AIDI 1002: Machine Learning Programming — Assignment - 3

1. Design a deep learning experiment for a multi class classification dataset https://www.kaggle.com/datasets/ abisheksudarshan/customer-segmentation. It is a multi class classification task where "var_1" is a class label column having 3 categories as Cat_6: 65%, Cat_4: 13%, Other: 22%. There is a slight imblance in class distribution. This link contains two files 'train.csv' and 'test.csv'. You need to divide the 'train.csv' in appropriate percentage to get the validation set. Your experiment should involve following step in appropriate order.

- Shuffling of the data before training (2 points)
- Design and train a neural network model (e.g. you can use DNN network or if you want to use any other models it is also acceptable) (10 points)
- Use validation data for model tuning and monitor the f1-score while applying the early stopping logic from keras library (10 points)
- Use test data to calculate the appropriate classification metrics. (5 points)
- Explain the significance of each metrics. e.g what recall denotes in terms of multi class classification. (3 points)
- Generate the loss and f1-score curve for training and validation set. (10 points)
- Generate a ROC-AUC curve and comment on your model accuracy and find the optimal threshold from the curve. (10 points)
- Repeat the steps from 1.1 to 1.7 with sampling in training set. (you can do over sampling to increase the instances of majority class in training set) Compare and comment on the results you get from sampled data and original data distribution. (50 points)

```
### Importing Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import classification_report
import seaborn as sn
import matplotlib.pyplot as plt
import joblib
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from imblearn.over_sampling import SMOTE

import warnings
warnings.filterwarnings("ignore")
```

```
df_train = pd.read_csv("C:/Users/Manju/Documents/Assignments/data set/train.csv")
df_test = pd.read_csv("C:/Users/Manju/Documents/Assignments/data set/test.csv")
```

```
df_train.head()
```

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_
0	462809	Male	No	22	No	Healthcare	1.0	
1	462643	Female	Yes	38	Yes	Engineer	NaN	A
2	466315	Female	Yes	67	Yes	Engineer	1.0	
3	461735	Male	Yes	67	Yes	Lawyer	0.0	
4	462669	Female	Yes	40	Yes	Entertainment	NaN	

```
df_test.head()
```

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_S
0	458989	Female	Yes	36	Yes	Engineer	0.0	
1	458994	Male	Yes	37	Yes	Healthcare	8.0	Ave
2	458996	Female	Yes	69	No	NaN	0.0	
3	459000	Male	Yes	59	No	Executive	11.0	
4	459001	Female	No	19	No	Marketing	NaN	

```
df_train.isnull().sum()
```

```
ID                   0
Gender               0
Ever_Married       140
Age                  0
Graduated           78
Profession         124
Work_Experience    829
Spending_Score       0
Family_Size        335
Var_1               76
Segmentation         0
dtype: int64
```

```
df_test.isnull().sum()
```

```
ID                   0
Gender               0
Ever_Married        50
Age                  0
Graduated           24
Profession          38
Work_Experience    269
Spending_Score       0
Family_Size        113
Var_1               32
dtype: int64
```

```
def col_encode(col,df):
    sum_dict = {}
    c=0
    for i in list(df[col].unique()):
        sum_dict[i]=c+1
        c=c+1
    return sum_dict
```

```
# Shuffling data before training
def df_preprocess(df):

    df['Var_1'].mask(df['Var_1'] == 'Cat_1', 'Others', inplace=True)
    df['Var_1'].mask(df['Var_1'] == 'Cat_2', 'Others', inplace=True)
    df['Var_1'].mask(df['Var_1'] == 'Cat_3', 'Others', inplace=True)
    df['Var_1'].mask(df['Var_1'] == 'Cat_5', 'Others', inplace=True)
    df['Var_1'].mask(df['Var_1'] == 'Cat_7', 'Others', inplace=True)

    df['Ever_Married'] = df['Ever_Married'].fillna(pd.Series(np.random.choice([ i for i in l
                                                  p=list(df["Ever_Married"].value_counts


    df_nulls  = df.dropna()
    cat_cols = ["Gender","Ever_Married","Graduated","Profession","Spending_Score","Var_1"]


    label_encoder = LabelEncoder()
    for col in cat_cols:
        df_nulls[col] = label_encoder.fit_transform(df_nulls[col])

    df_shuffled = df_nulls.sample(frac = 1)

    return df_shuffled


df_train_shuffled  = df_preprocess(df_train)
df_test_shuffled = df_preprocess(df_test)


df_train_shuffled.head()
```

