

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN



XỬ LÝ NGÔN NGỮ TỰ NHIÊN NÂNG CAO

**D10: Xây dựng các API và demo đánh giá ý kiến người
dùng cho văn bản tiếng việt**

Sinh viên: 21127050 - Trần Nguyên Huân

21127131 - Trần Hải Phát

21127240 - Nguyễn Phát Đạt

Lớp: HP2-K33

Giảng viên hướng dẫn: PGS.TS. Đinh Điền

TS. Nguyễn Hồng Bửu Long

TS. Lương An Vinh

Tháng 7, năm 2024, TP.HCM

Nội dung

1	Giới thiệu	3
1	Bối cảnh	3
2	Phát biểu bài toán	4
2.1	Đầu vào	4
2.2	Đầu ra	5
3	Thách thức	5
4	Quy trình	6
4.1	Thu thập dữ liệu	6
4.2	Tiền xử lý dữ liệu	6
4.3	Word embedding	7
4.4	Trích xuất đặc trưng và phân loại	7
4.5	Phân loại	8
4.6	Đánh giá mô hình	8
5	Các phương pháp	9
5.1	Mô hình được tinh chỉnh để phân tích cảm tính dựa trên vinai/phobert-base	9
5.2	Mô hình LSTM	9
5.2.1	Khái niệm	9
5.2.2	Cài đặt thí nghiệm	10
5.3	Mô hình Bi-LSTM	12
5.3.1	Khái niệm	12
5.3.2	Cài đặt thí nghiệm	13
5.4	Mô hình GRU	14
5.4.1	Khái niệm	14
5.4.2	Cài đặt thí nghiệm	15
5.5	Mô hình kết hợp CNN và LSTM [8]	17
5.5.1	Khái niệm	17
5.5.2	Cài đặt thí nghiệm	17

6	Bộ dữ liệu	20
2	Cách thiết lập	22
1	Kiến trúc của ứng dụng	22
1.1	Mô Tả Quy Trình	22
1.2	Mô tả cài đặt Docker [2]	23
1.2.1	Docker Images	23
1.2.2	Docker Containers	23
1.2.3	Kết Nối Docker Containers	24
2	Cách chạy chương trình	24
2.1	Cấu trúc thư mục	25
2.2	Hướng dẫn chạy chương trình	26
3	Cách huấn luyện	26
3.1	Tổng quan	26
3.2	Quy trình các bước	27
3.2.1	Tiền xử lí	27
3.2.2	Thiết lập mô hình	31
4	Cách kiểm tra và đánh giá	33
3	Cách sử dụng	34
4	Thông số kỹ thuật API	38
1	Mô tả API phân tích theo tập tài liệu (document)	38
2	Mô tả API phân tích theo các câu (sentences)	40
5	Kết luận	43
1	So sánh kết quả các mô hình	43
1.1	Bảng so sánh kết quả các mô hình	43
1.2	Phân tích kết quả	43
1.3	Nhận xét tổng quan	44

Chương 1

Giới thiệu

1 Bối cảnh



Hình 1.1: Phân tích cảm xúc (nguồn: github.com/srini047)

Đánh giá ý kiến người dùng (Sentiment Analysis) là một lĩnh vực quan trọng trong xử lý ngôn ngữ tự nhiên. Mục tiêu của bài toán này là xác định cảm xúc hoặc thái độ của người viết đối với một vấn đề nào đó thông qua văn bản. Cảm xúc có thể được phân loại thành các nhóm chính như tích cực, tiêu cực, hoặc trung tính [5].

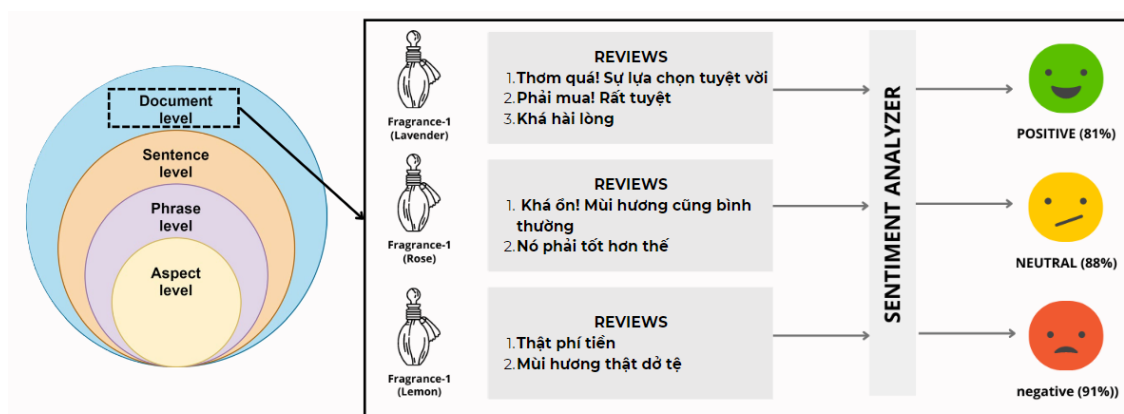
Với sự phát triển của các nền tảng trực tuyến nơi các cá nhân có thể bày tỏ một cách cởi mở ý kiến và quan điểm của mình, nó có ngày càng trở nên quan trọng khi các doanh nghiệp có thể khai thác những thông tin này nhằm đưa ra các quyết định và các chiến lược trong tương lai.

Đối với tiếng Việt, ta cần lưu ý do bài toán gặp nhiều thách thức hơn so với các ngôn ngữ phổ biến như tiếng Anh do những đặc thù của ngôn ngữ. Điển hình như:

- Đặc điểm ngữ pháp và cú pháp phức tạp: Tiếng Việt có cấu trúc câu linh hoạt và không theo khuôn mẫu cố định như tiếng Anh, điều này làm cho việc xử lý ngôn ngữ tự nhiên trở nên khó khăn hơn.
- Từ vựng đa dạng và nhiều từ đồng âm khác nghĩa: Tiếng Việt có nhiều từ đồng âm nhưng mang nghĩa khác nhau, điều này đòi hỏi mô hình phải có khả năng phân biệt ngữ cảnh rất tốt.
- Thiếu hụt dữ liệu huấn luyện: So với tiếng Anh, lượng dữ liệu được gán nhãn để huấn luyện các mô hình Sentiment Analysis tiếng Việt còn rất hạn chế, làm giảm hiệu quả của mô hình.

2 Phát biểu bài toán

Bài toán phân tích cảm xúc có nhiều mức độ phân tích khác nhau, mỗi mức độ tập trung vào một tầng nhỏ hơn của ngôn ngữ. Được mô tả như hình dưới:



Hình 1.2: Các mức độ phân tích cảm xúc

Ở đây, ta sẽ tập trung vào phân tích ở mức độ văn bản (document level): đưa ra một kết quả phân tích duy nhất đối với mỗi đoạn văn bản bất kỳ.

2.1 Đầu vào

Đầu vào thường là một câu hoặc một đoạn văn bản dưới dạng ngôn ngữ tự nhiên (xuất phát từ con người), ở dạng các kí tự (dấu câu, chữ số, kí tự đặc biệt...):

$$d = \{w_1, w_2, \dots, w_n\}$$

Trong đó:

- d là văn bản đầu vào.
- w_i có thể là từ, hoặc chữ số, hoặc kí tự đặc biệt, dấu câu... thứ i trong văn bản d .
- n là số lượng kí tự của văn bản.

2.2 Đầu ra

Đầu ra là nhãn cảm xúc được dự đoán dành cho đoạn văn bản đó, số lượng lớp có thể đa dạng nhưng ở đây ta tập trung vào dạng sau:

$$y = \begin{cases} 0 & (\text{tiêu cực}) \\ 1 & (\text{trung tính}) \\ 2 & (\text{tích cực}) \end{cases}$$

Có thể bổ sung thêm xác suất dự đoán của mô hình để người dùng biết được độ tự tin của dự đoán đó:

Ví dụ:

$$P(y=2) = 0.7, \quad P(y=1) = 0.2, \quad P(y=0) = 0.1$$

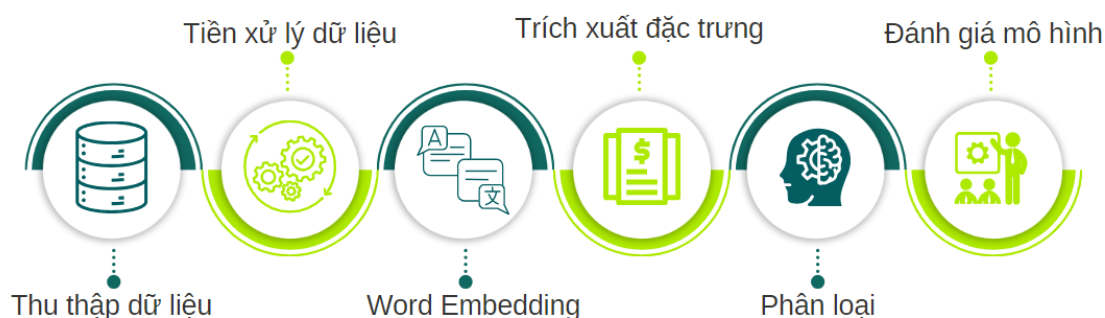
3 Thách thức

Một số thách thức thường gặp đối với bài toán phân tích cảm xúc mà ta phải lưu ý gồm:

1. Ngữ cảnh: Mô hình cần hiểu được ngữ cảnh để cho ra kết quả tốt hơn khi được áp dụng rộng rãi trên nhiều lĩnh vực.
2. Tính nhập nhằng: Các câu văn mang tính mỉa mai, giọng điệu gây nhầm lẫn, nhập nhằng dễ gây ảnh hưởng xấu đến kết quả.
3. Chuẩn hóa: Các văn bản thường được áp dụng (đặc biệt là các đánh giá của người dùng trên mạng) thường không được cấu trúc, thường xuyên mắc các lỗi ngữ pháp, sai chính tả, viết tắt...

4. Dữ liệu: Cần bộ dữ liệu huấn luyện đa dạng (về đặc trưng vùng miền, văn hóa, hành văn, ngữ cảnh, mục đích...) để mô hình đạt được tính tổng quát hóa tốt và tránh tình trạng thiên vị nếu dữ liệu không có tính đại diện tốt.

4 Quy trình



Hình 1.3: Quy trình thường gặp đối với bài toán phân tích cảm xúc

4.1 Thu thập dữ liệu

Thu thập dữ liệu là một bước quan trọng trong việc phát triển mô hình Sentiment Analysis cho tiếng Việt. Quá trình này bao gồm nhiều giai đoạn từ lựa chọn nguồn dữ liệu đến tiền xử lý và gán nhãn...

Đây là quá trình đòi hỏi rất nhiều chi phí về thời gian và tiền bạc khi tài nguyên chủ yếu là sức người, do nhãn cảm xúc phải được con người gán thì mới đảm bảo tính chính xác.

