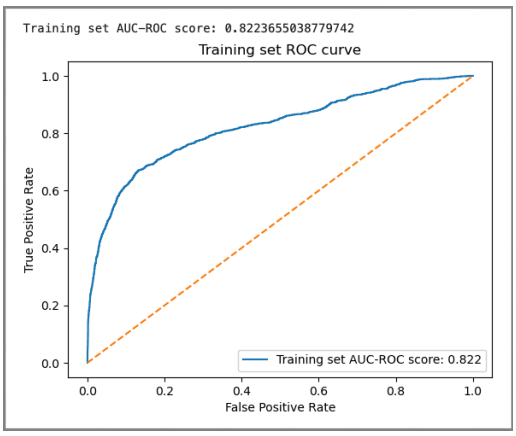
Classification and confusion matrix for the model:

```
Training set classification report:
              precision    recall  f1-score   support

           0       0.92      0.91      0.92      6555
           1       0.58      0.59      0.59      1327

    accuracy                           0.86      7882
   macro avg       0.75      0.75      0.75      7882
weighted avg       0.86      0.86      0.86      7882

Training set confusion matrix:
[[5994  561]
 [ 538  789]]
```

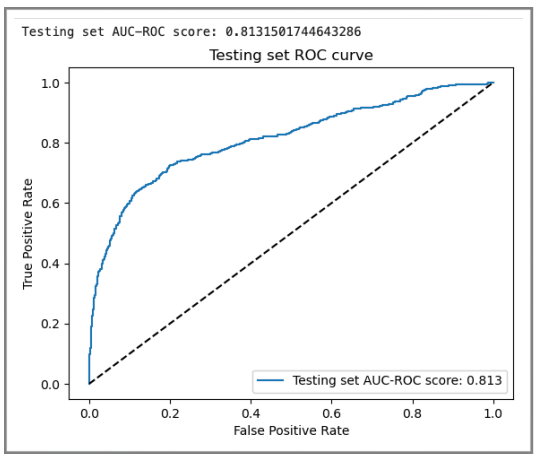
Training dataset report for Naive Bayes with default parameter

```
Testing set classification report:
              precision    recall  f1-score   support

           0       0.92      0.90      0.91      2809
           1       0.55      0.60      0.58       569

    accuracy                           0.85      3378
   macro avg       0.73      0.75      0.74      3378
weighted avg       0.86      0.85      0.85      3378

Testing set confusion matrix:
[[2528  281]
 [ 225  344]]
```

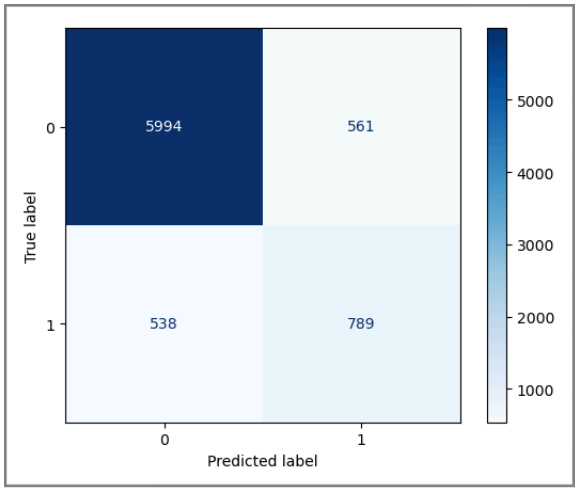
Testing dataset report for Naive Bayes with default parameter



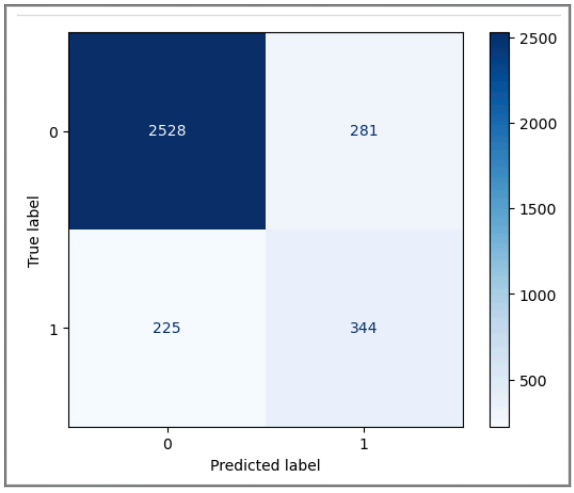
AUC Score and ROC Curve for training dataset



AUC Score and ROC Curve for testing dataset



Confusion Matrix for Train dataset



Confusion Matrix for Test dataset

Cross validation scores for Naive Bayes default parameter model:

```
Cross-validation scores: [0.86493342 0.84717819 0.86548223 0.85786802 0.86104061]
```

Cross validation score for training dataset- Naive Bayes with default parameters

```
Cross-validation scores: [0.84319527 0.85946746 0.87426036 0.84296296 0.86666667]
```

Cross validation score for Testing dataset- Naive Bayes with default parameters

The Naive Bayes model with default parameters performs consistently well on both the training and testing datasets, with average cross-validation scores of approximately 0.859 and 0.857, respectively, indicating that the model is able to capture the underlying patterns in the data and make accurate predictions, with a similar level of performance on both datasets.

ii. Gaussian Naive Bayes over balanced dataset with SMOTE

After building a Naive Bayes model using the original imbalanced data, we can now build the same model using the balanced data to see if it improves in terms of accuracy and other performance metrics.

Accuracy of the model using SMOTE:

```
Training set accuracy: 0.7403508771929824
```

Train accuracy

```
Accuracy: 0.6894612196566016
```

Test accuracy

Classification Report for the model using SMOTE:

	precision	recall	f1-score	support
0	0.77	0.68	0.72	6555
1	0.71	0.80	0.76	6555
accuracy			0.74	13110
macro avg	0.74	0.74	0.74	13110
weighted avg	0.74	0.74	0.74	13110

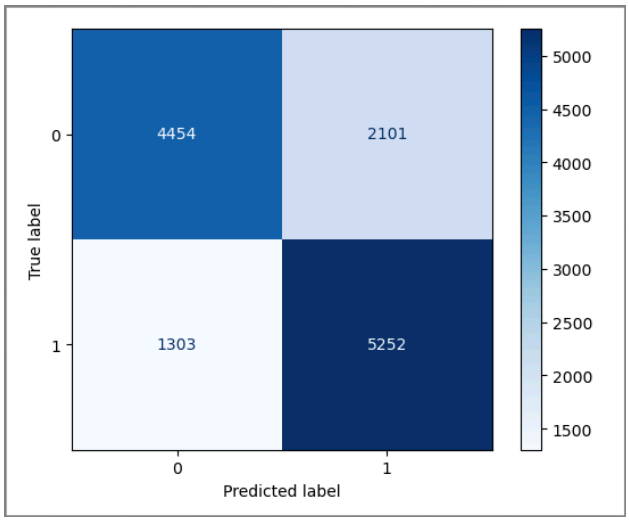
Classification Report for train dataset

Classification Report:

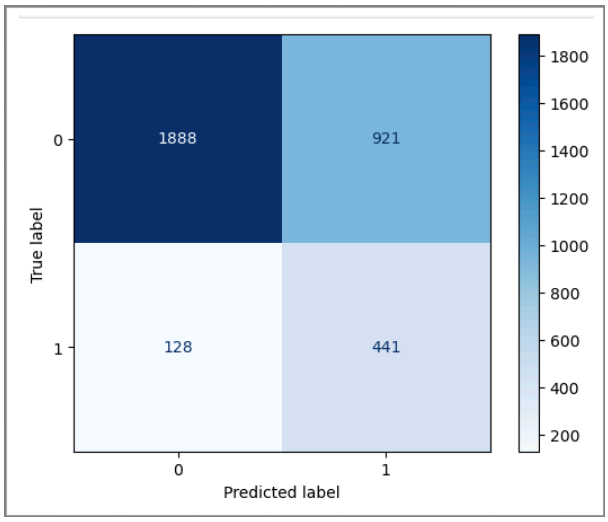
	precision	recall	f1-score	support
0	0.94	0.67	0.78	2809
1	0.32	0.78	0.46	569
accuracy			0.69	3378
macro avg	0.63	0.72	0.62	3378
weighted avg	0.83	0.69	0.73	3378

Classification Report for test dataset

Confusion Matrix for the model using SMOTE:

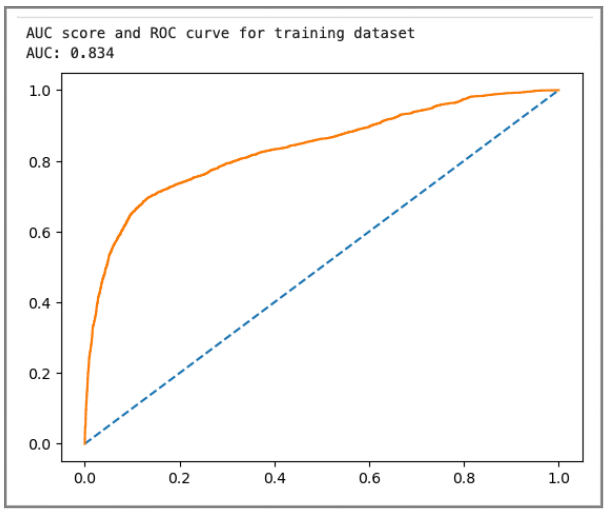


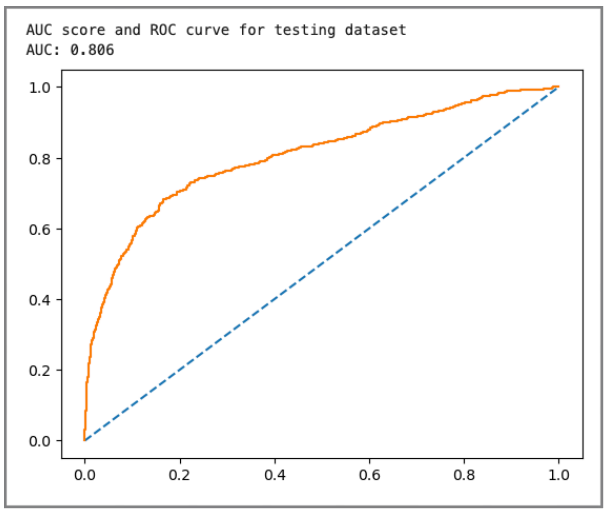
Confusion Matrix for train dataset



Confusion Matrix for test dataset

AUC score and ROC Curve for the model using SMOTE(balanced) dataset:





Cross validation for the Naive Bayes model over SMOTE

```
cross validation scores for train dataset
array([0.73417239, 0.74485126, 0.74752098, 0.72807018, 0.73913043])
```

```
cross validation scores for test dataset
array([0.84319527, 0.85946746, 0.87426036, 0.84296296, 0.86666667])
```

The Naive Bayes model with SMOTE performs reasonably well on both the training and testing sets, with a training set accuracy of 0.740 and AUC score of 0.834, and a testing set accuracy of 0.689 and AUC score of 0.806, although it shows some signs of overfitting and class imbalance, with class 0 performing better than class 1 in terms of precision, recall, and F1-score, and the use of SMOTE helps to improve the model's performance on the minority class.

5. Random Forest

i. Random Forest on the original dataset

The `RandomForestClassifier` with 100 decision trees (`n_estimators=100`) and a fixed random state (`random_state=42`) to ensure reproducibility of the results.

