72017912C
COMPBEB1326

GROUP B : ASSIGNMENT No. 3

Page No.
Date

Title : Neural Network Based Classification

Objective: To build a neural network based classifier that can determine whether a bank customer will leave or not in the next 6 months.

Problem Statement:

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, Credit Score, Geography, Gender, Age, Tenure, Balance, etc.

Perform the following steps:

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

Theory:

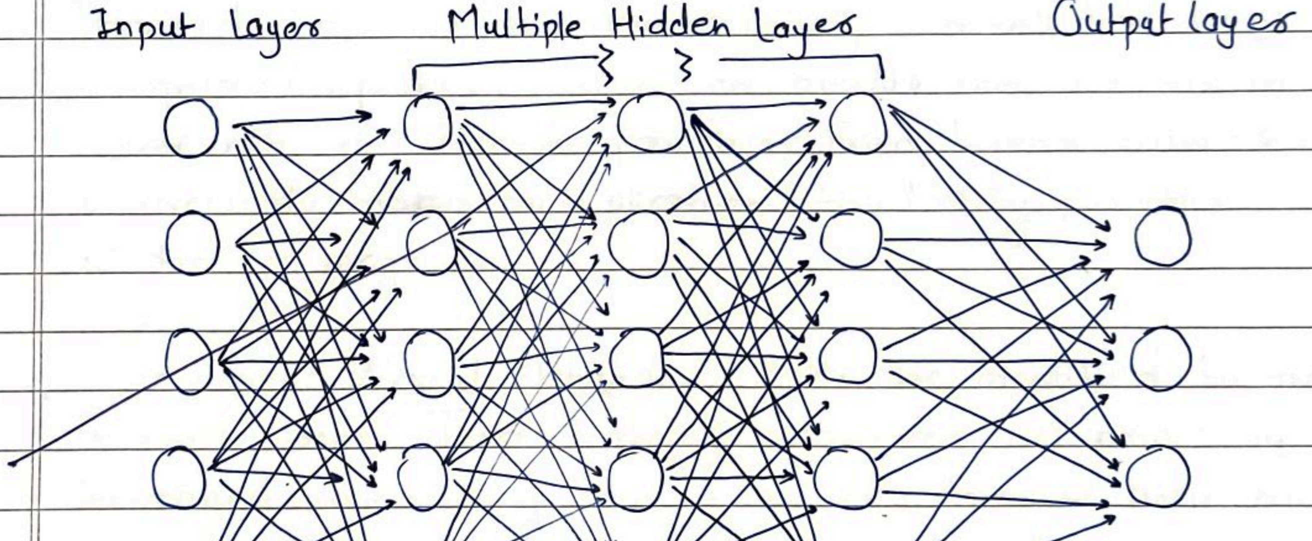
Neural Network :

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning & are at the heart of deep learning

Page No.
Date

algorithms. Their name & structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

Artificial neural networks (ANNs) are comprised of a node layers, containing an input layer, one or more hidden layers & an output layer. Each node, or artificial neuron, connects to another & has an associated weight & threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.



Neural networks rely on training data to learn & improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in CS & AI, allowing us to classify & cluster data at a high velocity.

Types of neural networks :

1> **Perceptron**: The perceptron is the oldest neural network, created by Frank Rosenblatt in 1958. It has a single neuron & is the simplest form of a neural network.

2> **Feedforward neural network**: FNN or multi-layer perceptron (MLPs), are ~~what we~~ comprised on an input layer, a hidden layer or layers & an output layer. They are actually comprised of sigmoid neurons, not perceptrons, as most of real-world problems are nonlinear.

3> **Convolutional Neural Network**: CNN are similar to Feed Forward networks, but they're usually utilized for image recognition, pattern recognition, and/or computer vision. These networks harness principles from linear algebra, particularly matrix multiplication, to identify patterns within an image.

