TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

**KHOA CÔNG NGHỆ THÔNG TIN**

--------------------------------



**BÁO CÁO** **CHUYÊN ĐỀ 3**

**Phân tích cảm xúc trong văn bản trên mạng xã hội Twitter**

**Sinh viên thực hiện:** **LỤC TẤN ANH**

**Mã sinh viên: DTC20H4802010103**

**Lớp/Khóa: CNTT K19K**

**Giảng viên hướng dẫn: TS.NGUYỄN TUẤN ANH**

**Thái Nguyên – 202****5**

# **MỤC LỤC**

[**MỤC LỤC** 2](#_Toc194331996)

[**MỤC LỤC ẢNH** 5](#_Toc194331997)

[**GIỚI THIỆU** 5](#_Toc194331998)

[**CHƯƠNG 1.TỔNG QUAN VỀ BÀI TOÁN** 6](#_Toc194331999)

[**1.1.Đọc và phân tích bài toán** 6](#_Toc194332000)

[**1.1.1.Giới thiệu đề tài** 6](#_Toc194332001)

[**1.1.2.Tính đặc thì của dữ liệu** 6](#_Toc194332002)

[**1.1.3.Mô tả bài toán Sentiment Analysis** 8](#_Toc194332003)

[**1.1.4.Ứng dụng của bài toán** 9](#_Toc194332004)

[**1.2.Thu thập dữ liệu và phân tích dữ liệu** 10](#_Toc194332005)

[**1.2.1.Tổng quan về tập dữ liệu** 10](#_Toc194332006)

[**1.2.2.Thu thập dữ liệu** 11](#_Toc194332007)

[**1.2.3.Phân tích dữ liệu** 11](#_Toc194332008)

[**1.2.4.Thống kê dữ liệu** 12](#_Toc194332009)

[**1.2.5.Phương pháp sử dụng** 13](#_Toc194332010)

[**1.3.Tiền xử lý dữ liệu** 13](#_Toc194332011)

[**1.3.1.** **Lý thuyết về tiền xử lý dữ liệu** 13](#_Toc194332012)

[**1.3.2.Tiền xử lý dữ liệu Train và Test** 14](#_Toc194332013)

[**1.3.2.1.Làm sạch dữ liệu** 14](#_Toc194332014)

[**1.3.2.2.Tiền xử lý văn bản** 15](#_Toc194332015)

[**1.3.2.3.Đọc dữ liệu từ CSV** 17](#_Toc194332016)

[**1.3.3.Tiền xử lý dữ liệu khi chạy demo** 18](#_Toc194332017)

[**1.3.3.1.Tiền xử lý dữ liệu văn bản nhập vào** 18](#_Toc194332018)

[**1.3.3.2.Tiền xử lý dữ liệu lấy từ API Twitter (API v2)** 19](#_Toc194332019)

[**1.3.3.3.Cấu hình và kết nối với Twitter API v2 để tìm dữ liệu** 20](#_Toc194332020)

[**CHƯƠNG 2. MỘT SỐ MÔ HÌNH HỌC MÁY** 21](#_Toc194332021)

[**2.1.** **Nghiên cứu và lựa chọn mô hình học máy** 21](#_Toc194332022)

[**2.1.1.Tổng quan về các mô hình** 21](#_Toc194332023)

[**2.1.2.Các mô hình Machine Learning sử dụng cho bài toán** 21](#_Toc194332024)

[**2.1.1.1.Naive Bayes** 21](#_Toc194332025)

[**2.1.1.1.1.Giới thiệu** 21](#_Toc194332026)

[**2.1.1.1.2.Thuật toán** 22](#_Toc194332027)

[**2.1.1.1.3.Ứng dụng vào bài toán** 23](#_Toc194332028)

[**2.1.1.2.** **Linear Support Vector Machine (LinearSVC)** 23](#_Toc194332029)

[**2.1.1.2.1.Giới thiệu** 23](#_Toc194332030)

[**2.1.1.2.2.Thuật toán** 24](#_Toc194332031)

[**2.1.1.2.3.Ứng dụng** 25](#_Toc194332032)

[**2.1.1.3.Logistic Regression** 26](#_Toc194332033)

[**2.1.1.3.1.Giới thiệu** 26](#_Toc194332034)

[**2.1.1.3.2.Thuật toán** 27](#_Toc194332035)

[**2.1.1.3.3.Ứng dụng** 27](#_Toc194332036)

[**2.1.1.4.Kết luận** 28](#_Toc194332037)

[**2.2.Xây dựng và cài đặt mô hình** 29](#_Toc194332038)

[**2.2.1.Naive Bayes** 29](#_Toc194332039)

[**2.2.1.1.Huấn luyện và đánh giá mô hình** 29](#_Toc194332040)

[**2.2.1.2.Tải mô hình đã huấn luyện** 34](#_Toc194332041)

[**2.2.2. Linear Support Vector Classification (LinearSVC)** 35](#_Toc194332042)

[**2.2.2.1.Huấn luyện và đánh giá mô hình** 35](#_Toc194332043)

[**2.2.2.2. Tải mô hình đã huấn luyện** 40](#_Toc194332044)

[**2.2.3. Logistic Regression** 40](#_Toc194332045)

[**2.2.3.1.Huấn luyện và đánh giá mô hình** 40](#_Toc194332046)

[**2.2.3.2.Tải mô hình đã huấn luyện** 46](#_Toc194332047)

[**2.3.Đánh giá mô hình trên tập test** 47](#_Toc194332048)

[**2.3.1.Lựa chọn độ đo** 47](#_Toc194332049)

[**2.3.1.Kết quả đánh giá các mô hình** 47](#_Toc194332050)

[**2.3.1.1.Naive Bayes** 47](#_Toc194332051)

[**2.3.1.2.LinearSVC** 48](#_Toc194332052)

[**2.3.1.3.Logistic Regression** 49](#_Toc194332053)

[**CHƯƠNG 3.THỬ NGHIỆM VÀ ĐÁNH GIÁ** 49](#_Toc194332054)

[**3.1.Điều chỉnh mô hình cải thiện hiệu năng của mô hình** 49](#_Toc194332055)

[**3.1.1.Naive Bayes** 49](#_Toc194332056)

[**3.1.2.Logistic Regression** 50](#_Toc194332057)

[**3.1.3.Linear Support Vector Classification (LinearSVC)** 51](#_Toc194332058)

[**3.2.Thực hiện dự đoán trên dữ liệu mới** 52](#_Toc194332059)

[**3.2.1.Naive Bayes** 52](#_Toc194332060)

[**3.2.2.Linear Support Vector Classification (LinearSVC)** 55](#_Toc194332061)

[**3.2.3. Logistic Regression** 57](#_Toc194332062)

# **MỤC LỤC ẢNH**

[**Hình 1: Hình ảnh mình họa sự đặc thù dữ liệu X 8**](#_Toc194340383)

[**Hình 2: Hình ảnh mô tả biểu cẩm, cảm xúc. 8**](#_Toc194340384)

[**Hình 3: Likelihood functions , Confusion matrix và ROC curve. 23**](#_Toc194340385)

[**Hình 4: Maximum-margin hyperplane và margins cho một LinearSVC được huấn luyện với các mẫu từ hai lớp 25**](#_Toc194340386)

[**Hình 5: Hàm logistic chuẩn σ(t); σ(t) ∈ (0, 1) với mọi t. 27**](#_Toc194340387)

[**Hình 6 : Hình ảnh đánh giá Naive Bayes 48**](#_Toc194340388)

[**Hình 7 : Hình ảnh đánh giá LinearSVC 49**](#_Toc194340389)

[**Hình 8 : Hình ảnh đánh giá Logistic Regression 50**](#_Toc194340390)

[**Hình 9. Đánh giá mô hình Naive Bayes sau khi điều chỉnh 51**](#_Toc194340391)

[**Hình 10. Đánh giá mô hình LogisticRegression sau khi điều chỉnh 52**](#_Toc194340392)

[**Hình 11. Đánh giá mô hình LinearSVC sau khi điều chỉnh 53**](#_Toc194340393)

[**Hình 12.Mô hình Naive Bayes thực hiện dự đoán trên dữ liệu mới 54**](#_Toc194340394)

[**Hình 13.Mô hình Naive Bayes thực hiện dự đoán trên các tweet mới qua API X 55**](#_Toc194340395)

[**Hình 14.Mô hình LinearSVC thực hiện dự đoán trên dữ liệu mới 56**](#_Toc194340396)

[**Hình 15.Mô hình LinearSVC thực hiện dự đoán trên các tweet mới qua API X 57**](#_Toc194340397)

[**Hình 16.Mô hình Logistic Regression thực hiện dự đoán trên dữ liệu mới 58**](#_Toc194340398)

# **GIỚI THIỆU**

Báo cáo tốt nghiệp này hướng đến việc xây dựng và triển khai một hệ thống phân tích cảm xúc dựa trên sự kết hợp giữa các mô hình machine learning và kỹ thuật biểu diễn từ qua vector. Mục tiêu chính của nghiên cứu là phát triển một giải pháp có thể nhận diện và phân loại cảm xúc từ các đánh giá các tweet, đặc biệt trong bối cảnh dữ liệu thực tế luôn có tính đa dạng và phức tạp. Để đạt được điều này, quá trình nghiên cứu được chia thành các giai đoạn: thu thập và xử lý dữ liệu, xây dựng mô hình, tinh chỉnh tham số, và cuối cùng là đánh giá hiệu suất của mô hình.

Việc ứng dụng mô hình machine learning cho phép hệ thống nắm bắt được thông tin ngữ cảnh của văn bản. Kết hợp với vector hóa văn bản, các từ được chuyển hóa thành các vector số học, giúp làm rõ mối quan hệ ngữ nghĩa giữa các từ, từ đó cải thiện độ chính xác trong việc phân loại các cảm xúc. Điều này cho thấy tiềm năng to lớn của việc ứng dụng machine learning trong xử lý ngôn ngữ tự nhiên.

Thông qua báo cáo này, em mong muốn đóng góp một giải pháp hiệu quả và ứng dụng được trong thực tiễn, đồng thời cung cấp một cái nhìn toàn diện về những thách thức cũng như cơ hội trong lĩnh vực phân tích cảm xúc. Kết quả nghiên cứu không chỉ mang lại giá trị về mặt lý thuyết mà còn hướng đến việc hỗ trợ các doanh nghiệp tối ưu hóa quy trình thu thập và xử lý thông tin người dùng, từ đó đưa ra những quyết định chiến lược đúng đắn.

Xin chân thành cảm ơn!

# 

# **CHƯƠNG 1.TỔNG QUAN VỀ BÀI TOÁN**

## **1.1.Đọc và phân tích bài toán**

### **1.1.1.Giới thiệu đề tài**

Trong bối cảnh Internet phát triển mạnh mẽ, Twitter trở thành nguồn dữ liệu quý giá để thu thập các ý kiến và cảm xúc của người dùng. Đề tài trên, việc tạo ra một hệ thống tự động phân loại cảm xúc (chẳng hạn: tích cực, tiêu cực, trung tính) dựa trên các tweet – những bài đăng ngắn gọn và thường mang tính thời gian thực. Bài toán này cần sự kết hợp giữa việc xử lý ngôn ngữ và học máy, đồng thời phải đối diện với nhiều khó khăn đặc thù do tính chất của dữ liệu trên Twitter. Các tweet thường chứa ngôn ngữ không chính thống, từ viết tắt, biểu tượng cảm xúc (emoji), hashtag và có thể xuất hiện những biểu hiện mỉa mai hay châm biếm, khiến cho việc xác định cảm xúc trở nên phức tạp hơn.

### **1.1.2.Tính đặc thì của dữ liệu**

* Độ ngắn gọn:

Mỗi tweet có giới hạn 280 ký tự, do đó thông tin truyền tải thường rất “chắt lọc”. Sự ngắn gọn này giúp tweet trở nên súc tích nhưng lại làm giảm bớt bối cảnh và thông tin nền cần thiết để hiểu rõ toàn bộ ý nghĩa.

* Ngôn ngữ không chính thống:

Người dùng Twitter có xu hướng sáng tạo ngôn ngữ theo cách riêng, sử dụng từ viết tắt, chữ lóng, kết hợp emoji và hashtag. Nhưng lại chứa đựng nhiều cảm xúc và ý nghĩa ẩn sau. Điều này đòi hỏi phải xử lý “thông minh” hơn so với các bài toán xử lý văn bản, text truyền thống.

Ví dụ, một tweet có thể chỉ là “OMG! Thật là #điên\_đảo 😱”

* Dữ liệu nhiễu:

Không phải tweet nào cũng mang tính cảm xúc rõ ràng. Ngoài những phản hồi chân thật từ người dùng, trên Twitter còn tồn tại rất nhiều thông tin không liên quan như tin quảng cáo, spam hoặc những tweet mang tính chất tự động phát. Việc lọc ra dữ liệu có giá trị và loại bỏ “tiếng ồn” là bước đầu tiên nhưng không kém phần quan trọng.

* Tính thay đổi liên tục:

Ngôn ngữ và xu hướng trên Twitter luôn trong trạng thái “biến động”, với các từ khóa, hashtag hay emoji mới xuất hiện liên tục. Ví dụ, một biểu tượng cảm xúc hay hashtag có thể trở nên “hot” trong vài giờ đồng hồ, rồi nhanh chóng lạc hậu. Do đó, hệ thống phân tích cần được cập nhật thường xuyên để không bỏ sót những thay đổi này.

* Đa dạng về chủ đề và ngôn ngữ:

Tweet có thể nói về chính trị, thể thao, giải trí hay đời sống cá nhân. Hơn nữa, chúng được viết bằng nhiều ngôn ngữ của các nước. Việc này khiến cho hệ thống cần có khả năng xử lý không chỉ về mặt nội dung mà còn về đa dạng văn hóa và ngôn ngữ, nhằm không bị lệch hướng khi phân tích cảm xúc.



Hình 1: Hình ảnh mình họa sự đặc thù dữ liệu X

### **1.1.3.Mô tả bài toán Sentiment Analysis**

Bài toán phân tích cảm xúc (Sentiment Analysis) trên Twitter với mục đích xây dựng một hệ thống tự động nhận diện, trích xuất và phân loại cảm xúc từ văn bản. Cụ thể, hệ thống cần phân biệt được các cảm xúc cơ bản như:



Hình 2: Hình ảnh mô tả biểu cẩm, cảm xúc.

