Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months. Dataset Description: The case study is from an open-source dataset from Kaggle. The dataset contains 10,000 sample points with 14 distinct features such as CustomerId, CreditScore, Geography, Gender, Age, Tenure, Balance, etc.

Link to the Kaggle project: https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling

Perform following steps:

1. Read the dataset.
2. Distinguish the feature and target set and divide the data set into training and test sets.
3. Normalize the train and test data.
4. Initialize and build the model. Identify the points of improvement and implement the same.
5. Print the accuracy score and confusion matrix.

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

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

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt #Importing the libraries
```

```
df = pd.read_csv("/content/drive/MyDrive/ML/Churn_Modelling.csv")
```

Preprocessing.

```
df.head()
```

```
   RowNumber  CustomerId   Surname  CreditScore Geography  Gender  Age  
\
0          1    15634602  Hargrave          619    France  Female   42   

1          2    15647311      Hill          608     Spain  Female   41   

2          3    15619304      Onio          502    France  Female   42   

3          4    15701354      Boni          699    France  Female   39   

4          5    15737888  Mitchell          850     Spain  Female   43   


   Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
```

```
0       2       0.00              1          1               1
1       1   83807.86              1          0               1
2       8  159660.80              3          1               0
3       1       0.00              2          0               0
4       2  125510.82              1          1               1

   EstimatedSalary  Exited
0        101348.88       1
1        112542.58       0
2        113931.57       1
3         93826.63       0
4         79084.10       0
```

```
df.shape
```

```
(10000, 14)
```

```
df.describe()
```

```
         RowNumber    CustomerId   CreditScore           Age
Tenure  \
count  10000.00000  1.000000e+04  10000.000000  10000.000000
10000.000000
mean    5000.50000  1.569094e+07    650.528800     38.921800
5.012800
std     2886.89568  7.193619e+04     96.653299     10.487806
2.892174
min        1.00000  1.556570e+07    350.000000     18.000000
0.000000
25%     2500.75000  1.562853e+07    584.000000     32.000000
3.000000
50%     5000.50000  1.569074e+07    652.000000     37.000000
5.000000
75%     7500.25000  1.575323e+07    718.000000     44.000000
7.000000
max    10000.00000  1.581569e+07    850.000000     92.000000
10.000000

             Balance  NumOfProducts    HasCrCard  IsActiveMember  \
count   10000.000000   10000.000000  10000.00000    10000.000000
mean    76485.889288       1.530200      0.70550        0.515100
std     62397.405202       0.581654      0.45584        0.499797
min         0.000000       1.000000      0.00000        0.000000
25%         0.000000       1.000000      0.00000        0.000000
50%     97198.540000       1.000000      1.00000        1.000000
75%    127644.240000       2.000000      1.00000        1.000000
max    250898.090000       4.000000      1.00000        1.000000

       EstimatedSalary        Exited
count     10000.000000  10000.000000
mean     100090.239881      0.203700
```

```
std      57510.492818      0.402769
min         11.580000      0.000000
25%      51002.110000      0.000000
50%     100193.915000      0.000000
75%     149388.247500      0.000000
max     199992.480000      1.000000
```

```
df.isnull()
```

```
      RowNumber  CustomerId  Surname  CreditScore  Geography  Gender
Age  \
0         False       False    False        False      False   False
False
1         False       False    False        False      False   False
False
2         False       False    False        False      False   False
False
3         False       False    False        False      False   False
False
4         False       False    False        False      False   False
False
...         ...         ...      ...          ...        ...     ...
...
9995      False       False    False        False      False   False
False
9996      False       False    False        False      False   False
False
9997      False       False    False        False      False   False
False
9998      False       False    False        False      False   False
False
9999      False       False    False        False      False   False
False

      Tenure  Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0      False    False          False      False           False
1      False    False          False      False           False
2      False    False          False      False           False
3      False    False          False      False           False
4      False    False          False      False           False
...      ...      ...            ...        ...             ...
9995   False    False          False      False           False
9996   False    False          False      False           False
9997   False    False          False      False           False
9998   False    False          False      False           False
9999   False    False          False      False           False

      EstimatedSalary  Exited
0               False   False
1               False   False
```