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spendin
7960	459013	1	1	47	1	0	3.0	
7534	460749	1	0	36	1	3	1.0	
577	465163	1	1	46	1	0	11.0	
2579	465351	0	1	51	1	0	8.0	
3345	461428	1	0	49	1	0	1.0	

```
X = df_train_shuffled.drop(["ID","Var_1","Segmentation"] , axis=1)
y = df_train_shuffled["Var_1"]

y = to_categorical(y)


df_train_shuffled.head()
```

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spendin
7960	459013	1	1	47	1	0	3.0	
7534	460749	1	0	36	1	3	1.0	
577	465163	1	1	46	1	0	11.0	
2579	465351	0	1	51	1	0	8.0	
3345	461428	1	0	49	1	0	1.0	

```
# Designing and training a neural network model
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.16,shuffle = True)


def get_f1(y_true, y_pred):
    y_true = tf.cast(y_true, tf.float32)
    y_pred = tf.cast(y_pred, tf.float32)
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val


from tensorflow.keras import backend as K


# validation data for model tuning and monitoring the f1-score while applying the early stop
```

```
def c_mod( ):
    model = Sequential([
        Dense(64, activation='relu', input_shape=(8,)),
        Dense(128, activation='relu'),
        Dense(128, activation='relu'),
        Dense(128, activation='relu'),
        Dense(128, activation='relu'),
        Dense(3, activation='sigmoid')
    ])

    return model



model = c_mod()
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy',get_f1 ]
)


early_stopping = EarlyStopping(
                    patience=140
                )

#validation data for model tuning and monitor the f1-score

history = model.fit(
    X_train,
    y_train,
    epochs=200,
    batch_size=50,
    validation_data=(X_val, y_val),
    verbose=1,
    callbacks = [early_stopping]

)

# evaluate the model
train_acc = model.evaluate(X_train, y_train, verbose=0)
val_acc = model.evaluate(X_val, y_val, verbose=0)
```

```
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7951 - get_f1: 0.7581 - loss:
Epoch 140/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7959 - get_f1: 0.7628 - loss:
Epoch 141/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8056 - get_f1: 0.7649 - loss:
Epoch 142/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7910 - get_f1: 0.7612 - loss:
Epoch 143/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7958 - get_f1: 0.7608 - loss:
Epoch 144/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8026 - get_f1: 0.7662 - loss:
Epoch 145/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7976 - get_f1: 0.7633 - loss:
Epoch 146/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7966 - get_f1: 0.7587 - loss:
Epoch 147/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8083 - get_f1: 0.7655 - loss:
Epoch 148/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8085 - get_f1: 0.7662 - loss:
Epoch 149/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7969 - get_f1: 0.7640 - loss:
Epoch 150/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8016 - get_f1: 0.7608 - loss:
Epoch 151/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7793 - get_f1: 0.7514 - loss:
Epoch 152/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7912 - get_f1: 0.7616 - loss:
Epoch 153/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8118 - get_f1: 0.7621 - loss:
Epoch 154/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8046 - get_f1: 0.7724 - loss:
Epoch 155/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7993 - get_f1: 0.7693 - loss:
Epoch 156/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8158 - get_f1: 0.7813 - loss:
Epoch 157/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8093 - get_f1: 0.7678 - loss:
Epoch 158/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8126 - get_f1: 0.7758 - loss:
Epoch 159/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8202 - get_f1: 0.7838 - loss:
Epoch 160/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8014 - get_f1: 0.7688 - loss:
Epoch 161/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8112 - get_f1: 0.7709 - loss:
Epoch 162/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8255 - get_f1: 0.7773 - loss:
Epoch 163/200
114/114 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8281 - get_f1: 0.7834 - loss:
```

```
## 4.Use test data to calculate the appropriate classification metrics.
X_test = df_test_shuffled.drop(["ID","Var_1"] , axis=1)
y_test = df_test_shuffled["Var_1"]

X_test = np.asarray(X_test)
y_test = np.asarray(y_test)

y_pred = model.predict(X_test, batch_size=64, verbose=0)
y_pred_bool = np.argmax(y_pred, axis=1)
print(classification_report(y_test, y_pred_bool))
```

```
              precision    recall  f1-score   support

           0       0.50      0.31      0.38       327
           1       0.74      0.86      0.80      1438
           2       0.41      0.30      0.35       429

    accuracy                           0.67      2194
   macro avg       0.55      0.49      0.51      2194
weighted avg       0.64      0.67      0.65      2194
```

Explaination of the significance of each metrics.

