Madhur Jaripatke

Roll No. 52

BE A Computer

RMDSSOE, Warje, Pune

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 (5 points).

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from sklearn.metrics import confusion_matrix, accuracy_score, classificatior
```

```
2024-10-24 22:04:27.219031: I tensorflow/core/util/port.cc:153] oneDNN custo
m operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn t
hem off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-10-24 22:04:27.219821: I external/local_xla/xla/tsl/cuda/cudart_stub.c
c:32] Could not find cuda drivers on your machine, GPU will not be used.
2024-10-24 22:04:27.223789: I external/local_xla/xla/tsl/cuda/cudart_stub.c
c:32] Could not find cuda drivers on your machine, GPU will not be used.
2024-10-24 22:04:27.233450: E external/local_xla/xla/stream_executor/cuda/cu
da_fft.cc:485] Unable to register cuFFT factory: Attempting to register fact
ory for plugin cuFFT when one has already been registered
2024-10-24 22:04:27.247799: E external/local_xla/xla/stream_executor/cuda/cu
da_dnn.cc:8454] Unable to register cuDNN factory: Attempting to register fac
tory for plugin cuDNN when one has already been registered
2024-10-24 22:04:27.253778: E external/local_xla/xla/stream_executor/cuda/cu
da_blas.cc:1452] Unable to register cuBLAS factory: Attempting to register f
actory for plugin cuBLAS when one has already been registered
2024-10-24 22:04:27.267679: I tensorflow/core/platform/cpu_feature_guard.cc:
210] This TensorFlow binary is optimized to use available CPU instructions i
n performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other
operations, rebuild TensorFlow with the appropriate compiler flags.
2024-10-24 22:04:28.514490: W tensorflow/compiler/tf2tensorrt/utils/py_util
s.cc:38] TF-TRT Warning: Could not find TensorRT
```

In [2]:
```
df = pd.read_csv('./Datasets/churn_modelling.csv')
df.head()
```

Out[2]:

	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

In [3]:
```
df.shape
```

Out[3]:
```
(10000, 14)
```

In [4]:
```
df.describe
```

Out[4]:
```
<bound method NDFrame.describe of       RowNumber  CustomerId    Surname  C
reditScore 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
...         ...         ...        ...          ...       ...     ...  ...
9995       9996    15606229   Obijiaku          771    France    Male   39
9996       9997    15569892  Johnstone          516    France    Male   35
9997       9998    15584532        Liu          709    France  Female   36
9998       9999    15682355  Sabbatini          772   Germany    Male   42
9999      10000    15628319     Walker          792    France  Female   28

      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
...      ...        ...            ...        ...             ...
9995       5       0.00              2          1               0
9996      10   57369.61              1          1               1
9997       7       0.00              1          0               1
9998       3   75075.31              2          1               0
9999       4  130142.79              1          1               0

      EstimatedSalary  Exited
0           101348.88       1
1           112542.58       0
2           113931.57       1
3            93826.63       0
4            79084.10       0
...               ...     ...
9995         96270.64       0
9996        101699.77       0
9997         42085.58       1
9998         92888.52       1
9999         38190.78       0

[10000 rows x 14 columns]>
```

In [5]:
```
df.isnull()
df.isnull().sum()
```

Out[5]:
```
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
```

In [6]:
```
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
```

In [7]:
```
df.dtypes
```

Out[7]:
```
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
```

In [8]:
```
df.columns
```

Out[8]:
```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

In [9]:
```
df = df.drop(['RowNumber', 'Surname', 'CustomerId'], axis = 1)
df.head()
```

