To Churn or Not to Churn: Predicting Bank Customer Behavior Through Machine Learning Algorithms

By: Rhea Sethi

```
pip install mixed_naive_bayes
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg
Collecting mixed_naive_bayes
  Downloading mixed_naive_bayes-0.0.3-py3-none-any.whl (11 kB)
Requirement already satisfied: numpy>=1.22.0 in /usr/local/lib/pyt
Requirement already satisfied: scikit-learn>=0.20.2 in /usr/local/
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/pyth
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/pyt
Installing collected packages: mixed_naive_bayes
Successfully installed mixed_naive_bayes-0.0.3
```

```
#Importing necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import classification_report
import seaborn as sns
import scipy.stats as stats
from tabulate import tabulate
from sklearn import metrics
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
```

```
#Importing model packages
from mixed_naive_bayes import MixedNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
```

```
from google.colab import drive
drive.mount('/content/drive')
```

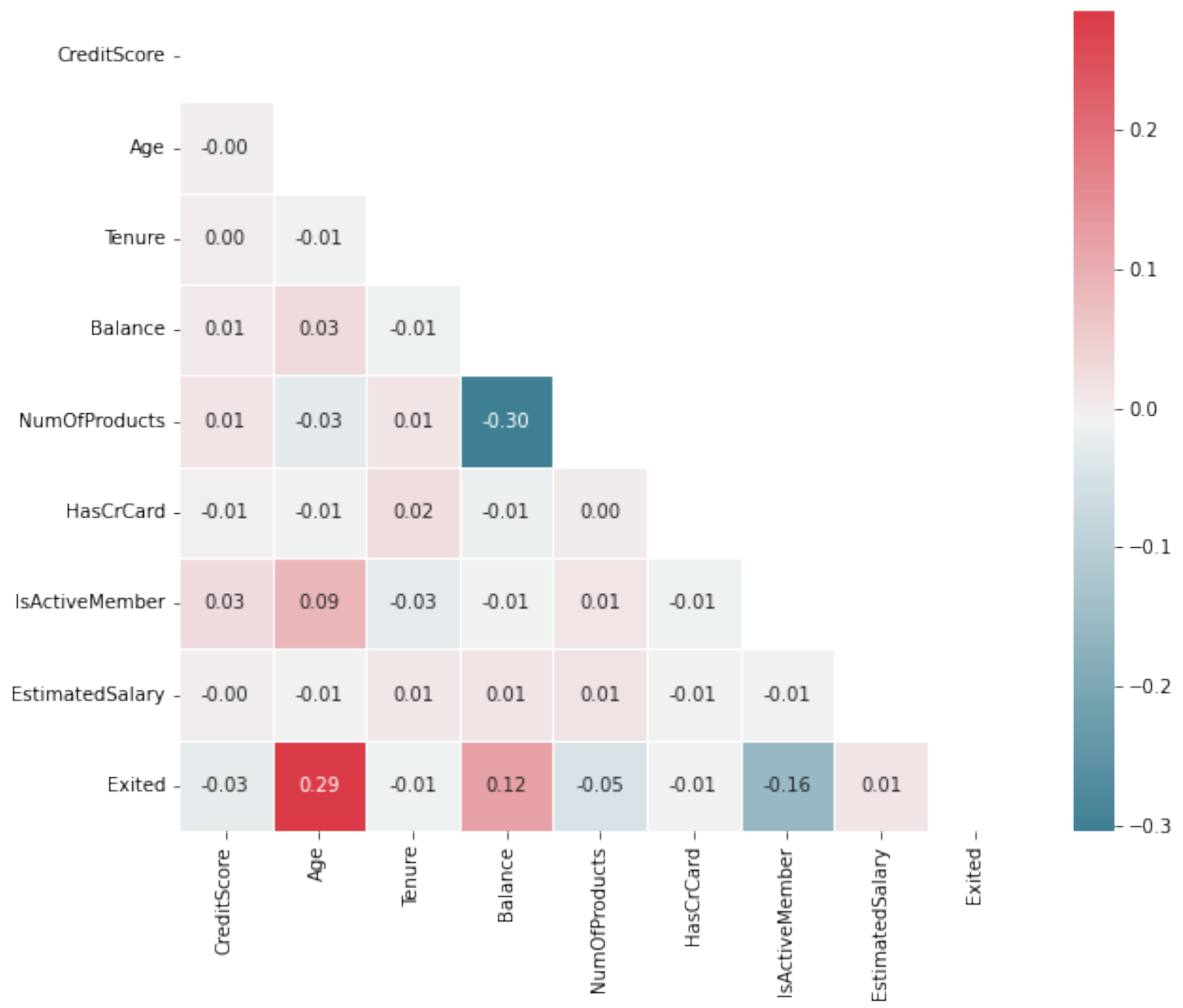
```
    Mounted at /content/drive
```

```
drive.mount('/content/gdrive/', force_remount = True)

    Mounted at /content/gdrive/
```

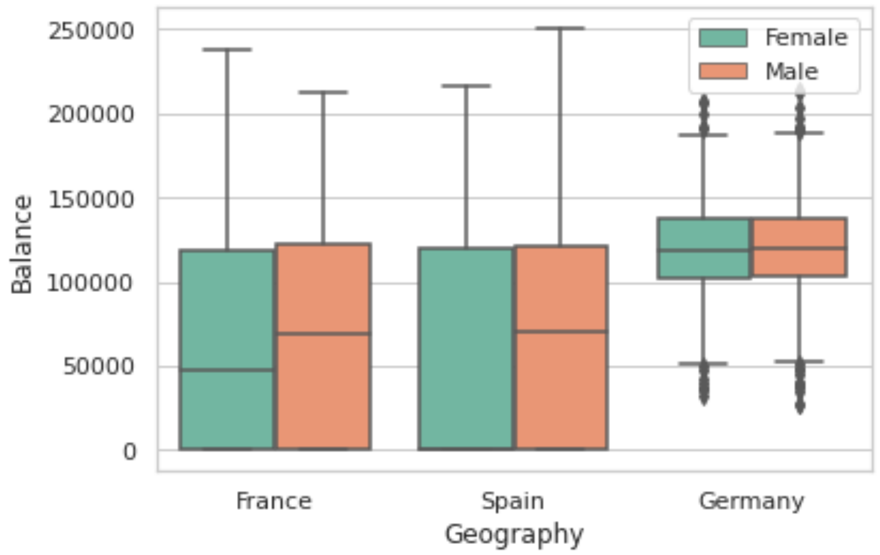
```
#Reading the CSV file
df = pd.read_csv("/content/drive/MyDrive/Churn_Modelling.csv")

# setting index
df.set_index("RowNumber", inplace=True)
```

Exploratory Data Analysis

```
# Correlation plot
sub_df =  df.drop(['CustomerId', 'Surname', 'Geography',  'Gender'], axis = 1)

corr = sub_df.corr()
fig, ax = plt.subplots(figsize=(10, 8))
colormap = sns.diverging_palette(220, 10, as_cmap = True)
dropvals = np.zeros_like(corr)
dropvals[np.triu_indices_from(dropvals)] = True
sns.heatmap(corr, cmap = colormap, linewidths = .5, annot = True, fmt = ".2f", mask = dropvals)
plt.show()
```


CreditScore
Age
Tenure
Balance
NumOfProducts
HasCrCard
IsActiveMember
EstimatedSalary
Exited
-0.00
0.00
-0.01
0.01
0.03
-0.01
0.01
-0.03
0.01
-0.30
-0.01
-0.01
0.02
-0.01
0.00
0.03
0.09
-0.03
-0.01
0.01
-0.01
-0.00
-0.01
0.01
0.01
0.01
-0.01
-0.01
-0.03
0.29
-0.01
0.12
-0.05
-0.01
-0.16
0.01
0.2
0.1
0.0
−0.1
−0.2
−0.3

```
# Set style of background of plot
sns.set(style="whitegrid")

# Select features of Geography, Balance, and hue as Gender
sns.boxplot(x="Geography", y="Balance",
                hue="Gender",
                data=df, palette="Set2",
                dodge=True)

plt.legend(loc='upper right')
plt.show()
```



```
# Check if there is null in any of the features
df.isnull().sum()

    CustomerId         0
    Surname            0
    CreditScore        0
    Geography          0
    Gender             0
    Age                0
    Tenure             0
    Balance            0
    NumOfProducts      0
    HasCrCard          0
    IsActiveMember     0
    EstimatedSalary    0
    Exited             0
    dtype: int64
```

```
# Summary statistics in excel format
df[['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'EstimatedSalary']].describe().to_excel('stat
```

```
# 1. Drop unnecessary columns, including CustomerId and Surname.
df = df.drop(['CustomerId','Surname'], axis = 1)
```

```
# 2. Get dummies for Geography, Gender.

df = pd.get_dummies(df, columns = ['Geography', 'Gender'], drop_first = True)
```

```
# Split the data into training, validation and testing sets.
X = df.drop(['Exited'], axis = 1)
y = df['Exited']

X_train_temp, X_test_unscaled, y_train, y_test = train_test_split(X, y, test_size = .5, random_state = 21)
X_val_temp, X_test_unscaled, y_val, y_test = train_test_split(X_test_unscaled, y_test, test_size = .5, random_
```

```
# 3. Scaling by MinMaxScaler (keeping df for descriptive analysis)
scaler = preprocessing.MinMaxScaler()
X_train = scaler.fit_transform(X_train_temp)
X_val = scaler.transform(X_val_temp)
X_test = scaler.transform(X_test_unscaled)
```

SVM

We perform a grid search in order to optimize the hyperparameters for SVM.

```
# defining parameter range
param_grid = {'C': [2,4,6,10,100,1000],
              'gamma': [0.01, 0.1, 1, 10],
              'kernel': ['rbf']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)

# fitting the model for grid search
grid.fit(X_train, y_train)

# print best parameter after tuning
```

```
print(grid.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid.best_estimator_)
```

```
    [CV 3/5] END ......C=10, gamma=0.01, kernel=rbf;, score=[illegible] tota
    [CV 4/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.795 tota
    [CV 5/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.795 tota
    [CV 1/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.842 tota
    [CV 2/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.840 tota
    [CV 3/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.841 tota
    [CV 4/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.839 tota
    [CV 5/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.838 tota
    [CV 1/5] END .........C=10, gamma=1, kernel=rbf;, score=0.844 tota
    [CV 2/5] END .........C=10, gamma=1, kernel=rbf;, score=0.856 tota
    [CV 3/5] END .........C=10, gamma=1, kernel=rbf;, score=0.835 tota
    [CV 4/5] END .........C=10, gamma=1, kernel=rbf;, score=0.870 tota
    [CV 5/5] END .........C=10, gamma=1, kernel=rbf;, score=0.852 tota
    [CV 1/5] END ........C=10, gamma=10, kernel=rbf;, score=0.787 tota
    [CV 2/5] END ........C=10, gamma=10, kernel=rbf;, score=0.782 tota
    [CV 3/5] END ........C=10, gamma=10, kernel=rbf;, score=0.771 tota
    [CV 4/5] END ........C=10, gamma=10, kernel=rbf;, score=0.796 tota
    [CV 5/5] END ........C=10, gamma=10, kernel=rbf;, score=0.776 tota
    [CV 1/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.798 tota
    [CV 2/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.796 tota
    [CV 3/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.796 tota
    [CV 4/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.795 tota
    [CV 5/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.795 tota
    [CV 1/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.861 tota
    [CV 2/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.857 tota
    [CV 3/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.858 tota
    [CV 4/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.857 tota
    [CV 5/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.857 tota
    [CV 1/5] END ........C=100, gamma=1, kernel=rbf;, score=0.816 tota
    [CV 2/5] END ........C=100, gamma=1, kernel=rbf;, score=0.811 tota
    [CV 3/5] END ........C=100, gamma=1, kernel=rbf;, score=0.829 tota
    [CV 4/5] END ........C=100, gamma=1, kernel=rbf;, score=0.844 tota
    [CV 5/5] END ........C=100, gamma=1, kernel=rbf;, score=0.812 tota
```

```
[CV 1/5] END .......C=100, gamma=10, kernel=rbf;, score=0.777 tota
[CV 2/5] END .......C=100, gamma=10, kernel=rbf;, score=0.778 tota
[CV 3/5] END .......C=100, gamma=10, kernel=rbf;, score=0.760 tota
[CV 4/5] END .......C=100, gamma=10, kernel=rbf;, score=0.790 tota
[CV 5/5] END .......C=100, gamma=10, kernel=rbf;, score=0.775 tota
[CV 1/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.844 tota
[CV 2/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.845 tota
[CV 3/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.843 tota
[CV 4/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.840 tota
[CV 5/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.834 tota
[CV 1/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.852 tota
[CV 2/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.861 tota
[CV 3/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.850 tota
[CV 4/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.865 tota
[CV 5/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.859 tota
[CV 1/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.781 tota
[CV 2/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.773 tota
[CV 3/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.794 tota
[CV 4/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.789 tota
[CV 5/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.782 tota
[CV 1/5] END ......C=1000, gamma=10, kernel=rbf;, score=0.777 tota
[CV 2/5] END ......C=1000, gamma=10, kernel=rbf;, score=0.778 tota
[CV 3/5] END ......C=1000, gamma=10, kernel=rbf;, score=0.757 tota
[CV 4/5] END ......C=1000, gamma=10, kernel=rbf;, score=0.792 tota
[CV 5/5] END ......C=1000, gamma=10, kernel=rbf;, score=0.775 tota
{'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
SVC(C=100, gamma=0.1)
```

