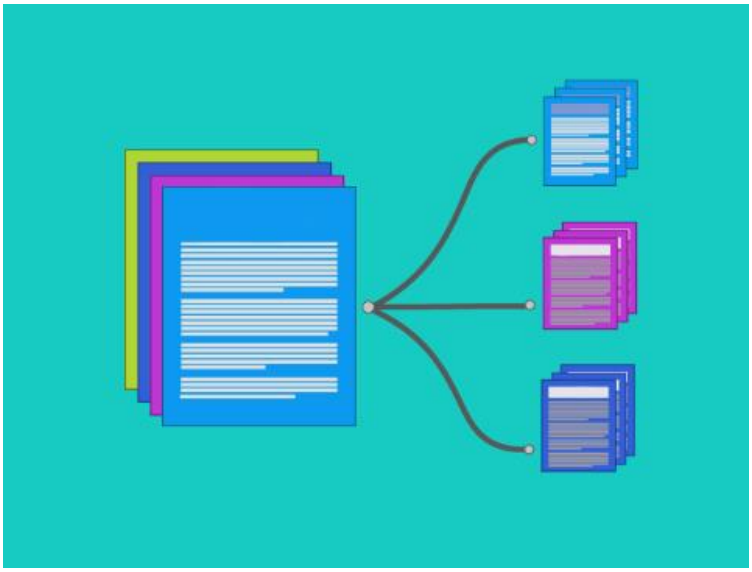
BÁO CÁO ĐỒ ÁN CUỐI KÌ
TEXT CLASSIFICATION - NLP

MÔN: HỌC THỐNG KÊ
HK3-2023



AUGUST 14

GVHD: Mr. Ngo Minh Nhut
Mr. Le Long Quoc
Authored by: 20127674 – Lê Đức Đạt
20127141 – Bùi Tuấn Dũng
20127268 – Hà Đăng Nhuận

TRUONG DAI HOC KHOA HOC TU NHIEN
KHTN
TP. HO CHI MINH

Mục lục

1. Danh sách thành viên nhóm: 4
2. Giới thiệu: 4
   a. Bối cảnh, mục đích nghiên cứu và làm đề tài: 4
   b. Tổng quan về bài toán phân loại cảm xúc văn bản: 5
3. Tập dữ liệu: 5
   a. Nguồn gốc – cách thu thập dữ liệu: 5
   b. Mô tả cấu trúc tập dữ liệu: 6
   c. Phân tích thống kê tập dữ liệu: 6
   d. Các bước xử lý, làm sạch, chuẩn hóa dữ liệu: 7
4. Mô hình: 12
   a. SVM: 12
   b. LSTM/BiLSTM: 14
   c. BERT: 15
5. Huấn luyện mô hình: 15
   a. Cách chia tập dữ liệu cho mô hình: (Xem ở bảng trên) 16
   b. Quá trình huấn luyện mô hình: 16
6. Đánh giá mô hình: 17
   a. Đánh giá mô hình trên tập train, validation và test: 17
   b. Các metric đánh giá (accuracy, precision, recall, F1): 21
7. Hạn chế và hướng phát triển trong tương lai: 23
8. Tư liệu tham khảo: 24

1. Danh sách thành viên nhóm:

STT	Họ và tên	MSSV
1	Lê Đức Đạt	20127674
2	Bùi Tuấn Dũng	20127141
3	Hà Đăng Nhuận	20127268

2. Giới thiệu:

a. Bối cảnh, mục đích nghiên cứu và làm đề tài:

Trong thời đại công nghệ số ngày nay, việc phân tích và hiểu được cảm xúc của con người thông qua ngôn ngữ là vô cùng quan trọng. Khả năng phân loại cảm xúc chính xác sẽ mang lại nhiều ứng dụng thiết thực trong các lĩnh vực như chăm sóc khách hàng, trợ lý ảo, giám sát mạng xã hội, v.v.

Trong bối cảnh đó, việc nghiên cứu và xây dựng mô hình phân loại cảm xúc văn bản là hết sức cần thiết. Mục tiêu của đề tài này là xây dựng mô hình phân loại cảm xúc văn bản dựa trên các kỹ thuật deep learning như LSTM, BiLSTM, Word2Vec, SVM hay BERT.

Cụ thể, đề tài sẽ nghiên cứu và giải quyết các vấn đề:

(1) Xây dựng tập dữ liệu huấn luyện cho bài toán;

(2) Thử nghiệm các kiến trúc mô hình phù hợp;

(3) Tối ưu hóa siêu tham số để nâng cao hiệu năng mô hình;

(4) Đánh giá và so sánh các mô hình trên tập dữ liệu.

Kết quả của đề tài là mô hình phân loại cảm xúc văn bản với độ chính xác cao, góp phần phát triển các ứng dụng AI trong tương lai.

b. Tổng quan về bài toán phân loại cảm xúc văn bản:

Phân loại cảm xúc văn bản (sentiment analysis) là bài toán dự đoán cảm xúc/thái độ (tích cực, tiêu cực, trung tính) thể hiện trong một đoạn văn bản. Đây là một trong những ứng dụng quan trọng của xử lý ngôn ngữ tự nhiên (NLP) trong thời đại dữ liệu lớn hiện nay.

Các nghiên cứu về phân loại cảm xúc văn bản đã được tiến hành từ những năm 2000. Ban đầu các phương pháp máy học truyền thống như SVM, Naive Bayes được sử dụng với kết hợp kỹ thuật trích xuất đặc trưng. Gần đây, với sự phát triển mạnh mẽ của deep learning, các mô hình dựa trên mạng nơ-ron nhân tạo như CNN, LSTM/GRU đã cho kết quả vượt trội hơn nhiều. Các mô hình tiên tiến nhất hiện nay sử dụng kiến trúc Transformer và huấn luyện trên khối lượng dữ liệu lớn (BERT, RoBERTa, ELECTRA,...).

3. Tập dữ liệu:

a. Nguồn gốc – cách thu thập dữ liệu:

Tập dữ liệu này được tạo tự động bằng cách thu thập các tweet trên Twitter thông qua API của Twitter (hơn 10.000 tweet tiếng Anh). Quá trình thu thập diễn ra trong khoảng thời gian từ năm 2009 đến năm 2012.

Các tweet được lọc dựa trên các cụm từ tìm kiếm cụ thể (ví dụ: "lyx"). Ngoài ra, để tạo tập dữ liệu có nhãn, các tweet chứa biểu tượng cảm xúc tích cực như “:)” được gán nhãn Positive, các tweet chứa biểu tượng tiêu cực như “:(” được gán nhãn Negative. Có 3 nhãn: Positive, Negative và Neutral (Trung lập).

