

**Mục Lục**

[LỜI NÓI ĐẦU 4](#_Toc133069309)

[CHƯƠNG 1. GIỚI THIỆU 5](#_Toc133069310)

[1. Xác định đề tài 5](#_Toc133069311)

[2. Mục đích nghiên cứu 5](#_Toc133069312)

[3. Giới hạn phạm vi đề tài 5](#_Toc133069313)

[4. Phương pháp nghiên cứu 5](#_Toc133069314)

[5. Công cụ sử dụng 5](#_Toc133069315)

[CHƯƠNG 2. CƠ SỞ LÝ THUYẾT VÀ CÁC VẤN ĐỀ LIÊN QUAN 6](#_Toc133069316)

[1. Các thuật toán được dùng 6](#_Toc133069317)

[1.1. Logistics Regression 6](#_Toc133069318)

[1.2. Naive Bayes 6](#_Toc133069319)

[1.3. Long short term memory 6](#_Toc133069320)

[2. Các phương pháp biểu diễn văn bản thành dạng vector 7](#_Toc133069321)

[2.1. CountVectorizer 7](#_Toc133069322)

[2.2. TF-IDF Vectorizer 7](#_Toc133069323)

[CHƯƠNG 3. ĐỀ XUẤT MÔ HÌNH TRIỂN KHAI 7](#_Toc133069324)

[1. Các bước thực hiện 7](#_Toc133069325)

[1.1. Đánh giá tổng quan bộ dữ liệu đầu vào. 7](#_Toc133069326)

[1.2. Tiền xử lý. 7](#_Toc133069327)

[1.3. Khám phá dữ liệu (Data Exploration) 7](#_Toc133069328)

[1.4. Ứng dụng mô hình phân lớp 7](#_Toc133069329)

[a. Xây dựng và đánh giá mô hình bằng các thuật toán máy học Machine Learning. 7](#_Toc133069330)

[b. Xây dựng và đánh giá mô hình bằng các thuật toán học sâu Deep Learning 8](#_Toc133069331)

[2. Phương hướng triển khai thực nghiệm 8](#_Toc133069332)

[3. Kết quả mong muốn dự báo đạt được 8](#_Toc133069333)

[CHƯƠNG 4. TRIỂN KHAI VÀ ĐÁNH GIÁ 9](#_Toc133069334)

[1. Thực hiện tiền xử lý: 9](#_Toc133069335)

[1.1. Xác định một số chức năng để loại bỏ dữ liệu không mong muốn 9](#_Toc133069336)

[1.2. TOKENIZER (Tách từ) 12](#_Toc133069337)

[1.3. Tiền xử lý văn bản sử dụng các functions trên 13](#_Toc133069338)

[1.4. Kiểm tra duplicate tweets sau quá trình làm sạch 14](#_Toc133069339)

[1.5. Kiểm tra độ dài của các texts tweet: 14](#_Toc133069340)

[1.6. Áp dụng Stemming và Lemmatization vào bộ dữ liệu đã tiền xử lý 15](#_Toc133069341)

[2. Xây dựng và đánh giá mô hình phân tích dữ liệu Tweet 16](#_Toc133069342)

[2.1. Xây dựng và đánh giá mô hình bằng các thuật toán máy học 16](#_Toc133069343)

[2.1.1 Chia tập dữ liệu 16](#_Toc133069344)

[2.1.2. Xây dựng mô hình bằng các thuật toán máy học 17](#_Toc133069345)

[2.1.3. Đánh giá 18](#_Toc133069346)

[2.2. Xây dựng và đánh giá mô hình bằng các thuật toán học sâu 20](#_Toc133069347)

[2.2.1. Xây dựng mô hình học sâu: BiLSTM 20](#_Toc133069348)

[2.2.2. Đánh giá 30](#_Toc133069349)

[3. Áp dụng mô hình 32](#_Toc133069350)

[CHƯƠNG 5. KẾT LUẬN 33](#_Toc133069351)

[TÀI LIỆU THAM KHẢO 34](#_Toc133069352)

# LỜI NÓI ĐẦU

Twitter là một trang mạng xã hội cho phép người sử dụng kết nối mọi người với nhau thông qua việc viết và đọc nội dung có độ dài giới hạn 140 ký tự cùng hình ảnh  sử dụng đăng tải lên. Nội dung thông tin người dùng đăng lên có thể thuộc bất kỳ chủ đề nào, từ những xu hướng xã hội “nóng" nhất hiện nay cho đến những chủ đề liên quan đến cuộc sống cá nhân người dùng như tâm trạng, hoạt động đời sống hằng ngày,.. Vì đặc tính đăng tải một cách “tự phát” cá nhân, chỉ bị ràng buộc và cấm chia sẻ khi có liên quan đến các nội dung truyền tải theo hướng bạo lực nặng, xu hướng tôn giáo lệch lạc, xu hướng tính dục trái đạo đức. Tuy nhiên, đôi khi chúng ta lại bắt gặp những đoạn tweet được đăng tải mang xu hướng bạo lực mạng nhưng lại không bị “report”, nghĩa là không bị “báo xấu”, vẫn xuất hiện và tồn tại trên trang feed mạng xã hội Twitter. Mà động thái này, nổi bật nhất là sau thời điểm đợt dịch Covid, theo báo cáo thu thập được từ xu hướng tweet đăng tải, thì số lượng tweet có nội dung liên quan đến bạo lực đã gây ra tinh thần bạo lực ảnh hưởng đến nhóm người sử dụng về tâm lý cá nhân, xu hướng giảm sút ý nghĩ và tinh thần khi đọc những dòng trạng thái được đăng tải.

Vì lẽ đó, câu hỏi đặt ra là “Làm thế nào để nhận biết trạng thái bạo lực trong tweet nhằm ngăn chặn hoặc giảm thiểu hiển thị đến người sử dụng mạng xã hội Twitter ?”. Để trả lời cho  điều này, nhóm đã sử dụng bộ dữ liệu Cyberbullying Tweet Recognition Project thu thập data trong khoảng thời gian diễn ra đại dịch Covid-19 liên quan đến các tweet có nội dung bắt nạt trên mạng xã hội với mục đích tạo ra các mô hình học máy phân lớp để gán nhãn phân loại các Tweet có liên quan nội dung bạo lực thuộc phân loại nào, từ đó gắn cờ các Tweet đó để cảnh báo người sử dụng. Đây cũng chính là lý do nhóm chúng em quan tâm đến đề tài **Phân loại dạng bạo lực mạng từ các tweet bằng các mô hình học máy và học sâu** làm đồ án cuối kỳ học phần Máy học khoa Công nghệ thông tin Kinh doanh trường Đại học Kinh tế (UEH).

Do điều kiện về mặt thời gian và nhận thức còn nhiều hạn chế nên đồ án không khỏi có phần sai sót, kính mong quý giảng viên nhiệt tình góp ý để nhóm chúng em có những tiếp thu tốt hơn trong môn học này. Qua bài đồ án, nhóm chúng em xin cảm ơn giảng viên phụ trách môn Máy học, thầy Đặng Ngọc Hoàng Thành đã truyền đạt những kiến thức quý báu cho học viên bằng cả tấm lòng nhiệt tình và sự tận tâm của thầy.

# CHƯƠNG 1. GIỚI THIỆU

## 1. Xác định đề tài

Đề tài Phân loại dạng bạo lực mạng từ các tweet bằng các mô hình học máy và học sâu nhóm lựa chọn dựa theo bộ dữ liệu Cyberbullying Tweet Recognition Project thu thập data trong khoảng thời gian diễn ra đại dịch Covid-19 liên quan đến các tweet có nội dung bắt nạt trên mạng xã hội.

Sau khi thu được bộ dữ liệu và phân tích các thuộc tính liên quan cũng như quan sát các nội dung tweet đăng tải thì nhóm thấy rằng trong một danh sách các tweet đó, sẽ có các tweet được có nội dung liên quan hoặc không liên quan đến bạo lực mạng. Vì vậy, nhóm muốn biết nội dung có liên quan hay không thì nhóm sẽ thực hiện phân tích các từ ngữ trong cùng 1 đoạn để biết các chữ đó có liên quan đến hành vi bạo lực, và nếu có liên quan thì các chữ đó thuộc phân loại bạo lực nào. Điều này đã thực hiện đi đúng với định hướng chọn đề tài liên quan đến Machine Learning là ứng dụng Học máy và Học sâu, thuộc một trong những nhiệm vụ cơ bản trong xử lý ngôn ngữ tự nhiên là phân loại văn bản.

## 2. Mục đích nghiên cứu

Bộ dữ liệu cho thấy data liên quan đến bạo lực Tweet từ đó giúp tạo ra các mô hình học máy phân lớp để gán nhãn phân loại các Tweet có liên quan nội dung bạo lực thuộc phân loại nào, từ đó gắn cờ các Tweet đó để cảnh báo người sử dụng.

## 3. Giới hạn phạm vi đề tài

Trong bài báo cáo này, nhóm sử dụng dữ liệu từ bộ dữ liệu thu thập Cyberbullying Tweet Recognition Project thu thập data trong khoảng thời gian diễn ra đại dịch Covid-19. Bộ dữ liệu chứa hơn 47.000 Tweet phân vào 2 cột chính gồm:

* Nội dung: 46017 Unique Values
* Phân loại bạo lực: 6 phân loại

Với Phân loại bạo lực liên quan các nội dung khác nhau được gán các nhãn gồm:

* Tuổi: Age
* Dân tộc:  Ethnicity
* Giới tính: Gender
* Tôn Giáo: Religion
* Dạng bắt nạt khác: Other type of cyberbullying
* Không phải bắt nạt: Not Cyberbullying

## 4. Phương pháp nghiên cứu

Phương pháp xây dựng thuật toán máy học xây dựng mô hình CountVectorizer, TfidfVectorizer và xây dựng mô hình BiLSTM ứng dụng Học sâu vào việc nhận diện các tweet có chứa nội dung gây hại hoặc bắt nạt trên mạng xã hội.

## 5. Công cụ sử dụng

Ngôn ngữ lập trình phân tích dữ liệu Python.

# CHƯƠNG 2. CƠ SỞ LÝ THUYẾT VÀ CÁC VẤN ĐỀ LIÊN QUAN

## Các thuật toán được dùng

### Logistics Regression

Logistic Regression là 1 thuật toán phân loại được dùng để gán các đối tượng cho 1 tập hợp giá trị rời rạc (như 0, 1, 2, ...). Một ví dụ điển hình là phân loại Email, gồm có email công việc, email gia đình, email spam, … Hay cụ thể hơn là ở bài toán về phân loại đoạn Tweet này sẽ phân loại ra cá Tweet có nội dung liên quan bạo lực Cyberbullying như Religion, Gender, Age,..

### Naive Bayes

Naive Bayes là một thuật toán phân lớp được mô hình hoá dựa trên định lý Bayes trong xác suất thống kê. Với sự lựa chọn sử dụng mô hình Multinomial Naive Bayes là mô hình được sử dụng trong phân loại văn bản. Đặc trưng đầu vào ở đây chính là tần suất xuất hiện của từ trong văn bản đó. Tương tự như ý tưởng của nhóm là thực hiện tìm kiếm từ xuất hiện nhiều trong một đoạn Tweet để phân loại ra đoạn Tweet đó sẽ có nội dung liên quan đến phân loại bạo lực nào.