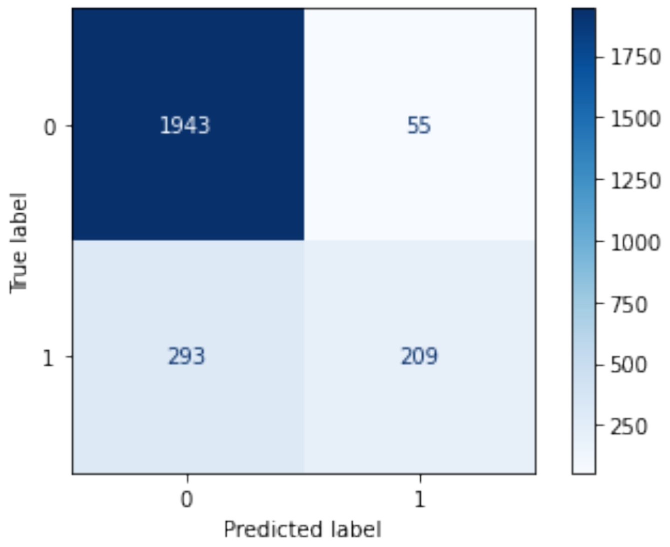
From the grid search, SVC with hyperparameters of (C=100, gamma=0.1) gives us the highest score.

```
# Train the SVM model with the optimized hyperparaters
svc = SVC(kernel='rbf', C=100, gamma=0.1).fit(X_train, y_train)
```

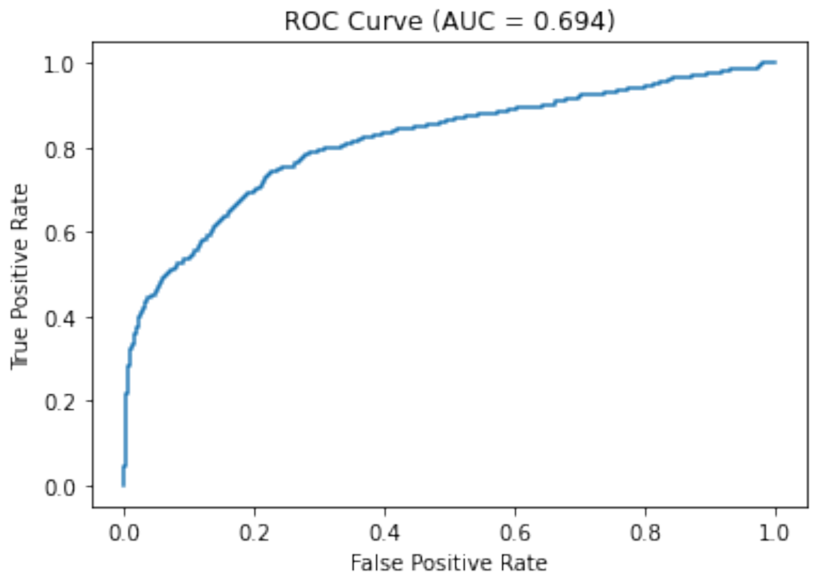
```
# Accuracy score after fitting the model with the validation set
print("SVM Accuracy on validation set: {:.3f}".format(svc.score(X_val, y_val)))
print('\n')

# AUC
y_pred_val = svc.predict(X_val)
auc= metrics.roc_auc_score(y_val, y_pred_val)
print('AUC: ', str(auc))
print('\n')

# False positive rate
TN = cm[0][0]
FN = cm[1][0]
TP = cm[1][1]
FP = cm[0][1]
FPR = FP/(FP+TN)
print('False Positive Rate: ', FPR)
print('\n')

# Classification report including precision and recall rates
print(classification_report(y_val, y_pred_val))
print('\n')

# Cofusion Matrix
cm = confusion_matrix(y_val, y_pred_val, labels=svc.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=svc.classes_)
disp.plot(cmap=plt.cm.Blues)
```

```
    SVM Accuracy on validation set: 0.861


    AUC:  0.6944035669135271
```

```
False Positive Rate:  0.027527527527527528

              precision    recall  f1-score   support

           0       0.87      0.97      0.92      1998
           1       0.79      0.42      0.55       502

    accuracy                           0.86      2500
   macro avg       0.83      0.69      0.73      2500
weighted avg       0.85      0.86      0.84      2500


<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fbde4f74a60>
```



```
# ROC curve
svc_roc = SVC(kernel='rbf', C=100, gamma=0.1, probability = True).fit(X_train, y_train)


decision_scores = svc_roc.decision_function(X_val)
fpr, tpr, thres = roc_curve(y_val, decision_scores)
auc= metrics.roc_auc_score(y_val, y_pred_val)

plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (AUC = {:.3f})'.format(auc))
plt.show()
```

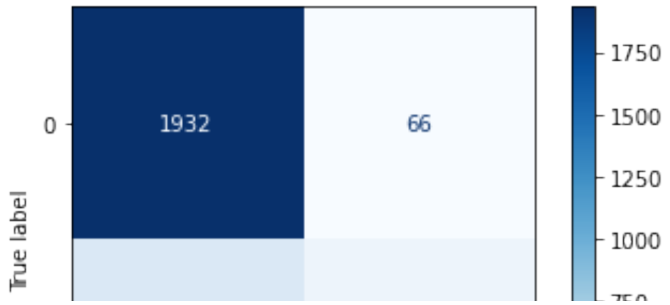


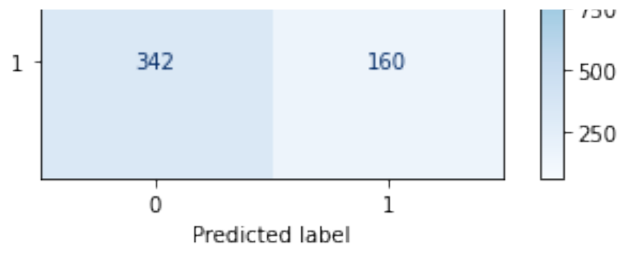
Naive Bayes

```
# Specify the categorical features
nb = MixedNB(categorical_features=[5, 6, 8, 9, 10])
nb.fit(X_train, y_train)

# Accuracy score
print("NB Accuracy on validation set: {:.3f}".format(nb.score(X_val, y_val)))
print('\n')

nb_pred_val = nb.predict(X_val)

# AUC
print('AUC: ', metrics.roc_auc_score(y_val, nb_pred_val))
print('\n')

# False positive rate
TN = cm[0][0]
FN = cm[1][0]
TP = cm[1][1]
FP = cm[0][1]
FPR = FP/(FP+TN)
print('False Positive Rate: ', FPR)
print('\n')

# Classification report including precision and recall rates
print(classification_report(y_val, nb_pred_val))
print('\n')

# Confusion matrix
```

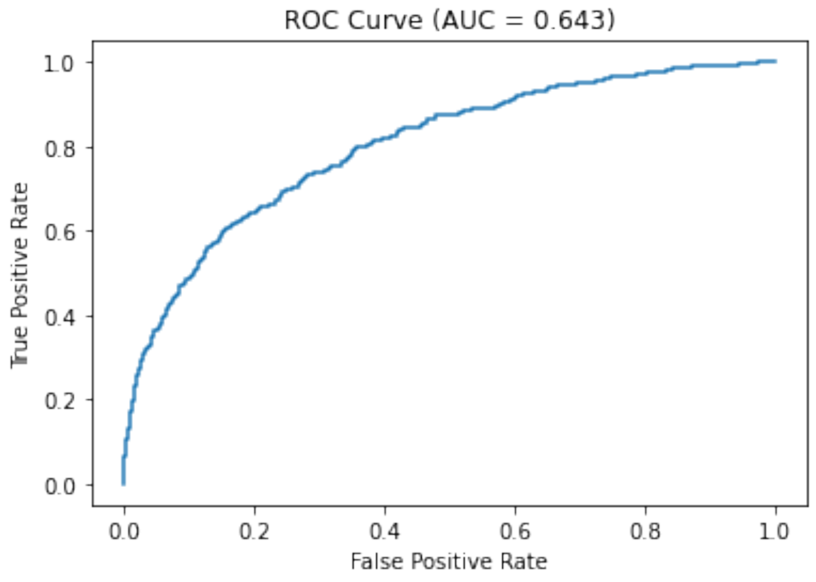
```
cm = confusion_matrix(y_val, nb_pred_val)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=svc.classes_)
disp.plot(cmap=plt.cm.Blues)
```

```
NB Accuracy on validation set: 0.837

AUC:  0.6428460332842802

False Positive Rate:  0.03303303303303303

              precision    recall  f1-score   support

           0       0.85      0.97      0.90      1998
           1       0.71      0.32      0.44       502

    accuracy                           0.84      2500
   macro avg       0.78      0.64      0.67      2500
weighted avg       0.82      0.84      0.81      2500


<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fbdddfc9f70>
```




1
342
160
750
500
250
0
1
Predicted label

```
# Plot ROC curve
y_score = nb.predict_proba(X_val)
fpr, tpr, thresholds = roc_curve(y_true=y_val, y_score=y_score[:,1])
auc= metrics.roc_auc_score(y_val, nb_pred_val)

plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (AUC = {:.3f})'.format(auc))
plt.show()
```



Random Forest Classifier

```
#First we get a benchmark model with default hyperparameter values

#Learning process
clf = RandomForestClassifier()
clf.fit(X_train, y_train)

#Making predictions over validation set
preds = clf.predict(X_val)

#Performance evaluation
print(clf.score(X_train, y_train))
print(clf.score(X_val, y_val))
```

```
    1.0
    0.8572
```