Sau khi thu thập, dữ liệu được làm sạch bằng cách loại bỏ các ký tự đặc biệt và biểu tượng cảm xúc. Cuối cùng, dữ liệu được lưu dưới dạng CSV với 6 trường: nhãn cảm xúc, id tweet, ngày tweet, cụm từ tìm kiếm, người dùng tweet, nội dung tweet.

Như vậy, tập dữ liệu này rất phù hợp để ta khám phá và thử nghiệm các mô hình phân loại cảm xúc trên text.

Acknowledgements:

Tập dữ liệu này được sáng tạo và chia sẻ công khai trên Kaggle bởi các chuyên gia dữ liệu.

Inspiration:

Một số câu hỏi có thể được tìm câu trả lời thông qua tập dữ liệu:

Mô hình nào cho kết quả phân loại tốt nhất trên tập dữ liệu này?

Làm thế nào để cải thiện độ chính xác của mô hình?

Các yếu tố nào ảnh hưởng đến cảm xúc của người dùng Twitter?

Có thể phát triển mô hình phân loại cho nhiều ngôn ngữ khác dựa trên mô hình huấn luyện trên tiếng Anh không?

b. Mô tả cấu trúc tập dữ liệu:

- Tập huấn luyện "train.csv":

Số lượng mẫu: 27,481

Số lượng trường dữ liệu: 10

Các trường chính: textID, text, selected_text, sentiment, Time of Tweet, Age of User, Country, Population -2020, Land Area (Km²), Density (P/Km²)

textID	text	selected_text	sentiment	Time of Tweet	Age of User	Country	Population -2020	Land Area	Density (P/Km²)
cb774db0d1	I`d have responded, if I wer	I`d have responded, if I were	neutral	morning	0-20	Afghanist	38928346	652860	60
549e992a42	Sooo SAD I will miss you he	Sooo SAD	negative	noon	21-30	Albania	2877797	27400	105
088c60f138	my boss is bullying me...	bullying me	negative	night	31-45	Algeria	43851044	2381740	18
9642c003ef	what interview! leave me a	leave me alone	negative	morning	46-60	Andorra	77265	470	164
358bd9e861	Sons of ****, why couldn`t	Sons of ****,	negative	noon	60-70	Angola	32866272	1246700	26

- Tập kiểm tra "test.csv":

Số lượng mẫu: 4,815

Số lượng trường dữ liệu: 9 (không có trường selected_text)

c. Phân tích thống kê tập dữ liệu:

- Độ dài văn bản:

Tập huấn luyện (train.csv):

Trung bình: 68.35 ký tự

Độ lệch chuẩn: 35.62 ký tự

Độ dài tối thiểu: 3 ký tự

Độ dài tối đa: 159 ký tự

Tập kiểm tra (test.csv):

Trung bình: 67.78 ký tự

Độ lệch chuẩn: 35.59 ký tự

Độ dài tối thiểu: 4 ký tự

Độ dài tối đa: 148 ký tự

- Phân phối các nhãn:

Tập huấn luyện (train.csv):

Neutral: 40.46%

Positive: 31.23%

Negative: 28.31%

Tập kiểm tra (test.csv):

Neutral: 40.46%

Positive: 31.21%

Negative: 28.32%

d. Các bước xử lý, làm sạch, chuẩn hóa dữ liệu:

NẾU MUỐN BIẾT RÕ HƠN THÌ CÓ THỂ VÀO PHẦN CODE, CÓ CÁC COMMENT GHI CHÚ TRÊN TỪNG DÒNG CODE VÀ PHẦN GHI CHÚ BÊN NGOÀI ĐỂ DỄ HIỂU HƠN.

<table>
<tr><th rowspan="2">STT</th><th rowspan="2">CÁC BƯỚC</th><th colspan="2">CÁC MÔ HÌNH</th></tr>
<tr><th>BERT</th><th>SVM (W2V, B-O-W, TF-IDF), LSTM, BiLSTM,</th></tr>
<tr><td>1</td><td>Chuyển đổi Kiểu Dữ liệu</td><td colspan="2">Đọc dữ liệu từ train.csv và test.csv với mã hóa 'ISO-8859-1'.
encoding='ISO-8859-1' định nghĩa kiểu mã hóa ký tự (character encoding) được sử dụng để đọc và ghi dữ liệu trong tệp CSV.
ISO-8859-1 là một chuẩn mã hóa ký tự phổ biến, bao gồm các ký tự Latin alphabet thông dụng trong nhiều ngôn ngữ châu Âu và phương Tây.
Đặc điểm của ISO-8859-1:
Mã hóa 1 byte cho mỗi ký tự, bao gồm các chữ cái, số, ký tự đặc biệt cơ bản, không hỗ trợ các ký tự tiếng Việt có dấu hoặc ký tự đặc biệt trong một số ngôn ngữ khác.
Khi đọc tệp CSV bằng Pandas, chúng ta cần chỉ định encoding phù hợp để đảm bảo dữ liệu được decode chính xác.
ISO-8859-1 là lựa chọn phổ biến nếu dữ liệu chủ yếu là tiếng Anh hoặc các ngôn ngữ châu Âu sử dụng Latin alphabet.
<pre># Đọc dữ liệu từ file tải lên
df = pd.read_csv('train.csv', encoding='ISO-8859-1')
df.head()</pre>
<pre># Đọc dữ liệu từ file tải lên và tiền xử lý
test_df_original = pd.read_csv('test.csv', encoding='ISO-8859-1')
test_processed = []</pre>
<pre># Load datasets
train_data = pd.read_csv('train.csv', encoding='ISO-8859-1')
test_data = pd.read_csv('test.csv', encoding='ISO-8859-1')</pre></td></tr>
<tr><td></td><td></td><td>x</td><td>Chuyển đổi tất cả dữ liệu văn bản (cột 'text') sang kiểu chuỗi (string).
<pre># Convert all text data to string type
train_data['text'] = train_data['text'].astype(str)
test_data['text'] = test_data['text'].astype(str)</pre></td></tr>
</table>

			Thay thế các giá trị NaN trong cột 'text' bằng giá trị placeholder 'placeholder_text'.
			`# Replace any NaN values with a placeholder`
			`train_data['text'].fillna('placeholder_text', inplace=True)`
			`# Thay thế các giá trị NaN trong cột 'text' bằng 'placeholder_text'`
			`test_data['text'].fillna('placeholder_text', inplace=True)`
			Chuyển đổi cột 'sentiment' sang kiểu chuỗi và thay thế giá trị NaN bằng 'placeholder_sentiment'.
			`# Convert sentiment labels to strings and replace NaNs with placeholder`
			`train_data['sentiment'] = train_data['sentiment'].astype(str).fillna('placeholder_sentiment')`
			`# Ép kiểu cột 'sentiment' về string`
			`test_data['sentiment'] = test_data['sentiment'].astype(str).fillna('placeholder_sentiment')`
			`# Thay thế NaN trong cột 'sentiment' bằng giá trị placeholder`
			Lý do:
			Tránh được lỗi khi máy học không xử lý được NaN
			Đảm bảo các nhãn sentiment đều có giá trị và kiểu string nhất quán
			Các placeholder sẽ bị loại bỏ trong quá trình huấn luyện
			sample(n) lấy ngẫu nhiên n mẫu dữ liệu để huấn luyện và kiểm tra nhanh. Giúp tránh OOM khi dữ liệu lớn.