* Tích cực: Phản ánh sự hài lòng, yêu thích, vui vẻ.
* Tiêu cực: Phản ánh sự không hài lòng, buồn bã, giận dữ.
* Trung lập: Các nội dung không mang cảm xúc rõ ràng .

Quá trình này bao gồm nhiều bước chính:

* Thu thập dữ liệu:

Bắt đầu từ việc sử dụng API của Twitter để “lấy mẫu” các tweet theo các tiêu chí như từ khóa, hashtag hoặc chủ đề đang được quan tâm. Quá trình này không chỉ nhằm thu thập lượng dữ liệu lớn mà còn phải đảm bảo sự đa dạng về nội dung và phong cách viết.

* Tiền xử lý dữ liệu:

Đây là bước “làm sạch” dữ liệu để giảm bớt các yếu tố gây nhiễu. Công việc ở đây bao gồm:

* + Loại bỏ các đường link, ký tự đặc biệt, từ “RT” của các retweet.
  + Chuyển đổi toàn bộ văn bản về dạng chữ thường và chuẩn hóa cách viết.
  + Xử lý emoji và hashtag: thay vì bỏ qua, ta cần chuyển các biểu tượng này thành từ ngữ có thể phản ánh cảm xúc (ví dụ “😊” thành “vui vẻ”).
  + Tách từ và xử lý cú pháp để chuẩn bị cho bước trích xuất đặc trưng.
* Trích xuất đặc trưng:

Sau khi dữ liệu được làm sạch, nhiệm vụ tiếp theo là chuyển các tweet từ dạng văn bản sang dạng vector, giúp máy tính “hiểu” được mối quan hệ giữa các từ. Các kỹ thuật từ truyền thống như Bag-of-Words hay TF-IDF có thể được sử dụng, tuy nhiên với dữ liệu tweet ngắn gọn và chứa nhiều yếu tố biểu cảm, các mô hình nhúng từ như Word2Vec, GloVe hay BERT sẽ cho kết quả tốt hơn với khả năng nắm bắt ngữ cảnh và sắc thái của từ ngữ.

* Xây dựng và huấn luyện mô hình:

Với dữ liệu đã được thể hiện theo dạng vector, tiến hành xây dựng mô hình phân loại. Có thể áp dụng các thuật toán machine learning truyền thống như Logistic Regression, SVM hay Naive Bayes, hoặc chuyển sang các mô hình deep learning như RNN, LSTM, CNN – đặc biệt là các kiến trúc dựa trên Transformer, vốn đã được chứng minh hiệu quả trong việc xử lý ngôn ngữ tự nhiên. Mục tiêu cuối cùng là cho hệ thống biết cách “đọc” một tweet và gán nhãn cảm xúc chính xác, kể cả những trường hợp khó như mỉa mai hay châm biếm.

* Đánh giá và tối ưu hóa:

Để đảm bảo có thể hoạt động tốt trên dữ liệu thực tế, các chỉ số như độ chính xác, độ nhạy, F1-score được sử dụng để đo lường hiệu quả. Ngoài ra, quá trình tinh chỉnh tham số và sử dụng các kỹ thuật kiểm tra chéo cũng rất quan trọng để tránh bị overfitting và nâng cao khả năng tổng thể của mô hình.

### **1.1.4.Ứng dụng của bài toán**

* Marketing và Quản lý thương hiệu:

Doanh nghiệp hoắc người dùng có thể sử dụng kết quả phân tích để nắm bắt phản hồi của khách hàng về dịch vụ, sản phẩm hoặc chiến dịch quảng cáo. Qua đó, các công ty có thể đánh giá hiệu quả của chiến lược truyền thông, nhận diện sớm những vấn đề tiêu cực và điều chỉnh chiến lược một cách kịp thời. Việc theo dõi cảm xúc khách hàng trên Twitter còn giúp công ty tạo ra các chiến dịch marketing đúng đắn, hướng đến việc xây dựng hình ảnh thương hiệu tốt hơn trên thị trường.

* Dự báo xu thế xã hội và phân tích dư luận công chúng:

Các tweet thường phản ánh tâm trạng và suy nghĩ của cộng đồng về các sự kiện nóng hổi trong xã hội. Phân tích cảm xúc từ các tweet giúp các nhà nghiên cứu, nhà hoạch định chính sách có thể dự báo xu hướng dư luận, từ đó đưa ra những quyết định phù hợp trong quản lý xã hội, an ninh quốc gia hay các chiến lược truyền thông đại chúng.

* Hỗ trợ ra quyết định trong chính sách công:

Các cơ quan chức năng, khi hiểu rõ cảm xúc và phản ứng của người dân về các chính sách, sự kiện hoặc vấn đề xã hội, sẽ có thêm thông tin để điều chỉnh, cải thiện các chính sách trong nỗ lực đạt được sự phát triển bền vững và ổn định xã hội. Việc phân tích cảm xúc giúp các nhà lãnh đạo nhận diện được những mối lo ngại, kỳ vọng của người dân một cách nhanh và chính xác.

* Ứng dụng trong lĩnh vực tài chính và đầu tư:

Tâm lý thị trường có thể được theo dõi bằng cách phân tích cảm xúc từ tweet. Ví dụ, các nhà đầu tư và các công ty chứng khoán có thể sử dụng thông tin này để dự báo khuynh hướng của thị trường, từ đó quyết định mua bán hợp lý dựa trên cảm nhận chung của cộng đồng nhà đầu tư.

* Cải thiện chất lượng dịch vụ:

Trong thời đại số, việc theo dõi và phản hồi kịp thời các đánh giá, phản hồi của khách hàng là điều vô cùng quan trọng. Hệ thống phân tích cảm xúc tự động giúp doanh nghiệp phát hiện sớm các phản hồi tiêu cực, qua đó đưa ra giải pháp nhanh chóng nhằm duy trì mối quan hệ tốt với khách hàng và cải thiện chất lượng dịch vụ.

## **1.2.Thu thập dữ liệu và phân tích dữ liệu**

### **1.2.1.Tổng quan về tập dữ liệu**

Bộ dữ liệu này được tạo ra từ các tweet thu thập qua Twitter API với tổng số khoảng 1.600.000 tweet. Mục đích chính của dataset là nghiên cứu phân loại cảm xúc, tức là tự động xác định xem một tweet mang cảm xúc tích cực hay tiêu cực. Một điểm đặc biệt của dataset này là nguồn gốc của các nhãn (label) được xác định theo phương pháp “distant supervision”. Cụ thể:

* Các tweet có chứa biểu tượng cảm xúc tích cực (ví dụ: “:)”) được gán nhãn là tích cực (positive – giá trị 4).
* Các tweet có chứa biểu tượng cảm xúc tiêu cực (ví dụ: “:(”) được gán nhãn là tiêu cực (negative – giá trị 0).

Lưu ý: rằng mặc dù có định nghĩa cho nhãn trung lập (neutral – giá trị 2), trong thực tế dataset này chỉ gồm các tweet mang nhãn 0 và 4.

### **1.2.2.Thu thập dữ liệu**

Đây là bộ dữ liệu có sẵn trên Kaggle, Nó chứa 1.600.000 tweet được trích xuất bằng api twitter. Các tweet đã được chú thích (0 = tiêu cực, 4 = tích cực) và chúng có thể được sử dụng để phát hiện cảm xúc.

Đường dẫn: https://www.kaggle.com/datasets/kazanova/sentiment140

### **1.2.3.Phân tích dữ liệu**

Mỗi dòng dữ liệu (tương ứng với một tweet) chứa 6 trường thông tin, giúp cung cấp cái nhìn tổng thể về nguồn gốc cũng như nội dung của tweet đó. Cụ thể:

* target (Nhãn Cảm Xúc):

Trường này biểu diễn cực tính của tweet, với giá trị 0 cho cảm xúc tiêu cực và 4 cho cảm xúc tích cực. Nó được xây dựng dựa trên sự hiện diện của các biểu tượng cảm xúc trong tweet.

* 0: tweet mang cảm xúc tiêu cực.
* 4: tweet mang cảm xúc tích cực.

(Lưu ý: mặc dù có quy định cho giá trị 2 – trung lập, nhưng bộ dữ liệu này không chứa nhãn trung lập.)

* ids (ID Tweet):

Đây là một mã số duy nhất cho mỗi tweet trên Twitter. Sự duy nhất của ID giúp dễ dàng nhận dạng và truy xuất tweet từ nguồn gốc ban đầu trên Twitter.

* date (Ngày Tháng):

Trường này ghi lại thời điểm tweet được đăng tải. Thông tin về ngày tháng không chỉ giúp theo dõi xu hướng cảm xúc theo thời gian mà còn hỗ trợ việc phân tích các sự kiện theo chu kỳ (ví dụ: so sánh cảm xúc trong các dịp lễ, sự kiện chính trị, hay khủng hoảng xã hội).

* flag (Truy Vấn):

Trường này lưu trữ thông tin về truy vấn tìm kiếm được sử dụng khi lấy tweet. Trong trường hợp bộ dữ liệu này, nếu không có truy vấn cụ thể, giá trị mặc định sẽ là “NO\_QUERY”. Điều này cho thấy rằng các tweet được thu thập không dựa trên một từ khóa hay cụm từ tìm kiếm cụ thể nào, mà chủ yếu là thông qua các biểu tượng cảm xúc.

* user (User):

Đây là tên người dùng (username) đã đăng tweet. Thông tin này hữu ích để xác định nguồn gốc của tweet, qua đó có thể phân tích mức độ đóng góp của từng người dùng hoặc nhóm người dùng vào tổng thể dữ liệu.

* text (Nội Dung Tweet):

Đây là nội dung của tweet, tức là bài viết mà người dùng đăng tải. Mặc dù tweet có giới hạn ký tự, nhưng nội dung này chứa đựng những thông tin cảm xúc chính yếu và là đối tượng chính để thực hiện các bước xử lý ngôn ngữ tự nhiên và phân loại cảm xúc.

### **1.2.4.Thống kê dữ liệu**

Tổng dữ liệu có chứa gần 1.600.000 tweets, một số thống kê sơ bộ:

* target:
  + Dataset chỉ có hai nhãn chính: 0 (negative) và 4 (positive).
  + Mặc dù hệ thống ban đầu có thể định nghĩa thêm nhãn trung lập (2), nhưng do cách thu thập dựa trên biểu tượng cảm xúc, các tweet trung lập hầu như không có trong dataset.
* ids:
  + Các tweet có ID khác nhau trong khoảng từ 1.47x.xxx.xxx đến 2.33x.xxx.xxx. Điều này cho thấy sự phong phú và liên tục trong quá trình thu thập dữ liệu từ Twitter.
* date:
  + Số lượng khoảng 774.362 giá trị khác nhau trong trường date chứng tỏ rằng dataset được thu thập qua nhiều thời điểm khác nhau.
  + Điều này hữu ích khi xem xét xu hướng theo thời gian, giúp xác định mốc thời gian có sự biến đổi cảm xúc đáng chú ý.
* flag:
  + Với duy nhất một giá trị “NO\_QUERY”, điều này xác nhận rằng không có truy vấn cụ thể nào khác được sử dụng ngoài tiêu chí biểu tượng cảm xúc, tạo nên một tập dữ liệu thuần túy.
* user:
  + có 659.775 người dùng khác nhau cho thấy dữ liệu không chỉ đến từ một vài cá nhân mà thực sự đại diện cho một cộng đồng đa dạng.
  + Điều này giúp giảm thiểu hiện tượng thiên lệch do quá nhiều tweet đến từ một số người dùng nhất định.
* text:
  + Với 1.581.465 giá trị duy nhất, trường text phản ánh tính đa dạng về nội dung.
  + Sự khác biệt trong các tweet đảm bảo chắc chắn rằng mô hình phân loại cảm xúc sẽ phải xử lý được nhiều kiểu ngôn ngữ và phong cách viết khác nhau, từ đó nâng cao khả năng tổng quát hóa khi sử dụng vào các dataset khác.

### **1.2.5.Phương pháp sử dụng**

Để xây dựng bộ dữ liệu này, hai cách chính đã được sử dụng:

* Distant Supervision:
  + Phương pháp này tận dụng các biểu tượng cảm xúc hiện có trong tweet để tự động gán nhãn cho dữ liệu.
  + Cách thực hiện:
* Tweet có chứa biểu tượng “:)” sẽ được gán nhãn là positive sentiment (giá trị 4).
* Tweet có chứa biểu tượng “:(” sẽ được gán nhãn là negative sentiment (giá trị 0).
  + Ưu điểm của phương pháp là tiết kiệm thời gian và nguồn lực so với việc gán nhãn thủ công. Tuy nhiên, nó cũng có hạn chế, vì không phải lúc nào sự hiện diện của một biểu tượng cảm xúc cũng phản ánh đúng cảm xúc tổng thể của tweet.
* Twitter Search API:
  + Dữ liệu được lấy thông qua Twitter Search API, bằng các từ khóa liên quan đến biểu tượng cảm xúc.
  + Phương pháp này giúp truy xuất các tweet chứa các biểu tượng cảm xúc cần thiết, đảm bảo rằng dữ liệu thu được phù hợp với mục tiêu phân loại cảm xúc.
  + Quá trình lấy dữ liệu qua API cũng giúp đảm bảo tính thời gian thực và cập nhật liên tục, mặc dù cách thu thập chủ yếu dựa vào biểu tượng cảm xúc cũng có thể dẫn đến việc bỏ sót những tweet không sử dụng biểu tượng nhưng vẫn mang thông điệp cảm xúc rõ ràng.

## **1.3.Tiền xử lý dữ liệu**

### **1.3.1.** **Lý thuyết về tiền xử lý dữ liệu**

Tiền xử lý dữ liệu văn bản giúp loại bỏ những yếu tố không hữu ích trong dữ liệu và chuẩn hóa chúng để mô hình có thể làm việc tốt hơn. Các bước chính bao gồm:

* Loại bỏ URL và ký tự đặc biệt: Các URL hoặc ký tự đặc biệt không chứa thông tin cảm xúc, cần phải loại bỏ.
* Chuyển đổi văn bản thành chữ thường: Giúp loại bỏ sự phân biệt giữa chữ hoa và chữ thường.
* Tokenization: Chia nội dung văn bản thành các đơn vị nhỏ (từ hoặc cụm từ).
* Loại bỏ Stopwords: Loại bỏ các từ không có nghĩa cụ thể, ví dụ như "the", "is", "at".
* Lemmatization: Là chuyển đổi từ về dạng gốc của nó, giúp giảm sự phức tạp trong dữ liệu.