While the model performs well on the validation set, the perfect accuracy score over the training data suggests overfitting. We thus try to optimize hyperparameters.

```
#Implementing GridSearchCV for hyperparameter optimization

param_grid = {
    'n_estimators': [100, 150, 200],
    'max_depth': [None, 210, 230],
    'min_samples_split': [2, 3, 4],
    'min_samples_leaf': [1, 3, 5]
}

grid = GridSearchCV(RandomForestClassifier(), param_grid, verbose = 3)
```

```
# fitting the model for grid search
grid.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
[CV 1/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max depth=None, min samples leaf=3, min samples split=
[CV 5/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 1/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 2/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 3/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 4/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 5/5] END max_depth=None, min_samples_leaf=5, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=2, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=3, min_samples_split
[CV 4/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 1/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 2/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 3/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 4/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 5/5] END max_depth=210, min_samples_leaf=5, min_samples_split=
[CV 1/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 2/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 3/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 4/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 5/5] END max_depth=230, min_samples_leaf=1, min_samples_split=
[CV 1/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 2/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 3/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 4/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 5/5] END max_depth=230, min_samples_leaf=3, min_samples_split=
[CV 1/5] END max_depth=230, min_samples_leaf=5, min_samples_split=
[CV 2/5] END max_depth=230, min_samples_leaf=5, min_samples_split=
[CV 3/5] END max_depth=230, min_samples_leaf=5, min_samples_split=
[CV 4/5] END max_depth=230, min_samples_leaf=5, min_samples_split=
[CV 5/5] END max_depth=230, min_samples_leaf=5, min_samples_split=
[CV 1/5] END max_depth=230, min_samples_leaf=5, min_samples_split=
[CV 2/5] END max_depth=230, min_samples_leaf=5, min_samples_split=2
[CV 3/5] END max_depth=230, min_samples_leaf=5, min_samples_split=2
[CV 4/5] END max_depth=230, min_samples_leaf=5, min_samples_split=2
[CV 5/5] END max_depth=230, min_samples_leaf=5, min_samples_split=2
[CV 1/5] END max_depth=230, min_samples_leaf=5, min_samples_split=2
[CV 2/5] END max_depth=230, min_samples_leaf=5, min_samples_split=2
[CV 3/5] END max_depth=230, min_samples_leaf=5, min_samples_split=2
[CV 4/5] END max_depth=230, min_samples_leaf=5, min_samples_split=2
[CV 5/5] END max_depth=230, min_samples_leaf=5, min_samples_split=2
[CV 1/5] END max_depth=230, min_samples_leaf=5, min_samples_split=3
[CV 2/5] END max_depth=230, min_samples_leaf=5, min_samples_split=3
[CV 3/5] END max_depth=230, min_samples_leaf=5, min_samples_split=3
[CV 4/5] END max_depth=230, min_samples_leaf=5, min_samples_split=3
[CV 5/5] END max_depth=230, min_samples_leaf=5, min_samples_split=3
[CV 1/5] END max_depth=230, min_samples_leaf=5, min_samples_split=3
[CV 2/5] END max_depth=230, min_samples_leaf=5, min_samples_split=3
[CV 3/5] END max_depth=230, min_samples_leaf=5, min_samples_split=3
[CV 4/5] END max_depth=230, min_samples_leaf=5, min_samples_split=3
[CV 5/5] END max_depth=230, min_samples_leaf=5, min_samples_split=3
[CV 1/5] END max_depth=230, min_samples_leaf=5, min_samples_split=3
[CV 2/5] END max_depth=230, min_samples_leaf=5, min_samples_split=3
[CV 3/5] END max_depth=230, min_samples_leaf=5, min_samples_split=3
[CV 4/5] END max_depth=230, min_samples_leaf=5, min_samples_split=3
[CV 5/5] END max_depth=230, min_samples_leaf=5, min_samples_split=3
[CV 1/5] END max_depth=230, min_samples_leaf=5, min_samples_split=4
[CV 2/5] END max_depth=230, min_samples_leaf=5, min_samples_split=4
[CV 3/5] END max_depth=230, min_samples_leaf=5, min_samples_split=4
[CV 4/5] END max_depth=230, min_samples_leaf=5, min_samples_split=4
[CV 5/5] END max_depth=230, min_samples_leaf=5, min_samples_split=4
[CV 1/5] END max_depth=230, min_samples_leaf=5, min_samples_split=4
[CV 2/5] END max_depth=230, min_samples_leaf=5, min_samples_split=4
[CV 3/5] END max_depth=230, min_samples_leaf=5, min_samples_split=4
[CV 4/5] END max_depth=230, min_samples_leaf=5, min_samples_split=4
[CV 5/5] END max_depth=230, min_samples_leaf=5, min_samples_split=4
[CV 1/5] END max_depth=230, min_samples_leaf=5, min_samples_split=4
[CV 2/5] END max_depth=230, min_samples_leaf=5, min_samples_split=4
[CV 3/5] END max_depth=230, min_samples_leaf=5, min_samples_split=4
[CV 4/5] END max_depth=230, min_samples_leaf=5, min_samples_split=4
```

```
[CV 4/5] END max_depth=230, min_samples_leaf=5, min_samples_split=
[CV 5/5] END max_depth=230, min_samples_leaf=5, min_samples_split=
```

```
▸            GridSearchCV
▸ estimator: RandomForestClassifier
      ▸ RandomForestClassifier
```

```
# print best parameters after tuning
print(grid.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid.best_estimator_)
```

```
{'max_depth': 230, 'min_samples_leaf': 1, 'min_samples_split': 3,
RandomForestClassifier(max_depth=230, min_samples_split=3)
```

```
#Refitting the model with optimal values for hyperparameters
clf_tuned = RandomForestClassifier(max_depth = 230, min_samples_leaf= 1,
                                   min_samples_split = 3, n_estimators = 100)

clf_tuned.fit(X_train, y_train)

#Making predictions over validation set
preds = clf_tuned.predict(X_val)

#Performance evaluation
print("Training Accuracy:", clf_tuned.score(X_train, y_train))
print("Validation Accuracy:", clf_tuned.score(X_val, y_val))
```

```
    Training Accuracy: 0.9984
    Validation Accuracy: 0.862
```

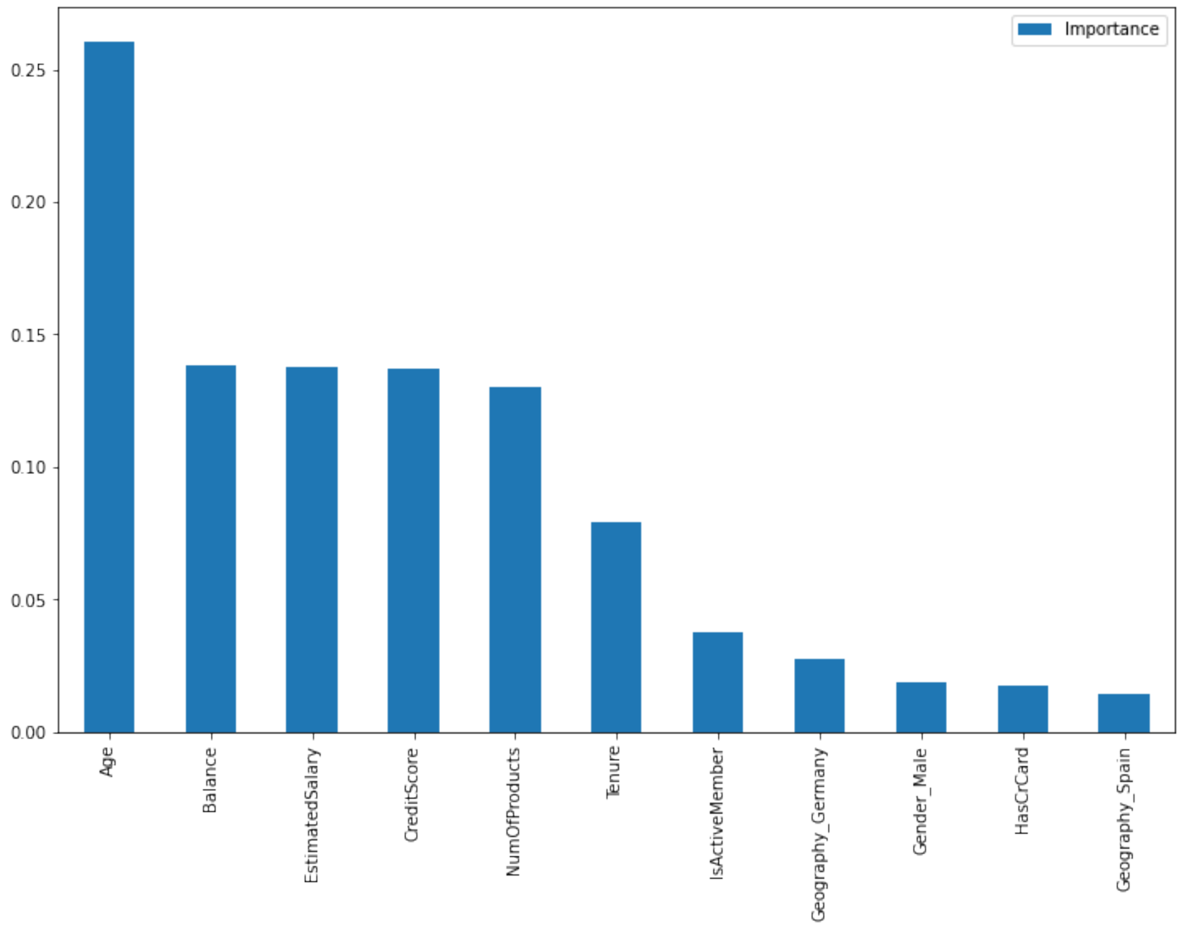
Training accuracy slightly decreases which means reduced overfitting while validation accuracy slightly increases.

```
#Checking feature importance

feature_imp = pd.DataFrame(clf_tuned.feature_importances_, index = X.columns).sort_values(by=0, ascending=Fal
feature_imp.columns = ["Importance"]
feature_imp
```

```
                   Importance
              Age    0.260543
          Balance    0.138286
  EstimatedSalary    0.137905
      CreditScore    0.136926
    NumOfProducts    0.130319
           Tenure    0.079341
   IsActiveMember    0.037759
Geography_Germany    0.027906
      Gender_Male    0.018778
        HasCrCard    0.017773
  Geography_Spain    0.014464
```

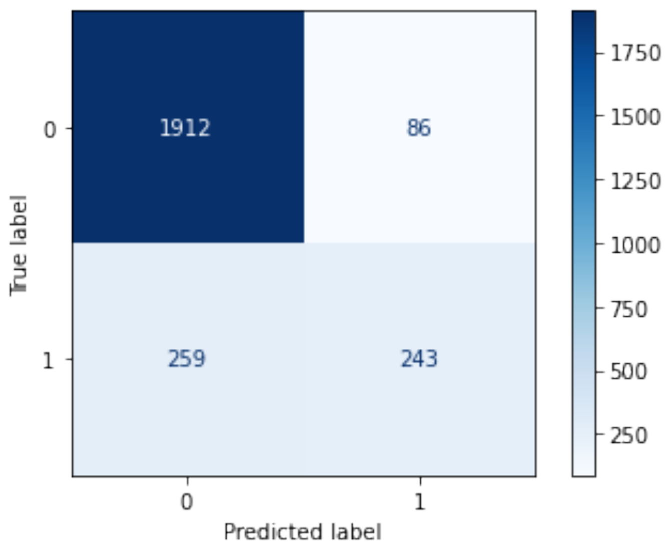
```
#Visualizing feature importance
feature_imp.plot.bar(figsize = (12,8))
```

```
<Axes: >
```


Importance
0.25
0.20
0.15
0.10
0.05
0.00
Age
Balance
EstimatedSalary
CreditScore
NumOfProducts
Tenure
IsActiveMember
Geography_Germany
Gender_Male
HasCrCard
Geography_Spain

We see that "Age" has the highest feature importance in predicting bank customer churn, followed by "Balance" and "Estimated Salary" and so on.