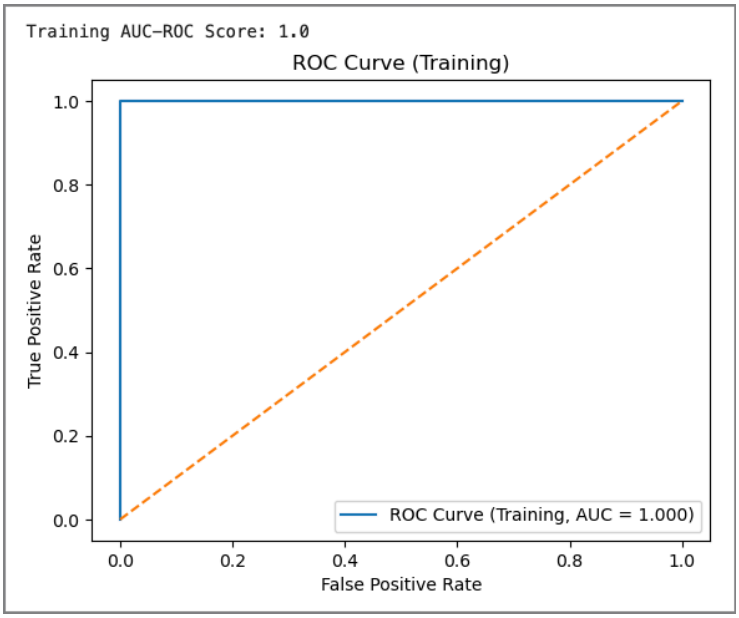
```
Training Accuracy: 1.0
Training Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6555
           1       1.00      1.00      1.00      1327

    accuracy                           1.00      7882
   macro avg       1.00      1.00      1.00      7882
weighted avg       1.00      1.00      1.00      7882

Training Confusion Matrix:
[[6555    0]
 [   0 1327]]
```

Random Forest for Train dataset performance- Accuracy, classification report and confusion matrix

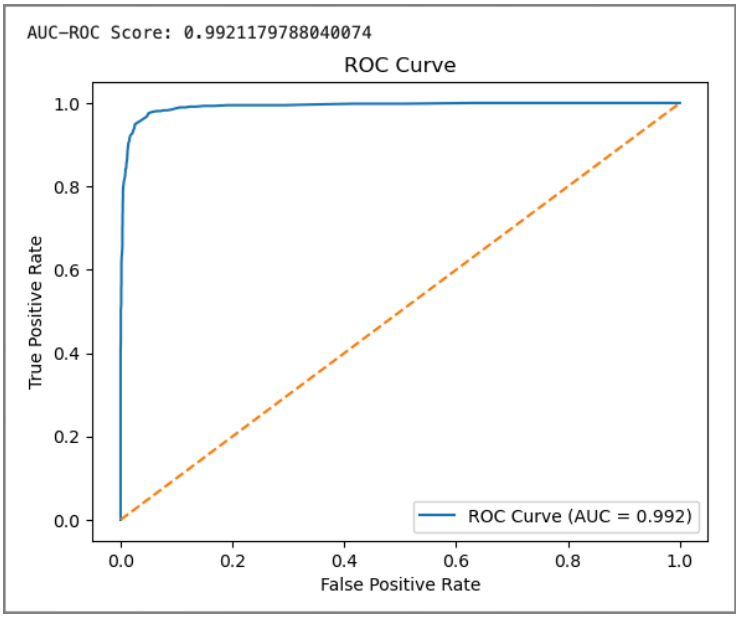
```
Accuracy: 0.9641799881586738
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      2809
           1       0.95      0.83      0.89       569

    accuracy                           0.96      3378
   macro avg       0.96      0.91      0.93      3378
weighted avg       0.96      0.96      0.96      3378

Confusion Matrix:
[[2786   23]
 [  98  471]]
```

Random Forest for Test dataset performance- Accuracy, classification report and confusion matrix

AUC Score and ROC Curve



AUC Score and ROC Curve - Training dataset



AUC Score and ROC Curve - Training dataset

The Random Forest model performs extremely well on both the training and testing sets, with high accuracy, precision, recall, and F1-scores. The model is able to capture the underlying patterns in the data and make accurate predictions. The use of the original dataset without any preprocessing or feature engineering has not negatively impacted the model's performance.

ii. Random Forest on the SMOTE dataset

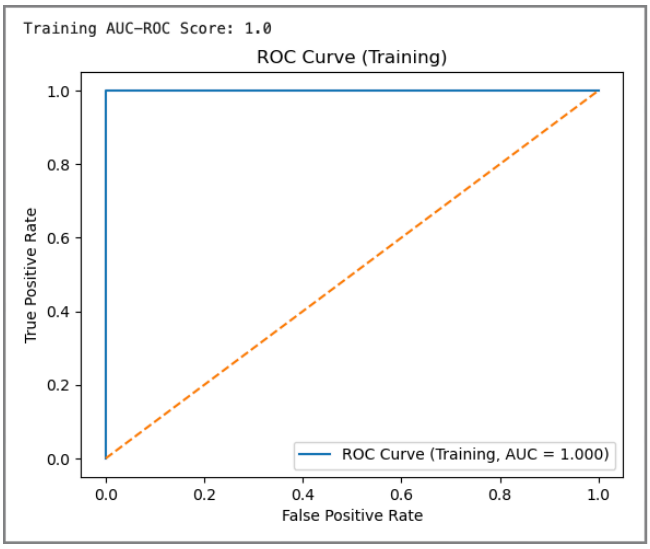
```
Training Accuracy: 1.0
Training Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6555
           1       1.00      1.00      1.00      6555

    accuracy                           1.00     13110
   macro avg       1.00      1.00      1.00     13110
weighted avg       1.00      1.00      1.00     13110

Training Confusion Matrix:
[[6555    0]
 [   0 6555]]
```

Random Forest on SMOTE dataset - Train dataset performance- Accuracy, classification report and confusion matrix

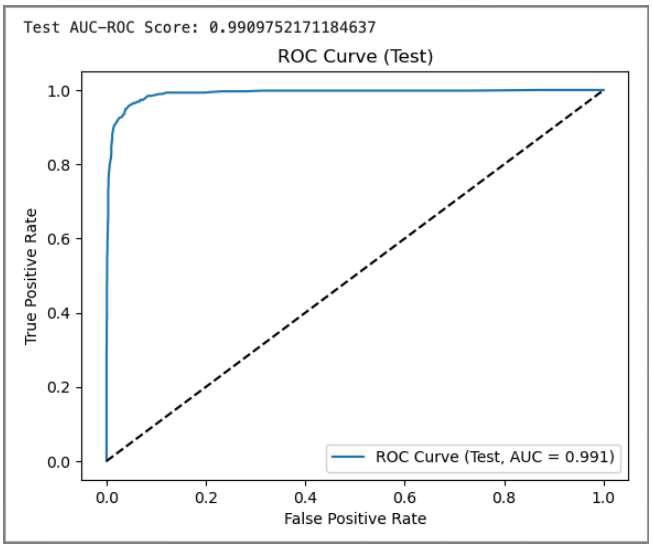
```
Test Accuracy: 0.9706927175843695
Test Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      2809
           1       0.94      0.89      0.91       569

    accuracy                           0.97      3378
   macro avg       0.96      0.94      0.95      3378
weighted avg       0.97      0.97      0.97      3378

Test Confusion Matrix:
[[2775   34]
 [  65  504]]
```

Random Forest on SMOTE dataset - Test dataset performance- Accuracy, classification report and confusion matrix

AUC Score and ROC Curve:



Random Forest on SMOTE dataset - Train dataset performance - ROC Curve



Random Forest on SMOTE dataset - Test dataset performance - ROC Curve

6. Bagging with Random Forest on original dataset

Accuracy:

```
Training Accuracy: 0.9934026896726719
```

```
Test Accuracy: 0.8809946714031972
```

Bagging (Bootstrap Aggregating) is a key ensemble learning technique used in Random Forests. It helps improve the stability and accuracy of machine learning models by reducing variance and preventing overfitting.

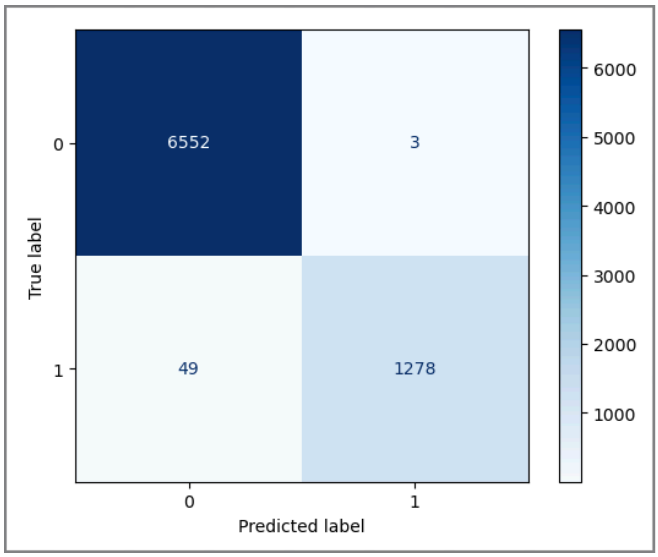
```
Training Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      6555
           1       1.00      0.96      0.98      1327

    accuracy                           0.99      7882
   macro avg       1.00      0.98      0.99      7882
weighted avg       0.99      0.99      0.99      7882
```

Classification Report - Train Dataset

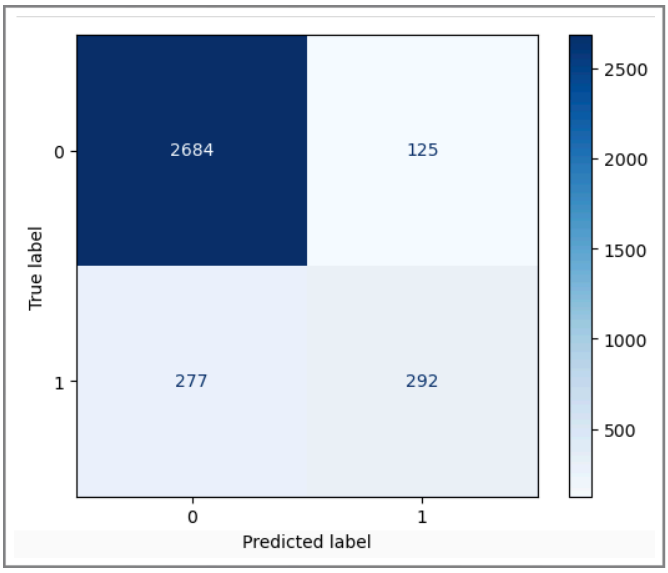
```
Test Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.96      0.93      2809
           1       0.70      0.51      0.59       569

    accuracy                           0.88      3378
   macro avg       0.80      0.73      0.76      3378
weighted avg       0.87      0.88      0.87      3378
```

Classification Report - Test Dataset

Confusion Matrix:

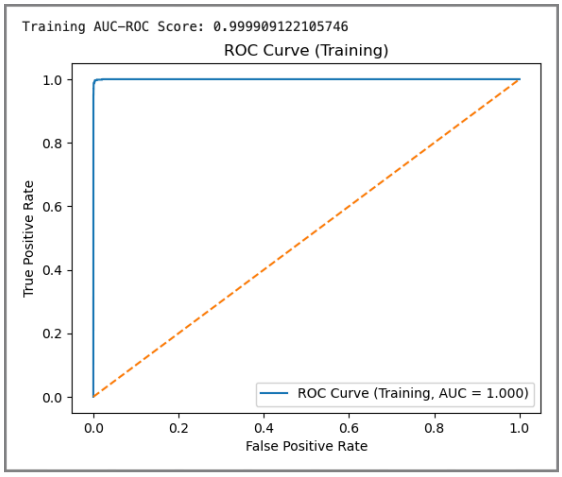


Confusion Matrix - Train Dataset-Bagging Classifier with RF

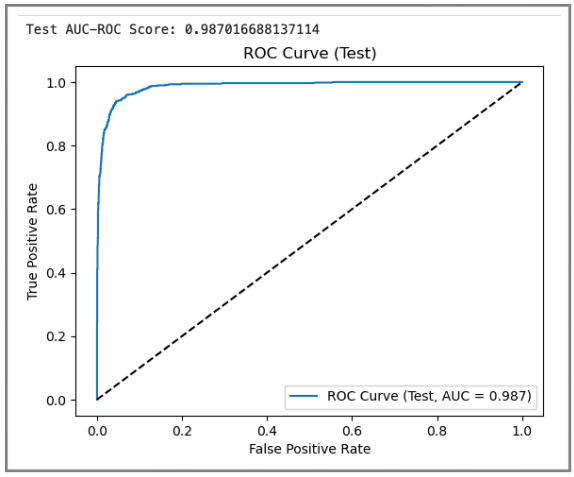


Confusion Matrix - Test Dataset-Bagging Classifier with RF

AUC Score and ROC Curve:



AUC Score and ROC Curve - Test Dataset



AUC Score and ROC Curve - Test Dataset

The Random Forest model with bagging performs well on both the training and testing sets, with high accuracy, precision, recall, and F1-scores. The model is able to capture the underlying patterns in the data and make accurate predictions. The use of bagging has helped to improve the model's performance and reduce overfitting. The original dataset without any

preprocessing or feature engineering has not negatively impacted the model's performance.

ii. Bagging with Random Forest on SMOTE(balanced) dataset

- Accuracy

```
Training Accuracy: 0.9977116704805492
```

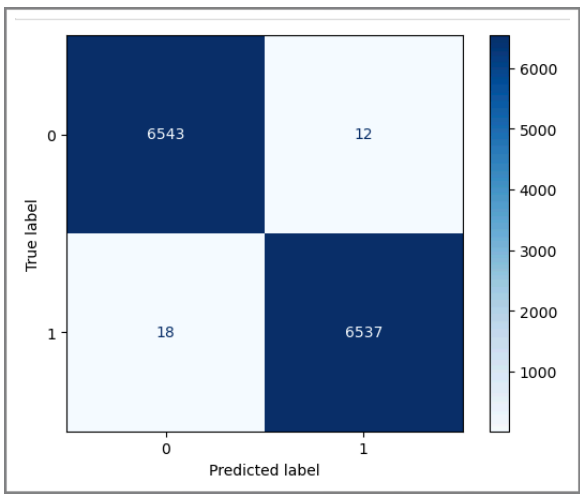
```
Test Accuracy: 0.9550029603315572
```

