

Câu 1 : sinh dữ liệu comment cho 500 user, mỗi người 50 comments về các chủ đề smarthome , clother , laptop , shoes, books

- Code

```
.
import random
import csv
from faker import Faker

# Initialize Faker and set random seed for reproducibility
faker = Faker()
Faker.seed(42)
random.seed(42)

# Define product categories
categories = ["mobilephone", "clothes", "books", "shoes"]

# Function to generate comments
def generate_comment():
    comments = [
        "Great quality!", "Terrible experience.", "Would buy again.",
        "Not worth the price.", "Excellent product!", "Highly
recommend!",
        "Poor customer service.", "Fast delivery!", "Color not as
expected.",
        "Exceeded my expectations!"
    ]
    return random.choice(comments)

# CSV file name
csv_file = "users_with_comments1.csv"

# Write to CSV
with open(csv_file, mode="w", newline="", encoding="utf-8") as file:
    writer = csv.writer(file)
    # Write the header
    writer.writerow(["user_id", "name", "email", "category",
```

```

"comment_text"]])
# Generate and write data
for user_id in range(1, 501): # 500 users
    user_name = faker.name()
    user_email = faker.email()
    for _ in range(50): # 50 comments per user
        category = random.choice(categories)
        comment_text = generate_comment()
        writer.writerow([user_id, user_name, user_email, category,
comment_text])

print(f"Generated data saved to {csv_file}")

```

- Kết quả :



- Giải thích

1. Nhập các thư viện

```
import random
```

```
import csv
```

```
from faker import Faker
```

random: Thư viện chuẩn của Python, dùng để chọn ngẫu nhiên các giá trị hoặc sinh số ngẫu nhiên.

csv: Thư viện để làm việc với file CSV (Comma Separated Values), hỗ trợ đọc và ghi file CSV.

Faker: Thư viện tạo dữ liệu giả lập, giúp sinh thông tin người dùng như tên, email, địa chỉ, v.v.

2. Khởi tạo các công cụ giả lập

```
faker = Faker()
```

```
Faker.seed(42)
```

```
random.seed(42)
```

faker = Faker(): Tạo một đối tượng Faker để tạo dữ liệu giả.

Faker.seed(42) và random.seed(42): Đặt hạt giống (seed) cho hai thư viện Faker và random. Điều này đảm bảo dữ liệu sinh ra luôn giống nhau khi chạy lại chương trình (để dễ dàng kiểm tra).

3. Xác định các danh mục sản phẩm

```
categories = ["mobilephone", "clothes", "books", "shoes"]
```

Danh sách các danh mục sản phẩm để gán ngẫu nhiên cho các bình luận.

4. Hàm tạo nội dung bình luận

```
def generate_comment():
```

```
    comments = [
```

```
        "Great quality!", "Terrible experience.", "Would buy again.",
```

```
        "Not worth the price.", "Excellent product!", "Highly
```

```
recommend!",
```

```
        "Poor customer service.", "Fast delivery!", "Color not as
```

```
expected.",
```

```
        "Exceeded my expectations!"
```

```
    ]
```

```
    return random.choice(comments)
```

comments: Danh sách các nội dung bình luận có sẵn.

random.choice(comments): Chọn ngẫu nhiên một bình luận từ danh sách.

5. Tên file CSV

```
csv_file = "users_with_comments1.csv"
```

Đặt tên file CSV mà dữ liệu sẽ được lưu.

6. Mở file và ghi dữ liệu

```
with open(csv_file, mode="w", newline="", encoding="utf-8") as file:
```

```
    writer = csv.writer(file)
```

open(csv_file, mode="w", ...): Mở file users_with_comments1.csv ở chế độ ghi ("w"). Nếu file đã tồn tại, nó sẽ bị ghi đè.

newline="": Ngăn cách các dòng trong file CSV đúng chuẩn (tránh tạo thêm dòng trống).

encoding="utf-8": Đảm bảo dữ liệu được ghi với mã hóa UTF-8 (hỗ trợ ký tự đặc biệt).

writer = csv.writer(file): Tạo đối tượng writer để ghi dữ liệu vào file CSV.

7. Ghi tiêu đề cho file CSV

```
writer.writerow(["user_id", "name", "email", "category",  
"comment_text"])
```

Ghi dòng tiêu đề (header) vào file CSV.

8. Vòng lặp sinh dữ liệu

```
for user_id in range(1, 501): # 500 users
```

```

user_name = faker.name()
user_email = faker.email()
for _ in range(50): # 50 comments per user
    category = random.choice(categories)
    comment_text = generate_comment()
    writer.writerow([user_id, user_name, user_email, category,
comment_text])
for user_id in range(1, 501):: Lặp qua 500 người dùng với user_id từ 1
đến 500.
user_name = faker.name(): Sinh tên ngẫu nhiên cho người dùng.
user_email = faker.email(): Sinh email ngẫu nhiên.
Vòng lặp bên trong (for _ in range(50)::): Lặp 50 lần cho mỗi người
dùng để tạo 50 bình luận:
random.choice(categories): Chọn một danh mục sản phẩm ngẫu nhiên.
generate_comment(): Sinh một nội dung bình luận ngẫu nhiên.
writer.writerow(...): Ghi một dòng dữ liệu (gồm user_id, tên, email,
danh mục, bình luận) vào file CSV.

```

9. In thông báo hoàn thành

```

print(f"Generated data saved to {csv_file}")

```

Thông báo rằng dữ liệu đã được sinh và lưu thành công vào file CSV.

Câu 2 : Phân tích comment để phân loại người dùng thích hay không thích, trung tính về sản phẩm .

- Code :

```

import pandas as pd
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

# Đọc dữ liệu từ file CSV
csv_file = "users_with_comments1.csv"
try:
    df = pd.read_csv(csv_file)
except FileNotFoundError:
    print(f"Error: File {csv_file} not found.")

