**Q4: Sentiment Classification Using RNN**

**Task:** Sentiment analysis determines if a given text expresses a positive or negative emotion. You will train an **LSTM-based sentiment classifier** using the IMDB dataset.

1. Load the **IMDB sentiment dataset** (tensorflow.keras.datasets.imdb).
2. Preprocess the text data by **tokenization** and **padding** sequences.
3. Train an **LSTM-based model** to classify reviews as **positive or negative**.
4. Generate a **confusion matrix** and classification report (accuracy, precision, recall, F1-score).
5. Interpret why **precision-recall tradeoff** is important in sentiment classification.

***Hint:*** *Use confusion\_matrix and classification\_report from sklearn.metrics.*

**1. Load the IMDB Sentiment Dataset**

The IMDB dataset is available in tensorflow.keras.datasets. It comprises 50,000 movie reviews, split equally into training and test sets, with labels indicating positive or negative sentiment.​

from tensorflow.keras.datasets import imdb

# Load the dataset, keeping only the top 10,000 most frequently occurring words

num\_words = 10000

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=num\_words)

**2. Preprocess the Text Data**

Since the reviews are already tokenized as sequences of word indices, the main preprocessing step is to pad these sequences to ensure uniform input length.

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

maxlen = 200 # Maximum review length

x\_train = pad\_sequences(x\_train, maxlen=maxlen)

x\_test = pad\_sequences(x\_test, maxlen=maxlen)

**3. Build and Train the LSTM Model**

Define an LSTM-based model for binary classification:

from tensorflow.keras.models import Sequential

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

embedding\_dim = 128

model = Sequential([

Embedding(input\_dim=num\_words, output\_dim=embedding\_dim, input\_length=maxlen),

LSTM(units=64, dropout=0.2, recurrent\_dropout=0.2),

Dense(1, activation='sigmoid')

])

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

Train the model:

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Embedding, LSTM, Dense

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

from tensorflow.keras.datasets import imdb

# Load the IMDB dataset

num\_words = 10000

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=num\_words)

# Preprocess the text data by padding sequences

maxlen = 200

x\_train = pad\_sequences(x\_train, maxlen=maxlen)

x\_test = pad\_sequences(x\_test, maxlen=maxlen)

# Define the input layer with the shape of the input data

input\_layer = Input(shape=(maxlen,))

# Define the embedding layer without the input\_length parameter

embedding\_layer = Embedding(input\_dim=num\_words, output\_dim=128)(input\_layer)

# Define the LSTM layer

lstm\_layer = LSTM(units=64, dropout=0.2, recurrent\_dropout=0.2)(embedding\_layer)

# Define the output layer

output\_layer = Dense(1, activation='sigmoid')(lstm\_layer)

# Create the model

model = Model(inputs=input\_layer, outputs=output\_layer)

# Compile the model

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

# Summary of the model

model.summary()

**4. Evaluate the Model**

After training, evaluate the model using a confusion matrix and classification report:

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

import numpy as np

# Predict classes

y\_pred = (model.predict(x\_test) > 0.5).astype("int32")

# Confusion Matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:\n", conf\_matrix)

# Classification Report

class\_report = classification\_report(y\_test, y\_pred, target\_names=['Negative', 'Positive'])

print("Classification Report:\n", class\_report)

**5. Understanding the Precision-Recall Tradeoff**

In sentiment classification, the precision-recall tradeoff is crucial:​

* **Precision** measures the accuracy of positive predictions. High precision indicates that when the model predicts a review as positive, it is likely correct.​
* **Recall** measures the ability of the model to identify all positive instances. High recall means the model captures most of the actual positive reviews.

Balancing precision and recall is essential. For instance:​

* A model with high precision but low recall is conservative in predicting positive reviews, missing many actual positives.​
* A model with high recall but low precision predicts many reviews as positive, including many negatives, leading to false positives.​

Depending on the application, you might prioritize one over the other. For example, in a scenario where falsely classifying a negative review as positive is costly, you would aim for higher precision. Conversely, if missing a positive review is more detrimental, higher recall is preferred.​

By following these steps, you can build an LSTM-based sentiment classifier for the IMDB dataset, evaluate its performance, and understand the importance of the precision-recall tradeoff in sentiment analysis.