Ngoài văn bản, những yếu tố như thời gian đăng tweet, độ dài tweet, hay số lượng hashtag có thể có thông tin hữu ích cho phân tích cảm xúc. Việc xử lý các dữ liệu này giúp mô hình hiểu được các yếu tố tác động đến cảm xúc.

### **1.3.2.Tiền xử lý dữ liệu Train và Test**

Tiền xử lý giúp cải thiện chất lượng và độ chính xác của mô hình học máy bằng cách chuẩn hóa và làm sạch dữ liệu đầu vào. Trong phần này, sẽ mô tả chi tiết các thao tác tiền xử lý văn bản trong bài toán. Các bước tiền xử lý bao gồm: làm sạch dữ liệu, loại bỏ các từ không dung cho biểu thị cảm xúc (stopwords), tách từ (tokenization), lemmatization,và vector hóa văn bản.

#### **1.3.2.1.Làm sạch dữ liệu**

Một trong những nhiệm vụ quan trọng trong tiền xử lý văn bản là làm sạch dữ liệu, bỏ các thành phần không dung đến như tài khoản người dùng, liên kết URL, và các ký tự đặc biệt. Điều này giúp giảm bớt nhiễu và giữ lại các từ quan trọng cho phân tích.

Trong hệ thống, quá trình này được thực hiện thông qua phương thức clean\_tweet(). Cụ thể, phương thức này sử dụng biểu thức chính quy (regular expressions) để thay thế các thành phần không mong muốn trong văn bản. Cụ thể:

* Các tài khoản người dùng (dấu hiệu bắt đầu bằng @ và theo sau là các ký tự alphanumeric) sẽ được thay thế bằng một khoảng trắng.
* Các liên kết URL (dấu hiệu bắt đầu bằng http:// hoặc https://) cũng sẽ bị loại bỏ.
* Ngoài ra, các ký tự đặc biệt khác chẳng hạn như chữ cái, số hoặc khoảng trắng sẽ bị loại bỏ.code của phương thức clean\_tweet() như sau:

**def clean\_tweet(self, tweet):**

**return ' '.join(**

**re.sub(r"(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)", " ", tweet).split()**

**)**

Giải thích:

* Biểu thức chính quy (@[A-Za-z0-9]+) tìm kiếm tất cả các tài khoản người dùng bắt đầu bằng @ và theo sau là các ký tự alphanumeric.
* Biểu thức ([^0-9A-Za-z \t]) : bỏ tất cả các ký tự không phải là chữ cái, số hoặc khoảng trắng.
* Biểu thức (\w+:\/\/\S+) tìm kiếm các liên kết URL bắt đầu bằng http:// hoặc https://.

#### **1.3.2.2.Tiền xử lý văn bản**

Sau khi làm sạch dữ liệu, bước tiếp theo là tiền xử lý các văn bản bằng cách chuẩn hóa và chuẩn bị dữ liệu để mô hình học máy có thể hiểu được. Các thao tác chính trong bước này bao gồm:

* Chuyển về chữ thường: Giảm sự khác biệt giữa chữ hoa và chữ thường để mô hình có thể làm việc với các từ một cách đồng nhất.
* Loại bỏ stopwords: Stopwords là những từ không mang nhiều ý nghĩa trong ngữ cảnh phân tích cảm xúc, như các từ nối, mạo từ (ví dụ: "the", "a", "and"). Việc loại bỏ các stopwords giúp giảm kích thước dữ liệu và tập trung vào các từ khóa có ý nghĩa hơn.
* Tách từ (Tokenization): Tách một câu thành các từ riêng biệt, giúp mô hình có thể phân tích chi tiết hơn các đơn vị ngữ nghĩa.
* Lemmatization: chuyển đổi các từ về dạng cơ bản của chúng (lemma). Ví dụ: từ "running" sẽ được chuyển thành "run", giúp giảm thiểu sự đa dạng của từ vựng.

Các thao tác này được thực hiện trong hàm preprocess\_text():

**def preprocess\_text(text, use\_lemmatization=True):**

**text = text.lower() # Chuyển văn bản về chữ thường**

**stop\_words = set(stopwords.words('english')) # Lấy stopwords**

**words = word\_tokenize(text) # Tách câu thành từ**

**words = [word for word in words if word not in stop\_words]**

**if use\_lemmatization:**

**lemmatizer = WordNetLemmatizer()**

**words = [lemmatizer.lemmatize(word) for word in words]**

**return' '.join(words) # Trả về văn bản đã tiền xử lý**

Giải thích:

* text.lower() chuyển tất cả các ký tự trong văn bản thành chữ thường.
* word\_tokenize(text) sử dụng thư viện nltk để tách câu thành các từ riêng biệt.
* stopwords.words('english') lấy danh sách các từ dừng (stopwords) từ thư viện nltk.
* lemmatizer.lemmatize(word) sử dụng lemmatization của thư viện nltk để giảm từ về dạng cơ bản.

**\*Tách từ (Tokenization)**

Mục đích của tách từ là biến văn bản thành các thành phần có thể xử lý được, tạo ra các từ hoặc cụm từ (tokens) mà mô hình có thể sử dụng để học và phân tích. Tách từ giúp mô hình nhận diện được từng từ và quan hệ giữa các từ trong một câu. Đặc biệt trong ngữ cảnh phân tích cảm xúc, từng từ (hoặc cụm từ) có thể mang ý nghĩa riêng biệt và có thể quyết định cảm xúc của câu.

sử dụng hàm word\_tokenize() từ thư viện nltk để tách từ:

**from nltk.tokenize import word\_tokenize**

**words = word\_tokenize(text**

word\_tokenize(text): Hàm này nhận vào một văn bản (dạng chuỗi) và tách nó thành một danh sách các từ. Đây là một bước rất quan trọng vì mô hình học máy cần phải làm việc với các từ riêng biệt trong văn bản, không phải toàn bộ câu

Ví dụ, ta có câu sau trong dữ liệu tweet:

"Love the new iPhone! #Amazing"

Khi áp dụng phương pháp “word\_tokenize”, kết quả sẽ là:

['Love', 'the', 'new', 'iPhone', '!', '#', 'Amazing']

Giải thích:

* Từ đơn lẻ: Mỗi từ trong câu đã được tách riêng biệt.

“ Love, the, new, iPhone, Amazing. ”

* Dấu câu và hashtag: Dấu câu như “ ! ” và các ký hiệu như “ # “ cũng được tách riêng thành các tokens riêng biệt

**\*Xử lý kết quả Tokenization**

**- Loại bỏ stopwords:** Các từ dừng như "the", "a", "an" không mang nhiều thông tin và sẽ được loại bỏ trong quá trình tiền xử lý.

Từ VD bên trên : ['love', 'new', 'iphone', 'amazing']

Giải thích:

- Như vậy, đã loại bỏ các từ không mang nhiều ý nghĩa như "the" và chuẩn hóa các từ về dạng cơ bản.

#### **1.3.2.3.Đọc dữ liệu từ CSV**

Dữ liệu tweet được lấy từ file CSV, trong đó mỗi dòng chứa các thông tin như cảm xúc (sentiment), ID, ngày, người dùng, và nội dung tweet. Hàm load\_tweets\_from\_csv() được sử dụng để đọc dữ liệu từ file CSV, chuẩn hóa nhãn cảm xúc và trích xuất các tweet cần thiết.

Hàm này thực hiện các bước sau:

* Đọc dữ liệu của file CSV với định dạng được chỉ định và gán tên cột cho dữ liệu.
* Chuyển nhãn cảm xúc từ giá trị 0 thành -1 (tiêu cực) và từ 4 thành 1 (tích cực).
* Trích xuất các tweet và nhãn cảm xúc vào danh sách các đối tượng tweet.

Các thao tác này được thực hiện trong hàm load\_tweets\_from\_csv(file\_path):

**def load\_tweets\_from\_csv(file\_path):**

**try:**

**df = pd.read\_csv(file\_path, encoding='latin1', header=None)**

**df.columns = ["sentiment", "id", "date", "query", "user", "text"]**

**df["sentiment"] = df["sentiment"].replace({0: -1, 4: 1})**

**tweets = []**

**for index, row in df.iterrows():**

**parsed\_tweet = {}**

**parsed\_tweet['text'] = row['text']**

**parsed\_tweet['sentiment'] = row['sentiment']**

**tweets.append(parsed\_tweet)**

**return tweets**

**except Exception as e:**

**print(f"Lỗi khi đọc file CSV: {e}")**

**return None**

Giải thích chi tiết:

* pd.read\_csv(file\_path, encoding='latin1', header=None) đọc dữ liệu từ file CSV. Tham số header=None chỉ ra rằng file không có dòng tiêu đề, và encoding='latin1' giúp xử lý các ký tự đặc biệt trong dữ liệu.
* df.columns gán tên cho các cột của DataFrame.
* df["sentiment"].replace({0: -1, 4: 1}) chuyển đổi các nhãn cảm xúc từ 0 (tiêu cực) thành -1 và từ 4 (tích cực) thành 1.

### **1.3.3.Tiền xử lý dữ liệu khi chạy demo**

#### **1.3.3.1.Tiền xử lý dữ liệu văn bản nhập vào**

1. Chuyển về chữ thường (Lowercasing)

- Văn bản được chuyển thành chữ thường để đảm bảo không có sự phân biệt giữa chữ hoa và chữ thường khi phân tích.

Ví dụ: "Happy" và "happy" sẽ được coi là giống nhau.

2. Tách từ (Tokenization)

- Dùng nltk.word\_tokenize để chia văn bản thành các từ riêng biệt (tokens).

Ví dụ: "I love Python!" sẽ thành ["I", "love", "Python"].

3. Loại bỏ Stop Words

- Stop words là các từ không mang nhiều ý nghĩa như "và", "của", "the", "is".

- Thư viện nltk.corpus.stopwords sẽ giúp loại bỏ các từ này từ văn bản.

4. Lemmatization (Đưa từ về dạng gốc)

- Dùng nltk.stem.WordNetLemmatizer để biến các từ thành dạng gốc của chúng (ví dụ: "running" → "run", "better" → "good").

- Việc này giúp giảm thiểu số lượng từ và cải thiện kết quả phân tích.

Hàm xử lý chính:

**def preprocess\_text(text, use\_lemmatization=True):**

**text = clean\_tweet(text) # Clean tweet first**

**text = text.lower()**

**sstop\_words = set(stopwords.words('english'))**

**wordss = word\_tokenize(text)**

**wordss = [word for word in words if word.isalnum() and word not in stop\_words]**

**if use\_lemmatization:**

**lemmatizer = WordNetLemmatizer()**

**wordss = [lemmatizer.lemmatize(word) for word in words]**

**return ' '.join(wordss)**

#### **1.3.3.2.Tiền xử lý dữ liệu lấy từ API Twitter (API v2)**

- Khi lấy văn bản từ Twitter API v2, cũng cần thực hiện các thao tác tiền xử lý tương tự, nhưng đối với các tweet lấy từ mạng xã hội, việc này cần làm để loại bỏ các yếu tố không liên quan và làm cho văn bản dễ phân tích.

1. Loại bỏ Mentions (Tài khoản Twitter)

- Các mentions (được biểu thị bằng ký tự @) không mang nhiều ý nghĩa trong việc phân tích cảm xúc, do đó chúng sẽ bị loại bỏ.

2. Loại bỏ URL

- Các liên kết (URLs) không cần thiết cho việc phân tích cảm xúc, vì vậy chúng cũng được loại bỏ.

3.Bỏ ký tự đặc biệt

- Những ký tự đặc biệt, như dấu câu hay ký tự không phải chữ, sẽ được loại bỏ.

#### **1.3.3.3.Cấu hình và kết nối với Twitter API v2 để tìm dữ liệu**

- Để kết nối và sử dụng Twitter API v2, cần phải có Bearer Token. Mã này phải được lưu trong biến môi trường (.env) và được gọi đến trong chương trình.

- Lưu Bearer Token trong biến môi trường: Để bảo mật, token không nên được cứng mã hóa trong mã nguồn mà nên lưu trữ trong các biến môi trường.

Code Sử dụng:

**from dotenv import load\_dotenv**

**load\_dotenv()**

* Biến BEARER\_TOKEN trong file .env chứa token cần thiết để truy cập vào Twitter API

Code sử dựng:

**CONSUMER\_KEY=" YOUR\_API\_KEY "**

**CONSUMER\_SECRET=" YOUR\_API\_SECRET\_KEY "**

**ACCESS\_TOKEN=" YOUR\_ACCESS\_TOKEN "**

**ACCESS\_TOKEN\_SECRET=" YOUR\_ACCESS\_TOKEN\_SECRET "**

**BEARER\_TOKEN=" YOUR\_TOKEN "**

* Tạo client cho API Twitter: Sau khi lấy được Bearer Token từ biến môi trường, sẽ sử dụng tweepy.Client để tạo kết nối với API.

Code sử dụng:

**bearer\_token = os.getenv("BEARER\_TOKEN")**

**if not bearer\_token:**

**st.error("Vui lòng thiết lập biến môi trường cho Twitter Bearer Token.")**

**return**

**api\_client = tweepy.Client(bearer\_token=bearer\_token)**

# 

# **CHƯƠNG 2. MỘT SỐ MÔ HÌNH HỌC MÁY**

## **2.1.** **Nghiên cứu và lựa chọn mô hình học máy**

### **2.1.1.Tổng quan về các mô hình**

* Naive Bayes:

Naive Bayes là một trong những thuật toán học máy đơn giản và hiệu quả, thường được sử dụng trong các bài toán phân loại văn bản, trong đó có phân tích cảm xúc. Đối với đề tài trên Naive Bayes có thể giúp xác định xem một tweet mang tính chất cảm xúc tích cực hay tiêu cực.

* Support Vector Machine (SVM):

Thuật toán học máy mạnh mẽ Support Vector Machine (SVM) được sử dụng rộng rãi trong các bài toán phân loại, chẳng hạn như phân tích cảm xúc trong văn bản. SVM hoạt động nhằm mục đích phân loại tốt nhất các điểm dữ liệu thuộc các lớp khác nhau bằng cách tìm ra một siêu phẳng (hyperplane) tối ưu trong không gian các đặc trưng của dữ liệu.

* Logistic Regression:

Logistic Regression, dựa trên các đặc điểm văn bản như từ ngữ, cú pháp và ngữ cảnh, có thể phân loại các tweet thành các nhóm cảm xúc khác nhau (tích cực, tiêu cực, trung lập). Hồi quy logistic giúp hiểu rõ hơn về cách cảm xúc được thể hiện trong các bài viết ngắn, đặc biệt là trên các nền tảng mạng xã hội.