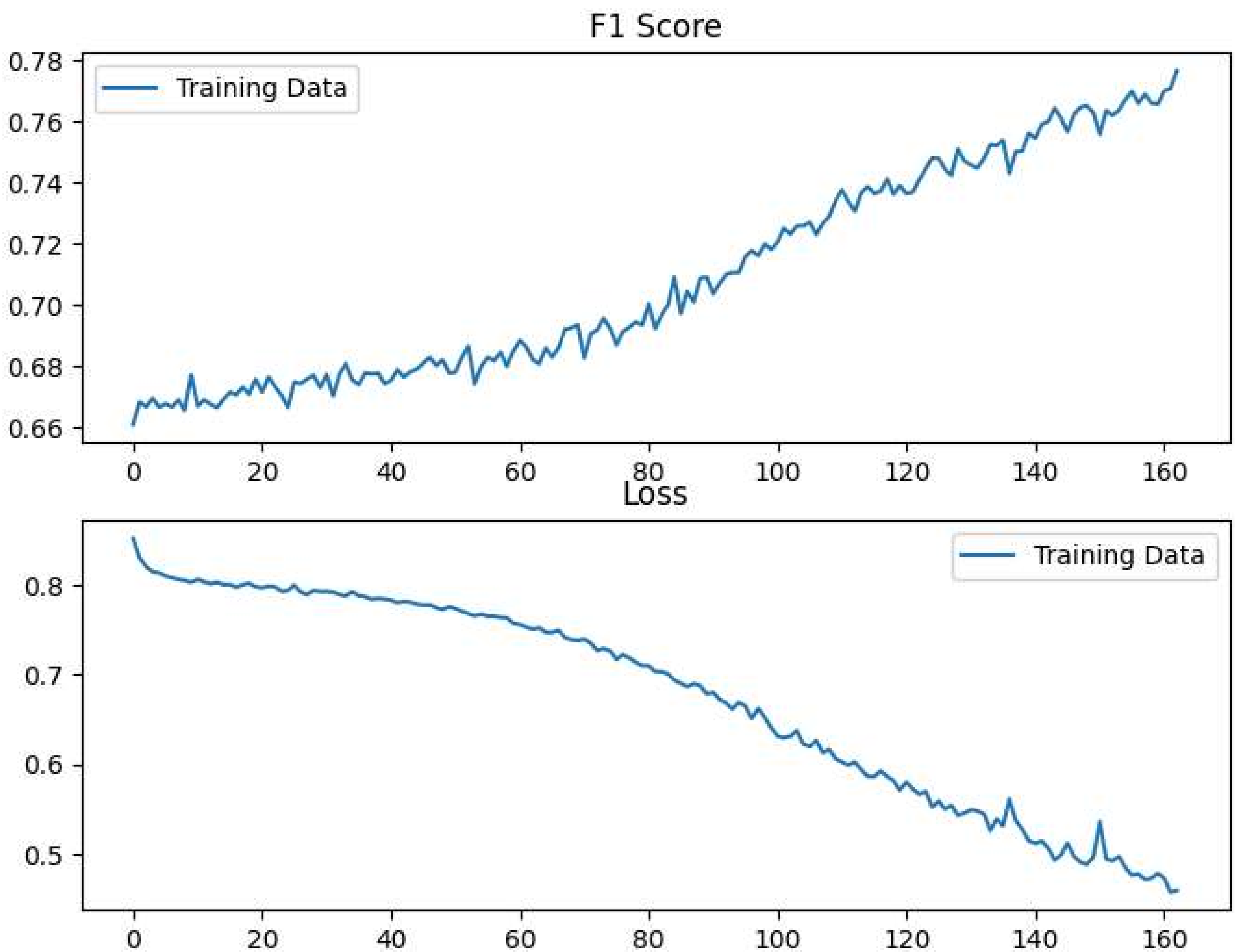
The key matrices used here are as follows:

- Precision: It is the ratio of true positive predictions to the total number of positive predictions.
- Recall: It is the ratio of true positive predictions to the total number of actual positives.
- F1-score: It is the mean of precision and recall.
- Support: It is the number of actual occurrences of the class in the dataset.
- Accuracy: It is the ratio of correctly predicted instances to the total instances in the dataset.
- Macro average: It is the unweighted mean of the metrics across all classes.
- Weighted average: It is the number of true instances for each class to calculate the average metrics.