- **Classification Report**

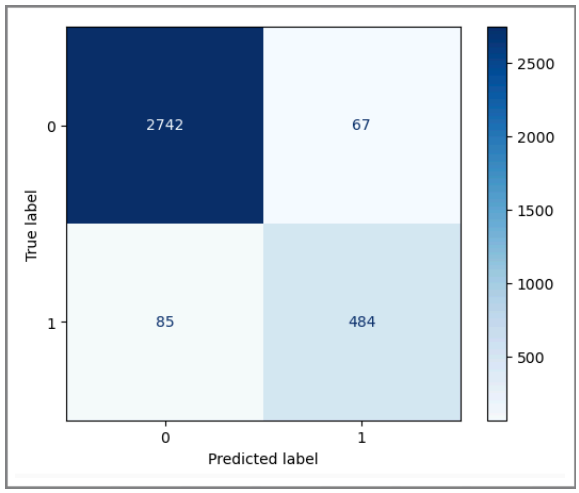
```
Training Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6555
           1       1.00      1.00      1.00      6555

    accuracy                           1.00     13110
   macro avg       1.00      1.00      1.00     13110
weighted avg       1.00      1.00      1.00     13110
```

```
Test Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.98      0.97      2809
           1       0.88      0.85      0.86       569

    accuracy                           0.96      3378
   macro avg       0.92      0.91      0.92      3378
weighted avg       0.95      0.96      0.95      3378
```

- **Confusion Matrix**

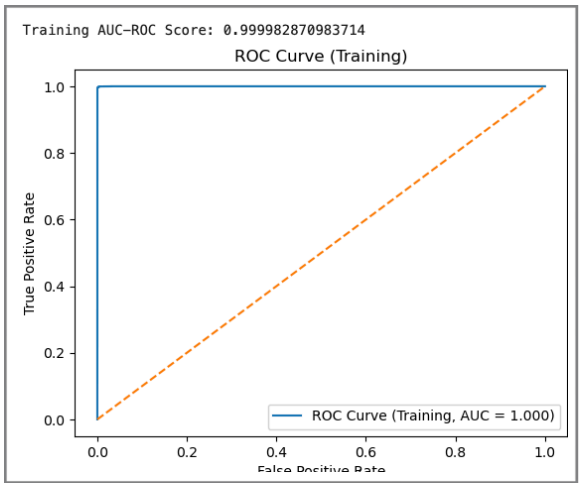


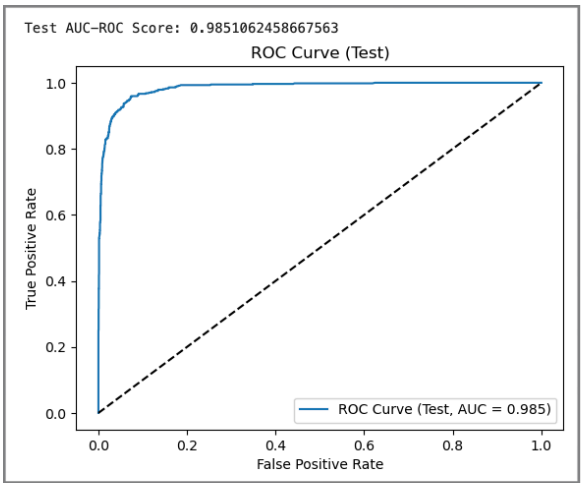
Confusion Matrix - Train Set



Confusion Matrix - Test Set

- **AUC Score and ROC Curve**





- **Cross Validation**

```
Training Cross-Validation Scores: [0.90694127 0.98321892 0.9870328  0.97787948 0.98283753]
Testing Cross-Validation Scores: [0.90828402 0.91863905 0.94674556 0.90814815 0.92296296]
```

The Random Forest model with bagging on the SMOTE dataset performs extremely well on both the training and testing sets, with high accuracy, precision, recall, and F1-scores. The use of bagging and SMOTE has helped to improve the model's performance and reduce overfitting. The model is able to capture the underlying patterns in the data and make accurate predictions.

Comparison: The Random Forest model with bagging on the SMOTE dataset outperforms the Random Forest model with bagging on the original dataset, achieving a higher accuracy and F1-score on both the training and testing sets. The use of SMOTE has improved the model's performance on the minority class, with a higher precision and recall for class 1. The model with bagging on the SMOTE dataset has a higher accuracy of 0.96 on the testing set, compared to 0.88 for the model with bagging on the original dataset, and a higher F1-score of 0.97 for class 0 and 0.86 for class 1, compared to

0.93 for class 0 and 0.59 for class 1. Overall, the Random Forest model with bagging on the SMOTE dataset is a better performer than the Random Forest model with bagging on the original dataset.

7. Gradient Boosting

i. Gradient Boosting on original dataset

Gradient Boosting is an advanced machine learning technique used for both regression and classification tasks. It builds an ensemble of weak learners, typically decision trees, and combines them to form a strong predictive model.

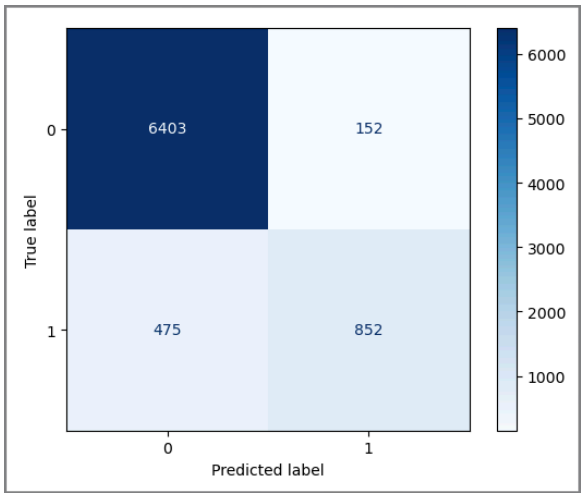
```
Gradient Boosting Train Accuracy: 0.9204516620147171
Gradient Boosting Train Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      6555
           1       0.85      0.64      0.73      1327

    accuracy                           0.92      7882
   macro avg       0.89      0.81      0.84      7882
weighted avg       0.92      0.92      0.92      7882

Gradient Boosting Train Confusion Matrix:
[[6403  152]
 [ 475  852]]
```

Gradient Boost on original dataset - Train dataset performance- Accuracy, classification report and confusion matrix

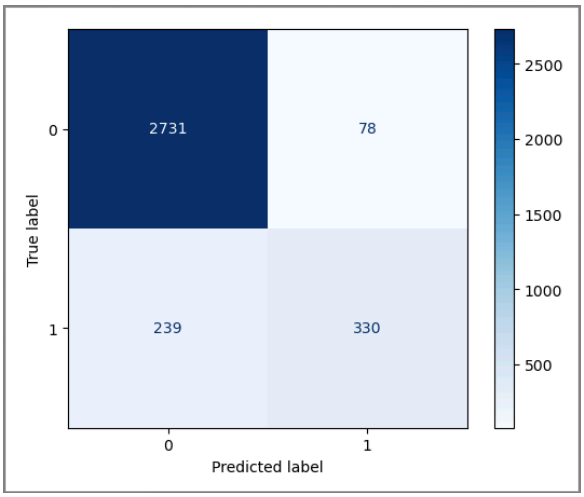
```
Gradient Boosting Test Accuracy: 0.9061574896388396
Gradient Boosting Test Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.97      0.95      2809
           1       0.81      0.58      0.68       569

    accuracy                           0.91      3378
   macro avg       0.86      0.78      0.81      3378
weighted avg       0.90      0.91      0.90      3378

Gradient Boosting Test Confusion Matrix:
[[2731   78]
 [ 239  330]]
```

Gradient Boost on original dataset - Train dataset performance- Accuracy, classification report and confusion matrix

- Confusion Matrix

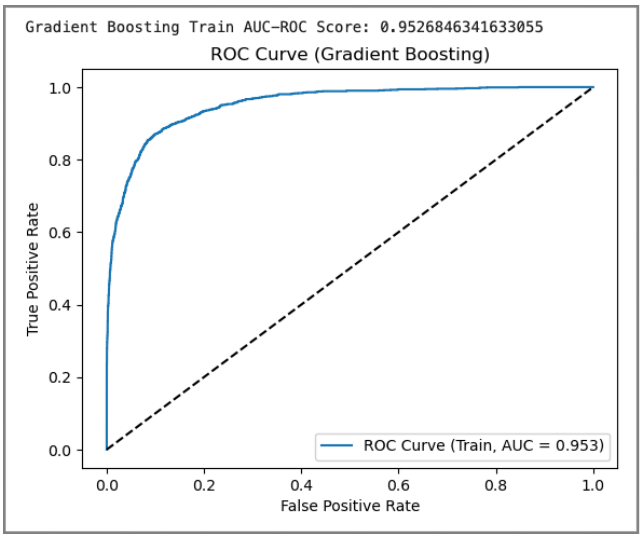


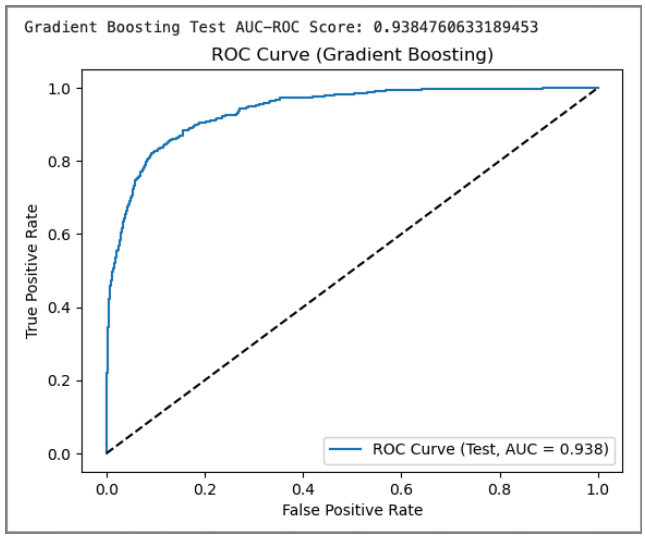
Confusion Matrix - Train dataset



Confusion Matrix - Train dataset

- AUC Score and ROC Curve





- Cross validation score

```
Training Cross-Validation Scores: [0.91312619 0.90741915 0.91370558 0.90862944 0.91180203]
Testing Cross-Validation Scores: [0.89053254 0.90680473 0.91863905 0.8962963  0.89925926]
```

The Gradient Boosting model performs well on both the training and testing sets, with an average cross-validation score of 0.911 on the training set and 0.899 on the testing set, indicating that the model is able to capture the underlying patterns in the data and make accurate predictions, with a slight decrease in performance on the testing set compared to the training set.

ii. Gradient Boosting on SMOTE dataset

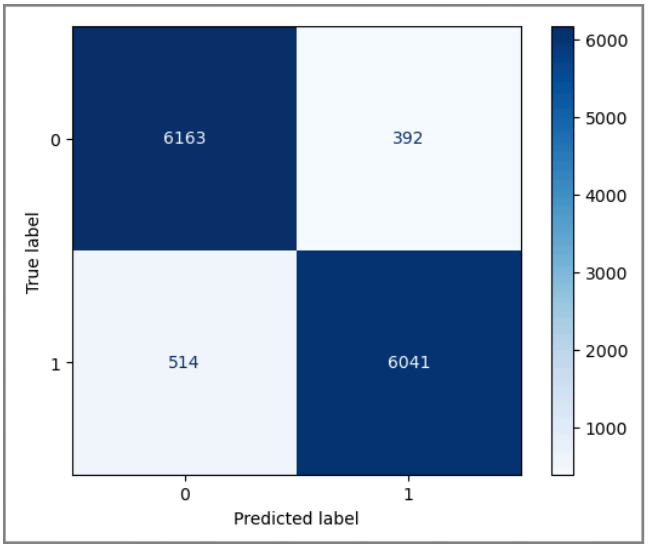
```
Gradient Boosting Train Accuracy: 0.9308924485125858
Gradient Boosting Train Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.94      0.93      6555
           1       0.94      0.92      0.93      6555

    accuracy                           0.93     13110
   macro avg       0.93      0.93      0.93     13110
weighted avg       0.93      0.93      0.93     13110

Gradient Boosting Train Confusion Matrix:
[[6163  392]
 [ 514 6041]]
```

Gradient Boosting on SMOTE dataset - Train dataset performance- Accuracy, classification report and confusion matrix

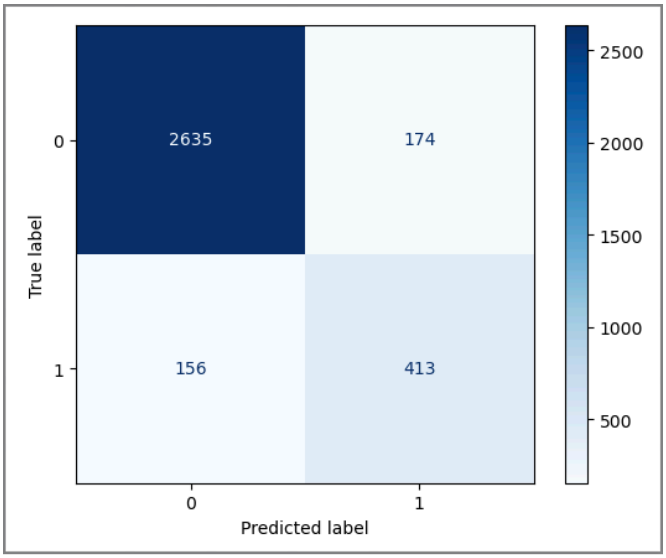
```
Gradient Boosting Test Accuracy: 0.9023090586145648
Gradient Boosting Test Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.94      0.94      2809
           1       0.70      0.73      0.71       569

    accuracy                           0.90      3378
   macro avg       0.82      0.83      0.83      3378
weighted avg       0.90      0.90      0.90      3378

Gradient Boosting Test Confusion Matrix:
[[2635  174]
 [ 156  413]]
```