### **2.1.2.Các mô hình Machine Learning sử dụng cho bài toán**

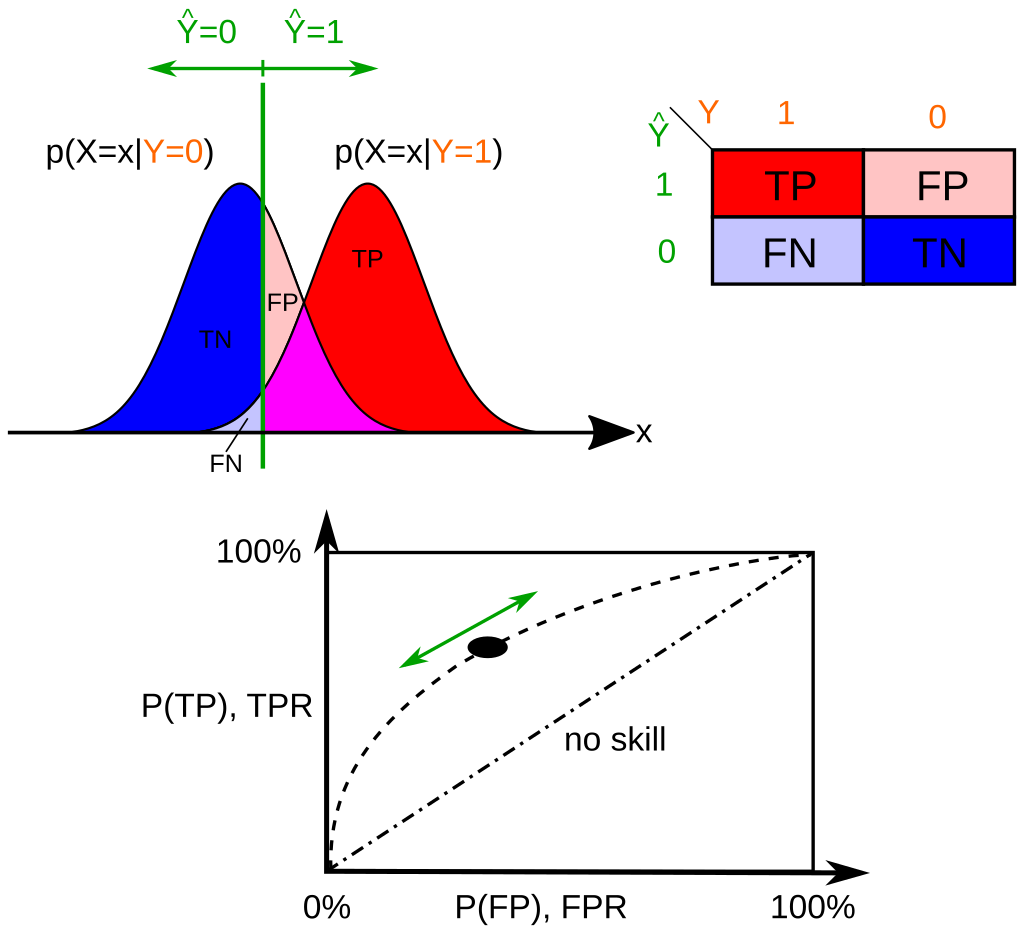
#### **2.1.1.1.Naive Bayes**

##### **2.1.1.1.1.Giới thiệu**

Naive Bayes là một phương pháp đơn giản được sử dụng để tạo ra các bộ phân loại, trong đó các đối tượng được biểu diễn dưới dạng các vector chứa các giá trị đặc trưng và được gán nhãn từ một tập hợp hữu hạn. Thay vì có một thuật toán duy nhất, phương pháp này gồm một loạt các thuật toán dựa trên nguyên tắc chung: giả định rằng, với điều kiện nhãn đã biết, giá trị của mỗi đặc trưng là độc lập với giá trị của những đặc trưng khác.

Trong thực tế, để ước lượng các tham số của mô hình Naive Bayes, thường áp dụng phương pháp tối đa hóa khả năng, cho phép làm việc với mô hình này mà không cần dựa vào lý thuyết xác suất Bayes hay các phương pháp liên quan. Mặc dù thiết kế của nó khá đơn giản và dựa trên những giả định có thể coi là quá tối giản, nhưng Naive Bayes vẫn cho thấy hiệu quả trong nhiều trường hợp phức tạp. Nghiên cứu vào năm 2004 đã đưa ra những cơ sở lý thuyết giải thích cho hiệu suất ấn tượng của các bộ phân loại này, mặc dù so với các thuật toán hiện đại như cây quyết định tăng cường hay rừng ngẫu nhiên, Naive Bayes có thể không luôn đạt được kết quả tốt nhất.

Một ưu điểm của phương pháp này là chỉ cần một lượng dữ liệu huấn luyện tương đối nhỏ để ước tính các tham số cần thiết cho việc phân loại.

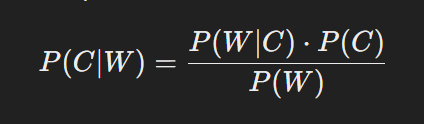


Hình 3: [Likelihood functions](https://en.wikipedia.org/wiki/Likelihood_function) , [Confusion matrix](https://en.wikipedia.org/wiki/Confusion_matrix) và [ROC curve](https://en.wikipedia.org/wiki/ROC_curve).

##### **2.1.1.1.2.Thuật toán**

Naive Bayes là một mô hình phân loại dựa trên định lý Bayes. Để hiểu cách thuật toán hoạt động, xét một tập hợp các thuộc tính (từ vựng) trong một văn bản. Giả sử có một văn bản cần phân loại, muốn tính toán xác suất của lớp **C** (ví dụ: cảm xúc tích cực hoặc tiêu cực) dựa trên các từ trong văn bản, ký hiệu là: **W** = {**w**1,**w**2,…,**w**n)

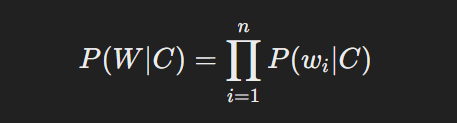
Áp dụng định lý Bayes, xác suất của lớp C được tính như sau:



Trong đó :

* **P(C ∣ W)** là xác suất của lớp **C** khi đã biết các từ **W**.
* **P(W∣C)** là xác suất của lớp **W** khi đã biết các từ **C**.
* **P**(**C)** là xác suất của lớp **C**
* **P**(**W)** là xác suất của từ **W** (thường không thay đổi khi so sánh các lớp).

Thuật toán đơn giản hóa việc tính **P(W∣C)** bằng cách giả định các từ là độc lập với nhau. Do đó, xác suất **P(W∣C)** có thể được tính bằng việc nhân xác suất của từng từ riêng biệt:



Với phương trình này, thuật toán có thể tính toán xác suất cho từng lớp và chọn lớp có xác suất cao nhất làm dự đoán.

##### **2.1.1.1.3.Ứng dụng vào bài toán**

Khi áp dụng Naive Bayes vào bài toán phân tích cảm xúc từ các tweet, quy trình bắt đầu bằng việc lấy các tweet và tiền xử lý dữ liệu. Các tweet sẽ được làm sạch (loại bỏ ký tự đặc biệt, viết tắt,...), sau đó tách thành các từ riêng. Mỗi từ sẽ dùng để xây dựng các đặc trưng cho mô hình. Sau khi huấn luyện trên một tập dữ liệu gắn nhãn (tích cực, tiêu cực, trung lập), mô hình Naive Bayes sẽ tính toán xác suất của các lớp cảm xúc và phân loại cảm xúc của các tweet mới.

Một ví dụ đơn giản: khi có một tweet như "Tôi yêu sản phẩm này! Tuyệt vời!", mô hình sẽ xác định từ "yêu" và "tuyệt vời" là các từ có xác suất cao liên quan đến cảm xúc tích cực, do đó tweet này sẽ được phân loại là cảm xúc tích cực. Ngược lại, với tweet "Sản phẩm này thật sự tệ, rất thất vọng", từ "tệ" và "thất vọng" sẽ có xác suất cao thuộc về cảm xúc tiêu cực.

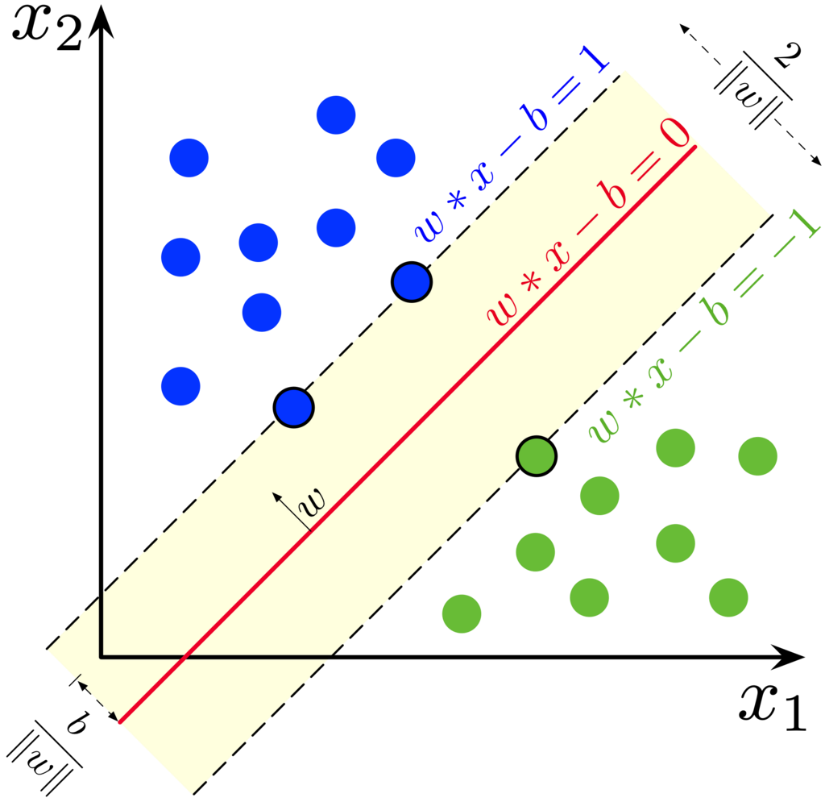
Với Naive Bayes, việc phân tích cảm xúc trở nên đơn giản và nhanh chóng, đặc biệt khi đối mặt với số lượng tweet lớn. Mặc dù có những hạn chế về giả thuyết độc lập giữa các từ, Với đặc tính đơn giản và khả năng xử lý dữ liệu không chuẩn, mô hình Naive Bayes vẫn thể hiện được hiệu quả nổi bật trong việc nhận diện cảm xúc từ các tweet trên Twitter

#### **2.1.1.2.** **Linear Support Vector Machine (LinearSVC)**

##### **2.1.1.2.1.Giới thiệu**

Máy hỗ trợ vectơ (LinearSVC) thực hiện việc tạo ra một hay nhiều siêu phẳng trong không gian có nhiều chiều (hoặc thậm chí vô hạn chiều) để giải quyết các bài toán phân loại, hồi quy và các nhiệm vụ khác. Ý tưởng chính là càng đẩy các siêu phẳng xa các điểm dữ liệu từ các lớp khác nhau (điều này gọi là "biên" càng rộng), thì khả năng tổng quát hóa của thuật toán sẽ càng tốt, tức là sai số giảm.

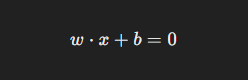
Trong nhiều trường hợp, việc phân chia tuyến tính các lớp dữ liệu trong không gian ban đầu là không thể. Do đó, thường cần ánh xạ các điểm dữ liệu từ không gian ban đầu vào một không gian mới có số chiều cao hơn để dễ dàng phân tách. Để đảm bảo tính hiệu quả trong quá trình tính toán, ánh xạ này được thiết kế sao cho việc tính tích vô hướng giữa các vectơ ở không gian mới có thể thực hiện nhanh chóng chỉ dựa trên tọa độ ban đầu.



Hình 4: Maximum-margin hyperplane và margins cho một LinearSVC được huấn luyện với các mẫu từ hai lớp

##### **2.1.1.2.2.Thuật toán**

Để thực hiện việc phân loại, LinearSVC tìm cách xác định một siêu phẳng tối ưu sao cho khoảng cách từ các điểm dữ liệu đến siêu phẳng này đạt mức lớn nhất. Công thức biểu diễn của siêu phẳng có thể được trình bày như sau:

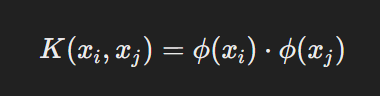


Trong đó:

* **w** là vector trọng số của siêu phẳng.
* **x** là vector đặc trưng của điểm dữ liệu.
* **b** là độ dịch chuyển của siêu phẳng.

Để tối ưu hóa siêu phẳng (hyperplane), LinearSVC sử dụng thuật toán tối thiểu hóa hàm mất mát, với mục tiêu tìm ra một giá trị **w** và **b** sao cho khoảng cách giữa các điểm dữ liệu gần siêu phẳng nhất.

Khi dữ liệu không thể phân loại tuyến tính, LinearSVC sẽ áp dụng các hàm kernel như Radial Basis Function (RBF) hay Polynomial để chuyển đổi không gian dữ liệu và tìm siêu phẳng phân tách trong không gian mới. Mô hình này có thể được biểu diễn như sau:



Trong đó:

* **K(xi,xj)** là hàm kernel
* **ϕ(xi)** là hàm ánh xạ dữ liệu vào không gian cao chiều.

##### **2.1.1.2.3.Ứng dụng**

Trong bài toán phân tích cảm xúc của văn bản, LinearSVC có thể được áp dụng để phân loại các tweet thành các nhóm cảm xúc như tích cực, tiêu cực hoặc trung lập. Các tweet trên Twitter thường có đặc điểm là ngắn gọn, chứa nhiều từ ngữ lóng và có sự biến đổi lớn trong ngữ nghĩa, điều này tạo ra thử thách lớn cho các mô hình phân tích cảm xúc.

Để áp dụng LinearSVC trong bài toán này, quá trình đầu tiên là tiền xử lý dữ liệu, trong đó các tweet sẽ được chuyển đổi thành các đặc trưng có thể sử dụng được cho mô hình LinearSVC, chẳng hạn như tần suất xuất hiện của các từ khóa (TF-IDF), các đặc trưng từ nhãn cảm xúc, hay từ khóa liên quan đến cảm xúc.