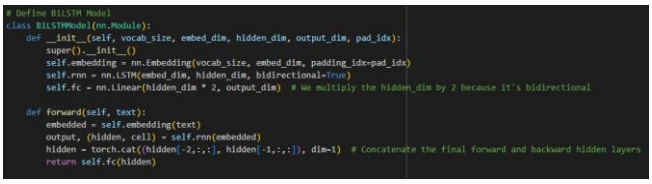
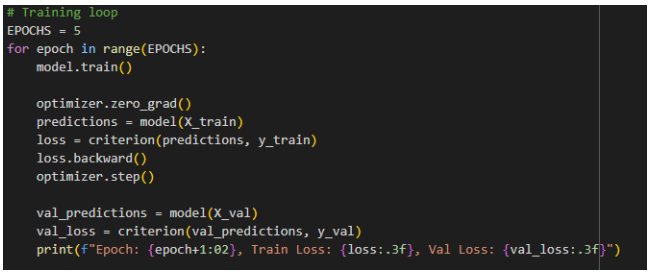
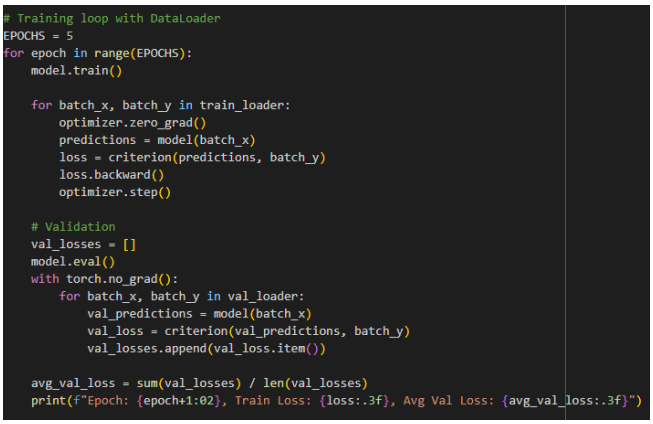
2	Chọn mẫu dữ liệu	Lấy mẫu 2000 phần tử đầu tiên từ DataFrame df để xử lý.
		`processed_data = []`
		`# Khởi tạo list rỗng để lưu kết quả.`
		`for i in range(len(df[:2000])):`
		`  processed_data.append(process_data(df.iloc[i]))`
			Lấy mẫu ngẫu nhiên 5000 mẫu từ tập huấn luyện và 1000 mẫu từ tập kiểm tra. Điều này giúp giảm kích thước của dữ liệu, tránh việc hết bộ nhớ khi dữ liệu quá lớn.
			`# Use only a subset of data for quick demonstration and to prevent OOM (Out-of-Memory)`
			`# Lấy mẫu ngẫu nhiên 5000 mẫu huấn luyện và 1000 mẫu kiểm tra từ tập dữ liệu ban đầu để tránh OOM.`
			`SAMPLE_SIZE = 5000`
			`train_data = train_data.sample(SAMPLE_SIZE, random_state=42)`
			`test_data = test_data.sample(int(SAMPLE_SIZE * 0.2), random_state=42)`

3	Tokenization	Khởi tạo tokenizer từ mô hình BERT pre-trained.
		Tokenize văn bản thành các token sử
			Khởi tạo một tokenizer từ thư viện Keras.
			`# Initialize the tokenizer`
			`# Lấy mẫu ngẫu nhiên 5000 mẫu huấn luyện và 1000 mẫu...`
			`# ...kiểm tra từ tập dữ liệu ban đầu để tránh OOM.`
			`tokenizer = Tokenizer()`
			`tokenizer.fit_on_texts(train_data['text'])`

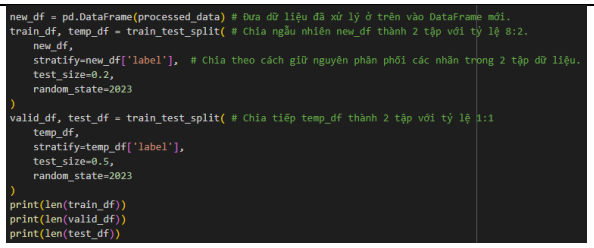
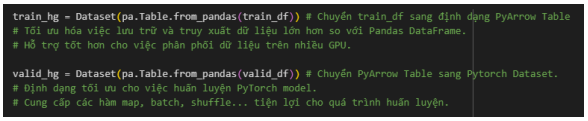
		dụng tokenizer BERT.
		`tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')`
		`# Khởi tạo tokenizer từ mô hình BERT pre-trained.`
		`# BERT yêu cầu tokenize text thành các token trước khi đưa vào mô hình.`
			Sử dụng tokenizer này để chuyển đổi văn bản thành các chuỗi số (sequences).
			`# Tokenize the text data`
			`# Áp dụng tokenizer đã huấn luyện cho cả tập huấn luyện`
			`# và kiểm tra để chuyển các câu text thành sequences số.`
			`train_sequences = tokenizer.texts_to_sequences(train_data['text'])`
			`test_sequences = tokenizer.texts_to_sequences(test_data['text'])`
			Áp dụng TF-IDF vectorization và Bag-of-Words (BoW) vectorization trên văn bản. Cả hai phương pháp này đều biến đổi văn bản thành một vector số.
			`# Adjust TF-IDF and Bag-of-Words vectorization`
			`MAX_FEATURES = 2000`
			`tfidf = TfidfVectorizer(max_features=2000)`
			`bow_vectorizer = CountVectorizer(max_features=MAX_FEATURES)`
			`# Bag-of-Words for SVM`
			`bow_vectorizer = CountVectorizer(max_features=2000)`
			`X_train_bow = bow_vectorizer.fit_transform(X_train_text).toarray()`
			`X_val_bow = bow_vectorizer.transform(X_val_text).toarray()`
			`X_test_bow = bow_vectorizer.transform(test_data['text']).toarray()`
			Sử dụng Word2Vec để biến đổi văn bản thành vector dựa trên không gian ngữ nghĩa của từ.
			 

