AI VIET NAM
@aivietnam.edu.vn

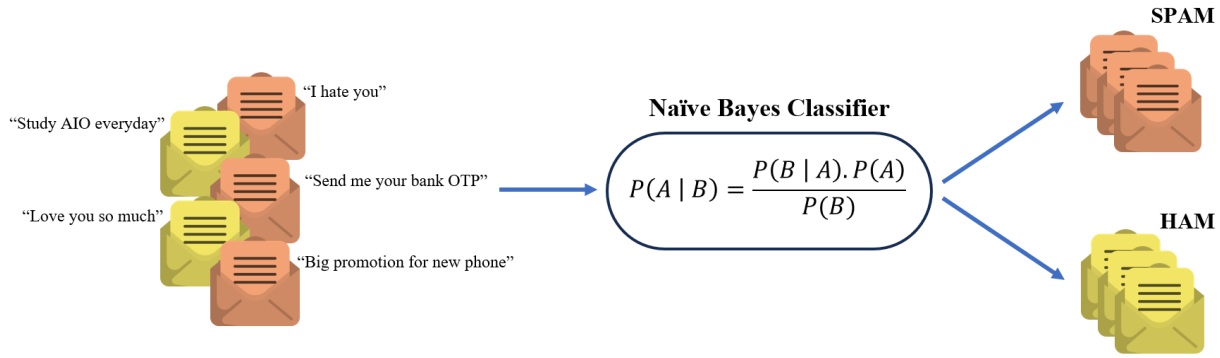
AI VIET NAM – AI COURSE 2025

Phân loại tin nhắn spam sử dụng Naive Bayes và cơ sở dữ liệu vector

Quoc-Thai Nguyen Quang-Hien Ho Quang-Vinh Dinh

I. Giới thiệu

Text Classification (Tạm dịch: Phân loại văn bản) là một trong những bài toán phổ biến trong lĩnh vực Machine Learning và Natural Language Processing. Trong đó, nhiệm vụ của chúng ta là xây dựng một chương trình có khả năng phân loại văn bản vào các phân lớp do chúng ta quy định. Các ứng dụng phổ biến liên quan đến loại chương trình này có thể kể đến phát hiện các bình luận tiêu cực trên không gian mạng, các đánh giá tích cực của sản phẩm...



Hình 1: Minh họa ứng dụng phân loại tin nhắn rác (spam) và tin nhắn thường (ham).

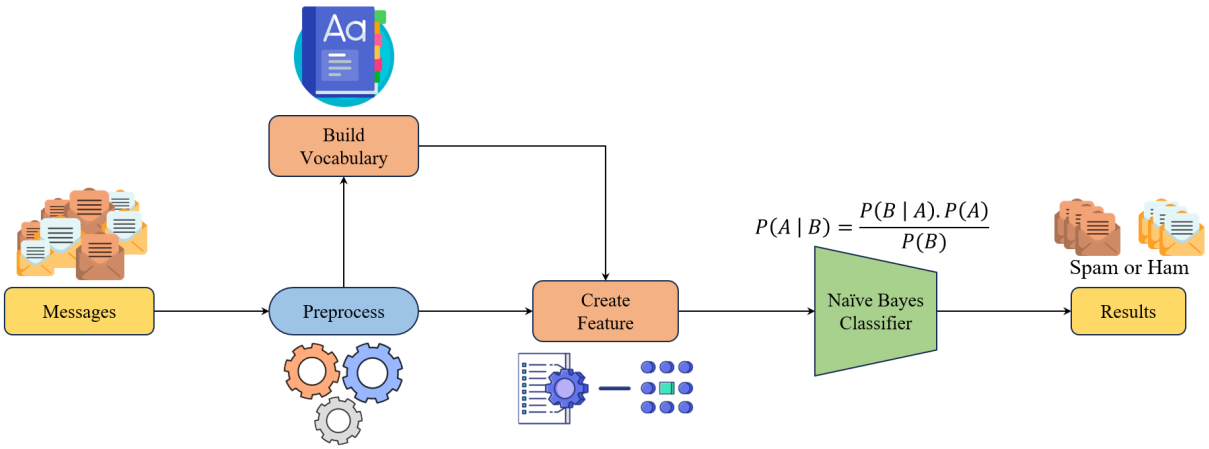
Trong project này, chúng ta sẽ xây dựng một chương trình Text Classification liên quan đến việc phân loại một đoạn tin nhắn là tin nhắn spam hay không. Dự án sẽ khám phá và triển khai hai phương pháp tiếp cận chính để giải quyết bài toán này:

- **Naive Bayes Classifier:** Một thuật toán học máy truyền thống dựa trên định lý Bayes với giả định độc lập có điều kiện giữa các đặc trưng, thường hiệu quả cho các bài toán phân loại văn bản.
- **Phân loại sử dụng Cơ sở dữ liệu Vector (Vector Database) và Sentence Embeddings:** Một phương pháp hiện đại hơn, trong đó các tin nhắn được biểu diễn dưới dạng vector ngữ nghĩa (embeddings) và việc phân loại dựa trên việc tìm kiếm sự tương đồng trong một cơ sở dữ liệu vector tốc độ cao như FAISS.

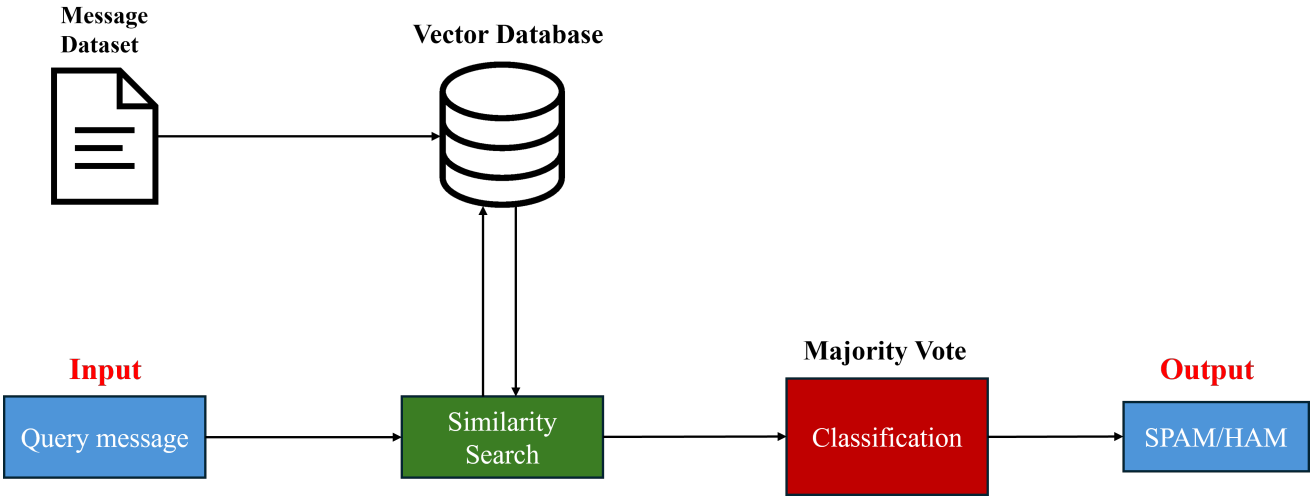
Theo đó, Input/Output chung của chương trình sẽ bao gồm:

- **Input:** Một đoạn tin nhắn (text).
- **Output:** Có là tin nhắn spam hay không (bool).