Tiếp theo, mô hình LinearSVC sẽ được huấn luyện trên bộ dữ liệu các tweet đã được gán nhãn cảm xúc. Sau khi huấn luyện, mô hình sẽ có khả năng phân loại cảm xúc của các tweet mới, giúp các công cụ phân tích cảm xúc tự động nhận diện thái độ của người dùng đối với các sự kiện, sản phẩm hoặc dịch vụ được đề cập trong các tweet.

Với các khả năng mạnh mẽ của LinearSVC, đặc biệt là khả năng làm việc hiệu quả với dữ liệu nhiều chiều và phi tuyến tính, mô hình này có thể cung cấp kết quả phân tích cảm xúc chính xác và nhanh chóng, hỗ trợ các doanh nghiệp và tổ chức trong việc hiểu rõ hơn về phản hồi của người dùng trên Twitter.

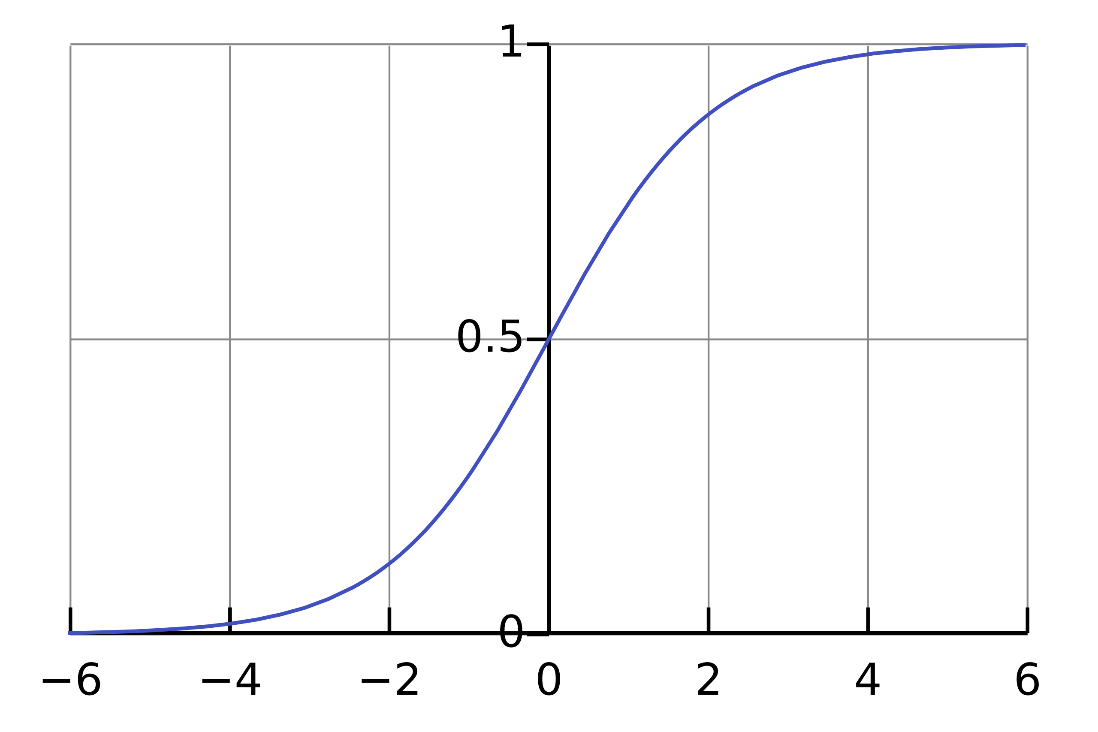
#### **2.1.1.3.Logistic Regression**

##### **2.1.1.3.1.Giới thiệu**

Logistic Regression là một thuật toán học máy mà các phương pháp học có giám sát thường được áp dụng rộng rãi trong các nhiệm vụ phân loại nhị phân.chẳng hạn như xác định xem một email có phải là thư rác hay không, hoặc chẩn đoán bệnh dựa trên kết quả xét nghiệm của bệnh nhân. Mô hình này sử dụng hàm logistic (hay còn gọi là hàm sigmoid) để chuyển đổi tổ hợp tuyến tính của các đặc trưng đầu vào thành một giá trị xác suất nằm trong khoảng từ 0 đến 1. Xác suất này cho biết khả năng của đầu vào thuộc về 1 trong 2 nhóm đã được định nghĩa trước.

Logistic regression đã được áp dụng rộng rãi trong nhiều lĩnh vực như y tế, xã hội học, marketing và kỹ thuật. Một trong những ví dụ điển hình là việc sử dụng logistic regression trong việc xác định tỷ lệ tử vong ở bệnh nhân chấn thương, cũng như dự đoán nguy cơ phát triển các bệnh lý như tiểu đường hay bệnh tim mạch. Trong bối cảnh mạng xã hội, logistic regression được sử dụng để phân tích cảm xúc trong bài viết, chẳng hạn như phân loại các tweet thành cảm xúc tích cực hoặc tiêu cực.

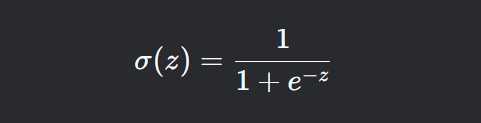
Nhờ vào tính chất chuyển đổi đầu ra thành xác suất giữa 0 và 1, logistic regression là một công cụ rất hiệu quả trong các bài toán phân loại nhị phân, cung cấp một khuôn khổ xác suất để hỗ trợ các quyết định có cơ sở



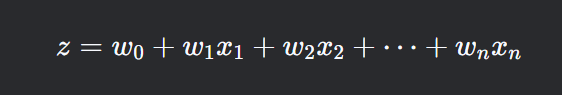
Hình 5: Hàm logistic chuẩn σ(t); σ(t) ∈ (0, 1) với mọi t.

##### **2.1.1.3.2.Thuật toán**

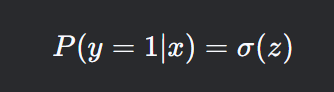
Hàm logistic là một hàm sigmoid, nhận bất kỳ giá trị thực nào **t** và trả về một giá trị nằm giữa 0 và 1. Đối với logit, điều này được hiểu là nhận đầu vào là log-odds và đầu ra xác suất. Hàm logistic chuẩn σ: **R → (0, 1)** được định nghĩa như sau:



Trong đó, **z** là giá trị đầu vào được tính bằng phương trình tuyến tính:

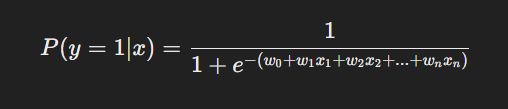


Ở đây, **w0,w1,…,wn** là các trọng số (weights) của mô hình, và **x1,x2,…,xn** là các đặc trưng đầu vào. Đầu ra của mô hình là xác suất **P(y=1∣x),** được tính bằng:



Trong quá trình huấn luyện, mô hình được tối ưu hóa thông qua việc sử dụng hàm mất mát (ví dụ: Cross-Entropy Loss) nhằm đánh giá độ lệch giữa dự đoán và giá trị thật. Quá trình điều chỉnh các trọng số được thực hiện bằng các thuật toán tối ưu như Gradient Descent để giảm thiểu hàm mất mát xuống mức thấp nhất có thể..

Cụ thể:



##### **2.1.1.3.3.Ứng dụng**

* Tiền xử lý dữ liệu

Thu thập dữ liệu từ Twitter, gồm các tweet và nhãn cảm xúc tương ứng (ví dụ: tích cực, tiêu cực, trung tính).

Tiến hành làm sạch dữ liệu bằng cách bỏ các ký tự đặc biệt, chuyển đổi văn bản về dạng chữ thường, và loại bỏ các stop words như "the", "or", "is".

Sử dụng kỹ thuật TF-IDF hoặc Bag of Words để biểu diễn văn bản dưới dạng vector số học.

* Train

Chia dữ liệu thành tập train (80%) và tập test (20%).Sử dụng Logistic Regression để train mô hình trên tập train. Các trọng số của mô hình sẽ được chỉnh sửa để tối ưu hóa hàm mất mát.

* Test

Để đo lường mức độ hoạt động của mô hình, có thể sử dụng các chỉ số nhưđộ accuracy, precision, recall, và F1-score.Nếu cần thiết, điều chỉnh các siêu tham số hoặc sử dụng các kỹ thuật như regularization để cải thiện hiệu suất.

* Dự đoán cảm xúc

Sau khi mô hình được huấn luyện, nó có thể được sử dụng để dự đoán cảm xúc của các tweet mới. Ví dụ, nếu xác suất dự đoán lớn hơn 0.5, tweet có thể được phân loại là tích cực, ngược lại là tiêu cực.

Ví dụ:

Giả sử có một tweet: "Hôm nay tôi cảm thấy rất hạnh phúc!". Mô hình sẽ phân tích các từ như "hạnh phúc" và dự đoán rằng tweet này có xác suất cao thuộc về lớp tích cực.

#### **2.1.1.4.Kết luận**

Qua nghiên cứu và phân tích các mô hình học máy phổ biến cho bài toán phân tích cảm xúc trên Twitter, bao gồm Naive Bayes, Linear Support Vector Machine (LinearSVC), Logistic Regression, và Random Forest. Mỗi mô hình có những ưu điểm và nhược điểm riêng, phù hợp với các tình huống khác nhau:

* Naive Bayes: Đơn giản, nhanh chóng, phù hợp với dữ liệu văn bản đơn giản nhưng hạn chế với dữ liệu phức tạp.
* LinearSVC: Hiệu quả với dữ liệu nhiều chiều và phi tuyến, nhưng tốn tài nguyên tính toán và khó giải thích.
* Logistic Regression: Dễ hiểu, ổn định với dữ liệu có số chiều thấp, nhưng kém hiệu quả với dữ liệu phi tuyến.

## **2.2.Xây dựng và cài đặt mô hình**

### **2.2.1.Naive Bayes**

#### **2.2.1.1.Huấn luyện và đánh giá mô hình**

**a) Tách dữ liệu và tiền xử lý**

Tách văn bản và nhãn:Đầu tiên, hàm trích xuất danh sách văn bản (tweets) và nhãn (sentiment) từ danh sách các tweet đã load. Điều này cho phép tách riêng phần dữ liệu đầu vào và các nhãn tương ứng.

Tiền xử lý văn bản: Mỗi văn bản được xử lý qua hàm preprocess\_text với các thao tác:

* Chuyển về chữ thường: Giúp giảm thiểu biến thể của từ (VD: "Hello" và "hello" được xem như cùng một từ).
* Loại bỏ stopwords: Các từ không có nhiều ý nghĩa như “and”, “the”, “is” bị loại bỏ.
* Tokenization: Tách câu thành từng từ riêng lẻ.
* Lemmatization: Chuyển đổi các từ về dạng gốc (lemma) nhằm chuẩn hóa dữ liệu đầu vào.

Trích dẫn code :

**# Tách văn bản và nhãn**

**texts = [tweet['text'] for tweet in tweets]**

**sentiments = [tweet['sentiment'] for tweet in tweets]**

**# Preprocess text**

**processed\_texts = []**

**for text in tqdm(texts, desc="Preprocessing text"):**

**processed\_texts.append(preprocess\_text(text))**

Giải thích chi tiết**:**

* Duyệt qua từng đối tượng tweet trong danh sách tweets và trích xuất trường 'text', lưu vào danh sách texts. Tương tự, trích xuất nhãn (sentiment) của từng tweet.
* Vòng lặp forkết hợp với tqdm không chỉ lặp qua toàn bộ văn bản mà còn cung cấp thông tin tiến trình, rất hữu ích khi xử lý tập dữ liệu lớn.
* Hàm preprocess\_text đảm bảo mỗi tweet được chuyển hóa thành dạng chuẩn, loại -bỏ nhiễu và các từ không mang ý nghĩa cho mô hình, từ đó giúp cải thiện hiệu quả học của mô hình.

**b) Vector hóa văn bản với TF-IDF**

Biểu diễn văn bản: Sau khi tiền xử lý, văn bản được chuyển đổi thành dạng số học thông qua TF-IDF (Term Frequency - Inverse Document Frequency). Điều này cho phép mô hình Logistic Regression làm việc trên dữ liệu số thay vì dữ liệu văn bản thô.

N-gram Range: Sử dụng tham số ngram\_range=(1, 2) giúp khai thác thêm thông tin từ cả từ đơn (unigrams) lẫn cặp từ (bigrams), qua đó có thể nắm bắt được ngữ cảnh và mối liên hệ giữa các từ.

Trích dẫn code:

**# Vector hóa văn bản sử dụng TF-IDF với ngram\_range từ 1 đến 2**

**if vectorizer\_type == 'tfidf':**

**vectorizer = TfidfVectorizer(ngram\_range=(1, 2))**

**X = vectorizer.fit\_transform(processed\_texts)**

**else:**

**raise ValueError('Invalid vectorizer type. Please use "tfidf"')**

Giải thích chi tiết:

* Nếu vectorizer\_type là 'tfidf', thì tạo một đối tượng TfidfVectorizer với tham số ngram\_range=(1, 2). Điều này có nghĩa là ngoài việc lấy từ đơn (unigrams), vectorizer còn xét đến các cặp từ (bigrams) để tăng cường ngữ cảnh.
* Hàm này vừa "fit" (tìm hiểu từ vựng và tính trọng số) trên dữ liệu đã tiền xử lý, vừa "transform" (chuyển đổi từng văn bản thành vector) dữ liệu.
* Kết quả được lưu vào biến X, là một ma trận đặc trưng dưới dạng sparse matrix.
* Nếu không sử dụng TF-IDF, sẽ báo lỗi với thông báo "Invalid vectorizer type."

**c) Mã hóa nhãn và chia dữ liệu**

Mã hóa nhãn:

* Các nhãn cảm xúc ban đầu (ví dụ: -1, 0, 1) không phù hợp với hầu hết các thuật toán học máy vì chúng có thể không trực tiếp biểu diễn thứ tự hay mức độ ưu tiên theo cách mà mô hình cần. Do đó, cần chuyển đổi các nhãn này sang dạng số học liên tục từ 0 đến n-1, trong đó n là số lượng các lớp cảm xúc khác nhau.