4	Padding	Trong quá trình tokenization với BERT tokenizer, việc padding đã được áp dụng để đảm bảo mỗi chuỗi có độ dài cố định là 128 tokens.
		`encodings = tokenizer(text, padding="max_length", truncation=True, max_length=128)`
		`# Tokenize text thành các tokens sử dụng tokenizer BERT.`
		`# Quy định độ dài cố định 128 tokens cho mỗi text bằng cách padding hoặc truncate.`
			Padding các chuỗi số này để đảm bảo chúng có cùng độ dài. Điều này giúp chuẩn bị đầu vào cho mô hình LSTM.
			`# Padding sequences`
			`# Sử dụng pad_sequences để căn chỉnh độ dài các sequences bằng cách padding 0 vào cuối.`
			`# Độ dài là max length của các sequences trong tập huấn luyện.`
			`X_train = pad_sequences(train_sequences)`
			`X_test = pad_sequences(test_sequences, maxlen=X_train.shape[1])`

5	Xử lý dữ liệu	Lấy ra và chuẩn hóa cột 'text' (xóa khoảng trắng thừa). `text = ' '.join(text.split())`	Khởi tạo và định nghĩa mô hình LSTM và BiLSTM.

		Mã hóa nhãn dựa trên giá trị của 'sentiment'.
		`# Gán nhãn dựa trên giá trị của 'sentiment'`
		`if row['sentiment'] == 'negative':`
		`label = 0`
		`elif row['sentiment'] == 'neutral':`
		`label = 1`
		`else: # positive`
		`label = 2`
		`encodings['label'] = label`
		`encodings['text'] = text`
		Áp dụng hàm process_data trên mỗi dòng của DataFrame.
			
			 
			Huấn luyện mô hình bằng vòng lặp.
			 
			 

6	Chia tập dữ liệu	Chia dữ liệu đã xử lý thành ba tập: tập huấn luyện, tập validation và tập kiểm tra. Việc chia này sử dụng phân chia chiến lược (stratified split) để giữ nguyên tỷ lệ phân phối của các nhãn trong các tập dữ liệu. Tỉ lệ giữa các tập là: 80% (train) - 10% (validation) - 10% (test).	Chia dữ liệu huấn luyện thành hai tập: tập huấn luyện và tập xác thực: X_train_text, X_val_text, y_train, y_val = train_test_split(...). Chia dữ liệu thành các batches sử dụng DataLoader.

			
7	Chuyển đổi dữ liệu	Chuyển dữ liệu từ DataFrame Pandas sang định dạng PyArrow Table và sau đó sang định dạng Pytorch Dataset. Điều này giúp tối ưu hóa việc lưu trữ và truy xuất dữ liệu, cung cấp một giao diện tiện lợi để huấn luyện mô hình PyTorch. 	Mã hóa nhãn dữ liệu từ chuỗi sang số nguyên sử dụng LabelEncoder Chuyển dữ liệu thành tensor của PyTorch đối với tập train, test và val. Chuyển dữ liệu thành Dataset và DataLoader để xử lý batch. (xem trong file code)

4. Mô hình:

a. SVM:

(Support Vector Machine) với kernel tuyến tính. Đây là mô hình phân loại cơ bản dựa trên học có giám sát.

- Kiến trúc mô hình gồm:

+ Lớp vector hóa đầu vào dữ liệu văn bản: sử dụng TF-IDF, Bag of Words hoặc Word2Vec để biểu diễn các câu văn bản dưới dạng vectơ số, cụ thể:

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

Vector hóa văn bản bằng cách biểu diễn mỗi văn bản thành một vectơ, trong đó mỗi chiều là một từ duy nhất.

Giá trị của mỗi chiều được tính bằng tần suất xuất hiện của từ đó trong văn bản nhân với trọng số IDF của từ đó.

IDF giúp đánh giá mức độ quan trọng của từ dựa vào tần suất xuất hiện trong toàn bộ tập văn bản.

Kết quả là mỗi văn bản được biểu diễn bằng một vectơ với chiều bằng số từ tối đa sử dụng.

- Bag of Words:

Cũng biểu diễn mỗi văn bản thành một vectơ với số chiều bằng số từ tối đa.

Nhưng chỉ sử dụng tần suất xuất hiện của mỗi từ mà không có trọng số IDF.

Đơn giản và nhanh hơn so với TF-IDF nhưng kém chính xác hơn.

- Word2Vec:

Mỗi từ được biểu diễn bằng một vectơ với số chiều cố định thay vì một chiều duy nhất.

Huấn luyện mô hình Word2Vec trên tập văn bản để tự học vectơ đại diện cho mỗi từ.

Mỗi văn bản được biểu diễn bằng trung bình cộng các vectơ của các từ có trong văn bản.

Bắt được ngữ nghĩa của từ ngữ.

+ Lớp phân loại SVM: sử dụng thuật toán LinearSVC trong thư viện Scikit-Learn, với hàm mục tiêu hinge loss và kỹ thuật tối ưu hóa Pegasos để huấn luyện mô hình.

Các siêu tham số của mô hình:

max_features: Siêu tham số này kiểm soát số lượng tính năng được sử dụng để huấn luyện mô hình. Giá trị mặc định là 10000, nhưng giá trị này đã được giảm xuống còn 2000 nhằm giảm thời gian đào tạo và sử dụng bộ nhớ của các mô hình.

C: Siêu tham số C kiểm soát sự đánh đổi giữa lỗi đào tạo và thuật ngữ chính quy hóa. Giá trị C cao hơn sẽ dẫn đến một mô hình có lỗi đào tạo thấp hơn, nhưng nó cũng có thể dễ bị trang bị quá mức. Giá trị C thấp hơn sẽ dẫn đến một mô hình có lỗi đào tạo cao hơn, nhưng nó có thể ít bị thừa hơn.

Kernel: Siêu tham số kiểm soát kernels được sử dụng bởi SVM. Giá trị mặc định là "rbf", nhưng giá trị này đã được thay đổi thành "tuyến tính" để cải thiện hiệu suất của các mô hình trên bộ dữ liệu phân tích tình cảm.

b. LSTM/BiLSTM:

- Mô hình LSTM/BiLSTM bao gồm các lớp sau:

Lớp embedding: chuyển đổi token của câu văn thành một không gian vector nhúng.

Lớp LSTM/BiLSTM: là lớp RNN được sử dụng để xử lý dữ liệu chuỗi.

Lớp kết nối đầy đủ (fully-connected layer): ánh xạ đầu ra từ lớp LSTM/BiLSTM đến số lớp nhãn của bài toán.

- Kiến trúc mô hình:

Lớp embedding: Kích thước từ điển là VOCAB_SIZE và kích thước không gian nhúng là EMBED_DIM.