4.2 Tiền xử lý dữ liệu

1. **Lọc dữ liệu**: Loại bỏ các văn bản không có nội dung cảm xúc hoặc không có ý nghĩa rõ ràng. Ví dụ, loại bỏ các văn bản chỉ chứa ký tự đặc biệt hoặc chỉ có một từ ngắn gọn như “OK”. Loại bỏ các văn bản trùng lặp để tránh bias trong dữ liệu...
2. **Tách từ**: Sử dụng các công cụ như `Underthesea`, `pyvi`... để tách từ trong tiếng Việt. Do tiếng Việt là ngôn ngữ đơn âm tiết không có dấu phân cách rõ ràng giữa các từ, nên việc tách từ chính xác là rất quan trọng.

3. **Chuẩn hóa văn bản:** thay thế các chữ hoa thành chữ thường, sửa lỗi chính tả, loại bỏ dấu câu, ký tự đặc biệt, từ viết tắt, loại bỏ các stopwords...

(Tùy thuộc vào độ lớn dữ liệu và mô hình mà ta có thể cân nhắc không cần thực hiện các bước này do các đặc điểm này đôi khi cũng thể hiện một phần cảm xúc người nói)

4.3 Word embedding

Máy tính không thể hiểu từ dưới dạng văn bản như con người mà ta phải chuyển đổi chúng thành dạng số để máy tính có thể tính toán được.

Word Embedding là một không gian vector dùng để biểu diễn từ ngữ một cách có liên hệ, tương đồng, ngữ cảnh có nghĩa. Không gian này bao gồm nhiều chiều và các từ trong không gian đó mà có cùng văn cảnh hoặc ngữ nghĩa sẽ có vị trí gần nhau.

Một số phương pháp thường dùng:

- **Dựa trên tần suất:** Phương pháp này tính toán mức liên quan về mặt ngữ nghĩa giữa các từ bằng cách thống kê số lần đồng xuất hiện của một từ so với các từ khác. Kỹ thuật TF-IDF (Term Frequency-Inverse Document Frequency) được sử dụng để đánh giá tầm quan trọng của một từ trong một văn bản so với toàn bộ tập dữ liệu.
- **Dựa trên dự đoán:** Tính toán sự tương đồng ngữ nghĩa giữa các từ để dự đoán từ tiếp theo bằng cách đưa qua một mô hình dự đoán đơn giản. Một context word có thể là một hoặc nhiều từ khác nhau. Có 2 phương pháp thường dùng là Continuous Bag-of-Words (CBOW) và Skip-gram.

4.4 Trích xuất đặc trưng và phân loại

Dùng một hoặc nhiều mô hình học máy / học sâu nhằm trích xuất các đặc trưng giúp mô hình hiểu được văn bản, từ ngữ và quan trọng nhất là các đặc trưng giúp mô hình đưa ra dự đoán cuối cùng cho lớp cảm xúc của đoạn văn bản.

- Học từ đầu (train from scratch): Học từ đầu thường được áp dụng khi bạn có một tập dữ liệu lớn và đa dạng, đủ để huấn luyện một mô

hình từ đầu mà không cần sử dụng mô hình pretrained. Hoặc tác vụ được áp dụng chỉ cần trên một lĩnh vực, ngữ cảnh nhỏ...

- Tận dụng các mô hình huấn luyện sẵn (pretrained models): Huấn luyện mô hình trước với các tác vụ không giám sát trên bộ ngữ liệu lớn. Sau đó có thể giúp tiết kiệm thời gian học cho tác vụ sau, khi ban đầu một phần mô hình đã hiểu sẵn các đặc trưng ngôn ngữ, ta chỉ cần thêm hoặc thay đổi lớp cuối cùng mà vẫn đảm bảo hiệu suất tốt (mô hình BERT [1], hoặc mô hình ngôn ngữ Việt như PhoBERT [6]...).

4.5 Phân loại

Từ các đặc trưng mô hình đã được chiết xuất và học được, ở lớp cuối cùng ta có thể dùng các kiến trúc khác để hỗ trợ cho quá trình phân lớp (cảm xúc) để đưa ra dự đoán cuối cùng.

- Các lớp này thường được tích hợp và huấn luyện chung với phần trích xuất đặc trưng nếu mô hình được huấn luyện từ đầu.
- Đối với phương pháp tận dụng mô hình huấn luyện sẵn thì các lớp phục vụ cho tác vụ phân loại có thể được huấn luyện riêng biệt.

4.6 Đánh giá mô hình

Sau khi cho mô hình học trên tập dữ liệu huấn luyện, ta đánh giá hiệu suất của chúng trên tập dữ liệu mà chúng chưa được thấy (dữ liệu kiểm tra). Tập dữ liệu cần có tính đại diện cao trên nhiều ngữ cảnh, lĩnh vực, tính cân bằng, đa dạng...

Các thông số đánh giá thường dùng:

- a. Độ Chính Xác (Accuracy)

$$\text{Accuracy} = \frac{\text{Số lượng dự đoán đúng}}{\text{Tổng số dự đoán}}$$

Độ chính xác đo lường tỷ lệ dự đoán chính xác so với tổng số dự đoán.

- b. Đánh giá ở mỗi lớp: Ví dụ đối với lớp 'tích cực':

$$\text{Precision} = \frac{\text{Số lượng dự đoán đúng là tích cực}}{\text{Số lượng dự đoán là tích cực}}$$

$$\text{Recall} = \frac{\text{Số lượng dự đoán đúng là tích cực}}{\text{Số lượng thực tế là tích cực}}$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

5 Các phương pháp

5.1 Mô hình được tinh chỉnh để phân tích cảm tính dựa trên vinai/phobert-base

Mô hình wonrax/phobert-base-vietnamese-sentiment là một mô hình ngôn ngữ dựa trên transformer, được pre-train trên một tập dữ liệu văn bản tiếng Việt lớn. Mô hình này được xây dựng dựa trên kiến trúc PhoBERT base, một mô hình BERT được thiết kế riêng cho tiếng Việt.

Mô hình này được fine-tune thêm trên một tập dữ liệu phân tích cảm xúc tiếng Việt, cho phép nó phân loại hiệu quả cảm xúc của văn bản tiếng Việt thành các nhãn tích cực, tiêu cực hoặc trung tính.

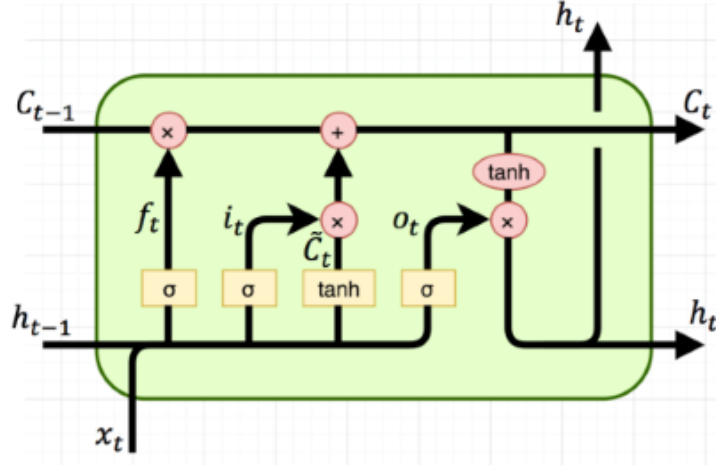
5.2 Mô hình LSTM

5.2.1 Khái niệm

Mạng trí nhớ ngắn hạn định hướng dài hạn còn được viết tắt là LSTM là một kiến trúc đặc biệt của RNN có khả năng học được sự phụ thuộc trong dài hạn (long-term dependencies).

Kiến trúc này đã được phổ biến và sử dụng rộng rãi cho tới ngày nay. LSTM đã tỏ ra khắc phục được rất nhiều những hạn chế của RNN trước đây về triệt tiêu đạo hàm. Tuy nhiên cấu trúc của chúng có phần phức tạp hơn mặc dù vẫn dĩ được tư tưởng chính của RNN là sự sao chép các kiến trúc theo dạng chuỗi.

LSTM cũng có một chuỗi dạng như RNN nhưng phần kiến trúc lặp lại có cấu trúc khác biệt hơn. Thay vì chỉ có một tầng đơn, chúng có tới 4 tầng ẩn (3 sigmoid và 1 tanh) tương tác với nhau theo một cấu trúc đặc biệt.



Hình 1.4: Kiến trúc LSTM

Trong đó:

- f_t , i_t , o_t tương ứng với forget gate, input gate và output gate.
 - $f_t = \sigma(U_f * x_t + W_f * h_{t-1} + b_f)$
 - $i_t = \sigma(U_i * x_t + W_i * h_{t-1} + b_i)$
 - $o_t = \sigma(U_o * x_t + W_o * h_{t-1} + b_o)$
- $\tilde{c}_t = \tanh(U_c * x_t + W_c * h_{t-1} + b_c)$, bước này giống như tính s_t trong RNN.
- $c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$, **forget gate** quyết định xem cần lấy bao nhiêu từ cell state trước và **input gate** sẽ quyết định lấy bao nhiêu từ input của state và hidden layer của layer trước.
- $h_t = o_t * \tanh(c_t)$, **output gate** quyết định xem cần lấy bao nhiêu từ cell state để trở thành output của hidden state. Ngoài ra h_t cũng được dùng để tính ra output y_t cho state t.

Nhận xét: h_t, \tilde{c}_t khá giống với RNN, nên model có short term memory. Trong khi đó c_t giống như một băng chuyền ở trên mô hình RNN, thông tin nào cần quan trọng và dùng ở sau sẽ được gửi vào và dùng khi cần. Do đó có thể mang thông tin từ đi xa (long term memory).