Chia dữ liệu:

* Phân chia dữ liệu thành tập huấn luyện và tập kiểm thử bằng train\_test\_split. Việc chia dữ liệu theo tỉ lệ cố định và sử dụng tham số stratify đảm bảo rằng tỷ lệ của các lớp cảm xúc được duy trì đồng đều ở cả hai tập, giúp đánh giá mô hình một cách khách quan.

Trích dấn code:

**y = np.array(sentiments)**

**# Encode target variable**

**label\_encoder = LabelEncoder()**

**y\_encoded = label\_encoder.fit\_transform(y)**

**# Chia dữ liệu thành tập huấn luyện và kiểm thử**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_encoded, test\_size=0.2, random\_state=2, stratify=y\_encoded**)

Giải thích :

- Chuyển đổi danh sách nhãn sentiments thành mảng NumPy

- Sử dụng LabelEncoder để chuyển đổi các nhãn gốc (có thể là -1, 0, 1) thành các số nguyên từ 0 đến n-1. Điều này giúp đảm bảo rằng các nhãn được biểu diễn dưới dạng số, thuận tiện cho quá trình xử lý và huấn luyện mô hình.

- Sử dụng hàm train\_test\_split để chia dữ liệu thành tập huấn luyện và tập kiểm thử với tỷ lệ 80:20.

- Thiết lập tham số stratify đảm bảo rằng phân bố nhãn của cả hai tập dữ liệu này giống với phân bố trong tập dữ liệu gốc, qua đó duy trì tính cân bằng và độ tin cậy của quá trình đánh giá mô hình.

**d) Cân bằng dữ liệu với SMOTE**

Xử lý dữ liệu mất cân bằng: Trong nhiều bài toán thực tế, số lượng mẫu của các lớp không đều, dẫn đến mô hình bị thiên vị. SMOTE (Synthetic Minority Oversampling Technique) được sử dụng để tổng hợp các mẫu mới cho lớp thiểu số, tạo ra một tập dữ liệu huấn luyện cân bằng hơn.

Trích dẫn code :

**# Áp dụng SMOTE để tăng cường dữ liệu cho lớp thiểu số**

**smote = SMOTE(random\_state=2)**

**X\_train\_resampled, y\_train\_resampled = smote.fit\_resample(X\_train, y\_train)**

Giải thích :

* SMOTE: Kỹ thuật này tạo ra các mẫu tổng hợp bằng cách nội suy giữa các mẫu hiện có của lớp thiểu số. Điều này giúp giảm độ lệch trong tập dữ liệu, đảm bảo rằng mô hình có cơ hội học được đặc trưng của tất cả các lớp một cách đầy đủ.
* Tham số random\_state: Giúp tái lập quá trình tổng hợp mẫu, đảm bảo tính nhất quán trong các lần chạy.

**e) Huấn luyện mô hình với Logistic Regression**

Chọn mô hình: Ở đây, mô hình được lựa chọn là Logistic Regression – một thuật toán đơn giản nhưng hiệu quả trong phân loại nhị phân và đa lớp.

Cấu hình tham số:

* max\_iter=2000: Cho phép thuật toán chạy đủ số vòng lặp để hội tụ, đặc biệt khi dữ liệu lớn.
* C: Tham số điều chỉnh độ phạt của regularization, ảnh hưởng đến độ phức tạp của mô hình.
* class\_weight='balanced': Đảm bảo rằng trọng số của các lớp được điều chỉnh phù hợp, giảm thiểu sai lệch khi dữ liệu không cân bằng (ngay cả khi đã dùng SMOTE).

Trích dẫn code:

**# Huấn luyện mô hình với Logistic Regression**

**if model\_type == 'logistic\_regression':**

**model = LogisticRegression(max\_iter=2000, C=C, class\_weight='balanced')**

**else:**

**raise ValueError('Loại mô hình không hợp lệ.')**

**model.fit(X\_train\_resampled, y\_train\_resampled)**

**# Dự đoán trên tập kiểm thử**

**y\_pred = model.predict(X\_test)**

Giải thích:

* Logistic Regression: Là mô hình tuyến tính đơn giản, hiệu quả cho các bài toán phân loại. Các tham số cấu hình đảm bảo rằng mô hình không chỉ học được mà còn tránh overfitting.
* Huấn luyện và Dự đoán: Quá trình huấn luyện dựa trên dữ liệu đã cân bằng từ SMOTE, sau đó thực hiện dự đoán trên tập dữ liệu kiểm thử để đánh giá hiệu năng của mô hình.

**f) Lưu mô hình và các đối tượng liên quan**

Lưu trữ pipeline: Sau khi hoàn tất huấn luyện, mô hình cùng với các đối tượng tiền xử lý như vectorizer và label encoder được lưu lại thành một tuple. Điều này giúp cho việc triển khai và tái sử dụng mô hình không cần phải huấn luyện lại từ đầu.

Sử dụng joblib: joblib.dump được sử dụng để lưu đối tượng mô hình vào file định dạng .sav, giúp nạp nhanh hơn so với một số phương pháp lưu trữ khác.

Trích dẫn code:

**# Tạo thư mục lưu trữ nếu chưa tồn tại và lưu mô hình cùng với vectorizer và label encoder**

**os.makedirs(os.path.dirname(model\_path), exist\_ok=True)**

**dump((model, vectorizer, label\_encoder), model\_path)**

**print(f"Model and Vectorizer saved to {model\_path}")**

Giải thích chi tiết:

* os.makedirs: Đảm bảo rằng thư mục chứa file mô hình tồn tại, tránh lỗi ghi file khi thư mục chưa được tạo.
* Joblib: Công cụ này rất hiệu quả trong việc lưu trữ các mô hình học máy có kích thước lớn và các đối tượng phức tạp.

**g) Đánh giá mô hình**

Giải mã kết quả dự đoán: Các kết quả dự đoán từ mô hình được chuyển đổi trở lại dạng nhãn ban đầu qua label\_encoder.inverse\_transform, giúp so sánh với dữ liệu gốc.

Đánh giá hiệu năng:

* accuracy\_score: Tính tỷ lệ dự đoán đúng trên tập kiểm thử.
* classification\_report: Cung cấp thông tin chi tiết như precision, recall, F1-score cho từng lớp cảm xúc, giúp đánh giá điểm mạnh và hạn chế của mô hình.

Trích dẫn code:

**# Giải mã kết quả dự đoán và nhãn thật từ tập kiểm thử**

**y\_pred\_decoded = label\_encoder.inverse\_transform(y\_pred)**

**y\_test\_decoded = label\_encoder.inverse\_transform(y\_test)**

**# Tính độ chính xác và tạo báo cáo phân loại**

**accuracy = accuracy\_score(y\_test\_decoded, y\_pred\_decoded)**

**report = classification\_report(y\_test\_decoded, y\_pred\_decoded, labels=[-1, 0, 1], zero\_division=0)**

Giải thích chi tiết:

* Accuracy và Classification Report: Cho phép đánh giá tổng quát cũng như chi tiết hiệu năng của mô hình theo từng lớp, cung cấp cái nhìn sâu sắc về khả năng phân loại của mô hình đối với từng loại cảm xúc.

#### **2.2.1.2.Tải mô hình đã huấn luyện**

Sử dụng joblib.load để đọc file đã lưu. File chứa tuple gồm mô hình đã huấn luyện, vectorizer và label encoder

Trích dấn code:

**def load\_model(model\_path="../models/TFIDF\_LogisticRegression\_model.sav"): # Thay đổi đường dẫn model**

**"""Load the trained model from a file."""**

**try:**

**model = load(model\_path)**

**print(f"Model loaded from {model\_path}")**

**return model**

**except Exception as e:**

**print(f"Error loading model: {e}")**

**return None**

Giải thích chi tiết:

* Joblib.load: Công cụ này được sử dụng để nạp lại các đối tượng được lưu từ quá trình huấn luyện, rất nhanh và hiệu quả cho các mô hình học máy.
* Xử lý ngoại lệ: Việc bắt và xử lý lỗi giúp chương trình duy trì tính ổn định, tránh gây ra crash khi gặp sự cố trong quá trình tải mô hình.

### **2.2.2. Linear Support Vector Classification (LinearSVC)**

#### **2.2.2.1.Huấn luyện và đánh giá mô hình**

**a) Tách dữ liệu và tiền xử lý văn bản**

Tách dữ liệu: Lấy ra văn bản (tweet) và nhãn (sentiment) từ file CSV.

Tiền xử lý: Chuẩn hóa văn bản để loại bỏ nhiễu, giảm số lượng biến thể không cần thiết, giúp tăng hiệu quả của mô hình học máy.

Trích dẫn code:

**# Tách văn bản và nhãn từ dữ liệu tweet**

**texts = [tweet['text'] for tweet in tweets]**

**sentiments = [tweet['sentiment'] for tweet in tweets]**

**# Tiền xử lý văn bản: chuẩn hóa bằng cách chuyển về chữ thường, loại stopwords, tokenization và lemmatization**

**processed\_texts = []**

**for text in tqdm(texts, desc="Preprocessing text"):**

**processed\_texts.append(preprocess\_text(text))**

Giải thích chi tiết:

Tách dữ liệu:

* Dòng đầu tiên tạo danh sách texts chứa trường 'text' từ từng tweet.
* Dòng thứ hai tạo danh sách sentiments chứa nhãn cảm xúc của từng tweet.

Tiền xử lý:

* Sử dụng vòng lặp kết hợp với tqdm để theo dõi tiến trình xử lý.
* Hàm preprocess\_text(text) chuyển văn bản về chữ thường, loại bỏ các stopwords, tách từ (tokenization) và thực hiện lemmatization để đưa từ về dạng gốc, giúp giảm nhiễu và chuẩn hóa dữ liệu đầu vào.

**b) Vector hóa văn bản với TF-IDF**

Biểu diễn văn bản: Chuyển đổi các văn bản đã tiền xử lý thành vector số học để mô hình có thể xử lý.

TF-IDF: Tính trọng số của từng từ dựa trên tần suất trong từng văn bản và toàn bộ tập dữ liệu, đồng thời sử dụng cả unigrams và bigrams (ngram\_range=(1,2)).

Trích dẫn code:

**if vectorizer\_type == 'tfidf':**

**vectorizer = TfidfVectorizer(ngram\_range=(1, 2)) # Sử dụng cả unigrams và bigrams**

**X = vectorizer.fit\_transform(processed\_texts)**

**else:**

**raise ValueError('Invalid vectorizer type. Please use "tfidf"')**

Giải thích chi tiết:

* Kiểm tra loại vectorizer: Nếu vectorizer\_type là 'tfidf', tạo đối tượng TfidfVectorizer với tham số ngram\_range=(1, 2) giúp thu thập thông tin từ cả đơn từ và cặp từ.
* Fit và Transform: Hàm fit\_transform(processed\_texts) vừa học từ vựng từ tập dữ liệu vừa chuyển đổi văn bản thành một ma trận vector dưới dạng sparse matrix.
* Thông báo lỗi: Nếu không dùng TF-IDF, chương trình sẽ báo lỗi với thông báo thích hợp.

**c) Mã hóa nhãn và chia dữ liệu**

Mã hóa nhãn: Chuyển đổi các nhãn cảm xúc (ban đầu có thể là -1, 0, 1) thành dạng số để mô hình xử lý.

Chia dữ liệu: Phân chia dữ liệu thành tập huấn luyện và kiểm thử, đảm bảo phân bố các lớp được duy trì (sử dụng stratified sampling).

Trích dẫn code:

**y = np.array(sentiments)**

**# Encode target variable: chuyển nhãn từ dạng gốc sang dạng số học**

**label\_encoder = LabelEncoder()**

**y\_encoded = label\_encoder.fit\_transform(y)**

**# Chia dữ liệu thành tập huấn luyện và kiểm thử (80% huấn luyện, 20% kiểm thử)**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, y\_encoded, test\_size=0.2, random\_state=2, stratify=y\_encoded )**

Giải thích chi tiết:

* Mã hóa nhãn: Chuyển danh sách sentiments sang mảng NumPy và dùng LabelEncoder để biến đổi các nhãn thành số (ví dụ: từ -1, 0, 1 thành các giá trị 0, 1, 2).
* Chia dữ liệu: Hàm train\_test\_split chia dữ liệu thành tập huấn luyện và tập kiểm thử với tỉ lệ 80/20, đồng thời sử dụng tham số stratify để đảm bảo tỷ lệ các lớp được giữ nguyên trong cả hai tập.

**d) Cân bằng dữ liệu với SMOTE**

Xử lý dữ liệu mất cân bằng: Áp dụng SMOTE để tạo thêm các mẫu tổng hợp cho lớp thiểu số, từ đó cân bằng số lượng mẫu trong tập huấn luyện.

Trích dẫn code:

**print(f"Số lượng mẫu trong tập huấn luyện trước SMOTE: {X\_train.shape[0]}")**

**smote = SMOTE(random\_state=2)**

**X\_train\_resampled, y\_train\_resampled = smote.fit\_resample(X\_train, y\_train)**

**print(f"Số lượng mẫu trong tập huấn luyện sau SMOTE: {X\_train\_resampled.shape[0]}")**

Giải thích chi tiết:

* In số mẫu ban đầu: Hiển thị số lượng mẫu trong tập huấn luyện trước khi áp dụng SMOTE.
* Áp dụng SMOTE: Tạo đối tượng SMOTE với random\_state=2 để đảm bảo tính nhất quán.Hàm fit\_resample tạo ra các mẫu tổng hợp cho lớp thiểu số, trả về tập dữ liệu huấn luyện đã được cân bằng.
* In số mẫu sau SMOTE: Giúp so sánh số lượng mẫu trước và sau khi cân bằng.

**e) Huấn luyện mô hình với LinearSVC**

Huấn luyện mô hình SVM: Sử dụng LinearSVC, một biến thể của SVM tối ưu cho dữ liệu lớn, để phân loại văn bản.

Cấu hình tham số:

* + C điều chỉnh độ phạt (regularization).
  + class\_weight='balanced' để xử lý dữ liệu không cân bằng.
  + dual=False là một tối ưu cho các bài toán với số mẫu lớn hơn số đặc trưng.

Trích dẫn code:

**try:**

**if model\_type == 'svm':**

**model = LinearSVC(C=C, class\_weight='balanced', dual=False) # Sử dụng LinearSVC**

**else:**

**raise ValueError('Loại mô hình không hợp lệ.')**

**model.fit(X\_train\_resampled, y\_train\_resampled)**

**y\_pred = model.predict(X\_test)**

**except ValueError as e:**

**print(f"ValueError trong quá trình huấn luyện model {model\_type}: {e}")**

**return None, None**

Giải thích chi tiết:

Chọn mô hình: Nếu model\_type bằng 'svm', khởi tạo một đối tượng LinearSVC với = các tham số:

* C=C: Giá trị điều chỉnh độ phạt.
* class\_weight='balanced': Tự động cân bằng trọng số các lớp.
* dual=False: Được sử dụng khi số mẫu lớn hơn số đặc trưng, giúp tối ưu quá trình huấn luyện.