```

```
exit()

# Kiểm tra dữ liệu đầu vào
print("Data preview:")
print(df.head())

# Kiểm tra các cột cần thiết
if 'comment_text' not in df.columns or 'category' not in df.columns:
    print("Error: Required columns 'comment_text' or 'category' are missing.")
    exit()

# Loại bỏ các hàng có giá trị bị thiếu
df = df.dropna(subset=['comment_text', 'category'])

# Tạo Tokenizer
tokenizer = Tokenizer(num_words=5000, oov_token="<OOV>")
tokenizer.fit_on_texts(df['comment_text'])

# Chuyển đổi bình luận thành chuỗi số
sequences = tokenizer.texts_to_sequences(df['comment_text'])

# Đệm chuỗi để có cùng độ dài
padded_sequences = pad_sequences(sequences, maxlen=30,
padding='post')

# In thử kết quả
print("Sample padded sequences:")
print(padded_sequences[:5])

# Mã hóa nhãn
label_encoder = LabelEncoder()
labels = label_encoder.fit_transform(df['category'])

# Kiểm tra kết quả
print("Sample labels:")
print(labels[:5])

# Tách dữ liệu
X_train, X_test, y_train, y_test = train_test_split(
```

```

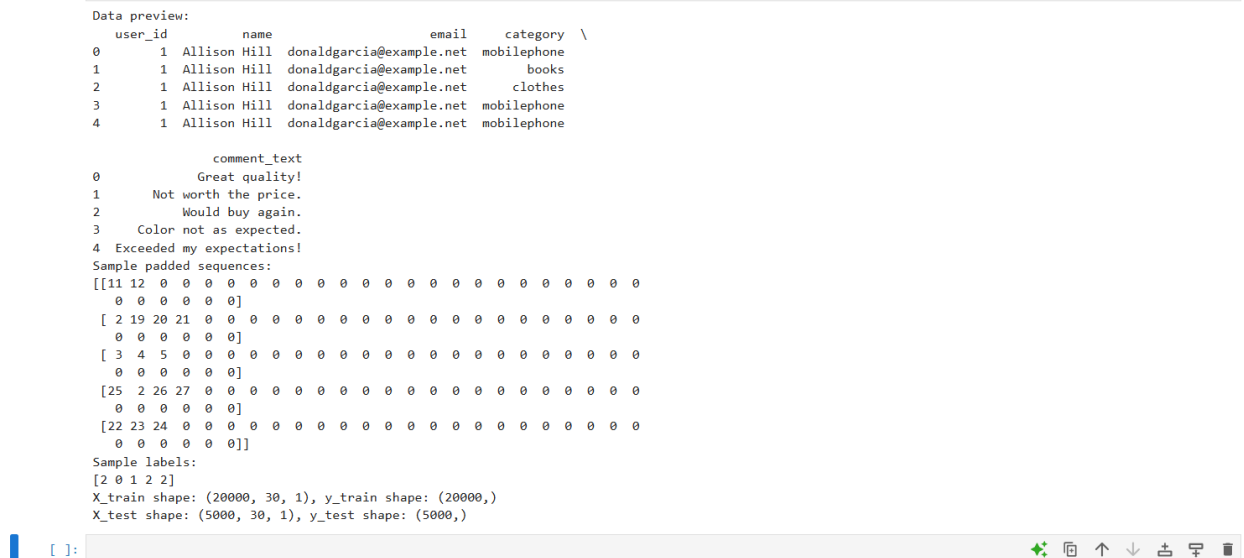
        padded_sequences, labels, test_size=0.2, random_state=42
    )

    # Định dạng dữ liệu cho mô hình CNN
    X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
    X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

    # In kích thước của dữ liệu
    print(f"X_train shape: {X_train.shape}, y_train shape:
    {y_train.shape}")
    print(f"X_test shape: {X_test.shape}, y_test shape: {y_test.shape}")

```

- Kết quả :



The screenshot shows a Jupyter Notebook interface with the following content:

```

Data preview:
  user_id  name  email  category \
0      1  Allison Hill  donaldegarcia@example.net  mobilephone
1      1  Allison Hill  donaldegarcia@example.net  books
2      1  Allison Hill  donaldegarcia@example.net  clothes
3      1  Allison Hill  donaldegarcia@example.net  mobilephone
4      1  Allison Hill  donaldegarcia@example.net  mobilephone

  comment_text
0  Great quality!
1  Not worth the price.
2  Would buy again.
3  Color not as expected.
4  Exceeded my expectations!
Sample padded sequences:
[[11 12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 2 19 20 21 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 3 4 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [25 2 26 27 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [22 23 24 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0]]
Sample labels:
[2 0 1 2 2]
X_train shape: (20000, 30, 1), y_train shape: (20000,)
X_test shape: (5000, 30, 1), y_test shape: (5000,)

```

- Giải thích

1. Đọc dữ liệu từ file CSV

```

python
Copy code
csv_file = "users_with_comments1.csv"
try:
    df = pd.read_csv(csv_file)
except FileNotFoundError:
    print(f"Error: File {csv_file} not found.")

```

exit()

- **Mục đích:** Đọc dữ liệu từ tệp CSV có chứa các bình luận (comment_text) và nhãn phân loại (category).
 - **Xử lý lỗi:** Nếu file không tồn tại, chương trình sẽ in thông báo lỗi và kết thúc.
-

2. Kiểm tra dữ liệu và cột cần thiết

python

Copy code

```
print("Data preview:")  
print(df.head())
```

```
if 'comment_text' not in df.columns or 'category' not in df.columns:  
    print("Error: Required columns 'comment_text' or 'category' are missing.")  
    exit()
```

- **Mục đích:** Xem trước dữ liệu và đảm bảo rằng file CSV chứa các cột cần thiết:
 - comment_text: Chứa nội dung bình luận.
 - category: Nhãn phân loại của bình luận (VD: "tích cực", "tiêu cực", "trung tính").
-

3. Xử lý giá trị bị thiếu

python

Copy code

```
df = df.dropna(subset=['comment_text', 'category'])
```

- **Mục đích:** Loại bỏ các hàng bị thiếu dữ liệu trong các cột quan trọng (comment_text và category).
-

4. Tạo Tokenizer và chuyển đổi văn bản thành chuỗi số

python

Copy code

```
tokenizer = Tokenizer(num_words=5000, oov_token="<OOV>")
tokenizer.fit_on_texts(df['comment_text'])
```

```
sequences = tokenizer.texts_to_sequences(df['comment_text'])
```

- **Tokenizer:**
 - **num_words=5000:** Chỉ giữ lại 5000 từ phổ biến nhất trong tập dữ liệu.
 - **oov_token="<OOV>":** Từ không có trong từ vựng sẽ được gán nhãn là "OOV" (Out Of Vocabulary).
 - **Chuyển đổi:** texts_to_sequences chuyển mỗi câu bình luận thành một danh sách các số nguyên tương ứng với các từ trong câu.
-

5. Đệm chuỗi để có độ dài đồng nhất

```
padded_sequences = pad_sequences(sequences, maxlen=30, padding='post')
```

- **pad_sequences:** Đảm bảo tất cả các chuỗi có độ dài bằng 30:
 - **maxlen=30:** Độ dài tối đa của mỗi chuỗi (các chuỗi dài hơn sẽ bị cắt, ngắn hơn sẽ được đệm).
 - **padding='post':** Đệm thêm số 0 ở cuối mỗi chuỗi (nếu cần).
-

6. Mã hóa nhãn

```
label_encoder = LabelEncoder()
labels = label_encoder.fit_transform(df['category'])
```

- **LabelEncoder:** Chuyển đổi nhãn phân loại (VD: "tích cực", "tiêu cực") thành các số nguyên:
 - VD: "tích cực" → 0, "tiêu cực" → 1.
-

7. Tách tập dữ liệu thành train và test


```
X_train, X_test, y_train, y_test = train_test_split(
    padded_sequences, labels, test_size=0.2, random_state=42
)
```

- **train_test_split**: Chia dữ liệu thành hai tập:
 - **80%** làm dữ liệu huấn luyện (X_train, y_train).
 - **20%** làm dữ liệu kiểm tra (X_test, y_test).
 - **random_state=42**: Đảm bảo việc chia dữ liệu là ngẫu nhiên nhưng có thể tái hiện.
-

8. Định dạng dữ liệu cho mô hình CNN

```
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

- **Mục đích**: Định dạng dữ liệu đầu vào để phù hợp với mô hình học sâu:
 - Thêm một chiều thứ ba (1) để mô phỏng dữ liệu có "kênh" (channel), vì CNN thường làm việc với dữ liệu hình ảnh 2D hoặc 3D.
-

9. In kích thước của dữ liệu

```
print(f"X_train shape: {X_train.shape}, y_train shape: {y_train.shape}")
print(f"X_test shape: {X_test.shape}, y_test shape: {y_test.shape}")
```

- **Mục đích**: Kiểm tra lại kích thước dữ liệu sau khi xử lý:
 - **X_train.shape**: (Số lượng mẫu, Độ dài chuỗi, Số kênh).
 - **y_train.shape**: (Số lượng mẫu,)

Câu 3 : Dự đoán các người dùng về thích hay không thích sản phẩm trên

- Mô hình cnn :

Code