4> **Recurrent Neural Network**: RNNs are identified by their feedback loops. These learning algorithms are primarily leveraged when using time-series data to make predictions about future outcomes, such as stock market predictions or sales forecasting.

Deep Neural Network:

Also known as a deep learning network, a deep neural network, at its most basic, is one that involves two or more processing layers. Deep neural networks rely on machine learning networks that continuously evolve by comparing estimated outcomes to actual results, then modifying future projections.

Applications of Neural Network:

Neural networks can be applied to a broad range of problems & can assess many different types of input, including images, videos, files, databases & more. They also do not require explicit programming to interpret the content of those inputs.

Some applications of neural networks today, include image/pattern recognition, self driving vehicle trajectory prediction, facial recognition, data mining, email spam filtering, medical diagnosis, & cancer research, etc.

Conclusion: Successfully built a neural network based classifier which can determine wheather a bank customer will leave or not in next 6 months.

Bambe
28/9/22

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt #Importing the libraries
```

In [2]:
```
df = pd.read_csv("Churn_Modelling.csv")
```

Preprocessing.

In [3]:
```
df.head()
```

Out[3]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance
0	1	15634602	Hargrave	619	France	Female	42	2	0.0C
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86
2	3	15619304	Onio	502	France	Female	42	8	159660.8C
3	4	15701354	Boni	699	France	Female	39	1	0.0C
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82

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

Out[4]:
```
(10000, 14)
```

In [5]:
```
df.describe()
```

Out[5]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	Nu
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	

In [6]:
```
df.isnull()
```

Out[6]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Bala
0	False	False	False	False	False	False	False	False	Fa
1	False	False	False	False	False	False	False	False	Fa
2	False	False	False	False	False	False	False	False	Fa
3	False	False	False	False	False	False	False	False	Fa
4	False	False	False	False	False	False	False	False	Fa
...	...	...	...	...	...	...	...	...	
9995	False	False	False	False	False	False	False	False	Fa
9996	False	False	False	False	False	False	False	False	Fa
9997	False	False	False	False	False	False	False	False	Fa
9998	False	False	False	False	False	False	False	False	Fa
9999	False	False	False	False	False	False	False	False	Fa

10000 rows × 14 columns

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
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 [8]:
```
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 [9]:
```
df.dtypes
```

Out[9]:
```
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 [10]:
```
df.columns
```

Out[10]:
```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

In [11]:
```
df = df.drop(['RowNumber', 'Surname', 'CustomerId'], axis= 1) #Dropping the unr
```

In [12]:
```
df.head()
```

Out[12]:

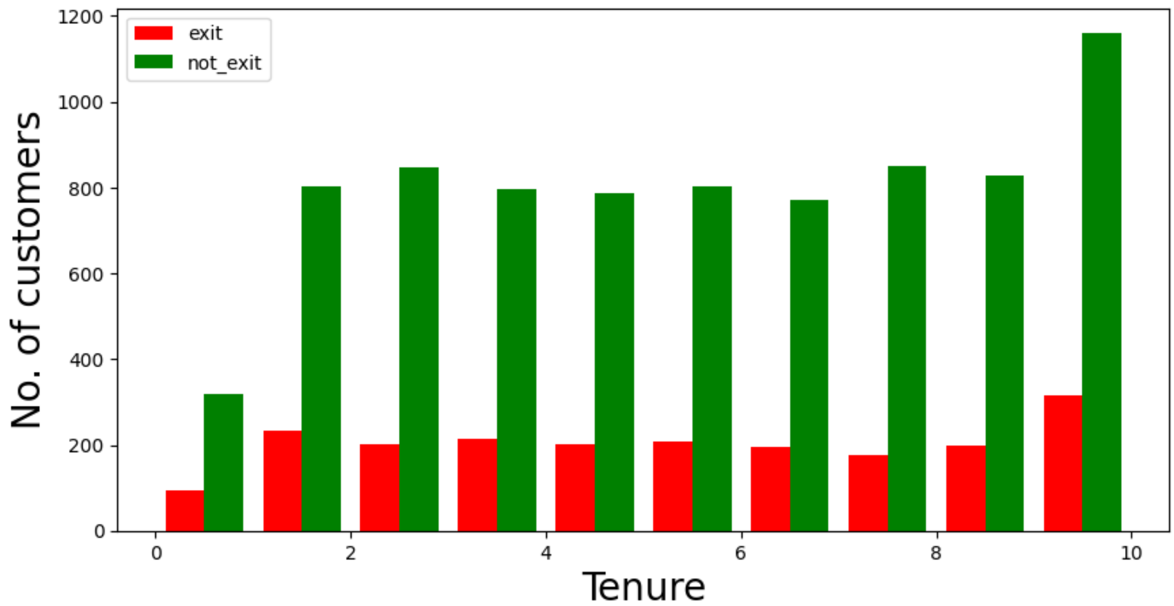
	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActi
0	619	France	Female	42	2	0.00	1	1	
1	608	Spain	Female	41	1	83807.86	1	0	
2	502	France	Female	42	8	159660.80	3	1	
3	699	France	Female	39	1	0.00	2	0	
4	850	Spain	Female	43	2	125510.82	1	1	

