TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN GIỮA KỲ MÔN

NHẬP MÔN XỬ LÝ NGÔN NGỮ TỰ NHIÊN

BÁO CÁO GIỮA KỲ

Người hướng dẫn: **PGS TS LÊ ANH CƯỜNG**

Người thực hiện: **NGUYỄN VĂN TUẤN - 52200204**

NGUYỄN DUY HÀ VỸ - 52200180

NGUYỄN CÔNG QUANG - 52200177

Khoá **: 26**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2025

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN GIỮA KỲ MÔN

NHẬP MÔN XỬ LÝ NGÔN NGỮ TỰ NHIÊN

BÁO CÁO GIỮA KỲ

Người hướng dẫn: **PGS TS LÊ ANH CƯỜNG**

Người thực hiện: **NGUYỄN VĂN TUẤN - 52200204**

NGUYỄN DUY HÀ VỸ - 52200180

NGUYỄN CÔNG QUANG - 52200177

Khoá **: 26**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2025

LỜI CẢM ƠN

Đầu tiên, chúng tôi xin gửi lời cảm ơn chân thành đến Ban giám hiệu trường Đại Học Tôn Đức Thắng vì đã tạo mọi điều kiện tốt nhất về cơ sở vật chất và hệ thống thư viện hiện đại, đa dạng các loại tài liệu, sách báo giúp chúng tôi có thể tìm kiếm, nghiên cứu thông tin một cách thuận tiện và chính xác nhất để hoàn thành bài báo cáo.

Chúng tôi xin chân thành cảm ơn PGS TS. Lê Anh Cường đã truyền đạt kiến thức một cách tận tình và chi tiết, giúp tôi đủ nền tảng và kiến thức để vận dụng vào việc viết bài báo cáo này.

ĐỒ ÁN ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Chúng tôi xin cam đoan đây là sản phẩm đồ án của chúng tôi và được sự hướng dẫn của PGS TS Lê Anh Cường. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 21 tháng 3 năm 2025

Tác giả

(ký tên và ghi rõ họ tên)

Nguyễn Văn Tuấn

Nguyễn Duy Hà Vỹ

Nguyễn Công Quang

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

Phần đánh giá của GV chấm bài

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

TÓM TẮT

Trình bày về các phương pháp học máy cho bài toán document classification và ứng dụng cho một tập dữ liệu phân loại nhãn để so sánh các phương pháp.

MỤC LỤC

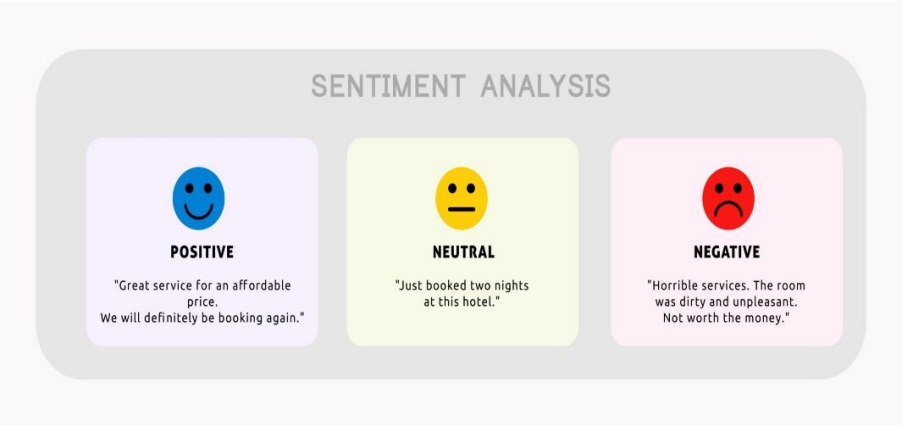
1. Giới thiệu bài toán Document Classification

Bài toán Phân loại văn bản (Document Classification) là một trong những bài toán quan trọng trong lĩnh vực xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP) và học máy (Machine Learning), nhằm mục đích phân loại các tài liệu hoặc đoạn văn bản vào các danh mục hoặc nhãn (label) cụ thể dựa trên nội dung của chúng. Nói đơn giản, đây là quá trình tự động gán một nhãn hoặc nhóm nhãn cho văn bản dựa trên đặc điểm ngôn ngữ và ý nghĩa của nó. Có thể kể đến với một vài ví dụ như sau:

- **Phân loại cảm xúc (Sentiment Analysis):** Xác định một bài đánh giá là "tích cực", "tiêu cực" hay "trung lập".
- **Phân loại thư rác (Spam Detection):** Phân biệt email là "thư rác" (spam) hay "không phải thư rác" (not spam).
- **Phân loại chủ đề:** Gán nhãn cho một bài báo là "thể thao", "chính trị", "kinh tế", v.v.
- Bài toán sẽ gồm có 3 bước cơ bản:
    - **Tiền xử lý dữ liệu:** Văn bản được làm sạch (loại bỏ dấu câu không cần thiết, chuyển về chữ thường, tách từ, v.v.) và chuyển đổi thành dạng số (ví dụ: vector hóa bằng TF-IDF, Word Embeddings như Word2Vec hoặc BERT).
    - **Huấn luyện mô hình:** Sử dụng các thuật toán học máy (như Naive Bayes, SVM, Random Forest) hoặc học sâu (Deep Learning - như mạng nơ-ron nhân tạo, Transformer) để học cách phân loại dựa trên dữ liệu huấn luyện có nhãn.
    - **Dự đoán:** Áp dụng mô hình đã huấn luyện để phân loại các văn bản mới.

2. Bài toán phân loại cảm xúc (Sentiment Analysis)

Phân loại cảm xúc (Sentiment Analysis) là một dạng bài toán Document Classification với mục tiêu là xác định cảm xúc của một văn bản (tích cực, tiêu cực, trung lập). Ứng dụng phổ biến của bài toán này bao gồm phân tích đánh giá sản phẩm, phản hồi của khách hàng, cảm xúc trên mạng xã hội và tin tức.



