ML LAB ASSIGNMENT 4 BC61

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).

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

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

```
# For dataframe operations and plots
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

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

```
df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1

```
# Gives basic data about dataframe like
# datatypes, null count , columsn names
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
```

```
# count of null values in 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
```

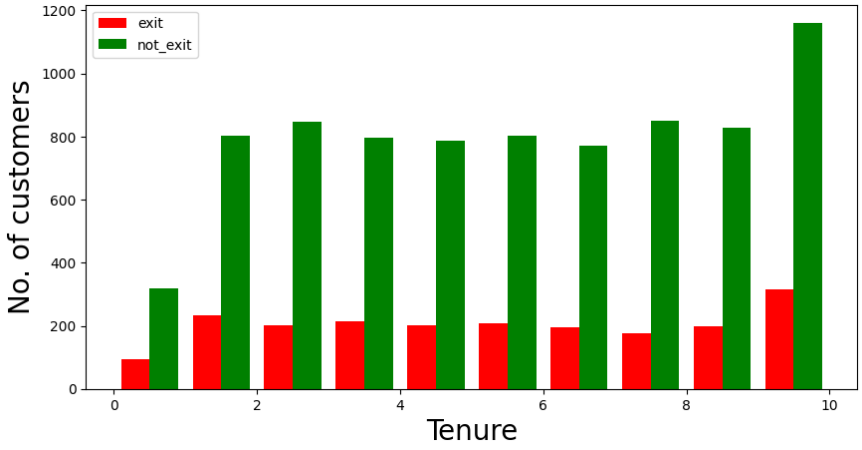
```
df.columns
```

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

```
# writing a function for plots
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()
```

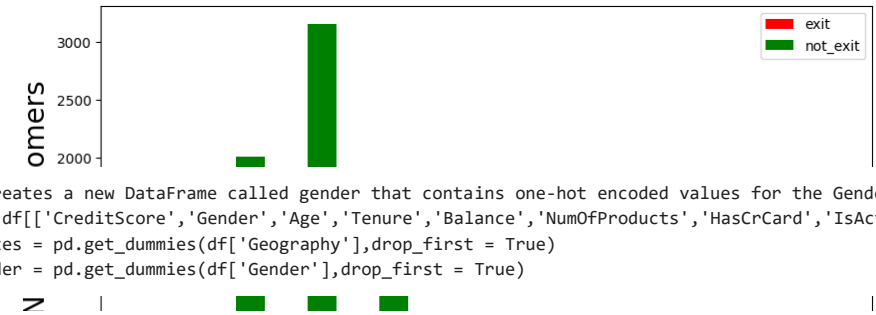
```
# create dataframes havingg tenure for exited=1&0
df_churn_exited = df[df['Exited']==1]['Tenure']
df_churn_not_exited = df[df['Exited']==0]['Tenure']
```

```
# plotting chart for exited,not_exited vs Tenure
visualization(df_churn_exited, df_churn_not_exited, "Tenure")
```




```
# doing the same with age vs exited or not
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')
```




```
# creates a new DataFrame called gender that contains one-hot encoded values for the Gender feature.
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)
```



```
gender
```

	Male
0	0
1	0
2	0
3	0
4	0
...	...
9995	1
9996	1
9997	0
9998	1
9999	0

10000 rows × 1 columns

```
# concating gender and status in df
df = pd.concat([df,gender,states], axis = 1)
df
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	
0	1	15634602	Hargrave	619	France	Female	42	2	
1	2	15647311	Hill	608	Spain	Female	41	1	
2	3	15619304	Onio	502	France	Female	42	8	1
3	4	15701354	Boni	699	France	Female	39	1	
4	5	15737888	Mitchell	850	Spain	Female	43	2	1
...	...	...	...	...	...	...	...	...	
9995	9996	15606229	Obijiaku	771	France	Male	39	5	
9996	9997	15569892	Johnstone	516	France	Male	35	10	
9997	9998	15584532	Liu	709	France	Female	36	7	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	
9999	10000	15628319	Walker	792	France	Female	28	4	1

10000 rows × 17 columns

```
# Creating new dataframes for training
X = df[['CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalary','Male','Germany','Spain']]
y=df[['Exited']]
```

```
# splitting data
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)
```

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

```
# standardizing the data
X_train  = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
import keras
```

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

```
classifier = Sequential()
```

```
classifier.add(Dense(activation = "relu",input_dim = 11,units = 6,kernel_initializer = "uniform"))
```

This line adds a dense layer to the model. The activation argument specifies the activation function to use for the layer. The input_dim argument specifies the number of inputs to the layer. The units argument specifies the number of neurons in the layer. The kernel_initializer argument specifies the initializer to use for the layer's weights.

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

```
classifier.add(Dense(activation = "sigmoid",units = 1,kernel_initializer = "uniform"))
```

```
classifier.compile(optimizer="adam",loss = 'binary_crossentropy',metrics = ['accuracy'])
```

This line compiles the model. The optimizer argument specifies the optimizer to use to train the model. The loss argument specifies the loss function to use to train the model. The metrics argument specifies the metrics to evaluate the model on.