Dựa vào các thông tin trên, ta sẽ xây dựng được 2 luồng xử lý (pipeline) cho bài toán này như sau:



Hình 2: Pipeline tổng quan của chương trình phân loại tin nhắn với Naive Bayes.



Hình 3: Pipeline tổng quan của chương trình phân loại tin nhắn với Cơ sở dữ liệu Vector.

Theo đó, với bộ dữ liệu có nhãn về tin nhắn spam hoặc không spam, cả hai phương pháp đều trải qua các bước chuẩn bị dữ liệu cần thiết, bao gồm tiền xử lý và trích xuất đặc trưng. Sau khi dữ liệu được chuẩn bị, tùy thuộc vào phương pháp lựa chọn, chúng ta sẽ tiến hành xây dựng mô hình Naive Bayes Classifier hoặc tạo cơ sở dữ liệu vector để lưu trữ các biểu diễn ngữ nghĩa của tin nhắn. Cuối cùng, sử dụng mô hình hoặc cơ sở dữ liệu đã xây dựng, chương trình có thể dự đoán một tin nhắn bất kỳ có phải là spam hay không. Như vậy, các chương trình trong project của chúng ta đã hoàn tất.

Mục lục

II. Xây dựng chương trình phân loại tin nhắn Spam với Naive Bayes

Trong phần này, ta sẽ tiến hành cài đặt chương trình phân loại tin nhắn rác hay không với Naive Bayes. Chương trình được cài đặt trên Google Colab.

II.1. Tải và khám phá bộ dữ liệu

Đầu tiên, chúng ta cùng nhìn qua bộ dataset sẽ được sử dụng trong bài này thông qua bảng sau:

Category	Message
ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got a...
ham	Ok lar... Joking wif u oni...
spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entr...
spam	FreeMsg Hey there darling it"s been 3 week"s now and no word back! I"d like some fun you up for it s...
...	...

Quan sát thấy bộ dữ liệu sẽ gồm có 2 cột:

1. **Category:** gồm 2 nhãn là *Ham* và *Spam*, với ý nghĩa như sau:
    - *Ham:* Là những tin nhắn bình thường, không có mục đích quảng cáo hoặc lừa đảo hoặc nói cách khác là người nhận mong muốn nhận được.
    - *Spam:* Là những tin nhắn không mong muốn, thường có mục đích quảng cáo sản phẩm, dịch vụ, hoặc lừa đảo.
2. **Message:** là những nội dung bên trong một Messages.

Nhiệm vụ của chúng ta là dựa vào nội dung Message để phân loại nhị phân với Naive Bayes, để xem xét rằng, liệu với nội dung như thế này thì Message đó là *Spam* hay *Ham*. Để huấn luyện mô hình Naive Bayes giải quyết bài toán này, ta cần tải bộ dữ liệu này về máy. Chúng ta có thể tải tại đây hoặc trực tiếp tại trang Kaggle của dataset này tại đây. Trong python, với đường dẫn google drive của bộ dữ liệu, ta có thể dùng lệnh sau đây để tải về một cách tự động về máy:

```
# https://drive.google.com/file/d/1N7rk-kfnDFIGMeXOROVTjKh71gcgx-7R/view?usp=sharing
!gdown --id 1N7rk-kfnDFIGMeXOROVTjKh71gcgx-7R
```

II.2. Cài đặt và Import thư viện

Trước tiên, chúng ta cùng điểm qua một số thư viện chính được sử dụng trong bài:

- **string:** Cung cấp các hàm cơ bản để thao tác với chuỗi ký tự.
- **nltk (Natural Language Toolkit):** Một trong những thư viện xử lý ngôn ngữ tự nhiên phổ biến nhất trong Python.
- **pandas:** Cung cấp các cấu trúc dữ liệu hiệu quả và các công cụ để làm việc với dữ liệu.
- **numpy:** Cung cấp các đối tượng mảng đa chiều và các hàm toán học để làm việc với các mảng này.
- **scikit-learn:** Thư viện học máy phổ biến, giúp xây dựng và triển khai các mô hình học máy phức tạp một cách nhanh chóng.

Hình 4: Logo của một số thư viện được sử dụng trong project.

Trong môi trường code, các bạn thực thi đoạn code sau:

```
import string
import nltk
nltk.download("stopwords")
nltk.download("punkt")
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
```

II.3. Đọc và tách dữ liệu

Để đọc bộ dữ liệu có dạng file '.csv', chúng ta sẽ dùng thư viện **pandas**. Để tách riêng biệt phần đặc trưng và nhãn, sau khi có dataframe, chúng ta đọc và lưu trữ dữ liệu của từng cột vào 2 biến tương ứng `messages` và `labels`:

```
DATASET_PATH = "/content/2cls_spam_text_cls.csv"
df = pd.read_csv(DATASET_PATH)

```

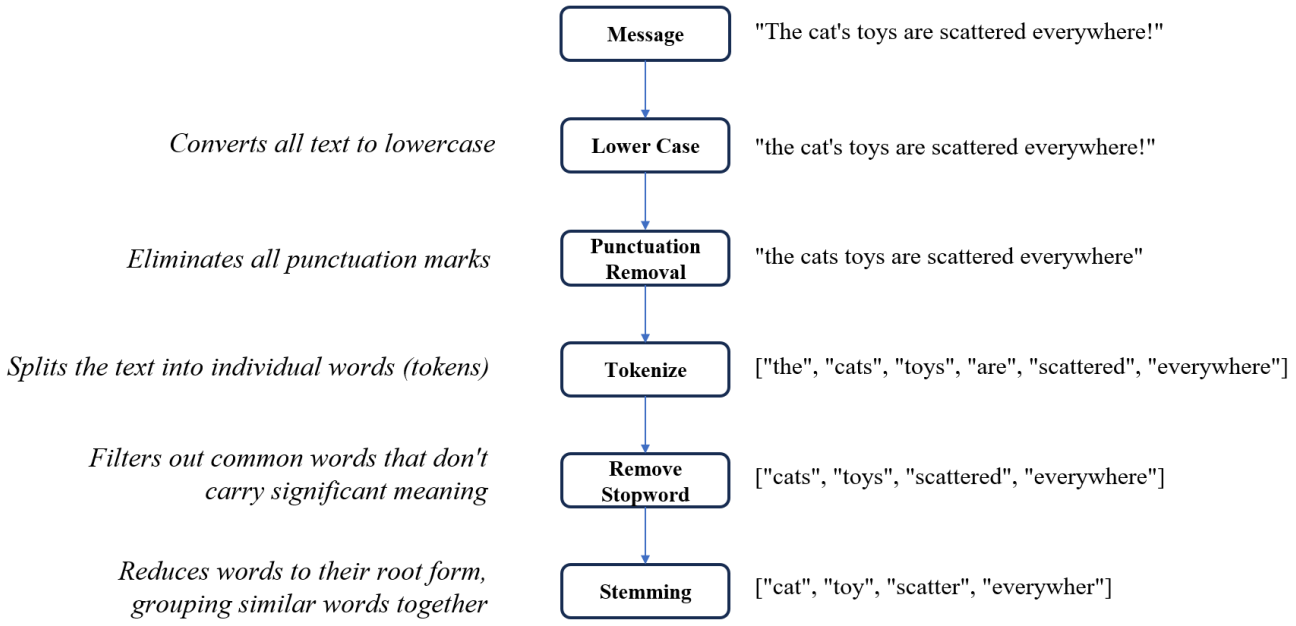
```
messages = df["Message"].values.tolist()
labels = df["Category"].values.tolist()
```

	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
...	...	...