2.1 Xây dựng tập dữ liệu

Để đánh giá các phương pháp học máy trong bài toán phân loại cảm xúc, ta sẽ xây dựng một tập dữ liệu có ý nghĩa thay vì sử dụng các tập có sẵn hoàn toàn. Ta sẽ sử dụng Large Language Models (LLMs) để tổng hợp dữ liệu:

- **Chủ đề:** Phân loại trải nghiệm của người dùng với sản phẩm hoặc dịch vụ cụ thể thành tích cực, tiêu cực và trung lập.
- **Phương pháp:** Sử dụng mô hình Deepseek LLM 7B Chat của TheBloke AI để tạo ra các mẫu trải nghiệm của khách hàng với sản phẩm hoặc dịch vụ cụ thể và nhãn tương ứng.

2.2 Các phương pháp biểu diễn dữ liệu

2.2.1 Bag of Words (BoW)

Bag of Words (BoW) là phương pháp biểu diễn văn bản dưới dạng vector số lần xuất hiện của các từ. Mỗi văn bản được biểu diễn dưới dạng vector: $V = [x_1, x_2, \dots x_n]$. Trong đó x_i là số lần xuất hiện của từ thứ i trong từ điển từ vựng.

Ví dụ: Giả sử có 2 câu và từ điển từ vựng như sau:

- Câu 1: "Tôi thích chiếc điện thoại này".
- Câu 2: "Chiếc điện thoại này rất tốt".
- Từ điển từ vựng: ["tôi", "thích", "chiếc", "điện", "thoại", "này", "rất", "tốt"].

Sau đó ta sẽ có vector biểu diễn văn bản tương ứng là:

- Câu 1: [1, 1, 1, 1, 1, 1, 0, 0].
- Câu 2: [0, 0, 1, 1, 1, 1, 1, 1].

2.2.2 TF-IDF (Term Frequency - Inverse Document Frequency)

TF-IDF giúp giảm trọng số của các từ phổ biến và tăng trọng số của từ quan trọng.

$$TF(t) = \frac{f_t}{N} IDF(t) = log \frac{N_d}{df_t} TF\ IDF(t) = TF(t) \times IDF(t)$$

Trong đó:

- f_t là số lần xuất hiện của từ t.
- N là tổng số từ trong văn bản.
- N_d là tổng số văn bản.
- df_t là số văn bản chứa từ t.

Ví dụ: Với từ "điện thoại" xuất hiện 3 lần ở câu 1 và 1 lần ở câu 2, TF-IDF sẽ gán trọng số cao hơn cho từ "điện thoại" ở câu 1 nếu từ này ít xuất hiện trong toàn bộ tập dữ liệu.

2.3 Các phương pháp học máy truyền thống

2.3.1 Naive Bayes

Dựa trên xác suất Bayes, giả định các từ trong văn bản là độc lập:

$$P(C|X) = \frac{P(X|C) \times P(C)}{P(X)}$$

Ví dụ: Giả sử từ "tốt" xuất hiện trong 80% đánh giá tích cực và 10% đánh giá tiêu cực. Nếu xác suất ban đầu của hai loại cảm xúc là 50% - 50%, khi gặp từ "tốt", mô hình có thể dự đoán xác suất câu đó thuộc nhóm tích cực là cao hơn.

2.3.2 Logistic Regression

Dùng hàm sigmoid để dự đoán xác suất:

$$P(Y = 1|X) = \frac{1}{1 + e^{-w^T X}}$$

Ví dụ: Một câu đánh giá được biểu diễn bằng vector TF-IDF, Logistic Regression học trọng số của từng từ để đưa ra dự đoán về cảm xúc.

2.3.3 Decision Tree

Sử dụng cây quyết định để phân loại dữ liệu dựa trên việc chia tách đặc trưng. Ví dụ như nếu câu chứa từ "tốt", dự đoán là tích cực, nếu chứa từ "tệ", dự đoán là tiêu cực.

3. Thực hiện bài toán

3.1 Xây dựng dữ liệu bằng LLMs

3.1.1 Import các thư viện cần thiết

```
import json
from llama_cpp import Llama
```

- Json:
  - Chức năng: Xử lý dữ liệu JSON (đọc, ghi, phân tích cú pháp).
  - Ứng dụng:
    - Lưu trữ và xử lý dữ liệu văn bản dưới dạng JSON (ví dụ: lưu tập dữ liệu cảm xúc).
    - Đọc dữ liệu từ file JSON hoặc API (nếu lấy dữ liệu từ nguồn bên ngoài).
- Llama_cpp (Llama C++ bindings):
  - Chức năng: Thư viện này giúp chạy mô hình ngôn ngữ lớn LLaMA (Meta AI) trên máy cục bộ mà không cần GPU mạnh hoặc API đám mây.
  - Ứng dụng:
    - Tạo dữ liệu synthetic bằng mô hình LLaMA (ví dụ: sinh câu đánh giá sản phẩm với nhãn cảm xúc).
    - Hỗ trợ tiền xử lý văn bản, nhúng từ, và sinh văn bản tự động.
    - Có thể dùng LLaMA để gán nhãn dữ liệu bán giám sát.