```
2               False   False
3               False   False
4               False   False
...               ...     ...
9995            False   False
9996            False   False
9997            False   False
9998            False   False
9999            False   False

[10000 rows x 14 columns]
```

```
df.isnull().sum()
```

```
RowNumber          0
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
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
```

```
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

```
df.dtypes
```

```
RowNumber            int64
CustomerId           int64
Surname             object
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

```
df.columns
```

```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore',
'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts',
'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

```
df = df.drop(['RowNumber', 'Surname', 'CustomerId'], axis= 1)
#Dropping the unnecessary columns
```

```
df.head()
```

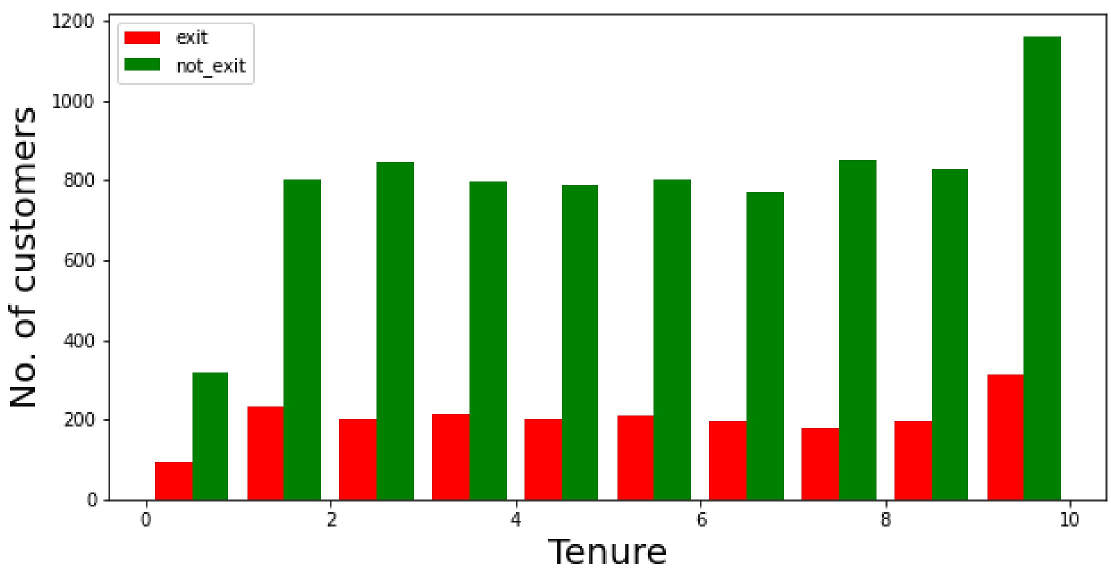
```
   CreditScore Geography  Gender  Age  Tenure    Balance
NumOfProducts  \
0          619    France  Female   42       2       0.00
1
1          608     Spain  Female   41       1   83807.86
1
2          502    France  Female   42       8  159660.80
3
3          699    France  Female   39       1       0.00
2
4          850     Spain  Female   43       2  125510.82
1

   HasCrCard  IsActiveMember  EstimatedSalary  Exited
0          1               1        101348.88       1
1          0               1        112542.58       0
2          1               0        113931.57       1
```

```
3           0                0         93826.63       0
4           1                1         79084.10       0
```