Hình 5: Hiển thị một vài mẫu dữ liệu đầu tiên từ file csv.

II.4. Tiền xử lý dữ liệu

Tiền xử lý dữ liệu đặc trưng: Nội dung của các tin nhắn vô cùng đa dạng. Chúng có thể chứa từ viết tắt, dấu câu, các biến thể của từ, v.v. Vì vậy, bước tiền xử lý dữ liệu là vô cùng quan trọng. Chúng ta sẽ thực hiện một số bước xử lý cơ bản như sau:

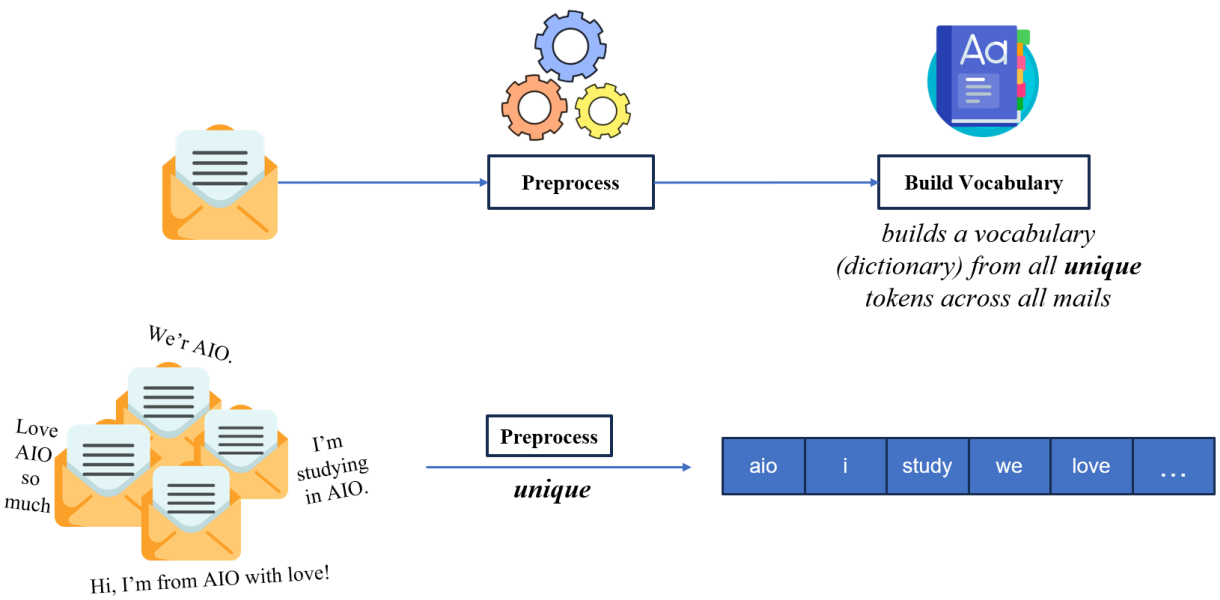


Hình 6: Minh họa các bước tiền xử lý văn bản.

Tương ứng với đoạn code sau:

```python
def lowercase(text):
    return # Your code here

def punctuation_removal(text):
    translator = # Your code here

    return text.translate(translator)

def tokenize(text):
    return # Your code here

def remove_stopwords(tokens):
    stop_words = # Your code here

    return [token for token in tokens if token not in stop_words]

def stemming(tokens):
    stemmer = # Your code here

    return [stemmer.stem(token) for token in tokens]

def preprocess_text(text):
    # Your code here

    return tokens

messages = [preprocess_text(message) for message in messages]
```

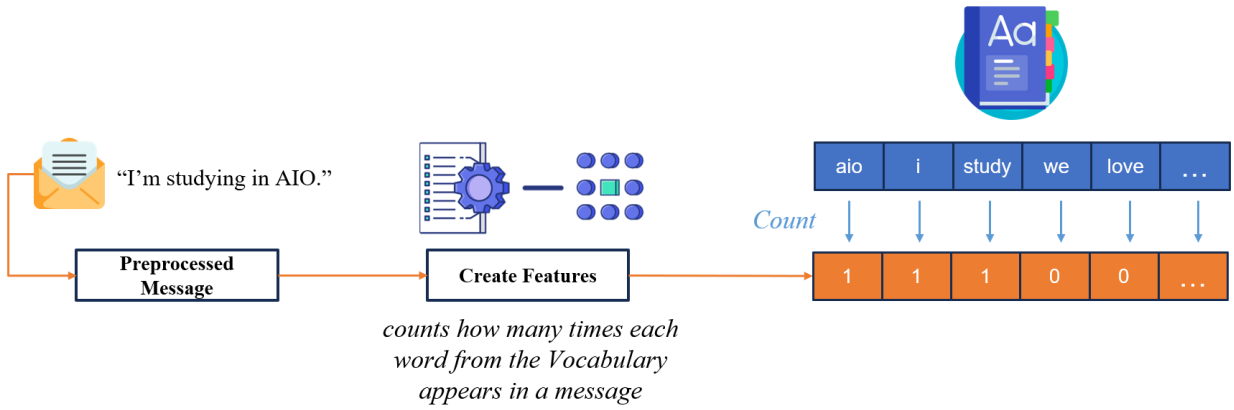
Tiếp theo, chúng ta cần tạo một bộ từ điển (Dictionary), chứa tất cả các từ có trong toàn bộ tin nhắn sau khi đã tiền xử lý (không tính trùng lặp).



Hình 7: Minh họa quá trình xây dựng từ điển từ các tin nhắn đã xử lý.

```python
def create_dictionary(messages):
    dictionary = []
    for tokens in messages:
        # Your code here
    return dictionary

dictionary = create_dictionary(messages)
```

Kế đến, chúng ta cần tạo ra những đặc trưng đại diện cho thông tin (là các từ) của các tin nhắn. Một trong những cách đơn giản nhất là dựa vào tần suất xuất hiện của từ (Bag-of-Words). Với mỗi tin nhắn, vector đại diện sẽ có kích thước bằng với số lượng từ có trong từ điển.