Visualization

In [13]:
```
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()
```

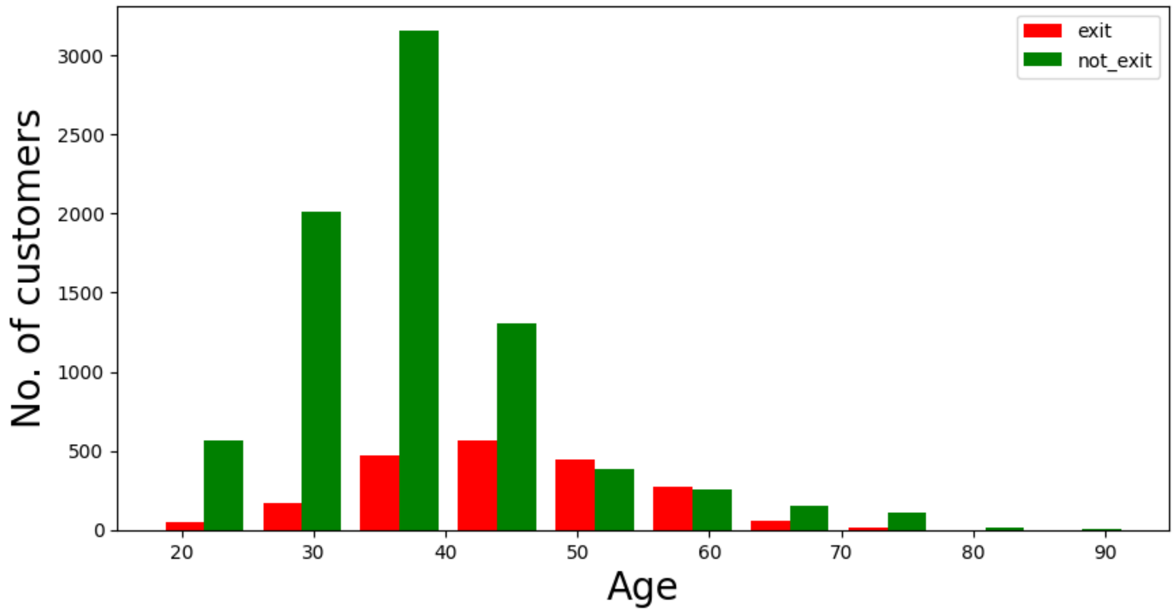
In [14]:
```
df_churn_exited = df[df['Exited']==1]['Tenure']
df_churn_not_exited = df[df['Exited']==0]['Tenure']
```