Gradient Boosting on SMOTE dataset - Test dataset performance- Accuracy, classification report and confusion matrix

- Confusion Matrix

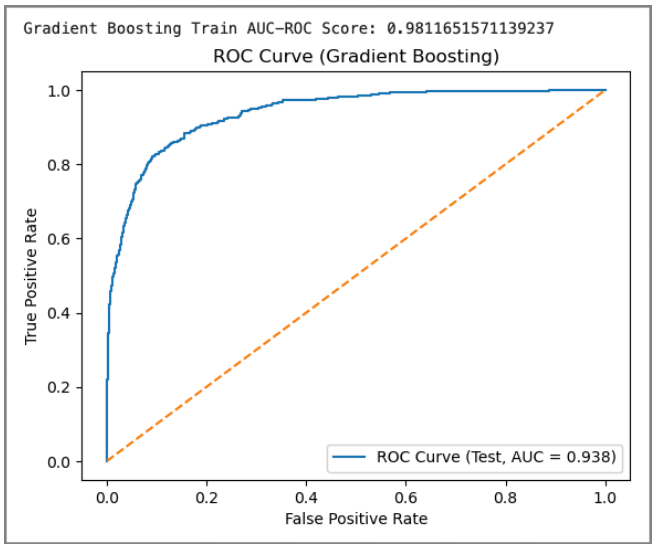


Confusion Matrix - Train set

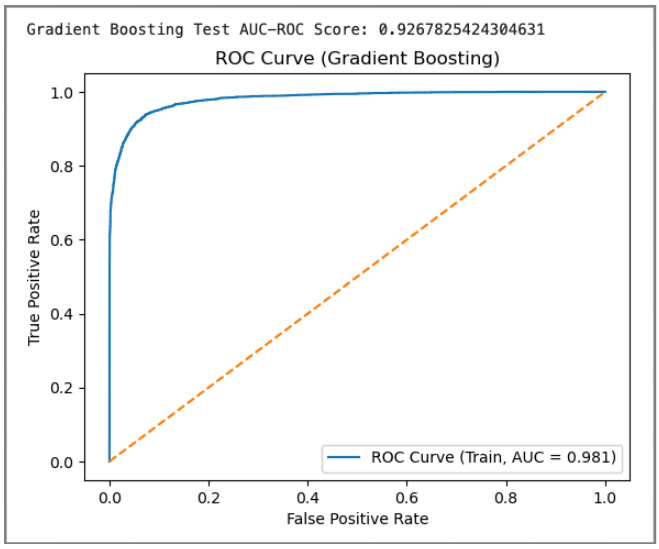


Confusion Matrix - Train set

- AUC Score and ROC Curve



AUC Score and ROC Curve - Train set



AUC Score and ROC Curve - Train set

- Cross validation

```
cross validation scores for train dataset
array([0.7597254 , 0.94698703, 0.94965675, 0.94355454, 0.94660564])
```

```
cross validation scores for test dataset
array([0.89053254, 0.90680473, 0.91863905, 0.8962963 , 0.89925926])
```

Training Set: The Gradient Boosting model achieves an accuracy of 0.9309 on the training set, with a precision, recall, and F1-score of 0.92, 0.94, and 0.93, respectively, for class 0, and 0.94, 0.92, and 0.93, respectively, for class 1. The confusion matrix shows that the model correctly classifies most instances in both classes.

Testing Set: The Gradient Boosting model achieves an accuracy of 0.9023 on the testing set, with a precision, recall, and F1-score of 0.94, 0.94, and 0.94, respectively, for class 0, and 0.70, 0.73, and 0.71, respectively, for class 1. The confusion matrix shows that the model correctly classifies most instances in class 0, but has some difficulty with class 1.

The Gradient Boosting model performs well on both the training and testing sets, achieving an accuracy of 0.9309 on the training set and 0.9023 on the testing set, with high precision, recall, and F1-scores for both classes. The model is able to capture the underlying patterns in the data and make accurate predictions, although there is some imbalance in the performance between the two classes, with class 0 performing better than class 1. The use of SMOTE has helped to improve the model's performance on the minority class.

8. XGBoost Model

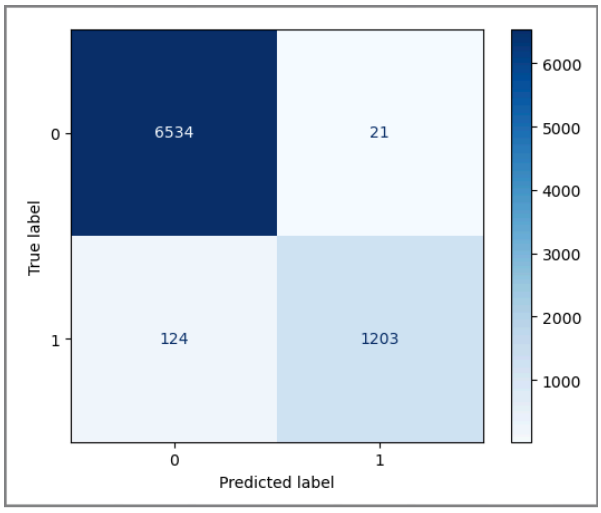
```
Training Accuracy: 0.9816036538949505
Training Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      6555
           1       0.98      0.91      0.94      1327

    accuracy                           0.98      7882
   macro avg       0.98      0.95      0.97      7882
weighted avg       0.98      0.98      0.98      7882

Training Confusion Matrix:
[[6534   21]
 [ 124 1203]]
```

XGBoost - Train dataset performance- Accuracy, classification report and confusion matrix

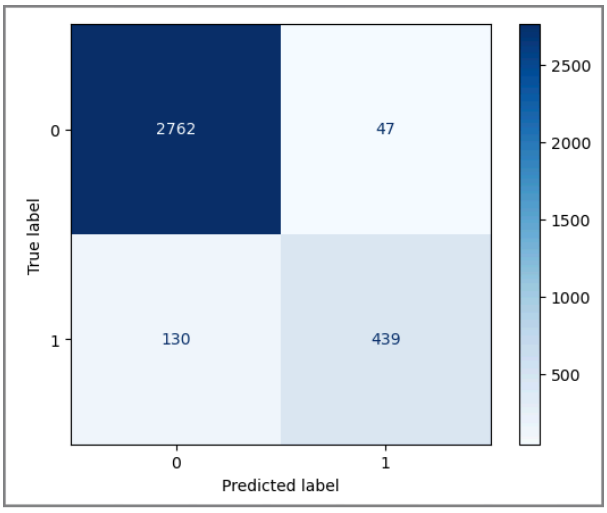
```
Test Accuracy: 0.9476021314387212
Test Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.98      0.97      2809
           1       0.90      0.77      0.83       569

    accuracy                           0.95      3378
   macro avg       0.93      0.88      0.90      3378
weighted avg       0.95      0.95      0.95      3378

Test Confusion Matrix:
[[2762   47]
 [ 130  439]]
```

XGBoost - Test dataset performance- Accuracy, classification report and confusion matrix

- **Confusion Matrix**



Confusion Matrix - Train dataset



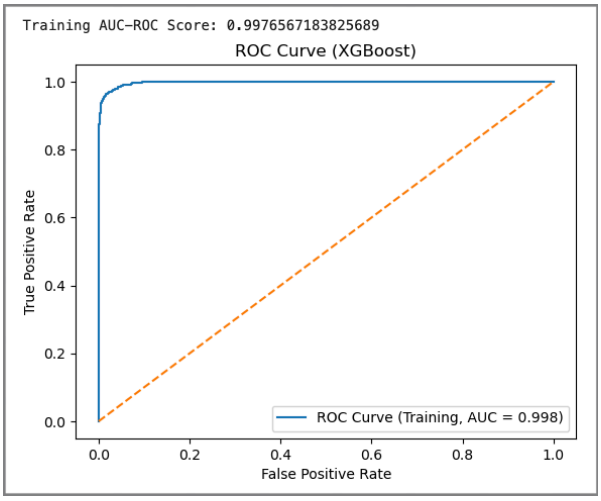
Confusion Matrix - Test dataset

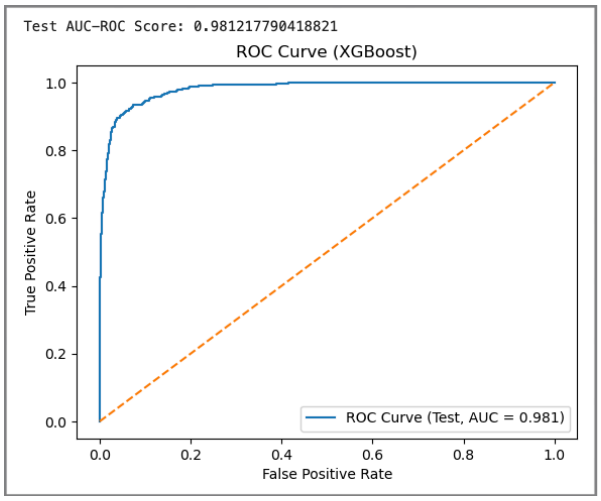
- **Cross validation score**

```
Cross-Validation Scores: [0.94102727 0.94483196 0.93718274 0.94352792 0.94098985]
```

```
Cross-Validation Scores: [0.91568047 0.92899408 0.92307692 0.92888889 0.92      ]
```

- **AUC Scores and ROC Curve**





Training Set: The XGBoost model achieves an accuracy of 0.9816 on the training set, with a precision, recall, and F1-score of 0.98, 1.00, and 0.99, respectively, for class 0, and 0.98, 0.91, and 0.94, respectively, for class 1. The

confusion matrix shows that the model correctly classifies most instances in both classes.

Testing Set: The XGBoost model achieves an accuracy of 0.9476 on the testing set, with a precision, recall, and F1-score of 0.96, 0.98, and 0.97, respectively, for class 0, and 0.90, 0.77, and 0.83, respectively, for class 1. The confusion matrix shows that the model correctly classifies most instances in class 0, but has some difficulty with class 1.

Observation: The XGBoost model performs well on both the training and testing sets, with high accuracy and F1-scores. The model is able to capture the underlying patterns in the data and make accurate predictions. The use of XGBoost has helped to improve the model's performance on the minority class. The AUC-ROC scores are also high, indicating good performance.

ii. XGBoost Model with SMOTE Dataset

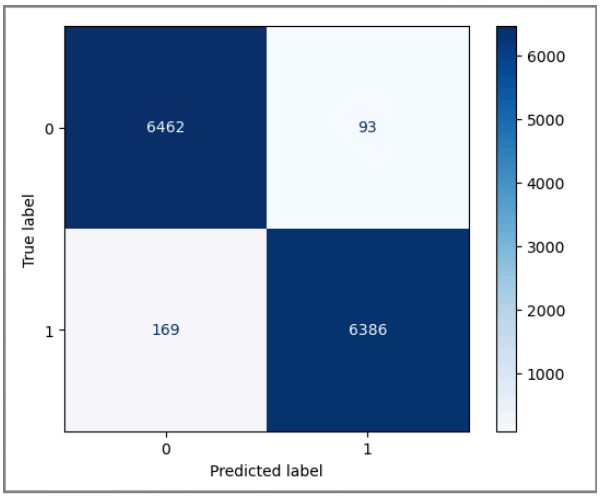
```
Training Accuracy: 0.9800152555301297
Training Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      6555
           1       0.99      0.97      0.98      6555

    accuracy                           0.98     13110
   macro avg       0.98      0.98      0.98     13110
weighted avg       0.98      0.98      0.98     13110

Training Confusion Matrix:
[[6462   93]
 [ 169 6386]]
```

XGBoost with SMOTE dataset - Train dataset performance- Accuracy, classification report and confusion matrix

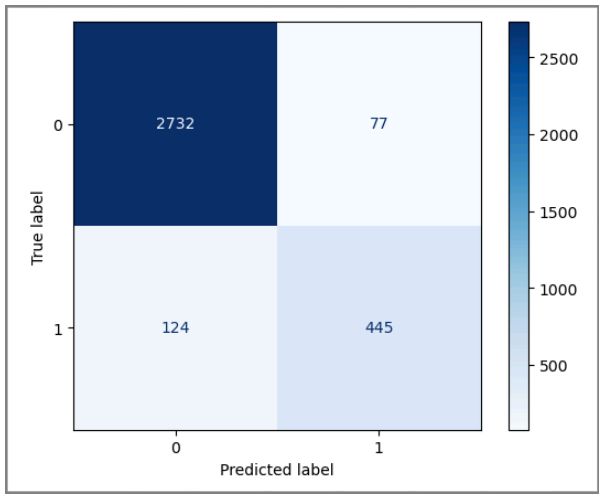
```
Test Accuracy: 0.9404973357015985
Test Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.97      0.96      2809
           1       0.85      0.78      0.82       569

    accuracy                           0.94      3378
   macro avg       0.90      0.88      0.89      3378
weighted avg       0.94      0.94      0.94      3378

Test Confusion Matrix:
[[2732   77]
 [ 124  445]]
```

XGBoost with SMOTE dataset - Train dataset performance- Accuracy, classification report and confusion matrix