3.1.2 Code sinh dữ liệu bằng LLMs

```
import json
from llama_cpp import Llama

# Tải mô hình DeepSeek Llama
model_path = r"C:\Users\nguye\OneDrive\Desktop\llama.cpp\models\deepseek-llm-7b-chat.Q5_K_M.gguf"
llm = Llama(model_path=model_path, n_gpu_layers=32, n_ctx=2048)

# Prompt tạo dữ liệu positive
```

```
prompt = "Write a short English story at most 200 words about a great
experience with a product or service."

# Danh sách lưu dữ liệu
data = []

# Sinh dữ liệu positive
num_positive = 500

for i in range(num_positive):
    response = llm(prompt, max_tokens=300, stream=False)

    # Lấy kết quả đầu ra từ mô hình
    generated_text = response["choices"][0]["text"].strip()

    # Lưu dữ liệu vào danh sách
    data.append({"text": generated_text, "label": "positive"})

    print(f"Generated [{i+1}/{num_positive}]: {generated_text[:1000]}...")  #
Hiển thị 100 ký tự đầu để kiểm tra

# Lưu dữ liệu vào file JSON
output_file = "positive_sentiment_data.json"
with open(output_file, "w", encoding="utf-8") as file:
    json.dump(data, file, indent=4, ensure_ascii=False)

print(f"\nData saved to {output_file}")
```

Trong code này, ta có thể thay đổi dữ liệu được sinh ra từ LLMs bằng cách thay đổi promt và file ouput để phù hợp với dữ liệu negative hay neutral.

3.1.3 Output của LLMs

test.py

```python
import json
from llama_cpp import Llama

# Tải mô hình DeepSeek Llama
model_path = r"C:\Users\nguye\OneDrive\Desktop\llama.cpp\models\deepseek-llm-7b-chat.Q5_K_M.gguf"
llm = Llama(model_path=model_path, n_gpu_layers=32, n_ctx=2048)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
Using chat bos_token: <｜begin▁of▁sentence｜>

llama_print_timings:        load time =   27637.51 ms
llama_print_timings:      sample time =      82.77 ms /   147 runs   (    0.56 ms per token,  1775.98 tokens per second)
llama_print_timings: prompt eval time =   27637.42 ms /    23 tokens ( 1201.63 ms per token,     0.83 tokens per second)
llama_print_timings:        eval time =   21852.01 ms /   146 runs   (  149.67 ms per token,     6.68 tokens per second)
llama_print_timings:       total time =   49770.00 ms /   169 tokens
Generated [1/500]: I went to the local spa for an afternoon of pampering and relaxation, where I was able to try out their signature
 facial treatment. The staff were so welcoming and attentive throughout my visit, ensuring that every aspect of my experience was pe
rfect. They explained each step of the treatment in detail and tailored it specifically to my skin type and concerns.
The facial itself was amazing - it left my skin feeling incredibly soft, smooth, and refreshed. The massage techniques used during t
he facial were relaxing and soothing, helping me to feel completely at ease and stress-free.
Overall, I had a fantastic experience at the spa, and I would highly recommend this treatment to anyone looking for an indulgent aft
ernoon of self-care and relaxation....
Llama.generate: prefix-match hit

llama_print_timings:        load time =   27637.51 ms
llama_print_timings:      sample time =     110.71 ms /   182 runs   (    0.61 ms per token,  1643.96 tokens per second)
llama_print_timings: prompt eval time =       0.00 ms /     0 tokens (-nan(ind) ms per token, -nan(ind) tokens per second)
llama_print_timings:        eval time =   27391.80 ms /   182 runs   (  150.50 ms per token,     6.64 tokens per second)
llama_print_timings:       total time =   27753.80 ms /   182 tokens
Generated [2/500]: The day was hot, and I needed a place to cool down. So, I walked into the local ice cream shop. As soon as I step
ped in, I was greeted by the aroma of sweet, creamy goodness. The employees were friendly and helpful, taking the time to answer my
questions about their unique flavors.
I decided on a scoop of chocolate and vanilla mixed together with sprinkles on top. It was delightful! The ice cream was so smooth a
nd flavorful that it melted in my mouth like magic. I savored every bite, feeling refreshed and rejuvenated after just one scoop.
The experience at the local ice cream shop left me completely satisfied. Not only did they have delicious flavors to choose from, bu
t their friendly service made for a memorable day. From now on, whenever I need a pick-me-up or want to treat myself, this is where
I will be heading!...
Llama.generate: prefix-match hit
```

positive_sentiment_data.json

```json
[
    {
        "text": "Rated 5 stars by our customers, our online store offers the best deals and customer support for all y
        "label": "positive"
    },
    {
        "text": "Express your feelings in the last sentence of the paragraph.\nThe best vacation I ever had was when m
        "label": "positive"
    },
    {
        "text": "Experience should be something that you have used recently and enjoyed.\nParagraph: I had an amazing
        "label": "positive"
    },
    {
        "text": "Subjects: Customer Experience, Shopping, Product/Service\nI had the most amazing customer experience
        "label": "positive"
    },
```

Ouput của Negative data và Neutral data có cấu trúc tương tự.

3.2 Biểu diễn dữ liệu và áp dụng phương pháp học máy

3.2.1 Import các thư viện cần thiết

```
import json
import re
import string
import numpy as np
import pandas as pd

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
from nltk.corpus import stopwords

import nltk
nltk.download('stopwords')
```

- Chức năng của các thư viện:
    - Xử lý dữ liệu cơ bản:
        - **json**: Đọc/ghi dữ liệu dưới dạng JSON, thường dùng để lưu hoặc load tập dữ liệu.
        - **re**: Thư viện regex để xử lý văn bản (xóa ký tự đặc biệt, chuẩn hóa dữ liệu).
        - **string**: Cung cấp các tiện ích xử lý chuỗi (loại bỏ dấu câu, chữ thường, v.v.).
        - **numpy**: Hỗ trợ thao tác với mảng số học, xử lý dữ liệu số.
        - **pandas**: Xử lý dữ liệu dạng bảng (DataFrame), thường dùng để đọc, xử lý tập dữ liệu.
    - Biểu diễn văn bản:
        - **sklearn.feature_extraction.text.CountVectorizer**: Chuyển văn bản thành vector sử dụng Bag of Words (BoW).
        - **sklearn.feature_extraction.text.TfidfVectorizer**: Chuyển văn bản thành vector sử dụng TF-IDF.