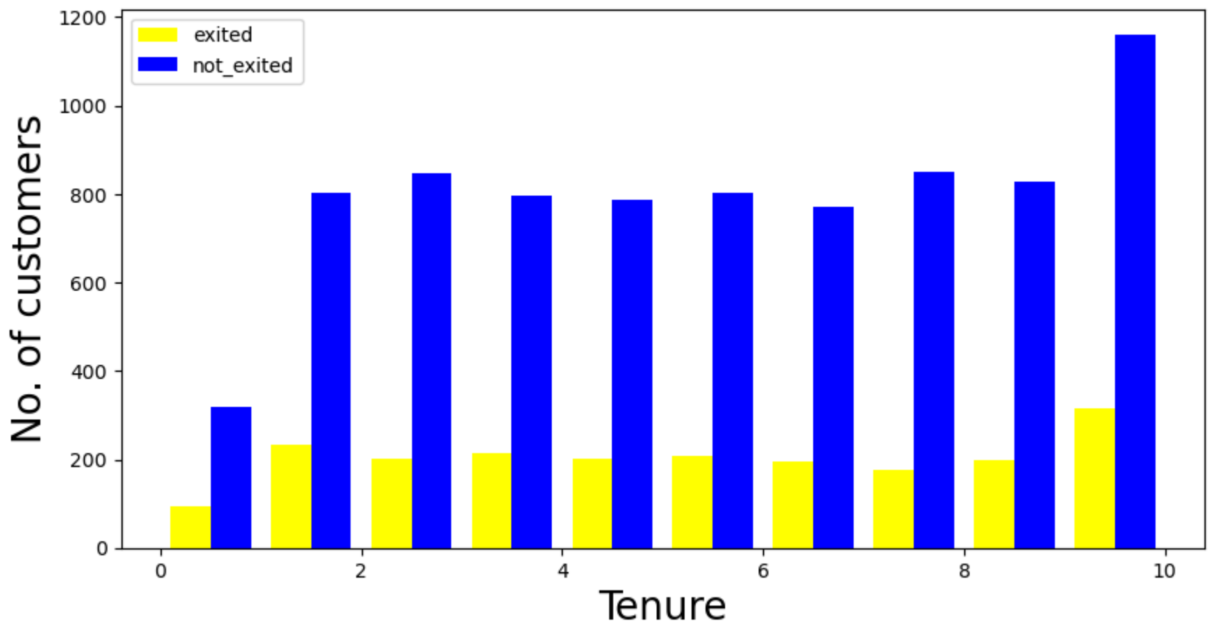
Out[9]:

	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

In [10]:
```
def visualization(x, y, xlabel):
    plt.figure(figsize=(10,5))
    plt.hist([x, y], color=['yellow', 'blue'], label = ['exited', 'not_exite
    plt.xlabel(xlabel,fontsize=20)
    plt.ylabel('No. of customers', fontsize=20)
    plt.legend()
```

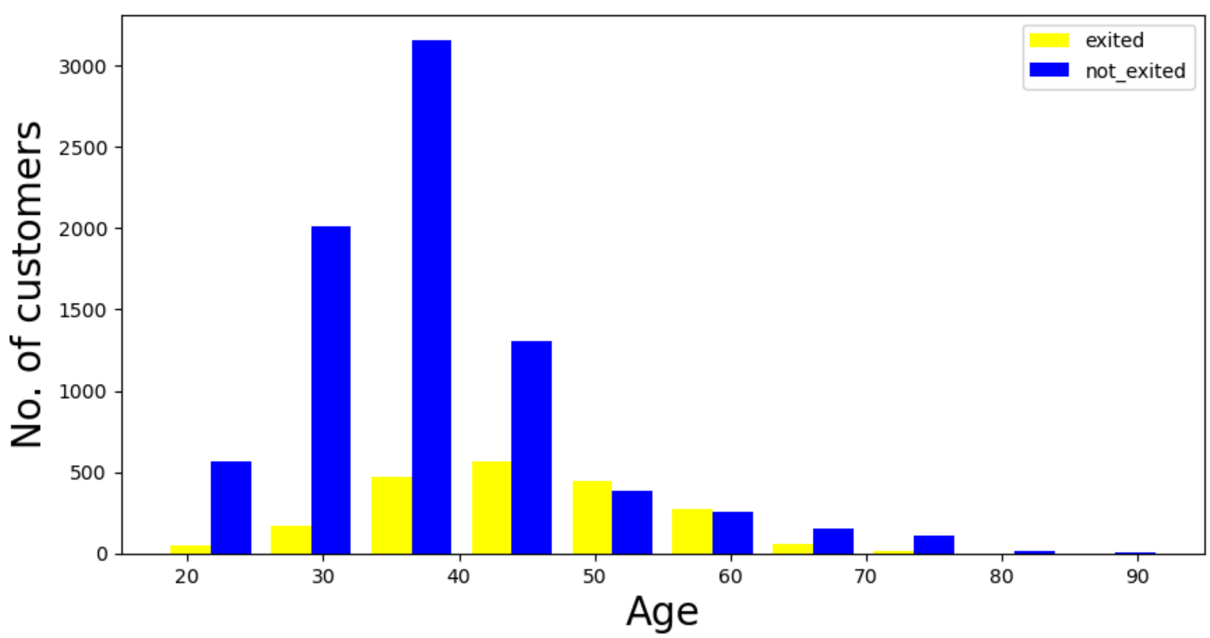
In [11]:
```
df_churn_exited = df[df['Exited']==1]['Tenure']
df_churn_not_exited = df[df['Exited']==0]['Tenure']
```