Hình 8: Minh họa quá trình tạo vector đặc trưng dựa trên tần suất từ.

```python
def create_features(tokens, dictionary):
    features = np.zeros(len(dictionary))
    for token in tokens:
        if token in dictionary:
            features[dictionary.index(token)] += 1
    return features

X = np.array([create_features(tokens, dictionary) for tokens in messages])
```

Tiền xử lý dữ liệu nhãn: Đối với nhãn, chúng ta sẽ chuyển 2 nhãn *ham* và *spam* thành các con số *0* và *1* để máy tính có thể hiểu.

```python
le = LabelEncoder()
y = le.fit_transform(labels)
print(f"Classes: {le.classes_}")
print(f"Encoded labels: {y}")

# >> Classes: ["ham" "spam"]
# >> Encoded labels: [0 0 1 ... 0 0 0]
```

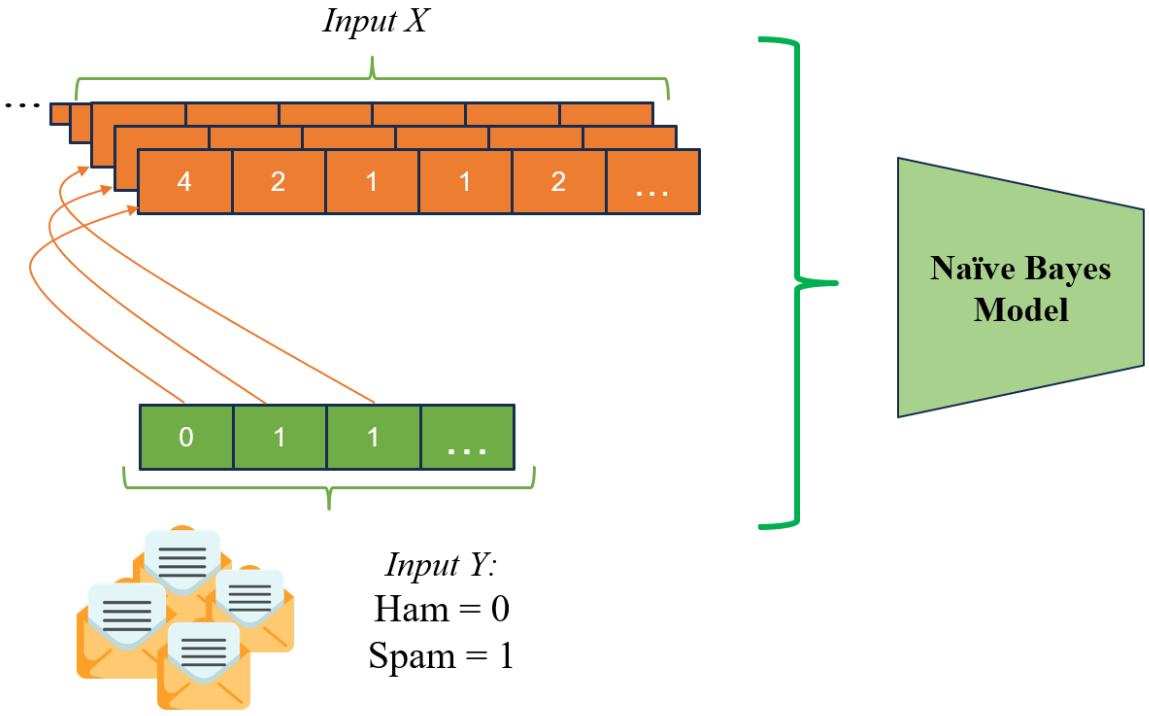
II.5. Chia bộ dữ liệu

Chúng ta sẽ chia bộ dữ liệu thành 3 phần: Train, Validation và Test theo tỉ lệ lần lượt là 70%, 20% và 10%.

```
VAL_SIZE = 0.2
TEST_SIZE = 0.125  # 0.1 / (1-0.2)
SEED = 0

X_train, X_val, y_train, y_val = train_test_split(X, y,
                                                  test_size=VAL_SIZE,
                                                  shuffle=True,
                                                  random_state=SEED)

X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,
                                                    test_size=TEST_SIZE,
                                                    shuffle=True,
                                                    random_state=SEED)
```

II.6. Huấn luyện mô hình

Sau các bước trên, chúng ta chỉ cần truyền dữ liệu đã xử lý vào mô hình Gaussian Naive Bayes và tiến hành huấn luyện.



Hình 9: Đầu vào cho mô hình Naive Bayes.

Thực thi đoạn code sau để khởi tạo và huấn luyện mô hình:

```
model = GaussianNB()
print("Start training...")
model = # Your code here
print("Training completed!")

# >> Start training...
# >> Training completed!
```

II.7. Đánh giá mô hình

Sau khi huấn luyện, chúng ta đánh giá hiệu suất của mô hình trên tập Validation và Test bằng độ đo Accuracy.

```
y_val_pred = model.predict(X_val)
y_test_pred = model.predict(X_test)

val_accuracy = accuracy_score(y_val, y_val_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

print(f"Val accuracy: {val_accuracy:.4f}")
print(f"Test accuracy: {test_accuracy:.4f}")

# >> Val accuracy: 0.8816
# >> Test accuracy: 0.8602
```

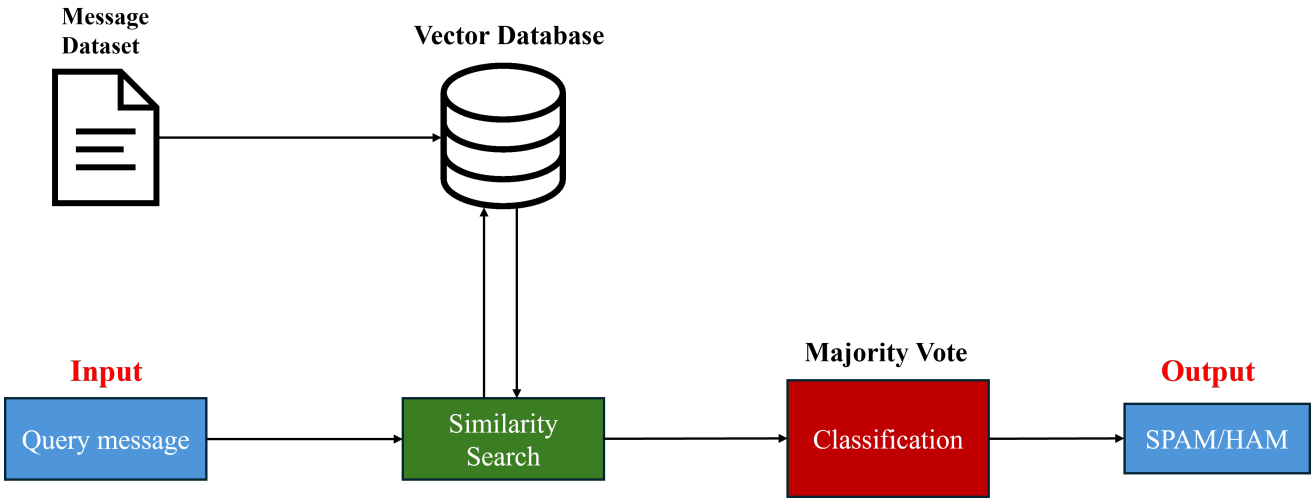
II.8. Thực hiện dự đoán

Cuối cùng, để sử dụng mô hình cho các tin nhắn mới, chúng ta sẽ phải thực hiện lại các công đoạn tiền xử lý, tạo đặc trưng và truyền vào mô hình để dự đoán.

```
def predict(text, model, dictionary, label_encoder):
    processed_text = preprocess_text(text)
    features = create_features(processed_text, dictionary)
    features = np.array(features).reshape(1, -1)
    prediction = model.predict(features)
    prediction_cls = label_encoder.inverse_transform(prediction)[0]
    return prediction_cls

test_input = "I am actually thinking a way of doing something useful"
prediction_cls = predict(test_input, model, dictionary, le)
print(f"Prediction: {prediction_cls}")

# >> Prediction: ham
```