In [15]:
```
visualization(df_churn_exited, df_churn_not_exited, "Tenure")
```



In [16]:
```
df_churn_exited2 = df[df['Exited']==1]['Age']
df_churn_not_exited2 = df[df['Exited']==0]['Age']
```

In [17]:
```
visualization(df_churn_exited2, df_churn_not_exited2, "Age")
```



Converting the Categorical Variables

In [18]:
```
X = df[['CreditScore','Gender','Age','Tenure','Balance','NumOfProducts','HasCr(
states = pd.get_dummies(df['Geography'],drop_first = True)
gender = pd.get_dummies(df['Gender'],drop_first = True)
```

In [19]:
```

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

Splitting the training and testing Dataset

In [20]:
```
df.head()
```

Out[20]:

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

In [21]:
```
X = df[['CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsA
```

In [22]:
```
y = df['Exited']
```

In [23]:
```
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

In [24]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [25]:
```
X_train  = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [26]:
```
X_train
```

Out[26]:
```
array([[-1.15724477, -0.17938641, -1.39353121, ...,  0.90976714,
        -0.57976965, -0.5740511 ],
       [ 0.12062469, -0.08385195, -0.35465317, ...,  0.90976714,
        -0.57976965,  1.7420052 ],
       [ 0.1000139 , -0.75259313,  0.33793219, ..., -1.09918237,
        -0.57976965, -0.5740511 ],
       ...,
       [-1.55915532, -0.37045531,  1.7231029 , ...,  0.90976714,
        -0.57976965, -0.5740511 ],
       [-1.31182575,  2.11344047,  1.03051754, ...,  0.90976714,
        -0.57976965, -0.5740511 ],
       [-0.34311826, -0.08385195,  0.68422486, ...,  0.90976714,
        -0.57976965,  1.7420052 ]])
```

In [27]:
```
X_test
```

Out[27]:
```
array([[-0.04426169,  1.25363039,  1.03051754, ...,  0.90976714,
        -0.57976965, -0.5740511 ],
       [-0.827472  ,  0.29828586, -1.73982389, ...,  0.90976714,
         1.72482295, -0.5740511 ],
       [-1.16755016, -1.03919649, -0.70094585, ...,  0.90976714,
        -0.57976965, -0.5740511 ],
       ...,
       [ 1.72826692, -0.75259313, -1.39353121, ..., -1.09918237,
        -0.57976965, -0.5740511 ],
       [ 0.33703807, -0.08385195,  0.68422486, ...,  0.90976714,
        -0.57976965, -0.5740511 ],
       [-0.50800464, -1.80347212,  1.37681022, ..., -1.09918237,
        -0.57976965,  1.7420052 ]])
```

Building the Classifier Model using Keras

In [28]:
```
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 initiall
```

In [29]:
```
from keras.models import Sequential #To create sequential neural network
from keras.layers import Dense #To create hidden layers
```

In [30]:
```
classifier = Sequential()
```

In [31]:
```
#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_initia
```

In [32]:
```
classifier.add(Dense(activation = "relu",units = 6,kernel_initializer = "unifor
```

In [33]:
```
classifier.add(Dense(activation = "sigmoid",units = 1,kernel_initializer = "uni
```

In [34]:
```
classifier.compile(optimizer="adam",loss = 'binary_crossentropy',metrics = ['ac
```

In [35]:
```
classifier.summary() #3 layers created. 6 neurons in 1st,6neurons in 2nd layer
```

```
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
_________________________________________________________________
```

In [36]:
```
classifier.fit(X_train,y_train,batch_size=10,epochs=50) #Fitting the ANN to tra
```

