ĐẠI HỌC UEH

TRƯỜNG CÔNG NGHỆ VÀ THIẾT KẾ

KHOA CÔNG NGHỆ THÔNG TIN KINH DOANH

ĐỒ ÁN MÔN HỌC: XỬ LÝ NGÔN NGỮ TỰ NHIÊN

ĐỀ TÀI: PHÂN LOẠI TWEET THẢM HỌA ÁP DỤNG KỸ THUẬT XỬ LÝ NGÔN NGỮ TỰ NHIÊN

Giảng viên: TS. Đặng Ngọc Hoàng Thành

Mã học phần: 23C1INF50907601

Nhóm sinh viên: Hà Gia Hiến - 31211021362

Lý Gia Thuận - 31211026754

Lê Minh Triều - 31211027681

Thành phố Hồ Chí Minh, ngày 21 tháng 12 năm 2023

MỤC LỤC

LỜI NÓI ĐẦU

Kính gửi thầy Thành,

Trong lời mở đầu của đồ án này, chúng em xin dành một lời cảm ơn sâu sắc đến thầy.

Những bài giảng và buổi thực hành của thầy đã không chỉ giúp chúng em nắm bắt được những lý thuyết cơ bản mà còn khám phá được những xu hướng mới nhất trong lĩnh vực A.I, cụ thể là NLP. Đó sẽ là những kiến thức quan trọng để chúng em chuẩn bị bước vào thế giới thực và ứng dụng chúng vào các dự án trong công việc sau này.

Chúng em đã nỗ lực hết mình để áp dụng những kiến thức này vào bài đồ án, nhưng tất nhiên sẽ không tránh được những sai sót và hạn chế, mong thầy nhiệt tình góp ý để nhóm chúng em rút kinh nghiệm và tiếp thu tốt hơn.

Chúc thầy nhiều sức khỏe, xin cảm ơn thầy đã truyền đạt kiến thức quý báu cho sinh viên bằng cả tấm lòng nhiệt tình và sự tận tâm!

CHƯƠNG 1: Tổng quan đề tài

1.1. Về mục tiêu

Bộ dữ liệu **Disaster Tweets** lưu trữ hơn 11.000 bài tweet, mỗi bài đều được phân lớp một cách thủ công: tweet nói về thảm họa (1) và tweet không nói về thảm họa (0), cũng như bao gồm từ khóa chính trong tweet đó.

Mục tiêu của nhóm là chọn và xây dựng 3 mô hình phù hợp cho việc phân lớp, dựa trên kết quả của chúng để đưa ra kết luận tổng. Cuối cùng là xây dựng một ứng dụng với mục đích demo.

1.2. Phương pháp nghiên cứu

- Xử lý dữ liệu bị thiếu và trùng lặp.
- Làm sạch dữ liệu bằng cách loại bỏ các dấu câu, URL, emoji, …
- Sử dụng các loại biểu đồ chuyên dụng để trực quan hoá dữ liệu.
- Xử lý mất cân bằng dữ liệu với kỹ thuật Undersampling.
- Véc-tơ hóa dữ liệu với kỹ thuật TF-IDF.
- Huấn luyện 3 mô hình phân lớp, đánh giá mô hình với các bộ chỉ số (Accuracy, Precision, Recall, F1-score).

1.3. Tài nguyên sử dụng

- Ngôn ngữ lập trình: Python.
- Những thư viện *pandas, matplotlib, nltk, tensorflow, …*
- Bộ dữ liệu *"tweets.csv"*.

CHƯƠNG 2: Xử lý ngôn ngữ tự nhiên

2.1. Khái niệm cơ bản

Xử lý ngôn ngữ tự nhiên (NLP) là một lĩnh vực thuộc khoa học máy tính và trí tuệ nhân tạo, tập trung vào việc phát triển các phương pháp để máy tính có khả năng hiểu và tương tác với ngôn ngữ tự nhiên của con người. NLP hướng đến mục tiêu làm cho máy có khả năng xử lý, hiểu và tạo ra ngôn ngữ tự nhiên một cách tự động.

NLP còn liên quan chặt chẽ đến việc nghiên cứu cách máy tính có thể hiểu cấu trúc ngôn ngữ, ngữ cảnh, ngữ pháp và ý nghĩa của câu trong văn bản hoặc ngôn ngữ nói. Đồng thời, nó tập trung vào việc phát triển các thuật toán và mô hình để máy tính có thể thực hiện các nhiệm vụ như dịch thuật, nhận diện giọng nói, phân loại văn bản và tóm tắt nội dung.

NLP đang ngày càng trở nên quan trọng khi mà ứng dụng trí tuệ nhân tạo và hệ thống thông minh ngày càng phát triển. Các ứng dụng của NLP rải rác từ các ứng dụng di động đến những dự án lớn như xử lý dữ liệu lớn và phân tích ý kiến trên mạng xã hội.

Tóm lại, NLP đóng vai trò quan trọng trong việc kết nối con người và máy tính thông qua ngôn ngữ tự nhiên, tạo ra trải nghiệm tương tác giữa người và máy một cách mượt mà và hiệu quả.

Về các loại mô hình được sử dụng trong đồ án này, nhóm chúng em sẽ trình bày cơ sở lý thuyết và đi sâu hơn tại chương 4, khi bắt đầu xây dựng mô hình.

CHƯƠNG 3: Tiền xử lý & trực quan hóa dữ liệu

3.1. Tổng quan về bộ dữ liệu

Bộ dữ liệu chứa hơn 11.000 tweet liên quan đến các từ khóa thảm họa như *"crash"*, *"quarantine"*, *"bush fires"* cùng với địa điểm và chính từ khóa đó. Những tweet này được thu thập vào ngày 14 tháng 1 năm 2020. Do đó có một số chủ đề nổi bật nhận được nhiều lượt tweet:

- Núi lửa Taal phun trào ở Batangas, Philippines
- Đại dịch Coronavirus
- Cháy rừng ở Australia
- Iran bắn hạ máy bay PS752
- …

Bộ dữ liệu được lưu trữ dưới dạng bảng, gồm 5 cột và 11370 dòng cụ thể như sau:

STT	Tên cột	Ý nghĩa	Ghi chú
1	ID	Mã định danh cho mỗi tweet	Unique
2	keyword	Một từ khóa cụ thể liên quan tới thảm họa từ tweet	
3	location	Vị trí nơi tweet được đăng tải	
4	text	Toàn bộ nội dung của tweet	
5	target	Biểu thị xem một tweet có nói về một thảm họa thực sự hay không	0: không 1: có

3.2. Tiền xử lý dữ liệu

3.2.1. Kiểm tra dữ liệu bị thiếu & trùng lặp

Tiến hành kiểm tra những ô dữ liệu bị thiếu và bị trùng lặp. Nhận thấy không có dữ liệu trùng lặp và tất cả dữ liệu bị thiếu đều nằm ở cột 'location'.

```
id              0
keyword         0
location     3418
text            0
target          0
dtype: int64
```

Nhận thấy giá trị ở cột 'location' không ảnh hưởng tới kết quả của bài toán nên nhóm quyết định bỏ cột location ra khỏi dataframe.

3.2.2. Làm sạch dữ liệu

Loại bỏ URL ra khỏi văn bản:

Tạo một biểu thức chính quy (regex pattern) để tìm kiếm các đoạn văn bản có dạng URL. Cụ thể:

- **https?://**: Kết hợp với "http://" hoặc "https://".
- **S+|www\.\S+**: Kết hợp với một chuỗi gồm ít nhất một ký tự không phải khoảng trắng (nghĩa là kết hợp với domain và phần path của URL).
- **|:** Ký hiệu OR, cho phép kết hợp cả URL bắt đầu bằng "http://" hoặc "https://" hoặc "www."

Sau đó kết hợp phương thức **'sub'** của biểu thức chính quy để thay thế mọi đoạn văn bản phù hợp với biểu thức chính quy bằng chuỗi rỗng (' ').

```
def remove_url(text):
    pattern = re.compile(r'https?://\S+|www\.\S+')
    result_text = pattern.sub(r'', text)
    return result_text
```

Loại bỏ các stop words trong văn bản:

Đầu tiên phân tách các từ thành token trong một danh sách bằng word_tokenizer của thư viện nltk sau loại bỏ các stop words bằng list comprehension sau đó kết hợp lại thành chuỗi.

```
def remove_stopwords(text):
    if not isinstance(text, str):
        return ""
    word_tokens = word_tokenize(text)
    filtered_text = [word for word in word_tokens if word not in stop_words]
    return ' '.join(filtered_text)
```

Loại bỏ tất cả các dấu câu từ một đoạn văn bản:

Tạo một bảng chuyển đổi (translation table) sử dụng phương thức str.maketrans. Bảng này sẽ được sử dụng bởi phương thức translate sau này để loại bỏ các ký tự dấu câu.

```
def remove_punct(text):
    translation_table = str.maketrans('', '', string.punctuation)
    return text.translate(translation_table)
```

Loại bỏ các ký tự HTML từ một đoạn văn bản thông thường:

Tạo một biểu thức chính quy (regex pattern) để tìm kiếm các đoạn ký tự HTML. Cụ thể:

- **<:** Ký tự mở đầu của một thẻ HTML. .
- ***?**: Kết hợp với bất kỳ ký tự nào (bao gồm cả ký tự xuống dòng) không-greedy, tức là kết hợp với ít ký tự nhất có thể.
- **>:** Ký tự đóng của một thẻ HTML.

```
def remove_html(text):
    html_pattern = re.compile(r'<.*?>')
    return html_pattern.sub(r'', text)
```

Loại bỏ các ký tự emoji từ một đoạn văn bản:

Tạo một biểu thức chính quy (emoji_pattern) sử dụng hàm **re.compile**. Biểu thức này được thiết kế để kết hợp với nhiều ký tự Unicode liên quan đến emoji. Bao gồm nhiều phạm vi Unicode, mỗi phạm vi tương ứng với các loại emoji khác nhau.

Sau đó sử dụng phương thức '**sub**' của biểu thức chính quy để thay thế tất cả các kết quả trùng khớp với biểu thức chính quy trong đoạn văn bản đầu vào (text) bằng chuỗi rỗng (' '), qua đó loại bỏ các ký tự emoji.

```
def remove_emoji(text):
    emoji_pattern = re.compile("["
```

```
                           u"\U0001F600-\U0001F64F"  # emoticons
                           u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                           u"\U0001F680-\U0001F6FF"  # transport & map
symbols
                           u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', text)
```

Chuyển các từ rút gọn, viết tắt (contractions) về dạng đầy đủ trong văn bản:

Thư viện re để sử dụng biểu thức chính quy. Các dòng tiếp theo chứa các biểu thức chính quy (regex) được sử dụng để thay thế các từ viết tắt bằng các từ đầy đủ. Ví dụ:

- re.sub(r"won\'t", " will not", text): Thay thế "won't" bằng "will not".
- re.sub(r"\'re", " are", text): Thay thế "'re" bằng " are".