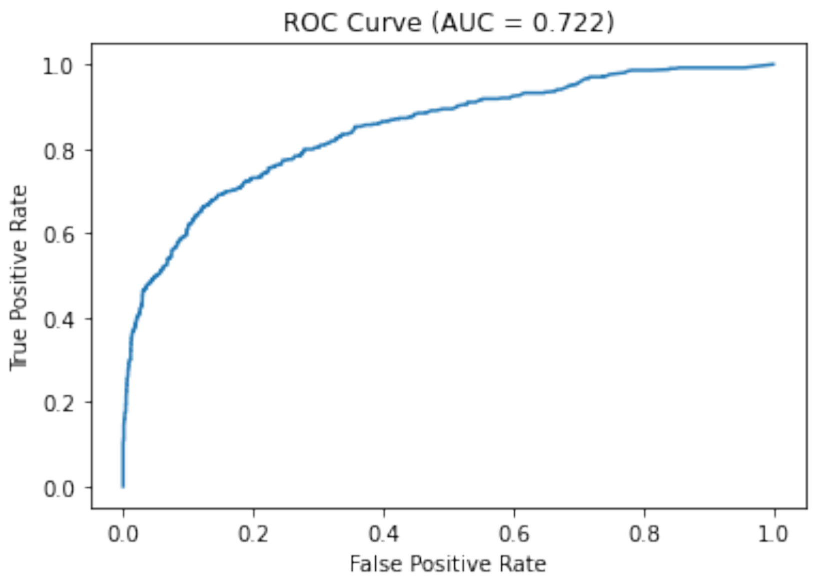
```
#More Evaluation Metrics apart from Validation Accuracy

#1. Create the confusion matrix
cm = confusion_matrix(y_val, preds)

ConfusionMatrixDisplay(confusion_matrix = cm).plot(cmap=plt.cm.Blues);
```



```
#2. Printing classification report
from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred_val))
```

```
              precision    recall  f1-score   support

           0       0.88      0.96      0.92      1998
           1       0.74      0.48      0.58       502

    accuracy                           0.86      2500
   macro avg       0.81      0.72      0.75      2500
weighted avg       0.85      0.86      0.85      2500
```

```
#3. ROC/AUC
y_score = clf_tuned.predict_proba(X_val)

fpr, tpr, thresholds = roc_curve(y_true=y_val, y_score=y_score[:,1])

auc= metrics.roc_auc_score(y_val, y_pred_val)


# Plot ROC curve
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (AUC = {:.3f})'.format(auc))
plt.show()
```



K-Nearest Neighbours

```
#Baseline model using randomly selected value of K = 3

knn = KNeighborsClassifier(n_neighbors = 3)
knn_base = knn.fit(X_train, y_train)


print("Training set score: {}".format(knn_base.score(X_train, y_train)))
print("Validation set score: {}".format(knn_base.score(X_val, y_val)))
```

```
    Training set score: 0.8906
    Validation set score: 0.8064
```

```
#Hyperparameter Optimization

####Approach 1: Tuning using GridSearchCV

k_range = list(range(1, 31))
param_grid = dict(n_neighbors = k_range)

# defining parameter range
grid = GridSearchCV(KNeighborsClassifier(), param_grid, verbose=3)

# fitting the model for grid search
grid.fit(X_train, y_train)
```

```
    Fitting 5 folds for each of 30 candidates, totalling 150 fits
    [CV 1/5] END .....................n_neighbors=1;, score=0.783 tota
    [CV 2/5] END .....................n_neighbors=1;, score=0.781 tota
```

```
[CV 3/5] END .....................n_neighbors=1;, score=0.761 tota
[CV 4/5] END .....................n_neighbors=1;, score=0.790 tota
[CV 5/5] END .....................n_neighbors=1;, score=0.793 tota
[CV 1/5] END .....................n_neighbors=2;, score=0.812 tota
[CV 2/5] END .....................n_neighbors=2;, score=0.801 tota
[CV 3/5] END .....................n_neighbors=2;, score=0.803 tota
[CV 4/5] END .....................n_neighbors=2;, score=0.807 tota
[CV 5/5] END .....................n_neighbors=2;, score=0.815 tota
[CV 1/5] END .....................n_neighbors=3;, score=0.816 tota
[CV 2/5] END .....................n_neighbors=3;, score=0.792 tota
[CV 3/5] END .....................n_neighbors=3;, score=0.778 tota
[CV 4/5] END .....................n_neighbors=3;, score=0.798 tota
[CV 5/5] END .....................n_neighbors=3;, score=0.804 tota
[CV 1/5] END .....................n_neighbors=4;, score=0.805 tota
[CV 2/5] END .....................n_neighbors=4;, score=0.803 tota
[CV 3/5] END .....................n_neighbors=4;, score=0.791 tota
[CV 4/5] END .....................n_neighbors=4;, score=0.798 tota
[CV 5/5] END .....................n_neighbors=4;, score=0.811 tota
[CV 1/5] END .....................n_neighbors=5;, score=0.809 tota
[CV 2/5] END .....................n_neighbors=5;, score=0.800 tota
[CV 3/5] END .....................n_neighbors=5;, score=0.788 tota
[CV 4/5] END .....................n_neighbors=5;, score=0.797 tota
[CV 5/5] END .....................n_neighbors=5;, score=0.811 tota
[CV 1/5] END .....................n_neighbors=6;, score=0.801 tota
[CV 2/5] END .....................n_neighbors=6;, score=0.812 tota
[CV 3/5] END .....................n_neighbors=6;, score=0.796 tota
[CV 4/5] END .....................n_neighbors=6;, score=0.800 tota
[CV 5/5] END .....................n_neighbors=6;, score=0.808 tota
[CV 1/5] END .....................n_neighbors=7;, score=0.809 tota
[CV 2/5] END .....................n_neighbors=7;, score=0.810 tota
[CV 3/5] END .....................n_neighbors=7;, score=0.794 tota
[CV 4/5] END .....................n_neighbors=7;, score=0.810 tota
[CV 5/5] END .....................n_neighbors=7;, score=0.812 tota
[CV 1/5] END .....................n_neighbors=8;, score=0.808 tota
[CV 2/5] END .....................n_neighbors=8;, score=0.816 tota
[CV 3/5] END .....................n_neighbors=8;, score=0.795 tota
[CV 4/5] END .....................n_neighbors=8;, score=0.806 tota
```

```
[CV 5/5] END .....................n_neighbors=8;, score=0.806 tota
[CV 1/5] END .....................n_neighbors=9;, score=0.803 tota
[CV 2/5] END .....................n_neighbors=9;, score=0.817 tota
[CV 3/5] END .....................n_neighbors=9;, score=0.795 tota
[CV 4/5] END .....................n_neighbors=9;, score=0.807 tota
[CV 5/5] END .....................n_neighbors=9;, score=0.815 tota
[CV 1/5] END ....................n_neighbors=10;, score=0.806 tota
[CV 2/5] END ....................n_neighbors=10;, score=0.813 tota
[CV 3/5] END ....................n_neighbors=10;, score=0.794 tota
[CV 4/5] END ....................n_neighbors=10;, score=0.809 tota
[CV 5/5] END ....................n_neighbors=10;, score=0.807 tota
[CV 1/5] END ....................n_neighbors=11;, score=0.804 tota
[CV 2/5] END ....................n_neighbors=11;, score=0.814 tota
[CV 3/5] END ....................n_neighbors=11;, score=0.797 tota
[CV 4/5] END ....................n_neighbors=11;, score=0.812 tota
[CV 5/5] END ....................n_neighbors=11;, score=0.814 tota
[CV 1/5] END ....................n_neighbors=12;, score=0.800 tota
[CV 2/5] END ....................n_neighbors=12;, score=0.810 tota
[CV 3/5] END ....................n_neighbors=12;, score=0.796 tota
[CV 4/5] END ....................n_neighbors=12;, score=0.811 tota
[CV 5/5] END ....................n_neighbors=12;, score=0.810 tota
[CV 1/5] END ....................n_neighbors=13;, score=0.804 tota
[CV 2/5] END ....................n_neighbors=13;, score=0.814 tota
[CV 3/5] END ....................n_neighbors=13;, score=0.796 tota
[CV 4/5] END ....................n_neighbors=13;, score=0.810 tota
[CV 5/5] END ....................n_neighbors=13;, score=0.811 tota
[CV 1/5] END ....................n_neighbors=14;, score=0.807 tota
[CV 2/5] END ....................n_neighbors=14;, score=0.812 tota
[CV 3/5] END ....................n_neighbors=14;, score=0.795 tota
[CV 4/5] END ....................n_neighbors=14;, score=0.810 tota
[CV 5/5] END ....................n_neighbors=14;, score=0.812 tota
[CV 1/5] END ....................n_neighbors=15;, score=0.810 tota
[CV 2/5] END ....................n_neighbors=15;, score=0.815 tota
[CV 3/5] END ....................n_neighbors=15;, score=0.796 tota
[CV 4/5] END ....................n_neighbors=15;, score=0.812 tota
[CV 5/5] END ....................n_neighbors=15;, score=0.813 tota
[CV 1/5] END ....................n_neighbors=16;, score=0.810 tota
```

```
[CV 2/5] END ....................n_neighbors=16;, score=0.806 tota
[CV 3/5] END ....................n_neighbors=16;, score=0.798 tota
[CV 4/5] END ....................n_neighbors=16;, score=0.810 tota
[CV 5/5] END ....................n_neighbors=16;, score=0.813 tota
[CV 1/5] END ....................n_neighbors=17;, score=0.810 tota
[CV 2/5] END ....................n_neighbors=17;, score=0.808 tota
[CV 3/5] END ....................n_neighbors=17;, score=0.792 tota
[CV 4/5] END ....................n_neighbors=17;, score=0.812 tota
[CV 5/5] END ....................n_neighbors=17;, score=0.814 tota
[CV 1/5] END ....................n_neighbors=18;, score=0.809 tota
[CV 2/5] END ....................n_neighbors=18;, score=0.808 tota
[CV 3/5] END ....................n_neighbors=18;, score=0.795 tota
[CV 4/5] END ....................n_neighbors=18;, score=0.807 tota
[CV 5/5] END ....................n_neighbors=18;, score=0.810 tota
[CV 1/5] END ....................n_neighbors=19;, score=0.808 tota
[CV 2/5] END ....................n_neighbors=19;, score=0.813 tota
[CV 3/5] END ....................n_neighbors=19;, score=0.791 tota
[CV 4/5] END ....................n_neighbors=19;, score=0.809 tota
[CV 5/5] END ....................n_neighbors=19;, score=0.812 tota
[CV 1/5] END ....................n_neighbors=20;, score=0.805 tota
[CV 2/5] END ....................n_neighbors=20;, score=0.808 tota
[CV 3/5] END ....................n_neighbors=20;, score=0.795 tota
[CV 4/5] END ....................n_neighbors=20;, score=0.807 tota
[CV 5/5] END ....................n_neighbors=20;, score=0.812 tota
[CV 1/5] END ....................n_neighbors=21;, score=0.803 tota
[CV 2/5] END ....................n_neighbors=21;, score=0.810 tota
[CV 3/5] END ....................n_neighbors=21;, score=0.797 tota
[CV 4/5] END ....................n_neighbors=21;, score=0.810 tota
[CV 5/5] END ....................n_neighbors=21;, score=0.812 tota
[CV 1/5] END ....................n_neighbors=22;, score=0.800 tota
[CV 2/5] END ....................n_neighbors=22;, score=0.806 tota
[CV 3/5] END ....................n_neighbors=22;, score=0.800 tota
[CV 4/5] END ....................n_neighbors=22;, score=0.806 tota
[CV 5/5] END ....................n_neighbors=22;, score=0.812 tota
[CV 1/5] END ....................n_neighbors=23;, score=0.800 tota
[CV 2/5] END ....................n_neighbors=23;, score=0.810 tota
[CV 3/5] END ....................n_neighbors=23;, score=0.799 tota
```

```
[CV 4/5] END ....................n_neighbors=23;, score=0.811 tota
[CV 5/5] END ....................n_neighbors=23;, score=0.810 tota
[CV 1/5] END ....................n_neighbors=24;, score=0.799 tota
[CV 2/5] END ....................n_neighbors=24;, score=0.807 tota
[CV 3/5] END ....................n_neighbors=24;, score=0.800 tota
[CV 4/5] END ....................n_neighbors=24;, score=0.803 tota
[CV 5/5] END ....................n_neighbors=24;, score=0.808 tota
[CV 1/5] END ....................n_neighbors=25;, score=0.798 tota
[CV 2/5] END ....................n_neighbors=25;, score=0.808 tota
[CV 3/5] END ....................n_neighbors=25;, score=0.794 tota
[CV 4/5] END ....................n_neighbors=25;, score=0.806 tota
[CV 5/5] END ....................n_neighbors=25;, score=0.806 tota
[CV 1/5] END ....................n_neighbors=26;, score=0.798 tota
[CV 2/5] END ....................n_neighbors=26;, score=0.806 tota
[CV 3/5] END ....................n_neighbors=26;, score=0.800 tota
[CV 4/5] END ....................n_neighbors=26;, score=0.799 tota
[CV 5/5] END ....................n_neighbors=26;, score=0.805 tota
[CV 1/5] END ....................n_neighbors=27;, score=0.801 tota
[CV 2/5] END ....................n_neighbors=27;, score=0.806 tota
[CV 3/5] END ....................n_neighbors=27;, score=0.799 tota
[CV 4/5] END ....................n_neighbors=27;, score=0.804 tota
[CV 5/5] END ....................n_neighbors=27;, score=0.806 tota
[CV 1/5] END ....................n_neighbors=28;, score=0.800 tota
[CV 2/5] END ....................n_neighbors=28;, score=0.805 tota
[CV 3/5] END ....................n_neighbors=28;, score=0.802 tota
[CV 4/5] END ....................n_neighbors=28;, score=0.798 tota
[CV 5/5] END ....................n_neighbors=28;, score=0.805 tota
[CV 1/5] END ....................n_neighbors=29;, score=0.802 tota
[CV 2/5] END ....................n_neighbors=29;, score=0.805 tota
[CV 3/5] END ....................n_neighbors=29;, score=0.799 tota
[CV 4/5] END ....................n_neighbors=29;, score=0.801 tota
[CV 5/5] END ....................n_neighbors=29;, score=0.808 tota
[CV 1/5] END ....................n_neighbors=30;, score=0.800 tota
[CV 2/5] END ....................n_neighbors=30;, score=0.803 tota
[CV 3/5] END ....................n_neighbors=30;, score=0.801 tota
[CV 4/5] END ....................n_neighbors=30;, score=0.795 tota
[CV 5/5] END ....................n_neighbors=30;, score=0.805 tota
```

```
▸            GridSearchCV
▸ estimator: KNeighborsClassifier
      ▸ KNeighborsClassifier
```

```
# print best parameter after tuning
print(grid.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid.best_estimator_)
```

```
{'n_neighbors': 15}
KNeighborsClassifier(n_neighbors=15)
```