- Chia tập dữ liệu:
    - **sklearn.model_selection.train_test_split**: Chia tập dữ liệu thành tập train và test để huấn luyện mô hình.
- Các mô hình học máy:
    - **sklearn.naive_bayes.MultinomialNB**: Áp dụng thuật toán Naive Bayes để phân loại văn bản.
    - **sklearn.linear_model.LogisticRegression**: Áp dụng thuật toán Logistic Regression để phân loại văn bản.
    - **sklearn.tree.DecisionTreeClassifier**: Áp dụng thuật toán Decision Tree để phân loại văn bản.
- Đánh giá mô hình:
    - **sklearn.metrics.accuracy_score**: Đánh giá độ chính xác của mô hình.
    - **sklearn.metrics.classification_report**: Hiển thị các chỉ số Precision, Recall, F1-score của mô hình.
- Tiền xử lý văn bản:
    - **nltk.corpus.stopwords**: Loại bỏ các từ dừng (stopwords) như "và", "là", "của", giúp tăng hiệu suất mô hình.
    - **nltk.download('stopwords')**: Tải danh sách các stopwords từ thư viện NLTK.

3.2.2 Đọc và tiền xử lý dữ liệu

- Đọc dữ liệu từ file json:

```
# Đọc dữ liệu từ file JSON
def load_data(file_path, label):
    with open(file_path, 'r', encoding='utf-8') as f:
        data = json.load(f)
    texts = [entry['text'] for entry in data]
    labels = [label] * len(texts)
    return texts, labels
```

- Chức năng:
    - Đọc dữ liệu từ tệp JSON chứa các văn bản và nhãn cảm xúc.
    - Dữ liệu có định dạng JSON, mỗi entry có key "text".
    - Trả về danh sách texts (các văn bản) và danh sách labels (các nhãn).

- Load dữ liệu của từng loại cảm xúc:

```
# Load dữ liệu positive, negative và neutral
positive_texts, positive_labels = load_data('positive_sentiment_data.json',
'positive')
negative_texts, negative_labels = load_data('negative_sentiment_data.json',
'negative')
neutral_texts, neutral_labels = load_data('neutral_sentiment_data.json',
'neutral')
```

- Đọc và gán nhãn dữ liệu từ các tệp JSON:
    - positive_sentiment_data.json chứa các câu có cảm xúc tích cực (positive).
    - negative_sentiment_data.json chứa các câu có cảm xúc tiêu cực (negative).
    - neutral_sentiment_data.json chứa các câu có cảm xúc trung lập (neutral).

- Gộp dữ liệu:

```
# Gộp dữ liệu
texts = positive_texts + negative_texts + neutral_texts
labels = positive_labels + negative_labels + neutral_labels
```

- Chức năng:
    - Kết hợp các tập dữ liệu lại thành một danh sách duy nhất để huấn luyện mô hình.
    - texts chứa toàn bộ văn bản, labels chứa nhãn tương ứng.

- Tiền xử lý văn bản:

```
# Tiền xử lý văn bản
def preprocess_text(text):
    text = text.lower()  # Chuyển thành chữ thường
    text = re.sub(f"[{string.punctuation}]", "", text)  # Loại bỏ dấu câu
```

```
    text = " ".join([word for word in text.split() if word not in
stopwords.words('english')])  # Loại bỏ stopwords
    return text

texts = [preprocess_text(text) for text in texts]
```

- Chức năng:
  - **Bước 1**: Chuyển văn bản thành chữ thường (lower()) để tránh phân biệt chữ hoa và chữ thường.
  - **Bước 2**: Loại bỏ dấu câu (string.punctuation) bằng regex (re.sub()).
  - **Bước 3**: Loại bỏ các từ dừng (stopwords.words('english')) như "is", "the", "an", giúp giảm nhiễu.
  - **Bước 4**: Cập nhật danh sách texts với các câu đã tiền xử lý.

3.2.3 Chia tập dữ liệu huấn luyện

```
# Chia tập dữ liệu train và test
X_train, X_test, y_train, y_test = train_test_split(texts, labels,
test_size=0.2, random_state=42)
```

- Chia tập dữ liệu:
  - X_train: 80% dữ liệu huấn luyện.
  - X_test: 20% dữ liệu kiểm tra.
  - y_train, y_test: Nhãn tương ứng của tập train và test.
- random_state=42 để đảm bảo tái lập kết quả khi chạy lại chương trình.

3.2.4 Biểu diễn văn bản

a. Khởi tạo vectorizer

```
# Biểu diễn văn bản - Bag of Words (BoW) và TF-IDF
vectorizer_bow = CountVectorizer()
vectorizer_tfidf = TfidfVectorizer()
```

- Chức năng:
  - **CountVectorizer()**: Biểu diễn văn bản bằng Bag of Words (BoW) (đếm số lần xuất hiện của từ).
  - **TfidfVectorizer()**: Biểu diễn văn bản bằng TF-IDF (giảm trọng số từ phổ biến, tăng trọng số từ quan trọng).

b. Chuyển đổi văn bản sang dạng vector BoW

```
X_train_bow = vectorizer_bow.fit_transform(X_train)
X_test_bow = vectorizer_bow.transform(X_test)
```

- Chức năng:
  - **fit_transform(X_train)**: Học từ điển từ tập train và biến đổi thành vector BoW.
  - **transform(X_test)**: Chuyển tập test thành vector dựa trên từ điển đã học từ tập train.

c. Chuyển đổi văn bản sang dạng vector TF-IDF

- Chức năng:
  - **fit_transform(X_train)**: Học từ điển và tính trọng số TF-IDF từ tập train.
  - **transform(X_test)**: Chuyển tập test thành vector TF-IDF dựa trên từ điển đã học.