```
Epoch 1/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4927 - acc
uracy: 0.7987
Epoch 2/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4185 - acc
uracy: 0.7989
Epoch 3/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4093 - acc
uracy: 0.7989
Epoch 4/50
700/700 [==============================] - 1s 980us/step - loss: 0.3997 - a
ccuracy: 0.8159
Epoch 5/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3914 - acc
uracy: 0.8297
Epoch 6/50
700/700 [==============================] - 1s 973us/step - loss: 0.3842 - a
ccuracy: 0.8319
Epoch 7/50
700/700 [==============================] - 1s 996us/step - loss: 0.3771 - a
ccuracy: 0.8347
Epoch 8/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3728 - acc
uracy: 0.8440
Epoch 9/50
700/700 [==============================] - 1s 979us/step - loss: 0.3687 - a
ccuracy: 0.8499
Epoch 10/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3652 - acc
uracy: 0.8519
Epoch 11/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3626 - acc
uracy: 0.8514
Epoch 12/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3600 - acc
uracy: 0.8531
Epoch 13/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3579 - acc
uracy: 0.8529
Epoch 14/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3560 - acc
uracy: 0.8551
Epoch 15/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3551 - acc
uracy: 0.8560
Epoch 16/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3534 - acc
uracy: 0.8566
Epoch 17/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3524 - acc
uracy: 0.8577
Epoch 18/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3525 - acc
uracy: 0.8557
Epoch 19/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3511 - acc
uracy: 0.8589
Epoch 20/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3507 - acc
```

```
uracy: 0.8569
Epoch 21/50
700/700 [==============================] - 1s 974us/step - loss: 0.3503 - a
ccuracy: 0.8584
Epoch 22/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3495 - acc
uracy: 0.8590
Epoch 23/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3487 - acc
uracy: 0.8576
Epoch 24/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3492 - acc
uracy: 0.8606
Epoch 25/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3491 - acc
uracy: 0.8593
Epoch 26/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3484 - acc
uracy: 0.8579
Epoch 27/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3473 - acc
uracy: 0.8593
Epoch 28/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3478 - acc
uracy: 0.8591
Epoch 29/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3472 - acc
uracy: 0.8594
Epoch 30/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3459 - acc
uracy: 0.8610
Epoch 31/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3462 - acc
uracy: 0.8614
Epoch 32/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3455 - acc
uracy: 0.8594
Epoch 33/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3461 - acc
uracy: 0.8593
Epoch 34/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3449 - acc
uracy: 0.8619
Epoch 35/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3451 - acc
uracy: 0.8579
Epoch 36/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3451 - acc
uracy: 0.8616
Epoch 37/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3442 - acc
uracy: 0.8606
Epoch 38/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3446 - acc
uracy: 0.8623
Epoch 39/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3437 - acc
uracy: 0.8591
Epoch 40/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3438 - acc
uracy: 0.8597
Epoch 41/50
```

```
700/700 [==============================] - 1s 1ms/step - loss: 0.3435 - acc
uracy: 0.8604
Epoch 42/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3432 - acc
uracy: 0.8606
Epoch 43/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3430 - acc
uracy: 0.8649
Epoch 44/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3429 - acc
uracy: 0.8630
Epoch 45/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3432 - acc
uracy: 0.8616
Epoch 46/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3424 - acc
uracy: 0.8616
Epoch 47/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3429 - acc
uracy: 0.8623
Epoch 48/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3431 - acc
uracy: 0.8624
Epoch 49/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3422 - acc
uracy: 0.8613
Epoch 50/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3415 - acc
uracy: 0.8611
```

Out[36]:
```
<keras.callbacks.History at 0x2a25e4eebe0>
```

In [37]:
```
y_pred =classifier.predict(X_test)
y_pred = (y_pred > 0.5) #Predicting the result
```