### Long short term memory

Long short-term memory (LSTM) là một kiến ​​trúc  artificial recurrent neural network (RNN) được sử dụng trong lĩnh vực Deep learning. LSTM được sử dụng để xử lý các chuỗi dữ liệu như văn bản, âm thanh, hình ảnh và video, và có thể được áp dụng trong các lĩnh vực như xử lý ngôn ngữ tự nhiên, nhận dạng giọng nói, dịch máy, v.v. Các thành phần chính của LSTM bao gồm cổng quên (forget gate), cổng đầu vào (input gate), cổng đầu ra (output gate) và tế bào trạng thái (cell state). Cổng quên (forget gate) quyết định thông tin nào sẽ được lưu trữ và thông tin nào sẽ bị xóa; cổng đầu vào (input gate) quyết định thông tin mới nào sẽ được lưu trữ vào tế bào trạng thái; cổng đầu ra (output gate) quyết định thông tin nào sẽ được truyền đi và tế bào trạng thái (cell state) là nơi lưu trữ thông tin lâu dài.

BiLSTM (Bidirectional Long Shot-Term Memory) là một kiến trúc mạng nơ-ron sử dụng trong xử lý ngôn ngữ tự nhiên, nó kết hợp cả một LSTM tiến và một LSTM lùi. Với cơ chế cập nhật trạng thái ẩn của nó, BiLSTM có thể hiểu được văn bản một cách "dài hạn" và "ngắn hạn", giúp cho nó có khả năng xử lý tốt trong các tác vụ như nhận dạng tên thực thể, phân loại cảm xúc và dịch máy. BiLSTM hoạt động bằng cách sử dụng hai LSTM song song, một tiến và một lùi. LSTM tiền xử lý dữ liệu từ trái sang phải và LSTM lùi xử lý từ phải sang trái. Trong quá trình xử lý, BiLSTM sẽ cập nhật trạng thái ẩn của một node dựa trên trạng thái ẩn của cả LSTM tiến và LSTM lùi. Như vậy, với mỗi từ trong văn bản, BiLSTM sẽ có hai trạng thái ẩn, một từ LSTM tiến và một từ LSTM lùi. Vì BiLSTM có khả năng hiểu được mối quan hệ giữa các từ trong văn bản từ cả hai phía, nó có khả năng xử lý tốt trong các tác vụ như phân tích cảm xúc, dịch máy và nhận dạng thực thể. Tuy nhiên, BiLSTM có số lượng tham số lớn, điều này làm tăng độ phức tạp tính toán và tăng thời gian huấn luyện.

## Các phương pháp biểu diễn văn bản thành dạng vector

### CountVectorizer

CountVectorizer là phương pháp biểu diễn văn bản thành vectơ đặc trưng bằng cách đếm số lần xuất hiện của từ trong tài liệu. Phương pháp này dựa trên mô hình bag-of-words, loại bỏ thông tin về thứ tự từ. Đầu tiên, xây dựng từ điển và mã hóa từ thành số nguyên duy nhất. Sau đó, đếm số lần xuất hiện của từ trong tài liệu và lưu trữ vào vectơ đặc trưng. CountVectorizer đơn giản và xử lý được lượng lớn tài liệu, nhưng không quan tâm đến thứ tự các từ và không xử lý được các từ đồng nghĩa hay viết tắt.

### TF-IDF Vectorizer

TF-IDF là phương pháp biểu diễn văn bản thành các vectơ đặc trưng, tính toán giá trị đại diện cho mỗi từ trong tài liệu dựa trên tần suất xuất hiện của từ đó trong tài liệu và tần suất xuất hiện của từ đó trong toàn bộ tập hợp các tài liệu. Nó giảm thiểu giá trị của các từ phổ biến và tăng giá trị của các từ quan trọng trong tài liệu. Tuy nhiên, TF-IDF không xử lý được các từ không xuất hiện trong từ điển và không giữ được thông tin về thứ tự các từ trong tài liệu.

# CHƯƠNG 3. ĐỀ XUẤT MÔ HÌNH TRIỂN KHAI

## Các bước thực hiện

### Đánh giá tổng quan bộ dữ liệu đầu vào.

* In ra các đoạn văn bản mẫu từ bộ dữ liệu dựa trên các nhãn đã phân loại.
* Đổi tên các cột sao cho dễ hiểu và phù hợp với đề tài nghiên cứu.
* Tạo biểu đồ thanh ngang để biểu diễn tần suất xuất hiện của 10 từ phổ biến nhất theo từng nhãn.
* Tạo ra danh sách 10 cặp từ thường đi cùng nhau và xuất hiện nhiều nhất theo từng nhãn và dùng biểu đồ thanh ngang để biểu diễn.
* Đếm số lần xuất hiện của các nhãn và biểu diễn bằng Count Plot theo cột dọc.

### Tiền xử lý.

* Kiểm tra Missing Values.
* Tạo ra các hàm chức năng để làm sạch và xử lý dữ liệu văn bản.
* Kiểm tra và loại bỏ các duplicate tweets sau khi làm sạch.
* Xoá các tweets có độ dài quá 100 từ và các tweets có độ dài bằng 0.

### Khám phá dữ liệu (Data Exploration)

Hiểu trước các đặc điểm của dữ liệu sẽ cho phép ta xây dựng **mô hình phân lớp** tốt hơn (nghĩa là đạt được độ chính xác cao hơn). Sau khi làm sạch dữ liệu ta vẽ WordCloud cho các nhãn, các đám mây từ này rất hữu ích để trực quan hóa các từ phổ biến trong dữ liệu. Chúng tương tự như bar plot nhưng sinh động hơn về mặt hình ảnh.

### **Ứng dụng mô hình phân lớp**

#### Xây dựng và đánh giá mô hình bằng các thuật toán máy học Machine Learning.

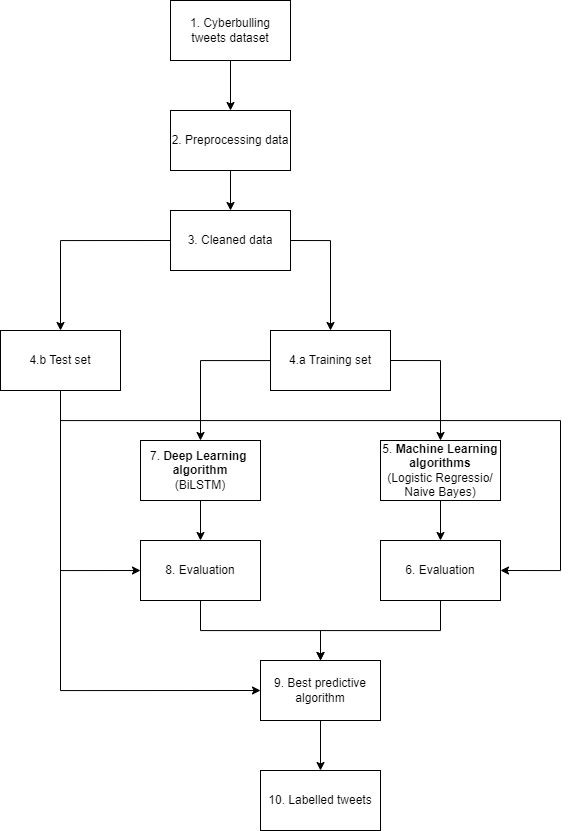
Ta thực hiện việc chia tập dữ liệu sử dụng thư viện Scikit-learn (sklearn) để chia tập dữ liệu thành tập huấn luyện và tập kiểm tra. Sau đó, ta dùng đến 2 phương pháp được sử dụng trong xử lý ngôn ngữ tự nhiên để vector hóa văn bản, đó là CountVectorizer và TF-IDF. Sau cùng, để xây dựng một hệ thống phân loại, chúng ta cần một cách để đánh giá hiệu suất của bộ phân loại. Tạo ra một bảng tóm tắt các điểm số phân loại cho các mô hình đã sử dụng sau đó trực quan hoá điểm số Accuracy trên tập huấn luyện và kiểm tra của từng mô hình bằng biểu đồ thang ngang. Từ đó, chọn ra mô hình tối ưu nhất của giữa 2 mô hình Machine Learning là Logistic Regression và Naive Bayes

#### Xây dựng và đánh giá mô hình bằng các thuật toán học sâu Deep Learning

Xây dựng mô hình BiLSTM sử dụng các thư viện hỗ trợ như Keras hoặc PyTorch để xây dựng mô hình BiLSTM với các lớp nhúng (embedding layer), LSTM layer và các lớp fully connected để phân loại tweet thành các nhãn tương ứng. Tiếp đến, thực hiện huấn luyện mô hình bằng việc sử dụng tập dữ liệu đã được xử lý, tùy chỉnh các tham số mô hình như số lớp, số nơ-ron, hệ số học và số vòng lặp để tối ưu hóa độ chính xác của mô hình. Cuối cùng, ta thực hiện kiểm tra và đánh giá mô hình sử dụng tập dữ liệu kiểm tra để đánh giá độ chính xác của mô hình.

## Phương hướng triển khai thực nghiệm

Nhóm thực hiện triển khai phương hướng thực hiện theo lưu đồ như sau:



## Kết quả mong muốn dự báo đạt được

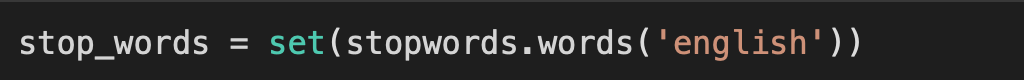
Mô hình Machine Learning sẽ so sánh về mức độ chính xác giữa dữ liệu thực tế phân loại bạo lực đã được gán nhãn cho thấy và dữ liệu dự báo khi được áp dụng mô hình máy học của nhóm khi phân loại bạo lực. Trong khi mô hình Deep Learning sẽ thu được các từ mang ý nghĩa phân loại bạo lực thường xuyên dùng nhất từ đó ngăn chặn việc sử dụng từ ngữ bạo lực khi Tweet. Deep Learning sẽ lược bỏ một số thao tác thủ công và thu được kết quả tốt hơn trên bộ dữ liệu lớn hơn. Như vậy, nhóm sẽ so sánh mức độ chính xác giữa 2 mô hình ứng dụng là Machine Learning và Deep Learning, nếu như thuật toán nào có độ chính xác cao hơn thì nhóm sẽ lựa chọn predict dựa trên thuật toán.

## CHƯƠNG 4. TRIỂN KHAI VÀ ĐÁNH GIÁ

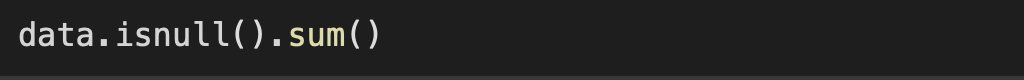
## Thực hiện tiền xử lý:

C:\Users\HOME\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\61A3AC48.tmpThay thế các giá trị trong cột "sentiment" được định nghĩa bởi một từ hoặc một cụm từ mô tả tình trạng cảm xúc của một bình luận hoặc tin nhắn, chẳng hạn như "religion", "age", "ethnicity", "gender", "other\_cyberbullying" hoặc "not\_cyberbullying"của bộ dữ liệu bằng các giá trị số tương ứng được mã hóa. Kết quả của quá trình này được lưu trữ trong cột mới được gọi là "sentiment\_encoded".