Lớp LSTM/BiLSTM: Số chiều ẩn là HIDDEN_DIM. Đối với mô hình BiLSTM, nó sẽ có chiều ẩn gấp đôi so với LSTM thông thường.

Lớp kết nối đầy đủ: Số chiều đầu ra là OUTPUT_DIM, tương ứng với số lớp nhãn của bài toán.

- Các siêu tham số:

VOCAB_SIZE: Kích thước từ điển.

EMBED_DIM: Kích thước không gian nhúng.

HIDDEN_DIM: Số chiều ẩn của lớp LSTM/BiLSTM.

OUTPUT_DIM: Số lớp đầu ra.

PAD_IDX: Chỉ số của token padding.

BATCH_SIZE: Kích thước của mỗi batch dữ liệu.

EPOCHS: Số epoch trong quá trình huấn luyện.

c. BERT:

- Kiến trúc mô hình gồm:

Encoder: sử dụng kiến trúc Transformer để mã hóa ngữ nghĩa của các token trong văn bản đầu vào.

Pooling layer: tổng hợp thông tin từ các token bằng cách lấy vector ở vị trí [CLS].

Dense layer: ánh xạ từ không gian vector của BERT xuống 3 nhãn phân loại.

- Mô hình được khởi tạo từ checkpoint BERT Base Uncased đã được huấn luyện sẵn. Một số siêu tham số chính:

Số lớp Encoder: 12

Kích thước embedding: 768

Tổng số tham số: 110M

- Quá trình huấn luyện:

Sử dụng thư viện Transformers và Trainer API.

Chia tập dữ liệu thành train, validation, test.

Huấn luyện trong 5 epochs với batch size mặc định.

Đánh giá trên tập validation sau mỗi epoch.

Lưu lại checkpoint mô hình tốt nhất.

5. Huấn luyện mô hình:

a. Cách chia tập dữ liệu cho mô hình: (Xem ở bảng trên)

b. Quá trình huấn luyện mô hình:

- BERT:

+ Về kiến trúc và siêu tham số mô hình:

Sử dụng BERT Base model đã được pretrain sẵn với kiến trúc Transformer.

Đầu vào là các câu văn bản được tokenize và padding về chiều dài cố định 128 tokens.

Thêm một dense layer cuối cùng với num_labels = 3 tương ứng với 3 nhãn phân loại.

Số lớp Encoder: 12

Hidden size: 768

Tổng số tham số: 110 triệu

+ Quá trình huấn luyện:

Sử dụng thư viện Transformers và API Trainer để huấn luyện.

Chia tập dữ liệu thành train, validation và test.

Huấn luyện trong 5 epochs với batch size mặc định là 32.

Đánh giá trên tập validation sau mỗi epoch.

Lưu lại checkpoint mô hình tốt nhất dựa trên validation loss.

Sau khi huấn luyện xong, đánh giá lại trên tập test độc lập để kiểm tra khả năng generalization.

- LSTM/BiLSTM:

+ Về kiến trúc và siêu tham số của mô hình:

VOCAB_SIZE: kích thước từ vựng

EMBED_DIM: chiều embedding

HIDDEN_DIM: số chiều ẩn của LSTM

OUTPUT_DIM: số chiều output

PAD_IDX: chỉ số của padding token

+ Về quá trình huấn luyện của mô hình:

Sử dụng hàm mất mát CrossEntropyLoss và optimizer Adam để huấn luyện.

Chia tập huấn luyện thành 2 tập train và validation để đánh giá quá trình huấn luyện.

Huấn luyện trong 5 epochs với batch size mặc định của PyTorch.

Mỗi epoch:

Forward pass qua toàn bộ tập train để dự đoán.

Tính loss so với nhãn ground truth.

Backpropagate để cập nhật trọng số mô hình.

Reset gradient.

Forward pass qua tập validation để đánh giá.

In ra loss trên cả train và validation.

Sau quá trình huấn luyện, đánh giá mô hình trên tập test độc lập để kiểm tra độ chính xác.

Sử dụng DataLoader để chia nhỏ tập dữ liệu thành các batch và nâng cao hiệu năng huấn luyện.

- SVM:

+ Về kiến trúc và siêu tham số của mô hình:

Sử dụng mô hình LinearSVC từ thư viện Scikit-Learn.

Áp dụng 3 phương pháp vector hóa văn bản: TF-IDF, Bag of Words, Word2Vec.

Giới hạn số lượng features tối đa là 2000 với tham số max_features.

+ Về quá trình huấn luyện của mô hình:

Chia tập train thành 2 tập train và validation để huấn luyện và đánh giá mô hình.

Thực hiện vector hóa riêng cho 3 tập dữ liệu train, validation và test.

Huấn luyện mô hình trên tập train vector.

Không thấy có quá trình tuning siêu tham số mô hình.

Đánh giá mô hình trên tập test bằng cách dự đoán và in classification report.

So sánh kết quả của 3 phương pháp vector hóa khác nhau.

6. Đánh giá mô hình:

a. Đánh giá mô hình trên tập train, validation và test:

- BERT:

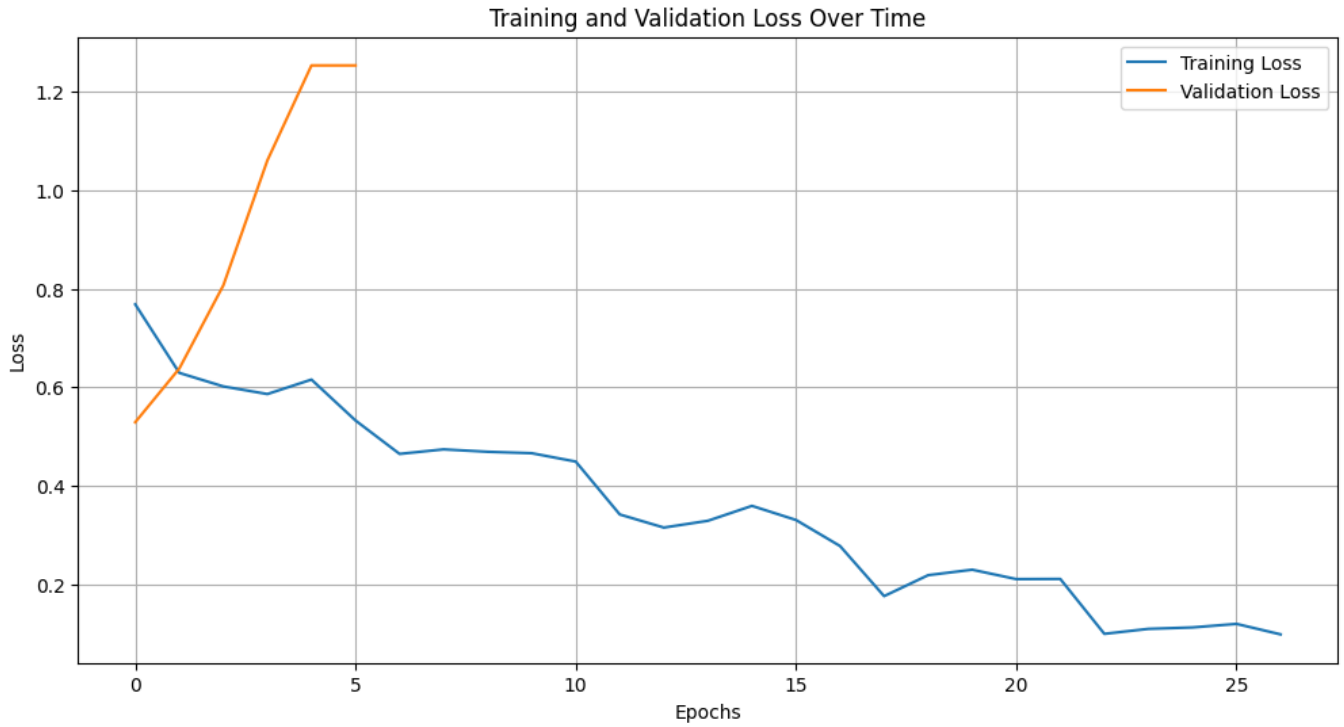
Epoch	Training Loss	Validation Loss
1	0.615700	0.529091
2	0.466100	0.638138
3	0.330400	0.807223
4	0.210300	1.061225
5	0.098000	1.254016