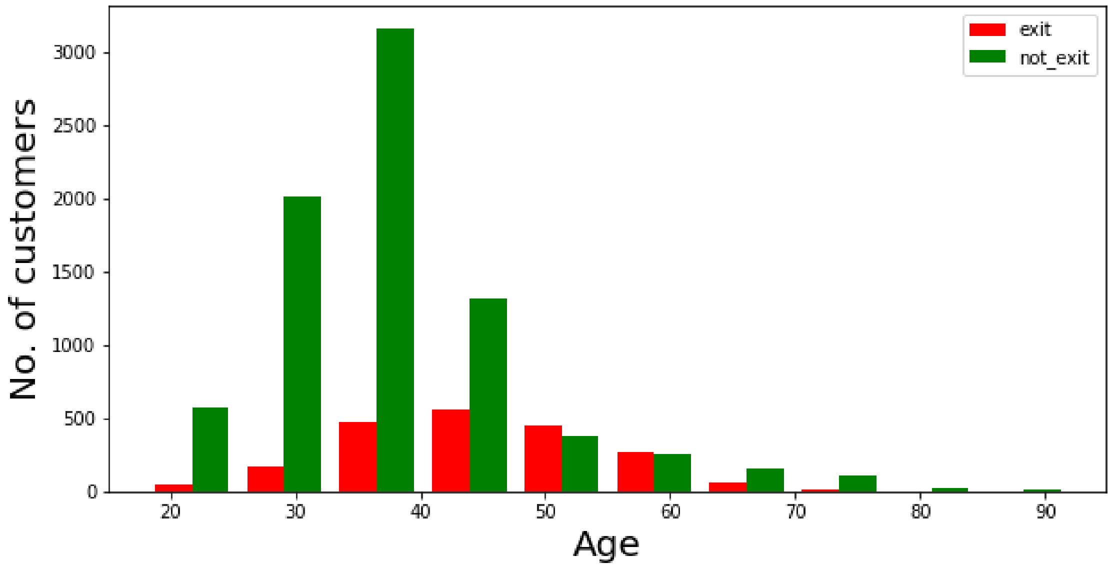
Visualization

```
def visualization(x, y, xlabel):
    plt.figure(figsize=(10,5))
    plt.hist([x, y], color=['red', 'green'], label = ['exit', 
'not_exit'])
    plt.xlabel(xlabel,fontsize=20)
    plt.ylabel("No. of customers", fontsize=20)
    plt.legend()

df_churn_exited = df[df['Exited']==1]['Tenure']
df_churn_not_exited = df[df['Exited']==0]['Tenure']

visualization(df_churn_exited, df_churn_not_exited, "Tenure")
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/fromnumeric.py:3208: 
VisibleDeprecationWarning: Creating an ndarray from ragged nested 
sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays 
with different lengths or shapes) is deprecated. If you meant to do 
this, you must specify 'dtype=object' when creating the ndarray.
  return asarray(a).size
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:13
76: VisibleDeprecationWarning: Creating an ndarray from ragged nested 
sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays 
with different lengths or shapes) is deprecated. If you meant to do 
this, you must specify 'dtype=object' when creating the ndarray.
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else 
np.asarray(X))
```



```
df_churn_exited2 = df[df['Exited']==1]['Age']
df_churn_not_exited2 = df[df['Exited']==0]['Age']

visualization(df_churn_exited2, df_churn_not_exited2, "Age")
```



Converting the Categorical Variables

```
X =
df[['CreditScore','Gender','Age','Tenure','Balance','NumOfProducts','H
asCrCard','IsActiveMember','EstimatedSalary']]
states = pd.get_dummies(df['Geography'],drop_first = True)
gender = pd.get_dummies(df['Gender'],drop_first = True)


df = pd.concat([df,gender,states], axis = 1)
```

Splitting the training and testing Dataset

```
df.head()

   CreditScore Geography  Gender  Age  Tenure    Balance
NumOfProducts  \
0          619    France  Female   42       2       0.00
1
1          608     Spain  Female   41       1   83807.86
1
2          502    France  Female   42       8  159660.80
3
3          699    France  Female   39       1       0.00
2
4          850     Spain  Female   43       2  125510.82
1
```

```
   HasCrCard  IsActiveMember  EstimatedSalary  Exited  Male  Germany  
Spain  
0          1               1        101348.88       1     0        0      
0  
1          0               1        112542.58       0     0        0      
1  
2          1               0        113931.57       1     0        0      
0  
3          0               0         93826.63       0     0        0      
0  
4          1               1         79084.10       0     0        0      
1  
```

```
X = 
df[['CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard'
,'IsActiveMember','EstimatedSalary','Male','Germany','Spain']]

y = df['Exited']

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.30)
```

Normalizing the values with mean as 0 and Standard Deviation as 1

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

X_train  = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

X_train

array([[ 0.18602342, -1.03446063, -1.38305456, ..., -1.08316117,
         1.7561361 , -0.57691038],
       [ 0.59018077,  0.87112474,  0.69263926, ..., -1.08316117,
         1.7561361 , -0.57691038],
       [-0.290675  , -0.27222648, -1.72900353, ..., -1.08316117,
        -0.56943195,  1.73337147],
       ...,
       [ 1.27413937,  0.29944913,  1.03858823, ..., -1.08316117,
         1.7561361 , -0.57691038],
       [ 0.99433812, -1.32029844, -0.69115662, ..., -1.08316117,
        -0.56943195,  1.73337147],
       [-1.39951697,  0.10889059,  1.73048617, ...,  0.92322364,
        -0.56943195,  1.73337147]])