Tạo 1 set chứa các từ stop word trong tiếng anh: Các từ dừng là những từ phổ biến trong ngôn ngữ, nhưng không mang tính quan trọng để phân tích ngôn ngữ hoặc phân loại văn bản. Ví dụ về các từ dừng là "a", "an", "the", "and", "but", "in", "on", "at", "is", "of", "to",...Việc lưu trữ các từ dừng trong một set sẽ giúp cho việc tìm kiếm các từ dừng trong một văn bản trở nên nhanh chóng và hiệu quả hơn so với việc lưu trữ chúng trong một danh sách.



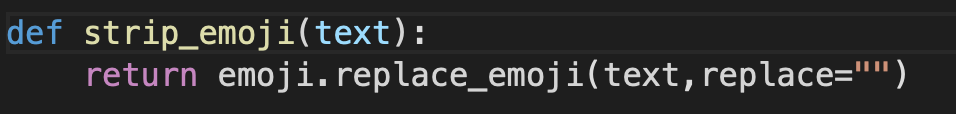
Kiểm tra các giá trị bị thiếu, kết quả trả về cho thấy bộ dữ liệu không tồn tại giá trị bị thiếu nào.



### Xác định một số chức năng để loại bỏ dữ liệu không mong muốn

Tạo hàm chức năng để xóa các Emoji (Emoji là các dạng ký tự dạng hình ảnh xuất hiện trong các tin nhắn điện tử và trang web). Ở đây ta không phân tích các ký tự cảm xúc này vì nó có thể gây nhiễu khi người dùng sử dụng chúng không đúng với cảm xúc thật sự của họ.

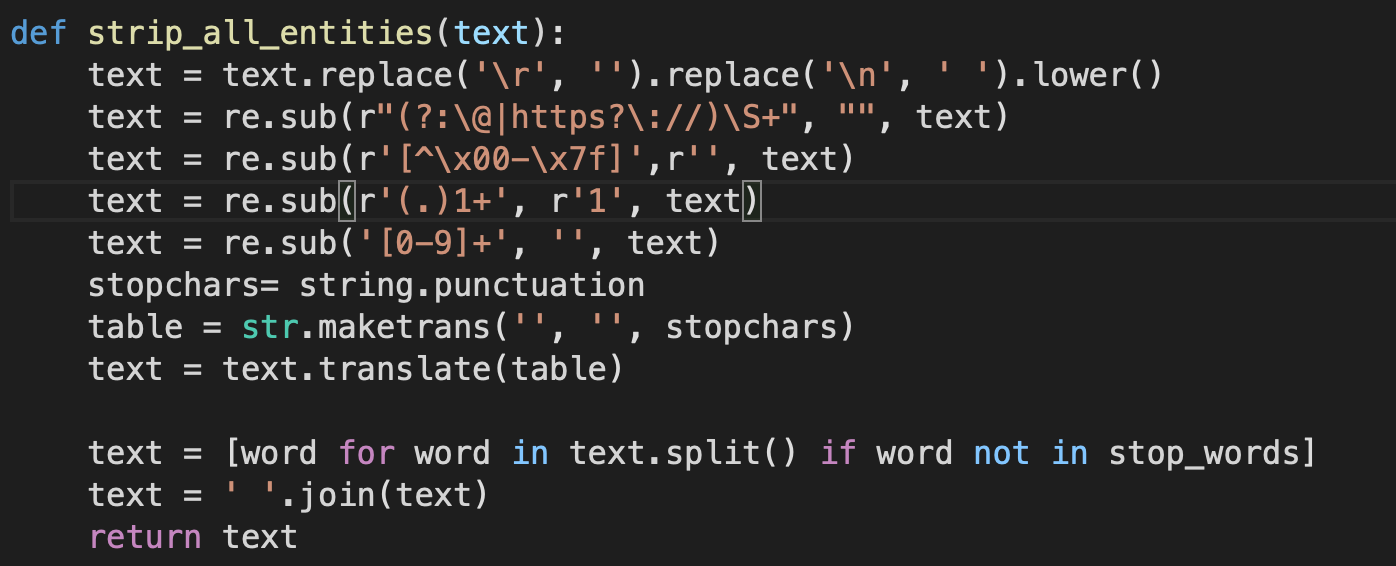
Cụ thể: Ta sử dụng hàm replace\_emoji(text, replace="") của thư viện emoji, trong đó text là chuỗi cần loại bỏ emoji và replace là chuỗi thay thế cho mỗi emoji. Trong trường hợp này, chuỗi thay thế được chỉ định là chuỗi rỗng, nghĩa là loại bỏ các emoji đó khỏi chuỗi text. Sau đó, hàm trả về kết quả là chuỗi text đã được xóa các emoji.



Tạo hàm có tên là strip\_all\_entities(text) để xóa các đối tượng (entities) trong một chuỗi văn bản text. Các đối tượng này bao gồm các ký tự đặc biệt, liên kết URL, tên người dùng Twitter, số và các stopwords trong tiếng Anh. Cụ thể:

1. Xóa các ký tự xuống dòng và các ký tự không phải ASCII bằng cách thay thế chúng bằng khoảng trắng và đưa tất cả các chữ cái về dạng chữ thường.
2. Loại bỏ các liên kết URL và tên người dùng Twitter bằng cách sử dụng biểu thức chính quy (regular expression).
3. Loại bỏ các ký tự không phải ASCII khác, bằng cách sử dụng biểu thức chính quy để tìm và thay thế chúng bằng chuỗi rỗng.
4. Loại bỏ các chữ số xuất hiện trong văn bản bằng cách sử dụng biểu thức chính quy để tìm và thay thế chúng bằng chuỗi rỗng.
5. Xóa các ký tự đặc biệt bằng cách sử dụng bảng chuyển đổi (translation table) của Python để loại bỏ chúng khỏi chuỗi text.
6. Loại bỏ các từ dừng (stopwords) trong tiếng Anh bằng cách tách chuỗi thành các từ riêng lẻ, lọc bỏ các từ dừng và nối lại thành một chuỗi mới.

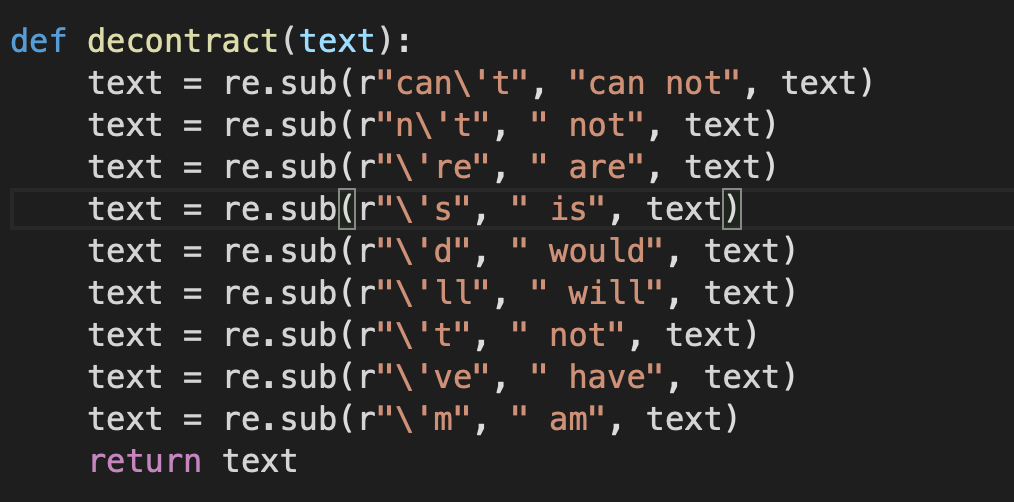
Cuối cùng, hàm trả về chuỗi text đã được xử lý, không chứa các đối tượng không cần thiết.



Tạo hàm chức năng xử lý các từ rút gọn, viết tắt. Các quy tắc được áp dụng trong hàm "decontract" bao gồm:

1. can't -> can not
2. n't -> not
3. 're -> are
4. 's -> is
5. 'd -> would
6. 'll -> will
7. 've -> have
8. 'm -> am

Sau khi thực hiện các thay thế, hàm trả về văn bản đã được chuyển đổi với các từ viết tắt được thay thế bằng các từ đầy đủ tương ứng.

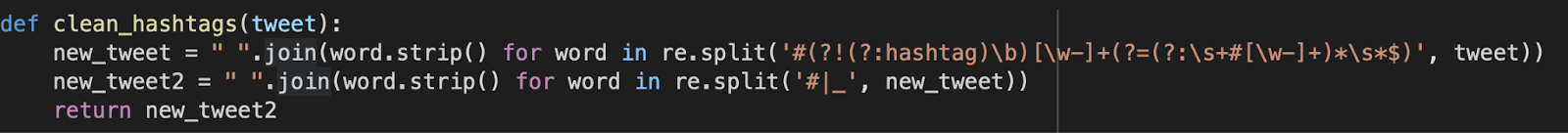


Tạo hàm chức năng để xử lý các Hashtag. Cụ thể, biểu thức **\#(?!(?:hashtag)\b)[\w-]+(?=(?:\s+#[\w-]+)\*\s\*$)** được sử dụng để tách các hashtag ra khỏi chuỗi. Biểu thức này có nghĩa là:

**#:** phải bắt đầu bằng ký tự #

**(?!(?:hashtag)\b)**: không được theo sau bởi từ "hashtag" và không được phép có các ký tự khác ngoài ký tự word hoặc dấu gạch ngang sau từ "hashtag".

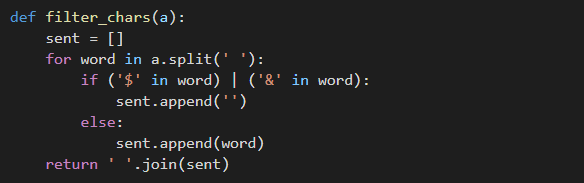
**[\w-]+:** có thể là một chuỗi các ký tự hoặc dấu gạch ngang.

**(?=(?:\s+#[\w-]+)\*\s\*$)**: phải kết thúc với 0 hoặc nhiều hashtag được phân tách bởi khoảng trắng và phải kết thúc bằng khoảng trắng.

Tạo hàm có tên filter\_chars(), nhận đầu vào là một chuỗi bất kỳ. Hàm này có chức năng lọc các từ trong chuỗi  mà chứa các ký tự đặc biệt là $ hoặc &. Cụ thể:

1. Khởi tạo một danh sách rỗng và đặt tên là “sent”.
2. Với mỗi từ trong chuỗi, tách từ bằng khoảng trắng (' ') bằng cách sử dụng phương thức split(). Sử dụng vòng lặp For() để lặp qua tất cả các từ này.
3. Nếu từ hiện tại chứa ít nhất một trong hai ký tự đặc biệt “$” hoặc “&”, thì thêm một chuỗi rỗng vào danh sách sent.
4. Ngược lại, nếu từ hiện tại không chứa các ký tự đặc biệt trên, thì thêm từ đó vào danh sách sent.

Cuối cùng, trả về chuỗi kết quả là danh sách “sent” đã được nối với nhau bằng khoảng trắng bằng cách sử dụng phương thức join().