Mô hình được huấn luyện trong 5 epochs.

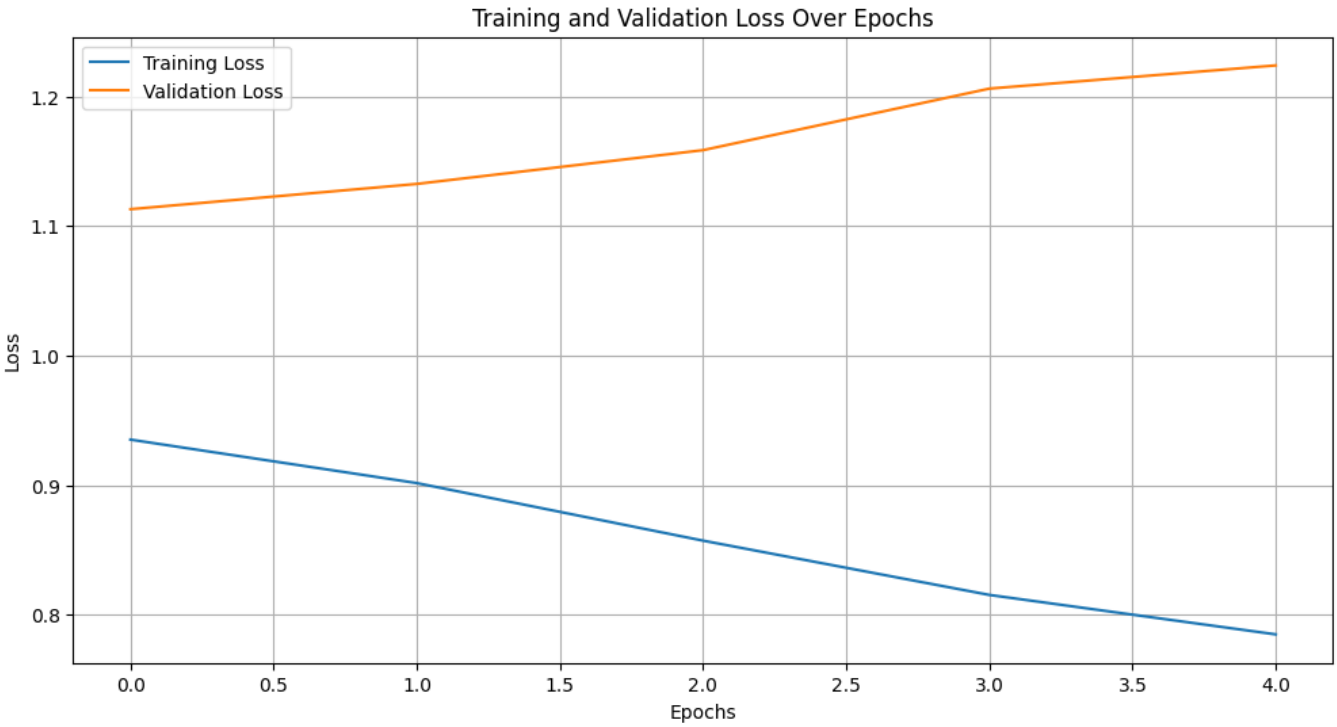
Sau mỗi epoch, mô hình được đánh giá trên cả tập huấn luyện (train) và tập xác thực (validation).

Loss trên tập huấn luyện (Training Loss) giảm dần sau mỗi epoch, từ 0.615 xuống 0.098. Điều này cho thấy mô hình hội tụ và học được pattern từ dữ liệu huấn luyện.

Tuy nhiên, loss trên tập validation (Validation Loss) lại tăng dần sau mỗi epoch, từ 0.529 lên 1.254.



- LSTM/BiLSTM:




```
Epoch: 01, Train Loss: 0.935, Avg Val Loss: 1.113
Epoch: 02, Train Loss: 0.902, Avg Val Loss: 1.132
Epoch: 03, Train Loss: 0.857, Avg Val Loss: 1.159
Epoch: 04, Train Loss: 0.815, Avg Val Loss: 1.206
Epoch: 05, Train Loss: 0.785, Avg Val Loss: 1.224
```

Mô hình được huấn luyện trong 5 epochs.

Sau mỗi epoch, tính toán loss trên tập huấn luyện (Train Loss) và trung bình loss trên tập validation (Avg Val Loss).

Train Loss có xu hướng giảm dần sau mỗi epoch, từ 0.935 xuống còn 0.785. Điều này cho thấy mô hình đang học được và phù hợp dần với dữ liệu huấn luyện.

Tuy nhiên, Validation Loss lại tăng nhẹ sau mỗi epoch, từ 1.113 lên 1.224.

- SVM:

```
Epoch: 01, Train Loss: 1.096, Val Loss: 1.090
Epoch: 02, Train Loss: 1.088, Val Loss: 1.086
Epoch: 03, Train Loss: 1.081, Val Loss: 1.082
Epoch: 04, Train Loss: 1.075, Val Loss: 1.080
Epoch: 05, Train Loss: 1.069, Val Loss: 1.077
Test Accuracy: 37.64%
```

Mô hình được huấn luyện qua 5 epochs.