```
.
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten,
Dense, Embedding, Dropout
```

```
# Xây dựng model CNN
```

```
cnn_model = Sequential([
    Conv1D(70, kernel_size=3, activation='relu',
input_shape=(X_train.shape[1], 1)), # Layer 1
    MaxPooling1D(pool_size=2), # Layer 2
    Conv1D(70, kernel_size=3, activation='relu'), # Layer 3
    MaxPooling1D(pool_size=2), # Layer 4
    Flatten(), # Layer 5
    Dense(70, activation='relu'), # Layer 6
    Dense(len(label_encoder.classes_), activation='softmax') # Layer 7
(Output)
])
```

```
# Compile model
```

```
cnn_model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
```

```
# Train model
```

```
cnn_history = cnn_model.fit(X_train, y_train, validation_data=(X_test,
y_test), epochs=10, batch_size=32)
```

- Kết quả :

```
Supervised Learning (activity_regularizer=activity_regularizer, ... Average)
625/625 — 6s 6ms/step - accuracy: 0.2554 - loss: 1.3976 - val_accuracy: 0.2636 - val_loss: 1.3862
Epoch 2/10
625/625 — 3s 4ms/step - accuracy: 0.2508 - loss: 1.3866 - val_accuracy: 0.2590 - val_loss: 1.3863
Epoch 3/10
625/625 — 3s 4ms/step - accuracy: 0.2601 - loss: 1.3862 - val_accuracy: 0.2402 - val_loss: 1.3865
Epoch 4/10
625/625 — 3s 4ms/step - accuracy: 0.2496 - loss: 1.3865 - val_accuracy: 0.2590 - val_loss: 1.3866
Epoch 5/10
625/625 — 3s 4ms/step - accuracy: 0.2540 - loss: 1.3864 - val_accuracy: 0.2590 - val_loss: 1.3866
Epoch 6/10
625/625 — 3s 4ms/step - accuracy: 0.2524 - loss: 1.3864 - val_accuracy: 0.2590 - val_loss: 1.3865
Epoch 7/10
625/625 — 3s 5ms/step - accuracy: 0.2566 - loss: 1.3862 - val_accuracy: 0.2590 - val_loss: 1.3862
Epoch 8/10
625/625 — 3s 4ms/step - accuracy: 0.2634 - loss: 1.3861 - val_accuracy: 0.2590 - val_loss: 1.3863
Epoch 9/10
625/625 — 3s 5ms/step - accuracy: 0.2530 - loss: 1.3864 - val_accuracy: 0.2590 - val_loss: 1.3860
Epoch 10/10
625/625 — 3s 4ms/step - accuracy: 0.2571 - loss: 1.3862 - val_accuracy: 0.2590 - val_loss: 1.3861
```

- Giải thích

1. Import các lớp cần thiết:

python

Copy code

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense,
Embedding, Dropout
```

- **Sequential:** Là một mô hình mạng nơ-ron tuyến tính, nơi các lớp được xếp theo thứ tự trong một chuỗi.
- **Conv1D:** Lớp tích chập 1D, thường được dùng để xử lý dữ liệu dạng chuỗi (ví dụ như văn bản hoặc âm thanh).
- **MaxPooling1D:** Lớp giảm chiều (pooling) trong không gian 1 chiều, giúp giảm độ phức tạp và giúp mô hình tổng quát hơn.
- **Flatten:** Là lớp làm phẳng dữ liệu sau các lớp tích chập để chuẩn bị cho các lớp fully connected.
- **Dense:** Là lớp fully connected, kết nối tất cả các nơ-ron từ lớp trước với lớp hiện tại.

2. Xây dựng mô hình CNN:

python

Copy code

```
cnn_model = Sequential([
    Conv1D(70, kernel_size=3, activation='relu', input_shape=(X_train.shape[1],
1)), # Layer 1
    MaxPooling1D(pool_size=2), # Layer 2
    Conv1D(70, kernel_size=3, activation='relu'), # Layer 3
    MaxPooling1D(pool_size=2), # Layer 4
    Flatten(), # Layer 5
    Dense(70, activation='relu'), # Layer 6
    Dense(len(label_encoder.classes_), activation='softmax') # Layer 7 (Output)
])
```

- **Conv1D(70, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)):**
 - 70: Số lượng filters (tính năng) trong lớp tích chập.
 - kernel_size=3: Kích thước kernel (hoặc cửa sổ) là 3, tức là lớp sẽ quét qua 3 từ tại mỗi lần.

- `activation='relu'`: Sử dụng hàm kích hoạt ReLU (Rectified Linear Unit) giúp mô hình học tốt hơn, giảm thiểu vấn đề vanishing gradient.
- `input_shape=(X_train.shape[1], 1)`: Định nghĩa hình dạng của dữ liệu đầu vào. Trong trường hợp này, dữ liệu đầu vào có chiều dài là `X_train.shape[1]` và 1 kênh (một chiều duy nhất).
- **MaxPooling1D(pool_size=2)**:
 - Là lớp giảm chiều (pooling). Ở đây, kích thước cửa sổ là 2, có nghĩa là mỗi lần lấy mẫu, mô hình sẽ giảm xuống còn một nửa chiều dài.
- **Conv1D(70, kernel_size=3, activation='relu')**: Lớp tích chập thứ hai, tương tự như lớp đầu tiên nhưng không cần khai báo `input_shape` nữa vì mô hình đã có đầu vào từ lớp trước.
- **MaxPooling1D(pool_size=2)**: Lớp giảm chiều tiếp theo, tương tự như lớp trước.
- **Flatten()**: Lớp này làm phẳng đầu ra của lớp tích chập để chuyển sang các lớp fully connected (Dense). Trước khi chuyển sang lớp Dense, dữ liệu cần phải có dạng vector một chiều.
- **Dense(70, activation='relu')**:
 - Lớp fully connected với 70 nơ-ron và hàm kích hoạt ReLU.
- **Dense(len(label_encoder.classes_), activation='softmax')**:
 - Lớp đầu ra của mô hình với số lượng nơ-ron bằng với số lớp phân loại (số lớp trong nhãn category).
 - `activation='softmax'`: Hàm kích hoạt Softmax giúp phân loại đa lớp. Mỗi nơ-ron đầu ra sẽ cho xác suất của mỗi lớp.

3. Compile mô hình:

python

Copy code

```
cnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
```

- **optimizer='adam'**: Adam là một thuật toán tối ưu hiệu quả cho học sâu, tự động điều chỉnh tốc độ học (learning rate).
- **loss='sparse_categorical_crossentropy'**: Hàm mất mát (loss function) này được sử dụng cho bài toán phân loại đa lớp khi nhãn chưa được mã hóa one-hot (nhãn là số nguyên thay vì mảng one-hot).
- **metrics=['accuracy']**: Đo lường độ chính xác (accuracy) của mô hình trong quá trình huấn luyện và đánh giá.

4. Huấn luyện mô hình:

python

Copy code

```
cnn_history = cnn_model.fit(X_train, y_train, validation_data=(X_test, y_test),  
epochs=10, batch_size=32)
```

- **X_train, y_train**: Dữ liệu huấn luyện (đầu vào và nhãn).
- **validation_data=(X_test, y_test)**: Dữ liệu kiểm tra (validation) để theo dõi hiệu suất của mô hình trong quá trình huấn luyện.
- **epochs=10**: Huấn luyện mô hình trong 10 vòng (epochs).
- **batch_size=32**: Chia dữ liệu thành các batch có kích thước 32. Mỗi batch sẽ được xử lý độc lập trước khi cập nhật trọng số mô hình.

