IMDB Movie Review Classification

This Python script classifies movie reviews from the IMDB dataset as positive or negative using a binary classification approach. The script includes steps to preprocess the text data, build a neural network model, and evaluate its performance.

# 1. Import Necessary Libraries

import tensorflow as tf  
import numpy as np  
from tensorflow import keras  
from tensorflow.keras import layers  
from tensorflow.keras.utils import to\_categorical  
import matplotlib.pyplot as plt

# 2. Load the IMDB Dataset

imdb = tf.keras.datasets.imdb  
(train\_data, train\_labels), (test\_data, test\_labels) = imdb.load\_data(num\_words=10000)  
# num\_words=10000 restricts the dataset to the 10,000 most frequent words.

# 3. Data Preprocessing: Vectorizing the Sequences

def vectorize\_sequences(sequences, dimension=10000):  
 results = np.zeros((len(sequences), dimension))  
 for i, sequence in enumerate(sequences):  
 for j in sequence:  
 results[i, j] = 1.  
 return results  
  
x\_train = vectorize\_sequences(train\_data)  
x\_test = vectorize\_sequences(test\_data)

# 4. Convert Labels to Float32 Format

y\_train = np.array(train\_labels).astype("float32")  
y\_test = np.array(test\_labels).astype("float32")

# 5. Build the Neural Network Model

model = keras.Sequential([  
 layers.Dense(16, activation="relu", input\_shape=(10000,)),  
 layers.Dense(16, activation="relu"),  
 layers.Dense(1, activation="sigmoid")  
])

# 6. Compile the Model

model.compile(optimizer="rmsprop",  
 loss="binary\_crossentropy",  
 metrics=["accuracy"])

# 7. Prepare the Validation Set

x\_val = x\_train[:10000]  
partial\_x\_train = x\_train[10000:]  
y\_val = y\_train[:10000]  
partial\_y\_train = y\_train[10000:]

# 8. Train the Model

history = model.fit(partial\_x\_train,  
 partial\_y\_train,  
 epochs=20,  
 batch\_size=512,  
 validation\_data=(x\_val, y\_val))

# 9. Evaluate the Model

results = model.evaluate(x\_test, y\_test)  
print(f"Test Loss: {results[0]}, Test Accuracy: {results[1]}")

# 10. Visualize Training and Validation Performance

history\_dict = history.history  
loss\_values = history\_dict['loss']  
val\_loss\_values = history\_dict['val\_loss']  
epochs = range(1, len(loss\_values) + 1)  
  
# Plot the loss  
plt.plot(epochs, loss\_values, 'bo', label='Training loss')  
plt.plot(epochs, val\_loss\_values, 'b', label='Validation loss')  
plt.title('Training and validation loss')  
plt.xlabel('Epochs')  
plt.ylabel('Loss')  
plt.legend()  
plt.show()  
  
# Plot the accuracy  
acc\_values = history\_dict['accuracy']  
val\_acc\_values = history\_dict['val\_accuracy']  
plt.plot(epochs, acc\_values, 'bo', label='Training accuracy')  
plt.plot(epochs, val\_acc\_values, 'b', label='Validation accuracy')  
plt.title('Training and validation accuracy')  
plt.xlabel('Epochs')  
plt.ylabel('Accuracy')  
plt.legend()  
plt.show()