Tạo hàm chức năng “remove\_mult\_spaces” có biểu thức chính quy **"\s\s+"** được sử dụng để tìm tất cả các chuỗi có chứa hai hoặc nhiều ký tự khoảng trắng liên tiếp. Sau đó, hàm re.sub được sử dụng để thay thế tất cả các chuỗi này bằng một ký tự khoảng trắng duy nhất, được biểu diễn bởi chuỗi " ". Kết quả là hàm trả về một chuỗi mới, mà trong đó các khoảng trắng dư thừa đã được loại bỏ.

C:\Users\HOME\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\6FE65BF8.tmp

### TOKENIZER (Tách từ)

Stemming và lemmatization là hai phương pháp tiền xử lý văn bản trong xử lý ngôn ngữ tự nhiên.

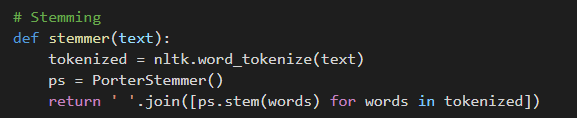
Stemming là quá trình loại bỏ các hậu tố và tiền tố của một từ để giảm thiểu độ phức tạp của từ và tạo ra các từ gốc giống nhau. Ví dụ: từ "running", "runs" và "run" có thể được chuyển thành "run" bằng cách sử dụng các thuật toán stemming khác nhau. Tuy nhiên, quá trình stemming có thể dẫn đến một số từ không tồn tại trong từ điển và không phù hợp với ngữ cảnh.

Lemmatization là quá trình chuyển đổi một từ về dạng cơ bản nhất của nó, được gọi là từ gốc hoặc từ đồng nghĩa. Quá trình này sử dụng kiến thức về ngữ pháp và ngữ nghĩa của một ngôn ngữ để tìm từ gốc chính xác. Ví dụ: từ "better" có thể được chuyển thành "good" bằng cách sử dụng lemmatization. Lemmatization cho kết quả chính xác hơn so với stemming, nhưng lại tốn nhiều thời gian và tài nguyên tính toán hơn.

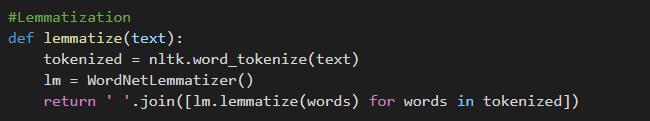
Tóm lại, cả stemming và lemmatization đều có vai trò quan trọng trong xử lý ngôn ngữ tự nhiên. Tuy nhiên, từng phương pháp lại có ưu và nhược điểm riêng, và phải được áp dụng phù hợp với mục đích và bối cảnh của bài toán. Ở đây, chúng ta sẽ sử dụng cả hai phương pháp.

Stemmer: sử dụng thư viện Natural Language Toolkit (nltk) để phân tích cú pháp và tách từ (tokenize) văn bản đầu vào thành các từ riêng lẻ, và lưu trữ kết quả vào biến "tokenized". Tiếp theo, sử dụng PorterStemmer của thư viện nltk để tạo ra một đối tượng stemmer, giúp loại bỏ các hậu tố và tiền tố của các từ và chuyển đổi chúng thành dạng gốc.

Cuối cùng, đoạn code sử dụng một list comprehension để thực hiện quá trình stemming trên từng từ trong danh sách các từ đã được tách ra từ văn bản đầu vào, và kết hợp các từ đã được stemming thành một chuỗi văn bản mới, được trả về bởi hàm stemmer.Tạo hàm chức năng để thực hiện tác vụ chuẩn hóa từ trên một chuỗi văn bản đầu vào. Để chuẩn hoá một từ, đối tượng PorterStemmer sẽ loại bỏ các hậu tố (suffix) của từ đó để giữ lại dạng gốc của từ.

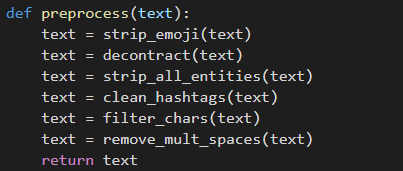


Lemmatization: sử dụng thư viện Natural Language Toolkit (nltk) để phân tích cú pháp và tách từ (tokenize) văn bản đầu vào thành các từ riêng lẻ, và lưu trữ kết quả vào biến "tokenized". Tiếp theo, sử dụng WordNetLemmatizer của thư viện nltk để tạo ra một đối tượng lemmatizer, giúp chuyển đổi các từ trong văn bản thành dạng cơ bản nhất của chúng. Cuối cùng, sử dụng một list comprehension để thực hiện quá trình lemmatization trên từng từ trong danh sách các từ đã được tách ra từ văn bản đầu vào, và kết hợp các từ đã được lemmatization thành một chuỗi văn bản mới, được trả về bởi hàm lemmatize.

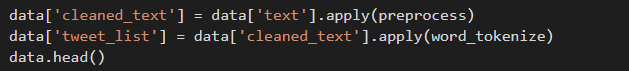


### Tiền xử lý văn bản sử dụng các functions trên

Tạo hàm Preprocess để tổng hợp lại các hàm chức năng ở trên và áp dụng cho dữ liệu input.



Lưu các dữ liệu đã được apply hàm Preprocess vào 1 cột mới trong dataframe, sau đó tiếp tục sử dụng phương thức apply trên cột 'cleaned\_text' của DataFrame data, áp dụng phương thức word\_tokenize của thư viện nltk để tách các từ trong chuỗi văn bản và lưu trữ danh sách các từ này trong cột 'tweet\_list' của dataframe data.



### Kiểm tra duplicate tweets sau quá trình làm sạch

Kiểm tra các duplicate tweets sau khi đã làm sạch dữ liệu.

C:\Users\HOME\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\80B7EDE.tmp

Vì có tồn tại duplicates nên ta tiến hành loại bỏ:

C:\Users\HOME\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\23D7EE9C.tmp

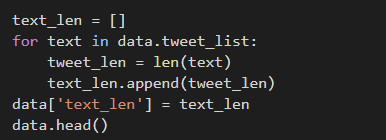
Kiểm tra các class balance sau khi đã loại bỏ duplicates:

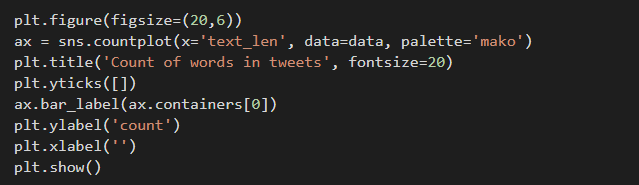
C:\Users\HOME\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\7985FB0A.tmp

C:\Users\HOME\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\2FF3B7A8.tmp

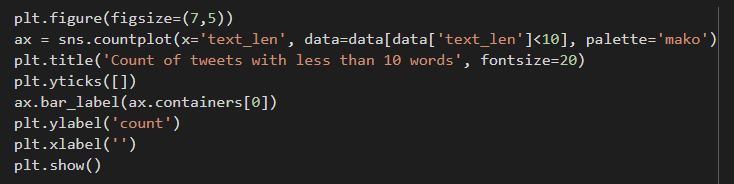
### Kiểm tra độ dài của các texts tweet:

Kiểm tra độ dài của các texts tweet và biểu diễn trực quan số lượng từ trong mỗi tweet của một tập dữ liệu data bằng biểu đồ đếm (countplot).





Trực quan hóa số lượng tweet có độ dài tối đa bằng hoặc nhỏ hơn 10 từ của tập dữ liệu data bằng biểu đồ đếm (countplot)



* + Xóa các dòng tweet có độ dài bằng 0:

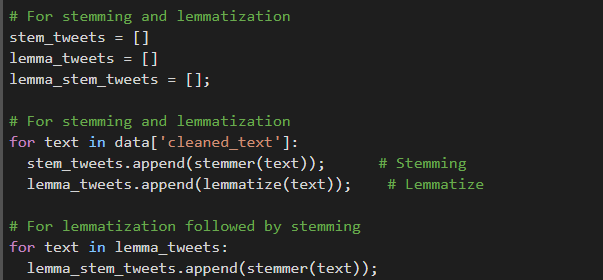
C:\Users\HOME\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\AFC75600.tmp

### Áp dụng Stemming và Lemmatization vào bộ dữ liệu đã tiền xử lý

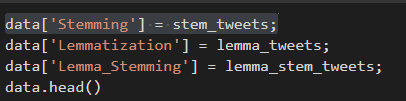
Khởi tạo ba biến rỗng là “stem\_tweets”, “lemma\_tweets” và “lemma\_stem\_tweets”.

Trong vòng lặp đầu tiên, lấy từng văn bản được xử lý sẵn từ cột 'cleaned\_text' trong dataframe 'data', thực hiện quá trình stemming và lemmatization trên văn bản đó bằng cách gọi hàm stemmer và lemmatize, và lưu trữ kết quả của hai quá trình đó vào hai biến “stem\_tweets” và “lemma\_tweets” tương ứng.

Sau đó, trong vòng lặp thứ hai, lấy từng văn bản trong lemma\_tweets (kết quả đã được lemmatize trước đó), và thực hiện quá trình stemming bằng cách gọi hàm stemmer, và lưu trữ kết quả của quá trình đó vào biến lemma\_stem\_tweets.



Lần lượt gán các giá trị của các list “stem\_tweets”, “lemma\_tweets” và “lemma\_stem\_tweets” vào đúng cột giá trị tương ứng trong dataframe “data”



## Xây dựng và đánh giá mô hình phân tích dữ liệu Tweet

### Xây dựng và đánh giá mô hình bằng các thuật toán máy học

#### 2.1.1 Chia tập dữ liệu

Sử dụng thư viện Scikit-learn (sklearn) để chia tập dữ liệu thành tập huấn luyện và tập kiểm tra.

Cụ thể, đầu vào của hàm train\_test\_split là 2 Series x và y. Với x chứa văn bản sau khi đã được xử lý và làm sạch, được lưu trữ trong cột 'cleaned\_text' của dataframe 'data'. Và y là dữ liệu về nhãn của các văn bản, được lưu trữ trong cột 'sentiment' của dataframe 'data'.

Sử dụng hàm train\_test\_split() trong sklearn để tách dữ liệu thành tập huấn luyện và tập kiểm tra. Mặc định, hàm train\_test\_split sẽ chia dữ liệu đầu vào thành hai phần với tỷ lệ 75% cho tập huấn luyện và 25% cho tập kiểm tra. Ta sử dụng tham số random\_state thiết lập giá trị ngẫu nhiên được sử dụng để xáo trộn dữ liệu trước khi chia. Bằng cách đặt giá trị này thành một số cụ thể (trong trường hợp này là 42), chúng ta có thể đảm bảo rằng cùng một chia ngẫu nhiên được tạo ra mỗi lần chạy mã. Điều này hữu ích cho tính nhất quán trong quá trình tái tạo.

Kết quả trả về của hàm train\_test\_split là 4 mảng x\_train, x\_test, y\_train, y\_test chứa dữ liệu đã được chia thành tập huấn luyện và tập kiểm tra tương ứng.

y = data['sentiment']

X = data['cleaned\_text']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=42, stratify=y)

#### Xây dựng mô hình bằng các thuật toán máy học

Ta dùng đến 2 phương pháp được sử dụng trong xử lý ngôn ngữ tự nhiên để vector hóa văn bản, đó là CountVectorizer và TF-IDF

* **CountVectorizer:**

Sử dụng pipeline của sklearn để tạo ra một mô hình phân loại cảm xúc trên dữ liệu văn bản: sử dụng CountVectorizer để chuyển đổi văn bản thành ma trận đếm các từ, bỏ qua các từ trong danh sách stop\_words (từ không có ý nghĩa như "và", "với",...), và chỉ tính toán các từ xuất hiện ít nhất min\_df lần trong toàn bộ tập dữ liệu.