- Confusion Matrix

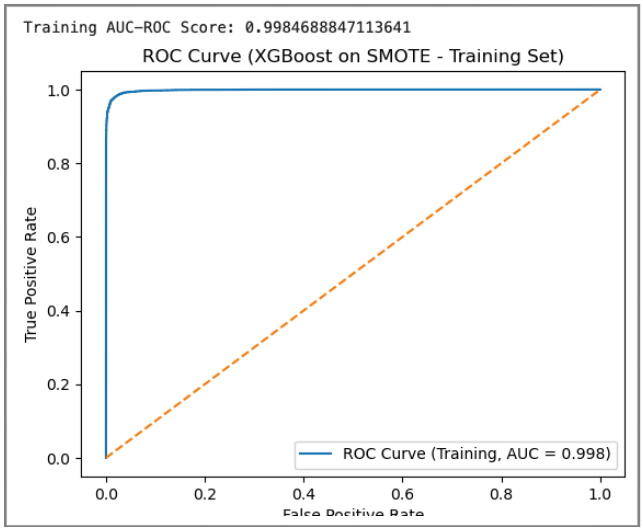


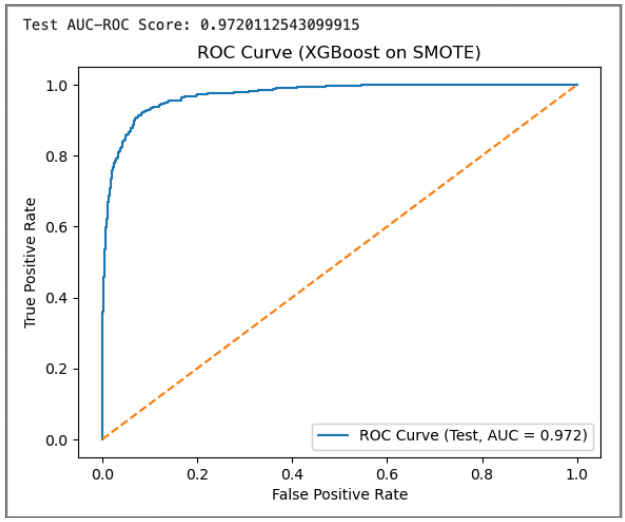
Confusion Matrix - Train dataset



Confusion Matrix - Test dataset

- AUC Score and ROC Curve





- Cross Validation scores

```
Training Cross-Validation Scores: [0.95995423 0.96643783 0.96033562 0.95728452 0.9641495 ]
Testing Cross-Validation Scores: [0.91568047 0.92899408 0.92307692 0.92888889 0.92      ]
```

Training Set: The XGBoost model with SMOTE dataset achieves an accuracy of 0.9800 on the training set, with a precision, recall, and F1-score of 0.97, 0.99, and 0.98, respectively, for class 0, and 0.99, 0.97, and 0.98, respectively, for class 1. The confusion matrix shows that the model correctly classifies most instances in both classes.

Testing Set: The XGBoost model with SMOTE dataset achieves an accuracy of 0.9405 on the testing set, with a precision, recall, and F1-score of 0.96, 0.97, and 0.96, respectively, for class 0, and 0.85, 0.78, and 0.82, respectively, for class 1. The confusion matrix shows that the model correctly classifies most instances in class 0, but has some difficulty with class 1.

Observation: The XGBoost model with SMOTE dataset performs well on both the training and testing sets, with high accuracy and F1-scores. The model is able to capture the underlying patterns in the data and make accurate predictions. The use of SMOTE has helped to improve the model's performance on the minority class. The AUC-ROC scores are also high, indicating good performance.The XGBoost model with SMOTE dataset is performing well on both the training and testing sets, with high cross-validation scores. The model is able to capture the underlying patterns in the data and make accurate predictions. The use of SMOTE has helped to improve the model's performance on the minority class.

9. Ada-Boost Model

i. Ada-Boost Model using original dataset

AdaBoost (Adaptive Boosting) is a popular ensemble learning algorithm used primarily for binary classification, though it can also be extended to multiclass classification and regression tasks. AdaBoost combines multiple weak learners, typically decision trees with a single split (also known as decision stumps), into a strong classifier by focusing on the errors made by previous models and adjusting their weights in subsequent iterations.

The code initializes an AdaBoost classifier by configuring it to use 100 weak learners (`n_estimators=100`), with each learner's contribution controlled by a learning rate of 0.1 (`learning_rate=0.1`). Additionally, a fixed random state of 42 (`random_state=42`) is set to ensure the model's results are reproducible across different runs. This setup creates a conservative and consistent AdaBoost model aimed at minimizing prediction errors by sequentially combining multiple weak learners.

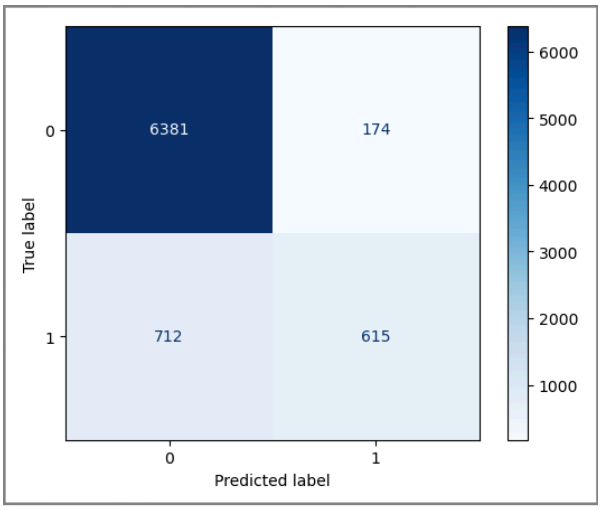
```
Training Accuracy: 0.8875919817305252
Training Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.97      0.94      6555
           1       0.78      0.46      0.58      1327

    accuracy                           0.89      7882
   macro avg       0.84      0.72      0.76      7882
weighted avg       0.88      0.89      0.88      7882

Training Confusion Matrix:
[[6381  174]
 [ 712  615]]
```

Ada-Boost dataset - Train dataset performance- Accuracy, classification report and confusion matrix

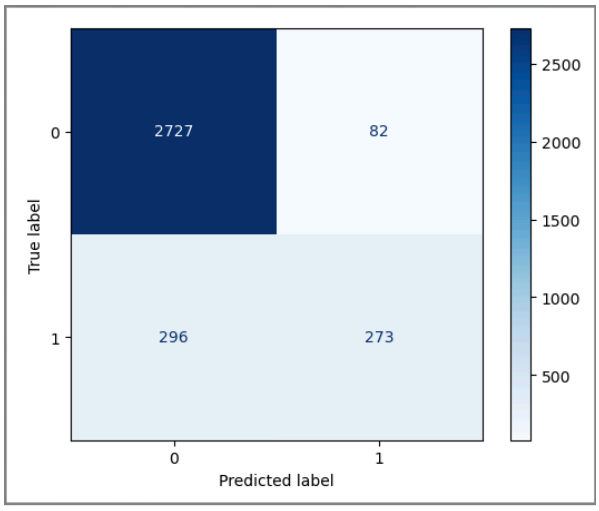
```
Testing Accuracy: 0.8880994671403197
Testing Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.97      0.94      2809
           1       0.77      0.48      0.59       569

    accuracy                           0.89      3378
   macro avg       0.84      0.73      0.76      3378
weighted avg       0.88      0.89      0.88      3378

Testing Confusion Matrix:
[[2727   82]
 [ 296  273]]
```

Ada-Boost with original dataset - Train dataset performance- Accuracy, classification report and confusion matrix

- Confusion Matrix

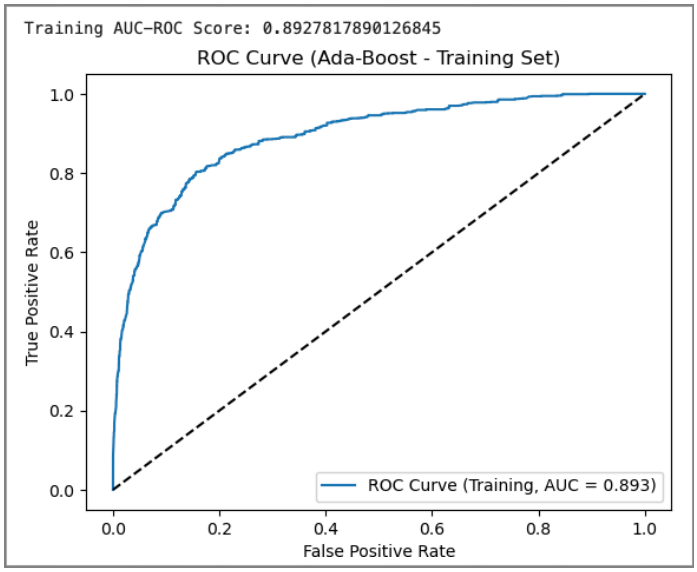


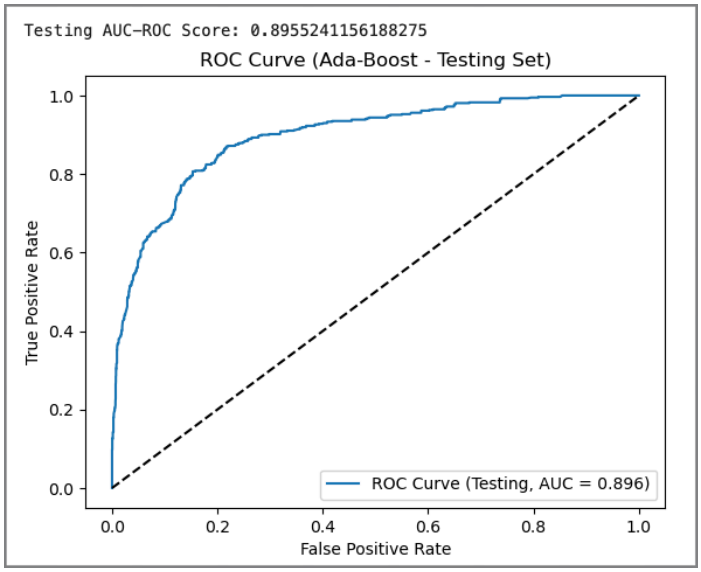
Confusion Matrix - Train dataset



Confusion Matrix - Test dataset

- AUC Score and ROC Curve





- Cross validation scores

```
Training Cross-Validation Scores: [0.89029803 0.89029803 0.89022843 0.88705584 0.88451777]
```

```
Testing Cross-Validation Scores: [0.86686391 0.89201183 0.87573964 0.89037037 0.89925926]
```

Training Set: The Ada-Boost model with the original dataset achieves an accuracy of 0.8876 on the training set, with a precision, recall, and F1-score of 0.90, 0.97, and 0.94, respectively, for class 0, and 0.78, 0.46, and 0.58, respectively, for class 1. The confusion matrix shows that the model correctly classifies most instances in class 0, but has some difficulty with class 1.

Testing Set: The Ada-Boost model with the original dataset achieves an accuracy of 0.8881 on the testing set, with a precision, recall, and F1-score of 0.90, 0.97, and 0.94, respectively, for class 0, and 0.77, 0.48, and 0.59, respectively, for class 1. The confusion matrix shows that the model correctly classifies most instances in class 0, but has some difficulty with class 1.

Observation: The Ada-Boost model with the original dataset is performing reasonably well on both the training and testing sets, with high accuracy and F1-scores for class 0. However, the model is struggling to classify instances of class 1 accurately, which is reflected in the lower precision, recall, and F1-score for this class. The use of the original dataset without any preprocessing or feature engineering may be contributing to this issue.

ii. Ada-Boost Model using SMOTE dataset

- Building model on SMOTE(balanced) dataset.