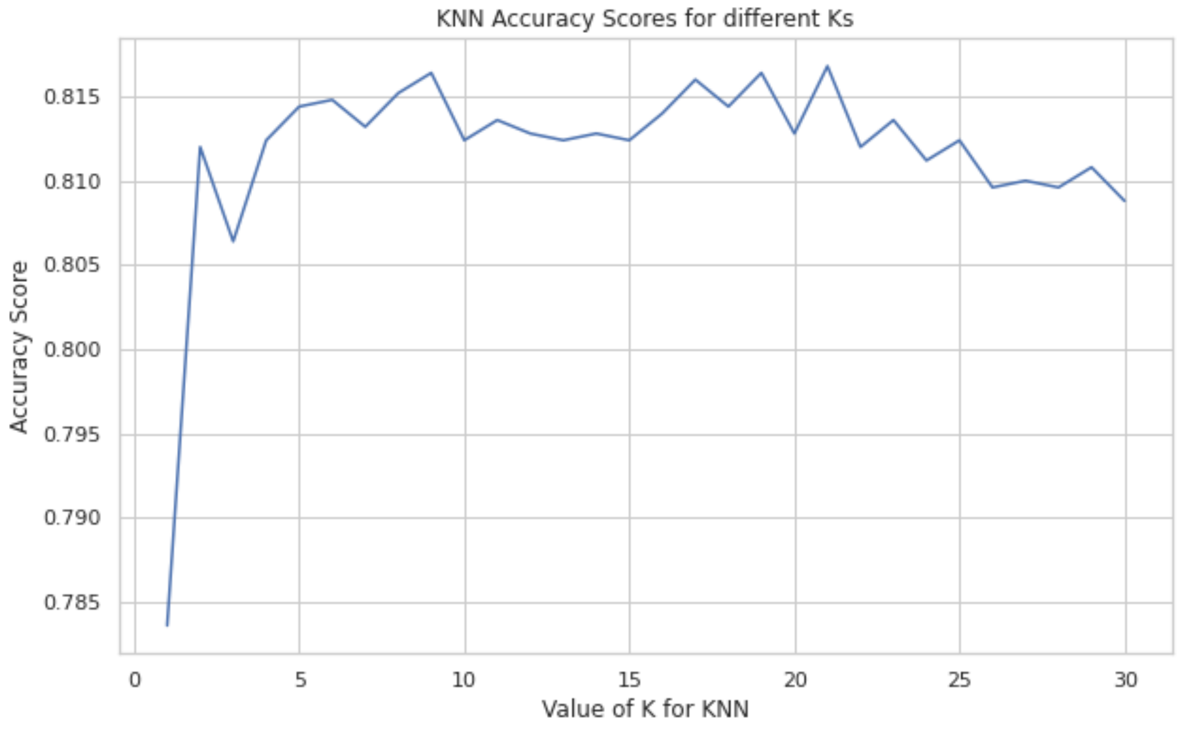
```
#Refitting the model with optimal values for hyperparameter 'K'
knn_tuned = KNeighborsClassifier(n_neighbors=15)

knn_tuned.fit(X_train, y_train)

#Performance evaluation
print("Training Accuracy:", knn_tuned.score(X_train, y_train))
print("Validation Accuracy:", knn_tuned.score(X_val, y_val))
```

```
Training Accuracy: 0.8228
Validation Accuracy: 0.8124
```

```
####Approach 2: Using Elbow Method to Optimize K

k_range = list(range(1, 31))
```

```
#Storing scores in a list
scores = []

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    scores.append(round(knn.score(X_val, y_val),4))

print(scores)

#Plotting Scores against Values of K

plt.figure(figsize= (10,6))
plt.plot(k_range, scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Accuracy Score')
plt.title("KNN Accuracy Scores for different Ks")
```

```
[0.7836, 0.812, 0.8064, 0.8124, 0.8144, 0.8148, 0.8132, 0.8152, 0.8
Text(0.5, 1.0, 'KNN Accuracy Scores for different Ks')
```



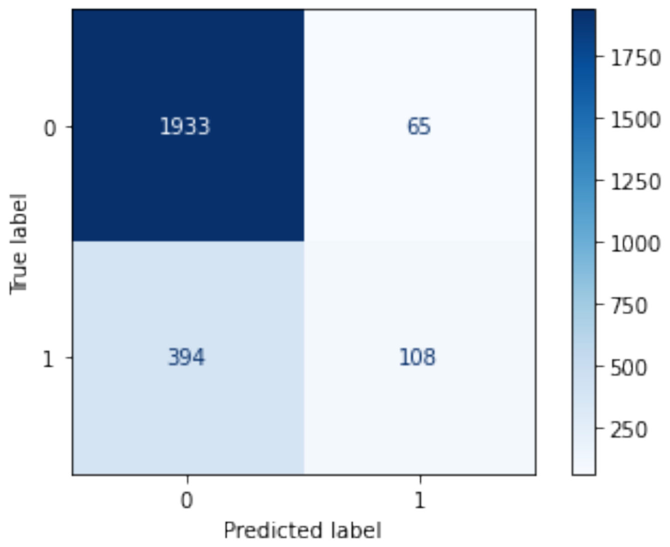
After K = 9 we don't see much improvement in accuracy. We can try this value.

```
#Refitting the model with optimal value for hyperparameter 'K': Elbow Method
knn_tuned2 = KNeighborsClassifier(n_neighbors=9)

knn_tuned2.fit(X_train, y_train)

#Performance evaluation
print("Training Accuracy:", knn_tuned2.score(X_train, y_train))
print("Validation Accuracy:", knn_tuned2.score(X_val, y_val))
```

```
    Training Accuracy: 0.8348
    Validation Accuracy: 0.8164
```

We see that this value of K yields better results than the GridSearchCV value.

```
#More Evaluation Metrics

#1. Making predictions
y_pred_val = knn_tuned2.predict(X_val)

#Create the confusion matrix
cm = confusion_matrix(y_val, y_pred_val)

ConfusionMatrixDisplay(confusion_matrix = cm).plot(cmap=plt.cm.Blues);
```



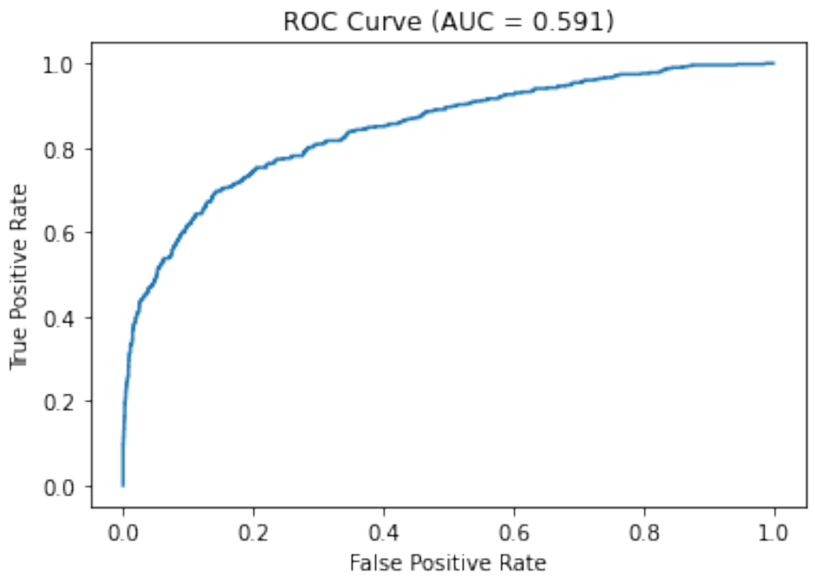
```
#2. Printing classification report
from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred_val))
```

```
              precision    recall  f1-score   support

           0       0.83      0.97      0.89      1998
           1       0.62      0.22      0.32       502

    accuracy                           0.82      2500
   macro avg       0.73      0.59      0.61      2500
weighted avg       0.79      0.82      0.78      2500
```