In [12]:
```
visualization(df_churn_exited, df_churn_not_exited, 'Tenure')
```



In [13]:
```
df_churn_exited2 = df[df['Exited']==1]['Age']
df_churn_not_exited2 = df[df['Exited']==0]['Age']
```

In [14]:
```
visualization(df_churn_exited2, df_churn_not_exited2, 'Age')
```



In [15]:
```
x = df[['CreditScore', 'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts'
        'HasCrCard', 'IsActiveMember', 'EstimatedSalary']]
states = pd.get_dummies(df['Geography'], drop_first = True)
gender = pd.get_dummies(df['Gender'], drop_first = True)
```

In [16]:
```
df = pd.concat([df,gender,states], axis = 1)
df.head()
```

Out[16]:

	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

In [17]:
```
x = df[['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'IsActiv
y = df['Exited']
```

In [18]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3)
```

In [19]:
```
sc = StandardScaler()
```

In [20]:
```
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

In [21]:
```
print('Training Data:\n',x_train)
```

```
Training Data:
 [[-0.25855342 -0.18536388 -0.33750179 ... -1.10522259  1.72416885
  -0.56811207]
 [ 1.06208156  0.00809775  1.04894922 ... -1.10522259  1.72416885
  -0.56811207]
 [ 0.31474549  0.20155938 -1.72395279 ... -1.10522259 -0.5799896
  -0.56811207]
 ...
 [ 0.73448218  0.10482857  0.35572371 ... -1.10522259  1.72416885
  -0.56811207]
 [ 1.2463562   1.16886755  1.39556197 ... -1.10522259  1.72416885
  -0.56811207]
 [-2.10129989 -1.34613368  0.35572371 ...  0.90479511 -0.5799896
  -0.56811207]]
```

In [22]:
```
print('Testing Data:\n',x_test)
```

```
Testing Data:
 [[-0.35069074 -0.18536388  1.39556197 ...  0.90479511 -0.5799896
  -0.56811207]
 [-1.48705107 -0.37882552  1.04894922 ...  0.90479511 -0.5799896
   1.76021608]
 [ 2.04487968 -0.2820947   0.70233647 ...  0.90479511  1.72416885
  -0.56811207]
 ...
 [ 0.10999588 -0.47555633 -1.72395279 ... -1.10522259 -0.5799896
  -0.56811207]
 [ 0.30450801 -0.18536388  1.74217472 ... -1.10522259 -0.5799896
  -0.56811207]
 [-0.95470209  0.2982902   0.35572371 ...  0.90479511 -0.5799896
  -0.56811207]]
```