- RNN :

Code

```
.  
  
from tensorflow.keras.layers import SimpleRNN, Dropout, Dense  
from tensorflow.keras.models import Sequential  
  
# Xây dựng model RNN  
rnn_model = Sequential([  
    SimpleRNN(70, activation='relu', return_sequences=True,  
input_shape=(X_train.shape[1], 1)), # Layer 1  
    Dropout(0.2), # Layer 2  
    SimpleRNN(70, activation='relu', return_sequences=True), # Layer  
3  
    Dropout(0.2), # Layer 4  
    SimpleRNN(70, activation='relu'), # Layer 5  
    Dense(70, activation='relu'), # Layer 6  
    Dense(len(label_encoder.classes_), activation='softmax') # Layer 7  
(Output)  
)  
  
# Compile model  
rnn_model.compile(optimizer='adam',  
                  loss='sparse_categorical_crossentropy',  
                  metrics=['accuracy'])
```

```
# Train model
rnn_history = rnn_model.fit(X_train, y_train, validation_data=(X_test,
y_test), epochs=10, batch_size=32)
```

- Kết quả :

```
625/625 ————— 16s 18ms/step - accuracy: 0.2524 - loss: 1.3868 - val_accuracy: 0.2590 - val_loss: 1.3868
Epoch 2/10
625/625 ————— 11s 18ms/step - accuracy: 0.2607 - loss: 1.3861 - val_accuracy: 0.2590 - val_loss: 1.3860
Epoch 3/10
625/625 ————— 11s 18ms/step - accuracy: 0.2559 - loss: 1.3863 - val_accuracy: 0.2590 - val_loss: 1.3861
Epoch 4/10
625/625 ————— 11s 18ms/step - accuracy: 0.2590 - loss: 1.3861 - val_accuracy: 0.2590 - val_loss: 1.3861
Epoch 5/10
625/625 ————— 10s 16ms/step - accuracy: 0.2553 - loss: 1.3863 - val_accuracy: 0.2590 - val_loss: 1.3864
Epoch 6/10
625/625 ————— 11s 18ms/step - accuracy: 0.2604 - loss: 1.3861 - val_accuracy: 0.2590 - val_loss: 1.3862
Epoch 7/10
625/625 ————— 11s 17ms/step - accuracy: 0.2617 - loss: 1.3861 - val_accuracy: 0.2590 - val_loss: 1.3863
Epoch 8/10
625/625 ————— 11s 17ms/step - accuracy: 0.2537 - loss: 1.3863 - val_accuracy: 0.2590 - val_loss: 1.3864
Epoch 9/10
625/625 ————— 10s 16ms/step - accuracy: 0.2592 - loss: 1.3860 - val_accuracy: 0.2590 - val_loss: 1.3862
Epoch 10/10
625/625 ————— 11s 17ms/step - accuracy: 0.2420 - loss: 1.3865 - val_accuracy: 0.2590 - val_loss: 1.3860
```

- Giải thích :

1. Import thư viện

```
from tensorflow.keras.layers import SimpleRNN, Dropout, Dense
from tensorflow.keras.models import Sequential
```

- **SimpleRNN**: Lớp đơn giản của Mạng Neural Hồi quy (RNN), dùng để xử lý dữ liệu chuỗi (sequence data).
- **Dropout**: Lớp Dropout để giảm overfitting bằng cách ngẫu nhiên loại bỏ một tỷ lệ các nơ-ron trong quá trình huấn luyện.
- **Dense**: Lớp fully connected, mỗi nơ-ron trong lớp này sẽ kết nối với tất cả các nơ-ron trong lớp trước đó.
- **Sequential**: Lớp mô hình tuần tự của Keras, giúp xếp các lớp trong mô hình theo thứ tự.

2. Xây dựng mô hình RNN

```
rnn_model = Sequential([
    SimpleRNN(70, activation='relu', return_sequences=True,
input_shape=(X_train.shape[1], 1)), # Layer 1
    Dropout(0.2), # Layer 2
    SimpleRNN(70, activation='relu', return_sequences=True), # Layer 3
    Dropout(0.2), # Layer 4
```

```

SimpleRNN(70, activation='relu'), # Layer 5
Dense(70, activation='relu'), # Layer 6
Dense(len(label_encoder.classes_), activation='softmax') # Layer 7 (Output)
])

```

Mô hình RNN gồm các lớp sau:

- **SimpleRNN(70, activation='relu', return_sequences=True, input_shape=(X_train.shape[1], 1)):**
 - Đây là lớp RNN đầu tiên với 70 nơ-ron và hàm kích hoạt ReLU (activation='relu').
 - input_shape=(X_train.shape[1], 1): Định dạng đầu vào là chuỗi dữ liệu có chiều dài X_train.shape[1] và một kênh (chiều dài chuỗi, mỗi bước có một giá trị).
 - return_sequences=True: Điều này có nghĩa là lớp này sẽ trả về tất cả các chuỗi (sequences) thay vì chỉ trả về trạng thái cuối cùng, để có thể truyền các chuỗi này tới lớp RNN tiếp theo.
- **Dropout(0.2):**
 - Lớp này có tác dụng ngẫu nhiên bỏ qua 20% các nơ-ron trong quá trình huấn luyện nhằm giảm overfitting.
- **SimpleRNN(70, activation='relu', return_sequences=True):**
 - Lớp RNN thứ hai có 70 nơ-ron và cũng trả về các chuỗi (sequences) vì return_sequences=True.
- **Dropout(0.2):**
 - Lớp Dropout thứ hai nhằm giúp giảm overfitting.
- **SimpleRNN(70, activation='relu'):**
 - Lớp RNN cuối cùng không trả về chuỗi mà chỉ trả về trạng thái cuối cùng của chuỗi. Nó có 70 nơ-ron và hàm kích hoạt ReLU.
- **Dense(70, activation='relu'):**
 - Lớp fully connected với 70 nơ-ron và hàm kích hoạt ReLU.
- **Dense(len(label_encoder.classes_), activation='softmax'):**
 - Lớp đầu ra của mô hình có số nơ-ron bằng với số lượng lớp trong label_encoder.classes_, tức là số lớp phân loại mà mô hình cần phân biệt.
 - activation='softmax' được sử dụng trong lớp đầu ra để đảm bảo các giá trị đầu ra là xác suất, vì đây là bài toán phân loại đa lớp.

3. Biên dịch mô hình

```
rnn_model.compile(optimizer='adam',
```

```
loss='sparse_categorical_crossentropy',  
metrics=['accuracy'])
```

- **optimizer='adam'**: Adam là một thuật toán tối ưu hóa phổ biến, tự động điều chỉnh tốc độ học (learning rate) và có hiệu quả tốt trong nhiều bài toán học máy.
- **loss='sparse_categorical_crossentropy'**: Hàm mất mát (loss function) sử dụng cho bài toán phân loại đa lớp. "Sparse" có nghĩa là nhãn của bạn là số nguyên (mỗi lớp có một số nguyên tương ứng, không phải one-hot encoding).
- **metrics=['accuracy']**: Đo lường độ chính xác của mô hình trong quá trình huấn luyện và kiểm tra.