M MALLIKA CHANDRA
Mar 13, 2023

Just a note - a lot of functions which are there in MLP and are not there in eras. After reading online, it was safe to use sigmoid instead of logistic and kernel_regularizer=regularizers.l2(0.001) instead of alpha = 0.001

```
#3. ROC/AUC
y_score = clf_tuned.predict_proba(X_val)

fpr, tpr, thresholds = roc_curve(y_true=y_val, y_score=y_score[:,1])

auc= metrics.roc_auc_score(y_val, y_pred_val)


# Plot ROC curve
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (AUC = {:.3f})'.format(auc))
plt.show()
```



Stacking

```
pip install mlens
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg
Collecting mlens
  Downloading mlens-0.2.3-py2.py3-none-any.whl (227 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 227.7/227.7 KB 5.6 MB
Requirement already satisfied: scipy>=0.17 in /usr/local/lib/pytho
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/pytho
Installing collected packages: mlens
Successfully installed mlens-0.2.3
```

```
from mlens.ensemble import SuperLearner
```

```
#Setting up base learners
base_learners = [LogisticRegression(),
                RandomForestClassifier(),
                KNeighborsClassifier(),
                MLPClassifier(),
                SVC()]

super_learner = SuperLearner(folds = 10, random_state = 42)

super_learner.add(base_learners)
```

```
#Fit to training data
super_learner.fit(X_train, y_train)
```

```
#Get base predictions
base_predictions = super_learner.predict(X_train)

    /usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
      warnings.warn(
    /usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
      warnings.warn(
    /usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
      warnings.warn(
    /usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
      warnings.warn(
    /usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
      warnings.warn(
    /usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
      warnings.warn(
    /usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
      warnings.warn(
    /usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
      warnings.warn(
    /usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
      warnings.warn(
    /usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
      warnings.warn(
    /usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
      warnings.warn(
```

```
#Training the metalearner-- we choose logistic regression
log_reg = LogisticRegression(fit_intercept = False).fit(base_predictions, y_train)
```

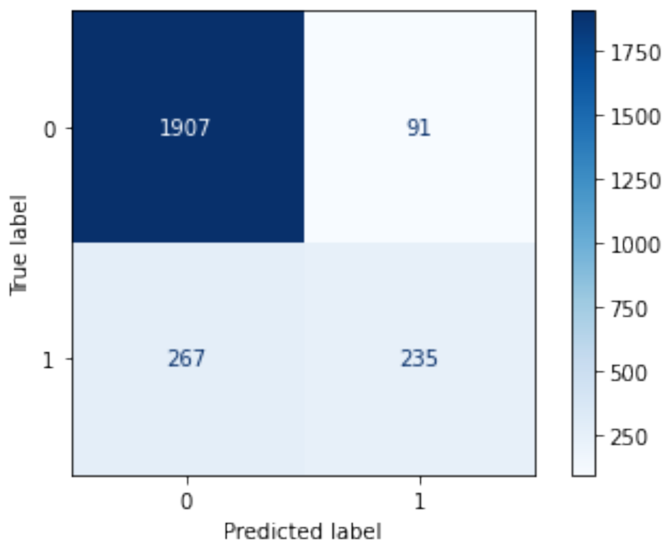
```
#Printing coefficients for each base learner
print("Coefficients:")
log_reg.coef_

    Coefficients:
    array([[-1.75376827, -2.28528552, -2.41801401,  8.5559338 ,
    1.21905046]])
```

```
#Making predictions over training set
y_pred_train = log_reg.predict(super_learner.predict(X_train))

#Making predictions over validation set
y_pred_val = log_reg.predict(super_learner.predict(X_val))
```

```
from sklearn.metrics import accuracy_score
```

```
#Evaluation Metrics

#1. Validation Accuracy
print("Training Accuracy:", accuracy_score(y_train, y_pred_train))
print("Validation Accuracy:", accuracy_score(y_val, y_pred_val))

    Training Accuracy: 0.9996
    Validation Accuracy: 0.8568
```

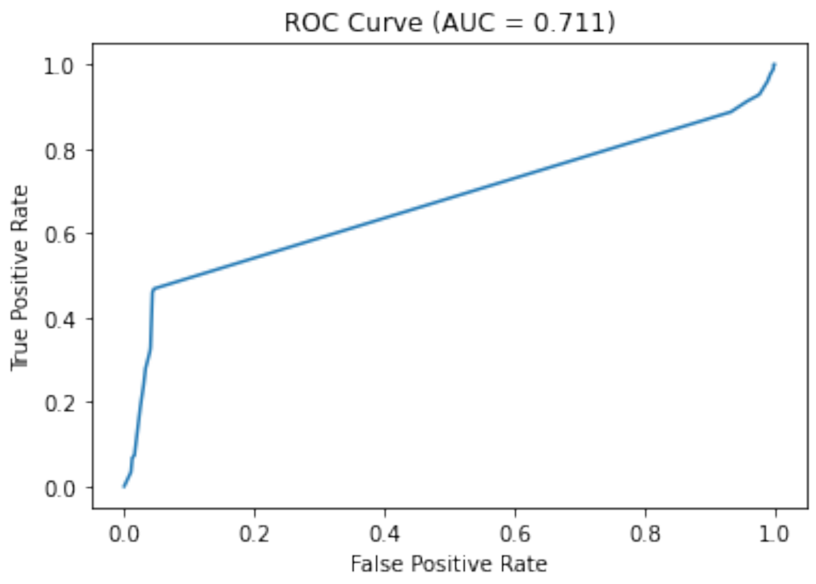
```
#2. Create the confusion matrix
cm = confusion_matrix(y_val, y_pred_val)

ConfusionMatrixDisplay(confusion_matrix = cm).plot(cmap=plt.cm.Blues);
```



```
#3. Printing classification report

print(classification_report(y_val, y_pred_val))
```

```
              precision    recall  f1-score   support

           0       0.88      0.95      0.91      1998
           1       0.72      0.47      0.57       502

    accuracy                           0.86      2500
   macro avg       0.80      0.71      0.74      2500
weighted avg       0.85      0.86      0.84      2500
```

```
#4. ROC/AUC
y_score = log_reg.predict_proba(super_learner.predict(X_val))

fpr, tpr, thresholds = roc_curve(y_true=y_val, y_score=y_score[:,1])

auc= metrics.roc_auc_score(y_val, y_pred_val)


# Plot ROC curve
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (AUC = {:.3f})'.format(auc))
plt.show()
```



Logistic Regression

Method: sklearn

```
from sklearn.linear_model import LogisticRegression
```

```
# performing grid search over potential hyperparameters
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'penalty': ['l1', 'l2', 'elasticnet', 'none'],
              'solver': ['lbfgs', 'liblinear', 'sag', 'saga']}


# Create a GridSearchCV object
grid_search = GridSearchCV(LogisticRegression(), param_grid, cv=5)

# Fit the GridSearchCV object on training data
grid_search.fit(X_train, y_train)

# Extract the best hyperparameters
best_params = grid_search.best_params_

# Create a new classifier with the best hyperparameters
clf = LogisticRegression(**best_params)
```

```
# Train the new classifier on the training data
clf.fit(X_train, y_train)

# Evaluate performance on validation set
val_score = clf.score(X_val, y_val)
print("Validation set score: {:.4f}".format(val_score))
```

```
  File "/usr/local/lib/python3.9/dist-packages/sklearn/model_selec
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_mode
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_mode
    raise ValueError(
ValueError: Solver sag supports only 'l2' or 'none' penalties, got

--------------------------------------------------------------------
30 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-packages/sklearn/model_selec
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_mode
    fold_coefs_ = Parallel(n_jobs=self.n_jobs, verbose=self.verbos
  File "/usr/local/lib/python3.9/dist-packages/sklearn/utils/paral
    return super().__call__(iterable_with_config)
  File "/usr/local/lib/python3.9/dist-packages/joblib/parallel.py"
    if self.dispatch_one_batch(iterator):
  File "/usr/local/lib/python3.9/dist-packages/joblib/parallel.py"
    self._dispatch(tasks)
  File "/usr/local/lib/python3.9/dist-packages/joblib/parallel.py"
    job = self._backend.apply_async(batch, callback=cb)
  File "/usr/local/lib/python3.9/dist-packages/joblib/_parallel_ba
    result = ImmediateResult(func)
  File "/usr/local/lib/python3.9/dist-packages/joblib/_parallel_ba
    self.results = batch()
  File "/usr/local/lib/python3.9/dist-packages/joblib/parallel.py"
    return [func(*args, **kwargs)
  File "/usr/local/lib/python3.9/dist-packages/joblib/parallel.py"
```

```
      File "/usr/local/lib/python3.9/dist-packages/joblib/parallel.py
        return [func(*args, **kwargs)
      File "/usr/local/lib/python3.9/dist-packages/sklearn/utils/paral
        return self.function(*args, **kwargs)
      File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_mode
        alpha = (1.0 / C) * (1 - l1_ratio)
    TypeError: unsupported operand type(s) for -: 'int' and 'NoneType'

    -------------------------------------------------------------------
    30 fits failed with the following error:
    Traceback (most recent call last):
      File "/usr/local/lib/python3.9/dist-packages/sklearn/model_selec
        estimator.fit(X_train, y_train, **fit_params)
      File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_mode
        solver = _check_solver(self.solver, self.penalty, self.dual)
      File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_mode
        raise ValueError("penalty='none' is not supported for the libl
    ValueError: penalty='none' is not supported for the liblinear solv

      warnings.warn(some_fits_failed_message, FitFailedWarning)
    /usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_se
        nan    nan 0.8094    nan 0.8094 0.8094    nan 0.7954    nan 0.
     0.7954 0.7954 0.7954 0.7954    nan    nan    nan    nan 0.8094
     0.8094 0.8094    nan 0.8098    nan 0.8092 0.81   0.8096 0.81   0.
        nan    nan    nan    nan 0.8094    nan 0.8094 0.8094    nan 0.
        nan 0.81   0.8092 0.8096 0.8092 0.8092    nan    nan    nan
     0.8094    nan 0.8094 0.8094    nan 0.8096    nan 0.8094 0.8098 0.
     0.8096 0.8098    nan    nan    nan    nan 0.8094    nan 0.8094 0.
        nan 0.8094    nan 0.8094 0.8094 0.8096 0.8094 0.8096    nan
        nan    nan 0.8094    nan 0.8094 0.8094]
      warnings.warn(
```

```
val_score
```

```
0.8172
```

```
best_params
```

```
    {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
```

```
# use sklearn class
clf = LogisticRegression()

# call the function fit() to train the class instance
clf.fit(X_train,y_train)

# Evaluate performance on validation set
val_score = clf.score(X_val, y_val)
print("Validation set score: {:.4f}".format(val_score))

# Evaluate performance on test set
test_score = clf.score(X_test, y_test)
print("Test set score: {:.4f}".format(test_score))
```

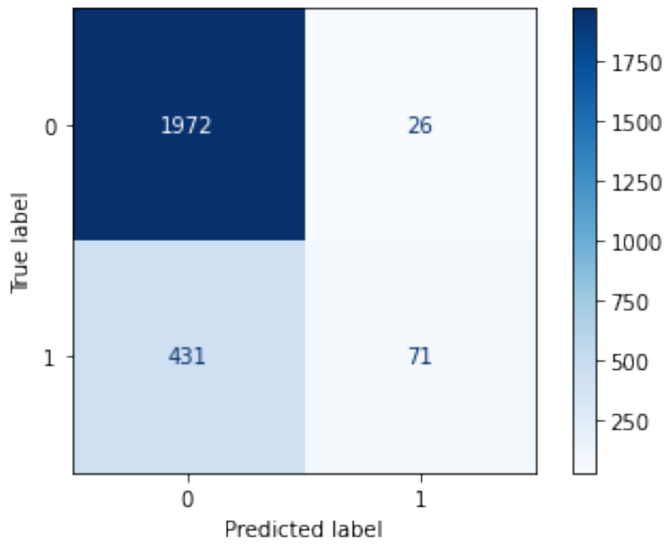
```
    Validation set score: 0.8188
    Test set score: 0.7988
```

An accuracy of 0.8188 on the validation set means that the model correctly predicted the target variable for about 81.88% of the samples in the validation set. Similarly, an accuracy of 0.7988 on the test set means that the model correctly predicted the target variable for about 79.88% of the samples in the test set.