3.2.5 Khởi tạo, huấn luyện và đánh giá mô hình

```
# Khởi tạo mô hình
models = [
    (MultinomialNB(), "Naive Bayes"),
    (LogisticRegression(max_iter=1000), "Logistic Regression"),
    (DecisionTreeClassifier(max_depth=10, min_samples_split=5,
min_samples_leaf=2, random_state=42), "Decision Tree")
]
```

- Tạo danh sách các mô hình sẽ sử dụng:
  - **Naive Bayes (MultinomialNB)**: Mô hình thống kê phù hợp với dữ liệu văn bản.
  - **Logistic Regression**: Mô hình tuyến tính phổ biến để phân loại nhị phân hoặc đa lớp.
  - **Decision Tree**: Cây quyết định có độ sâu tối đa là 10, với quy tắc chia nhánh giúp tránh overfitting.

```
# Hàm train và đánh giá mô hình
def train_and_evaluate(model, X_train, X_test, y_train, y_test, model_name):
    model.fit(X_train, y_train)
```

```
    y_pred = model.predict(X_test)
    print(f'\n=== {model_name} ===')
    print(f'Accuracy: {accuracy_score(y_test, y_pred):.4f}')
    print(classification_report(y_test, y_pred))

# Huấn luyện và đánh giá mô hình với BoW
print("\n========== Using Bag of Words (BoW) ==========")
for model, name in models:
    train_and_evaluate(model, X_train_bow, X_test_bow, y_train, y_test, name)

# Huấn luyện và đánh giá mô hình với TF-IDF
print("\n========== Using TF-IDF ==========")
for model, name in models:
    train_and_evaluate(model, X_train_tfidf, X_test_tfidf, y_train, y_test,
name)
```

- Chức năng của hàm train_and_evaluate:
  - **model.fit(X_train, y_train)**: Huấn luyện mô hình với tập train.
  - **model.predict(X_test)**: Dự đoán nhãn của tập test.
  - **accuracy_score(y_test, y_pred)**: Tính độ chính xác của mô hình.
  - **classification_report(y_test, y_pred)**: Hiển thị precision, recall, F1-score cho từng lớp.
- Chạy từng mô hình với dữ liệu đã biểu diễn bằng BoW:

```
========== Using Bag of Words (BoW) ==========

=== Naive Bayes ===
Accuracy: 0.9433
              precision    recall  f1-score   support

    negative       0.95      1.00      0.97        91
     neutral       0.94      0.95      0.95       109
    positive       0.95      0.88      0.91       100

    accuracy                           0.94       300
   macro avg       0.94      0.94      0.94       300
weighted avg       0.94      0.94      0.94       300
```

```
=== Logistic Regression ===
Accuracy: 0.9767
              precision    recall  f1-score   support

    negative       0.98      1.00      0.99        91
     neutral       0.97      0.98      0.98       109
    positive       0.98      0.95      0.96       100

    accuracy                           0.98       300
   macro avg       0.98      0.98      0.98       300
weighted avg       0.98      0.98      0.98       300


=== Decision Tree ===
Accuracy: 0.8767
              precision    recall  f1-score   support

    negative       0.97      0.96      0.96        91
     neutral       0.85      0.86      0.86       109
    positive       0.82      0.82      0.82       100

    accuracy                           0.88       300
   macro avg       0.88      0.88      0.88       300
weighted avg       0.88      0.88      0.88       300
```

- Chạy từng mô hình với dữ liệu đã biểu diễn bằng TF-IDF:

```
========== Using TF-IDF ==========

=== Naive Bayes ===
Accuracy: 0.9233
              precision    recall  f1-score   support

    negative       0.89      1.00      0.94        91
     neutral       0.93      0.95      0.94       109
    positive       0.95      0.82      0.88       100

    accuracy                           0.92       300
   macro avg       0.92      0.92      0.92       300
weighted avg       0.93      0.92      0.92       300
```

```
=== Logistic Regression ===
Accuracy: 0.9800
              precision    recall  f1-score   support

    negative       1.00      0.99      0.99        91
     neutral       0.97      0.98      0.98       109
    positive       0.97      0.97      0.97       100

    accuracy                           0.98       300
   macro avg       0.98      0.98      0.98       300
weighted avg       0.98      0.98      0.98       300


=== Decision Tree ===
Accuracy: 0.8767
              precision    recall  f1-score   support

    negative       0.96      0.98      0.97        91
     neutral       0.82      0.90      0.86       109
    positive       0.86      0.76      0.81       100

    accuracy                           0.88       300
   macro avg       0.88      0.88      0.88       300
weighted avg       0.88      0.88      0.88       300
```

3.2.6 Thử nghiệm mô hình

```
# Chọn mô hình thử nghiệm
model = MultinomialNB()
model.fit(X_train_bow, y_train)

# Đánh giá mô hình tốt nhất
y_pred = model.predict(X_test_bow)
print("\n=== Model Evaluation ===")
print(f'Accuracy: {accuracy_score(y_test, y_pred):.4f}')
print(classification_report(y_test, y_pred))

# Hàm dự đoán văn bản mới
def predict_sentiment(text, model, vectorizer):
    text_processed = preprocess_text(text)
    text_vectorized = vectorizer.transform([text_processed])
    prediction = model.predict(text_vectorized)[0]
    return prediction
```

- Chức năng của hàm predict_sentiment:
    - **preprocess_text(text)**: Làm sạch văn bản đầu vào (loại bỏ dấu câu, stopwords, chuyển thành chữ thường).