Cả loss trên tập huấn luyện (Train Loss) và tập kiểm tra (Val Loss) đều giảm dần sau mỗi

epoch.

Tuy nhiên, mức giảm rất nhỏ, chỉ khoảng 0.01 sau mỗi epoch.

Độ chính xác cuối cùng trên tập test là 37.64%

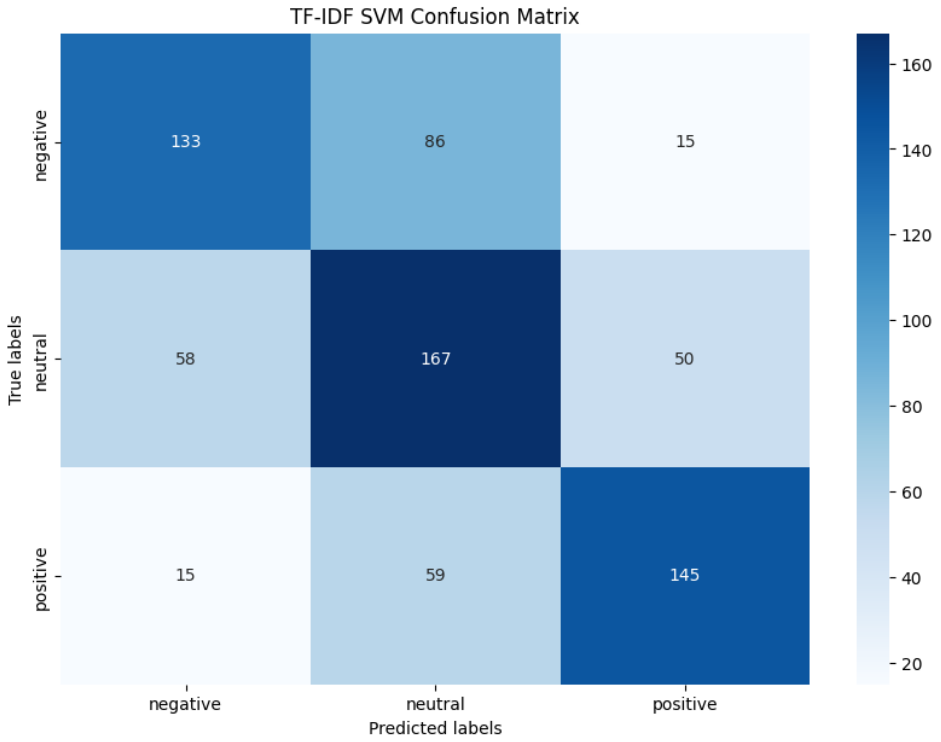
```
Epoch: 01, Train Loss: 1.096, Avg Val Loss: 1.072
Epoch: 02, Train Loss: 1.102, Avg Val Loss: 1.071
Epoch: 03, Train Loss: 0.953, Avg Val Loss: 1.074
Epoch: 04, Train Loss: 0.952, Avg Val Loss: 1.082
Epoch: 05, Train Loss: 1.056, Avg Val Loss: 1.101
Test Loss: 1.158, Test Acc: 37.23%
```

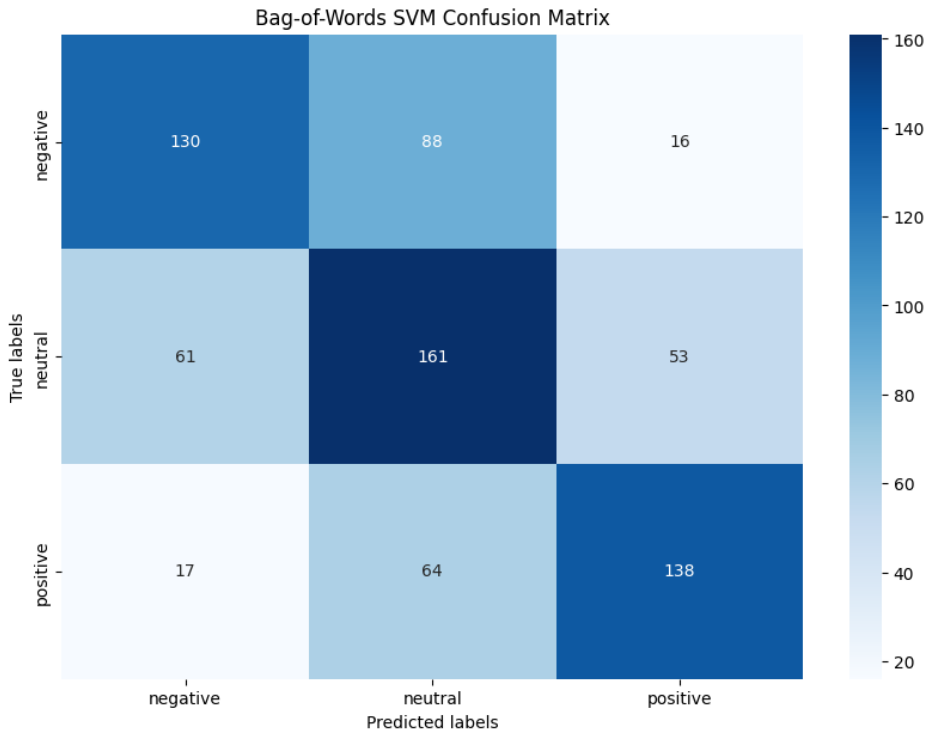
Mô hình được huấn luyện qua 5 epochs.

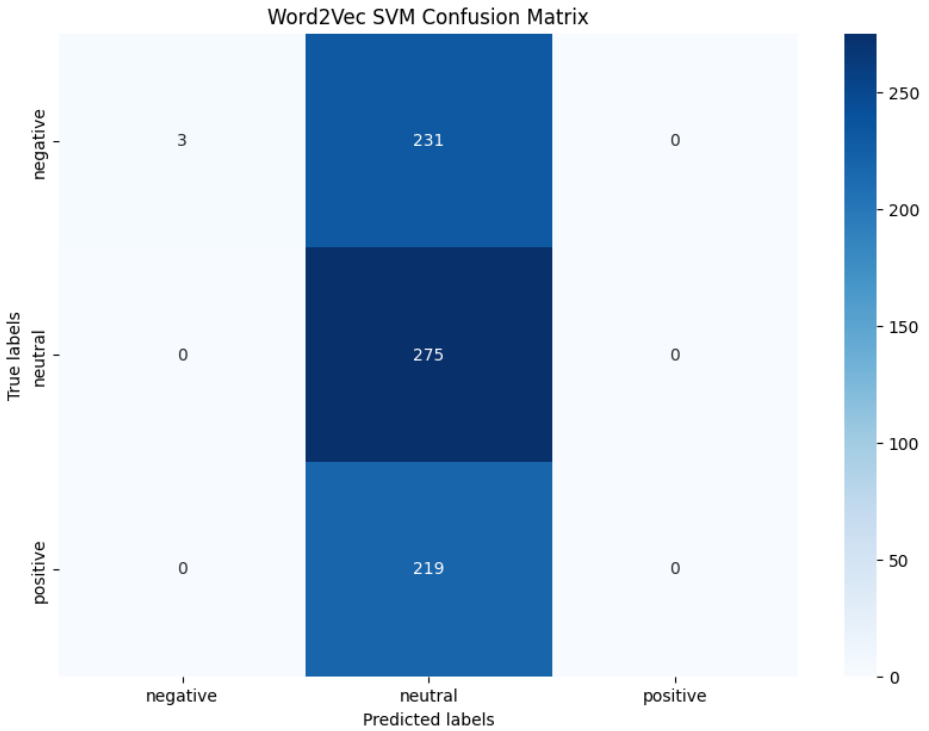
Tại epoch 1 và 2, cả Train Loss và Validation Loss không có sự cải thiện, thậm chí Train Loss còn tăng nhẹ ở epoch 2.