4. Huấn luyện mô hình

```
rnn_history = rnn_model.fit(X_train, y_train, validation_data=(X_test, y_test),  
epochs=10, batch_size=32)
```

- **X_train, y_train**: Dữ liệu huấn luyện (X_train là đầu vào, y_train là nhãn).
- **validation_data=(X_test, y_test)**: Dữ liệu kiểm tra dùng để đánh giá mô hình sau mỗi epoch (vòng lặp huấn luyện).
- **epochs=10**: Mô hình sẽ huấn luyện 10 vòng. Mỗi vòng huấn luyện sẽ chạy qua tất cả dữ liệu huấn luyện.
- **batch_size=32**: Số lượng mẫu trong mỗi batch khi huấn luyện. Mô hình sẽ cập nhật trọng số sau khi xử lý xong 32 mẫu.

- LSTM :

Code :

```
.  
from tensorflow.keras.layers import LSTM, Dropout, Dense  
from tensorflow.keras.models import Sequential  
  
# Xây dựng model LSTM  
lstm_model = Sequential([  
    LSTM(70, activation='relu', return_sequences=True,  
input_shape=(X_train.shape[1], 1)), # Layer 1  
    Dropout(0.2), # Layer 2  
    LSTM(70, activation='relu', return_sequences=True), # Layer 3
```



```

Dropout(0.2), # Layer 4
LSTM(70, activation='relu'), # Layer 5
Dense(70, activation='relu'), # Layer 6
Dense(len(label_encoder.classes_), activation='softmax') # Layer 7 (Output)
])

# Compile model
lstm_model.compile(optimizer='adam',
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])

# Train model
lstm_history = lstm_model.fit(X_train, y_train,
                             validation_data=(X_test, y_test),
                             epochs=10,
                             batch_size=32)

```

- Kết quả :

```

625/625 — 27s 35ms/step - accuracy: 0.2510 - loss: 1.3866 - val_accuracy: 0.2590 - val_loss: 1.3867
Epoch 2/10
625/625 — 21s 33ms/step - accuracy: 0.2585 - loss: 1.3863 - val_accuracy: 0.2590 - val_loss: 1.3862
Epoch 3/10
625/625 — 22s 35ms/step - accuracy: 0.2507 - loss: 1.3865 - val_accuracy: 0.2590 - val_loss: 1.3861
Epoch 4/10
625/625 — 22s 35ms/step - accuracy: 0.2619 - loss: 1.3861 - val_accuracy: 0.2590 - val_loss: 1.3864
Epoch 5/10
625/625 — 22s 34ms/step - accuracy: 0.2491 - loss: 1.3864 - val_accuracy: 0.2590 - val_loss: 1.3863
Epoch 6/10
625/625 — 21s 33ms/step - accuracy: 0.2563 - loss: 1.3863 - val_accuracy: 0.2590 - val_loss: 1.3866
Epoch 7/10
625/625 — 21s 33ms/step - accuracy: 0.2486 - loss: 1.3863 - val_accuracy: 0.2590 - val_loss: 1.3858
Epoch 8/10
625/625 — 24s 38ms/step - accuracy: 0.2487 - loss: 1.3864 - val_accuracy: 0.2590 - val_loss: 1.3861
Epoch 9/10
625/625 — 23s 37ms/step - accuracy: 0.2571 - loss: 1.3862 - val_accuracy: 0.2590 - val_loss: 1.3861
Epoch 10/10
625/625 — 23s 36ms/step - accuracy: 0.2567 - loss: 1.3863 - val_accuracy: 0.2590 - val_loss: 1.3863

```

- Giải thích :

1. Import Libraries

```

from tensorflow.keras.layers import LSTM, Dropout, Dense
from tensorflow.keras.models import Sequential

```

- LSTM: Lớp LSTM được sử dụng để xây dựng mạng nơ-ron hồi quy dài hạn (Long Short-Term Memory), một dạng đặc biệt của mạng nơ-ron hồi quy (RNN) rất hiệu quả trong việc xử lý dữ liệu chuỗi.
- Dropout: Lớp Dropout được sử dụng để giảm overfitting bằng cách ngẫu nhiên loại bỏ một phần của các kết nối trong mỗi vòng huấn luyện.

- Dense: Lớp Dense là lớp fully-connected (mỗi nơ-ron của lớp này kết nối đến mọi nơ-ron ở lớp kế tiếp).
- Sequential: Đây là cách xây dựng mô hình trong Keras, trong đó các lớp được thêm vào một cách tuần tự.

2. Xây Dựng Mô Hình LSTM

```
lstm_model = Sequential([
    LSTM(70, activation='relu', return_sequences=True,
input_shape=(X_train.shape[1], 1)), # Layer 1
    Dropout(0.2), # Layer 2
    LSTM(70, activation='relu', return_sequences=True), # Layer 3
    Dropout(0.2), # Layer 4
    LSTM(70, activation='relu'), # Layer 5
    Dense(70, activation='relu'), # Layer 6
    Dense(len(label_encoder.classes_), activation='softmax') # Layer 7 (Output)
])
```

- **Layer 1:** LSTM(70, activation='relu', return_sequences=True, input_shape=(X_train.shape[1], 1))
 - LSTM(70): Tạo một lớp LSTM với 70 đơn vị (nơ-ron). LSTM giúp mô hình ghi nhớ và xử lý dữ liệu chuỗi dài hạn.
 - activation='relu': Sử dụng hàm kích hoạt ReLU cho các nơ-ron LSTM.
 - return_sequences=True: Vì đây là lớp đầu tiên, chúng ta cần trả về toàn bộ chuỗi thay vì chỉ trạng thái cuối cùng. Điều này cần thiết cho việc xử lý chuỗi trong các lớp tiếp theo.
 - input_shape=(X_train.shape[1], 1): Xác định hình dạng đầu vào cho mô hình (số lượng bước thời gian của chuỗi và số lượng tính năng). Đây là lớp đầu vào, do đó chúng ta chỉ cần chỉ định hình dạng đầu vào tại đây.
- **Layer 2:** Dropout(0.2)
 - Lớp Dropout giúp giảm overfitting bằng cách ngẫu nhiên loại bỏ 20% các kết nối trong mỗi vòng huấn luyện.
- **Layer 3:** LSTM(70, activation='relu', return_sequences=True)
 - Lớp LSTM thứ hai với 70 đơn vị và hàm kích hoạt ReLU.
 - return_sequences=True: Lớp này cũng trả về chuỗi đầu ra, cần thiết để lớp tiếp theo có thể nhận đầu vào dạng chuỗi.
- **Layer 4:** Dropout(0.2)
 - Dropout 20% giúp giảm overfitting sau lớp LSTM thứ hai.

- **Layer 5:** LSTM(70, activation='relu')
 - Lớp LSTM thứ ba với 70 đơn vị. Ở đây không cần return_sequences=True vì đây là lớp cuối cùng xử lý chuỗi, trả về chỉ trạng thái cuối cùng của chuỗi.
- **Layer 6:** Dense(70, activation='relu')
 - Lớp Dense với 70 đơn vị và hàm kích hoạt ReLU. Lớp này xử lý đầu ra của lớp LSTM và truyền qua một lớp fully connected trước khi ra lớp phân loại cuối cùng.
- **Layer 7:** Dense(len(label_encoder.classes_), activation='softmax')
 - Lớp Dense cuối cùng với số lượng đơn vị bằng với số lượng lớp phân loại (len(label_encoder.classes_)).
 - activation='softmax': Sử dụng softmax để tính toán xác suất cho từng lớp. Đây là lựa chọn phổ biến khi làm bài toán phân loại đa lớp (multi-class classification), vì hàm softmax sẽ tạo ra một xác suất tổng hợp cho các lớp, với tổng của tất cả các xác suất bằng 1.