Mỗi biểu thức chính quy sử dụng hàm **re.sub** để thay thế một từ viết tắt bằng từ đầy đủ trong văn bản.

```
def decontraction(text):
    # Từ rút gọn với chữ 'not'
    text = re.sub(r"won\'t", " will not", text)
    text = re.sub(r"can\'t", " can not", text)
    text = re.sub(r"shan\'t", " shall not", text)
    text = re.sub(r"sha\n't", " shall not", text)
    text = re.sub(r"n\'t", " not", text)
    # Từ rút gọn với 'have'
    text = re.sub(r"won\'t've", " will not have", text)
    text = re.sub(r"can\'t've", " can not have", text)
    text = re.sub(r"n\'t've", " not have", text)
    text = re.sub(r"\'d've", " would have", text)
    text = re.sub(r"\'ll've", " will have", text)
    # Những từ rút gọn khác
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'s", " is", text)
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'t", " not", text)
    text = re.sub(r"\'ve", " have", text)
```

```
    text = re.sub(r"\'m", " am", text)
    # Từ viết tắt
    text = re.sub(r"ma\'am", " madam", text)
    text = re.sub(r"let\'s", " let us", text)
    text = re.sub(r"o\'clock", " of the clock", text)
    text = re.sub(r"y\'all", " you all", text)

    return text
```

Tách riêng các thành phần chữ cái và số từ một đoạn văn bản

Sử dụng biểu thức chính quy để tìm kiếm tất cả các chuỗi con trong văn bản text. Biểu thức chính quy này có hai phần:

- **[^\W\d_]**+: Kết hợp với các chuỗi con không chứa ký tự không phải chữ cái, không phải số, và không phải dấu gạch dưới (underscore).
- |**\d**+: Kết hợp với chuỗi số.

```
def separate_alphanumeric(text):
    words = re.findall(r"[^\W\d_]+|\d+", text)
    return " ".join(words)
```

Tạo một hàm để tổng các hàm đã tạo:

Việc tạo hàm này sẽ mang lại giá trị sử dụng không chỉ trong dataframe mà còn mà để xử lí những input của application (sẽ build ở sau)

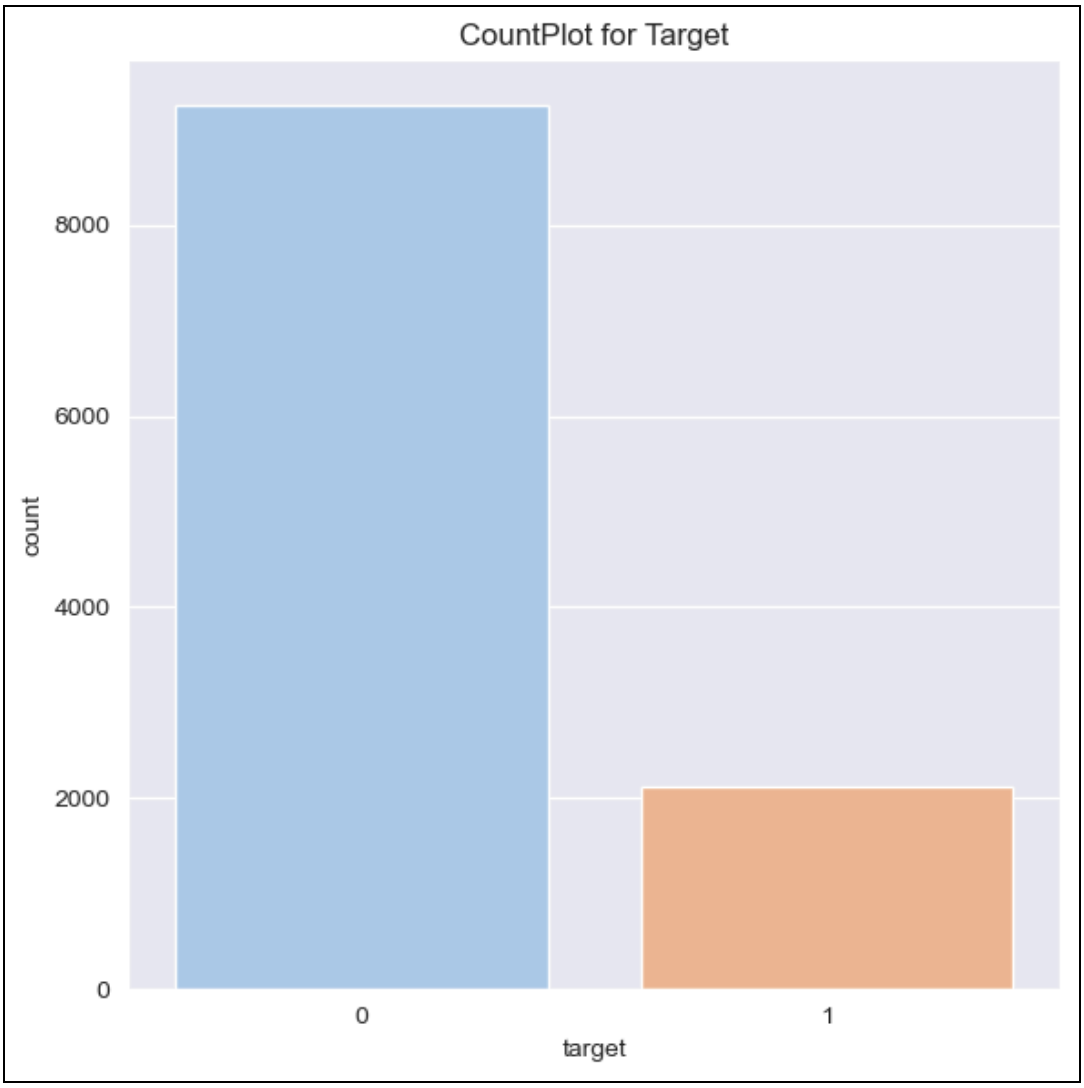
```
def preprocess(text):
    text = remove_url(text)
    text= remove_punct(text)
    text = remove_emoji(text)
    text = decontraction(text)
    text = separate_alphanumeric(text)
    text = unique_char(cont_rep_char,text)
    text = remove_stopwords(text)
    return text
```

Dữ liệu hoàn chỉnh sau khi làm sạch:

	text
0	Communal violence Bhainsa Telangana Stones pel...
1	Telangana Section 144 imposed Bhainsa January ...
2	Arsonist sets cars ablaze dealership
3	Arsonist sets cars ablaze dealership
4	Lord Jesus love brings freedom pardon Fill Hol...
...	...
11365	Media warned us well advance This wrecked whol...
11366	feel directly attacked consider moonbin amp ji...
11367	feel directly attacked consider moonbin amp ji...
11368	ok remember outcast nd dora au THOSE AU WRECKE...
11369	Jake Corway wrecked running 14 th IRP

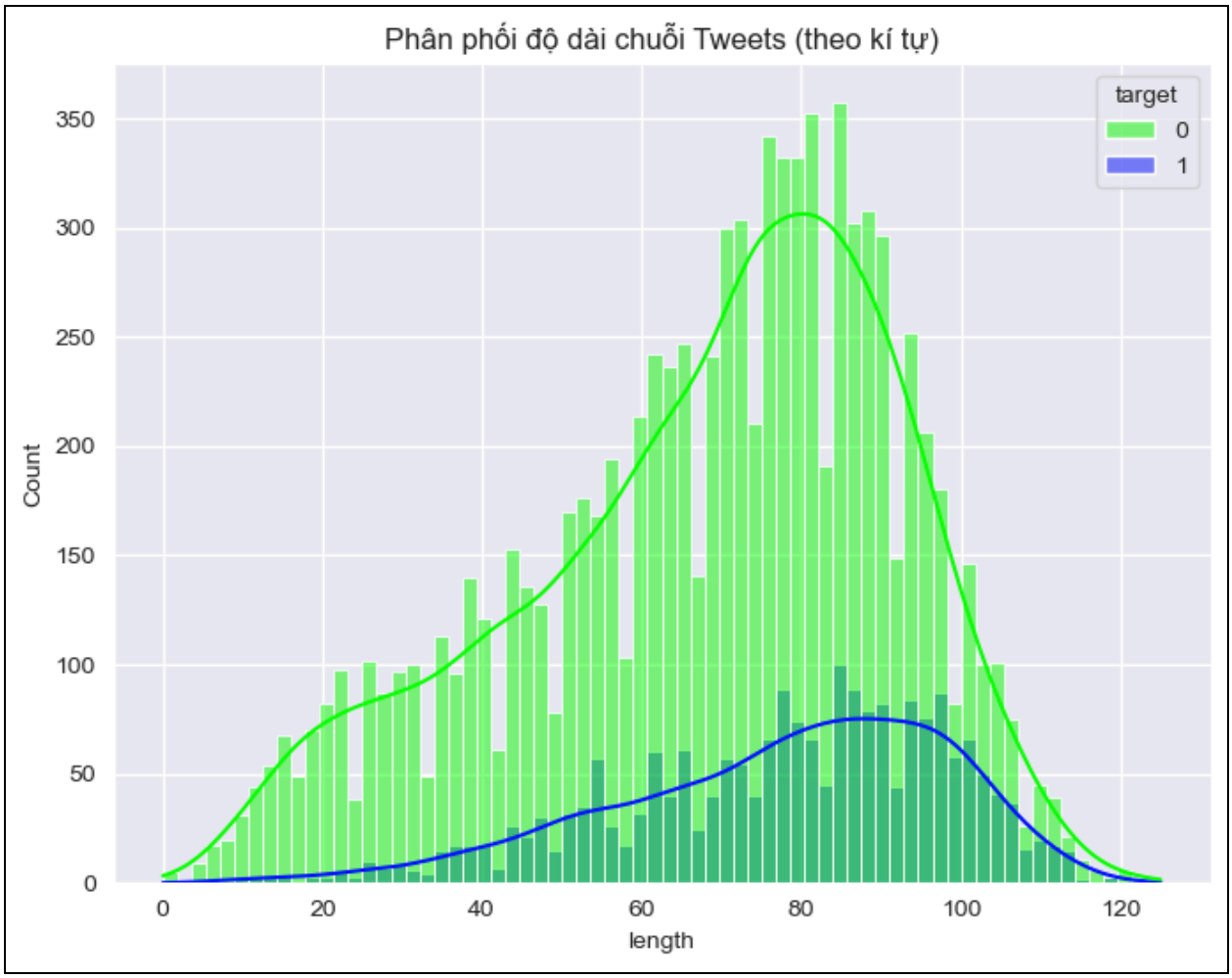
3.3. Trực quan dữ liệu

Biểu đồ thống kê số lượng label trong target:



Có thể thấy dữ liệu có sự chênh lệch rất lớn giữa các nhãn, điều này sẽ ảnh hưởng tiêu cực tới kết quả. Do đó, nhóm sẽ sử dụng các biện pháp để cân bằng dữ liệu target trước khi đưa dữ liệu vào huấn luyện mô hình.

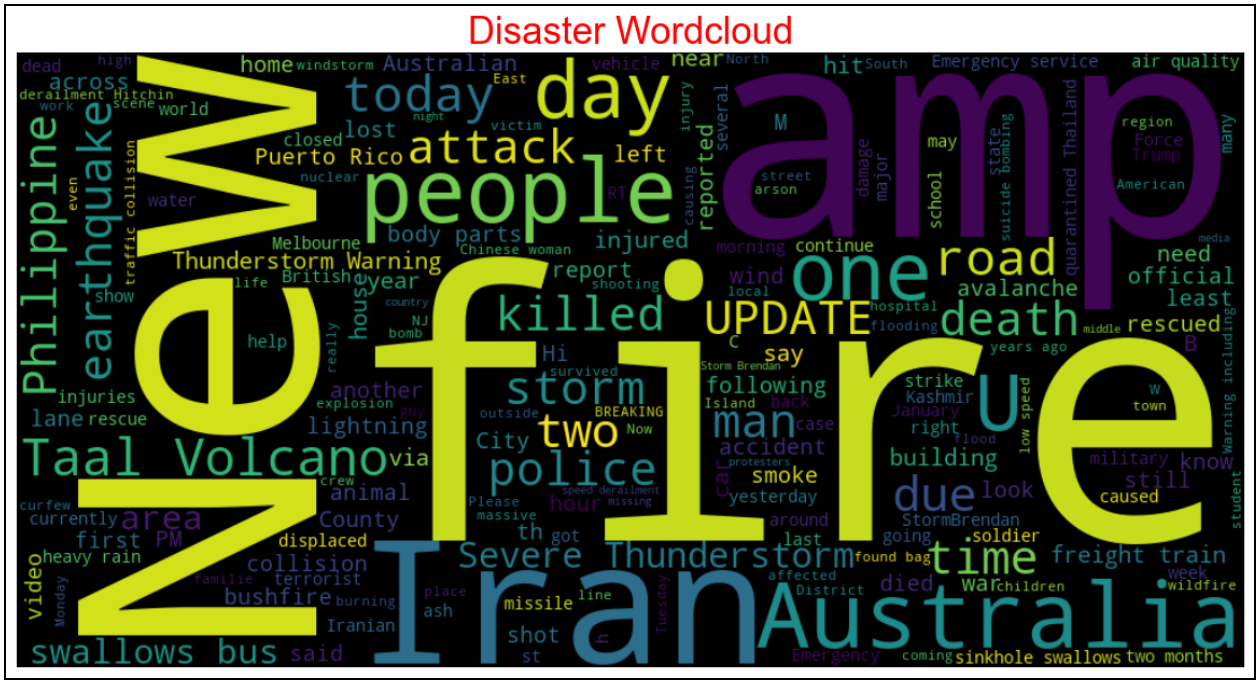
Biểu đồ phân phối độ dài các tweet:

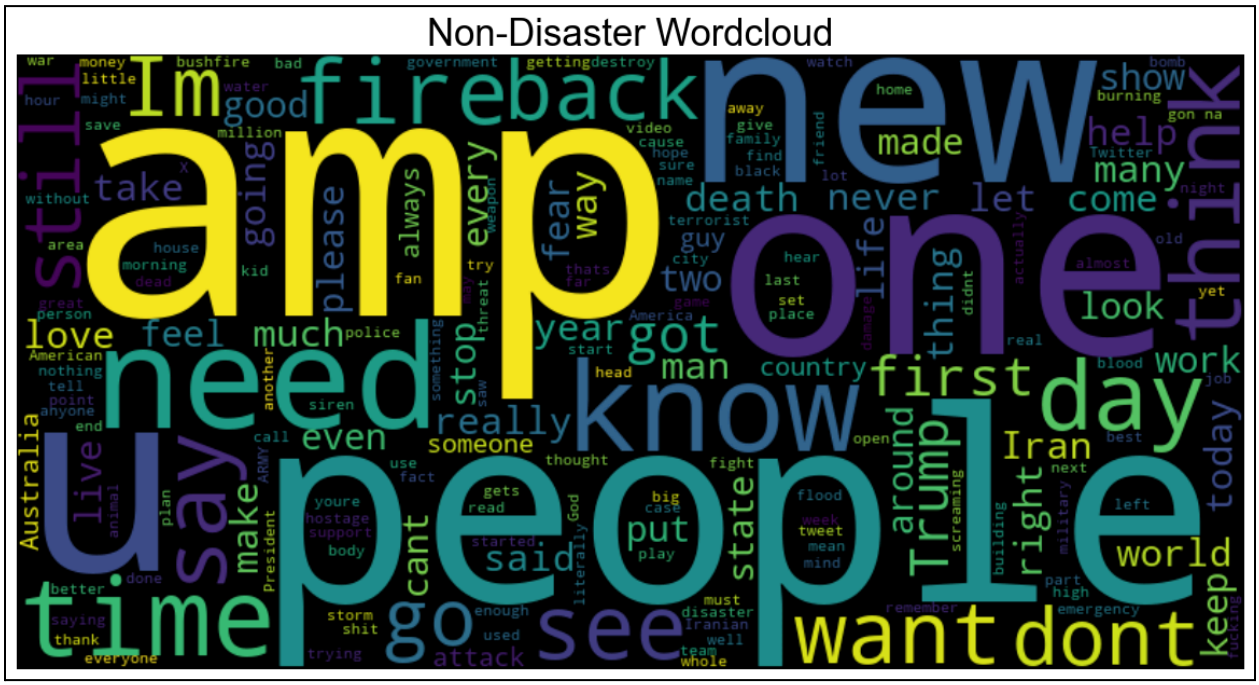


Do Twitter chỉ cho phép tối đa 140 kí tự với mỗi tweet nên giá trị độ dài sẽ phân phối từ 0 tới 140. Có thể thấy đa phần những tweet đều có độ dài trong khoảng 100 - 120 từ. Tuy nhiên, đối với những tweet được đánh dấu 'disaster' thì hầu như không xuất hiện những tweet với độ dài <40.

Biểu đồ tần suất xuất hiện của các từ trong tweet:

Wordcloud sẽ cho ta thấy tần suất xuất hiện của các từ trong văn bản. Từ xuất hiện càng nhiều thì sẽ được thể hiện càng lớn trên biểu đồ.





3.4. Xử lý mất cân bằng dữ liệu

Xử lý mất cân bằng dữ liệu là một kỹ thuật cần thiết trong học máy, nhằm giải quyết vấn đề phân bổ dữ liệu không đồng đều giữa các lớp. Mất cân bằng dữ liệu có thể dẫn đến việc mô hình học máy tập trung vào các lớp có nhiều mẫu, dẫn đến việc dự đoán kém chính xác cho các lớp có ít mẫu.

Có nhiều phương pháp xử lý mất cân bằng dữ liệu:

- Thu thập thêm dữ liệu: Đây là phương pháp hiệu quả nhất, nhưng cũng là phương pháp tốn kém và mất thời gian nhất.
- Under sampling: Phương pháp này loại bỏ một số mẫu từ các lớp có nhiều dữ liệu.
- Over sampling: Phương pháp này tạo ra thêm các mẫu từ các lớp có ít dữ liệu.
- Sử dụng các thuật toán học máy đặc biệt: Các thuật toán học máy đặc biệt được thiết kế để xử lý dữ liệu mất cân bằng, chẳng hạn như thuật toán SMOTE, ADASYN,...

Tiến hành xử lý mất cân bằng dữ liệu với RandomUnderSampler

Như cách trình bày bên trên, nhóm quyết định sử dụng kĩ thuật UnderSampling để thực hiện cân bằng dữ liệu.

Undersampling là một kỹ thuật xử lý mất cân bằng dữ liệu, nhằm giải quyết vấn đề phân bổ dữ liệu không đồng đều giữa các lớp.

Undersampling có thể được thực hiện theo nhiều cách khác nhau:

- Random undersampling: Phương pháp này loại bỏ ngẫu nhiên một số mẫu từ các lớp có nhiều dữ liệu.
- Clustered undersampling: Phương pháp này loại bỏ các cụm mẫu từ các lớp có nhiều dữ liệu.
- Edited nearest neighbors undersampling: Phương pháp này loại bỏ các mẫu từ các lớp có nhiều dữ liệu, nếu các mẫu đó có các điểm gần nhất trong các lớp có ít dữ liệu.

Như vậy, với bộ dữ liệu này nhóm sẽ sử dụng Random undersampling để loại bỏ những nhãn = 0 sao cho số lượng nhãn 0 và 1 là tương đương nhau:

Kết quả:

```
Before undersampling:
0    9256
1    2114
Name: target, dtype: int64

After undersampling:
0    2114
1    2114
Name: target, dtype: int64
```

	id	keyword	text	target
0	764	battle	1656 ABAF Battle ID I need backup Lvl 100 Colo...	0
1	1563	bombed	The place children called home bombed Assad Pu...	0
2	6749	hurricane	ordered one wonuhurricane good luck sis	0
3	4094	devastated	IM DEVASTATED	0
4	2453	collide	forgiveErrorIranVowPunishAllResponsibleDeadBoe...	0
...	...	...	...	...
4223	11338	wrecked	Kesian ular We wrecked natural habitat	1
4224	11354	wrecked	Yeah proper Liverpool fans wrecked Man City bu...	1
4225	11355	wrecked	Trump Sisi rejected foreign exploitation agree...	1
4226	11359	wrecked	Trump Sisi rejected foreign exploitation agree...	1
4227	11369	wrecked	Jake Corway wrecked running 14 th IRP	1

3.5. Véc-tơ hóa dữ liệu

TF-IDF là một trong những kỹ thuật cơ bản trong xử lý ngôn ngữ giúp đánh giá mức độ quan trọng của một từ trong văn bản.

Trong đó TF - term frequency : dùng để ước lượng tần suất xuất hiện của từ trong văn bản. Tuy nhiên với mỗi văn bản thì có độ dài khác nhau, vì thế số lần xuất hiện của từ có thể nhiều hơn

IDF- Inverse Document Frequency: dùng để ước lượng mức độ quan trọng của từ. Khi tính tần số xuất hiện (tf), các từ đều được coi là quan trọng như nhau. Tuy nhiên, có một số từ thường được sử dụng nhiều nhưng không quan trọng để thể hiện ý nghĩa của đoạn văn.

Sự kết hợp giữa tần số xuất hiện và tần suất tài liệu nghịch đảo tạo thành tần suất-tần suất tài liệu nghịch đảo (tf-idf). Tf-idf là một thước đo mạnh mẽ để đánh giá mức độ quan trọng của một từ trong một tài liệu.

```
X = df['text']
y = df['target']
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_df=0.7)
```

```
X_tfidf = tfidf_vectorizer.fit_transform(df['text'])
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

Tính độ thưa thớt của vector:

Độ thưa thớt (sparsity) của một ma trận là tỷ lệ phần trăm các phần tử có giá trị bằng không so với tổng số phần tử trong ma trận đó. Công thức:

$$\text{Sparsity} = \frac{\text{Số lượng phần tử bằng không}}{\text{Tổng số phần tử trong ma trận}}$$

```
dense_matrix = X_test_tfidf.todense()
sparsity = 1.0 - np.count_nonzero(dense_matrix) / dense_matrix.size
print(sparsity)
```

```
Output:
0.9992689547324056
```

Một ma trận với độ thưa thớt (sparsity) bằng 0.9990973731532767 là rất thưa thớt. Độ thưa thớt này có thể được hiểu là gần 99.91% các phần tử trong ma trận có giá trị bằng không.

Trong ngữ cảnh của ma trận TF-IDF, có thể nói là chỉ có khoảng 0.09% các phần tử trong ma trận chứa thông tin (có giá trị khác không). Điều này rất phổ biến trong xử lý ngôn ngữ tự nhiên (NLP), nơi ma trận TF-IDF thường chứa rất nhiều từ không xuất hiện trong mỗi tài liệu cụ thể, dẫn đến ma trận thưa thớt cao. Nên nhóm sẽ chọn những mô hình phù hợp để xử lý dữ liệu thưa thớt hiệu quả nhất.

CHƯƠNG 4: Xây dựng mô hình

4.1. Mô hình học máy - Support Vector Machines:

Cơ sở lý thuyết:

SVM, viết tắt của "Support Vector Machine", là một kỹ thuật trong lĩnh vực học máy và thống kê được sử dụng để phân loại và hồi quy. Ý tưởng cơ bản của SVM là tìm ra một mặt phẳng hoặc siêu phẳng (trong không gian nhiều chiều) phân chia dữ liệu thành hai hoặc nhiều lớp một cách rõ ràng nhất.

Cốt lõi của SVM là khái niệm về "margin", tức là khoảng cách giữa siêu phẳng quyết định (decision boundary) và những điểm dữ liệu gần nhất từ mỗi lớp. Những điểm này được gọi là "support vectors". SVM tối ưu hóa siêu phẳng quyết định bằng cách tối đa hóa margin, qua đó nâng cao khả năng phân loại của mô hình.

Trong trường hợp dữ liệu không phân tách tuyến tính, SVM sử dụng các hàm nhân (kernels) để chuyển dữ liệu sang không gian đa chiều, nơi việc phân loại tuyến tính trở nên khả thi. Các hàm nhân phổ biến bao gồm Linear Kernel, Polynomial Kernel, và RBF (Radial Basis Function) / Gaussian Kernel.

- Linear Kernel: Đây là trường hợp đơn giản nhất của SVM, khi dữ liệu có thể được phân chia bằng một đường thẳng (hoặc siêu phẳng).
- Polynomial Kernel: Sử dụng cho dữ liệu phức tạp hơn, có thể được phân loại bằng đường cong.
- RBF (Radial Basis Function) / Gaussian Kernel: Thích hợp cho hầu hết các tình huống, nhất là khi không rõ ràng về mối quan hệ tuyến tính của dữ liệu.

Công thức:

$$w \cdot x + b = 0$$

Trong đó:

- *w* là vector trọng số.
- *x* là vector đặc trưng của dữ liệu.
- *b* là hệ số điều chỉnh (bias term).

Với điều kiện phân loại:

- Nếu $w \cdot x + b \geq 1$, điểm dữ liệu được gán nhãn +1
- Nếu $w \cdot x + b \leq -1$, điểm dữ liệu được gán nhãn -1

SVM đã chứng minh hiệu quả trong nhiều lĩnh vực như nhận dạng hình ảnh, phân loại văn bản và trong các bài toán phân tích sinh học. Dù hiệu quả, SVM cũng có nhược điểm là khó hiểu và khó giải thích, đồng thời việc chọn lựa và tinh chỉnh các tham số như hàm nhân và tham số chính quy hóa có thể phức tạp.

Lý do chọn mô hình:

Việc sử dụng SVM - Support Vector Machines, cụ thể là SVC - Support Vector Classifier để dự đoán các tweets có liên quan đến thảm họa (Disaster) hoặc không (Non-disaster) là một lựa chọn tốt vì nhiều lý do. Mô hình SVC phù hợp với các tác vụ phân loại, đặc biệt là khi dữ liệu là phi tuyến tính và phức tạp, điều thường thấy trong dữ liệu văn bản như tweets.

Dưới đây là một số lý do cụ thể tại sao SVC là một lựa chọn tốt cho bài toán này:

- **Xử lý phi tuyến tính**

SVC có khả năng phân loại dữ liệu phi tuyến tính thông qua việc sử dụng các kernel khác nhau như RBF (Radial Basis Function), đa thức (polynomial), hoặc sigmoid. Kernel RBF đặc biệt hữu ích trong việc xử lý các mẫu dữ liệu phức tạp, nơi mà mối quan hệ giữa các thuộc tính không thể được mô tả bằng một đường thẳng.

- **Hiệu suất cao**

Trong thực tế, SVC thường cho kết quả tốt và có hiệu suất cao khi được cấu hình đúng cách. Việc tinh chỉnh các hyperparameter như C (độ mạnh của regularization) và gamma (đối với kernel RBF) có thể giúp mô hình tối ưu hóa biên giới quyết định giữa các lớp.

- **Xử lý dữ liệu có số chiều lớn**

Tweets thường được chuyển đổi thành dạng đặc trưng số thông qua các phương pháp vector hóa như TF-IDF hoặc Word2Vec. SVC hoạt động tốt với dữ liệu có số chiều đặc trưng cao, là điều phổ biến khi làm việc với văn bản.

- **Margin tối ưu**

SVC tìm cách tối đa hóa khoảng cách (margin) giữa các lớp, điều này không chỉ giúp phân loại dữ liệu hiện tại mà còn có thể cải thiện khả năng tổng quát hóa cho dữ liệu mới.

- **Xử lý overfitting**

Qua việc áp dụng regularization, SVC có thể ngăn chặn tình trạng overfitting - một vấn đề thường gặp khi mô hình hóa dữ liệu phức tạp như văn bản.

Vector đa chiều trong dữ liệu văn bản phản ánh sự phong phú của từ vựng: mỗi từ hoặc cụm từ được coi là một chiều trong không gian đặc trưng. Với hàng nghìn từ, không gian này trở nên rất lớn và thường thưa thớt vì mỗi tài liệu chỉ sử dụng một tập hợp nhỏ các từ có sẵn. Khi sử dụng phương pháp giảm chiều như tSNE có thể giảm số lượng chiều đặc trưng xuống còn hai hoặc ba để dễ dàng trực quan hóa. Điều này giúp hiểu rõ hơn về cấu trúc dữ liệu và các mối quan hệ giữa các điểm dữ liệu, điều không thể thấy trong không gian đa chiều.

Kernel RBF (Radial Basis Function) trong SVM là một hàm giúp biến đổi không gian đặc trưng ban đầu thành không gian mới nơi các mối quan hệ phi tuyến tính có thể được hiểu như là tuyến tính. Kernel RBF làm việc này bằng cách đo lường sự tương đồng giữa các điểm dữ liệu dựa trên khoảng cách giữa chúng và sử dụng hàm Gaussian để định lượng tương đồng đó.

RBF hiệu quả trong việc xử lý dữ liệu có nhiều chiều vì nó không yêu cầu ánh xạ rõ ràng dữ liệu vào không gian nhiều chiều. Thay vào đó, nó sử dụng thuộc tính toán học của hàm kernel để cho phép SVM hoạt động như thể dữ liệu đã được ánh xạ vào không gian đa chiều, giúp tìm ra biên phân cách mà không phải tăng độ phức tạp tính toán. Tham số γ giúp kiểm soát ảnh hưởng của từng điểm dữ liệu, cho phép mô hình tự điều chỉnh và xử lý hiệu quả dữ liệu đa chiều mà không bị quá tải bởi số lượng chiều lớn. Kernel RBF được định nghĩa như sau:

$$K(x, x') = \exp(-\gamma \|x - x'\|^2)$$

Trong đó: x, x’ là hai mẫu dữ liệu và γ là tham số điều chỉnh độ rộng của kernel.

Sau khi trực quan hóa, nếu thấy rằng dữ liệu không thể tách biệt rõ ràng trong không gian hai chiều, điều này cho thấy dữ liệu có thể phi tuyến tính. Điều này làm cho việc sử dụng kernel RBF trong SVC trở nên hợp lý, bởi kernel RBF có khả năng xử lý dữ liệu phi tuyến tính tốt, tạo ra một biên phân cách mà có thể tối ưu hóa việc phân loại trong không gian đa chiều ban đầu.

Áp dụng:

```
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X_tfidf.toarray())

plt.figure(figsize=(12, 8))
scatter = plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=df['target'], alpha=0.5,
cmap='viridis')
plt.title('t-SNE visualization of tweet data')
plt.xlabel('t-SNE feature 1')
```

```
plt.ylabel('t-SNE feature 2')
plt.colorbar(scatter, label='Disaster (1) vs. Non-disaster (0)')
plt.show()
```

Ta thấy rằng ở đây, khi biểu diễn theo không gian 2 chiều thì hai cụm dữ liệu theo target (1 hay 0) bị trùng lắp lên nhau khá nhiều nên một đường tuyến tuyến để chia cách hai cụm dữ liệu có thể dẫn đến việc sai số của phân lớp (misclassifications) khá nhiều nên nhóm chọn kernel RBF để làm tham số cho Mô hình SVC và sau đó thực hiện Gridsearch để tìm ra mô hình tốt nhất để dự đoán:

```
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [1, 0.1, 0.01, 0.001],
}

grid_search = GridSearchCV(SVC(kernel='rbf', random_state=42), param_grid,
```

```
cv=5, verbose=2, n_jobs=-1)
grid_search.fit(X_train_tfidf, y_train)

best_svm_params = grid_search.best_params_
best_svm_score = grid_search.best_score_
best_svm_classifier = grid_search.best_estimator_
```

Chọn tham số C regularization và các tham số gamma để tìm ra estimator tốt nhất sau đó dự đoán.

**** Giải thích về các tham số trong Cross Validation:***

Tham số C trong SVM là một tham số regularization, mà đóng vai trò như một "trade-off" giữa việc tìm một siêu phẳng quyết định có margin lớn nhất và việc giảm thiểu số lượng lỗi phân loại trên tập huấn luyện. Có hai điều chính mà C cân nhắc:

- Khi C nhỏ: Mô hình tăng cường margin nhưng có thể chấp nhận mức độ sai lệch cao hơn (high bias, low variance). Điều này có thể dẫn đến underfitting nếu C quá nhỏ.
- Khi C lớn: Mô hình cố gắng phân loại tất cả các mẫu huấn luyện một cách chính xác, ngay cả những điểm nằm gần ranh giới quyết định, có thể dẫn đến overfitting (low bias, high variance) nếu C quá lớn.

Tham số gamma chỉ có ý nghĩa khi sử dụng kernel như Radial Basis Function (RBF). Gamma xác định mức độ ảnh hưởng của một mẫu đơn lẻ; nói cách khác, nó xác định tầm xa của sự ảnh hưởng của một điểm huấn luyện đến quyết định phân loại.

- Khi gamma cao: Các điểm huấn luyện có ảnh hưởng lớn, dẫn đến ranh giới quyết định có độ cong cao, theo sát các điểm dữ liệu. Điều này có thể gây ra overfitting.
- Khi gamma thấp: Các điểm huấn luyện có ảnh hưởng rộng, dẫn đến ranh giới quyết định mượt mà hơn và ít có khả năng bị overfitting, nhưng có thể underfitting nếu gamma quá thấp.

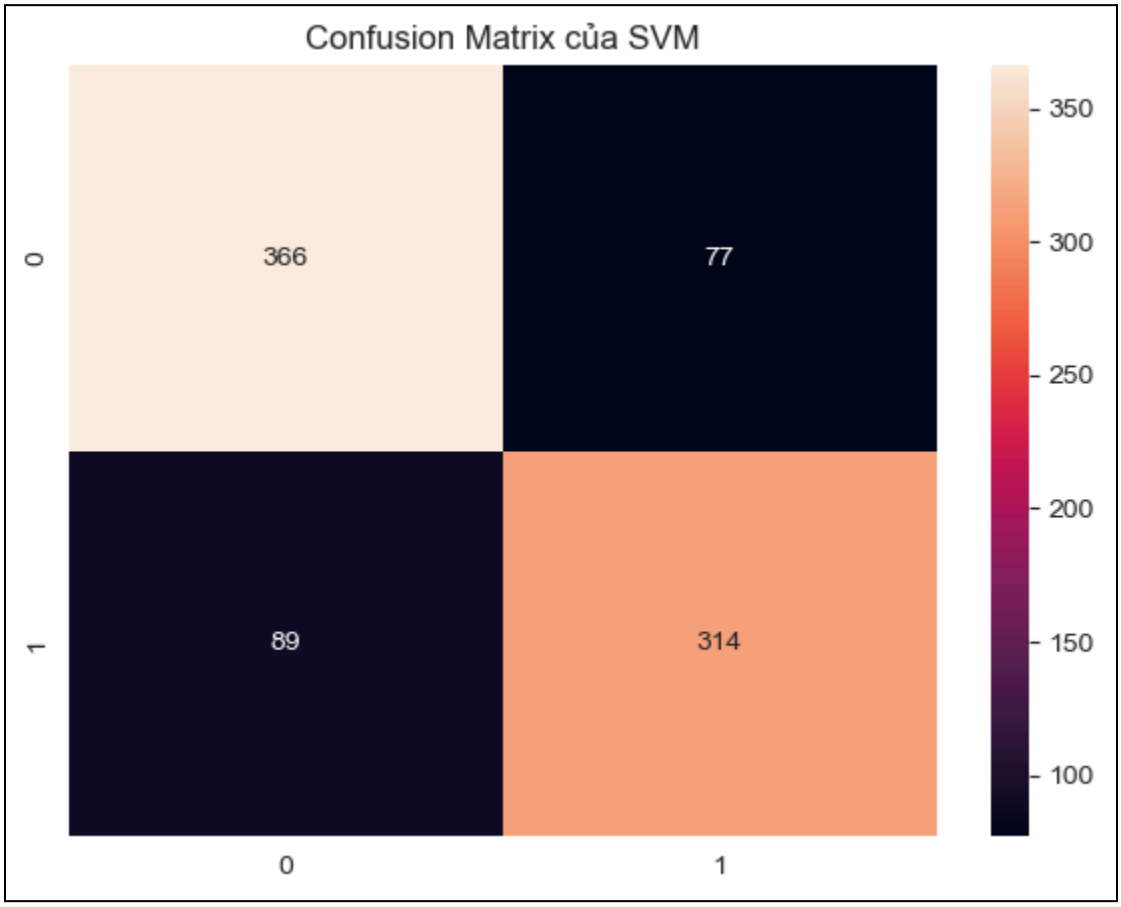
Vì vậy, việc chọn C và Gamma là một phần quan trọng của quá trình tinh chỉnh mô hình, và nó thường được thực hiện thông qua cross-validation như ở trên.

Sau đó nhận được kết quả dự đoán như sau:

```
Điểm chính xác của SVC: 0.8049645390070922

Report:               precision    recall  f1-score   support

           0           0.81        0.83    0.82       443
```

1	0.80	0.78	0.79	403

Confusion Matrix:



4.2. Mô hình Maxent - Logistic Regression:

Cơ sở Lý Thuyết:

Logistic Regression là một mô hình phân loại phổ biến trong học máy, đặc biệt hiệu quả trong các tác vụ phân loại nhị phân, nó có mối liên hệ sâu sắc với nguyên lý Maximum Entropy trong lý thuyết thông tin. Nguyên lý này khuyến khích việc chọn lựa một mô hình xác suất sao cho nó mang lại thông tin lớn nhất có thể (tức là có entropy cao nhất), miễn là mô hình vẫn tuân thủ các ràng buộc từ dữ liệu quan sát. Trong trường hợp của Logistic Regression, mô hình này cố gắng tối đa hóa likelihood của dữ liệu huấn luyện, điều này tương đương với việc tối đa hóa entropy của phân phối xác suất dự đoán, đồng thời đảm bảo rằng mô hình phù hợp chặt chẽ với mẫu dữ liệu.

Qua quá trình huấn luyện, Logistic Regression sử dụng hàm log-loss để tìm ra các tham số mô hình tối ưu, tạo ra một mô hình có khả năng đưa ra dự đoán chính xác mà không thêm giả định không cần thiết vào dữ liệu, phản ánh tinh thần của Maximum Entropy. Mô hình này sử dụng

hàm logistic, còn được biết đến là hàm sigmoid, để chuyển đổi tổng trọng số của các đặc trưng (features) thành một xác suất nằm trong khoảng từ 0 đến 1. Công thức:

$$p = \frac{1}{1+e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n)}}$$

Với: xác suất (*p*) mà một điểm dữ liệu thuộc về một lớp nhất định (thường là lớp "1") và

- *β0, β1, ..., βn* là các hệ số mô hình.
- *x1, x2, ..., xn* là các đặc trưng của điểm dữ liệu .

Trong quá trình huấn luyện, mô hình học các giá trị của các hệ số này sao cho phù hợp nhất với dữ liệu. Để phân loại, một ngưỡng thường được thiết lập (thường là 0.5) để quyết định một điểm dữ liệu thuộc lớp nào dựa trên xác suất (*p)* Logistic Regression còn được tích hợp các kỹ thuật regularization như L1 (Lasso), L2 (Ridge), hoặc Elastic Net để ngăn chặn hiện tượng overfitting, nơi mô hình quá khớp với dữ liệu huấn luyện và mất đi khả năng tổng quát hóa. Hyperparameter C trong scikit-learn điều chỉnh sức mạnh của regularization, với giá trị lớn hơn chỉ ra regularization yếu hơn.

Ngoài ra, việc lựa chọn solver phù hợp (ví dụ: liblinear, lbfgs, saga) cũng quan trọng, tùy thuộc vào đặc điểm của dữ liệu và loại regularization được sử dụng.

Dưới đây là giải thích về ba solver thường được sử dụng: liblinear, lbfgs, và saga:

- **liblinear**:
    - *liblinear* là một solver hiệu quả cho các tập dữ liệu nhỏ và trung bình.
    - Nó hỗ trợ cả L1 và L2 regularization, nhưng không hỗ trợ none hoặc elasticnet.
    - Solver này thực hiện tối ưu hóa dựa trên thuật toán coordinate descent và thường là lựa chọn tốt khi làm việc với các bài toán phân loại nhị phân.
- **lbfgs**:
    - *lbfgs* là một solver dựa trên phương pháp tối ưu hóa quasi-Newton.
    - Nó hỗ trợ L2 regularization hoặc không regularization (none), nhưng không hỗ trợ L1.
    - Solver này thích hợp cho các tập dữ liệu vừa và lớn và thường được sử dụng làm giải pháp mặc định trong scikit-learn vì nó cân bằng giữa hiệu suất và tốc độ tính toán.
- **saga**:

- *saga* là phiên bản cải tiến của solver *sag* và hỗ trợ cả L1, L2, và *elasticnet* regularization.
- Đây là solver lựa chọn cho các tập dữ liệu lớn và/hoặc dữ liệu thưa thớt.
- *saga* thực hiện tối ưu hóa bằng cách sử dụng stochastic average gradient descent, rất hữu ích cho các tình huống cần sự linh hoạt về loại regularization và xử lý hiệu quả với dữ liệu thưa.

Lý do nhóm chọn Logistic Regression cho phân loại lớp nhị phân trong NLP, cụ thể là bài toán phân loại văn bản:

- **Mô hình tuyến tính hiệu quả**

Logistic Regression là một phương pháp phân loại linh hoạt, mà ở đó sự lựa chọn của lớp được xác định thông qua một mối quan hệ tuyến tính giữa các đặc trưng đầu vào và xác suất của kết quả đầu ra. Trong ngữ cảnh xử lý ngôn ngữ tự nhiên (NLP), mô hình này hoạt động dựa trên giả định rằng có một mối liên kết tuyến tính giữa tần suất của từ và ngữ cảnh mà chúng xuất hiện với khả năng một đoạn văn bản thuộc về một nhãn cụ thể.

Tuy nhiên, mô hình logistic regression không cần thiết phải có dữ liệu hoàn hảo tuyến tính để hoạt động hiệu quả. Điểm mạnh của nó nằm ở khả năng phát hiện và sử dụng một ranh giới quyết định - thường được biểu diễn thông qua một đường cong logistic - để tách biệt các lớp. Điều này cho phép mô hình phân loại dữ liệu một cách chính xác ngay cả khi mối quan hệ giữa các đặc trưng và lớp không hoàn toàn tuyến tính, nhưng vẫn có thể được mô tả thông qua đường cong logistic phản ánh xác suất của từng lớp.

- **Xử lý tốt dữ liệu thưa thớt**

Trong NLP, đặc biệt khi sử dụng biểu diễn như bag-of-words hoặc TF-IDF, dữ liệu thường rất thưa (sparse) (như trong dữ liệu đã giải thích). Logistic Regression hoạt động tốt vì nó chỉ tập trung vào những đặc trưng (từ/cụm từ) có trọng số cao, mà bỏ qua những đặc trưng không mang thông tin (có trọng số bằng không hoặc rất thấp).

- **Xác suất dự đoán**

Logistic Regression cung cấp kết quả dưới dạng xác suất, cho phép đánh giá mức độ chắc chắn của mô hình đối với mỗi phân loại. Điều này hữu ích khi cần đánh giá rủi ro hoặc khi cần sự tin cậy trong quyết định.

- **Điều chỉnh dữ liệu không cân đối**

Trong thực tế, dữ liệu trong bài toán xử lý ngôn ngữ tự nhiên thường không cân đối (ví dụ, số lượng văn bản về thảm họa ít hơn nhiều so với văn bản không phải thảm họa). Logistic Regression cho phép điều chỉnh trọng số lớp hoặc tham số regularization để xử lý vấn đề này. (

nhóm đã undersample để cân đối lại dữ liệu như trên nên không cần thêm bước hiệu chỉnh tham số để cân bằng lại mô hình)

- **Hiệu suất tốt trong nhiều trường hợp**

Trong nhiều trường hợp, Logistic Regression cho hiệu suất phân loại tốt, cân nhắc giữa độ chính xác và thời gian huấn luyện, đặc biệt trong các ứng dụng NLP có kích thước dữ liệu vừa phải.

Áp dụng:

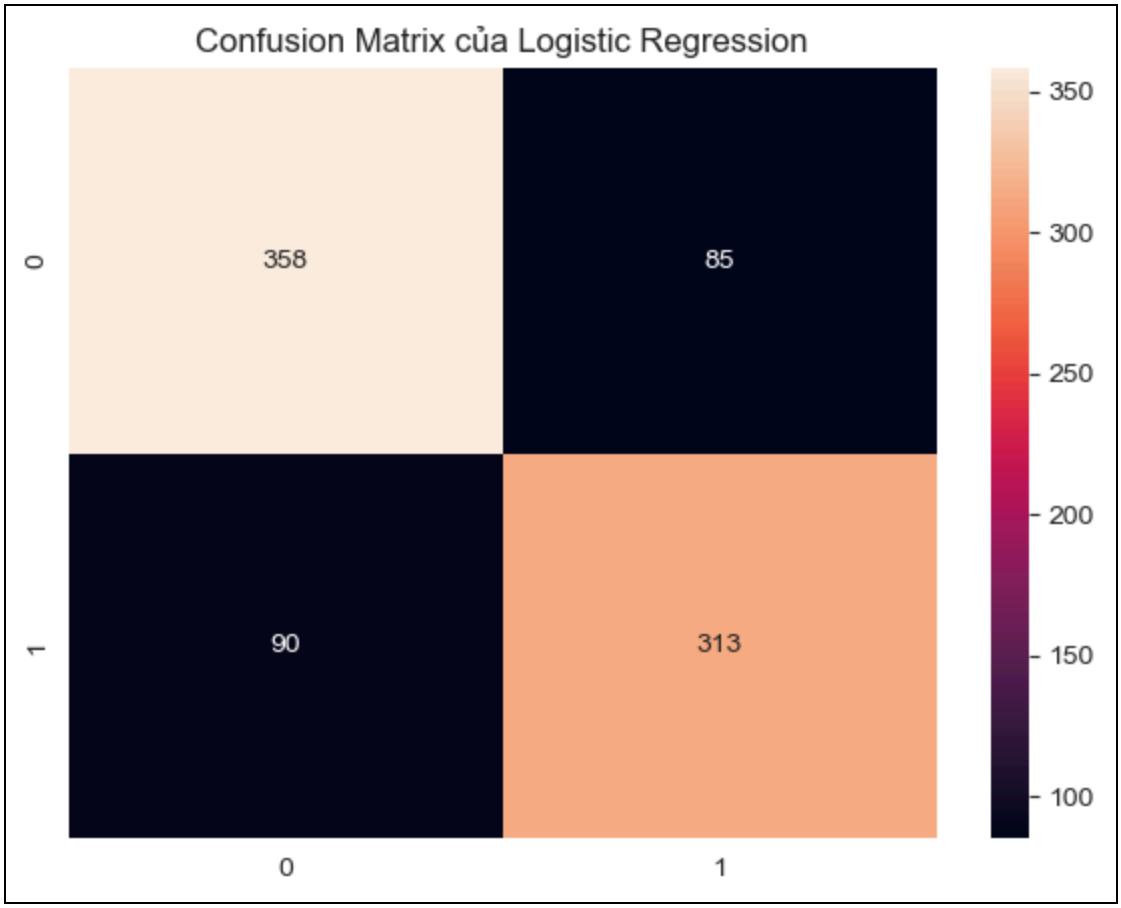
```
param_grid = {
    'C': [0.1, 1, 10, 100],
    'solver': ['liblinear', 'lbfgs', 'saga'],
}

grid_search = GridSearchCV(LogisticRegression(random_state=42), param_grid,
cv=5, verbose=2, n_jobs=-1)

grid_search.fit(X_train_tfidf, y_train)

best_lr_classifier = grid_search.best_estimator_
best_lr_params = grid_search.best_params_
best_lr_score = grid_search.best_score_
```

Chọn các mức C regularization tương tự như ở SVM và các solver để tìm ra estimator tốt nhất và sau đó dự đoán

Kết quả dự đoán của mô hình Logistic Regression:

```
Độ chính xác của Logistic Regression: 0.793144208037825

 Report:               precision    recall  f1-score   support

           0           0.80         0.81    0.80        443
           1           0.79         0.78    0.78        403
```

Confusion Matrix:



4.3. Mô hình học sâu - Long Short-Term Memory

LSTM, hay Long Short-Term Memory, là một loại thuật toán trong lĩnh vực học máy và mạng nơ-ron, được thiết kế để giải quyết vấn đề của việc "quên" thông tin trong quá trình xử lý dữ liệu chuỗi thời gian.

Thuật toán này là một dạng đặc biệt của mạng nơ-ron hồi quy (RNN). Điểm đặc biệt của LSTM là khả năng duy trì và điều chỉnh thông tin trong thời gian dài mà không bị ảnh hưởng quá mức bởi hiện tại hoặc quá khứ. Điều này giúp LSTM giải quyết vấn đề "gradient vanishing" gặp phải trong mô hình RNN thông thường khi xử lý dữ liệu chuỗi dài.

LSTM sử dụng các "cổng" để kiểm soát lưu thông thông tin, bao gồm cổng quên để quyết định nên giữ hay quên thông tin cũ, cổng đầu vào để quyết định thông tin mới nào sẽ được thêm vào, và cổng đầu ra để tạo ra đầu ra của mô hình. Nhờ vào cơ chế này, LSTM thường hiệu quả trong việc mô hình hóa các chuỗi dữ liệu thời gian phức tạp.

4.3.1. Các bước xử lý văn bản trước khi áp dụng thuật toán LSTM

- Phân tách văn bản (Tokenization):

Phân tách văn bản thành các đơn vị nhỏ hơn gọi là token và xây dựng ma trận từ của văn bản. Một văn bản có thể được coi như một "túi đựng từ". Tập hợp nhiều văn bản được gọi là corpus. Trong ma trận từ của văn bản:

1. Mỗi hàng đại diện cho một văn bản (túi từ)
2. Mỗi cột đại diện cho một token riêng biệt
3. Mỗi ô đại diện cho tần suất xuất hiện của token đó.

- Cân bằng chiều dài câu (Padding):

Trong khi xử lý văn bản bằng mạng nơ-ron, chúng ta cần đảm bảo tất cả các đầu vào có cùng kích thước chiều dài. Điều này là do mạng nơ-ron mong đợi dữ liệu có cấu trúc đồng nhất để thực hiện các phép tính hiệu quả. Tuy nhiên, trong thực tế, các câu trong văn bản thường có độ dài khác nhau.

Ví dụ:

Câu 1: "Tôi yêu Việt Nam" (chiều dài 4 từ)

Câu 2: "Hôm nay trời rất đẹp" (chiều dài 5 từ)

Sau khi padding, cả hai câu sẽ có cùng chiều dài 6 từ:

Câu 1 (cùng độ dài): "0 0 0 Tôi yêu Việt Nam"

Câu 2 (được padding): "0 Hôm nay trời rất đẹp"

Bằng cách này, các câu có độ dài khác nhau được chuẩn hóa về cùng một kích thước, cho phép mạng nơ-ron xử lý chúng một cách thống nhất và hiệu quả hơn.

Áp dụng Padding & Tokenization với bộ dữ liệu ‘Tweet Disasters’:

```
max_features=5000
tokenizer=Tokenizer(num_words=max_features,split=' ')
tokenizer.fit_on_texts(df['text'].values)
X = tokenizer.texts_to_sequences(df['text'].values)
X = pad_sequences(X)
```

```
[178] X.shape

(11370, 31)
```

```
[198] X[0]

array([   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0, 4208,  778,    5, 1644, 1728, 2120,   66, 4209,   12, 1153,
        740,    6,   78,  740,    6, 1326,   66,  267,  958], dtype=int32)
```

```
[199] tokenizer.sequences_to_texts([[  4208,  778,    5, 1644, 1728, 2120,   66, 4209,   12, 1153,
            740,    6,   78,  740,    6, 1326,   66,  267,  958]])

['communal violence in bhainsa telangana stones were pelted on muslims houses and some houses and vehicles were set ablaze']
```

4.3.2. LSTM Model

- Tầng nhúng (Embedding layer)

Là tầng đầu tiên của mạng nơ-ron và có 3 tham số:

1. input_dim: Số lượng vector token riêng biệt, với bài toán này nhóm chọn là 5000
2. output_dim: Kích thước của vector nhúng, nhóm chọn 32 chiều.
3. input_length: Kích thước của tầng đầu vào.

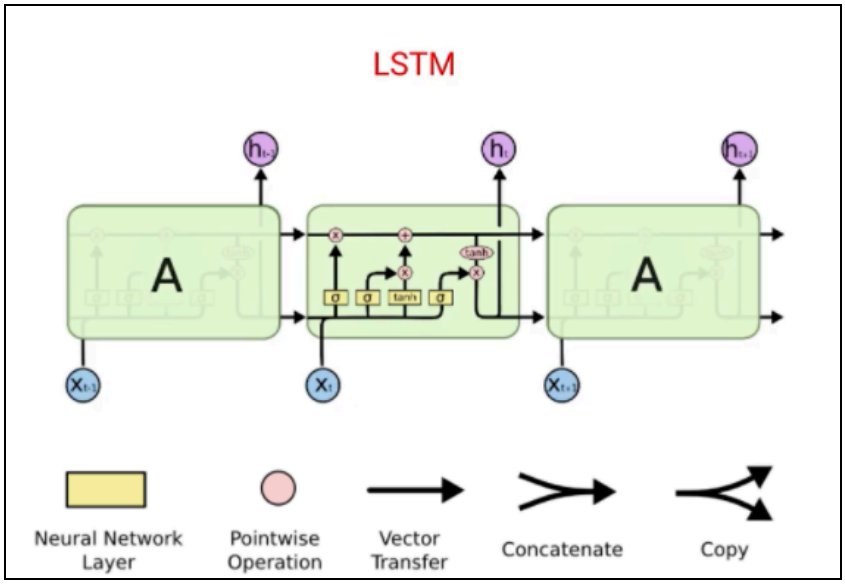
Như vậy, kích thước của tầng nhúng sẽ là (5000, 32).

Nói một cách đơn giản hơn:

1. Tầng nhúng chuyển đổi các từ riêng biệt thành các vector có kích thước thấp hơn (32 trong trường hợp này).
2. Số lượng vector riêng biệt là 5000 (words).
3. Tầng này là tầng đầu tiên trong mạng nơ-ron và thực hiện việc "nhúng" các từ thành các vector để các tầng tiếp theo dễ dàng xử lý hơn.

- LSTM

LSTM là một mạng nơ-ron hồi quy nhân tạo được thiết kế đặc biệt để tránh vấn đề phụ thuộc chuỗi dài. LSTM có 4 lớp mạng nơ-ron.



Hình ảnh minh họa 4 lớp mạng nơ-ron. 3 trong số đó sử dụng hàm kích hoạt sigmoid, cho kết quả 0 hoặc 1. Một lớp sử dụng hàm kích hoạt tanh, cho kết quả từ -1 đến 1.

LSTM có 3 cổng chính:

- Cổng forget: Quyết định thông tin nào từ trước đó không còn liên quan và có thể quên đi. Bao gồm lớp mạng nơ-ron đầu tiên với hàm kích hoạt sigmoid.
- Cổng input: Học thông tin mới từ đầu vào. Bao gồm 2 lớp mạng nơ-ron: Lớp đầu tiên với hàm sigmoid để quyết định giá trị nào cập nhật. Lớp thứ hai với hàm tanh để tạo ra một vector giá trị mới.
- Cổng output: Truyền thông tin từ thời điểm hiện tại đến thời điểm tiếp theo. Sử dụng hàm kích hoạt sigmoid.

Chi tiết cổng input: Trong cổng input, lớp sigmoid quyết định phần nào của trạng thái ô sẽ được truyền ra. Sau đó, trạng thái ô được truyền qua hàm tanh và nhân với kết quả của cổng sigmoid. Điều này đảm bảo chỉ những phần được chọn lọc mới được truyền ra.

Tiến hành xây dựng mô hình LSTM với bộ dữ liệu Tweet Disasters

Mô hình LSTM nhóm xây dựng bao gồm:

Hàm kích hoạt của lớp Dense, tức là output, được chọn là sigmoid. Lý do là vì:

- Sigmoid phù hợp với phân loại nhị phân (binary classification).
- Biến mục tiêu của bạn chỉ có hai giá trị 0 hoặc 1.

Dropout:

- Dropout được thêm vào để ngăn ngừa overfitting.

Hàm mất mát:

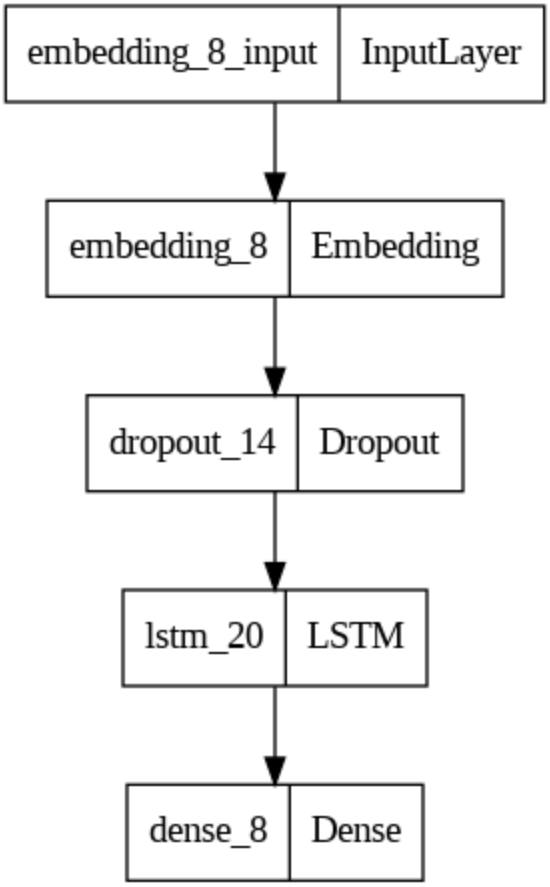
- Hàm mất mát cross-entropy được sử dụng để đánh giá mức độ sai lệch giữa dự đoán của mô hình và thực tế.
- Hàm binary_crossentropy cụ thể được dùng cho trường hợp phân loại nhị phân.

Trình tối ưu:

- Trình tối ưu Adam được sử dụng để thay thế cho thuật toán gradient descent ngẫu nhiên trong quá trình huấn luyện mô hình trong trường hợp này bạn đã khởi tạo nó thành 0.002.

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_8 (Embedding)     (None, 31, 32)            160000

 dropout_14 (Dropout)        (None, 31, 32)            0

 lstm_20 (LSTM)              (None, 32)                8320

 dense_8 (Dense)             (None, 1)                 33

=================================================================
Total params: 168353 (657.63 KB)
Trainable params: 168353 (657.63 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

Chuỗi LSTM Dự đoán nhị phân bộ dữ liệu Tweet Disaster

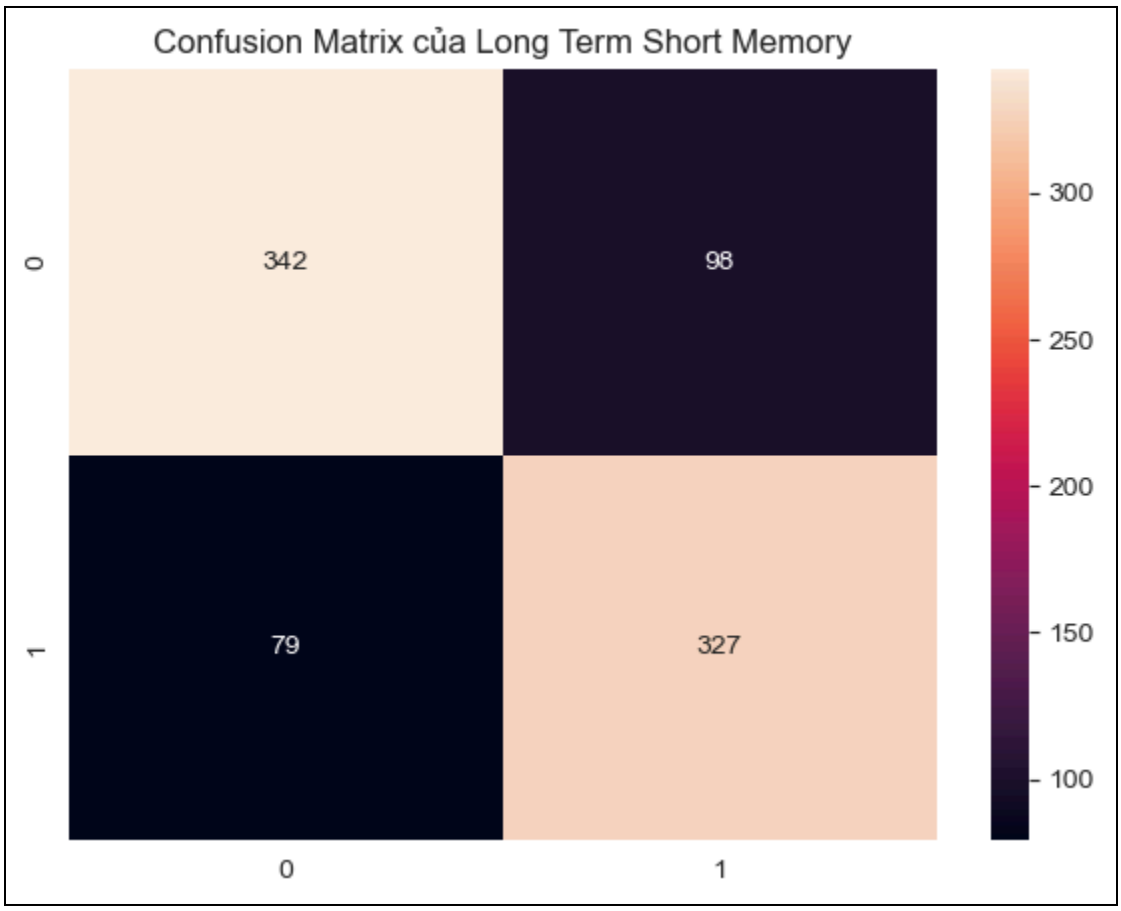


Kết quả:

```
Epoch 1/10
285/285 [==============================] - 19s 43ms/step - loss: 0.3768 - accuracy: 0.8531 - val_loss: 0.3041 - val_accuracy: 0.8857
Epoch 2/10
285/285 [==============================] - 11s 39ms/step - loss: 0.2261 - accuracy: 0.9169 - val_loss: 0.3208 - val_accuracy: 0.8940
Epoch 3/10
285/285 [==============================] - 11s 39ms/step - loss: 0.1712 - accuracy: 0.9384 - val_loss: 0.3127 - val_accuracy: 0.8843
Epoch 4/10
285/285 [==============================] - 12s 41ms/step - loss: 0.1389 - accuracy: 0.9490 - val_loss: 0.3407 - val_accuracy: 0.8738
Epoch 5/10
285/285 [==============================] - 11s 39ms/step - loss: 0.1106 - accuracy: 0.9603 - val_loss: 0.3464 - val_accuracy: 0.8826
Epoch 6/10
285/285 [==============================] - 10s 35ms/step - loss: 0.0920 - accuracy: 0.9675 - val_loss: 0.4179 - val_accuracy: 0.8861
Epoch 7/10
285/285 [==============================] - 12s 44ms/step - loss: 0.0731 - accuracy: 0.9725 - val_loss: 0.4327 - val_accuracy: 0.8843
Epoch 8/10
285/285 [==============================] - 12s 41ms/step - loss: 0.0556 - accuracy: 0.9811 - val_loss: 0.5503 - val_accuracy: 0.8703
Epoch 9/10
285/285 [==============================] - 12s 41ms/step - loss: 0.0478 - accuracy: 0.9828 - val_loss: 0.5761 - val_accuracy: 0.8795
Epoch 10/10
285/285 [==============================] - 12s 41ms/step - loss: 0.0507 - accuracy: 0.9820 - val_loss: 0.5482 - val_accuracy: 0.8795
```