Tại epoch 3, Train Loss giảm mạnh xuống 0.953 nhưng Validation Loss vẫn ở mức cao 1.074.

Độ chính xác cuối cùng trên tập test là 37.23%







b. Các metric đánh giá (accuracy, precision, recall, F1):

- BERT:

	Metric	Score
0	Accuracy	0.783921
1	Precision	0.785143
2	Recall	0.783921
3	F1-Score	0.784180

Độ chính xác (Accuracy) của mô hình là 78.39% trên tập dữ liệu đánh giá. Đây là tỷ lệ dự đoán chính xác trên tổng số mẫu.

Độ đo chính xác (Precision) là 78.51%, cho biết trong số các mẫu được dự đoán dương tính, có 78.51% là dự đoán đúng.

Độ nhạy (Recall) là 78.39%, cho biết trong số các mẫu dương tính, có 78.39% được dự đoán đúng.

F1-Score là 78.42%, là số đo hài hòa giữa Precision và Recall.

Nhìn chung, các metric đều cao, trên 78%, cho thấy mô hình dự đoán khá chính xác trên tập dữ liệu.

- SVM:

+ TF-IDF:

TF-IDF SVM Test Classification Report:

	precision	recall	f1-score	support
neutral	0.65	0.57	0.60	234
positive	0.54	0.61	0.57	275
negative	0.69	0.66	0.68	219
accuracy			0.61	728
macro avg	0.62	0.61	0.62	728
weighted avg	0.62	0.61	0.61	728

Độ chính xác chung (accuracy) là 61%.

Xét từng lớp:

Lớp neutral có độ chính xác (f1-score) thấp nhất, 60%. Độ chính xác (precision) 65% và độ nhạy (recall) 57%.

Lớp positive có f1-score 57%, precision 54%, recall 61%.

Lớp negative có kết quả tốt nhất với f1-score 68%, precision 69%, recall 66%.

+ BoW:

Bag-of-Words SVM Test Classification Report:

	precision	recall	f1-score	support
neutral	0.62	0.56	0.59	234
positive	0.51	0.59	0.55	275
negative	0.67	0.63	0.65	219
accuracy			0.59	728
macro avg	0.60	0.59	0.59	728
weighted avg	0.60	0.59	0.59	728

Độ chính xác chung là 59%, thấp hơn so với sử dụng TF-IDF (61%).

Tương tự TF-IDF, lớp neutral và positive có độ chính xác thấp hơn so với lớp negative.

Cụ thể:

Lớp neutral: f1-score 59%, precision 62%, recall 56%

Lớp positive: f1-score 55%, precision 51%, recall 59%

Lớp negative: f1-score 65%, precision 67%, recall 63%

So sánh với TF-IDF:

Lớp neutral và positive có độ chính xác thấp hơn so với TF-IDF.

Lớp negative có kết quả tương đương TF-IDF.

Như vậy, so với TF-IDF thì Bag of Words không làm tăng độ chính xác cho mô hình SVM.

+ W2V:

Word2Vec SVM Test Classification Report:

	precision	recall	f1-score	support
negative	1.00	0.01	0.03	234
neutral	0.38	1.00	0.55	275
positive	0.00	0.00	0.00	219
accuracy			0.38	728
macro avg	0.46	0.34	0.19	728
weighted avg	0.46	0.38	0.22	728

Độ chính xác chung chỉ đạt 38%.

Mô hình chỉ dự đoán tất cả các mẫu là lớp neutral (f1-score 55%, recall 100%).

Nguyên nhân có thể do Word2Vec chưa huấn luyện tốt nên không bắt được vector từ tốt.

Hoặc do siêu tham số của SVM chưa được tối ưu hóa.

7. Hạn chế và hướng phát triển trong tương lai:

- Hạn chế:

+ Chưa xem xét lại siêu tham số của mô hình một cách kĩ lưỡngđể tối ưu hóa kết quả.

+ Vẫn còn mắc phải tình trạng overfitting khi train model.

+ Chưa có khả năng train nhiều epoch hơn.

+ Độ chính xác chung của các mô hình đều còn khá thấp, dao động từ 38% đến 61%, chưa đạt yêu cầu cao trong thực tế.

+ Chưa có quá trình tối ưu hóa siêu tham số mô hình thực sự hiệu quả.

+ Chất lượng vector từ (Word2Vec) còn hạn chế, chưa bắt được tốt ngữ nghĩa.

- Hướng phát triển trong tương lai:

+ Thử nghiệm thêm nhiều kiến trúc mô hình khác nhau như CNN, RNN, để so sánh kết quả.

+ Tối ưu hóa siêu tham số mô hình thông qua tìm kiếm lưới (Grid Search) và bayesian optimization.

+ Sử dụng kỹ thuật regularization (Dropout, L2 norm) để tránh overfitting.

+ Huấn luyện lại Word2Vec với nhiều epoch hơn, dữ liệu lớn hơn để cải thiện chất lượng vectơ từ.

+ Kết hợp nhiều phương pháp vector hóa khác nhau để tăng độ chính xác cho mô hình.

8. Tư liệu tham khảo:

- Develop a Text Classifier [BERT] using Transformers and PyTorch on your Custom Dataset, https://github.com/RajKKapadia/Transformers-Text-Classification-BERT-Blog. https://www.youtube.com/watch?v=TmT-sKxovb0.

- Vietnamese-News-Classification, https://github.com/anhthuan1999/Vietnamese-News-Classification.

- Đức Đạt's source code, https://www.youtube.com/watch?v=noVHCObgXU8.

- Claude AI

- GPT 4.0