X_test

array([[-0.67410634, -0.17694721,  0.34669029, ...,  0.92322364,
         1.7561361 , -0.57691038],
```

```
       [-0.5186612 , -0.08166794,  1.3845372 , ..., -1.08316117,
        -0.56943195, -0.57691038],
       [ 0.20674943, -1.32029844,  0.69263926, ...,  0.92322364,
        -0.56943195, -0.57691038],
       ...,
       [ 0.94252308,  1.34752109, -0.69115662, ...,  0.92322364,
        -0.56943195,  1.73337147],
       [ 0.09275633,  0.01361132, -1.38305456, ..., -1.08316117,
         1.7561361 , -0.57691038],
       [ 0.49691369,  1.25224182, -1.38305456, ..., -1.08316117,
         1.7561361 , -0.57691038]])
```

Building the Classifier Model using Keras

```
import keras #Keras is the wrapper on the top of tenserflow
#Can use Tenserflow as well but won't be able to understand the errors 
initially.

from keras.models import Sequential #To create sequential neural 
network
from keras.layers import Dense #To create hidden layers

classifier = Sequential()

#To add the layers
#Dense helps to contruct the neurons
#Input Dimension means we have 11 features
# Units is to create the hidden layers
#Uniform helps to distribute the weight uniformly
classifier.add(Dense(activation = "relu",input_dim = 11,units = 
6,kernel_initializer = "uniform"))

classifier.add(Dense(activation = "relu",units = 6,kernel_initializer 
= "uniform"))   #Adding second hidden layers

classifier.add(Dense(activation = "sigmoid",units = 
1,kernel_initializer = "uniform")) #Final neuron will be having 
siigmoid function

classifier.compile(optimizer="adam",loss = 
'binary_crossentropy',metrics = ['accuracy']) #To compile the 
Artificial Neural Network. Ussed Binary crossentropy as we just have 
only two output

classifier.summary() #3 layers created. 6 neurons in 1st,6neurons in 
2nd layer and 1 neuron in last
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
```

```
 dense (Dense)               (None, 6)                 72

 dense_1 (Dense)             (None, 6)                 42

 dense_2 (Dense)             (None, 1)                 7

=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
```

```
classifier.fit(X_train,y_train,batch_size=10,epochs=50) #Fitting the
ANN to training dataset
```

```
Epoch 1/50
700/700 [==============================] - 1s 911us/step - loss:
0.4948 - accuracy: 0.7971
Epoch 2/50
700/700 [==============================] - 1s 942us/step - loss:
0.4291 - accuracy: 0.7977
Epoch 3/50
700/700 [==============================] - 1s 896us/step - loss:
0.4238 - accuracy: 0.8017
Epoch 4/50
700/700 [==============================] - 1s 919us/step - loss:
0.4185 - accuracy: 0.8230
Epoch 5/50
700/700 [==============================] - 1s 894us/step - loss:
0.4144 - accuracy: 0.8281
Epoch 6/50
700/700 [==============================] - 1s 921us/step - loss:
0.4108 - accuracy: 0.8299
Epoch 7/50
700/700 [==============================] - 1s 907us/step - loss:
0.4081 - accuracy: 0.8311
Epoch 8/50
700/700 [==============================] - 1s 889us/step - loss:
0.4064 - accuracy: 0.8316
Epoch 9/50
700/700 [==============================] - 1s 888us/step - loss:
0.4050 - accuracy: 0.8323
Epoch 10/50
700/700 [==============================] - 1s 893us/step - loss:
0.4037 - accuracy: 0.8333
Epoch 11/50
700/700 [==============================] - 1s 909us/step - loss:
0.4029 - accuracy: 0.8324
Epoch 12/50
700/700 [==============================] - 1s 931us/step - loss:
0.4023 - accuracy: 0.8333
```

