Assignment No. 3

Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months.

Step 1: Download the Dataset

1. Dataset link: [bank-customer-churn-modeling]
2. Go to the Kaggle dataset page.
3. Download bank-customer-churn-modeling.csv file.
4. Save the dataset in the directory where you will run the Jupyter notebook.

Step 2: Open Jupyter Notebook

1. Open Jupyter Notebook:
    - Launch Jupyter Notebook.
    - Navigate to the directory where you saved the bank-customer-churn-modeling.csv file.
    - Create a new Python notebook.

Step 3: Import Necessary Libraries

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
from keras.models import Sequential
from keras.layers import Dense
import io
```

- **Explanation**: Import essential libraries for data manipulation (numpy, pandas), preprocessing (LabelEncoder, StandardScaler), model evaluation (confusion_matrix, accuracy_score), and building the neural network (Sequential, Dense).

Step 3: Upload Dataset

```
from google.colab import files
uploaded = files.upload()
```

- **Explanation**: Use files.upload() to upload the dataset from your local machine to Google Colab.

Step 4. Load Dataset

```
dataset = pd.read_csv(io.StringIO(uploaded['Churn_Modelling.csv'].decode('utf-8')))
```

- **Explanation**: Read the uploaded CSV file into a pandas DataFrame.

Step 5. Explore the Dataset

```
dataset.head()
```

- **Explanation**: Display the first few rows of the dataset to understand its structure and features.

Step 6. Data Preprocessing

```
# Select necessary features and target variable
X = dataset.iloc[:, 3:13].values  # Features
y = dataset.iloc[:, 13].values  # Target variable (Exited)
```

- **Explanation**: Extract features (independent variables) and the target variable (whether the customer left the bank).

Step 7. Encode Categorical Variables

```
# Encode categorical data (Country, Gender)
labelencoder_X_1 = LabelEncoder()
X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])  # Encode Geography
labelencoder_X_2 = LabelEncoder()
X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])  # Encode Gender
```

- **Explanation**: Convert categorical string values (Country and Gender) into numerical labels using LabelEncoder.

Step 8. One-Hot Encoding for Geography

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

# Apply OneHotEncoder to the 'Geography' column (index 1)
ct = ColumnTransformer([("Geography", OneHotEncoder(), [1])], remainder='passthrough')
# Transform the dataset, encoding 'Geography' as one-hot vectors
X = ct.fit_transform(X)

# Avoid the dummy variable trap by removing the first one-hot encoded column
X = X[:, 1:]
```

- **Explanation**: Use OneHotEncoder to convert the geographical information into binary (dummy) variables. Remove the first dummy variable to avoid multicollinearity.

Step 9. Split Dataset into Training and Testing Sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

- **Explanation**: Split the dataset into training (80%) and testing (20%) sets to evaluate the model's performance.

Step 10. Feature Scaling

```
# Initialize the StandardScaler to perform feature scaling
sc = StandardScaler()

# Fit the scaler to the training data and transform it
X_train = sc.fit_transform(X_train)

# Apply the same transformation to the test data
X_test = sc.transform(X_test)
```

- **Explanation**: Scale the features to have zero mean and unit variance using StandardScaler, which helps in speeding up convergence during training.

Step 11. Building the Neural Network

```
# Initialize the neural network
classifier = Sequential()

# Add the input layer (11 features) and the first hidden layer with 6 neurons
classifier.add(Dense(units=6, activation='relu', input_dim=11))

# Add the second hidden layer with 6 neurons and ReLU activation
classifier.add(Dense(units=6, activation='relu'))

# Add the output layer with 1 neuron for binary classification, using sigmoid activation
classifier.add(Dense(units=1, activation='sigmoid'))
```

- **Explanation**: Construct a feedforward neural network:
  - **Input Layer**: 11 input features.
  - **Hidden Layers**: Two hidden layers with 6 neurons each, using the ReLU activation function.
  - **Output Layer**: A single neuron for binary classification (churn or not) with a sigmoid activation function.

Step 12. Compile the Model

```
# Compile the ANN with Adam optimizer, binary crossentropy loss, and accuracy metric
classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

- **Explanation**: Compile the model using the Adam optimizer and binary crossentropy loss function, tracking accuracy as a metric.

Step 13. Train the Model

```
# Train the model using the training data with 100 epochs
classifier.fit(X_train, y_train, epochs=100)
```

- **Explanation**: Fit the model to the training data for 100 epochs, allowing the network to learn the patterns.

Step 14. Evaluate Model Performance

```
# Predict the results for the test set
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)  # Convert probabilities to binary (0 or 1)

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

- **Explanation**:
  - Make predictions on the test set, converting probabilities to binary outcomes (0 or 1) using a threshold of 0.5.
  - Generate a confusion matrix to assess model performance.
  - Calculate and print the accuracy of the model.