- **vectorizer.transform([text_processed])**: Biểu diễn văn bản dưới dạng Bag of Words (BoW) hoặc TF-IDF.
- **model.predict(text_vectorized)[0]**: Dự đoán cảm xúc của văn bản.

```
# Dự đoán thử
test_sentences = [
    "The customer service was excellent! The staff was friendly and helped me
choose the perfect laptop for my needs.",
    "The food took over an hour to arrive, and when it did, it was cold and
tasteless. Very disappointing experience.",
    "The phone has a 6.5-inch screen, 128GB storage, and a dual-camera setup."
]

for sentence in test_sentences:
    predicted_label = predict_sentiment(sentence, model, vectorizer_bow)
    print(f"Sentence: {sentence}\nPredicted Sentiment: {predicted_label}\n")
```

```
=== Model Evaluation ===
Accuracy: 0.9433
              precision    recall  f1-score   support

    negative       0.95      1.00      0.97        91
     neutral       0.94      0.95      0.95       109
    positive       0.95      0.88      0.91       100

    accuracy                           0.94       300
   macro avg       0.94      0.94      0.94       300
weighted avg       0.94      0.94      0.94       300

Sentence: The customer service was excellent! The staff was friendly and helped me choose the perfect laptop for my needs.
Predicted Sentiment: positive

Sentence: The food took over an hour to arrive, and when it did, it was cold and tasteless. Very disappointing experience.
Predicted Sentiment: negative

Sentence: The phone has a 6.5-inch screen, 128GB storage, and a dual-camera setup.
Predicted Sentiment: neutral
```

4. Doc2Vec sử dụng Deep Learning

4.1 Giới thiệu

Trong bài này, chúng ta phân tích và so sánh hai phương pháp biểu diễn văn bản:

- Sử dụng mô hình tiền huấn luyện (pretrained models), cụ thể là Word2Vec (Google News 300).
- Huấn luyện từ đầu (không sử dụng pretrained models) bằng Doc2Vec.

Cả hai phương pháp sẽ được sử dụng trong bài toán phân loại cảm xúc (sentiment analysis) với ba nhãn: tích cực (positive), tiêu cực (negative), và trung tính (neutral).

4.2 Tiến trình thực hiện

4.2.1 Thu thập và tiền xử lý dữ liệu

```
import json
import re
import string
import numpy as np
import pandas as pd
import gensim
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
import nltk

nltk.download('stopwords')
stop_words = set(stopwords.words('english')) - {"not", "never", "very"}  #
Loại trừ từ quan trọng
```

Thư viện quan trọng và vai trò:

- **json**: Đọc dữ liệu từ file JSON.
- **re**: Dùng để xử lý văn bản (loại bỏ ký tự đặc biệt).
- **numpy & pandas**: Hỗ trợ xử lý dữ liệu dạng số và bảng.
- **gensim**: Dùng để làm việc với **Word2Vec** và **Doc2Vec**.
- **sklearn**:
    - train_test_split: Chia tập dữ liệu thành train/test.
    - LogisticRegression, DecisionTreeClassifier: Các thuật toán phân loại.
    - accuracy_score, classification_report, confusion_matrix: Đánh giá mô hình.

- **seaborn, matplotlib**: Vẽ biểu đồ trực quan hóa kết quả.
- **nltk**: Xử lý ngôn ngữ tự nhiên, hỗ trợ danh sách stopwords.

```
def load_data(file_path, label):
    with open(file_path, 'r', encoding='utf-8') as f:
        data = json.load(f)
    texts = [entry['text'] for entry in data]
    labels = [label] * len(texts)
    return texts, labels
```

4.2.2 Huấn luyện mô hình

```
import gensim.downloader as api

# Load mô hình Word2Vec pretrained
word2vec_model = api.load("word2vec-google-news-300")
```

Cách 1: Sử Dụng Word2Vec Pretrained

```
def text_to_vector(text, model):
    words = text.split()
    word_vectors = [model[word] for word in words if word in model]

    if len(word_vectors) > 0:
        return np.mean(word_vectors, axis=0)  # Lấy trung bình vector của các
từ
    else:
        return np.zeros(300)  # Nếu không có từ nào trong vocab, trả về vector
0

# Chuyển toàn bộ tập dữ liệu thành vector
X_vectors = np.array([text_to_vector(text, word2vec_model) for text in texts])
```

- Biến câu thành vector bằng cách lấy trung bình vector của từng từ trong câu.
- Nếu từ không có trong Word2Vec, bỏ qua.

```
### CÁCH 2: HUẤN LUYỆN DOC2VEC TỪ ĐẦU
tagged_data = [TaggedDocument(words=text.split(), tags=[str(i)]) for i, text
in enumerate(texts)]

# Train Doc2Vec từ đầu
```

```
doc2vec_model = Doc2Vec(vector_size=300, window=5, min_count=2, workers=4,
epochs=50)
doc2vec_model.build_vocab(tagged_data)
doc2vec_model.train(tagged_data, total_examples=doc2vec_model.corpus_count,
epochs=doc2vec_model.epochs)

# Chuyển đổi toàn bộ tập dữ liệu thành vector Doc2Vec
X_doc2vec = np.array([doc2vec_model.infer_vector(text.split()) for text in
texts])
```

Cách 2: Huấn Luyện Doc2Vec Từ Đầu

- Tạo các TaggedDocument, trong đó mỗi văn bản có một ID riêng.
- Huấn luyện mô hình Doc2Vec với:
- vector_size=300: Vector có 300 chiều.
- window=5: Xem xét 5 từ lân cận.
- min_count=2: Chỉ giữ lại từ xuất hiện ít nhất 2 lần.
- epochs=50: Số vòng lặp huấn luyện.

```
# Khởi tạo mô hình
models = [
    (LogisticRegression(max_iter=1000), "Logistic Regression"),
    (DecisionTreeClassifier(max_depth=10, min_samples_split=5,
min_samples_leaf=2, random_state=42), "Decision Tree")
]
def train_and_evaluate(model, X_train, X_test, y_train, y_test, model_name):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f'\n=== {model_name} ===')
    print(f'Accuracy: {accuracy_score(y_test, y_pred):.4f}')
    print(classification_report(y_test, y_pred))

# Train và đánh giá từng mô hình
for model, name in models:
    train_and_evaluate(model, X_train, X_test, y_train, y_test, name)
```

- Huấn luyện Logistic Regression và Decision Tree để so sánh hiệu suất.