3. Biên Dịch Mô Hình

```
lstm_model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
```

- optimizer='adam': Sử dụng thuật toán tối ưu Adam, một trong những thuật toán tối ưu phổ biến và hiệu quả trong học sâu.
- loss='sparse_categorical_crossentropy': Sử dụng hàm mất mát sparse_categorical_crossentropy, phù hợp cho các bài toán phân loại với nhiều lớp. Mỗi nhãn mục tiêu là một số nguyên.
- metrics=['accuracy']: Theo dõi độ chính xác của mô hình trong quá trình huấn luyện.

4. Huấn Luyện Mô Hình

```
lstm_history = lstm_model.fit(X_train, y_train,
                             validation_data=(X_test, y_test),
                             epochs=10,
                             batch_size=32)
```

- X_train: Dữ liệu huấn luyện đầu vào.
- y_train: Nhãn cho dữ liệu huấn luyện.
- validation_data=(X_test, y_test): Dữ liệu kiểm tra dùng để đánh giá mô hình trong quá trình huấn luyện.

- epochs=10: Mô hình sẽ huấn luyện qua 10 vòng (epochs).
- batch_size=32: Số lượng mẫu trong mỗi batch. Dữ liệu được chia thành các batch nhỏ để mô hình cập nhật trọng số trong mỗi bước huấn luyện.

Câu 4 : Sử dụng các độ đo để đánh giá độ chính xác accuracy, mae , mse , rmse , mape

- Code

```

import numpy as np
from sklearn.metrics import mean_absolute_error,
mean_squared_error, mean_absolute_percentage_error
from tensorflow.keras.metrics import Accuracy

# Giả sử mô hình đã được huấn luyện và bạn đã có dự đoán từ mô
hình
predictions = lstm_model.predict(X_test) # Dự đoán từ mô hình
LSTM

# Chuyển đổi dự đoán thành nhãn phân loại bằng np.argmax
predictions = np.argmax(predictions, axis=1)

# Tính Accuracy
accuracy = Accuracy()
accuracy.update_state(y_test, predictions)
print("Accuracy: ", accuracy.result().numpy())

# Tính MAE
mae = mean_absolute_error(y_test, predictions)
print("MAE: ", mae)

# Tính MSE
mse = mean_squared_error(y_test, predictions)
print("MSE: ", mse)

# Tính RMSE

```

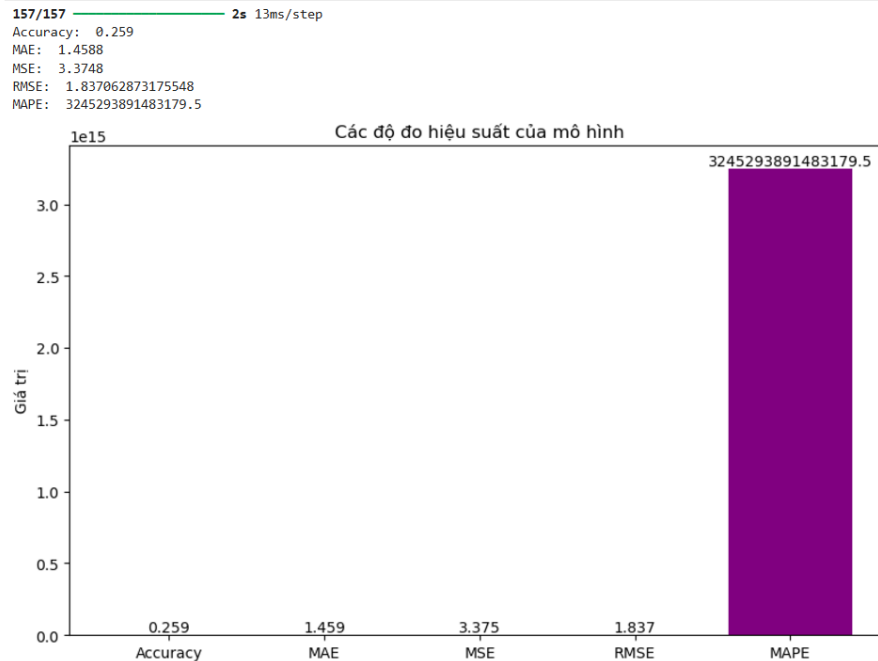
```

rmse = np.sqrt(mse)
print("RMSE: ", rmse)

# Tính MAPE
mape = mean_absolute_percentage_error(y_test, predictions)
print("MAPE: ", mape)

```

- Kết quả :



- Giải thích :

1. Dự đoán từ mô hình:

`predictions = lstm_model.predict(X_test)` # Dự đoán từ mô hình LSTM

- **`lstm_model.predict(X_test)`**: Mô hình LSTM được huấn luyện trước đó sẽ đưa ra các dự đoán dựa trên dữ liệu kiểm tra (`X_test`).
- Kết quả là một mảng có kích thước (số mẫu, số lớp) chứa các xác suất cho mỗi lớp. Nếu là bài toán phân loại với 4 lớp, mỗi dự đoán sẽ là một vector có 4 phần tử, mỗi phần tử là xác suất dự đoán cho một lớp.

2. Chuyển đổi dự đoán thành nhãn phân loại:

`predictions = np.argmax(predictions, axis=1)`

- **np.argmax(predictions, axis=1)**: Hàm np.argmax lấy chỉ số của phần tử lớn nhất trong mỗi hàng (mỗi mẫu). Vì dự đoán của mô hình là xác suất cho từng lớp, bạn chọn lớp có xác suất cao nhất cho mỗi mẫu.
- Kết quả là một mảng 1 chiều (có kích thước [số mẫu]), chứa nhãn phân loại dự đoán của mô hình cho mỗi mẫu.

3. Tính Accuracy:

```
accuracy = Accuracy()
accuracy.update_state(y_test, predictions)
print("Accuracy: ", accuracy.result().numpy())
```

- **Accuracy()**: Đối tượng này tính toán độ chính xác, tức là tỷ lệ mẫu đúng (số mẫu dự đoán đúng chia cho tổng số mẫu).
- **update_state(y_test, predictions)**: Hàm này cập nhật độ chính xác bằng cách so sánh nhãn thực tế (y_test) với các nhãn dự đoán (predictions).
- **accuracy.result().numpy()**: Sau khi cập nhật, bạn lấy giá trị của độ chính xác dưới dạng giá trị số thực và in ra.

4. Tính MAE (Mean Absolute Error):

```
mae = mean_absolute_error(y_test, predictions)
print("MAE: ", mae)
```

- **mean_absolute_error(y_test, predictions)**: Hàm này tính MAE, là giá trị trung bình của sai số tuyệt đối giữa nhãn thực tế (y_test) và nhãn dự đoán (predictions).
 - $MAE = (1/n) * \sum |y_test - predictions|$
 - MAE đo lường độ lệch tuyệt đối trung bình giữa các giá trị thực tế và dự đoán.

