ML Practical : 03

Title: 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:

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

```
import numpy as np
np.set_printoptions(threshold=np.inf)
import matplotlib.pyplot as plt
import pandas as pd
```

```
# Importing the dataset
dataset = pd.read_csv('/content/Churn_Modelling.csv')
```

```
dataset.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Ba
0	1	15634602	Hargrave	619	France	Female	42	2	
1	2	15647311	Hill	608	Spain	Female	41	1	838
2	3	15619304	Onio	502	France	Female	42	8	1596
3	4	15701354	Boni	699	France	Female	39	1	
4	5	15737888	Mitchell	850	Spain	Female	43	2	1255

```
dataset.tail()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure
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

```
dataset.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
```

```
dataset.shape
```

```
(10000, 14)
```

```
dataset.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
```

```
dataset.describe()
```

	RowNumber	CustomerId	CreditScore	Age	Tenure	Bala
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090

```
X = dataset.iloc[:, 3:13].values
y = dataset.iloc[:, 13].values


print(X[:10,:], '\n')

print(y[:10])
```

```
[[619 'France' 'Female' 42 2 0.0 1 1 1 101348.88]
 [608 'Spain' 'Female' 41 1 83807.86 1 0 1 112542.58]
 [502 'France' 'Female' 42 8 159660.8 3 1 0 113931.57]
 [699 'France' 'Female' 39 1 0.0 2 0 0 93826.63]
 [850 'Spain' 'Female' 43 2 125510.82 1 1 1 79084.1]
 [645 'Spain' 'Male' 44 8 113755.78 2 1 0 149756.71]
 [822 'France' 'Male' 50 7 0.0 2 1 1 10062.8]
 [376 'Germany' 'Female' 29 4 115046.74 4 1 0 119346.88]
 [501 'France' 'Male' 44 4 142051.07 2 0 1 74940.5]
 [684 'France' 'Male' 27 2 134603.88 1 1 1 71725.73]]

[1 0 1 0 0 1 0 1 0 0]
```

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
# Country
labelencoder_X = LabelEncoder()
X[:, 1] = labelencoder_X.fit_transform(X[:, 1])

# Gender
labelencoder_X_2 = LabelEncoder()
X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])

# Giving ordinal feature to our variables
onehotencoder = ColumnTransformer([("Country", OneHotEncoder(), [1])], remainder = 'passth
X = onehotencoder.fit_transform(X)
X = X[:, 1:]
```

```
print(X[:10,:], '\n')
print(y[:10])
```

```
[[0.0 0.0 619 0 42 2 0.0 1 1 1 101348.88]
 [0.0 1.0 608 0 41 1 83807.86 1 0 1 112542.58]
 [0.0 0.0 502 0 42 8 159660.8 3 1 0 113931.57]
 [0.0 0.0 699 0 39 1 0.0 2 0 0 93826.63]
 [0.0 1.0 850 0 43 2 125510.82 1 1 1 79084.1]
 [0.0 1.0 645 1 44 8 113755.78 2 1 0 149756.71]
 [0.0 0.0 822 1 50 7 0.0 2 1 1 10062.8]
 [1.0 0.0 376 0 29 4 115046.74 4 1 0 119346.88]
 [0.0 0.0 501 1 44 4 142051.07 2 0 1 74940.5]
 [0.0 0.0 684 1 27 2 134603.88 1 1 1 71725.73]]

[1 0 1 0 0 1 0 1 0 0]
```

```
X.shape
```

```
(10000, 11)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
import keras
import sys
from keras.models import Sequential #to initialize NN
from keras.layers import Dense #used to create layers in NN
from keras.layers import Dropout
```

```
classifier = Sequential()
```

```
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu', input
classifier.add(Dropout(rate=0.1))
```

```
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))
classifier.add(Dropout(rate=0.1))
```

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

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