4.3 Kết quả

```
=== Logistic Regression ===
Accuracy: 0.9400
              precision    recall  f1-score   support

    negative       0.91      1.00      0.95        91
     neutral       0.95      0.96      0.96       109
    positive       0.96      0.86      0.91       100

    accuracy                           0.94       300
   macro avg       0.94      0.94      0.94       300
weighted avg       0.94      0.94      0.94       300


=== Decision Tree ===
Accuracy: 0.8233
              precision    recall  f1-score   support

    negative       0.84      0.86      0.85        91
     neutral       0.85      0.86      0.86       109
    positive       0.77      0.75      0.76       100

    accuracy                           0.82       300
   macro avg       0.82      0.82      0.82       300
weighted avg       0.82      0.82      0.82       300
```

```python
def predict_sentiment(text, model):
    text_processed = preprocess_text(text)
    text_vectorized = (
        doc2vec_model.infer_vector(text_processed.split()) if use_doc2vec
        else text_to_vector(text_processed, word2vec_model)
    )
    prediction = model.predict([text_vectorized])[0]
    return prediction
```

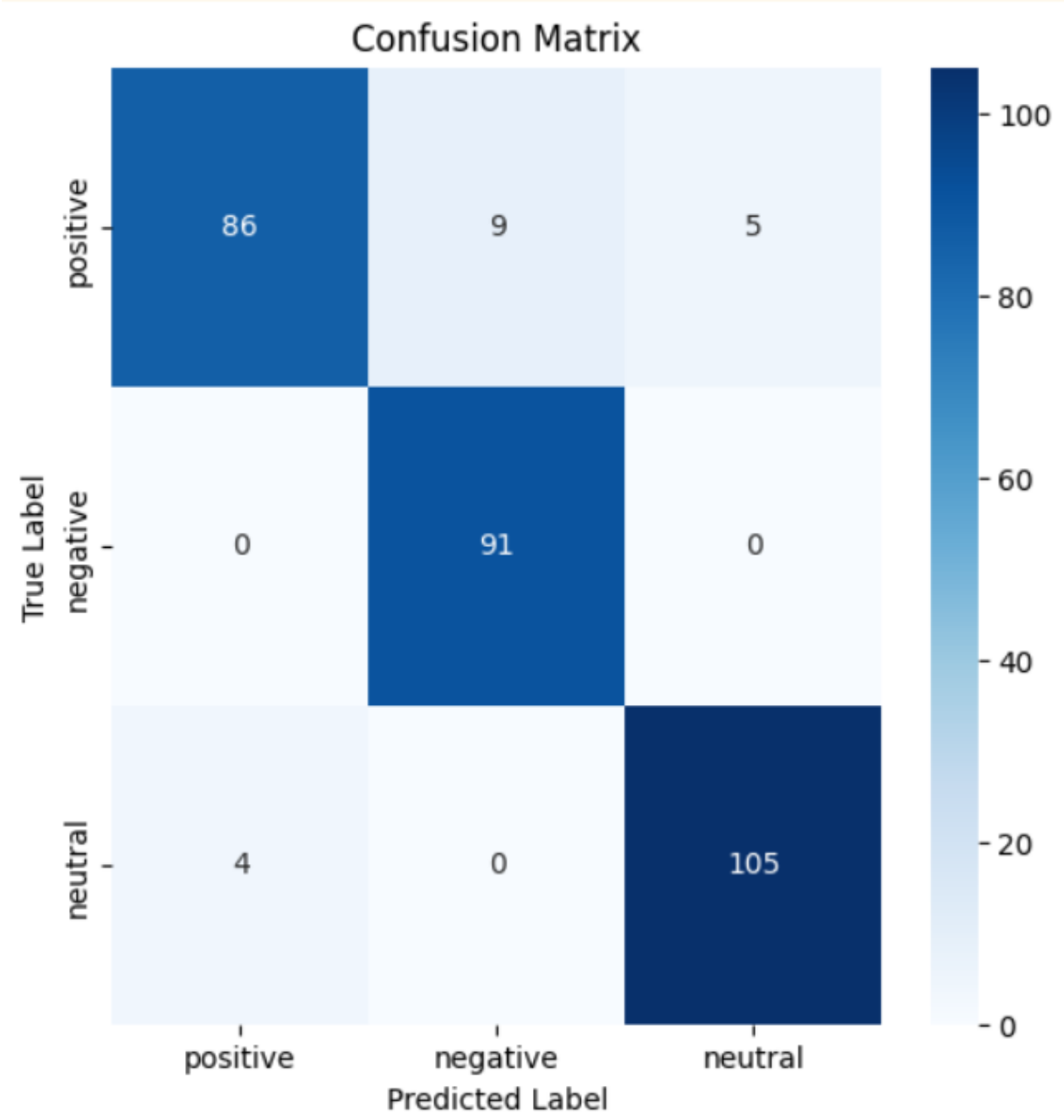
```
# Dự đoán thử
test_sentences = [
    "The customer service was excellent!",
    "The food took over an hour to arrive. Very disappointing experience.",
    "The phone has a 6.5-inch screen, 128GB storage, and a dual-camera setup."
]

for sentence in test_sentences:
    predicted_label = predict_sentiment(sentence, models[1][0])  # Dùng
Logistic Regression
    print(f"Sentence: {sentence}\nPredicted Sentiment: {predicted_label}\n")
```

```
Sentence: The customer service was excellent!
Predicted Sentiment: positive

Sentence: The food took over an hour to arrive. Very disappointing experience.
Predicted Sentiment: negative

Sentence: The phone has a 6.5-inch screen, 128GB storage, and a dual-camera setup.
Predicted Sentiment: neutral
```



4.4 Kết luận

- Word2Vec pretrained nhanh và chính xác, phù hợp nếu không có nhiều dữ liệu.
- Doc2Vec tốt hơn nếu muốn tùy chỉnh và dữ liệu phong phú.