5.2.2 Cài đặt thí nghiệm

a. Cấu trúc mô hình

Mô hình được xây dựng gồm các thành phần chính sau:

- **Lớp Embedding:** Đầu vào của mô hình là các chuỗi ký tự hoặc từ, được ánh xạ sang không gian vector thông qua lớp Embedding. Lớp này sử dụng hàm khởi tạo `RandomUniform` để đảm bảo việc khởi tạo ngẫu nhiên có tính ổn định.
- **Lớp LSTM:** Sau lớp Embedding, mô hình bao gồm hai lớp LSTM. Mỗi lớp có nhiệm vụ học các phụ thuộc trong chuỗi thời gian:
 - ◊ **LSTM Layer 1:** Lớp LSTM đầu tiên trả về toàn bộ chuỗi các hidden states (`return_sequences=True`). Lớp này giúp mô hình ghi nhớ thông tin tại mỗi bước thời gian.
 - ◊ **LSTM Layer 2:** Lớp LSTM thứ hai trả về chỉ hidden state cuối cùng của chuỗi (`return_sequences=False`), giúp mô hình tập trung vào trạng thái quan trọng nhất từ chuỗi dữ liệu.
- **Lớp Fully-Connected:** Sau khi xử lý qua các lớp LSTM, mô hình chuyển tiếp thông tin qua lớp fully-connected với hàm kích hoạt ReLU để tìm hiểu các đặc trưng phi tuyến.
- **Lớp Output:** Cuối cùng, đầu ra của mô hình được tính thông qua lớp Dense với hàm kích hoạt softmax, dự đoán phân loại cho ba lớp.

b. Chi tiết kỹ thuật

Model: "lstm_model"

Layer (type)	Output Shape	Param #
embedding_layer (Embedding)	(None, 200, 200)	1000200
lstm_layer_1 (LSTM)	(None, 200, 32)	29824
lstm_layer_2 (LSTM)	(None, 32)	8320
fl_layer_1 (Dense)	(None, 16)	528
output_layer (Dense)	(None, 3)	51

=====
 Total params: 1038923 (3.96 MB)
 Trainable params: 1038923 (3.96 MB)
 Non-trainable params: 0 (0.00 Byte)

Hình 1.5: Cấu hình cài đặt của mô hình LSTM

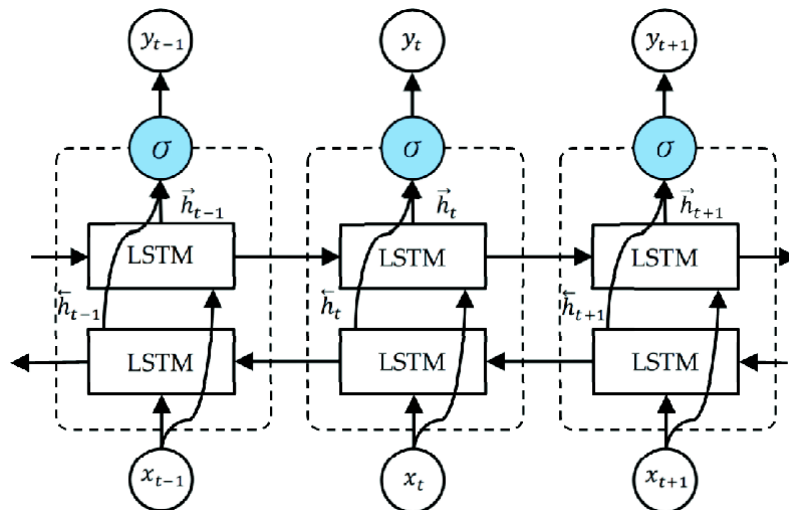
- **Optimizer:** Mô hình sử dụng optimizer Adamax, một biến thể của Adam, giúp tối ưu hóa việc học các tham số.
- **Loss function:** Hàm mất mát được sử dụng là sparse categorical crossentropy, phù hợp với bài toán phân loại đa lớp.
- **Training:** Trong quá trình huấn luyện, mô hình sử dụng các kỹ thuật khởi tạo trọng số như GlorotUniform để tăng tính ổn định và khả năng học tốt hơn.

5.3 Mô hình Bi-LSTM

5.3.1 Khái niệm

Mạng LSTM hai chiều (Bidirectional LSTM - biLSTM) là một biến thể của LSTM, trong đó hai mạng LSTM được huấn luyện song song nhưng theo hai hướng ngược nhau: từ quá khứ đến tương lai (forward) và từ tương lai về quá khứ (backward). Mô hình này giúp mạng có thể học được các phụ thuộc từ cả hai chiều trong chuỗi dữ liệu, từ đó cải thiện khả năng dự đoán trong các bài toán đòi hỏi ngữ cảnh toàn diện.

Kiến trúc biLSTM đặc biệt hiệu quả trong các ứng dụng như nhận diện giọng nói, xử lý ngôn ngữ tự nhiên, nơi mà thông tin quan trọng có thể đến từ cả trước và sau một vị trí trong chuỗi.



Hình 1.6: Kiến trúc biLSTM

Trong đó:

- h_t^{\rightarrow} và h_t^{\leftarrow} tương ứng với hidden state của các LSTM chạy theo chiều thuận và chiều nghịch.
- y_t là kết quả đầu ra được kết hợp từ cả hai chiều.
- $h_t = [h_t^{\rightarrow}; h_t^{\leftarrow}]$, đây là vector kết hợp giữa hidden state theo chiều thuận và chiều nghịch.

Nhận xét: biLSTM giúp mạng học được thông tin từ cả hai chiều của chuỗi, phù hợp với các bài toán yêu cầu sự hiểu biết ngữ cảnh toàn diện. Tuy nhiên, việc sử dụng biLSTM cũng có thể làm tăng thời gian tính toán do phải huấn luyện hai mạng LSTM song song.

5.3.2 Cài đặt thí nghiệm

a. Cấu trúc mô hình

Mô hình Bi-LSTM được xây dựng gồm các thành phần chính sau:

- **Lớp Embedding:** Đầu vào của mô hình là các chuỗi ký tự hoặc từ, được ánh xạ sang không gian vector thông qua lớp Embedding. Việc khởi tạo các vector embedding được thực hiện bằng hàm `RandomUniform`, đảm bảo sự ổn định trong quá trình huấn luyện.
- **Lớp Bi-LSTM:** Mô hình sử dụng hai lớp LSTM hai chiều (Bidirectional LSTM).
 - ◊ **Bi-LSTM Layer 1:** Lớp Bi-LSTM đầu tiên trả về toàn bộ chuỗi hidden states (`return_sequences=True`). Điều này giúp mô hình giữ lại thông tin ở mọi bước thời gian.
 - ◊ **Bi-LSTM Layer 2:** Lớp Bi-LSTM thứ hai chỉ trả về hidden state cuối cùng của chuỗi (`return_sequences=False`), tập trung vào thông tin quan trọng nhất ở cuối chuỗi.
- **Lớp Fully-Connected:** Sau khi thông tin được xử lý qua các lớp Bi-LSTM, mô hình tiếp tục qua lớp fully-connected với hàm kích hoạt ReLU để xử lý các đặc trưng phi tuyến.
- **Lớp Output:** Đầu ra của mô hình là một lớp Dense với hàm kích hoạt softmax, giúp phân loại chuỗi dữ liệu đầu vào thành một trong ba lớp.

b. Chi tiết kỹ thuật

Model: "bilstm_model"

Layer (type)	Output Shape	Param #
embedding_layer (Embedding)	(None, 200, 200)	1000200
bilstm_layer_1 (Bidirectional)	(None, 200, 64)	59648
bilstm_layer_2 (Bidirectional)	(None, 64)	24832
fl_layer_1 (Dense)	(None, 16)	1040
output_layer (Dense)	(None, 3)	51

=====
Total params: 1085771 (4.14 MB)
Trainable params: 1085771 (4.14 MB)
Non-trainable params: 0 (0.00 Byte)
=====

Hình 1.7: Cấu hình cài đặt của mô hình Bi-LSTM

- **Optimizer:** Mô hình sử dụng optimizer Adamax, một biến thể của Adam, giúp tối ưu hóa hiệu quả việc học các tham số.
- **Loss function:** Hàm mất mát được sử dụng là sparse categorical crossentropy, phù hợp cho các bài toán phân loại nhiều lớp.

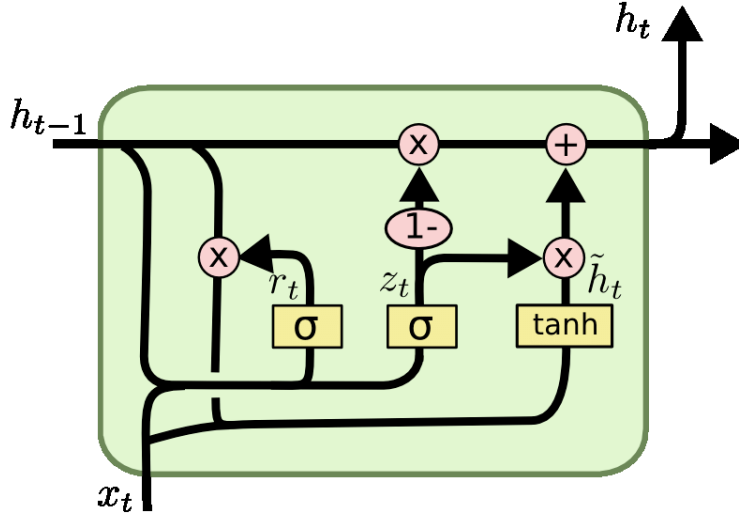
5.4 Mô hình GRU

5.4.1 Khái niệm

Mạng Gated Recurrent Unit (GRU) là một biến thể của RNN, được giới thiệu nhằm cải thiện các hạn chế về việc mất mát thông tin trong các chuỗi dữ liệu dài. GRU đơn giản hơn LSTM với ít cổng hơn, nhưng vẫn có khả năng học được sự phụ thuộc trong dài hạn (long-term dependencies).

GRU được thiết kế để giải quyết các vấn đề tương tự như LSTM, nhưng với cấu trúc đơn giản hơn, điều này giúp cho mô hình GRU tính toán nhanh hơn trong quá trình huấn luyện và dự đoán.

Thay vì sử dụng ba cổng như trong LSTM (forget gate, input gate, output gate), GRU chỉ sử dụng hai cổng chính là cổng cập nhật (update gate) và cổng xoá (reset gate).



Hình 1.8: Kiến trúc GRU

Trong đó:

- z_t, r_t tương ứng với update gate và reset gate.
 - $z_t = \sigma(U_z * x_t + W_z * h_{t-1} + b_z)$
 - $r_t = \sigma(U_r * x_t + W_r * h_{t-1} + b_r)$
- $\tilde{h}_t = \tanh(U_h * x_t + W_h * (r_t \odot h_{t-1}) + b_h)$, bước này giống như tính s_t trong RNN.
- $h_t = z_t * h_{t-1} + (1 - z_t) * \tilde{h}_t$, cổng cập nhật sẽ quyết định giữ lại bao nhiêu từ trạng thái trước đó và bao nhiêu từ trạng thái hiện tại sẽ được thêm vào.

Nhận xét: GRU đơn giản hơn LSTM về mặt cấu trúc nhưng vẫn duy trì được khả năng học long-term dependencies. Do đó, GRU thường được sử dụng trong các bài toán yêu cầu tốc độ tính toán nhanh hơn mà vẫn đạt hiệu suất cao.

5.4.2 Cài đặt thí nghiệm

a. Cấu trúc mô hình

Mô hình GRU được xây dựng gồm các thành phần chính sau:

- **Lớp Embedding:** Đầu vào của mô hình là các chuỗi ký tự hoặc từ, được ánh xạ sang không gian vector thông qua lớp Embedding. Việc khởi tạo các vector embedding được thực hiện bằng hàm `RandomUniform`, đảm bảo sự ổn định trong quá trình huấn luyện.

- **Lớp GRU:** Mô hình sử dụng hai lớp GRU (Gated Recurrent Unit).
 - ◊ **GRU Layer 1:** Lớp GRU đầu tiên trả về toàn bộ chuỗi hidden states (`return_sequences=True`), giúp giữ lại thông tin tại mỗi bước thời gian.
 - ◊ **GRU Layer 2:** Lớp GRU thứ hai chỉ trả về hidden state cuối cùng (`return_sequences=False`), giúp tập trung vào thông tin quan trọng nhất ở cuối chuỗi.
- **Lớp Fully-Connected:** Sau khi thông tin được xử lý qua các lớp GRU, mô hình chuyển qua lớp fully-connected với hàm kích hoạt ReLU để trích xuất các đặc trưng phi tuyến.
- **Lớp Output:** Đầu ra của mô hình là một lớp Dense với hàm kích hoạt softmax, giúp phân loại chuỗi dữ liệu đầu vào thành một trong ba lớp.

b. Chi tiết kỹ thuật

Model: "gru_model"

Layer (type)	Output Shape	Param #
embedding_layer (Embedding)	(None, 200, 200)	1000200
rnn_layer_1 (GRU)	(None, 200, 32)	22464
rnn_layer_2 (GRU)	(None, 32)	6336
fl_layer_1 (Dense)	(None, 16)	528
output_layer (Dense)	(None, 3)	51

=====
 Total params: 1029579 (3.93 MB)
 Trainable params: 1029579 (3.93 MB)
 Non-trainable params: 0 (0.00 Byte)

Hình 1.9: Cấu hình cài đặt của mô hình GRU

- **Optimizer:** Mô hình sử dụng optimizer Adamax, một biến thể của Adam, giúp tối ưu hóa quá trình học tập.
- **Loss function:** Hàm mất mát được sử dụng là sparse categorical crossentropy, phù hợp cho bài toán phân loại nhiều lớp.

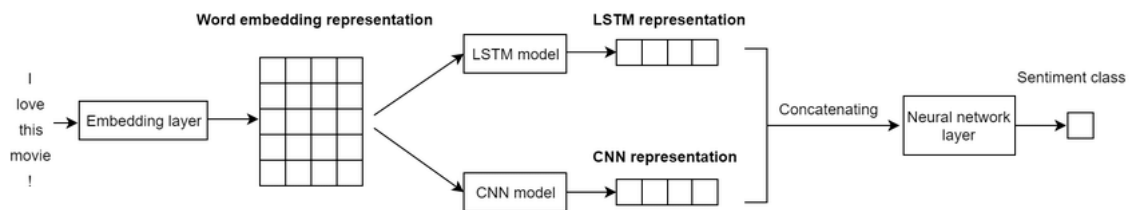
- **Training:** Mô hình sử dụng kỹ thuật khởi tạo trọng số GlorotUniform, giúp đảm bảo việc khởi tạo các trọng số diễn ra một cách ổn định và hiệu quả.

5.5 Mô hình kết hợp CNN và LSTM [8]

5.5.1 Khái niệm

Mô hình kết hợp giữa Convolutional Neural Network (CNN) và Long Short-Term Memory (LSTM) là một giải pháp mạnh mẽ cho các bài toán xử lý chuỗi dữ liệu, đặc biệt là văn bản. Mô hình này kết hợp khả năng trích xuất đặc trưng cục bộ của CNN với khả năng học các phụ thuộc theo thời gian của LSTM.

CNN có khả năng phát hiện các đặc trưng quan trọng từ dữ liệu, trong khi LSTM có khả năng ghi nhớ các thông tin quan trọng trong dài hạn từ chuỗi thời gian. Việc kết hợp hai mô hình này tạo ra một kiến trúc mạnh mẽ, phù hợp cho các bài toán yêu cầu cả việc trích xuất đặc trưng không gian và thời gian.



Hình 1.10: Kiến trúc kết hợp CNN và LSTM

Nhận xét: Mô hình kết hợp CNN và LSTM giúp khai thác tốt nhất cả các đặc trưng cục bộ và các phụ thuộc dài hạn trong chuỗi dữ liệu, mang lại hiệu suất cao cho các bài toán như phân loại văn bản hay chuỗi thời gian.

5.5.2 Cài đặt thí nghiệm

a. Cấu trúc mô hình

Mô hình được xây dựng gồm các thành phần chính sau:

- **Lớp Embedding:** Đầu vào của mô hình là các chuỗi ký tự hoặc từ, được ánh xạ sang không gian vector thông qua lớp Embedding.

- **Kênh CNN:** Sau lớp Embedding, một chuỗi các lớp tích chập 1D (Conv1D) được áp dụng với các bộ lọc có kích thước khác nhau. Các lớp này giúp trích xuất các đặc trưng cục bộ từ chuỗi dữ liệu.
 - ◊ Mỗi lớp Conv1D được theo sau bởi lớp MaxPooling1D để giảm kích thước đầu ra và tập trung vào các đặc trưng quan trọng nhất.
 - ◊ Kết quả của các lớp Conv1D và MaxPooling1D sau đó được làm phẳng và kết hợp lại thành một đầu ra duy nhất bằng lớp Concatenate.
- **Kênh LSTM:** Song song với kênh CNN, lớp LSTM được áp dụng lên dữ liệu sau lớp Embedding để học các phụ thuộc dài hạn trong chuỗi dữ liệu.
- **Kết hợp kênh CNN và LSTM:** Đầu ra từ kênh CNN và LSTM được kết hợp lại bằng lớp Concatenate, tạo thành một đầu vào duy nhất cho lớp Mạng Nơ-ron (Neural Network).
- **Lớp Dense và Output:** Đầu vào từ bước trên được đưa qua một lớp Dense có hàm kích hoạt sigmoid, sau đó là lớp Dropout để giảm thiểu hiện tượng overfitting. Cuối cùng, lớp Dense với hàm kích hoạt softmax được sử dụng để dự đoán phân loại cho ba lớp.

b. Chi tiết kỹ thuật

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
main_input (InputLayer)	[(None, 200)]	0	[]
embedding (Embedding)	(None, 200, 200)	1135200	['main_input[0][0]']
conv1d (Conv1D)	(None, 198, 150)	90150	['embedding[0][0]']
conv1d_1 (Conv1D)	(None, 196, 150)	150150	['embedding[0][0]']
conv1d_2 (Conv1D)	(None, 194, 150)	210150	['embedding[0][0]']
max_pooling1d (MaxPooling1D)	(None, 1, 150)	0	['conv1d[0][0]']
max_pooling1d_1 (MaxPooling1D)	(None, 1, 150)	0	['conv1d_1[0][0]']
max_pooling1d_2 (MaxPooling1D)	(None, 1, 150)	0	['conv1d_2[0][0]']
flatten (Flatten)	(None, 150)	0	['max_pooling1d[0][0]']
flatten_1 (Flatten)	(None, 150)	0	['max_pooling1d_1[0][0]']
flatten_2 (Flatten)	(None, 150)	0	['max_pooling1d_2[0][0]']
lstm (LSTM)	(None, 128)	168448	['embedding[0][0]']
concatenate (Concatenate)	(None, 450)	0	['flatten[0][0]', 'flatten_1[0][0]', 'flatten_2[0][0]']
concatenate_1 (Concatenate)	(None, 578)	0	['lstm[0][0]', 'concatenate[0][0]']
dense (Dense)	(None, 200)	115800	['concatenate_1[0][0]']
dropout (Dropout)	(None, 200)	0	['dense[0][0]']
dense_1 (Dense)	(None, 3)	603	['dropout[0][0]']

=====
Total params: 1870501 (7.14 MB)

Trainable params: 1870501 (7.14 MB)

Non-trainable params: 0 (0.00 Byte)

Hình 1.11: Cấu hình cài đặt của mô hình kết hợp CNN và LSTM

- **Optimizer:** Mô hình sử dụng optimizer Adamax, một biến thể của Adam, giúp tối ưu hóa việc học các tham số.
- **Loss function:** Hàm mất mát được sử dụng là sparse categorical crossentropy, phù hợp với bài toán phân loại đa lớp.
- **Training:** Trong quá trình huấn luyện, mô hình sử dụng các kỹ thuật như Dropout để tránh overfitting và tăng tính tổng quát hóa của mô

hình.

6 Bộ dữ liệu

Đường dẫn: [30K e-commerce reviews](#)

Mô tả: Bộ dữ liệu Vietnamese Sentiment Analysis trên Kaggle là một tập dữ liệu được thiết kế để hỗ trợ các nghiên cứu và ứng dụng trong lĩnh vực phân tích cảm xúc đối với văn bản tiếng Việt.

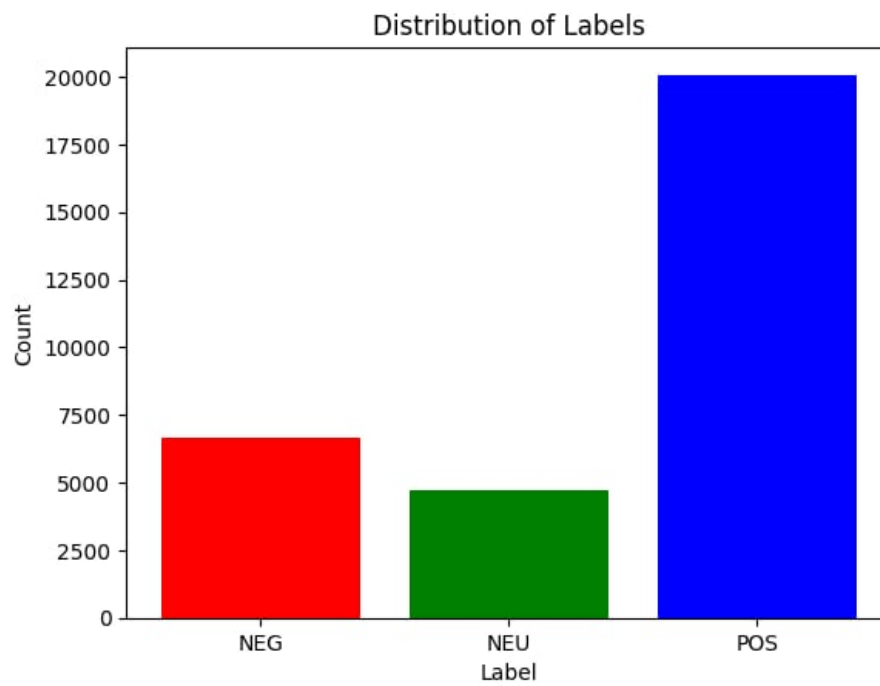
Bộ dữ liệu này bao gồm các bình luận hoặc văn bản tiếng Việt được gán nhãn với các nhãn cảm xúc khác nhau. Các nhãn này thường bao gồm:

- POS (Tích cực): Các bình luận thể hiện cảm xúc tích cực.
- NEG (Tiêu cực): Các bình luận thể hiện cảm xúc tiêu cực.
- NEU (Trung tính): Các bình luận không mang cảm xúc rõ ràng hoặc trung tính.

	content	label	start
0	Áo bao đẹp ạ!	POS	5
1	Tuyệt vời	POS	5
2	2day ao khong giiong trong	NEG	1
3	Mùi thơm,bôi lên da mềm da	POS	5
4	Vải đẹp, dày dặn	POS	5
...
31455	Không đáng tiền	NEG	1
31456	Quần rất đẹp	POS	5
31457	Hàng đẹp đúng giá tiền	POS	5
31458	Chất vải khá ổn	POS	4
31459	áo rất ok nhé , vải mịn , len cao cổ này phối ...	POS	5

[31436 rows x 3 columns]

Hình 1.12: Dữ liệu gốc

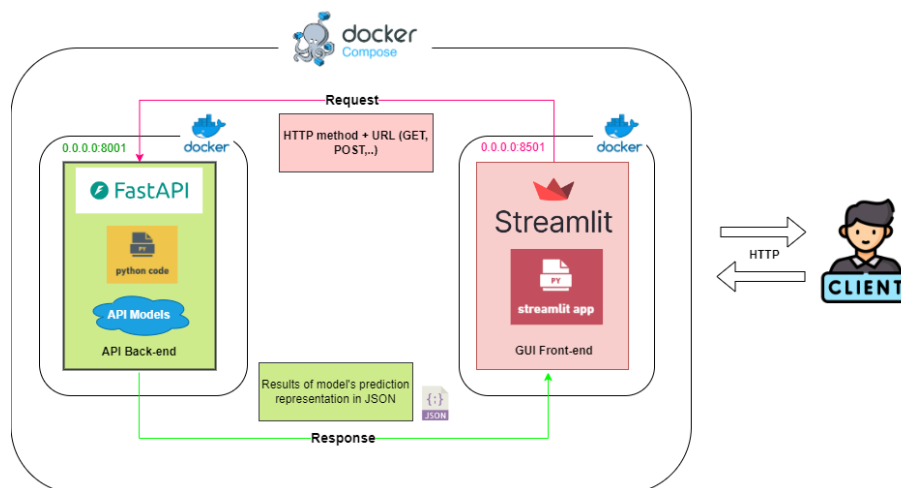


Hình 1.13: Biểu đồ phân phối nhãn

Chương 2

Cách thiết lập

1 Kiến trúc của ứng dụng



Hình 2.1: Sơ đồ quy trình giao tiếp giữa Client, Streamlit, FastAPI, và Model

1.1 Mô Tả Quy Trình

1. **Người dùng (Client):** Nhập văn bản vào giao diện người dùng.
2. **Streamlit [7]:**
 - Nhận dữ liệu từ người dùng.
 - Gửi yêu cầu POST chứa dữ liệu tới FastAPI.
3. **FastAPI [3]:**
 - Nhận yêu cầu từ Streamlit.

- Xử lý dữ liệu, thực hiện phân tích bằng mô hình.
- Gửi kết quả phân tích trở lại Streamlit.

4. Mô hình:

- Nhận dữ liệu từ FastAPI.
- Thực hiện phân tích và gửi kết quả phân tích trở lại FastAPI.

5. Streamlit:

- Nhận kết quả từ FastAPI.
- Hiển thị kết quả phân tích cho người dùng.

1.2 Mô tả cài đặt Docker [2]

1.2.1 Docker Images

- **Docker Image cho Streamlit (GUI):**
 - Chứa ứng dụng Streamlit và các thư viện cần thiết.
 - Cung cấp giao diện người dùng để nhập văn bản và hiển thị kết quả.
- **Docker Image cho FastAPI (API):**
 - Chứa ứng dụng FastAPI và các thư viện cần thiết.
 - Cung cấp API để xử lý yêu cầu từ Streamlit và giao tiếp với mô hình.

1.2.2 Docker Containers

- **Container cho Streamlit (GUI):**
 - Chạy Docker image chứa ứng dụng Streamlit.
 - Cung cấp giao diện người dùng tại cổng 8501.
- **Container cho FastAPI (API):**
 - Chạy Docker image chứa ứng dụng FastAPI.
 - Cung cấp API tại cổng 8001.
- **Container cho Model (Model):**

- Chạy Docker image chứa mô hình và các thư viện cần thiết.
- Kết nối với FastAPI để thực hiện phân tích dữ liệu.

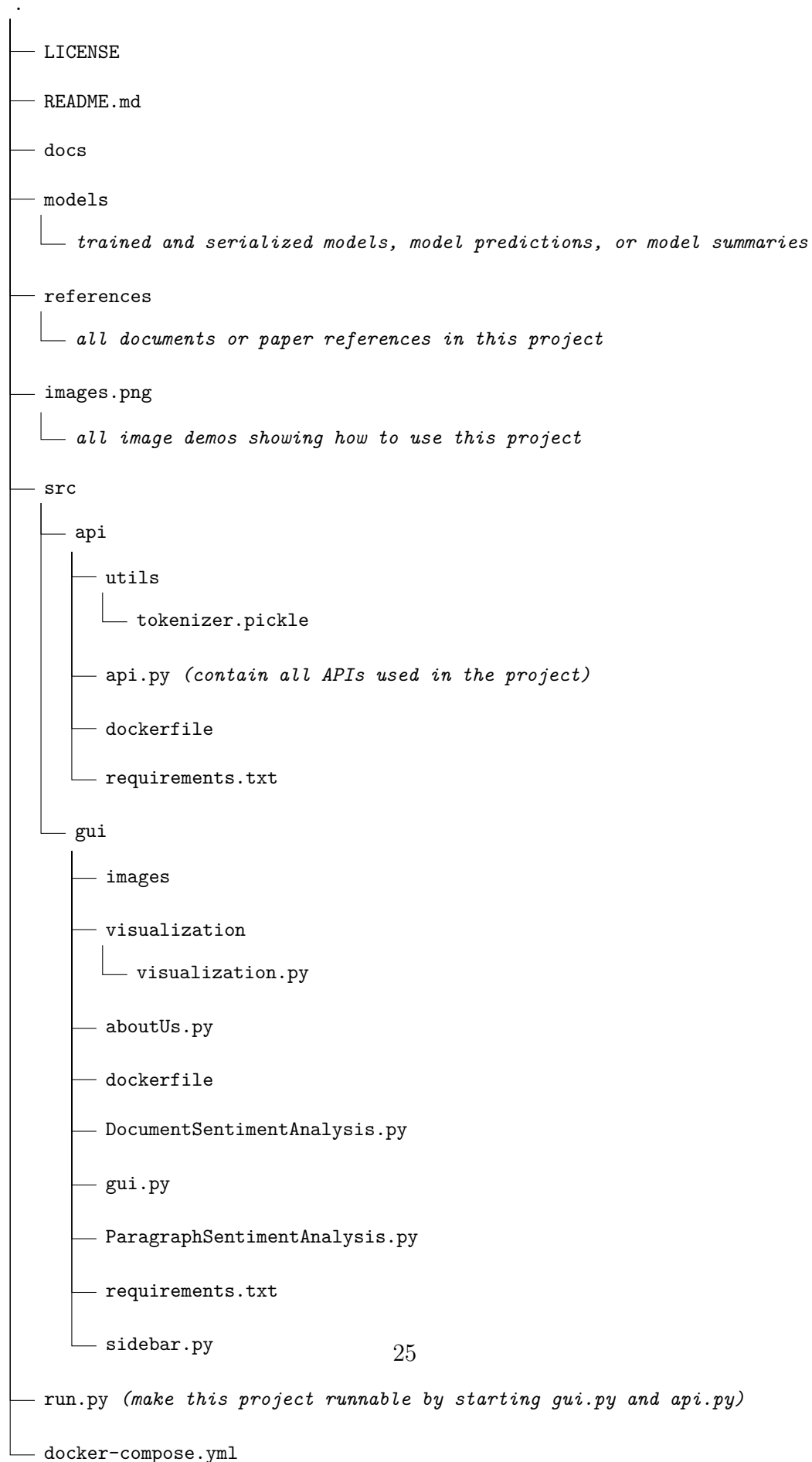
1.2.3 Kết Nối Docker Containers

– Docker Compose:

- Sử dụng Docker Compose để cấu hình và khởi động các container cùng lúc.
- Định nghĩa các dịch vụ (Streamlit, FastAPI, Model) và các kết nối mạng giữa chúng.

2 Cách chạy chương trình

2.1 Cấu trúc thư mục



2.2 Hướng dẫn chạy chương trình

Cách 1: Sử dụng Docker [4]

1. Clone repository này
2. Để khởi động ứng dụng, chạy lệnh sau trong terminal:

```
docker-compose up -d --build
```

3. Truy cập ứng dụng bằng cách mở trình duyệt và điều hướng đến <http://localhost:8501/>.
4. Để dừng ứng dụng, chạy lệnh sau:

```
docker-compose down
```

Hoặc, nếu bạn đang chạy Docker Compose từ terminal, bạn có thể dừng ứng dụng bằng tổ hợp phím Ctrl + C.

Cách 2: Không sử dụng Docker

1. Cài đặt các gói phụ thuộc từ file `requirements.txt` trong hai thư mục `src/gui` và `src/api`:

```
pip install -r src/gui/requirements.txt
pip install -r src/api/requirements.txt
```

2. Chạy lệnh sau để khởi động ứng dụng:

```
python run.py
```

3 Cách huấn luyện

3.1 Tổng quan

Chạy các tệp huấn luyện cho từng mô hình được lưu tại:

'src/api/models/train_<model_name>_model.ipynb'

Với <model_name> bao gồm:

- Mô hình LSTM: 'lstm'
- Mô hình Bi-LSTM: 'bilstm'
- Mô hình GRU: 'gru'
- Mô hình kết hợp CNN và LSTM: 'lstm_cnn'

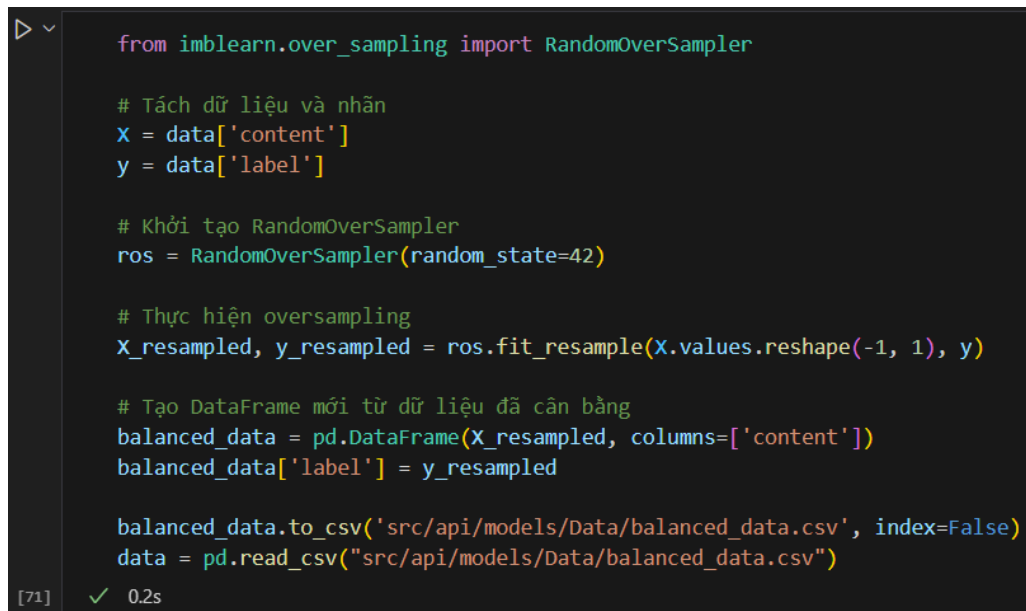
Đối với mỗi tập huấn luyện bao gồm các bước chính: đọc dữ liệu, tiền xử lý, thiết lập mô hình, huấn luyện, kiểm tra và đánh giá.

Chú ý: Đối với mô hình `wonrax/phobert-base-vietnamese-sentiment` đã được tinh chỉnh sẵn và gọi từ **Hugging Face**.

3.2 Qui trình các bước

3.2.1 Tiền xử lý

Cân bằng dữ liệu bằng việc sử dụng `RandomOverSampler` từ thư viện `imbalanced-learn` để thực hiện oversampling cho nhãn ít. Cell thực hiện:



```
from imblearn.over_sampling import RandomOverSampler

# Tách dữ liệu và nhãn
X = data['content']
y = data['label']

# Khởi tạo RandomOverSampler
ros = RandomOverSampler(random_state=42)

# Thực hiện oversampling
X_resampled, y_resampled = ros.fit_resample(X.values.reshape(-1, 1), y)

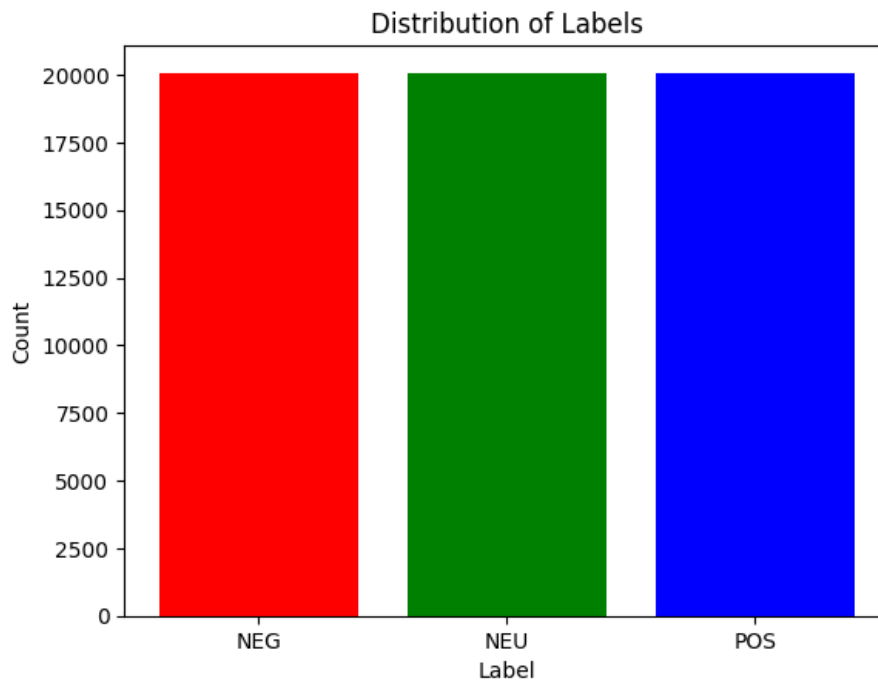
# Tạo DataFrame mới từ dữ liệu đã cân bằng
balanced_data = pd.DataFrame(X_resampled, columns=['content'])
balanced_data['label'] = y_resampled

balanced_data.to_csv('src/api/models/Data/balanced_data.csv', index=False)
data = pd.read_csv("src/api/models/Data/balanced_data.csv")
```

[71] ✓ 0.2s

Hình 2.2: Cell cân bằng dữ liệu

Dữ liệu sau khi cân bằng:



Hình 2.3: Biểu đồ phân phối nhãn sau khi cân bằng

Áp dụng các kĩ thuật xử lý văn bản cho dữ liệu bao gồm chuyển đổi thành chữ thường, chuẩn hóa, phân đoạn từ và tokenization. Đồng thời khai báo các tham số cần thiết như:

- `MAX_TOKENS`: kích thước từ vựng tối đa
- `MAX_LEN`: độ dài chuỗi để đệm đầu ra
- `EMBEDDING_DIMS`: kích thước embedding
- `kernel_sizes`: danh sách các kích thước của bộ lọc trong lớp tích chập.
- `n_filter`: số lượng bộ lọc trong lớp tích chập.

```

MAX_TOKENS = 5000 # Maximum vocab size
MAX_LEN = 200 # Sequence length to pad the outputs to
EMBEDDING_DIMS = 200
RANDOM_SEED = 240

def text_lowercase(text):
    return text.lower()

data['content'] = data['content'].apply(text_lowercase)

# Text normalization
data['normalized'] = data['content'].apply(underthesea.text_normalize)

# Word segmentation
data['segmented_sentence'] = data['normalized'].apply(underthesea.word_tokenize)

# Sentence length distribution
data['word_count'] = data['segmented_sentence'].apply(len)

# Tokenization
def to_original(segmented_sentence):
    return ' '.join([sub.replace(' ', '_') for sub in segmented_sentence])

data['processed_sentence'] = data['segmented_sentence'].apply(to_original)

```

[73] ✓ 33.0s

Hình 2.4: Xử lý văn bản

Kết quả sau khi được xử lí:

```

                                processed_sentence
0                                áo bao đẹp ạ !
1                                tuyệt_vời
2                                2 day ao khong_giong trong
3                                mùi thơm , bôi lên da mềm da
4                                vải đẹp , dày dặn
...
60229                            từ ngày 7/9 hnay là 22 / 9 rồi
60230                            mọi người có_thể yên_tâm ủng_hộ shop này
60231    k có tông_xanh , tím . 3 cái màu hồng giống nhau
60232    sp k có vấn_đề nhưng shipper cua shopee thô_lỗ...
60233                            sẽ ủng_hộ

```

Hình 2.5: Kết quả xử lí

Xây dựng từ điển từ vựng từ các câu đã được xử lý bằng phương thức `fit_on_texts`. Mỗi từ sẽ được gán một chỉ số duy nhất dựa trên tần suất xuất hiện của nó trong tập dữ liệu. Cell thực hiện:

```
tokenizer = Tokenizer(num_words=MAX_TOKENS, lower=True)
tokenizer.fit_on_texts(data['processed_sentence'])

tokenizer_file_path = 'src/api/utils/tokenizer.pickle'
with open(tokenizer_file_path, 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

[10] ✓ 0.5s

Hình 2.6: Xây dựng từ điển từ vựng

Mã hóa các nhãn thành số nguyên và sử dụng phương thức `train_test_split` để chia tập dữ liệu thành các tập huấn luyện, xác thực và kiểm tra. Cells thực hiện:

```
X_data = data["processed_sentence"]
y_data = []
for label in data["label"]:
    if label == "NEG":
        y_data.append(0)
    elif label == "NEU":
        y_data.append(1)
    elif label == "POS":
        y_data.append(2)
```

[11] ✓ 0.0s

```
X_temp, X_test, y_temp, y_test = train_test_split(X_data, y_data, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.25, random_state=42)

# Kết quả cuối cùng
print("Kích thước tập huấn luyện:", X_train.shape)
print("Kích thước tập xác thực:", X_val.shape)
print("Kích thước tập kiểm tra:", X_test.shape)
```

[12] ✓ 0.0s

Hình 2.7: Mã hóa nhãn và chia tập dữ liệu

Cell tiếp theo thực hiện tokenize các câu, tạo các DataFrame cho tập huấn luyện, xác thực và kiểm tra, và cuối cùng chuyển đổi dữ liệu sang định dạng NumPy array:

- Sử dụng phương thức `texts_to_sequences` để chuyển đổi các câu thành các chỉ số số nguyên tương ứng với các từ trong từ điển của tokenizer.
- Sử dụng phương thức `pad_sequences` để đảm bảo rằng tất cả các câu đều có độ dài bằng nhau bằng cách cắt giảm hoặc đệm thêm các số 0.
- Sử dụng phương thức `sample` để xáo trộn các hàng trong mỗi DataFrame.

```
# Tokenize sentences
def tokenize_sentences(sentences):
    return pad_sequences(tokenizer.texts_to_sequences(sentences), maxlen=MAX_LEN)

# Tạo DataFrame từ các tập dữ liệu
train_df = pd.DataFrame({'processed_sentence': X_train, 'sentiment': y_train})
validation_df = pd.DataFrame({'processed_sentence': X_val, 'sentiment': y_val})
test_df = pd.DataFrame({'processed_sentence': X_test, 'sentiment': y_test})

# Tokenize các câu
for df in [train_df, validation_df, test_df]:
    df['tokenized'] = list(tokenize_sentences(df['processed_sentence'].tolist()))

# Xáo trộn dữ liệu
for df in [train_df, validation_df, test_df]:
    df.sample(frac=1, random_state=1).reset_index(drop=True)

# Chuyển đổi dữ liệu sang NumPy array
X_train = np.array(train_df['tokenized'].tolist())
y_train = train_df['sentiment'].values

X_val = np.array(validation_df['tokenized'].tolist())
y_val = validation_df['sentiment'].values

X_test = np.array(test_df['tokenized'].tolist())
y_test = test_df['sentiment'].values
```

[13] ✓ 0.6s

Hình 2.8: Tokenize và Padding

3.2.2 Thiết lập mô hình

Cell xây dựng mô hình LSTM (tương tự đối với các mô hình khác):


```

lstm = Sequential([
    Input(shape=(MAX_LEN,)), name='input_layer'),

    # Embedding Layer (converts tokens into vectors)
    Embedding(input_dim=MAX_TOKENS + 1,
               output_dim=EMBEDDING_DIMS,
               embeddings_initializer=RandomUniform(seed=RANDOM_SEED),
               name='embedding_layer'),

    # RNN Layer 1
    LSTM(32,
         return_sequences=True,
         kernel_initializer=GlorotUniform(seed=RANDOM_SEED),
         name='lstm_layer_1'),

    # RNN Layer 2
    LSTM(32,
         return_sequences=False,
         kernel_initializer=GlorotUniform(seed=RANDOM_SEED),
         name='lstm_layer_2'),

    # Fully-connected Layer 1
    Dense(16,
          activation='relu',
          kernel_initializer=GlorotUniform(seed=RANDOM_SEED),
          name='f1_layer_1'),

    # Output Layer
    Dense(3,
          activation='softmax',
          kernel_initializer=GlorotUniform(seed=RANDOM_SEED),
          name='output_layer')
]),
name='lstm_model')
lstm.compile(loss='sparse_categorical_crossentropy', optimizer="adamax", metrics=['accuracy'])
lstm.summary()

```

[80] ✓ 0.6s

Hình 2.9: Kiến trúc mô hình LSTM

Cell huấn luyện mô hình LSTM và in biểu đồ theo dõi accuracy và loss (tương tự đối với các mô hình khác):

```
# Train model
checkpoint = ModelCheckpoint('src/api/checkpoints/lstm.keras', monitor='val_accuracy', verbose=0, save_best_only=True, mode='max')

history = lstm.fit(X_train, y_train, batch_size=64, epochs=20, validation_data=(X_val, y_val), callbacks=[checkpoint])
fig, axes = plt.subplots(2, 1, figsize=(10, 10))

# Plotting the accuracy
axes[0].plot(history.history['accuracy'], label='Training Accuracy')
axes[0].plot(history.history['val_accuracy'], label='Validation Accuracy')
axes[0].set_title('Training and Validation Accuracy')
axes[0].set_xlabel('Epochs')
axes[0].set_ylabel('Accuracy')
axes[0].legend()
axes[0].grid()

# Plotting the loss
axes[1].plot(history.history['loss'], label='Training Loss')
axes[1].plot(history.history['val_loss'], label='Validation Loss')
axes[1].set_title('Training and Validation Loss')
axes[1].set_xlabel('Epochs')
axes[1].set_ylabel('Loss')
axes[1].legend()
axes[1].grid()

# Adjust layout
plt.tight_layout()
plt.show()
```

Hình 2.10: Huấn luyện LSTM và in biểu đồ Accuracy, Loss

4 Cách kiểm tra và đánh giá

Trong cùng tệp huấn luyện tại **mục 2** sau khi quá trình huấn luyện kết thúc chạy cell dưới đây để in ra Classification report và Confusion matrix:

```
# Evaluate model
lstm.load_weights('src/api/checkpoints/lstm.keras')
lstm.compile(loss='sparse_categorical_crossentropy', optimizer="adamax", metrics=['accuracy'])

predictions = lstm.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)

# Print classification report and confusion matrix
print(classification_report(y_test, predicted_classes))

cm = confusion_matrix(y_test, predicted_classes)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()
```

Hình 2.11: Classification report và Confusion matrix

Chương 3

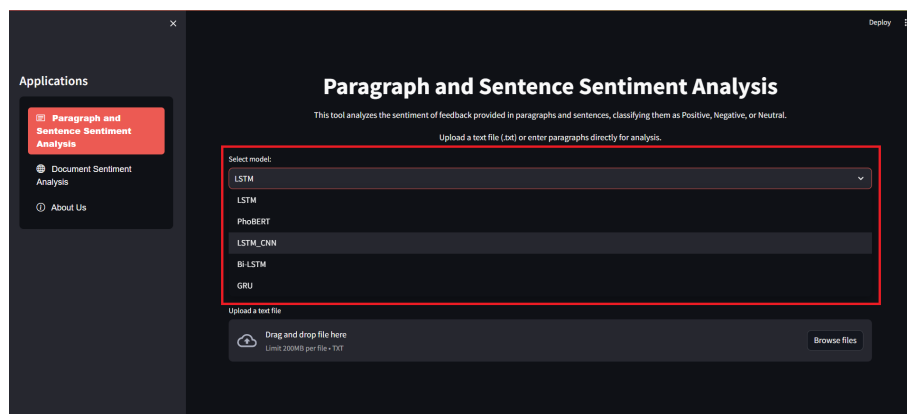
Cách sử dụng

1. Tab 1: Phân Tích Theo Câu

Ứng dụng hỗ trợ phân tích cảm xúc theo từng câu nối tiếp nhau hoặc từng câu riêng lẻ.

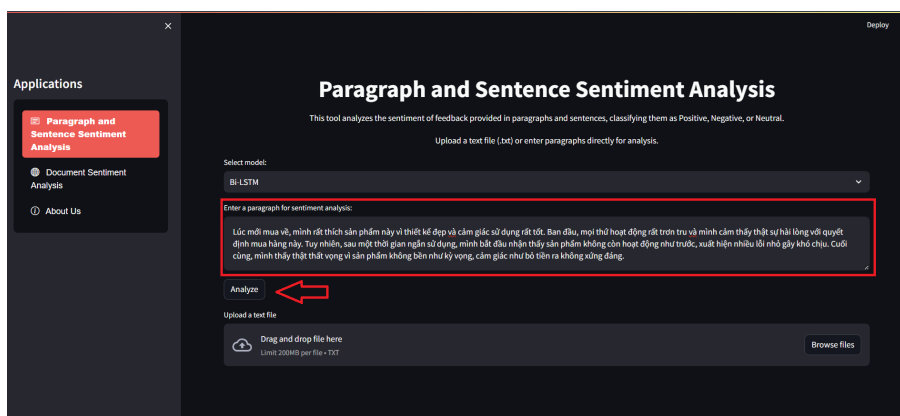
Phân Tích Chuyển Tiếp Các Câu Liên Tục

1. **Chọn model phân tích:** Chọn model bạn muốn sử dụng từ danh sách có sẵn.

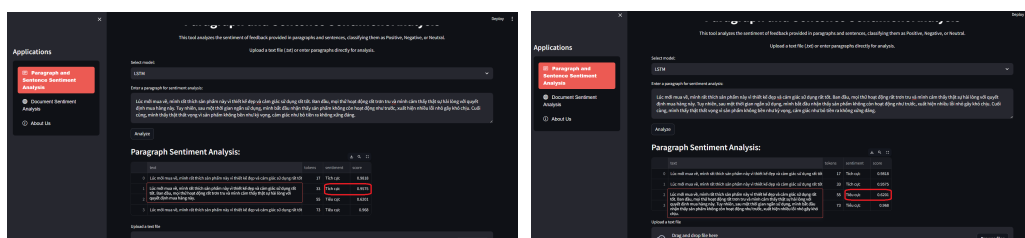


2. Nhập đoạn text và phân tích:

- Nhập đoạn text bạn muốn phân tích vào ô nhập liệu.
- Bấm vào nút "Analyze" để bắt đầu phân tích.



3. Kết quả phân tích:

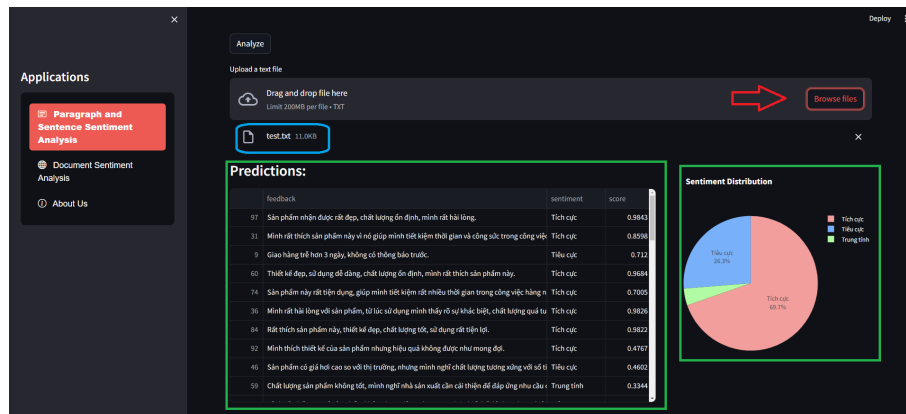


Nhận xét: Mô hình đã nhận diện tốt sự chuyển tiếp cảm xúc trong đoạn văn. Ví dụ, câu đầu tiên được phân loại là tích cực với điểm số cao 0.9575, thể hiện sự hài lòng rõ ràng. Tuy nhiên, ở câu tiếp theo, cảm xúc tiêu cực với điểm số 0.6201 được phát hiện khi sản phẩm bắt đầu gặp vấn đề, chứng tỏ mô hình có khả năng phân tích và phản ánh sự thay đổi cảm xúc hiệu quả.

Phân Tích Theo Từng Câu Và Thống Kê Cảm Xúc Cho Scopus

Chức năng này cho phép bạn phân tích cảm xúc từng câu trong một tài liệu Scopus.

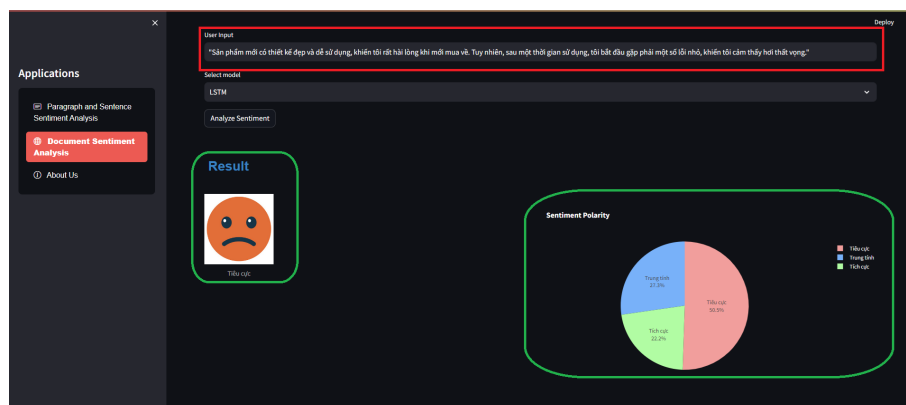
- 1. Tải lên file văn bản (.txt):** Chứa nội dung Scopus mà bạn muốn phân tích.
- 2. Kết quả phân tích:** Kết quả sẽ hiển thị phân tích cảm xúc cho từng câu trong tài liệu ở bên trái, cùng với biểu đồ tròn minh họa tỷ lệ các nhãn cảm xúc: Tiêu cực, Trung tính và Tích cực.



2. Tab 2: Phân Tích Theo Document

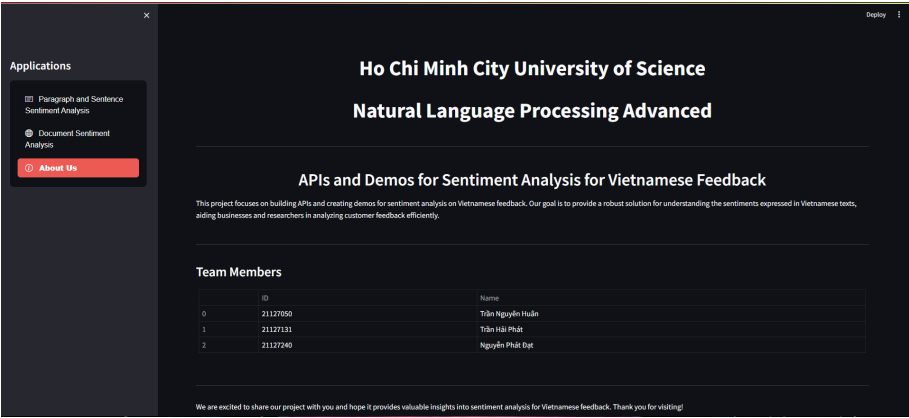
Ứng dụng tập trung hỗ trợ phân tích cho toàn bộ document và trả ra số điểm (tỉ lệ) cho từng cảm xúc mà mô hình dự đoán cho document tương ứng.

1. Chọn model phân tích.
2. Nhập text trực tiếp hoặc upload file txt document.
3. Bấm vào nút "Analyze" để bắt đầu phân tích.
4. **Kết quả phân tích:** Kết quả trả về sẽ cho biết nhãn có tỉ lệ dự đoán cao nhất và thống kê tỉ lệ từng nhãn mà mô hình dự đoán được.



3. Tab 3: Thông Tin Nhóm

Ngoài ra, nếu muốn biết thêm thông tin của nhóm, bạn có thể chuyển sang tab "About Us".



Chương 4

Thông số kỹ thuật API

1 Mô tả API phân tích theo tập tài liệu (document)

Endpoint: /lstm_document, /phobert_document, /lstm_cnn_document, /bilstm_document, /gru_document

Phương thức HTTP: POST

Mô hình sử dụng:

- /lstm_document: Mô hình LSTM.
- /phobert_document: Mô hình PhoBERT.
- /lstm_cnn_document: Mô hình LSTM-CNN.
- /bilstm_document: Mô hình BiLSTM.
- /gru_document: Mô hình GRU.

Yêu cầu:

- **Body:** Một đối tượng `TextRequest` chứa văn bản tài liệu cần phân tích cảm xúc.
- **Cấu trúc JSON:**

```
{  
  "text": "Toàn bộ nội dung tài liệu cần phân tích cảm xúc."  
}
```

Quá trình:

1. **Chuyển chuỗi đầu vào thành chữ thường:** Tất cả các ký tự trong văn bản được chuyển thành chữ thường để đồng nhất.
2. **Chuẩn hóa văn bản:** Loại bỏ các ký tự không cần thiết như dấu câu, các ký tự đặc biệt.
3. **Phân đoạn từ:** Tách văn bản thành các từ hoặc cụm từ nhỏ hơn.
4. **Định dạng lại từ đã phân đoạn:** Thay thế các khoảng trắng giữa các từ bằng dấu gạch dưới, nếu cần thiết theo yêu cầu của mô hình.
5. **Token hóa và đệm chuỗi:** Chuyển đổi các từ thành các chỉ số số học và đảm bảo tất cả các chuỗi đều có độ dài đồng nhất bằng cách thêm các giá trị đệm nếu cần.
6. **Dự đoán cảm xúc:** Sử dụng mô hình đã huấn luyện để phân tích và dự đoán cảm xúc của văn bản.
7. **Xác định nhãn dự đoán:** Chọn nhãn cảm xúc có xác suất cao nhất từ kết quả của mô hình.
8. **Ánh xạ nhãn dự đoán sang trạng thái cảm xúc:** Chuyển nhãn dự đoán thành trạng thái cảm xúc dễ hiểu.
9. **Chuẩn bị kết quả trả về:** Tạo cấu trúc dữ liệu chứa trạng thái cảm xúc, điểm số và xác suất cho từng trạng thái cảm xúc.

Kết quả:

– Cấu trúc JSON:

```
{
  "sentiment": "Tích cực",
  "score": 0.85,
  "probabilities": {
    "Tiêu cực": 0.10,
    "Trung tính": 0.05,
    "Tích cực": 0.85
  }
}
```


Ví dụ:

– **Input JSON:**

```
{
  "text": "Tài liệu này rất hữu ích và thú vị."
}
```

– **Output JSON:**

```
{
  "sentiment": "Tích cực",
  "score": 0.90,
  "probabilities": {
    "Tiêu cực": 0.05,
    "Trung tính": 0.05,
    "Tích cực": 0.90
  }
}
```

2 Mô tả API phân tích theo các câu (sentences)

Endpoint: /lstm_paragraph, /phobert_paragraph, /lstm_cnn_paragraph, /bilstm_paragraph, /gru_paragraph

Phương thức HTTP: POST

Mô hình sử dụng:

- /lstm_paragraph: Mô hình LSTM.
- /phobert_paragraph: Mô hình PhoBERT.
- /lstm_cnn_paragraph: Mô hình LSTM-CNN.
- /bilstm_paragraph: Mô hình BiLSTM.
- /gru_paragraph: Mô hình GRU.

Yêu cầu:

- **Body:** Danh sách các đối tượng `TextRequest`, mỗi đối tượng chứa một chuỗi văn bản cần phân tích cảm xúc.

- **Cấu trúc JSON:**

```
[  
  { "text": "Câu văn đầu tiên trong tài liệu." },  
  { "text": "Câu văn thứ hai trong tài liệu." }  
]
```

Quá trình:

1. **Chuyển chuỗi đầu vào thành chữ thường:** Tất cả các ký tự trong từng câu được chuyển thành chữ thường.
2. **Chuẩn hóa văn bản:** Loại bỏ các ký tự không cần thiết từ từng câu.
3. **Phân đoạn từ:** Tách từng câu thành các từ hoặc cụm từ nhỏ hơn.
4. **Định dạng lại từ đã phân đoạn:** Thay thế các khoảng trắng trong từng câu bằng dấu gạch dưới nếu cần.
5. **Token hóa và đệm chuỗi:** Chuyển đổi từng câu thành các chỉ số số học và đảm bảo các câu có độ dài đồng nhất bằng cách thêm các giá trị đệm nếu cần.
6. **Dự đoán cảm xúc:** Sử dụng mô hình đã huấn luyện để phân tích và dự đoán cảm xúc của từng câu.
7. **Xác định nhãn dự đoán:** Chọn nhãn cảm xúc có xác suất cao nhất từ kết quả của mô hình cho từng câu.
8. **Ánh xạ nhãn dự đoán sang trạng thái cảm xúc:** Chuyển nhãn dự đoán thành trạng thái cảm xúc dễ hiểu cho từng câu.
9. **Chuẩn bị kết quả trả về:** Tạo danh sách kết quả cho từng câu, bao gồm trạng thái cảm xúc và điểm số.

Kết quả:

- **Cấu trúc JSON:**

```
[
  {
    "sentiment": "Tích cực",
    "score": 0.95
  },
  {
    "sentiment": "Tiêu cực",
    "score": 0.75
  }
]
```

Ví dụ:

– Input JSON:

```
[
  { "text": "Tài liệu này rất dễ hiểu và chi tiết." },
  { "text": "Tuy nhiên, một số phần có vẻ thiếu thông tin." }
]
```

– Output JSON:

```
[
  {
    "sentiment": "Tích cực",
    "score": 0.92
  },
  {
    "sentiment": "Trung tính",
    "score": 0.68
  }
]
```

Chương 5

Kết luận

1 So sánh kết quả các mô hình

1.1 Bảng so sánh kết quả các mô hình

Bảng 5.1: So sánh kết quả các mô hình LSTM, GRU, Bi-LSTM và LSTM-CNN

Mô hình	Accuracy	Precision	Recall	F1-Score
LSTM	0.82	0.82	0.82	0.82
GRU	0.82	0.82	0.82	0.82
Bi-LSTM	0.83	0.83	0.83	0.83
LSTM-CNN	0.90	0.89	0.90	0.90

1.2 Phân tích kết quả

Dựa trên Bảng 5.1, ta có thể rút ra các nhận xét sau:

- **Hiệu suất chung:** Cả bốn mô hình đều đạt độ chính xác (Accuracy) tương đối cao, tuy nhiên Bi-LSTM và LSTM-CNN cho thấy hiệu suất vượt trội so với LSTM và GRU với độ chính xác đạt 90% so với 82% của hai mô hình đầu tiên.
- **Precision, Recall và F1-Score:** Các mô hình Bi-LSTM và LSTM-CNN không chỉ đạt độ chính xác cao mà còn có các chỉ số Precision, Recall và F1-Score đều ở mức 0.89 - 0.90, cho thấy khả năng phân loại chính xác và khả năng nhận diện các lớp một cách đồng đều.
- **So sánh LSTM và GRU:** Cả hai mô hình LSTM và GRU đạt được kết quả tương tự nhau với Accuracy là 0.82. Điều này cho thấy trong trường hợp này, GRU không có lợi thế đáng kể so với LSTM về mặt hiệu suất.

- **Ưu điểm của Bi-LSTM và LSTM-CNN:** Bi-LSTM tận dụng được thông tin từ cả hai hướng của chuỗi dữ liệu, giúp cải thiện khả năng học các phụ thuộc dài hạn. Trong khi đó, LSTM-CNN kết hợp khả năng trích xuất đặc trưng cục bộ của CNN với khả năng học các phụ thuộc thời gian của LSTM, tạo nên một mô hình mạnh mẽ và linh hoạt hơn.
- **Thời gian tính toán:** Mặc dù Bi-LSTM và LSTM-CNN đạt hiệu suất cao hơn, nhưng chúng cũng có thể yêu cầu thời gian tính toán lâu hơn so với LSTM và GRU do độ phức tạp tăng lên.

1.3 Nhận xét tổng quan

Kết quả cho thấy mô hình Bi-LSTM và LSTM-CNN vượt trội hơn so với mô hình LSTM và GRU đơn thuần trong việc phân loại dữ liệu với độ chính xác cao hơn và các chỉ số đánh giá tốt hơn. Điều này chứng tỏ việc kết hợp các kỹ thuật tiên tiến như Bidirectional RNN và kết hợp CNN với LSTM có thể nâng cao đáng kể hiệu suất của mô hình trong các bài toán xử lý chuỗi dữ liệu.

Tuy nhiên, cần cân nhắc về khía cạnh tính toán và tài nguyên khi lựa chọn mô hình phù hợp. Trong các ứng dụng yêu cầu thời gian phản hồi nhanh, mô hình LSTM hoặc GRU có thể được ưu tiên sử dụng do tính đơn giản và tốc độ tính toán cao hơn, mặc dù hiệu suất có thể không cao bằng các mô hình phức tạp hơn như Bi-LSTM và LSTM-CNN.

Tài liệu tham khảo

- [1] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL]. URL: <https://arxiv.org/abs/1810.04805>.
- [2] *Docker Documentation*. <https://docs.docker.com/>.
- [3] *FastAPI Advanced User Guide*. <https://fastapi.tiangolo.com/tutorial/#advanced-user-guide>.
- [4] Subhasis Jana. *FastAPI-Streamlit-Docker-NLP*. <https://github.com/subhasisj/FastAPI-Streamlit-Docker-NLP>. 2024.
- [5] Annavarapu Chandra Sekhara Rao Chaitanya Kulkarni Mayur Wankhade. *A survey on sentiment analysis methods, applications, and challenges*. <https://link.springer.com/article/10.1007/s10462-022-10144-1>. 2022.
- [6] Dat Quoc Nguyen and Anh Tuan Nguyen. “PhoBERT: Pre-trained language models for Vietnamese”. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Ed. by Trevor Cohn, Yulan He, and Yang Liu. Online: Association for Computational Linguistics, Nov. 2020, pp. 1037–1042. DOI: 10.18653/v1/2020.findings-emnlp.92. URL: <https://aclanthology.org/2020.findings-emnlp.92>.
- [7] *Streamlit Documentation*. <https://docs.streamlit.io/>.
- [8] Quan-Hoang Vo et al. *Multi-channel LSTM-CNN model for Vietnamese sentiment analysis*. https://www.researchgate.net/publication/321259272_Multi-channel_LSTM-CNN_model_for_Vietnamese_sentiment_analysis. 2017.