```
classifier.fit(X_train, y_train, batch_size = 10, epochs = 100  )
```

```
Epoch 1/100
800/800 [==============================] - 3s 2ms/step - loss: 0.4854 - accuracy:
Epoch 2/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4350 - accuracy:
Epoch 3/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4279 - accuracy:
Epoch 4/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4274 - accuracy:
Epoch 5/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4272 - accuracy:
Epoch 6/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4258 - accuracy:
Epoch 7/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4244 - accuracy:
Epoch 8/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4237 - accuracy:
Epoch 9/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4239 - accuracy:
Epoch 10/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4224 - accuracy:
Epoch 11/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4202 - accuracy:
Epoch 12/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4201 - accuracy:
Epoch 13/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4207 - accuracy:
Epoch 14/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4199 - accuracy:
Epoch 15/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4200 - accuracy:
Epoch 16/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4216 - accuracy:
Epoch 17/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4234 - accuracy:
Epoch 18/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4204 - accuracy:
Epoch 19/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4194 - accuracy:
Epoch 20/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4200 - accuracy:
Epoch 21/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4191 - accuracy:
Epoch 22/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4174 - accuracy:
Epoch 23/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4227 - accuracy:
Epoch 24/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4189 - accuracy:
Epoch 25/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4190 - accuracy:
Epoch 26/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4200 - accuracy:
Epoch 27/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4189 - accuracy:
Epoch 28/100
800/800 [==============================] - 2s 2ms/step - loss: 0.4198 - accuracy:
Epoch 29/100
```

```
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)

y_pred
```

```
63/63 [==============================] - 0s 1ms/step
array([[False],
       [False],
       [False],
       [False],
       [False],
       [ True],
       [False],
       [False],
       [False],
       [ True],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [ True],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [ True],
       [False],
       [False],
       [False],
       [False],
       [False],
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

```
cm
```

```
array([[1546,   49],
       [ 274,  131]])
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_scor
```

```
print("F1 Score: ",f1_score(y_test,y_pred))
```

```
F1 Score:  0.4478632478632479
```

```
print("Accuracy: ",accuracy_score(y_test,y_pred))
```

```
Accuracy:  0.8385
```

```
print("Precision: ",precision_score(y_test,y_pred))
print("Recall: ",recall_score(y_test,y_pred))
```

```
Precision:  0.7277777777777777
Recall:  0.3234567901234568
```

```
print("Error Rate: ",1-accuracy_score(y_test,y_pred))
```

```
Error Rate:  0.16149999999999998
```

```
"""Predict if the customer with the following informations will leave the bank:
Geography: France
Credit Score: 600
Gender: Male
Age: 40
Tenure: 3
Balance: 60000
Number of Products: 2
Has Credit Card: Yes
Is Active Member: Yes
Estimated Salary: 50000"""

new_prediction = classifier.predict(sc.transform(np.array([[0.0, 0, 600, 1, 40, 3, 60000,
new_prediction = (new_prediction > 0.5)
```

```
new_prediction
```

```
1/1 [==============================] - 0s 19ms/step
array([[False]])
```

```
import keras
import sys
from keras.models import Sequential #to initialize NN
from keras.layers import Dense #used to create layers in NN

#Evaluating the ANN
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score

def build_classifier():
    classifier = Sequential()
    classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu', i
    classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))
    classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'
    classifier.compile(optimizer = 'adam',loss= "binary_crossentropy",metrics=["accuracy"]

#fit our model to the traing data using KerasClassifier
classifier = KerasClassifier(build_fn = build_classifier,  batch_size = 10, epochs = 100

#estimator - object to fit the data
#X - data to fit
#y - Target variable to try to predict
#cv - number of train test folds
#n_jobs - number of CPUs to use to do the computation. -1 means 'all CPUs'
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10, n_
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:18: DeprecationWarning:
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.py:372: F
10 fits failed out of a total of 10.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score

Below are more details about the failures:
--------------------------------------------------------------------------------
10 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.p
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.7/dist-packages/keras/wrappers/scikit_learn.py", line
    return super(KerasClassifier, self).fit(x, y, **kwargs)
  File "/usr/local/lib/python3.7/dist-packages/keras/wrappers/scikit_learn.py", line
    if (losses.is_categorical_crossentropy(self.model.loss) and
AttributeError: 'NoneType' object has no attribute 'loss'

  warnings.warn(some_fits_failed_message, FitFailedWarning)
```