III. Xây dựng chương trình phân loại tin nhắn Spam với Cơ sở dữ liệu Vector

Trong phần này, ta sẽ tiến hành xây dựng một chương trình phân loại tin nhắn spam (tin rác) và ham (tin thường) bằng một phương pháp hiện đại: sử dụng sentence embeddings và cơ sở dữ liệu vector. Thay vì dùng các mô hình học máy truyền thống như Naive Bayes, chúng ta sẽ chuyển đổi mỗi tin nhắn thành một vector số đại diện cho ngữ nghĩa của nó. Sau đó, việc phân loại một tin nhắn mới sẽ dựa trên việc tìm kiếm các tin nhắn tương tự nhất trong cơ sở dữ liệu đã có và lấy nhãn của chúng. Phương pháp này rất mạnh mẽ và hiệu quả, đặc biệt với sự hỗ trợ của thư viện `FAISS` từ Meta AI.



Hình 10: Chương trình phân loại tin nhắn Spam với Cơ sở dữ liệu Vector.

III.1. Thiết lập môi trường và Tải dữ liệu

Bước đầu tiên là chuẩn bị môi trường làm việc. Thao tác này bao gồm việc tải bộ dữ liệu từ Google Drive bằng lệnh `gdown` và import tất cả các thư viện cần thiết. Các thư viện chính bao gồm:

- **pandas, numpy:** Dành cho việc xử lý và thao tác dữ liệu.
- **torch, transformers:** Để tải và sử dụng mô hình embedding ngôn ngữ từ Hugging Face.
- **faiss:** Thư viện cho việc tìm kiếm tương đồng trên vector một cách hiệu quả.
- **sklearn:** Dùng để chia dữ liệu và mã hóa nhãn.
- **tqdm:** Để hiển thị thanh tiến trình (progress bar) khi xử lý các tác vụ tốn thời gian.

Hình 11: Facebook AI Similarity Search (FAISS)

```
!pip install -qq faiss-cpu
!pip install -qq transformers
!pip install -qq pandas
!pip install -qq numpy
!pip install -qq scikit-learn
!pip install -qq tqdm

# Tải dữ liệu từ Google Drive
# https://drive.google.com/file/d/1N7rk-kfnDFIGMeXOROVTjKh71gcgx-7R/view?usp=sharing
!gdown --id 1N7rk-kfnDFIGMeXOROVTjKh71gcgx-7R

# 1. Import các thư viện cần thiết
import pandas as pd
import numpy as np
import torch
import torch.nn.functional as F
import faiss
from transformers import AutoTokenizer, AutoModel
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tqdm import tqdm
```

III.2. Đọc và Chuẩn bị Dữ liệu

Sau khi tải file về môi trường, chúng ta sử dụng thư viện `pandas` để đọc file `.csv` vào một cấu trúc dữ liệu gọi là DataFrame. Từ DataFrame này, chúng ta sẽ tách riêng cột chứa nội dung tin nhắn (`"Message"`) và cột chứa nhãn phân loại (`"Category"`) ra thành hai danh sách (list) riêng biệt để thuận tiện cho các bước xử lý và vector hóa sau này.

```
# 2. Đọc bộ dữ liệu
DATASET_PATH = "/content/2cls_spam_text_cls.csv"
df = pd.read_csv(DATASET_PATH)

# Tách tin nhắn và nhãn vào các list
messages = df["Message"].values.tolist()
```

```
labels = df["Category"].values.tolist()
```

III.3. Chuẩn bị Mô hình Embedding

Đây là bước cốt lõi để chuyển đổi văn bản thành các vector (embedding). Chúng ta tải mô hình ngôn ngữ `intfloat/multilingual-e5-base` từ Hugging Face. Đây là một mô hình mạnh mẽ, được huấn luyện để tạo ra các vector (embedding) đại diện ngữ nghĩa cho văn bản. Đoạn code cũng tự động phát hiện và ưu tiên sử dụng GPU (`cuda`) nếu có để tăng tốc tính toán. Hàm `average_pool` được định nghĩa để tính toán vector đại diện chung cho cả câu từ các vector output của mô hình transformer, một kỹ thuật phổ biến để có được sentence embedding chất lượng.

```
# 3.1. Load mô hình embedding
MODEL_NAME = "intfloat/multilingual-e5-base"
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
model = AutoModel.from_pretrained(MODEL_NAME)

# Set device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)
model.eval()

# Hàm để trích xuất embedding từ output của model
def average_pool(last_hidden_states, attention_mask):
    last_hidden = last_hidden_states.masked_fill(
        ~attention_mask[..., None].bool(), 0.0
        )
    return last_hidden.sum(dim=1) / attention_mask.sum(dim=1)[..., None]
```

III.4. Vector hóa Dữ liệu và Tạo Metadata

Ở bước này, chúng ta định nghĩa hàm `get_embeddings` để thực hiện quá trình vector hóa toàn bộ dữ liệu văn bản. Hàm này xử lý các tin nhắn theo từng lô (batch) để tối ưu hiệu năng. Một kỹ thuật quan trọng được áp dụng là thêm tiền tố `"passage: "` vào mỗi tin nhắn, giúp cải thiện chất lượng embedding cho mô hình E5. Sau khi nhận được các vector, chúng ta chuẩn hóa (normalize) chúng.

Song song đó, các nhãn dạng chữ (‘ham‘/‘spam‘) được mã hóa thành số (‘0‘/‘1‘) bằng `LabelEncoder`. Một bộ `metadata` (dữ liệu mô tả) cũng được tạo ra để lưu trữ thông tin gốc của mỗi tin nhắn, giúp liên kết các vector trở lại với dữ liệu ban đầu một cách dễ dàng.

```
# 3.2. Tạo sentence embeddings
def get_embeddings(texts, model, tokenizer, device, batch_size=32):
    """Tạo embeddings cho một danh sách các văn bản"""
    embeddings = []
```

```
    for i in tqdm(range(0, len(texts), batch_size), desc="Generating embeddings"):
        batch_texts = texts[i:i+batch_size]
        batch_texts_with_prefix = [f"passage: {text}" for text in batch_texts]

        batch_dict = tokenizer(batch_texts_with_prefix, max_length=512,
                               padding=True, truncation=True, return_tensors="pt")
        batch_dict = {k: v.to(device) for k, v in batch_dict.items()}

        with torch.no_grad():
            outputs = model(**batch_dict)
            batch_embeddings = average_pool(outputs.last_hidden_state, batch_dict["
                                            attention_mask"])
            batch_embeddings = F.normalize(batch_embeddings, p=2, dim=1)
            embeddings.append(batch_embeddings.cpu().numpy())

    return np.vstack(embeddings)

# Chuẩn bị nhãn
le = LabelEncoder()
y = le.fit_transform(labels)

# Tạo embeddings cho tất cả tin nhắn
X_embeddings = get_embeddings(messages, model, tokenizer, device)

# Tạo metadata cho mỗi tài liệu
metadata = [{"index": i, "message": message, "label": label, "label_encoded": y[i]}
            for i, (message, label) in enumerate(zip(messages, labels))]
```

III.5. Xây dựng Cơ sở dữ liệu Vector và Chia Dữ liệu

Tại đây, chúng ta chia bộ dữ liệu đã được vector hóa thành hai tập: 90% cho huấn luyện (training) và 10% cho kiểm thử (testing) bằng cách sử dụng các chỉ số (indices). Việc chia theo `stratify=y` đảm bảo tỉ lệ spam/ham trong hai tập là tương đương nhau, tránh sai lệch.