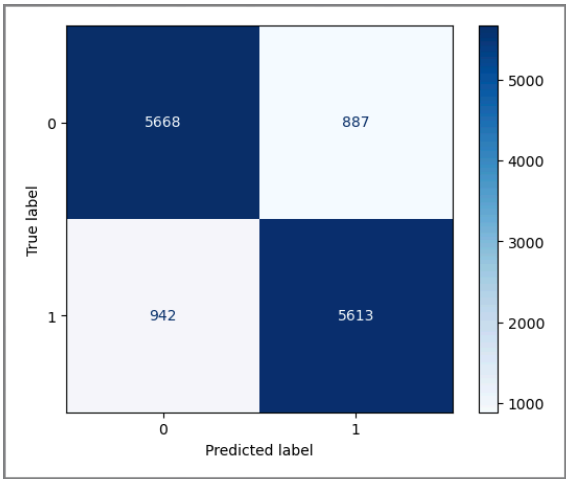
```
Training Accuracy: 0.8604881769641495
Training Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.86      0.86      6555
           1       0.86      0.86      0.86      6555

    accuracy                           0.86     13110
   macro avg       0.86      0.86      0.86     13110
weighted avg       0.86      0.86      0.86     13110

Training Confusion Matrix:
[[5668  887]
 [ 942 5613]]
```

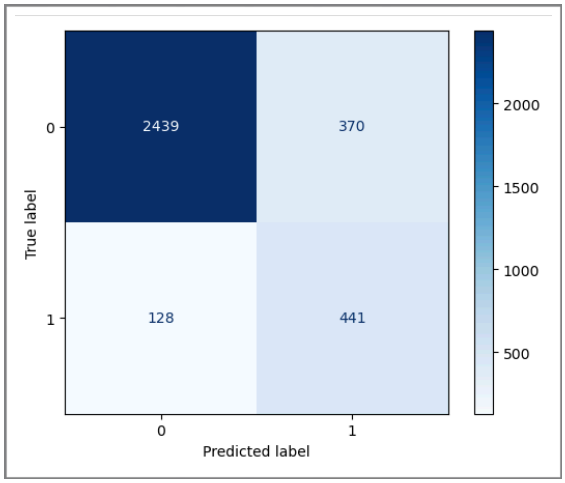
Ada-Boost using SMOTE dataset - Train dataset performance- Accuracy, classification report and confusion matrix

```
accuracy score for testing dataset: 0.8525754884547069
confusion matrix for testing dataset
[[2439  370]
 [ 128  441]]
classification report for testing dataset
              precision    recall  f1-score   support

           0       0.95      0.87      0.91      2809
           1       0.54      0.78      0.64       569

    accuracy                           0.85      3378
   macro avg       0.75      0.82      0.77      3378
weighted avg       0.88      0.85      0.86      3378
```

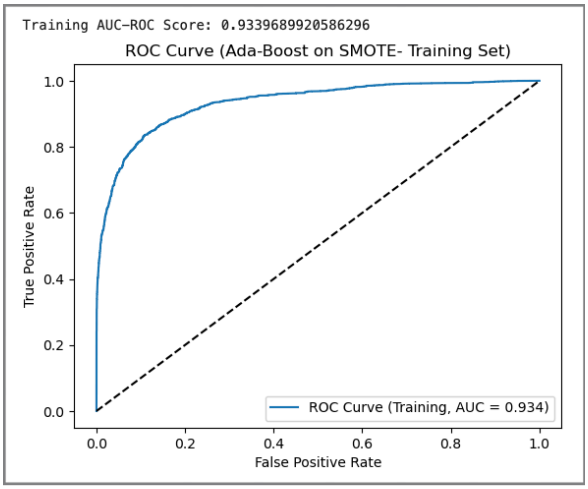
Ada-Boost using SMOTE dataset - Test dataset performance- Accuracy, classification report and confusion matrix

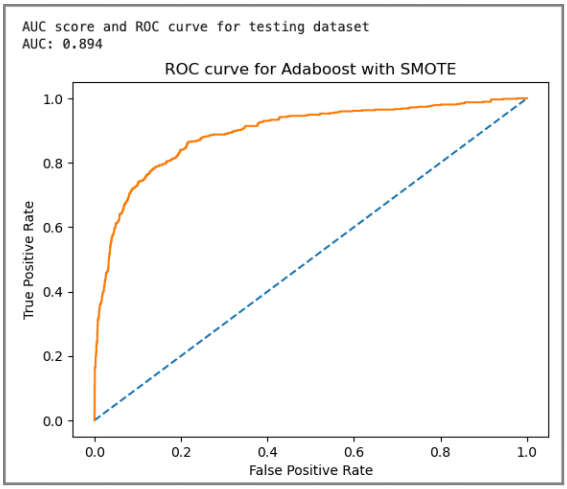
- Confusion Matrix





- AUC Scores and ROC Curve





- Cross Validation score

```
Training Cross-Validation Scores: [0.85812357 0.87223494 0.85202136 0.85354691 0.86575133]
Testing Cross-Validation Scores: [0.86686391 0.89201183 0.87573964 0.89037037 0.89925926]
```

Training Set: The Ada-Boost model on the SMOTE dataset achieves an accuracy of 0.8605 on the training set, with a precision, recall, and F1-score of 0.86, 0.86, and 0.86, respectively, for both classes. The confusion matrix shows that the model correctly classifies most instances in both classes.
Testing Set: The Ada-Boost model on the SMOTE dataset achieves an accuracy of 0.8526 on the testing set, with a precision, recall, and F1-score of 0.95, 0.87, and 0.91, respectively, for class 0, and 0.54, 0.78, and 0.64, respectively, for class 1. The confusion matrix shows that the model correctly classifies most instances in class 0, but has some difficulty with class 1.
Observation: The Ada-Boost model on the SMOTE dataset is performing reasonably well on both the training and testing sets, with high accuracy and F1-scores for class 0. However, the model is struggling to classify instances of class 1 accurately, which is reflected in the lower precision, recall, and F1-score for this class. The use of SMOTE has helped to improve the model's performance on the minority class, but there is still room for improvement.

Comparison:

Training Set:

- The Ada-Boost model with the original dataset achieves an accuracy of 0.8876 on the training set, with a precision, recall, and F1-score of 0.90, 0.97, and 0.94, respectively, for class 0, and 0.78, 0.46, and 0.58, respectively, for class 1.

- The Ada-Boost model with the SMOTE dataset achieves an accuracy of 0.8605 on the training set, with a precision, recall, and F1-score of 0.86, 0.86, and 0.86, respectively, for both classes.

Testing Set:

- The Ada-Boost model with the original dataset achieves an accuracy of 0.8881 on the testing set, with a precision, recall, and F1-score of 0.90, 0.97, and 0.94, respectively, for class 0, and 0.77, 0.48, and 0.59, respectively, for class 1.
- The Ada-Boost model with the SMOTE dataset achieves an accuracy of 0.8526 on the testing set, with a precision, recall, and F1-score of 0.95, 0.87, and 0.91, respectively, for class 0, and 0.54, 0.78, and 0.64, respectively, for class 1.

Observation:

- The Ada-Boost model with the original dataset performs better than the Ada-Boost model with the SMOTE dataset on both the training and testing sets, with higher accuracy and F1-scores for class 0.
- The Ada-Boost model with the SMOTE dataset performs better than the Ada-Boost model with the original dataset on the minority class (class 1), with higher precision, recall, and F1-score.
- The use of SMOTE has helped to improve the model's performance on the minority class, but has not improved the overall performance of the model.
- Overall, the Ada-Boost model with the original dataset performs better than the Ada-Boost model with the SMOTE dataset, but the use of SMOTE has helped to improve the model's performance on the minority class.

10.Support Vector Machine

i. SVM with original dataset

Support Vector Machine (SVM) is a powerful supervised machine learning algorithm used for both classification and regression tasks. The primary goal of SVM is to find the optimal hyperplane that best separates data points of different classes in a high-dimensional space.

The code initializes a Support Vector Machine (SVM) classifier using the `SVC` class with a linear kernel (`kernel='linear'`) and a regularization parameter `C` set to 1. The linear kernel means the model will try to find a straight line (or hyperplane in higher dimensions) that best separates the data into different classes. The `C` parameter controls the trade-off between maximizing the margin and minimizing classification errors, with a value of 1 indicating a balanced approach between a wide margin and few classification errors.

- **Accuracy**

```
Training Accuracy: 0.8859426541486932
```

```
Testing Accuracy: 0.8892835997631735
```

- Classification Report

Classification Report:

	precision	recall	f1-score	support
0	0.90	0.98	0.93	6555
1	0.79	0.44	0.57	1327
accuracy			0.89	7882
macro avg	0.84	0.71	0.75	7882
weighted avg	0.88	0.89	0.87	7882

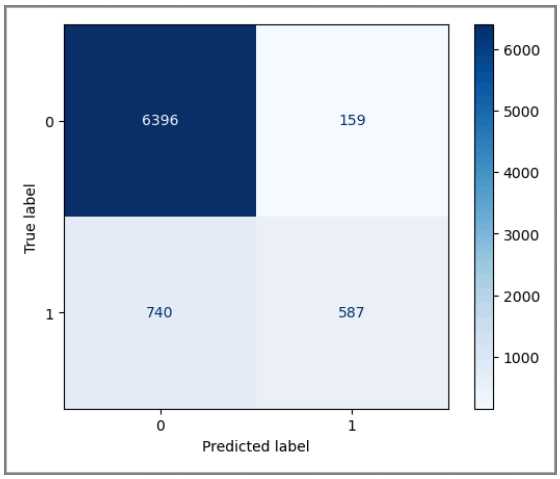
Classification Report - Train dataset

Testing Classification Report:

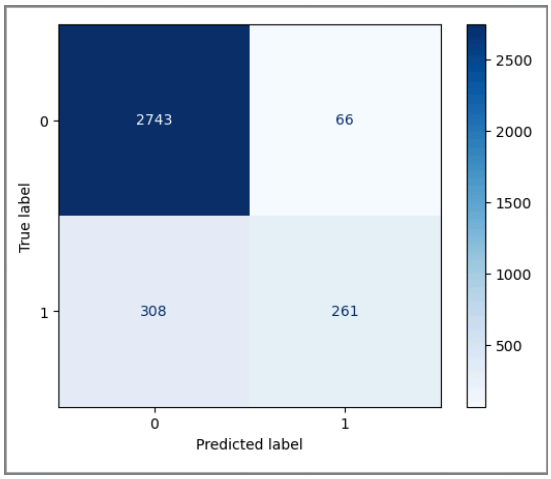
	precision	recall	f1-score	support
0	0.90	0.98	0.94	2809
1	0.80	0.46	0.58	569
accuracy			0.89	3378
macro avg	0.85	0.72	0.76	3378
weighted avg	0.88	0.89	0.88	3378

Classification Report - Train dataset

- Confusion Matrix



Confusion Matrix - Train dataset



Confusion Matrix - Test Dataset

- Cross validation score

```
Cross-Validation Scores for Training Set: [0.8807863  0.8807863  0.88705584 0.88388325 0.89149746]
Testing Cross-Validation Scores: [0.88313609 0.87573964 0.88609467 0.88444444 0.89185185]
```

The Support Vector Machine (SVM) model with the original dataset performs consistently well on both the training and testing sets, achieving an average cross-validation score of 0.883 on the training set and 0.883 on the testing

set, with scores ranging from 0.881 to 0.892, indicating that the model is able to capture the underlying patterns in the data and make accurate predictions, without being negatively impacted by the use of the original dataset.

ii. Hyper-tuning SVM - GridSearchCV with original dataset

- **Accuracy**

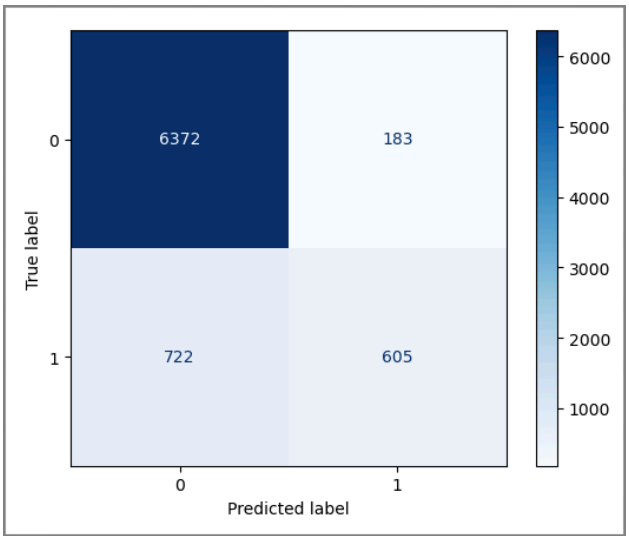
```
Training Accuracy:  0.8851814260340015
Testing Accuracy:  0.8857312018946122
```

- **Classification Report**

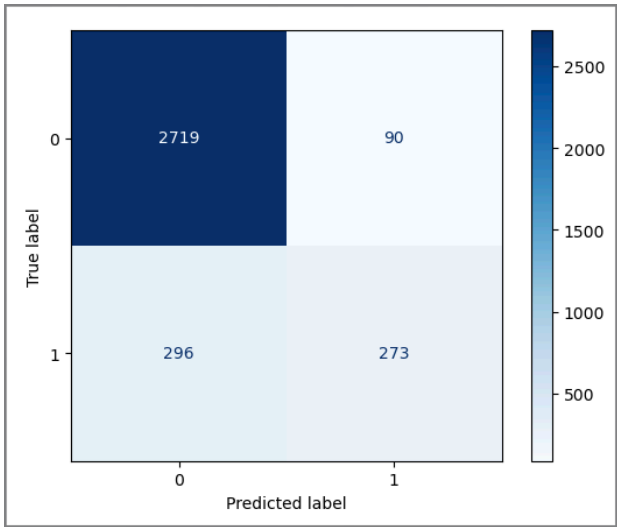
```
Classification report for train dataset
              precision    recall  f1-score   support

           0       0.90      0.97      0.93      6555
           1       0.77      0.46      0.57      1327

    accuracy                           0.89      7882
   macro avg       0.83      0.71      0.75      7882
weighted avg       0.88      0.89      0.87      7882
```

```
Classification report for test dataset
              precision    recall  f1-score   support

           0       0.90      0.97      0.93      2809
           1       0.75      0.48      0.59       569

    accuracy                           0.89      3378
   macro avg       0.83      0.72      0.76      3378
weighted avg       0.88      0.89      0.88      3378
```