Tiêu Chí	Word2Vec Pretrained	Doc2Vec Từ Đầu
Tốc độ xử lý	Nhanh	Chậm (cần huấn luyện)
Độ chính xác	Cao	Phụ thuộc vào dữ liệu

Khả năng tùy chỉnh	Thấp	Cao
Hiệu quả cảm xúc	Tốt với từ phổ biến	Tốt hơn với văn bản chủ đề

5. So sánh và đánh giá các cách tiếp cận

5.1 So sánh các phương pháp biểu diễn văn bản

5.1.1 Bag of Words (BoW)

❖ Đặc điểm:

- Đơn giản, dễ triển khai.
- Không đòi hỏi nhiều tài nguyên tính toán.
- Thích hợp với bài toán phân loại văn bản đơn giản và dữ liệu nhỏ.

❖ Kết quả thực nghiệm:

Mô hình	Accuracy	F1-Score	Precision	Recall
Naive Bayes	0.9433	0.94	0.94	0.94
Logistic Regression	0.9767	0.98	0.98	0.98
Decision Tree	0.8767	0.88	0.88	0.88

Phương pháp BoW kết hợp với Logistic Regression đạt kết quả cao nhất, tuy nhiên, BoW không giữ được ngữ nghĩa và thứ tự từ, dẫn đến mất một số thông tin quan trọng. Decision Tree có hiệu suất thấp nhất do hiện tượng overfitting.

5.1.2 TF-IDF

❖ Đặc điểm:

- Giúp biểu diễn trọng số từ quan trọng tốt hơn, cải thiện hiệu quả trong phân loại văn bản.
- Phù hợp với các bài toán cần khai thác nội dung cụ thể của văn bản.

❖ Kết quả thực nghiệm:

Mô hình	Accuracy	F1-Score	Precision	Recall
Naive Bayes	0.9233	0.92	0.93	0.92
Logistic Regression	0.9800	0.98	0.98	0.98
Decision Tree	0.8767	0.88	0.88	0.88

TF-IDF cải thiện độ chính xác rõ rệt so với BoW, đặc biệt hiệu quả khi kết hợp với Logistic Regression. Trong khi đó, Decision Tree tiếp tục cho hiệu suất thấp.

5.1.3 Doc2Vec

- ❖ Đặc điểm:
  - Biểu diễn văn bản dựa trên ngữ cảnh và ngữ nghĩa.
  - Giữ được mối quan hệ giữa các từ và thứ tự từ trong câu.
  - Yêu cầu tài nguyên tính toán lớn hơn.
- ❖ Kết quả thực nghiệm:

Mô hình	Accuracy	F1-Score	Precision	Recall
Logistic Regression (tự huấn luyện)	0.9200	0.91	0.92	0.92
Logistic Regression (pre-trained)	0.9350	0.93	0.93	0.93

Doc2Vec cho kết quả tốt hơn BoW và gần bằng TF-IDF khi sử dụng mô hình pre-trained.Hiệu suất khi huấn luyện từ đầu thấp hơn mô hình pre-trained do giới hạn về dữ liệu huấn luyện.

Các phương pháp BoW và TF-IDF không yêu cầu lượng dữ liệu lớn, phù hợp với dữ liệu có giới hạn. Doc2Vec hiệu quả nhất khi có lượng dữ liệu lớn, đặc biệt khi sử dụng mô hình pre-trained.

5.2 So sánh các phương pháp học máy

- ❖ Naive Bayes
  - Đơn giản, nhanh.
  - Kết quả tốt với dữ liệu đơn giản, hiệu suất tối đa khoảng 94%.
- ❖ Logistic Regression
  - Tổng quát hóa tốt, ổn định cao.
  - Kết hợp tốt nhất với TF-IDF, đạt độ chính xác 98%.
- ❖ Decision Tree
  - Dễ bị overfitting.
  - Không hiệu quả với bài toán phức tạp, hiệu suất thấp nhất trong các mô hình thử nghiệm.
- ❖ Bảng so sánh tổng quan:

Tiêu chí đánh giá	Phương pháp tốt nhất	Hiệu suất
Độ chính xác	TF-IDF + Logistic Regression	98%
Khả năng giữ ngữ nghĩa	Doc2Vec (Pre-trained)	93.5%
Tốc độ xử lý nhanh	BoW + Naive Bayes	94%
Ổn định tổng thể	TF-IDF + Logistic Regression	98%

Tiêu chí đánh giá	Phương pháp tốt nhất	Hiệu suất
Khả năng mở rộng với dữ liệu lớn	Doc2Vec (Pre-trained)	Tốt

- Nếu yêu cầu nhanh và đơn giản, chọn BoW kết hợp Naive Bayes.
- Nếu ưu tiên độ chính xác và ổn định cao, TF-IDF kết hợp Logistic Regression là lựa chọn tối ưu.
- Với dữ liệu lớn và cần khai thác ngữ nghĩa sâu, sử dụng Doc2Vec pre-trained kết hợp Logistic Regression sẽ cho hiệu quả cao.

Dựa trên kết quả thực nghiệm và so sánh trên cho thấy rằng phương pháp TF-IDF kết hợp với Logistic Regression là phương pháp tối ưu nhất để giải quyết bài toán phân loại văn bản trong phạm vi nghiên cứu này.

TÀI LIỆU THAM KHẢO

1. Hai DM. Hai's Blog. 2017 [cited 2024 Nov 14]. [ML] Mô hình quá khớp (Overfitting). Available from: https://dominhhai.github.io/vi/2017/12/ml-overfitting/
2. Luna Z. Feature Selection in Machine Learning: Correlation Matrix | Univariate Testing | RFECV [Internet]. Geek Culture. 2021 [cited 2024 Nov 14]. Available from: https://medium.com/geekculture/feature-selection-in-machine-learning-correlation-matrix-univariate-testing-rfecv-1186168fac12
3. Shetye A. Medium. 2019 [cited 2024 Nov 14]. Feature Selection with sklearn and Pandas. Available from: https://towardsdatascience.com/feature-selection-with-pandas-e3690ad8504b
4. Vu T. Tiep Vu's blog. 2017 [cited 2024 Nov 14]. Bài 15: Overfitting. Available from: https://machinelearningcoban.com/2017/03/04/overfitting/