5. Tính MSE (Mean Squared Error):

```
mse = mean_squared_error(y_test, predictions)
print("MSE: ", mse)
```

- **mean_squared_error(y_test, predictions)**: Hàm này tính MSE, là giá trị trung bình của sai số bình phương giữa nhãn thực tế và nhãn dự đoán.
 - $MSE = (1/n) * \sum (y_test - predictions)^2$
 - MSE có xu hướng trừng phạt các sai số lớn hơn so với MAE vì sai số được bình phương.

6. Tính RMSE (Root Mean Squared Error):

```
rmse = np.sqrt(mse)
print("RMSE: ", rmse)
```

- **np.sqrt(mse)**: RMSE là căn bậc hai của MSE. Nó giúp đưa sai số về cùng đơn vị với dữ liệu gốc, giúp dễ hiểu hơn.
 - $RMSE = \sqrt{MSE}$.
 - RMSE thường được sử dụng để đánh giá độ chính xác trong các bài toán hồi quy hoặc dự đoán có sai số.

7. Tính MAPE (Mean Absolute Percentage Error):

```
mape = mean_absolute_percentage_error(y_test, predictions)
print("MAPE: ", mape)
```

- **mean_absolute_percentage_error(y_test, predictions)**: Hàm này tính MAPE, là giá trị trung bình của sai số tuyệt đối tính theo tỷ lệ phần trăm giữa nhãn thực tế và nhãn dự đoán.
 - $MAPE = (1/n) * \sum |(y_test - predictions) / y_test| * 100$
 - MAPE giúp đánh giá sai số trung bình dưới dạng tỷ lệ phần trăm, dễ dàng so sánh giữa các mô hình hoặc dữ liệu khác nhau.

Tổng quan:

- **Accuracy**: Tỷ lệ chính xác, phần trăm các dự đoán đúng.
- **MAE**: Sai số tuyệt đối trung bình.
- **MSE**: Sai số bình phương trung bình.
- **RMSE**: Căn bậc hai của MSE, giúp đưa sai số về đơn vị của dữ liệu.
- **MAPE**: Sai số tuyệt đối tính theo tỷ lệ phần trăm.

Câu 5 : Show các biểu đồ + Biểu đồ về thay đổi sở thích người dùng theo thời gian

- Code

```
.
import numpy as np
```

```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import mean_absolute_error,
mean_squared_error, mean_absolute_percentage_error
from tensorflow.keras.metrics import Accuracy

# Giả sử mô hình đã được huấn luyện và bạn đã có dự đoán từ mô
hình
predictions = lstm_model.predict(X_test) # Dự đoán từ mô hình
LSTM

# Chuyển đổi dự đoán thành nhãn phân loại bằng np.argmax
predictions = np.argmax(predictions, axis=1)

# Tính Accuracy
accuracy = Accuracy()
accuracy.update_state(y_test, predictions)
accuracy_value = accuracy.result().numpy()

# Tính MAE (Mean Absolute Error)
mae = mean_absolute_error(y_test, predictions)

# Tính MSE (Mean Squared Error)
mse = mean_squared_error(y_test, predictions)

# Tính RMSE (Root Mean Squared Error)
rmse = np.sqrt(mse)

# Tính MAPE (Mean Absolute Percentage Error)
mape = mean_absolute_percentage_error(y_test, predictions)

# In các giá trị tính toán
print("Accuracy: ", accuracy_value)
print("MAE: ", mae)
print("MSE: ", mse)
print("RMSE: ", rmse)
print("MAPE: ", mape)

# Vẽ biểu đồ các độ đo hiệu suất của mô hình
metrics = ['Accuracy', 'MAE', 'MSE', 'RMSE', 'MAPE']
```



```

values = [accuracy_value, mae, mse, rmse, mape]

# Đảm bảo dữ liệu là float để tránh việc vẽ biểu đồ sai
values = np.array(values, dtype=float)

plt.figure(figsize=(10, 6))
plt.bar(metrics, values, color=['blue', 'orange', 'green', 'red', 'purple'])
plt.ylabel('Giá trị')
plt.title('Các độ đo hiệu suất của mô hình')

# Hiển thị giá trị của từng thanh trên đồ thị
for i in range(len(metrics)):
    plt.text(i, values[i] + 0.02, round(values[i], 3), ha='center',
             va='bottom')

plt.show()

# Biểu đồ thay đổi sở thích người dùng theo thời gian
# Giả sử bạn có dữ liệu về sở thích người dùng theo ngày
# Dữ liệu giả lập, thay thế với dữ liệu thực tế của bạn
dates = pd.date_range(start='2024-01-01', periods=30, freq='D') # 30
ngày
user_preferences = np.random.randint(1, 100, size=30) # Giả lập số
lượt tương tác (hoặc sở thích)

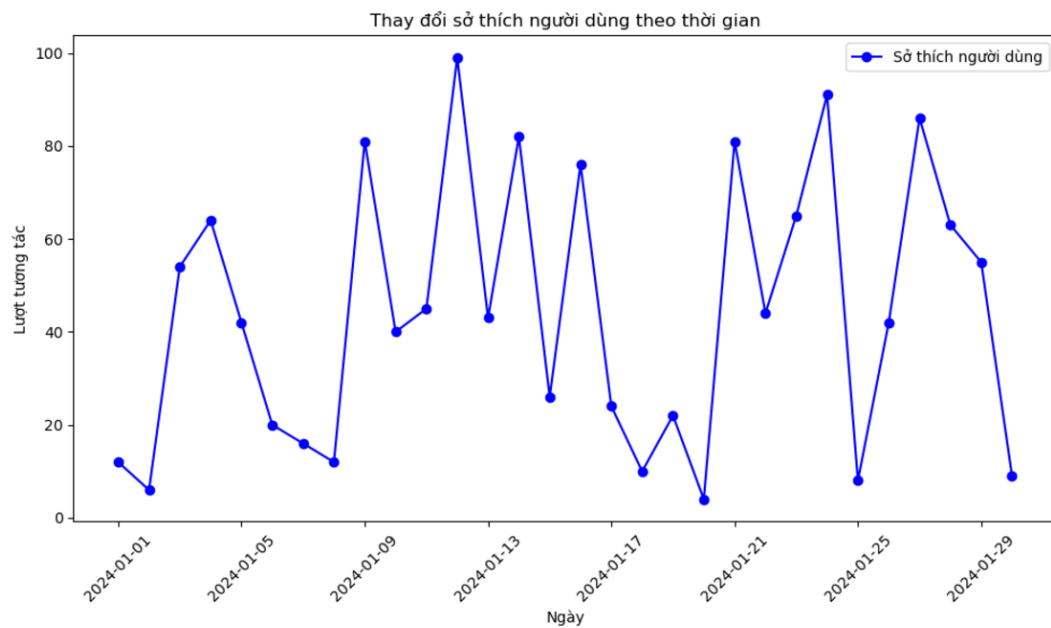
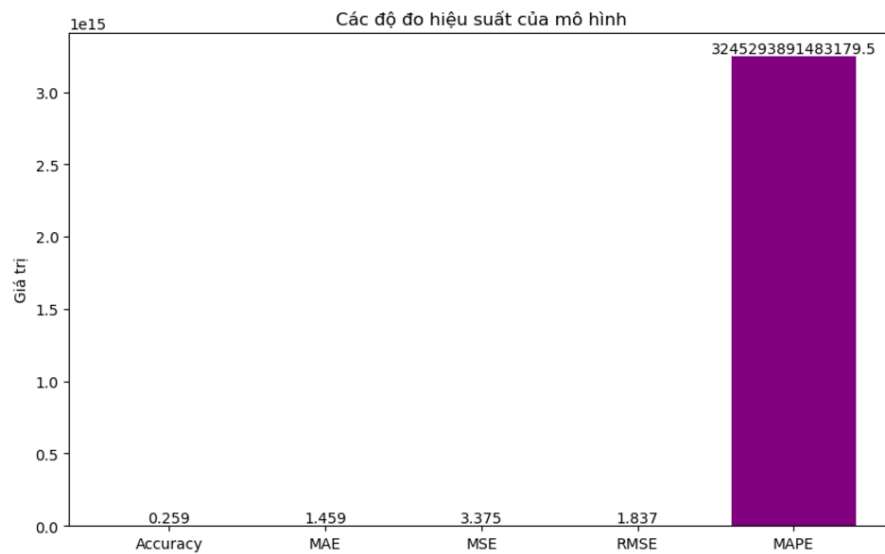
# Vẽ biểu đồ thay đổi sở thích người dùng theo thời gian
plt.figure(figsize=(10, 6))
plt.plot(dates, user_preferences, marker='o', color='b', linestyle='-',
label='Sở thích người dùng')
plt.xlabel('Ngày')
plt.ylabel('Lượt tương tác')
plt.title('Thay đổi sở thích người dùng theo thời gian')
plt.xticks(rotation=45) # Xoay nhãn trục X để dễ đọc
plt.legend()

plt.tight_layout()
plt.show()