Sau đó sử dụng LogisticRegression với thuật toán lbfgs để huấn luyện một mô hình phân loại đa lớp trên ma trận đếm từ ở bước trước.

cv\_log = Pipeline([

    ('cv', CountVectorizer(stop\_words = 'english', min\_df=4)),

    ('lr', LogisticRegression(multi\_class='multinomial', solver='lbfgs', max\_iter=200))])

cv\_log.fit(X\_train, y\_train)

cv\_log.score(X\_train, y\_train), cv\_log.score(X\_test, y\_test)

Và sử dụng thuật toán MultinomialNB để xây dựng một mô hình phân loại Naive Bayes đa thức từ các vectơ đặc trưng.

cv\_mnb = Pipeline([

    ('cv', CountVectorizer(stop\_words = 'english', min\_df=4)),

    ('mnb', MultinomialNB())])

cv\_mnb.fit(X\_train, y\_train)

cv\_mnb.score(X\_train, y\_train), cv\_mnb.score(X\_test, y\_test)

Mỗi mô hình trên đều được sử dụng để dự đoán nhãn của tập dữ liệu X\_train và X\_test, và tính toán độ chính xác của mô hình trên cả tập dữ liệu huấn luyện và tập dữ liệu kiểm tra bằng phương thức score()

* **TF-IDF:**

Cũng như phương pháp CountVectorize, sử dụng pipeline cho TF IDF. Tuy nhiên TF IDF không chỉ tính đến số lần một từ xuất hiện trong tài liệu mà còn cả mức độ quan trọng của từ đó đến toàn bộ khối văn bản. Điều này được thực hiện bằng cách loại bỏ những từ thường xuất hiện trên tất cả các tài liệu, giảm số lượng những từ này vì những từ này có thể ít quan trọng hơn.

Ta đưa từng mô hình học máy vào Pipeline và tính toán độ chính xác.

tf\_log = Pipeline([

    ('tf', TfidfVectorizer(stop\_words = 'english', min\_df=4)),

    ('lr', LogisticRegression(multi\_class='multinomial', solver='lbfgs', max\_iter=200))])

tf\_log.fit(X\_train, y\_train)

tf\_log.score(X\_train, y\_train), tf\_log.score(X\_test, y\_test)

tf\_mnb = Pipeline([

    ('tf', TfidfVectorizer(stop\_words = 'english', min\_df=3)),

    ('mnb', MultinomialNB())])

tf\_mnb.fit(X\_train, y\_train)

tf\_mnb.score(X\_train, y\_train), tf\_mnb.score(X\_test, y\_test)

#### Đánh giá

Khi xây dựng một hệ thống phân loại, chúng ta cần một cách để đánh giá hiệu suất của bộ phân loại. Ta dùng các chỉ số được sử dụng phổ biến nhất và cách chúng giúp cung cấp một cái nhìn cân bằng về hiệu suất của bộ phân loại. Hàm "classifier\_metrics" được sử dụng để tính toán và in ra các chỉ số đánh giá hiệu suất của mô hình phân loại được đưa vào. Đầu vào của hàm gồm hai tham số: report\_title là tên của báo cáo đánh giá và model là mô hình phân loại cần đánh giá.

Đầu tiên, hàm sử dụng mô hình để dự đoán nhãn cho dữ liệu kiểm tra bằng phương thức "predict" và tính toán các chỉ số đánh giá hiệu suất của mô hình bằng cách sử dụng các hàm từ thư viện scikit-learn, bao gồm độ chính xác trên tập huấn luyện (train\_accuracy) và độ chính xác trên tập kiểm tra (test\_accuracy).

Sau đó, hàm lưu các giá trị đánh giá hiệu suất vào một dictionary với các keys tương ứng và trả về một dataframe chứa các giá trị đó. Cuối cùng, hàm trả về dataframe này để có thể được sử dụng cho việc tạo bảng tóm tắt hiệu suất của các mô hình khác nhau.

def classifier\_metrics(report\_title, model):

    preds = model.predict(X\_test)

    train\_accuracy = model.score(X\_train, y\_train)

    test\_accuracy = accuracy\_score(y\_test, preds)

    metrics = {

        'model': report\_title,

        'train\_accuracy': train\_accuracy,

        'test\_accuracy': test\_accuracy,

    }

    df = pd.DataFrame(metrics.items()).T

    df.columns = df.iloc[0]

    df = df[1:]

    return df

Để tạo ra một bảng tóm tắt các điểm số phân loại cho các mô hình đã sử dụng, ta sẽ gọi hàm "classifier\_metrics" để tính toán các chỉ số đánh giá hiệu suất của 4 mô hình phân loại khác nhau và lưu kết quả vào các dataframe metrics1, metrics2, metrics3 và metrics4 tương ứng

metrics1 = classifier\_metrics('CountVectorizer + Multinomial Logistic Regression', cv\_log)

metrics2 = classifier\_metrics('TF-IDF + Multinomial Logistic Regression', tf\_log)

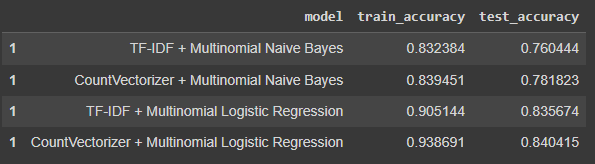
metrics3 = classifier\_metrics('CountVectorizer + Multinomial Naive Bayes', cv\_mnb)

metrics4 = classifier\_metrics('TF-IDF + Multinomial Naive Bayes', tf\_mnb)

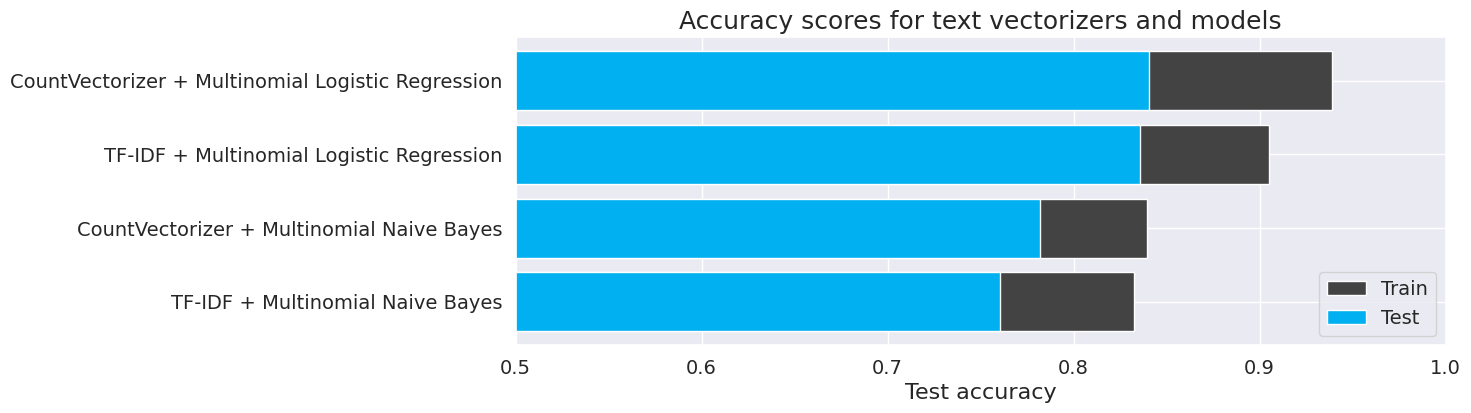
metrics = pd.concat([metrics1, metrics2, metrics3, metrics4]).sort\_values('test\_accuracy')

metrics

Kết quả:



Trực quan hoá điểm số Accuracy trên tập huấn luyện và kiểm tra của từng mô hình bằng biểu đồ thang ngang.



### Xây dựng và đánh giá mô hình bằng các thuật toán học sâu

#### Xây dựng mô hình học sâu: BiLSTM

Bài toán Cyberbullying Tweet Recognition là một bài toán phân loại văn bản, mục đích là nhận diện các tweet có chứa nội dung gây hại hoặc bắt nạt trên mạng xã hội. Vì vậy, BiLSTM là một lựa chọn phù hợp để sử dụng trong bài toán này.

Với khả năng hiểu được các mối quan hệ giữa các từ trong văn bản từ cả hai phía, BiLSTM có thể hiểu được văn bản theo cách “dài hạn” và “ngắn hạn”. Với việc xử lý đồng thời các từ trong một câu, BiLSTM có khả năng hiểu được ngữ cảnh của từ trong câu. Điều này làm cho BiLSTM phù hợp với việc nhận diện các tweet có chứa nội dung gây hại hoặc bắt nạt trên mạng xã hội.

Ngoài ra, BiLSTM còn có khả năng học được các đặc trưng ẩn của văn bản và có thể phân loại các tweet dựa trên những đặc trưng này. BiLSTM đã được sử dụng trong nhiều nghiên cứu về phân loại văn bản và có kết quả khá tốt. Do đó, sử dụng BiLSTM trong bài toán của nhóm có thể giúp nâng cao độ chính xác của mô hình và giúp người dùng đưa ra quyết định chính xác hơn khi đối mặt với các tweet có chứa nội dung gây hại hoặc bắt nạt trên mạng xã hội.

* Xây dựng mô hình BiLSTM: Sử dụng các thư viện hỗ trợ như Keras hoặc PyTorch để xây dựng mô hình BiLSTM với các lớp nhúng (embedding layer), LSTM layer và các lớp fully connected để phân loại tweet thành các nhãn tương ứng.
* Huấn luyện mô hình: Sử dụng tập dữ liệu đã được xử lý, tùy chỉnh các tham số mô hình như số lớp, số nơ-ron, hệ số học và số vòng lặp để tối ưu hóa độ chính xác của mô hình.
* Kiểm tra và đánh giá mô hình: Sử dụng tập dữ liệu kiểm tra để đánh giá độ chính xác của mô hình. Điều chỉnh các tham số mô hình và tập dữ liệu để cải thiện độ chính xác nếu cần thiết.

Hàm Tokenize được sử dụng để chuyển đổi các câu văn bản trong một cột của DataFrame thành một ma trận các số nguyên (integer - encoded matrix), mà mỗi hàng của ma trận tương ứng với một câu văn bản.

Hàm Tokenize nhận được 2 đối số là column và seq-len:

* Đối số column và một cột trong DataFrame chứa các câu văn bản cần chuyển đổi.
* Đối số seq\_len là độ dài tối đa của một câu văn bản, nếu độ dài câu văn bản vượt quá giá trị này thì sẽ cắt bớt, nếu độ dài câu văn bản nhỏ hơn sẽ được thêm các ký tự padding để đạt độ dài bằng seq\_len.

