You are comparing GRU and LSTM models for a sequence

prediction task in natural language processing. How would you evaluate

which is more suitable for your task?

import pandas as pd

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

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 GRU, LSTM, Embedding, Dense

import matplotlib.pyplot as plt

# Load the dataset (replace with your file path)

data = pd.read\_csv('train.csv') # Adjust to your actual file path

# Extract text (X) and labels (y)

X = data['Title'] # Use the 'Title' column for text data

y = data['Class Index'] # Use the 'Class Index' column for labels

# Split the data into training and test sets (80% training, 20% testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 1: Tokenize the text data (convert words into integers)

tokenizer = Tokenizer(num\_words=5000) # Limit to 5000 words in the vocabulary

tokenizer.fit\_on\_texts(X\_train) # Fit the tokenizer on the training data

# Step 2: Convert text to sequences (convert each title into a sequence of tokens)

X\_train\_seq = tokenizer.texts\_to\_sequences(X\_train)

X\_test\_seq = tokenizer.texts\_to\_sequences(X\_test)

# Step 3: Pad the sequences to ensure they have the same length

max\_sequence\_length = 100 # Set maximum sequence length

X\_train\_pad = pad\_sequences(X\_train\_seq, maxlen=max\_sequence\_length)

X\_test\_pad = pad\_sequences(X\_test\_seq, maxlen=max\_sequence\_length)

# Now X\_train\_pad is a 2D array where each row represents a sequence of tokens

# Example: X\_train\_pad.shape should give you (number\_of\_samples, sequence\_length)

# Step 4: Define model parameters

input\_dim = 5000 # Vocabulary size (set to 5000 based on the tokenizer)

output\_dim = 128 # Embedding output dimension

input\_length = X\_train\_pad.shape[1] # The length of each padded sequence

# Build a simple GRU model

gru\_model = Sequential([

Embedding(input\_dim=input\_dim,

output\_dim=output\_dim, input\_length=input\_length),

GRU(64, return\_sequences=False),

Dense(1, activation='sigmoid')

])

# Compile the GRU model

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

# Step 5: Train the GRU model and capture the training history

gru\_history = gru\_model.fit(X\_train\_pad, y\_train, epochs=5, batch\_size=32, validation\_data=(X\_test\_pad, y\_test))

# Evaluate the GRU model on the test set

gru\_accuracy = gru\_model.evaluate(X\_test\_pad, y\_test)

print("GRU Model Accuracy:", gru\_accuracy)

# You can now define and train the LSTM model similarly

# Build a simple LSTM model

lstm\_model = Sequential([

Embedding(input\_dim=input\_dim, output\_dim=output\_dim, input\_length=input\_length),

LSTM(64, return\_sequences=False),

Dense(1, activation='sigmoid')

])

# Compile the LSTM model

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

# Train the LSTM model and capture the training history

lstm\_history = lstm\_model.fit(X\_train\_pad, y\_train, epochs=5, batch\_size=32, validation\_data=(X\_test\_pad, y\_test))

# Evaluate the LSTM model on the test set

lstm\_accuracy = lstm\_model.evaluate(X\_test\_pad, y\_test)

print("LSTM Model Accuracy:", lstm\_accuracy)

# Step 6: Plot the training and validation accuracy for both models

def plot\_history(history, title):

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

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

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.title(title)

plt.legend()

plt.show()

# Plot histories

plot\_history(gru\_history, "GRU Model")

plot\_history(lstm\_history, "LSTM Model")

**OUTPUT :**