```

- kết quả :

157/157 — 2s 14ms/step
Accuracy: 0.259
MAE: 1.4588
MSE: 3.3748
RMSE: 1.837062873175548
MAPE: 3245293891483179.5



- Giải thích

1. Dự đoán từ mô hình LSTM và tính toán các độ đo hiệu suất

- Mô hình đã được huấn luyện và dự đoán:

```
predictions = lstm_model.predict(X_test)
```

Mô hình LSTM (`lstm_model`) được sử dụng để đưa ra dự đoán (`predictions`) từ dữ liệu kiểm tra (`X_test`). Dự đoán này có thể là xác suất cho các lớp phân loại, do đó bạn sử dụng `np.argmax` để chuyển đổi các giá trị xác suất thành nhãn phân loại cụ thể:

```
python  
Copy code  
predictions = np.argmax(predictions, axis=1)
```

Lệnh này lấy chỉ số của giá trị lớn nhất dọc theo trục của mảng xác suất, tương đương với lớp được dự đoán.

- **Tính toán độ chính xác (Accuracy):**

```
accuracy = Accuracy()  
accuracy.update_state(y_test, predictions)  
accuracy_value = accuracy.result().numpy()
```

Đo lường độ chính xác của mô hình, đó là tỷ lệ số dự đoán chính xác so với tổng số mẫu trong `y_test` (giá trị thực tế). Accuracy là một metric được cung cấp bởi TensorFlow/Keras.

- **Tính toán các độ đo lỗi khác:**

- **MAE (Mean Absolute Error):**

```
python  
Copy code  
mae = mean_absolute_error(y_test, predictions)
```

MAE tính toán trung bình của giá trị tuyệt đối giữa giá trị thực và giá trị dự đoán.

- **MSE (Mean Squared Error):**

```
python  
Copy code  
mse = mean_squared_error(y_test, predictions)
```

MSE tính toán trung bình của bình phương sai số giữa giá trị thực và dự đoán. Đây là một chỉ số được dùng nhiều trong các mô hình học máy.

- **RMSE (Root Mean Squared Error):**

```
rmse = np.sqrt(mse)
```

RMSE là căn bậc hai của MSE. Đây là một chỉ số phổ biến để đánh giá độ chính xác của mô hình, với đặc điểm là càng lớn sai số, RMSE càng cao.

- **MAPE (Mean Absolute Percentage Error):**

```
mape = mean_absolute_percentage_error(y_test, predictions)
```

MAPE tính toán sai số tuyệt đối giữa giá trị thực tế và giá trị dự đoán, sau đó tính tỷ lệ phần trăm của sai số này so với giá trị thực tế. MAPE rất hữu ích khi muốn so sánh sai số giữa các mô hình hoặc tập dữ liệu có các đơn vị khác nhau.

- **In ra các giá trị tính toán:** Sau khi tính toán các độ đo, các giá trị này được in ra để người dùng có thể tham khảo.

2. Vẽ biểu đồ các độ đo hiệu suất của mô hình

Mục tiêu của đoạn mã này là vẽ biểu đồ thể hiện các độ đo hiệu suất của mô hình như Accuracy, MAE, MSE, RMSE và MAPE.

- **Vẽ biểu đồ thanh (Bar Chart):**

```
metrics = ['Accuracy', 'MAE', 'MSE', 'RMSE', 'MAPE']  
values = [accuracy_value, mae, mse, rmse, mape]
```

metrics chứa tên của các độ đo, và values chứa giá trị tương ứng cho từng độ đo. Sau đó, bạn vẽ biểu đồ thanh bằng matplotlib:

```
plt.figure(figsize=(10, 6))  
plt.bar(metrics, values, color=['blue', 'orange', 'green', 'red', 'purple'])
```

Biểu đồ này sẽ giúp bạn dễ dàng so sánh các độ đo hiệu suất.

- **Hiển thị giá trị trên các thanh:** Để làm cho biểu đồ dễ hiểu hơn, các giá trị của từng thanh được hiển thị trực tiếp trên biểu đồ:

```
for i in range(len(metrics)):
    plt.text(i, values[i] + 0.02, round(values[i], 3), ha='center', va='bottom')
```

3. Vẽ biểu đồ thay đổi sở thích người dùng theo thời gian

Biểu đồ này giả lập sự thay đổi của sở thích người dùng theo thời gian (30 ngày).

- **Dữ liệu giả lập:**

```
dates = pd.date_range(start='2024-01-01', periods=30, freq='D') # 30 ngày
user_preferences = np.random.randint(1, 100, size=30) # Giả lập số lượt
tương tác (hoặc sở thích)
```

Sử dụng `pd.date_range` để tạo một dãy ngày từ 1 tháng 1 năm 2024 đến 30 ngày sau. `user_preferences` là một mảng giả lập số lượt tương tác của người dùng, sử dụng `np.random.randint` để tạo dữ liệu ngẫu nhiên trong khoảng từ 1 đến 100.

- **Vẽ biểu đồ đường (Line Chart):** Dữ liệu sẽ được vẽ dưới dạng đồ thị đường để thể hiện sự thay đổi sở thích người dùng qua các ngày:

```
plt.figure(figsize=(10, 6))
plt.plot(dates, user_preferences, marker='o', color='b', linestyle='-', label='Sở
thích người dùng')
```

Đồ thị đường giúp bạn dễ dàng theo dõi xu hướng thay đổi của sở thích người dùng theo thời gian.

- **Hiển thị nhãn và tiêu đề:** Các nhãn cho trục X và Y được thêm vào, và tiêu đề của biểu đồ cũng được hiển thị:

```
plt.xlabel('Ngày')
plt.ylabel('Lượt tương tác')
plt.title('Thay đổi sở thích người dùng theo thời gian')
```

- **Điều chỉnh vị trí nhãn trên trục X:** Để dễ đọc các nhãn ngày tháng trên trục X, bạn xoay chúng một góc 45 độ:

```
plt.xticks(rotation=45)
```

- **Hiển thị biểu đồ:** Cuối cùng, gọi `plt.show()` để hiển thị biểu đồ:

```
plt.tight_layout()
```

```
plt.show()
```