```
Điểm chính xác: 0.8226950354609929

 Report:               precision    recall    f1-score   support

           0               0.81       0.87      0.84         440
           1               0.84       0.78      0.81         406
```

Confusion Matrix:



CHƯƠNG 5: Đánh giá & kết luận

Mô hình	Accuracy	Precision	Recall	F1 Score
SVM	0.80	0.80	0.81	0.80
Log Regression	0.79	0.80	0.79	0.80
LSTM	0.82	0.82	0.825	0.82

Từ bảng dữ liệu, có thể thấy LSTM là mô hình có hiệu suất tốt nhất, với các chỉ số Accuracy, Precision, Recall và F1 Score cao nhất.

Cụ thể, LSTM có Accuracy là 0.82, Precision là 0.82, Recall là 0.825 và F1 Score là 0.82. Nhận xét chi tiết Accuracy là chỉ số đánh giá khả năng phân loại chính xác của mô hình.

Kết quả Accuracy của cả ba mô hình đều cao, trên 0.79, cho thấy các mô hình đều có khả năng phân loại chính xác tương đối tốt. Tuy nhiên, Accuracy của LSTM là cao nhất, cho thấy LSTM có khả năng phân loại chính xác tốt hơn SVM và Logistic Regression.

Precision là chỉ số đánh giá khả năng phân loại đúng các trường hợp dương. Precision của cả ba mô hình đều cao, trên 0.8, cho thấy các mô hình đều có khả năng phân loại đúng các trường hợp dương tương đối tốt.

Tương tự, chỉ số Recall của cả ba mô hình đều cao, trên 0.79, cho thấy các mô hình đều có khả năng phân loại đúng các trường hợp dương thực tế tương đối tốt. T

F1 Score là chỉ số đánh giá khả năng phân loại tổng thể của mô hình. F1 Score của cả ba mô hình đều cao, trên 0.8, cho thấy các mô hình đều có khả năng phân loại tổng thể tương đối tốt. Tuy nhiên, F1 Score của LSTM là cao nhất, cho thấy LSTM có khả năng phân loại tổng thể tốt hơn SVM và Logistic Regression.

Kết luận: Từ các nhận xét trên, có thể thấy LSTM là mô hình có hiệu suất tốt nhất trong các mô hình được so sánh. LSTM có khả năng phân loại chính xác, phân loại đúng các trường hợp dương và phân loại tổng thể tốt hơn SVM và Logistic Regression.

CHƯƠNG 6: Xây dựng ứng dụng demo

6.1. Mô tả về ứng dụng

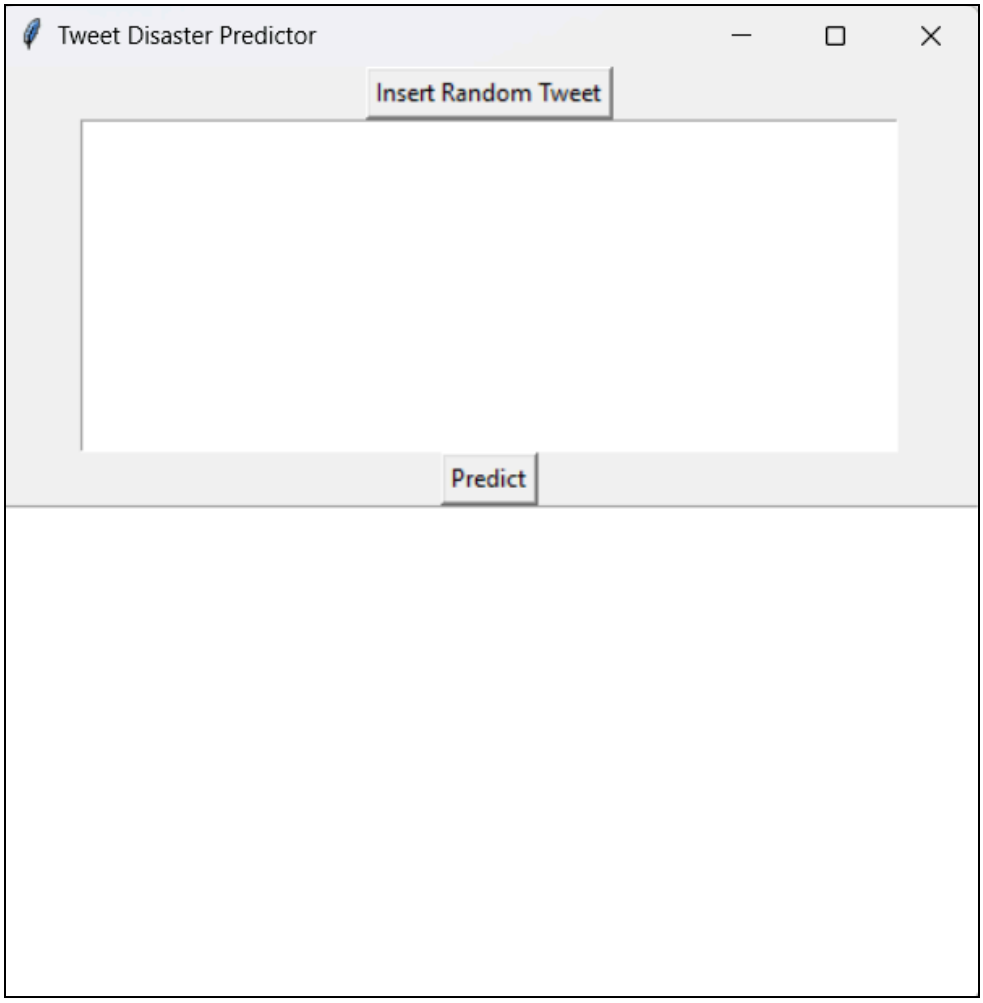
Ứng dụng có tên là '**Tweet Disaster Predictor'** và nó được làm ra với mục đích dự đoán Tweet có phải là một Tweet thảm họa hay không với kết quả của cả ba mô hình và sau đó đưa ra kết luận nếu 2/3 mô hình dự đoán vượt trội hơn.

Input: là một chuỗi được nhập vào từ người dùng

Output:

- Cleaned tweet: là một chuỗi được đã xử lý bằng các hàm đã xây dựng
- Kết quả dự đoán của cả ba mô hình
- Kết quả dự đoán cuối cùng (dựa vào kết quả dự đoán của 2/3 mô hình)