Huấn luyện và dự đoán:

* Gọi model.fit để huấn luyện trên tập dữ liệu đã cân bằng.
* Sau đó, sử dụng model.predict trên tập kiểm thử để thu được dự đoán.

Xử lý lỗi: Nếu có lỗi xảy ra (ví dụ, sai tham số), khối except sẽ in thông báo lỗi và trả về None cho độ chính xác và báo cáo phân loại.

**f) Lưu mô hình và các đối tợng liên quan**

Lưu pipeline: Lưu mô hình, vectorizer và label encoder dưới dạng một tuple để có thể tái sử dụng mà không cần huấn luyện lại từ đầu.

Trích dẫn code:

**os.makedirs(os.path.dirname(model\_path), exist\_ok=True)**

**dump((model, vectorizer, label\_encoder), model\_path)**

**print(f"Model and Vectorizer saved to {model\_path}")**

Giải thích chi tiết:

* Tạo thư mục: Sử dụng os.makedirs để đảm bảo rằng thư mục chứa file mô hình tồn tại, nếu chưa có, nó sẽ được tạo.
* Lưu mô hình: Hàm dump từ thư viện joblib lưu một tuple chứa (model, vectorizer, label\_encoder) vào file với đường dẫn model\_path.
* Thông báo lưu mô hình: In ra thông báo xác nhận mô hình đã được lưu thành công.

**g) Đánh giá mô hình**

Giải mã dự đoán: Chuyển đổi các nhãn dự đoán từ dạng mã hóa trở lại nhãn ban đầu để so sánh với dữ liệu gốc.

Tính toán chỉ số: Sử dụng các thước đo như accuracy và báo cáo phân loại để đánh giá hiệu năng của mô hình.

Trích dẫn code:

**y\_pred\_decoded = label\_encoder.inverse\_transform(y\_pred)**

**y\_test\_decoded = label\_encoder.inverse\_transform(y\_test)**

**accuracy = accuracy\_score(y\_test\_decoded, y\_pred\_decoded)**

**labels = [-1, 0, 1]**

**report = classification\_report(y\_test\_decoded, y\_pred\_decoded, labels=labels, zero\_division=0)**

Giải thích chi tiết:

Giải mã nhãn: Sử dụng inverse\_transform của label\_encoder để chuyển các nhãn dự đoán và nhãn gốc về dạng ban đầu (ví dụ: -1, 0, 1).

Đánh giá:

* Hàm accuracy\_score tính tỷ lệ dự đoán đúng.
* Hàm classification\_report tạo báo cáo chi tiết (precision, recall, F1-score) cho từng lớp, với tham số zero\_division=0 để tránh lỗi khi có lớp không có mẫu dự đoán.

#### **2.2.2.2. Tải mô hình đã huấn luyện**

Nạp lại mô hình: Đảm bảo có thể tải lại mô hình đã lưu kèm theo vectorizer và label encoder cho các tác vụ dự đoán mới hoặc kiểm thử mà không cần huấn luyện lại.

Trích dẫn code:

**def load\_model(model\_path="../models/TFIDF\_SVM\_model.sav"):**

**try:**

**model = load(model\_path)**

**print(f"Model loaded from {model\_path}")**

**return model**

**except Exception as e:**

**print(f"Error loading model: {e}")**

**return None**

Giải thích chi tiết:

* Nạp mô hình: Dùng load từ joblib để đọc file chứa tuple (model, vectorizer, label\_encoder) từ model\_path.
* Xử lý ngoại lệ: Nếu có lỗi (file không tồn tại, định dạng lỗi, ...), khối except sẽ in thông báo lỗi và trả về None để báo hiệu việc tải mô hình không thành công.

### **2.2.3. Logistic Regression**

#### **2.2.3.1.Huấn luyện và đánh giá mô hình**

**a) Tách dữ liệu và tiền xử lý văn bản**

Tách dữ liệu: Lấy ra trường văn bản (text) và nhãn cảm xúc (sentiment) từ danh sách các tweet.

Tiền xử lý: Chuẩn hóa văn bản để giảm nhiễu, qua đó giúp mô hình học máy có dữ liệu đầu vào sạch và nhất quán.

Trích dẫn code:

**# Tách văn bản và nhãn**

**texts = [tweet['text'] for tweet in tweets]**

**sentiments = [tweet['sentiment'] for tweet in tweets]**

**# Preprocess text: chuẩn hóa văn bản bằng cách chuyển thành chữ thường, loại bỏ stopwords, tokenization và lemmatization**

**processed\_texts = []**

**for text in tqdm(texts, desc="Preprocessing text"):**

**processed\_texts.append(preprocess\_text(text))**

Giải thích:

Tách dữ liệu:

* Dòng đầu tiên duyệt qua mỗi tweet để lấy trường 'text' và lưu vào danh sách texts.
* Tương tự, danh sách sentiments chứa các nhãn tương ứng.

Tiền xử lý:

* Sử dụng vòng lặp kết hợp với tqdm nhằm hiển thị tiến trình xử lý, rất hữu ích khi làm việc với dữ liệu lớn.
* Hàm preprocess\_text(text) sẽ:

+ Chuyển toàn bộ văn bản về chữ thường.

+ Loại bỏ stopwords không cần thiết.

+ Thực hiện tokenization (tách từ) và lemmatization để đưa các từ về dạng gốc.

Những bước này giúp giảm sự đa dạng không cần thiết của từ ngữ, làm cho dữ liệu đầu vào cho mô hình trở nên nhất quán và hiệu quả hơn.

**b) Vector hóa băn bản với TF-IDF**

Biểu diễn văn bản: Chuyển đổi văn bản đã tiền xử lý thành dạng vector số học bằng TF-IDF.

Tăng cường thông tin: Sử dụng n-grams (cụ thể là unigrams và bigrams) để khai thác thêm thông tin ngữ cảnh.

Trích dẫn code:

**if vectorizer\_type == 'tfidf':**

**vectorizer = TfidfVectorizer(ngram\_range=(1, 2)) # Sử dụng cả unigrams và bigrams**

**X = vectorizer.fit\_transform(processed\_texts)**

**else:**

**raise ValueError('Invalid vectorizer type. Please use "tfidf"')**

Giải thích:

* Kiểm tra loại vectorizer: Nếu biến vectorizer\_type bằng 'tfidf', tiến hành tạo đối tượng TfidfVectorizer với tham số ngram\_range=(1, 2).
* Fit và transform: Hàm fit\_transform(processed\_texts) vừa học từ vựng từ tập dữ liệu đã tiền xử lý vừa chuyển đổi các văn bản thành ma trận vector số học (sparse matrix).
* Xử lý lỗi: Nếu không sử dụng TF-IDF, chương trình sẽ báo lỗi với thông báo rõ ràng.

**c. Mã hóa nhãn và chia dữ liệu**

Mã hóa nhãn: Chuyển đổi nhãn gốc (ví dụ: -1, 0, 1) sang dạng số liên tục để mô hình có thể xử lý.

Chia dữ liệu: Phân chia dữ liệu thành tập huấn luyện và kiểm thử với tỷ lệ cố định, sử dụng stratified sampling để duy trì phân bố các lớp.

Trích dẫn code:

**y = np.array(sentiments)**

**# Encode the target variable to numeric values**

**label\_encoder = LabelEncoder()**

**y\_encoded = label\_encoder.fit\_transform(y)**

**# Chia dữ liệu thành tập huấn luyện và tập kiểm thử**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_encoded, test\_size=0.2, random\_state=2, stratify=y\_encoded)**

Giải thích:

Mã hóa nhãn:

* Chuyển danh sách sentiments thành mảng NumPy.
* Sử dụng LabelEncoder để biến đổi các nhãn thành các giá trị số, giúp việc tính toán trở nên đơn giản hơn.

Chia dữ liệu:

* Hàm train\_test\_split chia dữ liệu thành 80% cho tập huấn luyện và 20% cho tập kiểm thử.
* Tham số stratify=y\_encoded đảm bảo rằng tỉ lệ giữa các lớp nhãn trong cả hai tập là giống nhau, giảm thiểu sai lệch trong đánh giá mô hình.

**d) Cân bằng dữ liệu với SMOTE**

Xử lý dữ liệu không cân bằng: Áp dụng SMOTE để tạo ra các mẫu tổng hợp cho lớp thiểu số, qua đó cân bằng số lượng mẫu giữa các lớp trong tập huấn luyện.

Trích dẫn code:

**print(f"Số lượng mẫu trong tập huấn luyện trước SMOTE: {X\_train.shape[0]}")**

**# Áp dụng SMOTE để tăng cường dữ liệu cho lớp thiểu số**

**smote = SMOTE(random\_state=2)**

**X\_train\_resampled, y\_train\_resampled = smote.fit\_resample(X\_train, y\_train)**

**print(f"Số lượng mẫu trong tập huấn luyện sau SMOTE: {X\_train\_resampled.shape[0]}")**

Giải thích:

In số mẫu ban đầu: In ra số lượng mẫu trong tập huấn luyện trước khi áp dụng SMOTE để có thể so sánh.

SMOTE:

* Khởi tạo đối tượng SMOTE với random\_state=2 để đảm bảo tính nhất quán trong các lần chạy.
* Hàm fit\_resample tạo ra các mẫu tổng hợp cho lớp thiểu số, kết quả trả về là tập dữ liệu đã được cân bằng.

In số mẫu sau SMOTE: Cho thấy số lượng mẫu sau khi áp dụng SMOTE, giúp minh họa hiệu quả của kỹ thuật này.

**d) Huấn luyện mô hình với Logistic Regression**

Huấn luyện mô hình: Sử dụng Logistic Regression với các tham số được cấu hình phù hợp (như C, max\_iter và class\_weight) trên tập dữ liệu đã được cân bằng.

Dự đoán: Sau khi huấn luyện, thực hiện dự đoán trên tập kiểm thử.

Trích dẫn code:

**try:**

**if model\_type == 'logistic\_regression':**

**model = LogisticRegression(max\_iter=2000, C=C, class\_weight='balanced') # Thêm class\_weight='balanced' và C**

**else:**

**raise ValueError('Loại mô hình không hợp lệ.')**

**model.fit(X\_train\_resampled, y\_train\_resampled)**

**y\_pred = model.predict(X\_test)**

**except ValueError as e:**

**print(f"ValueError trong quá trình huấn luyện model {model\_type}: {e}")**

**return None, None**

Giải thích:

Chọn mô hình: Kiểm tra biến model\_type. Nếu bằng 'logistic\_regression', tạo đối tượng LogisticRegression với:

* max\_iter=2000 để đảm bảo hội tụ trong trường hợp tập dữ liệu lớn.
* C=C điều chỉnh độ phạt của regularization, giúp kiểm soát độ phức tạp của mô hình.
* class\_weight='balanced' để tự động điều chỉnh trọng số cho các lớp, giúp xử lý dữ liệu không cân bằng.

Huấn luyện và dự đoán:

* Gọi model.fit để huấn luyện mô hình trên tập dữ liệu đã được cân bằng.
* Sau đó, sử dụng model.predict trên tập kiểm thử để thu được dự đoán.

Xử lý lỗi: Nếu có lỗi trong quá trình khởi tạo hoặc huấn luyện mô hình, khối except sẽ in thông báo lỗi và trả về None.

**e) Lưu mô hình và các đối tượng liên quan**

Lưu lại pipeline: Lưu mô hình đã huấn luyện cùng với vectorizer và label encoder dưới dạng một tuple để có thể tái sử dụng mà không cần huấn luyện lại từ đầu.

Trích dẫn code:

**# Tạo thư mục nếu chưa tồn tại và lưu mô hình, vectorizer, label\_encoder**

**os.makedirs(os.path.dirname(model\_path), exist\_ok=True)**

**dump((model, vectorizer, label\_encoder), model\_path)**

**print(f"Model and Vectorizer saved to {model\_path}")**

Giải thích:

* Tạo thư mục: Dùng os.makedirs để tạo thư mục chứa file lưu mô hình nếu thư mục đó chưa tồn tại (với exist\_ok=True để tránh lỗi nếu đã có).
* Lưu mô hình: Hàm dump từ thư viện joblib lưu một tuple gồm (model, vectorizer, label\_encoder) vào đường dẫn model\_path.
* Thông báo lưu: In ra thông báo xác nhận rằng mô hình và các đối tượng liên quan đã được lưu thành công.

**f) Giải mã và đánh giá mô hình**

Giải mã nhãn dự đoán: Chuyển các nhãn dự đoán từ dạng mã hóa trở lại dạng ban đầu để so sánh với dữ liệu gốc.

Đánh giá: Tính toán độ chính xác (accuracy) và tạo báo cáo phân loại (classification report).

Trích dẫn code:

**# Decode the predicted labels**

**y\_pred\_decoded = label\_encoder.inverse\_transform(y\_pred)**

**y\_test\_decoded = label\_encoder.inverse\_transform(y\_test)**

**# Đánh giá**

**accuracy = accuracy\_score(y\_test\_decoded, y\_pred\_decoded)**

**labels = [-1, 0, 1]**

**report = classification\_report(y\_test\_decoded, y\_pred\_decoded, labels=labels, zero\_division=0)**

Giải thích:

Giải mã nhãn:Dùng inverse\_transform của label\_encoder để chuyển các nhãn dự đoán và nhãn thật về dạng ban đầu (ví dụ: -1, 0, 1) nhằm thuận tiện cho việc so sánh.

Đánh giá:

* accuracy\_score tính tỷ lệ dự đoán đúng trên tập kiểm thử.
* classification\_report tạo báo cáo chi tiết bao gồm các chỉ số như precision, recall và F1-score cho từng lớp. Tham số zero\_division=0 dùng để tránh lỗi chia cho 0 nếu có lớp không có mẫu dự đoán.

#### **2.2.3.2.Tải mô hình đã huấn luyện**

Nạp lại mô hình: Tải mô hình đã lưu cùng với vectorizer và label encoder để sử dụng trong dự đoán mới hoặc trong quá trình kiểm thử mà không cần huấn luyện lại.