Bên trong hàm Tokenize có các bước thực hiện như sau:

1. Tạo một danh sách (corpus) chứa tất cả các từ trong các câu văn bản trong cột column.
2. Đếm số lần xuất hiện của mỗi từ trong corpus sử dụng Counter.
3. Sắp xếp các từ theo thứ tự giảm dần theo tần suất xuất hiện.
4. Tạo một từ điển (vocab\_to\_int) để ánh xạ mỗi từ sang một số nguyên tương ứng.
5. Chuyển đổi các câu văn bản trong cột column thành các ma trận số nguyên bằng cách sử dụng từ điển đã tạo ở bước trước.
6. Thêm ký tự padding (sử dụng số 0) cho các câu văn bản để đạt độ dài bằng seq\_len, lưu các ma trận kết quả vào biến features.
7. Hàm trả về danh sách các từ đã sắp xếp theo thứ tự tần suất xuất hiện giảm dần và ma trận features chứa các số nguyên tương ứng với các câu văn bản.

def Tokenize(column, seq\_len):

    ##Create vocabulary of words from column

    corpus = [word for text in column for word in text.split()]

    count\_words = Counter(corpus)

    sorted\_words = count\_words.most\_common()

    vocab\_to\_int = {w:i+1 for i, (w,c) in enumerate(sorted\_words)}

    ##Tokenize the columns text using the vocabulary

    text\_int = []

    for text in column:

        r = [vocab\_to\_int[word] for word in text.split()]

        text\_int.append(r)

    ##Add padding to tokens

    features = np.zeros((len(text\_int), seq\_len), dtype = int)

    for i, review in enumerate(text\_int):

        if len(review) <= seq\_len:

            zeros = list(np.zeros(seq\_len - len(review)))

            new = zeros + review

        else:

          new = review[: seq\_len]

        features[i, :] = np.array(new)

    return sorted\_words, features

Sau đó sử dụng hàm Tokenize chuyển đổi data thành một ma trận các số nguyên (integer - encoded matrix), tính toán số lượng từ vựng trong danh sách vocabulary bằng cách thêm đơn vị do từ điển được tạo bắt đầu từ số nguyên 1.

Tiếp tục khởi tạo 2 list rỗng keys và values, lặp qua 20 phần tử đầu tiên của danh sách từ vựng vocabulary bằng vòng lặp for. Mỗi phần tử trong danh sách này là một tuple (word, count) bao gồm từ và số lần xuất hiện trong dữ liệu. Trong mỗi vòng lặp, lấy ra từ (key) và số lần xuất hiện (value) từ tuple hiện tại và thêm chúng vào keys và values tương ứng bằng phương thức append(). Kết quả trả về là 2 list keys và values chứa 20 từ và số lần xuất hiện của chúng trong dữ liệu.

keys = []

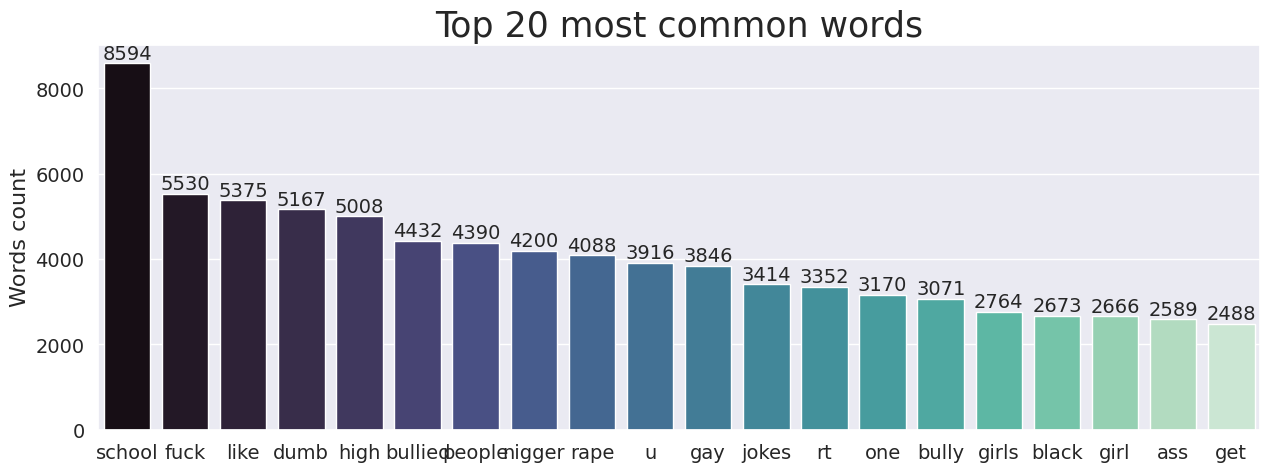
values = []

for key, value in vocabulary[:20]:

    keys.append(key)

    values.append(value)

Sử dụng thư viện matplotlib để vẽ biểu đồ thể hiện số lần xuất hiện của 20 từ phổ biến nhất trong dữ liệu.



**Word Embedding by Wor2vec**

Tạo một danh sách Word2vec\_train\_data chứa các câu được tách thành các từ riêng lẻ, bằng cách sử dụng hàm **split()**. Bên cạnh đó, định nghĩa kích thước của các vector từ trong không gian nhúng bằng biến EMBEDDING\_DIM có giá trị là 200, tạo một ma trận nhúng có kích thước (jml\_vocab, EMBEDDING\_DIM) và toàn bộ các giá trị ban đầu được đặt bằng 0.

Duyệt qua từng từ trong danh sách vocabulary và kiểm tra xem từ đó có tồn tại trong mô hình Word2vec hay không bằng cách sử dụng phương thức **contains()** của đối tượng được trả về từ word2vec\_model. Nếu từ đó tồn tại trong mô hình Word2vec, ma trận nhúng sẽ được cập nhật với các giá trị được lấy từ vector tương ứng của từ đó trong mô hình Word2vec bằng cách sử dụng phương thức **getitem()**

Word2vec\_train\_data = list(map(lambda x: x.split(), X\_train))

EMBEDDING\_DIM = 200

word2vec\_model=Word2Vec(Word2vec\_train\_data, vector\_size=EMBEDDING\_DIM)

#define empty embedding matrix

embedding\_matrix = np.zeros((jml\_vocab, EMBEDDING\_DIM))

#fill the embedding matrix with the pre trained values from word2vec

#    corresponding to word (string), token (number associated to the word)

for word, token in vocabulary:

    if word2vec\_model.wv.\_\_contains\_\_(word):

        embedding\_matrix[token] = word2vec\_model.wv.\_\_getitem\_\_(word)

print("Embedding Matrix Shape:", embedding\_matrix.shape)

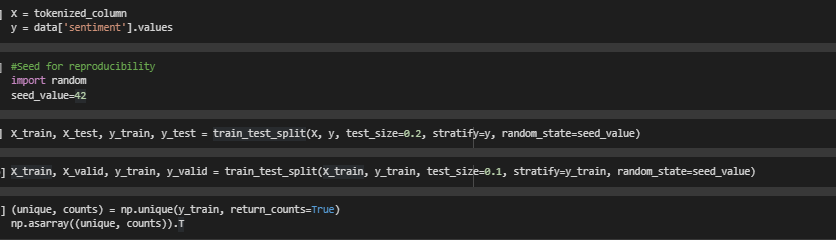
**Train - Validation - Test split**

Chuẩn bị dữ liệu cho quá trình huấn luyện mô hình bằng cách chia tập dữ liệu thành các tập huấn luyện, xác thực và kiểm tra.

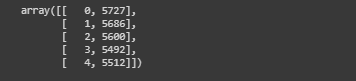
* **X** chứa dữ liệu văn bản đã được mã hóa.
* **y** chứa nhãn tâm trạng mục tiêu.

Hàm train\_test\_split từ thư viện scikit-learn được sử dụng để chia dữ liệu thành các tập huấn luyện, xác thực và kiểm tra. Kích thước tập kiểm tra là 20% của dữ liệu và tập huấn luyện được chia tiếp thành tập xác thực với kích thước là 10% của dữ liệu. Tham số stratify được sử dụng để đảm bảo phân bố các lớp được bảo tồn trong các tập chia. Giá trị ngẫu nhiên được sử dụng để thiết lập một giá trị hạt giống là 42 để đảm bảo khả năng tái sản xuất.

Cuối cùng, mã tính số lượng mẫu cho mỗi lớp trong tập huấn luyện và in chúng dưới dạng một mảng numpy hai chiều trong đó mỗi hàng tương ứng với một lớp, cột đầu tiên là nhãn lớp và cột thứ hai là số lượng mẫu trong lớp đó.



Kết quả cho thấy dữ liệu giữa các lớp bị mất cân bằng.



Tiến hành sử dụng thuật toán Random Over Sampling để giải quyết vấn đề này. Cụ thể, đầu vào của thuật toán này là tập huấn luyện (X\_train và y\_train), trong đó X\_train là tập các câu đã được mã hóa và y\_train là nhãn tương ứng với mỗi câu. RandomOverSampler sẽ sinh ra các mẫu dữ liệu mới cho lớp thiếu số lượng mẫu dữ liệu bằng cách lấy mẫu ngẫu nhiên từ các mẫu dữ liệu hiện có của lớp đó.

Sau khi thực hiện Random Over Sampling, tập huấn luyện đã được cân bằng giữa các lớp và được lưu vào X\_train\_os và y\_train\_os. Cuối cùng, đoạn code in ra số lượng mẫu dữ liệu của từng lớp trong tập huấn luyện sau khi thực hiện Over Sampling. Kết quả thu được:



**PyTorch datasets and dataloaders**

Ở đoạn này, ta tạo các đối tượng DataLoader từ các TensorDataset tương ứng cho tập train, test và validation data.

Đầu tiên, với mỗi tập dữ liệu (train, test, validation), ta dùng hàm TensorDataset để tạo ra một TensorDataset object, với input là hai tensors, một tensor cho features và một tensor cho labels. Ta dùng hàm torch.from\_numpy để chuyển đổi numpy arrays thành tensors.

Tiếp theo, ta sử dụng DataLoader để tạo các mini-batches cho việc train model. Với mỗi DataLoader object, ta cung cấp cho nó TensorDataset object tương ứng và sử dụng các tham số như shuffle, batch\_size, drop\_last để chỉ định các tham số cho việc tạo batch, bao gồm:

* shuffle=True để shuffle các mẫu trước khi tạo batch.
* batch\_size=BATCH\_SIZE để chỉ định kích thước cho mỗi batch.
* drop\_last=True để bỏ qua các mẫu không đủ để tạo batch cuối cùng.