```
classifier.summary()
```

```
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 (484.00 Byte)
Trainable params: 121 (484.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Training the model
classifier.fit(X_train,y_train,batch_size=10,epochs=50)
```

```
Epoch 1/50
700/700 [==============================] - 2s 2ms/step - loss: 0.4876 - accuracy: 0.8004
Epoch 2/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4255 - accuracy: 0.8006
Epoch 3/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4210 - accuracy: 0.8006
Epoch 4/50
700/700 [==============================] - 2s 2ms/step - loss: 0.4168 - accuracy: 0.8090
Epoch 5/50
700/700 [==============================] - 2s 2ms/step - loss: 0.4137 - accuracy: 0.8259
Epoch 6/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4118 - accuracy: 0.8317
Epoch 7/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4102 - accuracy: 0.8327
Epoch 8/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4092 - accuracy: 0.8357
Epoch 9/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4082 - accuracy: 0.8354
Epoch 10/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4076 - accuracy: 0.8336
Epoch 11/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4070 - accuracy: 0.8367
Epoch 12/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4058 - accuracy: 0.8373
Epoch 13/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4056 - accuracy: 0.8383
Epoch 14/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4051 - accuracy: 0.8373
Epoch 15/50
700/700 [==============================] - 2s 2ms/step - loss: 0.4048 - accuracy: 0.8393
```

```
Epoch 16/50
700/700 [==============================] - 2s 2ms/step - loss: 0.4040 - accuracy: 0.8364
Epoch 17/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4036 - accuracy: 0.8376
Epoch 18/50
700/700 [==============================] - 2s 2ms/step - loss: 0.4026 - accuracy: 0.8380
Epoch 19/50
700/700 [==============================] - 2s 3ms/step - loss: 0.4022 - accuracy: 0.8369
Epoch 20/50
700/700 [==============================] - 3s 4ms/step - loss: 0.4019 - accuracy: 0.8386
Epoch 21/50
700/700 [==============================] - 2s 3ms/step - loss: 0.4020 - accuracy: 0.8386
Epoch 22/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4019 - accuracy: 0.8381
Epoch 23/50
700/700 [==============================] - 2s 2ms/step - loss: 0.4017 - accuracy: 0.8389
Epoch 24/50
700/700 [==============================] - 3s 4ms/step - loss: 0.4010 - accuracy: 0.8386
Epoch 25/50
700/700 [==============================] - 2s 3ms/step - loss: 0.4012 - accuracy: 0.8397
Epoch 26/50
700/700 [==============================] - 2s 3ms/step - loss: 0.4009 - accuracy: 0.8376
Epoch 27/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4006 - accuracy: 0.8384
Epoch 28/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4002 - accuracy: 0.8386
Epoch 29/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4002 - accuracy: 0.8381
```

```
# Predicting the values
y_pred =classifier.predict(X_test)
print(y_pred)
print(y_pred.shape)
y_pred = (y_pred > 0.5)
print(y_pred)
```

```
94/94 [==============================] - 0s 1ms/step
[[0.39574802]
 [0.68484145]
 [0.37360382]
 ...
 [0.19618267]
 [0.73893285]
 [0.37226972]]
(3000, 1)
[[False]
 [ True]
 [False]
 ...
 [False]
 [ True]
 [False]]
```

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

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

```
cm
```

```
array([[2282,   77],
       [ 433,  208]])
```

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

```
accuracy
```

```
0.83
```

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

```
Text(95.72222222222221, 0.5, 'Truth')
```


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

```
                  precision    recall  f1-score   support

               0       0.84      0.97      0.90      2359
               1       0.73      0.32      0.45       641

        accuracy                           0.83      3000
       macro avg       0.79      0.65      0.67      3000
    weighted avg       0.82      0.83      0.80      3000
```

```
# predicting for new input of customer data
new_customer = [[0, 0, 600, 1, 40, 3, 60000, 2, 1, 1, 50000]]
```

```
new_customer = sc.transform(sc.transform(new_customer))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler wa
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler wa
  warnings.warn(
```

```
new_prediction = classifier.predict(new_customer)
new_prediction
```

```
1/1 [==============================] - 0s 20ms/step
array([[0.]], dtype=float32)
```

```
new_prediction = (new_prediction > 0.5)
new_prediction
```

```
array([[False]])
```