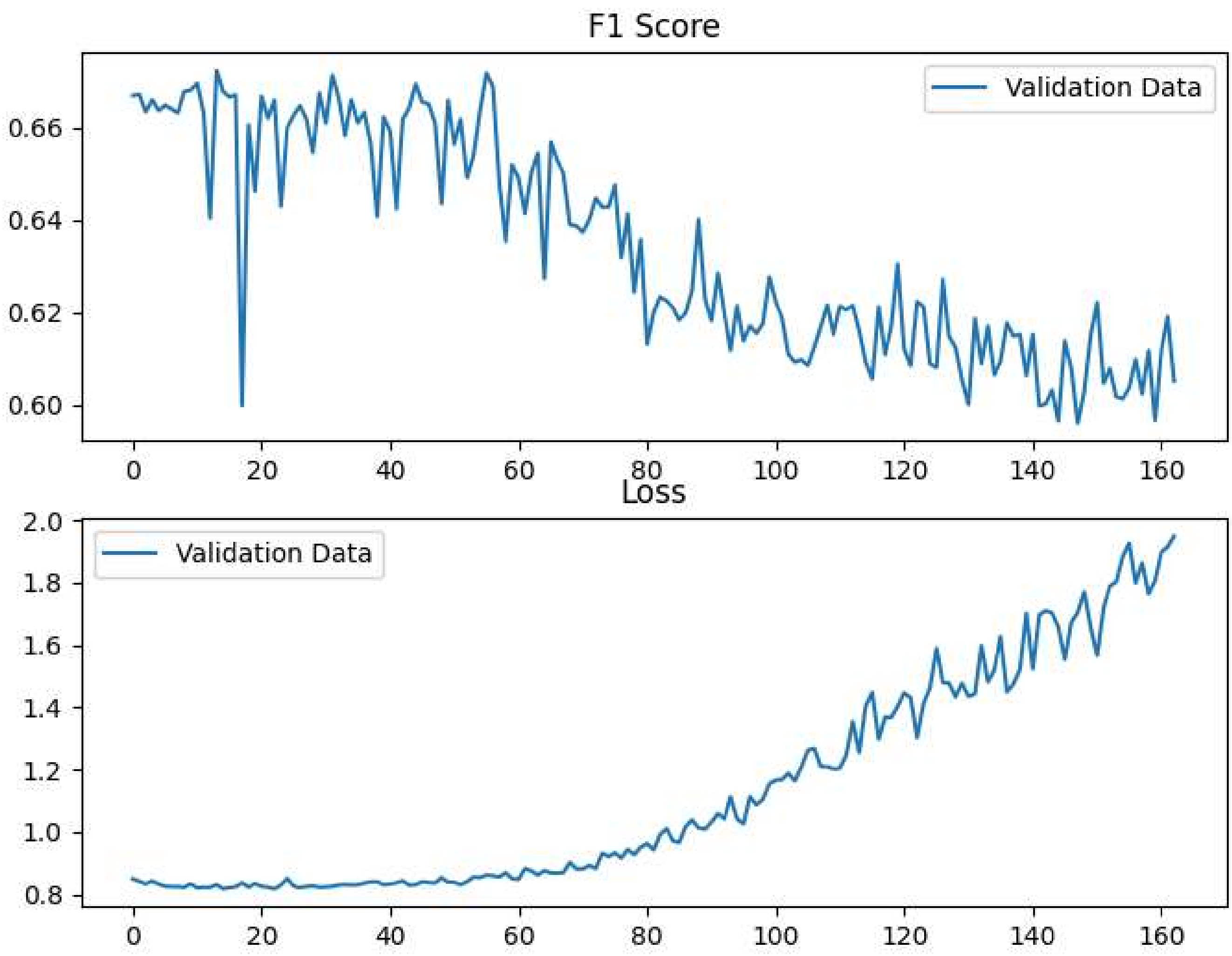
From the above results, Class 1 is well-predicted with high support, higher precision, recall, and F1-score. Classes 0 and 2 have lower recall and F1-scores, indicating the model struggles more with the classes.

The model predicted 67%(overall accurancy (0.67)) of the instances in the test set. classes 0 and 2 needs inprovement.