Cuối cùng, các DataLoader objects này sẽ được sử dụng để train model.

train\_data = TensorDataset(torch.from\_numpy(X\_train\_os), torch.from\_numpy(y\_train\_os))

test\_data = TensorDataset(torch.from\_numpy(X\_test), torch.from\_numpy(y\_test))

valid\_data = TensorDataset(torch.from\_numpy(X\_valid), torch.from\_numpy(y\_valid))

BATCH\_SIZE = 32

train\_loader = DataLoader(train\_data, shuffle=True, batch\_size=BATCH\_SIZE, drop\_last=True)

valid\_loader = DataLoader(valid\_data, shuffle=True, batch\_size=BATCH\_SIZE, drop\_last=True)

test\_loader = DataLoader(test\_data, shuffle=True, batch\_size=BATCH\_SIZE, drop\_last=True)

**PyTorch LSTM modeling**

Các biến được sử dụng ở phần này có ý nghĩa:

* **HIDDEN\_DIM**: số lượng neuron trong mỗi lớp ẩn của mạng LSTM. Càng lớn thì mạng càng phức tạp và có khả năng học mẫu phức tạp hơn.
* **LSTM\_LAYERS**: số lượng lớp LSTM xếp chồng lên nhau. Càng nhiều thì mô hình càng phức tạp và có khả năng học mẫu phức tạp hơn.
* **LR**: tỉ lệ học của mô hình. Đây là tham số quan trọng để tinh chỉnh mô hình, quyết định mức độ điều chỉnh trọng số của các tế bào LSTM trong mỗi vòng lặp.
* **DROPOUT**: giá trị dropout của LSTM. Đây là một kỹ thuật chính để giảm overfitting trong mô hình, nó làm cho một số neuron trong mạng bị "dropout" ngẫu nhiên, từ đó giảm khả năng mô hình học các mẫu sai.
* **BIDIRECTIONAL**: là một biến boolean cho biết liệu mô hình sử dụng mạng LSTM hai chiều hay không. Nếu đặt là True, mô hình sử dụng cả forward LSTM và backward LSTM để học từ các thông tin của từ đầu đến cuối câu, và ngược lại.
* **EPOCHS**: số lượng epoch trong quá trình huấn luyện mô hình.
* **DEVICE**: là thiết bị sử dụng để thực hiện tính toán trên mô hình. Nếu máy tính có GPU thì sử dụng 'cuda', nếu không thì sử dụng 'cpu'.

import torch

import torch.nn as nn

NUM\_CLASSES = 5 #We are dealing with a multiclass classification of 5 classes

HIDDEN\_DIM = 100 #number of neurons of the internal state (internal neural network in the LSTM)

LSTM\_LAYERS = 1 #Number of stacked LSTM layers

LR = 3e-4 #Learning rate

DROPOUT = 0.5 #LSTM Dropout

BIDIRECTIONAL = True #Boolean value to choose if to use a bidirectional LSTM or not

EPOCHS = 5 #Number of training epoch

DEVICE = 'cuda' if torch.cuda.is\_available() else 'cpu'

Xây dựng lớp class định nghĩa một mô hình mạng neural network (NN) sử dụng một kiến trúc **BiLSTM** (Bidirectional Long Short-Term Memory), được sử dụng để phân loại cảm xúc từ văn bản.

Class **BiLSTM\_Sentiment\_Classifier** là một subclass của nn.Module. Các tham số được khởi tạo trong hàm khởi tạo của class bao gồm:

* jml\_vocab: số lượng từ vựng trong bộ từ vựng
* embedding\_dim: kích thước của ma trận embedding
* hidden\_dim: số lượng neuron của LSTM (internal state) trong mỗi lớp
* num\_classes: số lượng lớp phân loại
* lstm\_layers: số lượng lớp LSTM
* bidirectional: biến boolean xác định việc sử dụng mạng LSTM hai chiều hay một chiều
* batch\_size: số lượng dữ liệu đầu vào được truyền vào mô hình mỗi lần (batch size)
* dropout: tỷ lệ dropout giữa các lớp của mô hình

Mô hình bao gồm các phần chính sau:

* embedding: lớp embedding
* lstm: lớp BiLSTM
* fc: lớp fully connected layer để đưa ra dự đoán cho mỗi lớp
* softmax: hàm softmax để tính xác suất dự đoán cho mỗi lớp

Hàm **forward** của mô hình thực hiện tính toán output của mạng khi truyền vào đầu vào x và trạng thái ẩn hidden. Hàm **init\_hidden** được sử dụng để khởi tạo trạng thái ẩn của mô hình.

class BiLSTM\_Sentiment\_Classifier(nn.Module):

    def \_\_init\_\_(self, jml\_vocab, embedding\_dim, hidden\_dim, num\_classes, lstm\_layers, bidirectional, batch\_size, dropout):

        super(BiLSTM\_Sentiment\_Classifier, self).\_\_init\_\_()

        self.lstm\_layers = lstm\_layers

        self.num\_directions = 2 if bidirectional else 1

        self.hidden\_dim = hidden\_dim

        self.num\_classes = num\_classes

        self.batch\_size = batch\_size

        self.embedding = nn.Embedding(jml\_vocab, embedding\_dim)

        self.lstm = nn.LSTM(embedding\_dim,

                            hidden\_dim,

                            num\_layers=lstm\_layers,

                            dropout=dropout,

                            bidirectional=bidirectional,

                            batch\_first=True)

        self.fc = nn.Linear(hidden\_dim \* self.num\_directions, num\_classes)

        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, x, hidden):

        ##EMBEDDING LAYER

        embedded = self.embedding(x)

        #LSTM LAYERS

        out, hidden = self.lstm(embedded, hidden)

        #Extract only the hidden state from the last LSTM cell

        out = out[:, -1, :]

        #FULLY CONNECTED LAYERS

        out = self.fc(out)

        out = self.softmax(out)

        return out, hidden

    def init\_hidden(self, batch\_size):

        #Initialization of the LSTM hidden and cell states

        h0 = torch.zeros((self.lstm\_layers \* self.num\_directions, batch\_size, self.hidden\_dim)).to(DEVICE)

        c0 = torch.zeros((self.lstm\_layers \* self.num\_directions, batch\_size, self.hidden\_dim)).to(DEVICE)

        hidden = (h0, c0)

        return hidden

Tiếp theo tạo ra một đối tượng mô hình **BiLSTM\_Sentiment\_Classifier** với các tham số đã được xác định trước. Các tham số của mô hình được chuyển sang thiết bị **DEVICE** (GPU nếu có hoặc CPU). Mô hình cũng được khởi tạo với ma trận embedding được truyền vào và cho phép ma trận này được điều chỉnh để cải thiện độ chính xác của mô hình. Cuối cùng, mô hình được in ra để xác nhận lại các thông số của mô hình.

model = BiLSTM\_Sentiment\_Classifier(jml\_vocab, EMBEDDING\_DIM, HIDDEN\_DIM, NUM\_CLASSES, LSTM\_LAYERS, BIDIRECTIONAL, BATCH\_SIZE, DROPOUT)

model = model.to(DEVICE)

#Initialize embedding with the previously defined embedding matrix

model.embedding.weight.data.copy\_(torch.from\_numpy(embedding\_matrix))

#Allow the embedding matrix to be fined tuned to better adapt to out dataset and get higher accuracy

model.embedding.weight.requires\_grad=True

print(model)

Nhóm khởi tạo **criterion** và **optimizer** để sử dụng trong quá trình huấn luyện mô hình:

* **nn.NLLLoss()** là hàm mất mát cho bài toán phân loại đa lớp với đầu ra là xác suất, trong đó đầu vào được tính toán bằng hàm **softmax**. Nó tính toán mất mát bằng cách lấy đầu ra của hàm **softmax**, tính toán tổng **logarit** số xác suất dự đoán đúng của mỗi lớp.
* **torch.optim.AdamW()** là một phương pháp tối ưu hóa thường được sử dụng cho mạng nơ-ron sâu. Trong đó, ta truyền vào **model.parameters()** để tối ưu hóa toàn bộ các tham số của mô hình. Tham số **lr** là learning rate để điều chỉnh tốc độ học của mô hình. **weight\_decay** được sử dụng để kiểm soát việc **regularization** trên trọng số của mô hình để tránh tình trạng overfitting.

Trong PyTorch, hàm nn.NLLLoss() (negative log likelihood loss) được sử dụng để tính toán độ lỗi giữa các dự đoán và nhãn thực tế. Hàm này thường được sử dụng cho các bài toán phân loại với nhiều lớp.

Trong khi đó, torch.optim.AdamW() là một thuật toán tối ưu hóa được sử dụng để tối ưu hóa các tham số của mô hình. Nó là một biến thể của thuật toán Adam và được tối ưu hóa để làm việc tốt hơn với các kiến trúc mạng lớn. Trong đoạn code này, ta sử dụng AdamW với learning rate (LR) là LR và hệ số giảm trọng lượng (weight decay) là 5e-6.

**criterion = nn.NLLLoss()**

**optimizer = torch.optim.AdamW(model.parameters(), lr=LR, weight\_decay = 5e-6)**

**BiLSTM Training Loop**

Đoạn code này là vòng lặp huấn luyện mô hình LSTM sử dụng các dữ liệu huấn luyện và kiểm định để đánh giá mô hình.

Đầu tiên, chương trình tính toán số bước huấn luyện và số bước kiểm định của mỗi **epoch**. Sau đó, chương trình sử dụng **early stopping** để dừng huấn luyện trước nếu mô hình không cải thiện độ chính xác của tập kiểm định sau một số **epoch** nhất định.

Tiếp theo, vòng lặp huấn luyện được thực hiện. Các biến được khởi tạo để lưu giữ giá trị loss, accuracy và dự đoán đối với từng batch của tập huấn luyện và kiểm định.

Trong quá trình huấn luyện, mô hình được đưa vào chế độ train để tính toán giá trị loss dựa trên output của mô hình và nhãn đích, sử dụng thuật toán lan truyền ngược. Sau đó, biến đếm được tăng lên để lưu giữ số lượng các dữ liệu được phân loại đúng và tổng số dữ liệu trong mỗi batch. Cuối cùng, các giá trị loss và accuracy được cập nhật vào danh sách tương ứng.

Trong quá trình kiểm định, mô hình được đưa vào chế độ evaluation để không tính toán đạo hàm và xây dựng mô hình. Các giá trị loss và accuracy được tính dựa trên output của mô hình và nhãn đích. Cuối cùng, các giá trị loss và accuracy được cập nhật vào danh sách tương ứng.

Nếu độ chính xác của tập kiểm định tốt hơn so với mô hình trước đó, mô hình được lưu lại. Ngược lại, nếu độ chính xác của tập kiểm định không cải thiện sau một số epoch nhất định, mô hình sẽ được dừng sớm.