```
Epoch 13/50
700/700 [==============================] - 1s 877us/step - loss:
0.4012 - accuracy: 0.8347
Epoch 14/50
700/700 [==============================] - 1s 910us/step - loss:
0.4011 - accuracy: 0.8324
Epoch 15/50
700/700 [==============================] - 1s 917us/step - loss:
0.4001 - accuracy: 0.8344
Epoch 16/50
700/700 [==============================] - 1s 939us/step - loss:
0.4002 - accuracy: 0.8329
Epoch 17/50
700/700 [==============================] - 1s 881us/step - loss:
0.4000 - accuracy: 0.8354
Epoch 18/50
700/700 [==============================] - 1s 902us/step - loss:
0.3995 - accuracy: 0.8346
Epoch 19/50
700/700 [==============================] - 1s 901us/step - loss:
0.3992 - accuracy: 0.8337
Epoch 20/50
700/700 [==============================] - 1s 903us/step - loss:
0.3991 - accuracy: 0.8349
Epoch 21/50
700/700 [==============================] - 1s 904us/step - loss:
0.3988 - accuracy: 0.8320
Epoch 22/50
700/700 [==============================] - 1s 881us/step - loss:
0.3986 - accuracy: 0.8346
Epoch 23/50
700/700 [==============================] - 1s 878us/step - loss:
0.3985 - accuracy: 0.8341
Epoch 24/50
700/700 [==============================] - 1s 878us/step - loss:
0.3978 - accuracy: 0.8346
Epoch 25/50
700/700 [==============================] - 1s 886us/step - loss:
0.3982 - accuracy: 0.8347
Epoch 26/50
700/700 [==============================] - 1s 920us/step - loss:
0.3973 - accuracy: 0.8359
Epoch 27/50
700/700 [==============================] - 1s 895us/step - loss:
0.3977 - accuracy: 0.8350
Epoch 28/50
700/700 [==============================] - 1s 933us/step - loss:
0.3976 - accuracy: 0.8350
Epoch 29/50
700/700 [==============================] - 1s 895us/step - loss:
```

```
0.3974 - accuracy: 0.8350
Epoch 30/50
700/700 [==============================] - 1s 890us/step - loss:
0.3970 - accuracy: 0.8359
Epoch 31/50
700/700 [==============================] - 1s 903us/step - loss:
0.3972 - accuracy: 0.8360
Epoch 32/50
700/700 [==============================] - 1s 905us/step - loss:
0.3972 - accuracy: 0.8357
Epoch 33/50
700/700 [==============================] - 1s 887us/step - loss:
0.3970 - accuracy: 0.8351
Epoch 34/50
700/700 [==============================] - 1s 906us/step - loss:
0.3970 - accuracy: 0.8361
Epoch 35/50
700/700 [==============================] - 1s 969us/step - loss:
0.3967 - accuracy: 0.8346
Epoch 36/50
700/700 [==============================] - 1s 914us/step - loss:
0.3963 - accuracy: 0.8360
Epoch 37/50
700/700 [==============================] - 1s 900us/step - loss:
0.3965 - accuracy: 0.8360
Epoch 38/50
700/700 [==============================] - 1s 905us/step - loss:
0.3964 - accuracy: 0.8359
Epoch 39/50
700/700 [==============================] - 1s 916us/step - loss:
0.3964 - accuracy: 0.8349
Epoch 40/50
700/700 [==============================] - 1s 918us/step - loss:
0.3960 - accuracy: 0.8366
Epoch 41/50
700/700 [==============================] - 1s 890us/step - loss:
0.3957 - accuracy: 0.8366
Epoch 42/50
700/700 [==============================] - 1s 914us/step - loss:
0.3962 - accuracy: 0.8360
Epoch 43/50
700/700 [==============================] - 1s 882us/step - loss:
0.3955 - accuracy: 0.8361
Epoch 44/50
700/700 [==============================] - 1s 870us/step - loss:
0.3953 - accuracy: 0.8369
Epoch 45/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3948
- accuracy: 0.8387
Epoch 46/50
```

```
700/700 [==============================] - 1s 1ms/step - loss: 0.3946
- accuracy: 0.8386
Epoch 47/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3948
- accuracy: 0.8377
Epoch 48/50
700/700 [==============================] - 1s 884us/step - loss:
0.3946 - accuracy: 0.8380
Epoch 49/50
700/700 [==============================] - 1s 905us/step - loss:
0.3941 - accuracy: 0.8376
Epoch 50/50
700/700 [==============================] - 1s 908us/step - loss:
0.3935 - accuracy: 0.8376