In [23]:
```
classifier = Sequential()
```

In [24]:
```
classifier.add(Dense(units = 6, kernel_initializer = 'he_uniform', activatic
classifier.add(Dense(units = 6, kernel_initializer = 'he_uniform', activatic
classifier.add(Dense(units = 1, kernel_initializer = 'glorot_uniform', activ
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics
```

```
/home/madhurj20/.local/lib/python3.12/site-packages/keras/src/layers/core/de
nse.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to
a layer. When using Sequential models, prefer using an `Input(shape)` object
as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [25]:
```
classifier.summary()
```

Model: "sequential"

Layer (type)	**Output Shape**	**Par**
dense (Dense)	(None, 6)	
dense_1 (Dense)	(None, 6)	
dense_2 (Dense)	(None, 1)	

Total params: 115 (460.00 B)

Trainable params: 115 (460.00 B)

Non-trainable params: 0 (0.00 B)

In [26]:
```
classifier.fit(x_train, y_train, batch_size = 10, epochs = 50)
```

```
Epoch 1/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.7247 - loss: 0.5849
Epoch 2/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8061 - loss: 0.4421
Epoch 3/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8136 - loss: 0.4182
Epoch 4/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8304 - loss: 0.3938
Epoch 5/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8349 - loss: 0.3846
Epoch 6/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8385 - loss: 0.3726
Epoch 7/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8504 - loss: 0.3520
Epoch 8/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8521 - loss: 0.3589
Epoch 9/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8591 - loss: 0.3498
Epoch 10/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8552 - loss: 0.3442
Epoch 11/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8570 - loss: 0.3499
Epoch 12/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8606 - loss: 0.3435
Epoch 13/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8533 - loss: 0.3508
Epoch 14/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8572 - loss: 0.3427
Epoch 15/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8560 - loss: 0.3404
Epoch 16/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8582 - loss: 0.3394
Epoch 17/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 967us/step - accuracy: 0.8659 - loss: 0.3279
Epoch 18/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 996us/step - accuracy: 0.8664 - loss: 0.3179
Epoch 19/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8630 - loss: 0.3274
Epoch 20/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8603 - loss: 0.3418
Epoch 21/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8630 - loss: 0.3286
Epoch 22/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8563 - loss: 0.3371
Epoch 23/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8549 - loss: 0.3363
Epoch 24/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8547 - loss: 0.3454
Epoch 25/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8482 - loss: 0.3524
Epoch 26/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8600 - loss: 0.3393
Epoch 27/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8607 - loss: 0.3342
Epoch 28/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8637 - loss: 0.3305
```

```
Epoch 29/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8624 - loss: 0.3296
Epoch 30/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8633 - loss: 0.3279
Epoch 31/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8646 - loss: 0.3298
Epoch 32/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8602 - loss: 0.3331
Epoch 33/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8624 - loss: 0.3312
Epoch 34/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8614 - loss: 0.3294
Epoch 35/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8526 - loss: 0.3440
Epoch 36/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8642 - loss: 0.3304
Epoch 37/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8522 - loss: 0.3449
Epoch 38/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8548 - loss: 0.3444
Epoch 39/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8596 - loss: 0.3384
Epoch 40/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8659 - loss: 0.3312
Epoch 41/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8650 - loss: 0.3241
Epoch 42/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8564 - loss: 0.3475
Epoch 43/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8604 - loss: 0.3286
Epoch 44/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8682 - loss: 0.3208
Epoch 45/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8537 - loss: 0.3402
Epoch 46/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8629 - loss: 0.3341
Epoch 47/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8620 - loss: 0.3321
Epoch 48/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8540 - loss: 0.3443
Epoch 49/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8569 - loss: 0.3421
Epoch 50/50
700/700 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8697 - loss: 0.3246
```

Out[26]:

```
<keras.src.callbacks.history.History at 0x7f248e49b500>
```

In [27]:

```python
y_pred =classifier.predict(x_test)
y_pred = (y_pred > 0.5)
```

```
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

In [28]:

```python
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:\n', cm)
```