Evaluating the Model

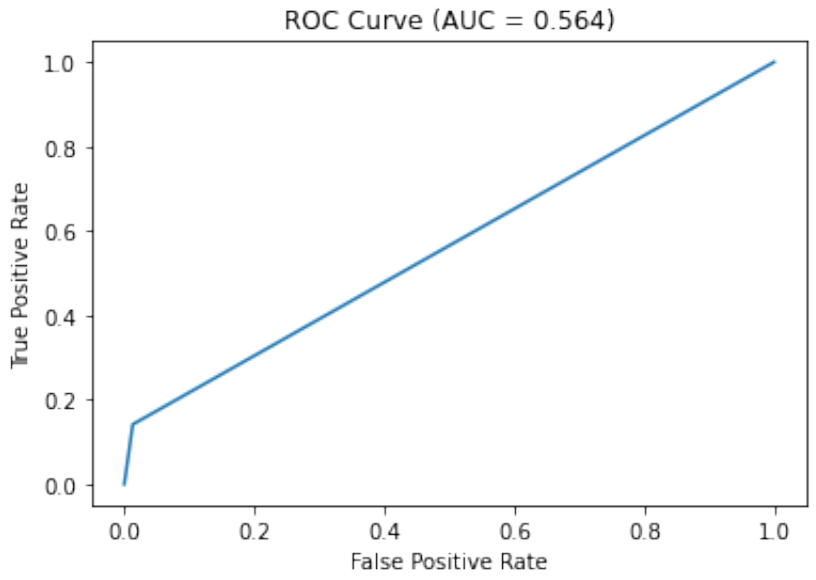
```
# fitting the model on the validation set
y_pred = clf.predict(X_val)


# evaluation metric I: confusion matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

class_labels = [0, 1]

# compute confusion matrix
cm = confusion_matrix(y_true=y_val, y_pred=y_pred, labels=class_labels)

# display confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_labels)
disp.plot(cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fb7dd22fca0>
```



```
from sklearn.metrics import precision_score, recall_score, roc_curve, roc_auc_score


# evaluation metric II and III:
# compute precision and recall rates
precision = precision_score(y_true=y_val, y_pred=y_pred)
recall = recall_score(y_true=y_val, y_pred=y_pred)

print("Precision: {:.3f}".format(precision))
print("Recall: {:.3f}".format(recall))

# evaluation metric IV
# compute ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_true=y_val, y_score=y_pred)
auc = roc_auc_score(y_true=y_val, y_score=y_pred)

# plot ROC curve
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (AUC = {:.3f})'.format(auc))
plt.show()
```

```
Precision: 0.732
Recall: 0.141
```



```
# since accuracy high, AUC low - studying the distribution of the data
import pandas as pd

# count the number of samples for each class in the training set
train_counts = pd.Series(y_train).value_counts()

# count the number of samples for each class in the validation set
val_counts = pd.Series(y_val).value_counts()

# print the counts
print("Training data:")
print(train_counts)
print("Validation data:")
print(val_counts)
```

```
Training data:
0    3977
1    1023
Name: Exited, dtype: int64
Validation data:
0    1998
1     502
Name: Exited, dtype: int64
```

Feedforward Neural Networks

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import r2_score, accuracy_score, precision_recall_fscore_support



# performing grid search for optimal hyperparameters
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV

# Define the parameter grid to search over
param_grid = {
    'hidden_layer_sizes': [(5,), (10,), (5,5), (10,10)],
    'activation': ['logistic', 'tanh', 'relu'],
    'solver': ['adam', 'sgd'],
    'alpha': [0.001, 0.01, 0.1],
    'max_iter': [100, 500]
}

# Create a MLPClassifier object
mlp = MLPClassifier()

# Create a GridSearchCV object and fit it to the training data
grid_search = GridSearchCV(mlp, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters and corresponding validation score
print("Best hyperparameters:", grid_search.best_params_)
print("Validation accuracy:", grid_search.best_score_)
```

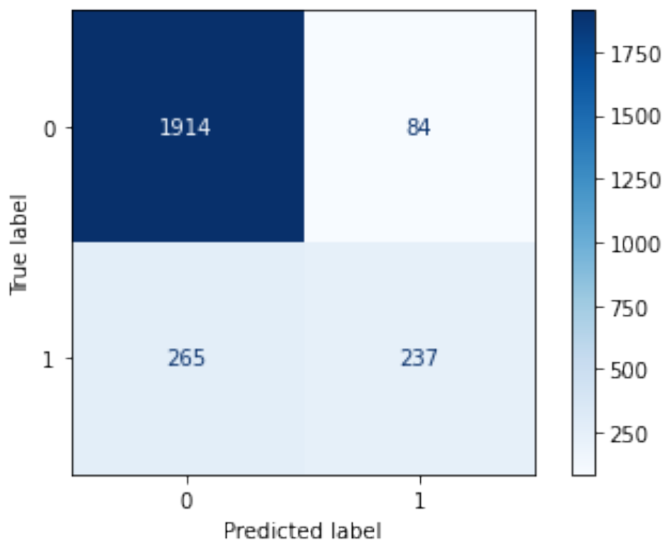
```
    /usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
      warnings.warn(
    /usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
      warnings.warn(
    /usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_mul
```

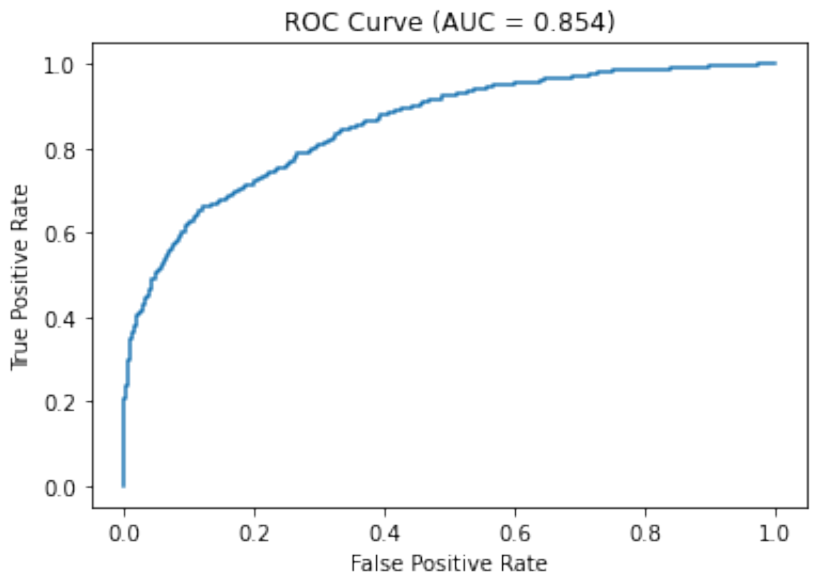
```
# evaluation metric I: Confusion Matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, precision_score, recall_score, roc_curv

# Predict labels for test set using the best model from grid search
y_pred = grid_search.predict(X_val)

# Compute confusion matrix
class_labels = [0, 1]
cm = confusion_matrix(y_true=y_val, y_pred=y_pred, labels=class_labels)

# Display confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_labels)
disp.plot(cmap=plt.cm.Blues)

# evaluation metric II and III:
# Compute accuracy, precision, and recall rates
```

```
accuracy = grid_search.score(X_val, y_val)
precision = precision_score(y_true=y_val, y_pred=y_pred)
recall = recall_score(y_true=y_val, y_pred=y_pred)

print("Accuracy: {:.3f}".format(accuracy))
print("Precision: {:.3f}".format(precision))
print("Recall: {:.3f}".format(recall))
```

```
Accuracy: 0.860
Precision: 0.738
Recall: 0.472
```



```
# evaluation metric IV:
# Compute ROC curve and AUC
y_score = grid_search.predict_proba(X_val)[:,1]
fpr, tpr, thresholds = roc_curve(y_true=y_val, y_score=y_score)
auc = roc_auc_score(y_true=y_val, y_score=y_score)

# Plot ROC curve
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (AUC = {:.3f})'.format(auc))
plt.show()
```



Building a Keras with the architecture defined by GridSearchCV

Just for my knowledge trying to visualise the architecture of NN

```
import keras.models
from keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import Dense
from keras import regularizers



# Create Keras Sequential model with specified architecture and hyperparameters
model = Sequential()
model.add(Dense(5, input_dim=X_train.shape[1], activation='sigmoid', kernel_regularizer=regularizers.l2(0.001
model.add(Dense(1, activation='linear'))

# Compile the model with specified optimizer and loss function
model.compile(optimizer='adam', loss='binary_crossentropy')

model.fit(X_train, y_train, epochs = 100)
```

```
    Epoch 1/100
    157/157 [==============================] - 1s 1ms/step - loss: 3.1
    Epoch 2/100
    157/157 [==============================] - 0s 1ms/step - loss: 3.1
    Epoch 3/100
    157/157 [==============================] - 0s 1ms/step - loss: 3.1
    Epoch 4/100
    157/157 [==============================] - 0s 1ms/step - loss: 3.1
```

```
157/157 [==============================] - 0s 1ms/step - loss: 3.1
Epoch 5/100
157/157 [==============================] - 0s 1ms/step - loss: 3.1
Epoch 6/100
157/157 [==============================] - 0s 1ms/step - loss: 3.1
Epoch 7/100
157/157 [==============================] - 0s 1ms/step - loss: 3.1
Epoch 8/100
157/157 [==============================] - 0s 1ms/step - loss: 3.1
Epoch 9/100
157/157 [==============================] - 0s 1ms/step - loss: 3.1
Epoch 10/100
157/157 [==============================] - 0s 1ms/step - loss: 3.1
Epoch 11/100
157/157 [==============================] - 0s 1ms/step - loss: 3.1
Epoch 12/100
157/157 [==============================] - 0s 1ms/step - loss: 3.1
Epoch 13/100
157/157 [==============================] - 0s 1ms/step - loss: 3.1
Epoch 14/100
157/157 [==============================] - 0s 1ms/step - loss: 3.1
Epoch 15/100
157/157 [==============================] - 0s 1ms/step - loss: 3.1
Epoch 16/100
157/157 [==============================] - 0s 1ms/step - loss: 3.1
Epoch 17/100
157/157 [==============================] - 0s 1ms/step - loss: 3.1
Epoch 18/100
157/157 [==============================] - 0s 1ms/step - loss: 3.1
Epoch 19/100
157/157 [==============================] - 0s 1ms/step - loss: 3.1
Epoch 20/100
157/157 [==============================] - 0s 1ms/step - loss: 3.1
Epoch 21/100
157/157 [==============================] - 0s 1ms/step - loss: 3.1
Epoch 22/100
157/157 [==============================] - 0s 1ms/step - loss: 3.1
Epoch 23/100
```