<keras.callbacks.History at 0x7fd6f5d29410>
```

```
y_pred =classifier.predict(X_test)
y_pred = (y_pred > 0.5) #Predicting the result
```

```
from sklearn.metrics import
confusion_matrix,accuracy_score,classification_report
```

```
cm = confusion_matrix(y_test,y_pred)
```

```
cm
```

```
array([[2308,   71],
       [ 395,  226]])
```

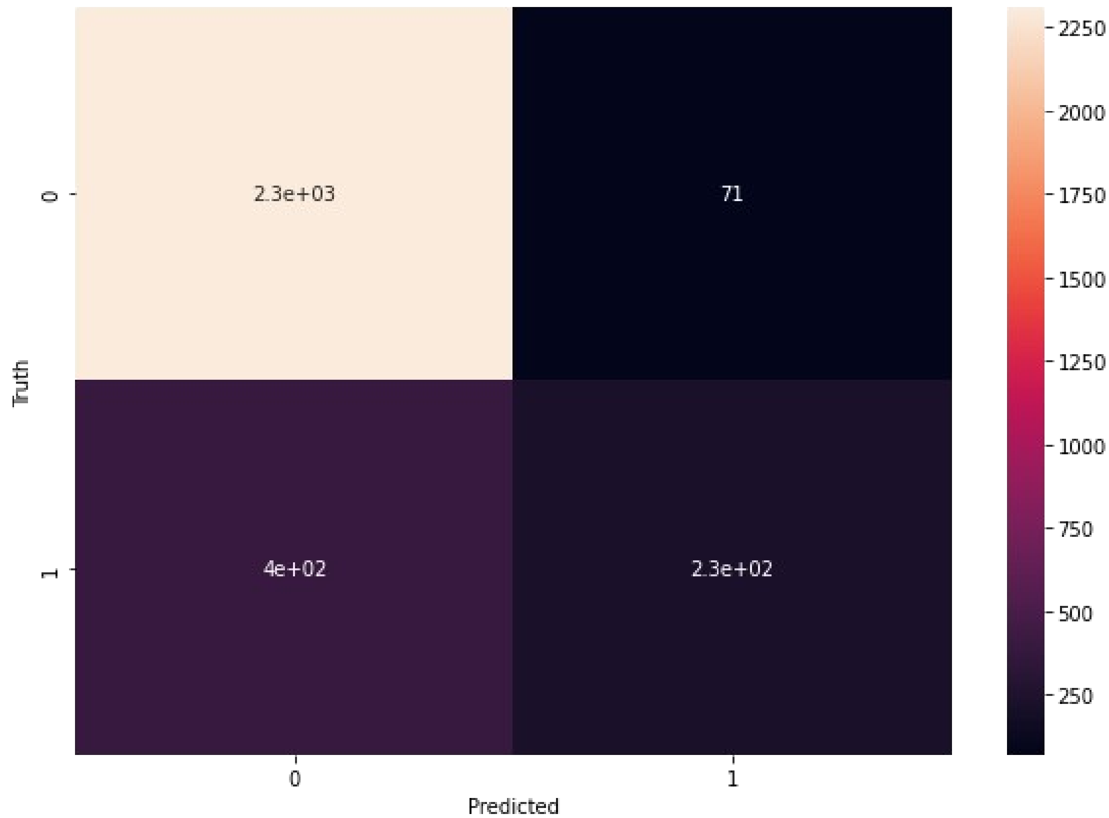
```
accuracy = accuracy_score(y_test,y_pred)
```

```
accuracy
```

```
0.8446666666666667
```

```
plt.figure(figsize = (10,7))
sns.heatmap(cm,annot = True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Text(69.0, 0.5, 'Truth')
```




```
print(classification_report(y_test,y_pred))

              precision    recall  f1-score   support

           0       0.85      0.97      0.91      2379
           1       0.76      0.36      0.49       621

    accuracy                           0.84      3000
   macro avg       0.81      0.67      0.70      3000
weighted avg       0.83      0.84      0.82      3000
```