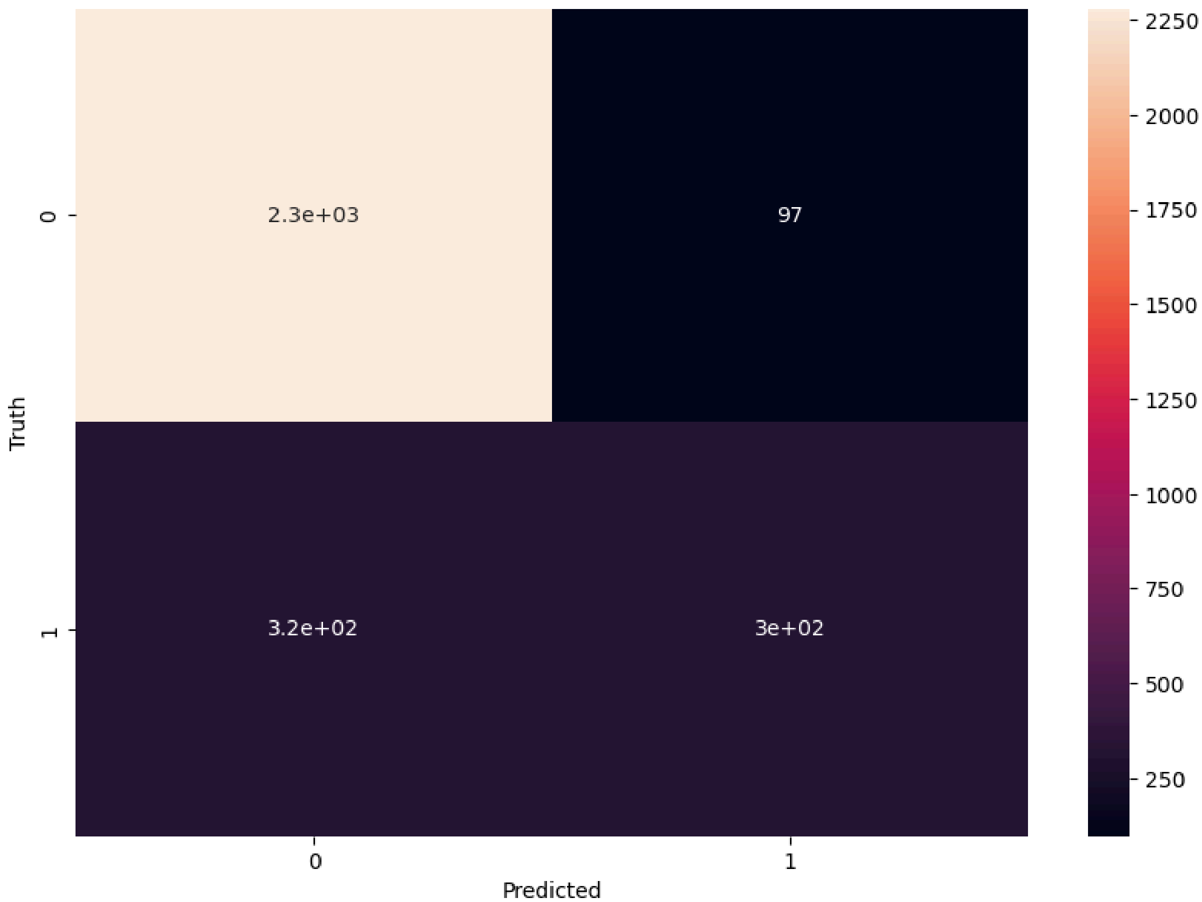
```
Confusion Matrix:
 [[2280   97]
 [ 320  303]]
```

In [29]:
```
acc = accuracy_score(y_test, y_pred)
print('Accuracy Score:', acc)
```

```
Accuracy Score: 0.861
```

In [30]:
```
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot = True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Out[30]:
```
Text(95.72222222222221, 0.5, 'Truth')
```



In [31]:
```
print('Classification Report:\n',classification_report(y_test, y_pred))
```

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

           0       0.88      0.96      0.92      2377
           1       0.76      0.49      0.59       623

    accuracy                           0.86      3000
   macro avg       0.82      0.72      0.75      3000
weighted avg       0.85      0.86      0.85      3000
```