```
    Epoch 23/100
    157/157 [==============================] - 0s 1ms/step - loss: 3.1
    Epoch 24/100
    157/157 [==============================] - 0s 1ms/step - loss: 3.1
    Epoch 25/100
    157/157 [==============================] - 0s 1ms/step - loss: 3.1
    Epoch 26/100
    157/157 [==============================] - 0s 1ms/step - loss: 3.1
    Epoch 27/100
    157/157 [==============================] - 0s 1ms/step - loss: 3.1
    Epoch 28/100
    157/157 [==============================] - 0s 1ms/step - loss: 3.1
    Epoch 29/100
    157/157 [==============================] - 0s 1ms/step - loss: 3.1
    Epoch 30/100
    157/157 [==============================] - 0s 1ms/step - loss: 3.1
```

```
y_pred = model.predict(X_val)
```

```
    79/79 [==============================] - 1s 4ms/step
```

```
y_pred
```

```
    array([[-0.65068996],
           [-0.65068996],
           [-0.65068996],
           ...,
           [-0.65068996],
           [-0.65068996],
           [-0.65068996]], dtype=float32)
```

```
# Saving the model
keras.models.save_model(model, "/folder/model.pb")

# Loading the model
mod = keras.models.load_model("/folder/model.pb")
```

```
model.summary()
```

```
from tensorflow.keras.utils import plot_model
```

```
plot_model(model, show_shapes = True)
```

```
!pip install graphviz ann_visualizer
```

```
from ann_visualizer.visualize import ann_viz
```

```
ann_viz(model, title="CLV NN Viz", filename="model.png")
```

```
from IPython.display import Image
Image(filename = "/content/gdrive/MyDrive/ML Final Project/image.png")
```

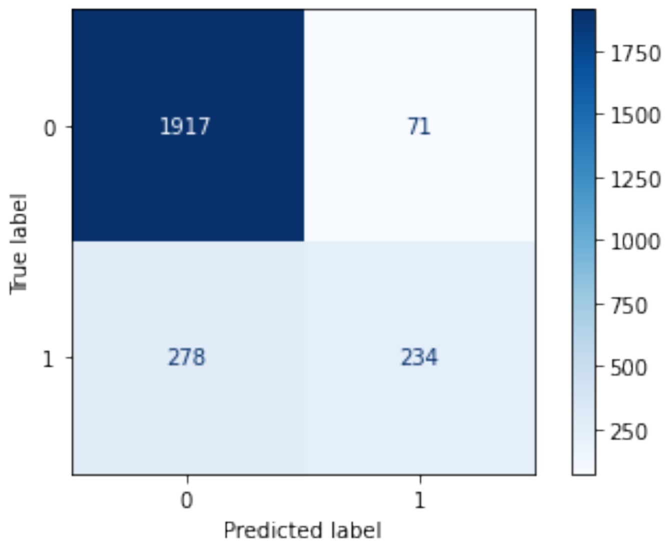
Best-Performing Model

Testing the model

```
#Random Forest Classifier on Testing Set
clf_tuned = RandomForestClassifier(max_depth = 230, min_samples_leaf= 1,
                                   min_samples_split = 4, n_estimators = 200)

clf_tuned.fit(X_train, y_train)

#Making predictions over testing set
preds = clf_tuned.predict(X_test)

#Performance evaluation
print("Training Accuracy:", clf_tuned.score(X_train, y_train))
print("Testing Accuracy:", clf_tuned.score(X_test, y_test))
```

```
Training Accuracy: 0.9918
Testing Accuracy: 0.8604
```

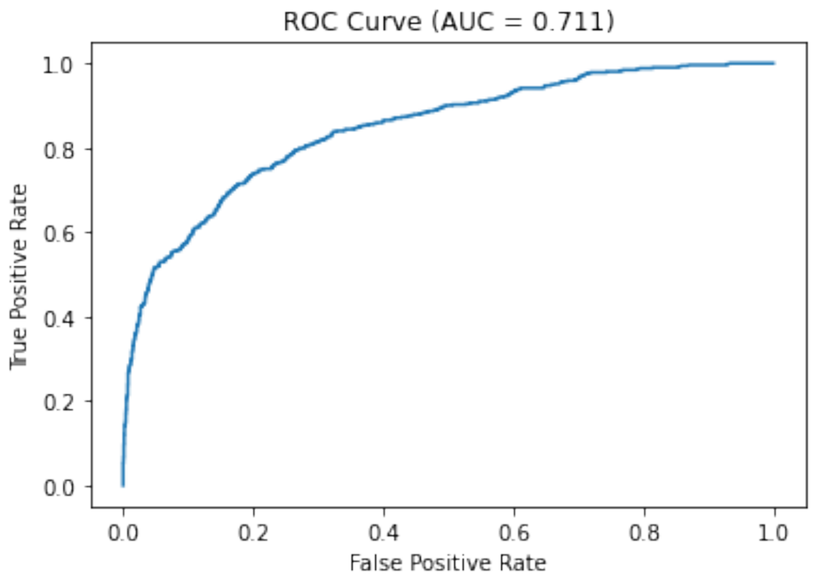
```
#Evaluation Metrics

#Create the confusion matrix
cm = confusion_matrix(y_test, preds)

ConfusionMatrixDisplay(confusion_matrix = cm).plot(cmap=plt.cm.Blues);
```



```
#Printing classification report
from sklearn.metrics import classification_report

print(classification_report(y_test, preds))
```

```
              precision    recall  f1-score   support

           0       0.87      0.96      0.92      1988
           1       0.77      0.46      0.57       512

    accuracy                           0.86      2500
   macro avg       0.82      0.71      0.74      2500
weighted avg       0.85      0.86      0.85      2500
```

```
#ROC/AUC
y_score = clf_tuned.predict_proba(X_test)

fpr, tpr, thresholds = roc_curve(y_true=y_test, y_score=y_score[:,1])

auc= metrics.roc_auc_score(y_test, preds)


# Plot ROC curve
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (AUC = {:.3f})'.format(auc))
plt.show()
```



Evaluating success and failure samples

```
y_pred_df = pd.DataFrame(preds)
Y_test_df = pd.DataFrame(y_test)

y_df = pd.merge(Y_test_df, y_pred_df, left_index = True, right_index = True)
y_df.columns = "Actual", "Predicted"
y_df
```

	Actual	Predicted
421	1	0
1022	0	0
2186	0	0
271	0	0
1121	0	0
...	...	...
1610	0	1
138	0	0
423	1	0
1148	0	0
366	0	0

620 rows × 2 columns

```
# Failure samples: samples for which our model can not correctly predict their labels

failures = y_df[y_df["Actual"] != y_df["Predicted"]]
failures.head()
```

	Actual	Predicted
421	1	0
408	0	1
1480	0	1
1949	0	1
2378	0	1

```
# Success samples: samples for which you model can correctly predict their labels

successes = y_df[y_df["Actual"] == y_df["Predicted"]]
successes.head()
```

	Actual	Predicted
1022	0	0
2186	0	0
271	0	0
1121	0	0
1649	0	0

```
from prettytable import PrettyTable


#Converting features dataframe to an array
X_test_unscaled = np.array(X_test_unscaled)
X_test_unscaled

    array([[621.,  43.,   8., ...,   0.,   1.,   0.],
           [850.,  40.,   6., ...,   0.,   0.,   1.],
           [604.,  56.,   0., ...,   0.,   0.,   0.],
           ...,
           [704.,  38.,   6., ...,   0.,   1.,   1.],
           [628.,  33.,   3., ...,   0.,   1.,   0.],
           [662.,  42.,   6., ...,   0.,   1.,   0.]])
```

```
#Features for 5 success samples
result_table_successes = PrettyTable()
result_table_successes.field_names = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard'
       'IsActiveMember', 'EstimatedSalary', 'Geography_Germany',
       'Geography_Spain', 'Gender_Male', "Actual", "Predicted"]

#Pulling out feature values for 5 successful predictions and storing into a table
for j in successes[:5].index:
    result_table_successes.add_row([X_test_unscaled[j,0], X_test_unscaled[j,1], X_test_unscaled[j,2],
                          X_test_unscaled[j,3], X_test_unscaled[j,4], X_test_unscaled[j,5],
                          X_test_unscaled[j,6], X_test_unscaled[j,7], X_test_unscaled[j,8],
                          X_test_unscaled[j,9], X_test_unscaled[j,10],
                          successes.loc[j, "Predicted"], successes.loc[j, "Actual"]])

print(result_table_successes)
```

```
    +-------------+------+--------+-----------+---------------+-------
    | CreditScore | Age  | Tenure |  Balance  | NumOfProducts | HasCrC
    +-------------+------+--------+-----------+---------------+-------
    |    604.0    | 25.0 |  5.0   | 157780.84 |      2.0      |    1.0
    |    667.0    | 42.0 |  7.0   |    0.0    |      1.0      |    0.0
    |    449.0    | 31.0 |  1.0   |  113693.0 |      1.0      |    0.0
    |    651.0    | 33.0 |  1.0   |  96834.78 |      1.0      |    1.0
    |    675.0    | 57.0 |  8.0   |    0.0    |      2.0      |    0.0
    +-------------+------+--------+-----------+---------------+-------
```

```
#Features for 5 failures

result_table_failures = PrettyTable()
result_table_failures.field_names = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Geography_Germany',
       'Geography_Spain', 'Gender_Male', "Actual", "Predicted"]

#Pulling out feature values for 5 failed predictions and storing into table
for i in failures[:5].index:
    result_table_failures.add_row([X_test_unscaled[i,0], X_test_unscaled[i,1], X_test_unscaled[i,2],
                          X_test_unscaled[i,3], X_test_unscaled[i,4], X_test_unscaled[i,5],
                          X_test_unscaled[i,6], X_test_unscaled[i,7], X_test_unscaled[i,8],
                          X_test_unscaled[i,9], X_test_unscaled[i,10],
                          failures.loc[i, "Predicted"], failures.loc[i, "Actual"]])

#result_table.add_row(["-"*10, "-"*10, "-"*10, "-"*10, "-"*10, "-"*10, "-"*10, "-"*10])
print(result_table_failures)
```

```
+-------------+------+--------+-----------+---------------+-------
| CreditScore | Age  | Tenure |  Balance  | NumOfProducts | HasCrC
+-------------+------+--------+-----------+---------------+-------
|    695.0    | 31.0 |  5.0   |    0.0    |      2.0      |    0.0
|    845.0    | 52.0 |  0.0   |    0.0    |      1.0      |    1.0
|    599.0    | 50.0 |  3.0   | 121159.65 |      1.0      |    0.0
|    434.0    | 55.0 |  8.0   | 109339.17 |      2.0      |    1.0
|    617.0    | 39.0 |  5.0   |  83348.89 |      3.0      |    1.0
+-------------+------+--------+-----------+---------------+-------
```

```
X_test_df = pd.DataFrame(X_test_unscaled)

test_df = pd.merge(X_test_df, y_df, left_index = True, right_index = True)
test_df.columns = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Geography_Germany',
       'Geography_Spain', 'Gender_Male', "Actual", "Predicted"]
test_df.groupby("Actual").mean()
```

```
        CreditScore        Age    Tenure       Balance  NumOfProducts
Actual
     0   648.453061  38.504082  5.153061  77936.988571       1.540816
     1   657.084615  38.469231  4.900000  77460.767462       1.538462
```

From the analysis above we try to dig into the reasons for the failure cases by comparing the values of their features to the mean value of the features for their actual classes. We see that the failure cases stem from the fact that these samples' features were more closely aligned with the other class' feature values than their actual class'.

For example, Class 0 has a slightly higher mean value for Age than Class 1. We see that most of the failures that were misclassified as negative when they were actually positive have higher values for "Age". On the other hand, the one sample misclassified as positive when it is actually negative has a low value for age, as do most of the samples under class 1.