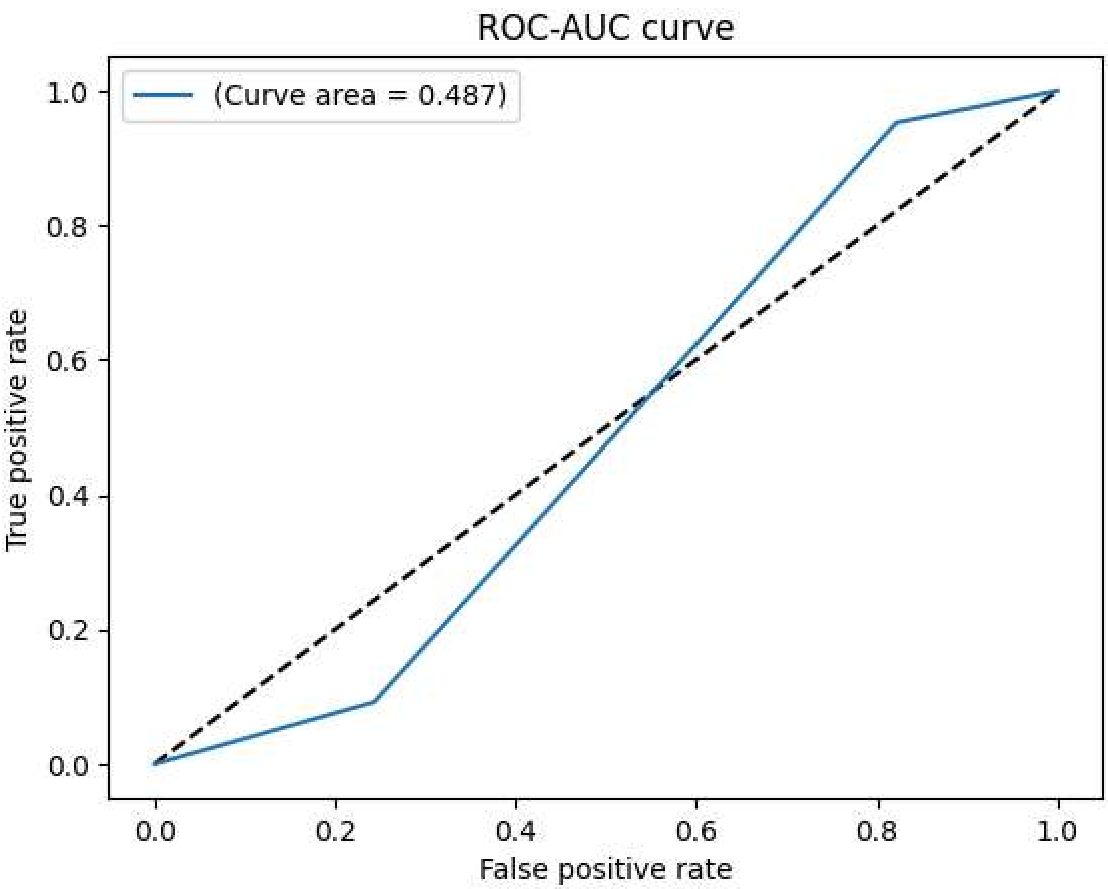
```
## Generating the loss and f1-score curve for training and validation set.
plt.figure(figsize=(8, 6))

plt.subplot(2, 1, 1)
plt.title('F1 Score')
plt.plot(history.history['get_f1'], label='Training Data')
plt.legend()

plt.subplot(2, 1, 2)

plt.title('Loss')
plt.plot(history.history['loss'], label='Training Data')
plt.legend()
plt.show()
```



```
# plot loss during training
plt.figure(figsize=(8, 6))

plt.subplot(2, 1, 1)
plt.title('F1 Score')
plt.plot(history.history['val_get_f1'], label='Validation Data')
plt.legend()
# plot accuracy during training
plt.subplot(2, 1, 2)

plt.title('Loss')
plt.plot(history.history['val_loss'], label='Validation Data')
plt.legend()
plt.show()
```



```
# Generate a ROC-AUC curve and comment on your model accuracy and find the optimal thresholc
y_pred_keras = model.predict(X_test,verbose=0)
y_pred_bool = np.argmax(y_pred_keras,axis=1)
fpr_keras, tpr_keras, thresholds_keras = roc_curve(y_test, y_pred_bool, pos_label=1)
auc_keras = auc(fpr_keras, tpr_keras)
plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_keras, tpr_keras, label='(Curve area = {:.3f})'.format(auc_keras))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC-AUC curve')
plt.legend(loc='best')
plt.show()
```