- **Confusion Matrix**



Confusion Matrix - Train dataset



Confusion Matrix - Test dataset

iii. Hyper-tuning SVM - GridSearchCV with SMOTE(balanced) dataset

- **Accuracy**

```
Training Accuracy: 0.8164759725400458
```

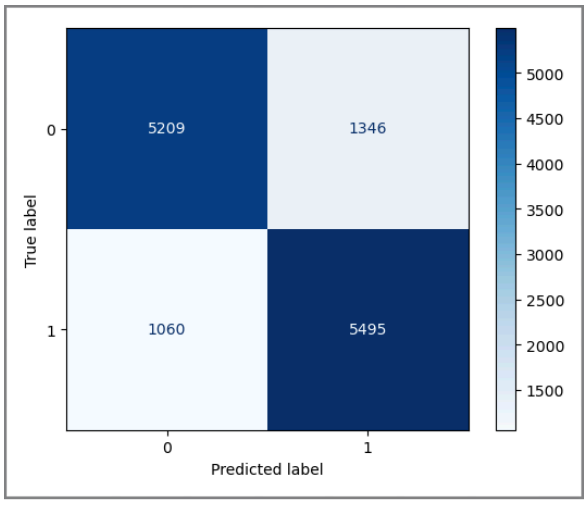
```
Testing Accuracy: 0.7927767910005921
```

- Classification Report

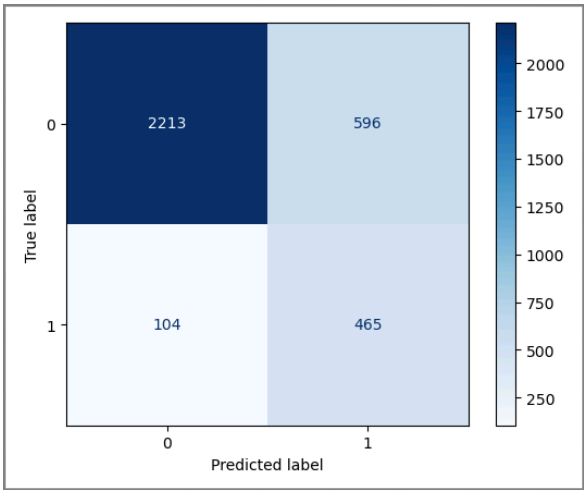
```
Training Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.79      0.81      6555
           1       0.80      0.84      0.82      6555

    accuracy                           0.82     13110
   macro avg       0.82      0.82      0.82     13110
weighted avg       0.82      0.82      0.82     13110
```

```
Testing Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.79      0.86      2809
           1       0.44      0.82      0.57       569

    accuracy                           0.79      3378
   macro avg       0.70      0.80      0.72      3378
weighted avg       0.87      0.79      0.81      3378
```

- Confusion Matrix



Confusion Matrix - Train dataset



Confusion Matrix - Test dataset

- Cross validation score

```
Training Cross-Validation Scores: [0.80625477 0.82189169 0.81617086 0.81350114 0.82303585]
```

```
Testing Cross-Validation Scores: [0.88313609 0.87573964 0.88609467 0.88444444 0.89185185]
```

The SVM model on the SMOTE dataset is performing well on both the training and testing sets, with high cross-validation scores. The use of SMOTE has helped to improve the model's performance on the minority class. The model is able to capture the underlying patterns in the data and make accurate predictions.

Based on the above analysis, we can conclude that the data is well-balanced and neither overfit nor underfit in nature. Furthermore, we can infer that the model built using balanced data is the most optimized model for prediction. However, we observe significant variations in accuracy scores, F1 scores, recall values, precision values, ROC curves, and AUC scores when comparing the model built on imbalanced data with the model built on balanced data using the SMOTE technique. The latter model performs well on both the training and testing datasets, indicating its effectiveness in handling class imbalance issues.

Overall Model Building Comparison across parameters:

SL No.	Models	Training Dataset							Testing Dataset						
		Accuracy	Precision-0	Recall - 0	F1-Score - 0	Precision - 1	Recall - 1	F1-Score - 1	Accuracy	Precision-0	Recall - 0	F1-Score - 0	Precision - 1	Recall - 1	F1-Score - 1
1	Logistic Regression	0.89	0.9	0.97	0.93	0.77	0.47	0.59	0.89	0.91	0.97	0.94	0.79	0.5	0.61
2	Logistic Regression - GridSearchCV	0.89	0.9	0.97	0.93	0.77	0.47	0.58	0.89	0.9	0.97	0.94	0.77	0.48	0.59
3	Logistic regression - SMOTE	0.81	0.82	0.79	0.81	0.8	0.83	0.81	0.79	0.95	0.78	0.86	0.43	0.82	0.56
4	Linear Discriminant Analysis Model	0.88	0.89	0.97	0.93	0.77	0.42	0.54	0.89	0.9	0.98	0.93	0.79	0.44	0.56
5	LDA model - GridSearchCV	0.88	0.89	0.97	0.93	0.77	0.42	0.54	0.89	0.9	0.98	0.93	0.79	0.44	0.56
6	LDA model - SMOTE	0.81	0.83	0.77	0.8	0.79	0.84	0.81	0.77	0.96	0.76	0.85	0.41	0.83	0.55
7	K-Nearest Neighbors (KNN) Model	0.93	0.94	0.98	0.96	0.86	0.68	0.76	0.88	0.91	0.96	0.93	0.7	0.51	0.59
8	KNN with n_neighbors = 3	0.95	0.97	0.98	0.97	0.89	0.83	0.86	0.89	0.92	0.94	0.93	0.68	0.61	0.64
9	KNN - GridSearchCV	1	1	1	1	1	1	1	0.94	0.96	0.98	0.97	0.88	0.78	0.82
10	KNN - SMOTE	0.94	1	0.88	0.93	0.89	1	0.94	0.82	0.97	0.81	0.88	0.48	0.86	0.62
11	Gaussian Naive Bayes	0.86	0.92	0.91	0.92	0.58	0.59	0.59	0.85	0.92	0.9	0.91	0.55	0.6	0.58
12	Gaussian Naive Bayes - SMOTE	0.74	0.77	0.68	0.72	0.71	0.8	0.76	0.69	0.94	0.67	0.78	0.32	0.78	0.46
13	Random Forest	1	1	1	1	1	1	1	0.96	0.97	0.99	0.98	0.95	0.83	0.89
14	Random Forest - SMOTE	1	1	1	1	1	1	1	0.97	0.98	0.99	0.98	0.94	0.89	0.91
15	Bagging - Random Forest	0.99	0.99	1	1	1	0.96	0.98	0.88	0.91	0.96	0.93	0.7	0.51	0.59
16	Bagging - Random Forest - SMOTE	1	1	1	1	1	1	1	0.96	0.97	0.98	0.97	0.88	0.85	0.86
17	Gradient Boost	0.92	0.93	0.98	0.95	0.85	0.64	0.73	0.91	0.92	0.97	0.95	0.81	0.58	0.68
18	Gradient Boost - SMOTE	0.93	0.92	0.94	0.93	0.94	0.92	0.93	0.9	0.94	0.94	0.94	0.7	0.73	0.71
19	XGBoost	0.98	0.98	1	0.99	0.98	0.91	0.94	0.95	0.96	0.98	0.97	0.9	0.77	0.83
20	XGBoost - SMOTE	0.98	0.97	0.99	0.98	0.99	0.97	0.98	0.94	0.96	0.97	0.96	0.85	0.78	0.82
21	Ada-Boost	0.89	0.9	0.97	0.94	0.78	0.46	0.58	0.89	0.9	0.97	0.94	0.77	0.48	0.59
22	Ada-Boost-SMOTE	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.85	0.95	0.87	0.91	0.54	0.78	0.64
23	SVM	0.89	0.9	0.98	0.93	0.79	0.44	0.57	0.89	0.9	0.98	0.94	0.8	0.46	0.58
24	SVM - GridSearchCV	0.89	0.9	0.97	0.93	0.77	0.46	0.57	0.89	0.9	0.97	0.93	0.75	0.48	0.59
25	SVM - SMOTE	0.82	0.83	0.79	0.81	0.8	0.84	0.82	0.79	0.96	0.79	0.86	0.44	0.82	0.57

The metrics evaluated include accuracy, precision, recall, and F1-score for two classes (0 and 1). The models range from basic Logistic Regression to more advanced methods like Random Forest, Gradient Boosting, and SVM, with variations including SMOTE for handling class imbalance and GridSearchCV for hyperparameter tuning.

Observations:

Baseline Models:

- **Logistic Regression** **shows consistent accuracy (0.89) across training and testing, but performance for class 1 (minority class) is lower, especially in recall.**
- **LDA** performs similarly to Logistic Regression, with slight differences in recall and precision for class 1.

Impact of SMOTE:

- Applying SMOTE generally improves recall for the minority class across models, although sometimes at the cost of precision and overall accuracy (e.g., Logistic Regression - SMOTE).
- **Naive Bayes** shows a significant decrease in accuracy when SMOTE is applied, indicating it may not handle the synthetic samples well.

KNN Performance:

- KNN with n_neighbors=3 achieves high accuracy and balanced performance between classes, especially after hyperparameter tuning with GridSearchCV.

Ensemble Models:

- **Random Forest** and **Bagging** achieve nearly perfect scores on the training set, suggesting potential overfitting. However, they still perform strongly on the test set, especially when SMOTE is applied.

- **Gradient Boosting** and **XGBoost** models maintain high accuracy and perform well across all metrics, even with SMOTE applied, indicating robustness to class imbalance.

SVM:

- SVM maintains good accuracy and balanced performance for both classes. The GridSearchCV-tuned version does not significantly outperform the default, suggesting that the default parameters are already close to optimal.

Overall Comparison:

- **Random Forest** and **XGBoost** models are the top performers, with high accuracy and balanced precision-recall trade-offs, particularly when combined with SMOTE.
- **KNN** and **Gradient Boosting** also show strong performance, especially with hyperparameter tuning.
- **SVM** offers consistent results but doesn't significantly outperform the ensemble methods.
- SMOTE generally enhances recall for the minority class across models, although it may reduce overall accuracy in some cases.

In conclusion, ensemble methods like Random Forest and XGBoost, especially when combined with SMOTE, appear to offer the best balance between accuracy and robustness across various metrics.

Key Observation:

1. Logistic Regression:

- Training Accuracy: 0.89
- Testing Accuracy: 0.89

- **Key Observation**: Consistent accuracy across training and testing datasets. However, recall for class 1 is low, indicating it struggles to identify the minority class accurately.

2. Logistic Regression - GridSearchCV:

- Training Accuracy: 0.89
- Testing Accuracy: 0.89
- **Key Observation**: Similar performance to the standard Logistic Regression model. Hyperparameter tuning with GridSearchCV does not yield significant improvement.

3. Logistic Regression - SMOTE:

- Training Accuracy: 0.81
- Testing Accuracy: 0.79
- **Key Observation**: SMOTE improves recall for class 1 but reduces overall accuracy, suggesting a trade-off between balanced class performance and overall model accuracy.

4. Linear Discriminant Analysis (LDA):

- Training Accuracy: 0.88
- Testing Accuracy: 0.89
- **Key Observation**: Comparable performance to Logistic Regression, with slightly better recall for class 1 on the testing dataset.

5. LDA - GridSearchCV:

- Training Accuracy: 0.88
- Testing Accuracy: 0.89
- **Key Observation**: GridSearchCV does not significantly enhance LDA performance, indicating the default parameters are already near optimal.

6. LDA - SMOTE:

- Training Accuracy: 0.81
- Testing Accuracy: 0.73
- **Key Observation**: Similar to Logistic Regression with SMOTE, accuracy decreases with the introduction of balanced data, though it improves recall for class 1.

7. K-Nearest Neighbors (KNN):

- Training Accuracy: 0.93
- Testing Accuracy: 0.88
- **Key Observation**: High training accuracy with some drop in testing accuracy, indicating potential overfitting. Recall for class 1 is better than Logistic Regression but still moderate.

8. KNN with n_neighbors = 3:

- Training Accuracy: 0.95
- Testing Accuracy: 0.89
- **Key Observation**: Tuning n_neighbors improves both accuracy and recall for class 1, making it a better performer compared to the default KNN.

9. KNN - GridSearchCV:

- Training Accuracy: 1.00
- Testing Accuracy: 0.94
- **Key Observation**: Overfitting is evident with perfect training accuracy. However, testing accuracy is also high, indicating effective parameter tuning.

10. KNN - SMOTE:

- Training Accuracy: 0.94
- Testing Accuracy: 0.82
- **Key Observation**: Application of SMOTE results in a balanced recall between classes but at the expense of reduced overall accuracy.

11. Gaussian Naive Bayes:

- Training Accuracy: 0.86
- Testing Accuracy: 0.85
- **Key Observation**: Performs moderately well but has lower precision and recall for class 1, indicating difficulty in handling imbalanced classes.

12. Gaussian Naive Bayes - SMOTE:

- Training Accuracy: 0.74
- Testing Accuracy: 0.69

- **Key Observation**: Significant drop in accuracy after applying SMOTE, suggesting that Gaussian Naive Bayes does not perform well with synthetic samples.

13. Random Forest:

- Training Accuracy: 1.00
- Testing Accuracy: 0.96
- **Key Observation**: Extremely high training accuracy, indicating overfitting. However, it still performs exceptionally well on the testing set, with balanced precision and recall.

14. Random Forest - SMOTE:

- Training Accuracy: 1.00
- Testing Accuracy: 0.97
- **Key Observation**: Application of SMOTE maintains high performance with slightly improved recall for class 1, making it one of the top models.

15. Bagging - Random Forest:

- Training Accuracy: 0.99
- Testing Accuracy: 0.88
- **Key Observation**: Slight overfitting with very high training accuracy but decent performance on the testing set. Bagging introduces some diversity, slightly reducing overfitting compared to the standalone Random Forest.