```
94/94 [==============================] - 0s 775us/step
```

In [38]:
```
from sklearn.metrics import confusion_matrix,accuracy_score,classification_repo
```

In [39]:
```
cm = confusion_matrix(y_test,y_pred)
```

In [40]:
```
cm
```

Out[40]:
```
array([[2275,   96],
       [ 341,  288]], dtype=int64)
```

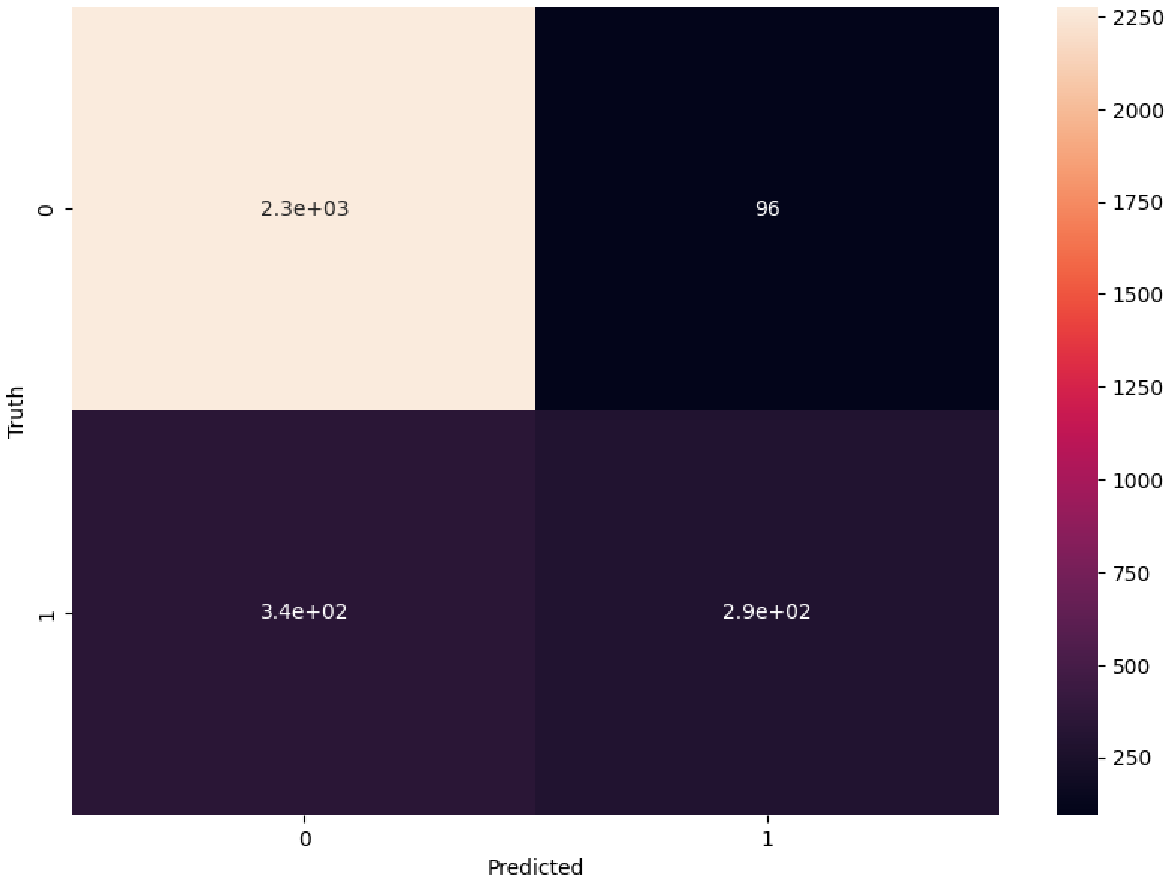
In [41]:
```
accuracy = accuracy_score(y_test,y_pred)
```

In [42]:
```
accuracy
```

Out[42]:
```
0.8543333333333333
```

In [43]:
```
plt.figure(figsize = (10,7))
sns.heatmap(cm,annot = True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Out[43]:
```
Text(95.72222222222221, 0.5, 'Truth')
```



In [44]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.96      0.91      2371
           1       0.75      0.46      0.57       629

    accuracy                           0.85      3000
   macro avg       0.81      0.71      0.74      3000
weighted avg       0.84      0.85      0.84      3000
```