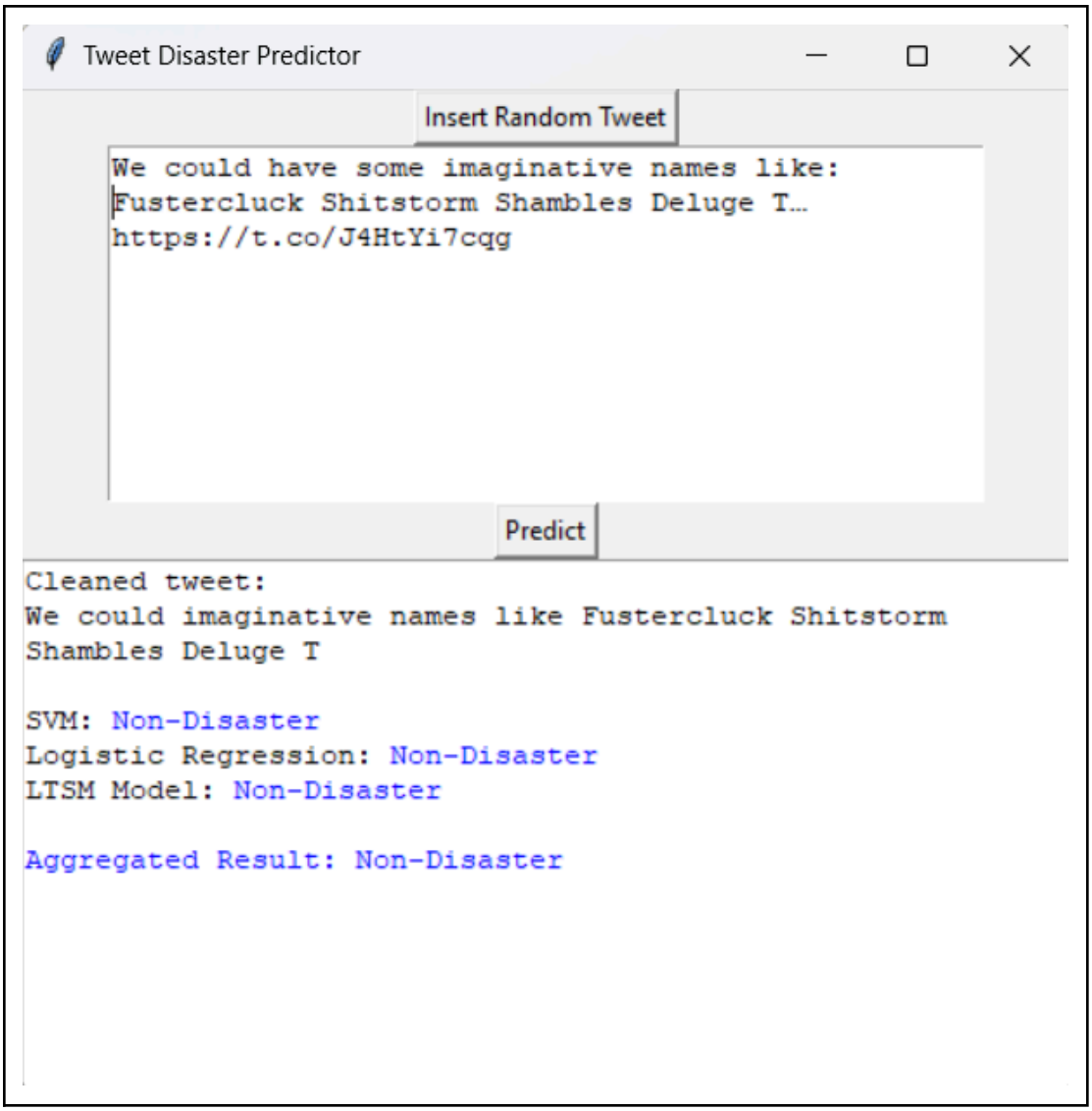
6.2. Giao diện

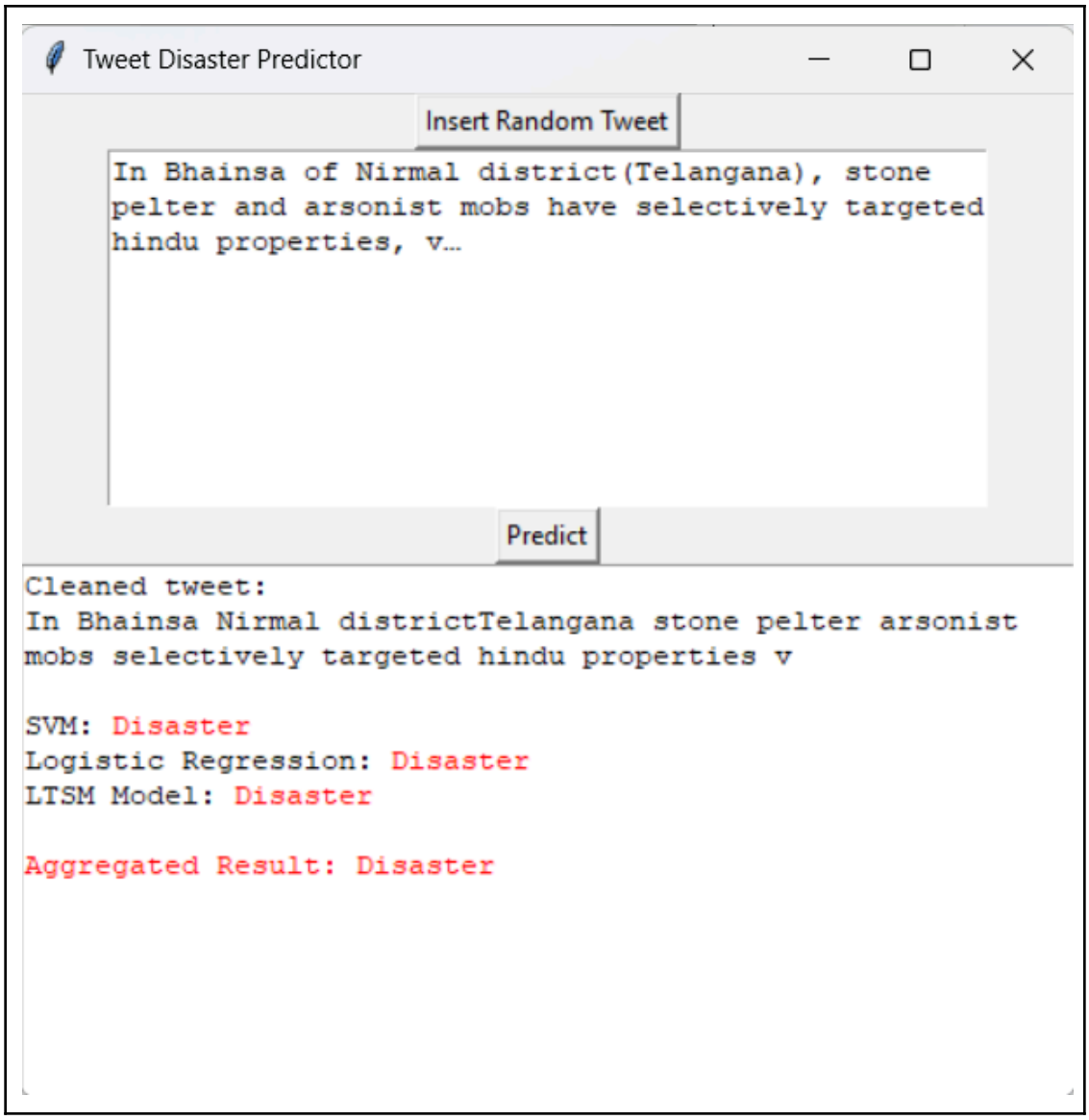


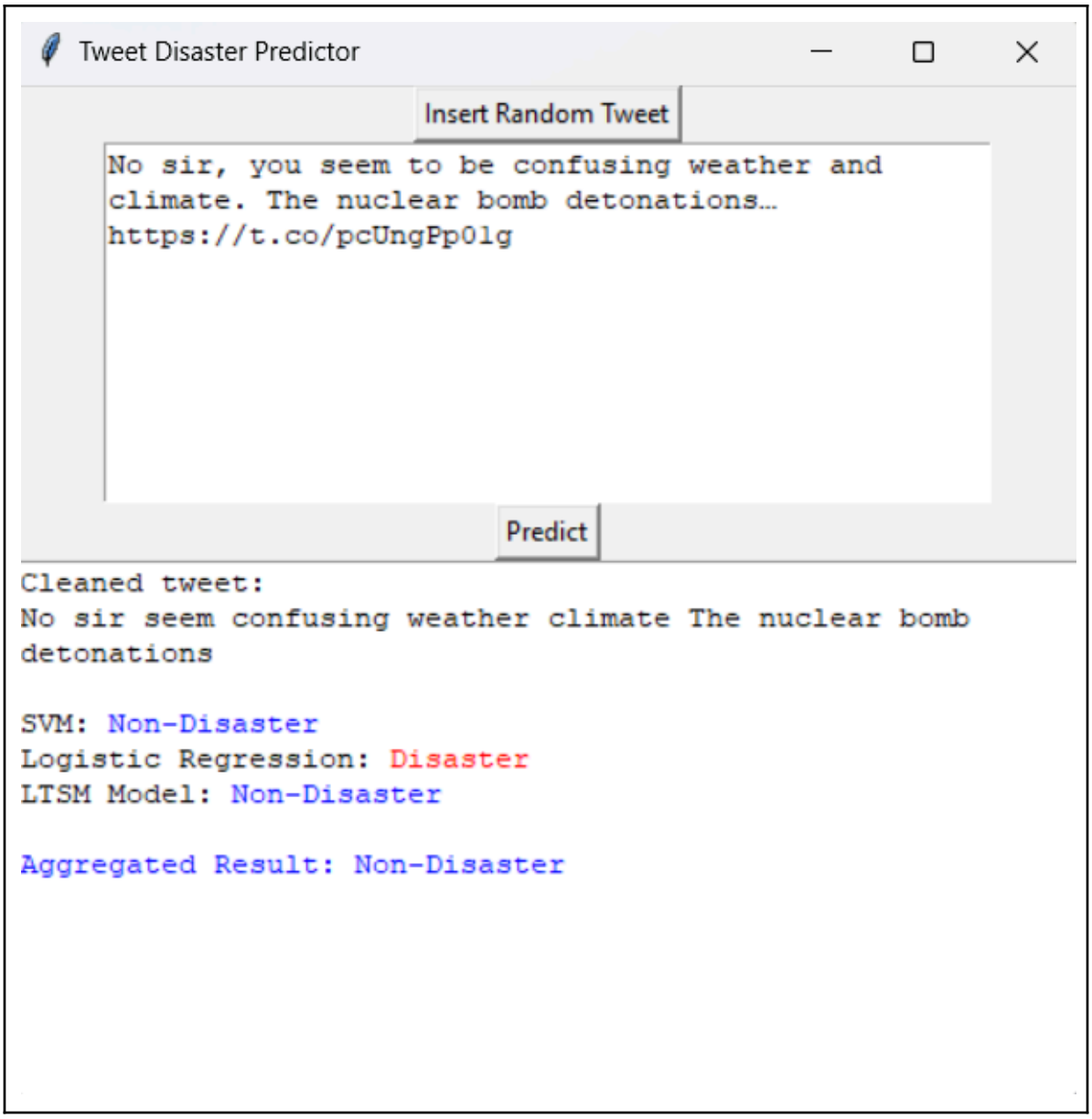
Button:

- **Insert Random Tweet**: nút tích hợp hàm để tạo ra một tweet random từ bộ dữ liệu chưa xử lý ban đầu.
- **Predict:** nút tích hợp hàm để fit tweet đã được xử lý (cleaned tweet) và dự đoán kết quả của cả ba mô hình được chọn (Disaster hoặc là Non-Disaster)

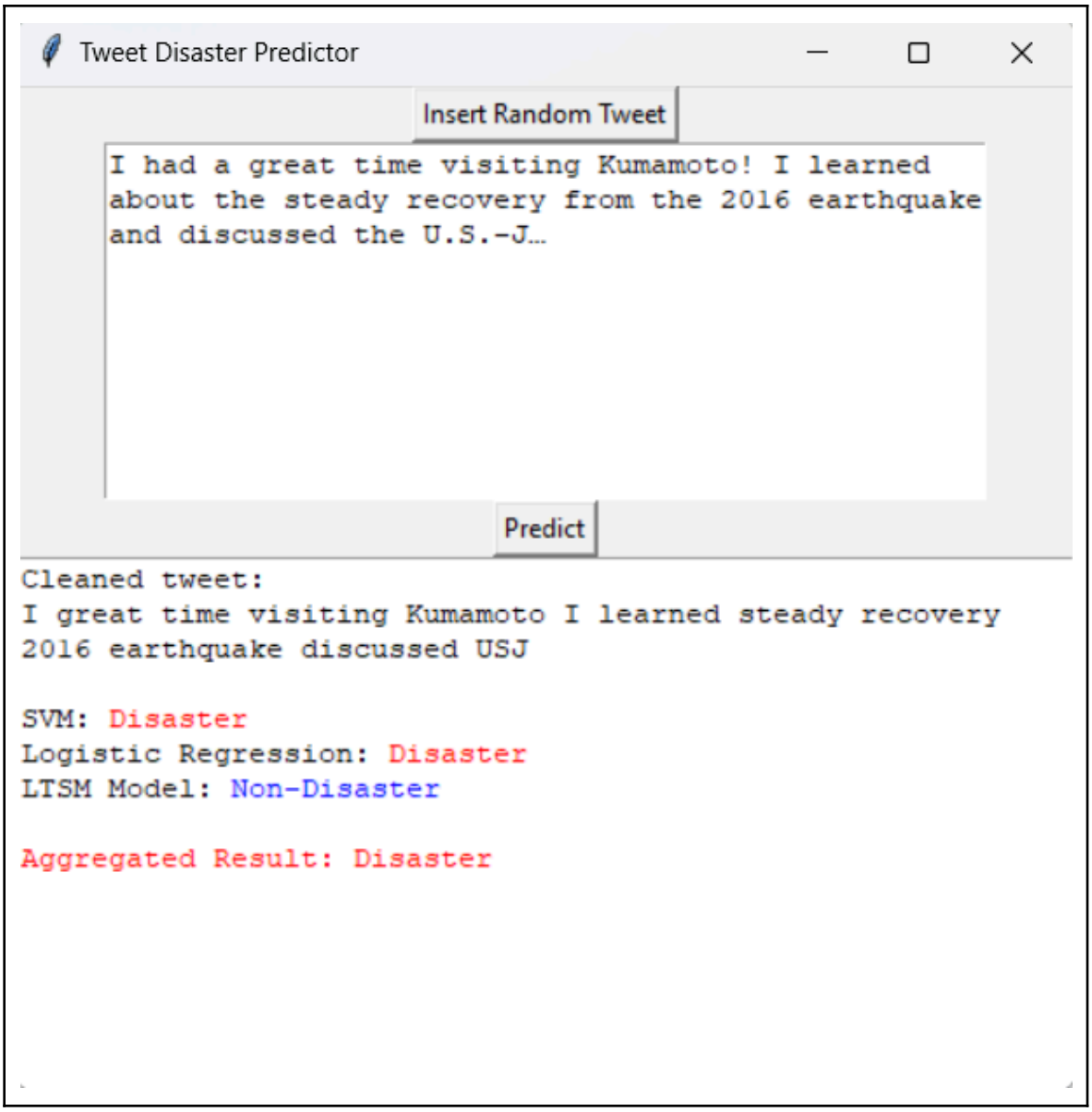
6.3. Áp dụng




Tweet Disaster Predictor
Insert Random Tweet
In Bhainsa of Nirmal district(Telangana), stone pelter and arsonist mobs have selectively targeted hindu properties, v…
Predict
Cleaned tweet:
In Bhainsa Nirmal districtTelangana stone pelter arsonist mobs selectively targeted hindu properties v
SVM: Disaster
Logistic Regression: Disaster
LTSM Model: Disaster
Aggregated Result: Disaster



Như ta thấy ở đây thì hai mô hình SVM và LTSM dự đoán là Output không phải tweet thảm họa nhưng mô hình Logistic Regression lại dự đoán là có nên quyết định cuối vẫn là không phải thảm họa.



Ở ví dụ này thì hai mô hình SVM và Logistic Regression dự đoán là một tweet thảm họa nhưng mô hình LTSM lại dự đoán là không phải tweet thảm họa nên quyết định cuối Output là một tweet thảm họa.

BẢNG PHÂN CÔNG

Thành viên	Công việc	Đánh giá
Hà Gia Hiến	Lý thuyết chương 1, 2, 4 Viết, format báo cáo Làm slide	100%
Lý Gia Thuận	Chương 3, 4, 5 Viết báo cáo Làm slide	100%
Lê Minh Triều	Chương 3, 4, 6 Viết báo cáo Làm slide	100%

TÀI LIỆU THAM KHẢO

[1] Bài giảng học phần Xử lý ngôn ngữ tự nhiên, TS. Đặng Ngọc Hoàng Thành, khoa Công Nghệ Thông Tin Kinh Doanh, trường Công nghệ và Thiết kế UEH, 2023.

[2] Viktor S, *Dataset "Disaster Tweets"*, [Trực tuyến]. Địa chỉ:

https://www.kaggle.com/datasets/vstepanenko/disaster-tweets/data

[3] Kulkarni, A., & Shivananda, A. (2019). *Natural language processing recipes: Unlocking text data with machine learning and deep learning using Python* (2nd ed.). Apress.

[4] Jurafsky, D., & Martin, J. H. (2023, January 7). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition* (3rd ed. draft). Stanford University.

[5] Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python: analyzing text with the natural language toolkit*. O'Reilly Media, Inc.

[6] Bird, S. (2017, September 28). *NLTK documentation (Release 3.2.5)*, [Trực tuyến]. Địa chỉ:

https://buildmedia.readthedocs.org/media/pdf/nltk/stable/nltk.pdf

SOURCE CODE

https://github.com/MinhTrjeu/NLP_Final/blob/main/NLP_FINAL%20(1).ipynb