Cuối cùng, giá trị loss và accuracy trung bình của các batch được in ra sau mỗi epoch.

total\_step = len(train\_loader)

total\_step\_val = len(valid\_loader)

early\_stopping\_patience = 4

early\_stopping\_counter = 0

valid\_acc\_max = 0 # Initialize best accuracy top 0

for e in range(EPOCHS):

    #lists to host the train and validation losses of every batch for each epoch

    train\_loss, valid\_loss  = [], []

    #lists to host the train and validation accuracy of every batch for each epoch

    train\_acc, valid\_acc  = [], []

    #lists to host the train and validation predictions of every batch for each epoch

    y\_train\_list, y\_val\_list = [], []

    #initalize number of total and correctly classified texts during training and validation

    correct, correct\_val = 0, 0

    total, total\_val = 0, 0

    running\_loss, running\_loss\_val = 0, 0

    ####TRAINING LOOP####

    model.train()

    for inputs, labels in train\_loader:

        inputs, labels = inputs.to(DEVICE), labels.to(DEVICE) #load features and targets in device

        h = model.init\_hidden(labels.size(0))

        model.zero\_grad() #reset gradients

        output, h = model(inputs,h) #get output and hidden states from LSTM network

        loss = criterion(output, labels)

        loss.backward()

        running\_loss += loss.item()

        optimizer.step()

        y\_pred\_train = torch.argmax(output, dim=1) #get tensor of predicted values on the training set

        y\_train\_list.extend(y\_pred\_train.squeeze().tolist()) #transform tensor to list and the values to the list

        correct += torch.sum(y\_pred\_train==labels).item() #count correctly classified texts per batch

        total += labels.size(0) #count total texts per batch

    train\_loss.append(running\_loss / total\_step)

    train\_acc.append(100 \* correct / total)

    ####VALIDATION LOOP####

    with torch.no\_grad():

        model.eval()

        for inputs, labels in valid\_loader:

            inputs, labels = inputs.to(DEVICE), labels.to(DEVICE)

            val\_h = model.init\_hidden(labels.size(0))

            output, val\_h = model(inputs, val\_h)

            val\_loss = criterion(output, labels)

            running\_loss\_val += val\_loss.item()

            y\_pred\_val = torch.argmax(output, dim=1)

            y\_val\_list.extend(y\_pred\_val.squeeze().tolist())

            correct\_val += torch.sum(y\_pred\_val==labels).item()

            total\_val += labels.size(0)

        valid\_loss.append(running\_loss\_val / total\_step\_val)

        valid\_acc.append(100 \* correct\_val / total\_val)

    #Save model if validation accuracy increases

    if np.mean(valid\_acc) >= valid\_acc\_max:

        torch.save(model.state\_dict(), './state\_dict.pt')

        print(f'Epoch {e+1}:Validation accuracy increased ({valid\_acc\_max:.6f} --> {np.mean(valid\_acc):.6f}).  Saving model ...')

        valid\_acc\_max = np.mean(valid\_acc)

        early\_stopping\_counter=0 #reset counter if validation accuracy increases

    else:

        print(f'Epoch {e+1}:Validation accuracy did not increase')

        early\_stopping\_counter+=1 #increase counter if validation accuracy does not increase

    if early\_stopping\_counter > early\_stopping\_patience:

      print('Early stopped at epoch :', e+1)

      break

    print(f'\tTrain\_loss : {np.mean(train\_loss):.4f} Val\_loss : {np.mean(valid\_loss):.4f}')

    print(f'\tTrain\_acc : {np.mean(train\_acc):.3f}% Val\_acc : {np.mean(valid\_acc):.3f}%')

#### Đánh giá

Để dự đoán nhãn cho tập dữ liệu kiểm tra (test set), nhóm đặt mô hình vào chế độ đánh giá (evaluation mode) bằng cách gọi hàm **model.eval()**. Sau đó, ta dùng một vòng lặp để lấy ra từng đầu vào và nhãn trong tập kiểm tra (test set). Đầu vào và nhãn được đưa vào trên thiết bị tính toán (DEVICE), sau đó, ta khởi tạo hidden state cho kiểm tra (test) bằng cách gọi hàm **model.init\_hidden()**.

Sau đó, ta sử dụng mô hình đã được huấn luyện để tính toán đầu ra (output) dựa trên đầu vào (**inputs**) và hidden state **(test\_h**) đã được khởi tạo. Tiếp theo, ta sử dụng hàm **torch.argmax()** để lấy ra chỉ số của lớp có xác suất cao nhất trong đầu ra của mô hình, từ đó dùng phương thức **squeeze()** và **tolist()** để chuyển đổi định dạng và lưu giữ vào danh sách dự đoán **y\_pred\_list**. Tương tự, ta lưu giữ nhãn thực tế của tập kiểm tra vào danh sách **y\_test\_list**.

Cuối cùng, đoạn code sẽ trả về hai danh sách **y\_pred\_list** và **y\_test\_list** chứa kết quả dự đoán và nhãn thực tế tương ứng cho tập kiểm tra.

model.eval()

y\_pred\_list = []

y\_test\_list = []

for inputs, labels in test\_loader:

    inputs, labels = inputs.to(DEVICE), labels.to(DEVICE)

    test\_h = model.init\_hidden(labels.size(0))

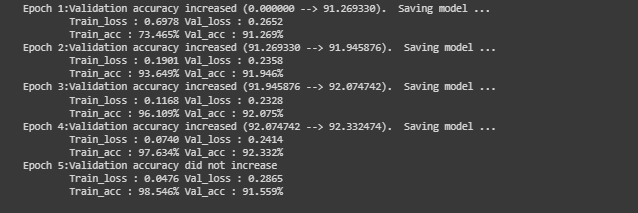
    output, val\_h = model(inputs, test\_h)

    y\_pred\_test = torch.argmax(output, dim=1)

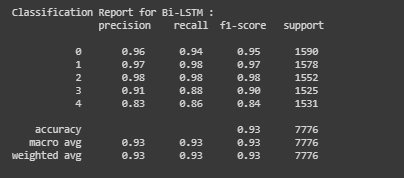
    y\_pred\_list.extend(y\_pred\_test.squeeze().tolist())

    y\_test\_list.extend(labels.squeeze().tolist())

Dưới đây là kết quả sau khi Training Loop



Đây là kết quả nhận được sau khi áp dụng BiLSTM cho việc dự đoán nhãn cho tập dữ liệu:

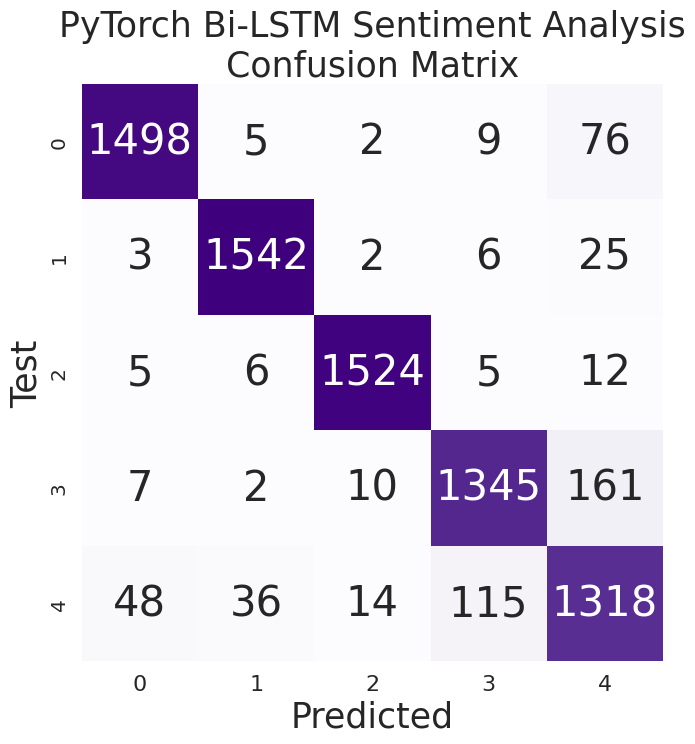


Đầu tiên, đánh giá được thực hiện bằng việc in ra một báo cáo phân loại, bao gồm các chỉ số precision, recall và f1-score cho mỗi lớp và cho toàn bộ tập dữ liệu.

Từ kết quả báo cáo này, có thể thấy rằng mô hình BiLSTM đã được độ chính xác (accuracy) trung bình 93% cho các lớp. Các chỉ số precision, recall và f1-score cho mỗi lớp đều khá cao, chỉ có nhãn ‘not\_cyberbullying’ có kết quả thấp hơn so với các lớp khác. Tổng thể, kết quả báo cáo phân loại cho thấy mô hình có khả năng phân loại tốt và đáng tin cậy.

So với các chỉ số của thuật toán máy học CountVectorizer + Multinomial Logistic Regression (mô hình đánh giá có kết quả tốt nhất), các chỉ số của mô hình BiLSTM đều cao hơn, chính vì vậy nhóm tiến hành sử dụng mô hình BiLSTM để dự đoán thử một câu tweet xem đây là loại bạo lực mạng ở dạng nào.

Để trực quan hóa các chỉ số này, nhóm xây dựng hàm **‘conf\_matrix’** để tạo ra ma trận nhầm lẫn và biểu đồ heatmap.



### Áp dụng mô hình

Nhóm tiến hành lấy một câu ví dụ để đánh giá mô hình BiLSTM.

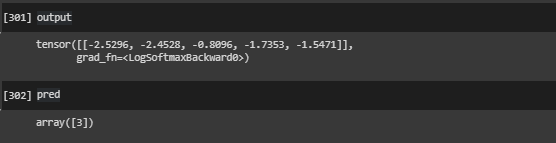
Câu ví dụ:

sentence = "But it’s ok for a female reporter to call Kayleigh a bitch on national television."

Tiếp theo sẽ làm sạch câu văn bản này, xây dựng chức năng chuyển đổi câu đầu vào thành vector số nguyên (token) bằng cách sử dụng một từ điển (vocabulary) được xây dựng sẵn và một mô hình đã được huấn luyện trước đó. Cụ thể:

* Đầu tiên, câu đầu vào được tách thành các token (từ) bằng hàm split().
* Sau đó, từ điển được dùng để gán cho mỗi từ một chỉ số nguyên (index) thông qua dictionary comprehension: {word: i for i, word in enumerate(vocabulary)}.
* Nếu từ không có trong từ điển thì sẽ được gán chỉ số của token <unk> (tức là từ không xác định) bằng word2idx['<unk>'] = len(vocabulary).
* Các chỉ số này được sử dụng để tạo ra một mảng numpy gồm các token ids.
* Tiếp theo, câu đầu vào được chuẩn bị thành đầu vào cho mô hình. Mảng các token ids này được chuyển thành tensor input\_sentence với unsqueeze(0) để tạo ra batch\_size của 1. input\_sentence cũng được chuyển sang thiết bị DEVICE trước khi đưa vào mô hình.
* Một hidden state mới được khởi tạo với batch\_size của 1 bằng hàm init\_hidden(batch\_size).
* Cuối cùng, đầu ra được lấy ra từ mô hình thông qua hàm forward(input\_sentence, hidden). Chỉ số của lớp có xác suất dự đoán cao nhất được tính toán thông qua hàm np.argmax(), rồi được tăng lên một đơn vị (từ chỉ số lớp bắt đầu từ 1) và lưu trữ vào biến pred.

Đây là kết quả:



Như đã xử lý, thay thế các sentiments thành các số:

data['sentiment'].replace({'religion':0,'age':1,'ethnicity':2,'gender':3,'not\_cyberbullying':4})

Kết quả cho thấy, câu ví dụ này là loại Gender Cyberbullying hay bạo lực mạng giới tính.

# CHƯƠNG 5. KẾT LUẬN

Phân loại văn bản là một vấn đề học máy cơ bản với các ứng dụng trên nhiều sản phẩm, nhiều lĩnh vực trong đời sống. Trong phần trình bày trên, chúng em đã chia nhỏ quy trình phân loại văn bản thành nhiều bước. Tuy nhiên với nhiều bài toán thực tế khác nhau, đối với mỗi bước sẽ có các tiếp cận tùy chỉnh dựa trên các đặc điểm cụ thể của tập dữ liệu ta có và mục đích sử dụng khác nhau.

# TÀI LIỆU THAM KHẢO

1. Learn About Long Short-Term Memory (LSTM) Algorithms. Analytics Vidhya. Retrieved April 20, 2023, from <https://www.analyticsvidhya.com/blog/2021/03/introduction-to-longshort-term-memory-lstm/>
2. What Is The Difference Between TfidfVectorizer And CountVectorizer?EML. Retrieved April 20, 2023, from <https://enjoymachinelearning.com/blog/countvectorizer-vstfidfvectorizer/#:~:text=CountVectorizer%20simply%20counts%20the%20number,is%2>
3. Logistic Regression. Towardsdatascience. Retrieved April 20, 2023, from

<https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc>