TASK#04

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

import tensorflow as tf

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional

import tensorflow\_datasets as tfds

# Load the dataset

dataset, info = tfds.load("imdb\_reviews", with\_info=True, as\_supervised=True)

# Split the dataset

train\_data, test\_data = dataset['train'], dataset['test']

# Initialize tokenizer

tokenizer = Tokenizer(num\_words=10000, oov\_token='<OOV>')

train\_sentences = []

train\_labels = []

# Extract sentences and labels from the dataset

for sentence, label in train\_data:

train\_sentences.append(str(sentence.numpy()))

train\_labels.append(label.numpy())

# Fit the tokenizer on the training data

tokenizer.fit\_on\_texts(train\_sentences)

# Tokenize and pad the sequences

train\_sequences = tokenizer.texts\_to\_sequences(train\_sentences)

train\_padded = pad\_sequences(train\_sequences, padding='post', maxlen=120)

# Convert labels to numpy array

train\_labels = np.array(train\_labels)

# Prepare test data

test\_sentences = []

test\_labels = []

for sentence, label in test\_data:

test\_sentences.append(str(sentence.numpy()))

test\_labels.append(label.numpy())

test\_sequences = tokenizer.texts\_to\_sequences(test\_sentences)

test\_padded = pad\_sequences(test\_sequences, padding='post', maxlen=120)

test\_labels = np.array(test\_labels)

# Build the model

model = Sequential([

Embedding(10000, 64, input\_length=120),

Bidirectional(LSTM(64, return\_sequences=True)),

Bidirectional(LSTM(64)),

Dense(64, activation='relu'),

Dropout(0.5),

Dense(1, activation='sigmoid')

])

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

model.summary()

# Train the model

history = model.fit(train\_padded, train\_labels, epochs=10, validation\_data=(test\_padded, test\_labels), verbose=1)

# Evaluate the model

test\_loss, test\_acc = model.evaluate(test\_padded, test\_labels)

print(f"Test Accuracy: {test\_acc:.4f}")

# Plot training history

def plot\_history(history):

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend(loc='lower right')

plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend(loc='upper right')

plt.title('Training and Validation Loss')

plt.show()

plot\_history(history)

# Make predictions

def predict\_sentiment(text):

sequence = tokenizer.texts\_to\_sequences([text])

padded\_sequence = pad\_sequences(sequence, maxlen=120, padding='post')

prediction = model.predict(padded\_sequence)[0][0]

sentiment = "Positive" if prediction >= 0.5 else "Negative"

return sentiment, prediction

# Example prediction

text = "The movie was fantastic and I enjoyed it a lot!"

sentiment, score = predict\_sentiment(text)

print(f"Sentiment: {sentiment}, Score: {score:.4f}")