The above ROC-AUC curve shows the area under the curve 0.487, where the value is very close to 0.5. The model's performance is not good than random guessing

```
# Repeaion of steps with sampling in training set using SMOTE.
distribution.
smote=SMOTE()
X_over,Y_over=smote.fit_resample(X_train,y_train)
```

```
X_train.shape , X_over.shape
```

```
((5685, 8), (11403, 8))
```

```
model_sampled = c_mod()
model_sampled.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy',get_f1 ]
)


early_stopping = EarlyStopping(
                    patience=80
                )

history = model_sampled.fit(
    X_over,
    Y_over,
    epochs=200,
    batch_size=50,
    validation_data=(X_val, y_val),
    verbose=1,
    callbacks = [early_stopping]

)

# evaluate the model
train_acc = model_sampled.evaluate(X_over, Y_over, verbose=0)
val_acc = model_sampled.evaluate(X_val, y_val, verbose=0)
```

```
Epoch 67/200
229/229 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7255 - get_f1: 0.6846 - loss:
Epoch 68/200
229/229 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7195 - get_f1: 0.6915 - loss:
Epoch 69/200
229/229 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7106 - get_f1: 0.6818 - loss:
Epoch 70/200
229/229 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7303 - get_f1: 0.6917 - loss:
Epoch 71/200
229/229 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7302 - get_f1: 0.6991 - loss:
Epoch 72/200
229/229 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7348 - get_f1: 0.6968 - loss:
Epoch 73/200
229/229 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7116 - get_f1: 0.6806 - loss:
Epoch 74/200
229/229 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6870 - get_f1: 0.6693 - loss:
Epoch 75/200
229/229 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7460 - get_f1: 0.7119 - loss:
Epoch 76/200
229/229 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7421 - get_f1: 0.7058 - loss:
Epoch 77/200
229/229 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7497 - get_f1: 0.7078 - loss:
Epoch 78/200
229/229 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7435 - get_f1: 0.7073 - loss:
Epoch 79/200
229/229 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7536 - get_f1: 0.7095 - loss:
Epoch 80/200
229/229 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7488 - get_f1: 0.7110 - loss:
Epoch 81/200
229/229 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7548 - get_f1: 0.7117 - loss:
Epoch 82/200
229/229 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7573 - get_f1: 0.7179 - loss:
Epoch 83/200
229/229 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7456 - get_f1: 0.7096 - loss:
```

```python
X_test = df_test_shuffled.drop(["ID","Var_1"] , axis=1)
y_test = df_test_shuffled["Var_1"]

X_test = np.asarray(X_test)
y_test = np.asarray(y_test)

y_pred = model_sampled.predict(X_test, batch_size=64, verbose=0)

y_pred_bool = np.argmax(y_pred, axis=1)

print(classification_report(y_test, y_pred_bool   ))
```

```
              precision    recall  f1-score   support

           0       0.34      0.45      0.39       327
           1       0.79      0.66      0.72      1438
           2       0.34      0.46      0.39       429
```

```
    accuracy                           0.59      2194
   macro avg       0.49      0.52      0.50      2194
weighted avg       0.63      0.59      0.60      2194
```

From the above results, Class 0 has low precision (0.34) and recall (0.45), which suggests difficulty in correctly identifying and predicting this class. Class 1 performs better with a precision of 0.79 and recall of 0.66. Class 2 is similar to class 0, the model struggles with precision at 0.34 and recall at 0.46.

```
# plot loss during training
plt.figure(figsize=(8, 6))

plt.subplot(2, 1, 1)
plt.title('F1 Score')
plt.plot(history.history['get_f1'], label='Training Data')
plt.legend()
# plot accuracy during training
plt.subplot(2, 1, 2)

plt.title('Loss')
plt.plot(history.history['loss'], label='Training Data')
plt.legend()
plt.show()
```


```
# plot loss during training
plt.figure(figsize=(8, 6))

plt.subplot(2, 1, 1)
plt.title('F1 Score')
plt.plot(history.history['val_get_f1'], label='Validation Data')
plt.legend()
# plot accuracy during training
plt.subplot(2, 1, 2)

plt.title('Loss')
plt.plot(history.history['val_loss'], label='Validataion Data')
plt.legend()
plt.show()
```