Sau đó, chúng ta tạo một cơ sở dữ liệu vector hiệu năng cao bằng thư viện `FAISS`. Chúng ta khởi tạo một chỉ mục `IndexFlatIP`, sử dụng phép toán Tích Vô hướng (Inner Product) để đo độ tương đồng. Chỉ các vector của tập huấn luyện được thêm vào cơ sở dữ liệu này để phục vụ cho việc tìm kiếm và phân loại sau này.

```
# 3.3. Tạo FAISS index và chia dữ liệu
TEST_SIZE = 0.1
SEED = 42

train_indices, test_indices = train_test_split(
    range(len(messages)), test_size=TEST_SIZE, stratify=y, random_state=SEED
)

# Tách embeddings và metadata theo chỉ số đã chia
X_train_emb = X_embeddings[train_indices]
X_test_emb = X_embeddings[test_indices]
```

```
train_metadata = [metadata[i] for i in train_indices]
test_metadata = [metadata[i] for i in test_indices]

# Tạo FAISS index
embedding_dim = X_train_emb.shape[1]
index = faiss.IndexFlatIP(embedding_dim)
index.add(X_train_emb.astype("float32"))
```

III.6. Xây dựng Logic Phân loại và Đánh giá

Đoạn code này định nghĩa hai hàm cốt lõi cho việc phân loại và đánh giá:

1. **`classify_with_knn`:** Đây là hàm thực hiện việc phân loại. Nó nhận một văn bản mới (query), tạo embedding cho nó (với tiền tố `"query: "`), sau đó dùng chỉ mục FAISS để tìm kiếm k tin nhắn giống nhất trong tập huấn luyện. Nhãn của tin nhắn mới được quyết định dựa trên nhãn chiếm đa số (majority vote) của k "hàng xóm"này.

2. **`evaluate_knn_accuracy`:** Hàm này được dùng để đánh giá độ chính xác của phương pháp trên tập test. Nó duyệt qua từng mẫu trong tập test, phân loại chúng, so sánh với nhãn thật và tính toán độ chính xác tổng thể.

```
# 4. Triển khai phân loại với embedding similarity
def classify_with_knn(query_text, model, tokenizer, device, index, train_metadata, k=1)
                                                    :
    """Classify text using k-nearest neighbors with embeddings"""

    # Get query embedding
    query_with_prefix = f"query: {query_text}"
    batch_dict = tokenizer([query_with_prefix],
                           max_length=512,
                           padding=True,
                           truncation=True,
                           return_tensors="pt")

    batch_dict = {k: v.to(device) for k, v in batch_dict.items()}

    with torch.no_grad():
        outputs = model(**batch_dict)
        query_embedding = average_pool(outputs.last_hidden_state, batch_dict["
                                               attention_mask"])
        query_embedding = F.normalize(query_embedding, p=2, dim=1)
        query_embedding = query_embedding.cpu().numpy().astype("float32")

    # Search in FAISS index
    scores, indices = index.search(query_embedding, k)

    # Get predictions from top-k neighbors
    predictions = []
    neighbor_info = []

```

```
    for i in range(k):
        neighbor_idx = indices[0][i]
        neighbor_score = scores[0][i]
        neighbor_label = train_metadata[neighbor_idx]["label"]
        neighbor_message = train_metadata[neighbor_idx]["message"]

        predictions.append(neighbor_label)
        neighbor_info.append({
            "score": float(neighbor_score),
            "label": neighbor_label,
            "message": neighbor_message[:100] + "..." if len(neighbor_message) > 100
                                                else neighbor_message
        })

    # Majority vote for final prediction
    unique_labels, counts = np.unique(predictions, return_counts=True)
    final_prediction = unique_labels[np.argmax(counts)]

    return final_prediction, neighbor_info

def evaluate_knn_accuracy(test_embeddings, test_labels, test_metadata, index,
                                                train_metadata, k_values=[1, 3, 5]):
    """Evaluate accuracy for different k values using precomputed embeddings"""
    results = {}
    all_errors = {}

    for k in k_values:
        correct = 0
        total = len(test_embeddings)
        errors = []

        for i in tqdm(range(total), desc=f"Evaluating k={k}"):
            query_embedding = test_embeddings[i:i+1].astype("float32")
            true_label = test_metadata[i]["label"]
            true_message = test_metadata[i]["message"]

            # Search in FAISS index
            scores, indices = index.search(query_embedding, k)

            # Get predictions from top-k neighbors
            predictions = []
            neighbor_details = []
            for j in range(k):
                neighbor_idx = indices[0][j]
                neighbor_label = train_metadata[neighbor_idx]["label"]
                neighbor_message = train_metadata[neighbor_idx]["message"]
                neighbor_score = float(scores[0][j])

                predictions.append(neighbor_label)
                neighbor_details.append({
                    "label": neighbor_label,
                    "message": neighbor_message,
                    "score": neighbor_score
                })
```

```
            # Majority vote
            unique_labels, counts = np.unique(predictions, return_counts=True)
            predicted_label = unique_labels[np.argmax(counts)]

            if predicted_label == true_label:
                correct += 1
            else:
                # Collect error information
                error_info = {
                    "index": i,
                    "original_index": test_metadata[i]["index"],
                    "message": true_message,
                    "true_label": true_label,
                    "predicted_label": predicted_label,
                    "neighbors": neighbor_details,
                    "label_distribution": {label: int(count) for label, count in zip(
                                            unique_labels, counts)}
                }
                errors.append(error_info)

        accuracy = correct / total
        error_count = total - correct

        results[k] = accuracy
        all_errors[k] = errors

        print(f"Accuracy with k={k}: {accuracy:.4f}")
        print(f"Number of errors with k={k}: {error_count}/{total} ({(error_count/total
                                        )*100:.2f}%)")

    return results, all_errors
```

III.7. Đánh giá Mô hình trên Tập Test

Bây giờ, chúng ta áp dụng hàm `evaluate_knn_accuracy` đã xây dựng lên tập dữ liệu test. Việc này nhằm mục đích đo lường hiệu năng thực tế của phương pháp phân loại trên những dữ liệu mà mô hình chưa từng thấy. Chúng ta thử nghiệm với các giá trị `k` khác nhau (1, 3, và 5) để quan sát xem việc thay đổi số lượng "hàng xóm"ảnh hưởng đến độ chính xác như thế nào. Kết quả cuối cùng sẽ được in ra màn hình và một file JSON chi tiết về các trường hợp dự đoán sai cũng được lưu lại để có thể phân tích sâu hơn.

```
# 5. Đánh giá accuracy trên test set
%%time
print("Evaluating accuracy on test set...")
accuracy_results, error_results = evaluate_knn_accuracy(
    X_test_emb,
    y_test,
    test_metadata,
```

```
    index,
    train_metadata,
    k_values=[1, 3, 5]
)

# Hiển thị kết quả
print("\n" + "="*50)
print("ACCURACY RESULTS")
print("="*50)
for k, accuracy in accuracy_results.items():
    print(f"Top-{k} accuracy: {accuracy:.4f} ({accuracy*100:.2f}%)")
print("="*50)

# Lưu phân tích lỗi ra file
import json
from datetime import datetime

error_analysis = {
    "timestamp": datetime.now().isoformat(),
    "model": MODEL_NAME,
    "test_size": len(X_test_emb),
    "accuracy_results": accuracy_results,
    "errors_by_k": {}
}

for k, errors in error_results.items():
    error_analysis["errors_by_k"][f"k_{k}"] = {
        "total_errors": len(errors),
        "error_rate": len(errors) / len(X_test_emb),
        "errors": errors
    }

# Lưu JSON file ghi lỗi
output_file = "error_analysis.json"
with open(output_file, "w", encoding="utf-8") as f:
    json.dump(error_analysis, f, ensure_ascii=False, indent=2)

print(f"\n***Error analysis saved to: {output_file}***")
print()
print(f"***Summary:")
for k, errors in error_results.items():
    print(f"   k={k}: {len(errors)} errors out of {len(X_test_emb)} samples")
```

III.8. Xây dựng Pipeline Phân loại Hoàn chỉnh

Để thuận tiện cho việc sử dụng và tái sử dụng, chúng ta xây dựng hàm `spam_classifier_pipeline`. Hàm này đóng gói toàn bộ quy trình, từ việc nhận một văn bản đầu vào của người dùng, tạo embedding, tìm kiếm trong FAISS, cho đến việc đưa ra dự đoán cuối cùng và hiển thị kết quả. Pipeline này không chỉ trả về nhãn dự đoán mà còn cung cấp thông tin về các "hàng xóm"gần nhất đã được dùng để đưa ra quyết định đó, giúp tăng tính minh bạch và dễ hiểu cho người dùng.

```python
# 6. Pipeline classification cho user input
def spam_classifier_pipeline(user_input, k=3):
    """
    Complete pipeline for spam classification

    Args:
        user_input (str): Text to classify
        k (int): Number of nearest neighbors to consider

    Returns:
        dict: Classification results with details
    """

    print()
    print(f"***Classifying: "{user_input}"")
    print()
    print(f"***Using top-{k} nearest neighbors")
    print()

    # Get prediction and neighbors
    prediction, neighbors = classify_with_knn(
        user_input, model, tokenizer, device, index, train_metadata, k=k
    )

    # Display results
    print(f"***Prediction: {prediction.upper()}")
    print()

    print("***Top neighbors:")
    for i, neighbor in enumerate(neighbors, 1):
        print(f"{i}. Label: {neighbor["label"]} | Score: {neighbor["score"]:.4f}")
        print(f"   Message: {neighbor["message"]}")
        print()

    # Count label distribution
    labels = [n["label"] for n in neighbors]
    label_counts = {label: labels.count(label) for label in set(labels)}

    return {
        "prediction": prediction,
        "neighbors": neighbors,
        "label_distribution": label_counts
    }
```

III.9. Kiểm thử Pipeline

Đây là bước cuối cùng để xác nhận rằng pipeline của chúng ta hoạt động chính xác. Chúng ta kiểm tra nó với một danh sách các ví dụ có sẵn, bao gồm cả tin nhắn thường và tin nhắn spam điển hình. Ngoài ra, một khối code "tương tác"được cung cấp, cho phép người dùng có thể dễ dàng thay đổi nội dung tin nhắn trong biến `user_text` và giá trị `k` trong biến `k_value` để tự mình thử nghiệm với các trường hợp khác nhau.

```python
# 7. Test pipeline với các ví dụ khác nhau
test_examples = [
    "I am actually thinking a way of doing something useful",
    "FREE!! Click here to win \$1000 NOW! Limited time offer!",
]
for i, example in enumerate(test_examples, 1):
    print(f"\n--- Example {i}: "{example}" ---")
    result = spam_classifier_pipeline(example, k=3)

# Interactive testing - người dùng có thể thay đổi text và k value
print("\n--- Interactive Testing ---")
user_text = "Win a free iPhone! Click here now!"
k_value = 5
result = spam_classifier_pipeline(user_text, k=k_value)
```

IV. Câu hỏi trắc nghiệm

1. Khi sử dụng `faiss.IndexFlatIP` với các embedding đã được chuẩn hóa L2-normalized, phép đo độ tương đồng "Inner Product"thực chất tương đương với phép đo nào sau đây?

   (A) Khoảng cách Euclidean.

   (B) Khoảng cách Manhattan.

   (C) Độ tương đồng Cosine.

   (D) Khoảng cách Chebyshev.

2. Mục đích chính của việc thêm tiền tố `"passage: "` cho tin nhắn trong `get_embeddings` và `"query: "` cho đầu vào trong `classify_with_knn` là gì?

   (A) Để tăng độ dài của chuỗi đầu vào, giúp mô hình học tốt hơn.

   (B) Để hướng dẫn mô hình E5 tạo ra embedding phù hợp với ngữ cảnh tìm kiếm và truy vấn.

   (C) Để phân biệt rõ ràng giữa tin nhắn gốc và tin nhắn được tạo ra trong quá trình xử lý.

   (D) Để giảm thiểu số lượng token và tăng tốc độ xử lý của tokenizer.

3. Tại sao `stratify=y` lại quan trọng khi chia dữ liệu với `train_test_split` trong dự án phân loại Spam này?

   (A) Đảm bảo dữ liệu được chia đều theo thứ tự alphabet của nhãn.

   (B) Giúp tăng tốc quá trình chia dữ liệu.

   (C) Để chia ngẫu nhiên dữ liệu một cách tối ưu.

   (D) Để duy trì tỉ lệ các lớp (spam/ham) giống nhau giữa tập huấn luyện và tập kiểm thử.

4. Nếu một tin nhắn vượt quá `max_length=512` trong quá trình token hóa và được `truncation=True`, điều gì có thể xảy ra?

   (A) Mô hình sẽ tự động điều chỉnh để xử lý toàn bộ tin nhắn.

   (B) Tin nhắn sẽ bị loại bỏ hoàn toàn khỏi bộ dữ liệu.

   (C) Thông tin ngữ cảnh quan trọng ở cuối tin nhắn có thể bị mất, ảnh hưởng đến chất lượng embedding.

   (D) Mô hình sẽ báo lỗi vì không thể xử lý chuỗi quá dài.

5. Hàm `average_pool` sau khi lấy `last_hidden_states` sử dụng `masked_fill` với `attention_mask[..., None].bool()`. Mục đích của việc này là gì?

   (A) Để đảm bảo rằng chỉ các token quan trọng nhất được giữ lại.

   (B) Để loại bỏ các token không có ý nghĩa trong câu.

   (C) Để thêm các giá trị ngẫu nhiên vào các vị trí bị mask.

(D) Để gán giá trị 0.0 cho các vị trí bị padding (mask) trước khi tính tổng, tránh ảnh hưởng đến average.

6. Việc tạo ra `metadata` (bao gồm `index`, `message`, `label`, `label_encoded`) trong dự án có ý nghĩa quan trọng nhất nào đối với khả năng phân tích và debug?

   (A) Giúp giảm kích thước bộ nhớ lưu trữ embeddings.

   (B) Tăng tốc độ tìm kiếm trong FAISS.

   (C) Cho phép dễ dàng liên kết lại các embedding với tin nhắn và nhãn gốc, đặc biệt khi phân tích lỗi.

   (D) Là yêu cầu bắt buộc của thư viện FAISS.

7. Tại sao việc tăng giá trị `k` (số lượng hàng xóm) trong K-NN từ 1 lên 3 hoặc 5 đôi khi có thể cải thiện độ chính xác của phân loại?

   (A) Vì mô hình có nhiều dữ liệu hơn để học.

   (B) Vì việc lấy ý kiến đa số từ nhiều hàng xóm giúp giảm ảnh hưởng của nhiễu hoặc sai lệch từ một vài điểm dữ liệu đơn lẻ.

   (C) Vì nó làm tăng tốc độ tìm kiếm trong cơ sở dữ liệu vector.

   (D) Vì nó đơn giản hóa ranh giới quyết định giữa các lớp.

8. Trong hàm `classify_with_knn`, tại sao `query_embedding` sau khi được tạo phải được chuyển đổi thành `astype("float32")` trước khi gọi `index.search` của FAISS?

   (A) `float32` yêu cầu ít bộ nhớ hơn, giúp tiết kiệm tài nguyên.

   (B) FAISS yêu cầu kiểu dữ liệu `float32` cho các vector đầu vào để hoạt động hiệu quả.

   (C) Để giảm độ chính xác của embedding, giúp phân loại nhanh hơn.

   (D) Đây là một bước tùy chọn, không ảnh hưởng đến kết quả.

9. Giả sử một tin nhắn Spam thực tế được phân loại nhầm thành "Ham". Dựa vào `error_analysis.json` được tạo ra, thông tin nào sau đây sẽ hữu ích nhất để hiểu nguyên nhân gây ra lỗi này?

   (A) `timestamp` và `total_errors`.

   (B) `model` và `test_size`.

   (C) `accuracy_results` cho các giá trị `k` khác nhau.

   (D) `neighbors` và `label_distribution` của các hàng xóm gần nhất.

10. Mặc dù mạnh mẽ, một nhược điểm tiềm tàng của hệ thống phân loại Spam dựa trên tìm kiếm tương đồng vector (k-NN) là gì, đặc biệt khi dữ liệu huấn luyện rất lớn?

    (A) Khó khăn trong việc hiểu ngữ nghĩa của tin nhắn.

    (B) Không thể phát hiện tin nhắn spam mới.

    (C) Chi phí tính toán và bộ nhớ cao cho việc tìm kiếm k-NN trong cơ sở dữ liệu lớn (mặc dù FAISS giúp giảm thiểu).

    (D) Không thể áp dụng cho các ngôn ngữ khác ngoài tiếng Anh.

Phụ lục

1. **Hint:** Các file code gợi ý và dữ liệu được lưu trong thư mục có thể được tải tại đây.

2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại đây (Lưu ý: Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).

3. **Kiến thức cần cho project:**

   Để hoàn thành và hiểu rõ phần project một cách hiệu quả nhất, các học viên cần nắm rõ các kiến thức được trang bị

   - Kiến thức lập trình Python và xử lý chuỗi
   - Hiểu về cơ sở dữ liệu vector (vector database) ở phần project 2.1
   - Cách thao tác và xử lý với numpy, xác suất thống kê, đại số tuyến tính như độ tương đồng cosine, ...

4. **Đề xuất phương pháp cải tiến project:**

   Project tiếp cận bài toán phân loại email theo hướng sử dụng các phương pháp đơn giản để biểu diễn vector và naive bayes hoặc sử dụng vector database. Để tiếp tục cải tiến, tối ưu hệ thống; nhóm tác biên soạn gợi ý một số phương pháp như sau:

   - Tiền xử lý nâng cao: Sử dụng các kỹ thuật tiền xử lý nâng cao hơn như loại bỏ từ ngữ ít xuất hiện (rare words), chuẩn hoá cú pháp (normalization), phát hiện và xử lý các từ phủ định (negation handling) để cải thiện độ chính xác phân loại
   - Kỹ thuật biểu diễn văn bản khác: Thử nghiệm phương pháp TF-IDF thay cho Bag of Words để giảm trọng số các từ phổ biến và nhấn mạnh các từ quan trọng trong ngữ cảnh cảm xúc
   - Tăng cường dữ liệu (Data Augmentation): Áp dụng kỹ thuật tăng cường dữ liệu bằng cách thay thế từ đồng nghĩa, đảo vị trí câu, hoặc sinh dữ liệu mới để làm phong phú tập huấn luyện
   - Đánh giá mô hình đa chỉ số: Áp dụng thêm các chỉ số đánh giá như F1-score, ROC-AUC, confusion matrix để có cái nhìn toàn diện hơn về hiệu quả mô hình, thay vì chỉ dựa vào accuracy
   - Tri thức cho truy vấn: Bổ sung thêm những dạng tri thức biểu diễn khác cho văn bản hoặc kết hợp thêm các phương pháp tìm kiếm khác nhau như Bm25 kết hợp cosine similarity,...

   Trên đây là một số gợi ý, giúp học viện có thể cải tiến thêm hệ thống để đạt hiểu quả tốt hơn về cả tốc độ xử lý và độ chính xác.

5. Rubric:

Mục	Kiến Thức	Đánh Giá
II.	- Lý thuyết về bài toán Text Classification trong Machine Learning. - Lý thuyết về mô hình Naive Bayes. - Kiến thức về lập trình python cơ bản và cách sử dụng các thư viện liên quan đến machine learning cơ bản như: pandas, scikit-learn. - Các bước tiền xử lý dữ liệu văn bản trong Natural Language Processing. - Quy trình cơ bản của một chương trình huấn luyện mô hình Machine Learning.	- Nắm được khái niệm về bài toán Text Classification trong Machine Learning. - Hiểu được cách mô hình Naive Bayes có thể giải quyết bài toán phân loại văn bản. - Có khả năng sử dụng Python để xây dựng một mô hình machine learning cơ bản nhằm giải quyết bài toán phân loại. - Nắm được các kỹ thuật tiền xử lý văn bản cơ bản trong Natural Language Processing, phục vụ cho bài toán phân loại văn bản. - Hiểu được các bước làm cơ bản để huấn luyện một mô hình machine learning.
III.	- Áp dụng vector database để giải quyết bài toán trên.	- Thực thi thông qua thư viện FAISS cho vector database.