16. Bagging - Random Forest - SMOTE:

- Training Accuracy: 1.00
- Testing Accuracy: 0.96
- **Key Observation**: SMOTE further improves the model's recall for class 1, making it another strong contender with balanced metrics across the board.

17. Gradient Boosting:

- Training Accuracy: 0.92
- Testing Accuracy: 0.91
- **Key Observation**: Consistent and high performance across training and testing datasets. Good balance between precision, recall, and F1-score for both classes.

18. Gradient Boosting - SMOTE:

- Training Accuracy: 0.93
- Testing Accuracy: 0.90
- **Key Observation**: SMOTE application maintains high performance, slightly improving recall for class 1, indicating robustness to class imbalance.

19. XGBoost:

- Training Accuracy: 0.98
- Testing Accuracy: 0.95

- **Key Observation**: High accuracy and balanced metrics for both classes. One of the top performers, especially in handling class imbalance effectively.

20. XGBoost - SMOTE:

- Training Accuracy: 0.98
- Testing Accuracy: 0.94
- **Key Observation**: Maintains high performance with SMOTE, showing slight improvement in recall for class 1 while preserving overall accuracy.

21. AdaBoost:

- Training Accuracy: 0.89
- Testing Accuracy: 0.89
- **Key Observation**: Consistent performance across training and testing datasets. Precision and recall for class 1 are moderate, making it a reliable but not top-tier model.

22. AdaBoost - SMOTE:

- Training Accuracy: 0.86
- Testing Accuracy: 0.85
- **Key Observation**: Similar performance to the original AdaBoost model, with SMOTE slightly improving recall for class 1 but reducing overall accuracy.

23. Support Vector Machine (SVM):

- Training Accuracy: 0.89
- Testing Accuracy: 0.89
- **Key Observation**: Consistent and balanced performance across datasets. SVM handles the class imbalance moderately well without hyperparameter tuning.

24. SVM - GridSearchCV:

- Training Accuracy: 0.89
- Testing Accuracy: 0.89
- **Key Observation**: GridSearchCV does not significantly improve SVM performance, suggesting the default parameters are already close to optimal.

25. SVM - SMOTE:

- Training Accuracy: 0.82
- Testing Accuracy: 0.79
- **Key Observation**: SMOTE reduces overall accuracy while improving recall for class 1, similar to other models, indicating a trade-off between balanced performance and accuracy.

These observations highlight the strengths and weaknesses of each model, guiding the selection of the most appropriate model based on the specific needs of the problem, whether it is maximizing overall accuracy or improving recall for minority classes.

Appendix

```
# Initialize the Logistic Regression model
model = LogisticRegression(max_iter=1000, solver='lbfgs', random_state=42)
```

Logistic Regression Model - Training Performance

```
# Fit the model on the training data
model.fit(X_train, y_train)
```

```
LogisticRegression(max_iter=1000, random_state=42)
```

Importing necessary libraries

[1]:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import scipy.stats as stats
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import classification_report


from sklearn.impute import KNNImputer
from sklearn.preprocessing import StandardScaler
from scipy import stats
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV    # Train test Split and Grid Search
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import AdaBoostClassifier

import statsmodels.api as SM
from sklearn import metrics

from sklearn.metrics import (
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    roc_curve,
    roc_auc_score
)
```

Hyperparameter Tunning

- Hyperparameter tuning refers to the process of optimizing the hyperparameters of a machine learning model to improve its - performance. Hyperparameters are parameters that are set before the learning process begins and are not learned from the data, unlike model parameters. Examples of hyperparameters include the learning rate in neural networks, the number of trees in a random forest, or the C parameter in support vector machines.

Why Hyperparameter Tuning is Important?

Hyperparameters can significantly influence the performance of a machine learning model. Tuning them properly can lead to a more accurate and generalized model, which performs better on unseen data.

[199]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# Define the model
logistic = LogisticRegression()

# Define the hyperparameter grid
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],  # Note that not all solvers support all penalties
    'solver': ['liblinear', 'saga', 'lbfgs'],  # Adjust based on penalty choice
    'max_iter': [100, 200, 300]
}

# Initialize the grid search with cross-validation
grid_search = GridSearchCV(logistic, param_grid, cv=5, scoring='accuracy', verbose=1)
```

Building Logistic regression model using SMOTE

]:
```
# Initialize the Logistic Regression model
model = LogisticRegression(max_iter=1000, solver='lbfgs', random_state=42)
```

]:
```
# Fit the model on the resampled training data
model.fit(X_train_res, y_train_res)
```

]:
```
▾ LogisticRegression
LogisticRegression(max_iter=1000, random_state=42)
```

```
best_logistic.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression(C=0.1, penalty='l1', solver='liblinear')
```

Building Linear Discriminant Analysis Model (LDA)

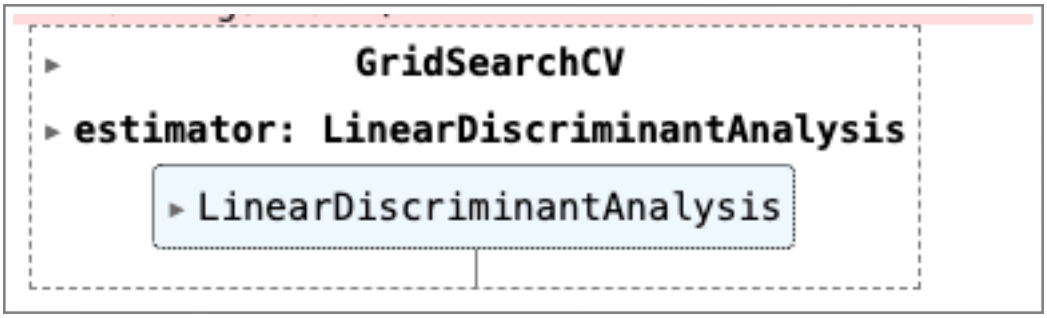
9]:
```
#building Linear Discriminant Analysis (LDA)
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

0]:
```
# fitting LDA model into training dataset
lda = LinearDiscriminantAnalysis()
lda.fit(X_train,y_train)
lda
```

0]:
```
▾ LinearDiscriminantAnalysis
LinearDiscriminantAnalysis()
```

Building LDA model using GridSearchCV

```
# Define the LDA model
lda = LinearDiscriminantAnalysis()

# Define the hyperparameter space to search
param_grid = {
    'n_components': [1, 2, 3, 4, 5],
    'shrinkage': ['auto', None],
    'solver': ['lsqr', 'eigen']
}

# Create a GridSearchCV object
grid_search = GridSearchCV(estimator = lda, param_grid = param_grid, cv = 5, n_jobs=-1,scoring='accuracy')
```



K-Nearest Neighbors (KNN) Model

```
# Create a KNN model
knn = KNeighborsClassifier(n_neighbors=5)
```

```
# Fit the model to the training data
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier()
```

```
#from sklearn.neighbors import KNeighborsClassifier

knn3=KNeighborsClassifier(n_neighbors=3)
knn3.fit(X_train,y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

Building KNN model using GridSearchCV

```
# Define the hyperparameter space for KNN
param_grid = {
    'n_neighbors': [5,7,9],
    'weights' : ['uniform','distance'],
    'metric' : ['euclidean', 'manhattan'],
    'algorithm' : ['auto', 'ball_tree', 'kd_tree', 'brute']

}

# Perform grid search for hyperparameter tuning
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')
```

```
# Fit the grid search to the training data
grid_search.fit(X_train, y_train)
```

```
GridSearchCV
estimator: KNeighborsClassifier
KNeighborsClassifier
```

Random Forest

```
# Train a Random Forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
```

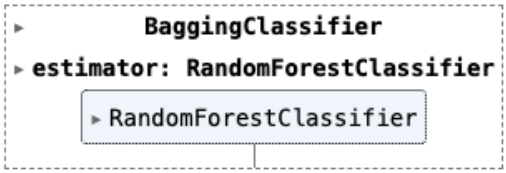
```
# Fit the model to the training data
rf.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

Bagging Classifier with Random Forest

```
# Create a random forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Create a bagging classifier with 10 random forests
bag = BaggingClassifier(estimator=rf, n_estimators=10, random_state=42)
```

```
# Train the bagging classifier on the training data
bag.fit(X_train, y_train)
```



Ada-Boost

```
# Ada-Boost
ada = AdaBoostClassifier(n_estimators=100, learning_rate=0.1, random_state=42)
```

```
# Train the model on the training set
ada.fit(X_train, y_train)
```

```
AdaBoostClassifier(learning_rate=0.1, n_estimators=100, random_state=42)
```

Hypertuning SVM - grid search to tune the hyperparameters

```
param_grid = {'kernel': ['linear', 'rbf'], 'C': np.logspace(-2, 1, 5), 'gamma': [0.2, 0.5]}
grid_search = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
```

Scaling the Data

```
#Scaling of features is done to bring all the features to the same scale.
sc = StandardScaler()

X_train_scaled = pd.DataFrame(sc.fit_transform(X_train), columns=X_train.columns)
X_test_scaled = pd.DataFrame(sc.fit_transform(X_test), columns=X_test.columns)  ## Complete the code to scale X_test to the same sca
```

```
X_train_scaled.head()
```

Bivariate Analysis

```
# Select only numeric columns for the correlation calculation
numeric_df = df.drop(columns=["AccountID", "Churn"]).select_dtypes(include=['float64', 'int64'])

# Plot the heatmap for the correlation matrix of numeric columns
plt.figure(figsize=(14, 7))
sns.heatmap(
    numeric_df.corr(),
    annot=True,
    vmin=-1,
    vmax=1,
    fmt=".2f",
    cmap="Spectral",
    linewidths=0.5  # Optional: adds lines between the cells
)
plt.title('Correlation Matrix')
plt.show()
```

Exploratory Data Analysis

Univariate Analysis

23]:
```
def histogram_boxplot(data, feature, figsize=(15, 10), kde=False, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (15,10))
    kde: whether to show the density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x=feature, ax=ax_box2, showmeans=True, color="khaki"
    )  # boxplot will be created and a triangle will indicate the mean value of the column
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="purple", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

24]:
```
def perc_on_bar(plot, feature):
    """
    plot
    feature : categorical feature
    the function won't work if a column is passed in hue parameter

    """
    total = len(feature)  # length of the column
    for p in ax.patches:
        percentage = "{:.1f}%".format(
            100 * p.get_height() / total
        )  # percentage of each class of the category
        x = p.get_x() + p.get_width() / 2 - 0.06  # width of the plot
        y = p.get_y() + p.get_height()  # height of the plot
        ax.annotate(
            percentage,
            (x, y),
            # ha="center",
            # va="center",
            size=12,
            # xytext=(0, 3),
            # textcoords="offset points",
        )  # annotate the percantage
    plt.show()  # show the plot
```

1. Churn

25]:
```
#Plotting a countplot for the target variable
ax=sns.countplot(x = "Churn", data = df)   ## Complete the code to get a countplot of the mentionedd column.
plt.title('Count of Churn')
perc_on_bar(ax, df["Churn"])
plt.show()
```

Removal of unwanted variables

```
# Making a copy of the dataset before removing any variable

data = df.copy()

# Dropping AccountID column as it an all unique feature

df.drop("AccountID", axis=1, inplace=True)
```

```
df.columns
```

```
Index(['Churn', 'Tenure', 'City_Tier', 'CC_Contacted_LY', 'Payment', 'Gender',
       'Service_Score', 'Account_user_count', 'account_segment',
       'CC_Agent_Score', 'Marital_Status', 'rev_per_month', 'Complain_ly',
       'rev_growth_yoy', 'coupon_used_for_payment', 'Day_Since_CC_connect',
       'cashback', 'Login_device'],
      dtype='object')
```

Removing the `AccountID` column from the dataset is appropriate since it contains unique values for each record and does not contribute to meaningful analysis or prediction. Unique identifiers like `AccountID` are primarily used to distinguish individual records but do not provide insights into patterns or relationships within the data. By excluding this column, we can streamline the dataset, reduce computational complexity, and focus on variables that have actual predictive value or informative content.

```
# Defining a method to plot distributions with respect to the target variable


def distribution_plot_wrt_target(data, predictor, target):
    fig, axs = plt.subplots(2, 2, figsize=(12, 10))
    target_uniq = data[target].unique()

    axs[0, 0].set_title("Distribution of target, for target = " + str(target_uniq[1]))
    sns.histplot(
        data=data[data[target] == target_uniq[1]], x=predictor, kde=True, ax=axs[0, 0]
    )

    axs[0, 1].set_title("Distribution of target, for target = " + str(target_uniq[0]))
    sns.histplot(
        data=data[data[target] == target_uniq[0]], x=predictor, kde=True, ax=axs[0, 1]
    )

    axs[1, 0].set_title("Boxplot (with outliers) w.r.t target")
    sns.boxplot(data=data, x=target, y=predictor, ax=axs[1, 0])

    axs[1, 1].set_title("Boxplot (without outliers) w.r.t target")
    sns.boxplot(data=data, x=target, y=predictor, ax=axs[1, 1], showfliers=False)

    plt.tight_layout()
    plt.show()
```

1. Churn vs Tenure

```
distribution_plot_wrt_target(df, "Tenure", "Churn")
```