Trích dẫn code:

**def load\_model(model\_path="../models/TFIDF\_LogisticRegression\_model.sav"):**

**"""Load the trained model from a file."""**

**try:**

**model = load(model\_path)**

**print(f"Model loaded from {model\_path}")**

**return model**

**except Exception as e:**

**print(f"Error loading model: {e}")**

**return None**

Giải thích:

* Nạp mô hình: Sử dụng load từ joblib để nạp file chứa tuple (model, vectorizer, label\_encoder) từ đường dẫn model\_path.
* Xử lý ngoại lệ: Nếu có lỗi xảy ra (ví dụ: file không tồn tại hoặc định dạng không đúng), khối except sẽ bắt lỗi, in thông báo lỗi và trả về None.

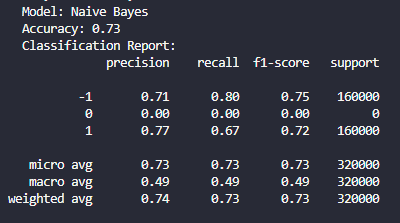
## **2.3.Đánh giá mô hình trên tập test**

### **2.3.1.Lựa chọn độ đo**

* Accuracy: Tỷ lệ dự đoán đúng tổng thể.
* Classification Report: Bảng này cung cấp độ đo chi tiết hơn cho từng lớp:
* Precision: Đối với mỗi lớp, nó đo lường tỷ lệ các dự đoán tích cực thực sự là chính xác.
* Recall: Đối với mỗi lớp, nó đo lường tỷ lệ các trường hợp thực tế tích cực được xác định
* F1-score: Trung bình điều hòa của precision và recall, cung cấp một số liệu duy nhất cân bằng giữa hai yếu tố này.
* Support: Số lượng mẫu thực tế cho mỗi lớp.
* Averages (micro, macro, weighted): Các giá trị này tổng hợp hiệu suất trên tất cả các lớp:
* Micro avg: Tính toán độ đo chung bằng cách đếm tổng số các dương tính thực sự, âm tính thực sự và dương tính giả.
* Macro avg: Tính trung bình độ đo cho mỗi lớp một cách riêng biệt, sau đó lấy trung bình của các trung bình đó. Nó không tính đến sự mất cân bằng của lớp.
* Weighted avg: Tính trung bình độ đo cho mỗi lớp, được nhân với số lượng mẫu của mỗi lớp. Điều này tính đến sự mất cân bằng của lớp.

### **2.3.1.Kết quả đánh giá các mô hình**

#### **2.3.1.1.Naive Bayes**



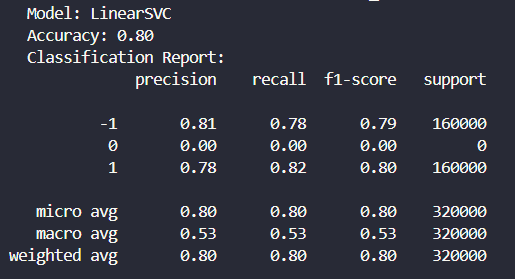
Hình 6 : Hình ảnh đánh giá Naive Bayes

Giải thích:

* Mô hình Naive Bayes đạt độ chính xác 0.73 với hai nhãn chính -1 và 1.
* Nhãn -1: precision = 0.71, recall = 0.80, f1 = 0.75
* Nhãn 1: precision = 0.77, recall = 0.67, f1 = 0.72
* Nhãn 0 không có mẫu (support = 0), nên các chỉ số đều = 0.
* Macro avg bị kéo xuống do nhãn 0 không có dữ liệu, còn micro và weighted avg đều ~0.73.

Mô hình Naive Bayes đạt độ chính xác 73% khi phân loại hai lớp -1 và 1.Lớp -1 có F1-score 0.75 (precision 0.71, recall 0.80), trong khi lớp 1 đạt F1-score 0.72 (precision 0.77, recall 0.67).Nhìn chung, mô hình hoạt động khá ổn định, nhưng có thể cần cải thiện recall cho lớp 1.

#### **2.3.1.2.LinearSVC**



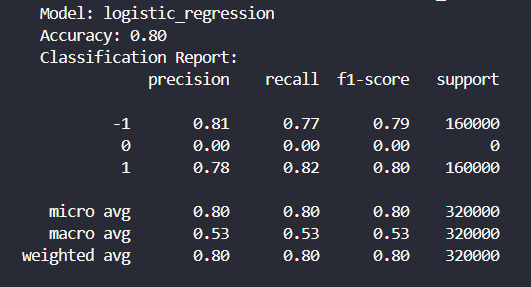
Hình 7 : Hình ảnh đánh giá LinearSVC

Giải thích:

* Mô hình LinearSVC đạt độ chính xác 0.80 với hai nhãn chính -1 và 1.
* Nhãn -1: precision = 0.81, recall = 0.78, f1 = 0.79
* Nhãn 1: precision = 0.78, recall = 0.82, f1 = 0.80
* Nhãn 0 không có mẫu (support = 0), nên các chỉ số đều = 0.
* Macro avg bị kéo xuống do nhãn 0 không có dữ liệu, còn micro và weighted avg đều ~0.80.

Mô hình LinearSVC đạt độ chính xác 80% khi phân loại hai lớp -1 và 1. Lớp -1 có F1-score 0.79 (precision 0.81, recall 0.78), trong khi lớp 1 đạt F1-score 0.80 (precision 0.78, recall 0.82). Nhìn chung, mô hình hoạt động tốt với độ chính xác cao và F1-score khá tốt cho cả hai lớp.

#### **2.3.1.3.Logistic Regression**



Hình 8 : Hình ảnh đánh giá Logistic Regression

Giải thích:

* Mô hình Logistic Regression đạt độ chính xác 0.80 với hai nhãn chính -1 và 1.
* Nhãn -1: precision = 0.81, recall = 0.77, f1 = 0.79
* Nhãn 1: precision = 0.78, recall = 0.82, f1 = 0.80
* Nhãn 0 không có mẫu (support = 0), nên các chỉ số đều = 0.
* Macro avg bị kéo xuống do nhãn 0 không có dữ liệu, còn micro và weighted avg đều ~0.80.

Mô hình Logistic Regression đạt độ chính xác 80% khi phân loại hai lớp -1 và 1. Lớp -1 có F1-score 0.79 (precision 0.81, recall 0.77), trong khi lớp 1 đạt F1-score 0.80 (precision 0.78, recall 0.82). Nhìn chung, mô hình hoạt động tốt với độ chính xác cao và F1-score khá tốt cho cả hai lớp.

# **CHƯƠNG 3.THỬ NGHIỆM VÀ ĐÁNH GIÁ**

## **3.1.Điều chỉnh mô hình cải thiện hiệu năng của mô hình**

### **3.1.1.Naive Bayes**

Trong hàm train\_and\_evaluate, tham số C được truyền vào mô hình Naive Bayes. Tùy chỉnh các tham số nhiều để train thì lựa chọn tham số C = 1.0 đem lại hiệu năng cao nhất cho mô hình.

**def train\_and\_evaluate(tweets, model\_type='naive\_bayes', model\_path="../models/TFIDF\_NaiveBayes1\_model.sav", c=1.0):**

Sử dụng thêm bigram (n-gram) trong vector hóa văn bẳn để trích xuất đặng trưng từ văn bản giúp cho mô hình hiểu ngữ cảnh các từ, cải thiện khả năng phân loại cho mô hình.

**vectorizer = TfidfVectorizer(ngram\_range=(2,3))**

****

Hình 9. Đánh giá mô hình Naive Bayes sau khi điều chỉnh

### **3.1.2.Logistic Regression**

Trong hàm train\_and\_evaluate, tham số C được truyền vào mô hình LogisticRegression:

**model = LogisticRegression(max\_iter=2000, C=C, class\_weight='balanced')**

* C là tham số điều chỉnh độ phạt của regularization trong hồi quy logistic.
* Nó kiểm soát mức độ mà mô hình cố gắng tối ưu hóa độ chính xác trên tập huấn luyện so với việc giảm overfitting.
* C là nghịch đảo của tham số regularization strength (λ), nghĩa là:

+ Giá trị C lớn → Regularization yếu hơn → Mô hình có thể bị overfitting.

+ Giá trị C nhỏ → Regularization mạnh hơn → Mô hình có thể bị underfitting.

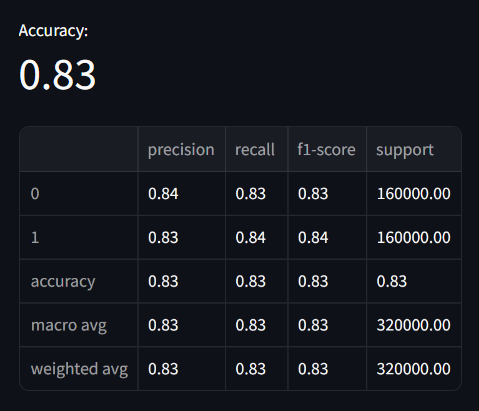
ngram\_range=(1, 2) có nghĩa là sử dụng cả unigrams (1 từ) và bigrams (2 từ liên tiếp) để trích xuất đặc trưng từ văn bản.

**vectorizer = TfidfVectorizer(ngram\_range=(1, 2))**

Ví dụ, nếu văn bản là "this is great", các n-grams sẽ là:

* Unigrams: ["this", "is", "great"]
* Bigrams: ["this is", "is great"]

Việc sử dụng bigrams giúp mô hình hiểu được ngữ cảnh giữa các từ, cải thiện khả năng phân loại so với chỉ sử dụng unigrams.



Hình 10. Đánh giá mô hình LogisticRegression sau khi điều chỉnh

### **3.1.3.Linear Support Vector Classification (LinearSVC)**

C là một tham số quan trọng trong Support Vector Machine (SVM), quyết định lỗi của mô hình.

Trong code, tham số C được truyền vào LinearSVC:

**model = LinearSVC(C=C, class\_weight='balanced', dual=False)**

Trong code, Giá trị C được sử dụng C=1.0. Giá trị này cho ra kết quả đánh giá mô hình ổn định hơn so với ban đầu.

Trong bước tiền xử lý, được sử dụng TfidfVectorizer với tham số ngram\_range=(1,2), tức là mô hình sử dụng cả:

* Unigram (từ đơn): Giúp nắm bắt thông tin cơ bản từ từng từ.
* Bigram (cụm hai từ liên tiếp): Giúp mô hình hiểu ngữ cảnh của các từ khi đi kèm nhau,

ví dụ: "not good" được xem là một cụm từ thay vì chỉ tách riêng "not" và "good".

Việc sử dụng bigram có thể giúp cải thiện hiệu suất mô hình vì nó giữ được thông tin quan trọng từ các cụm từ.



Hình 11. Đánh giá mô hình LinearSVC sau khi điều chỉnh

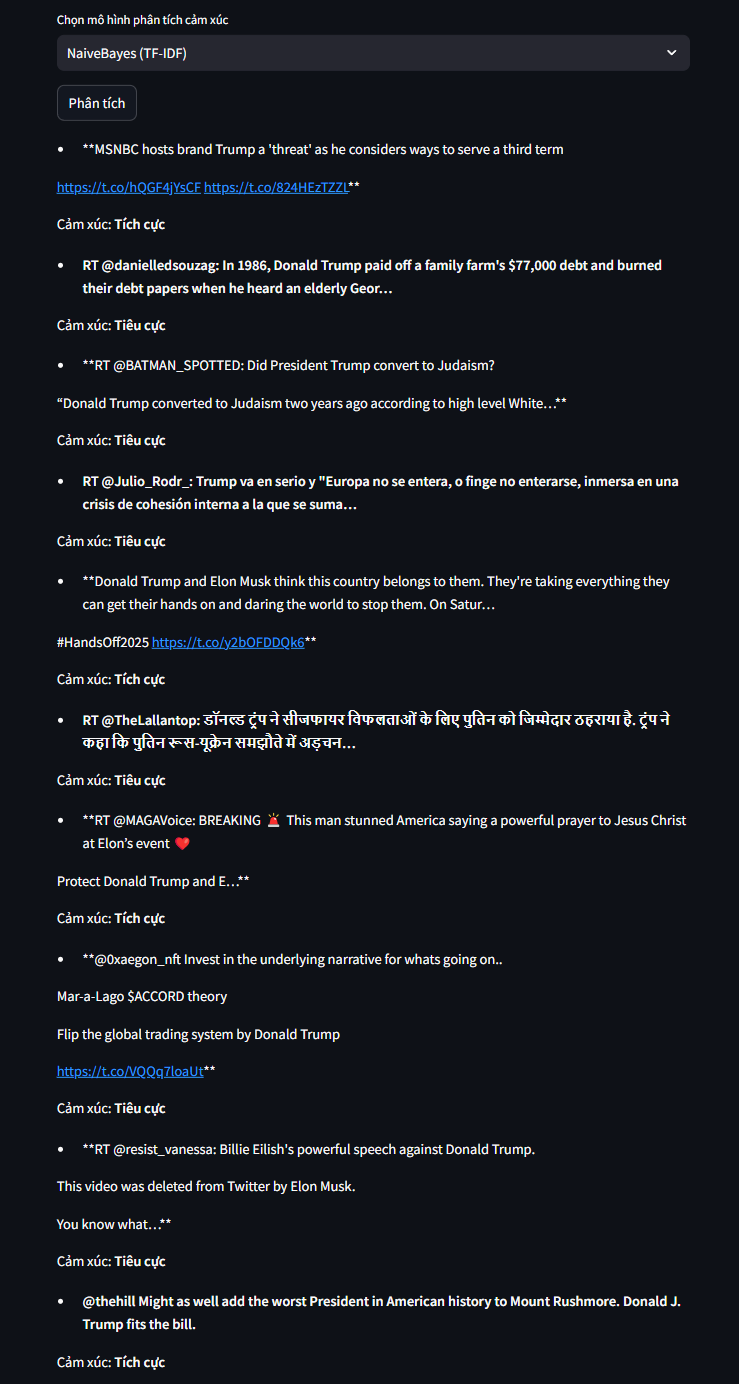
## **3.2.Thực hiện dự đoán trên dữ liệu mới**

### **3.2.1.Naive Bayes**

Thực hiện test dữ liệu mới với mô hình train và kết quả trả về như ảnh. Như có thế thấy dựa vào bài viết mô hình đã đánh giá được cảm xúc trong văn bản.



Hình 12.Mô hình Naive Bayes thực hiện dự đoán trên dữ liệu mới



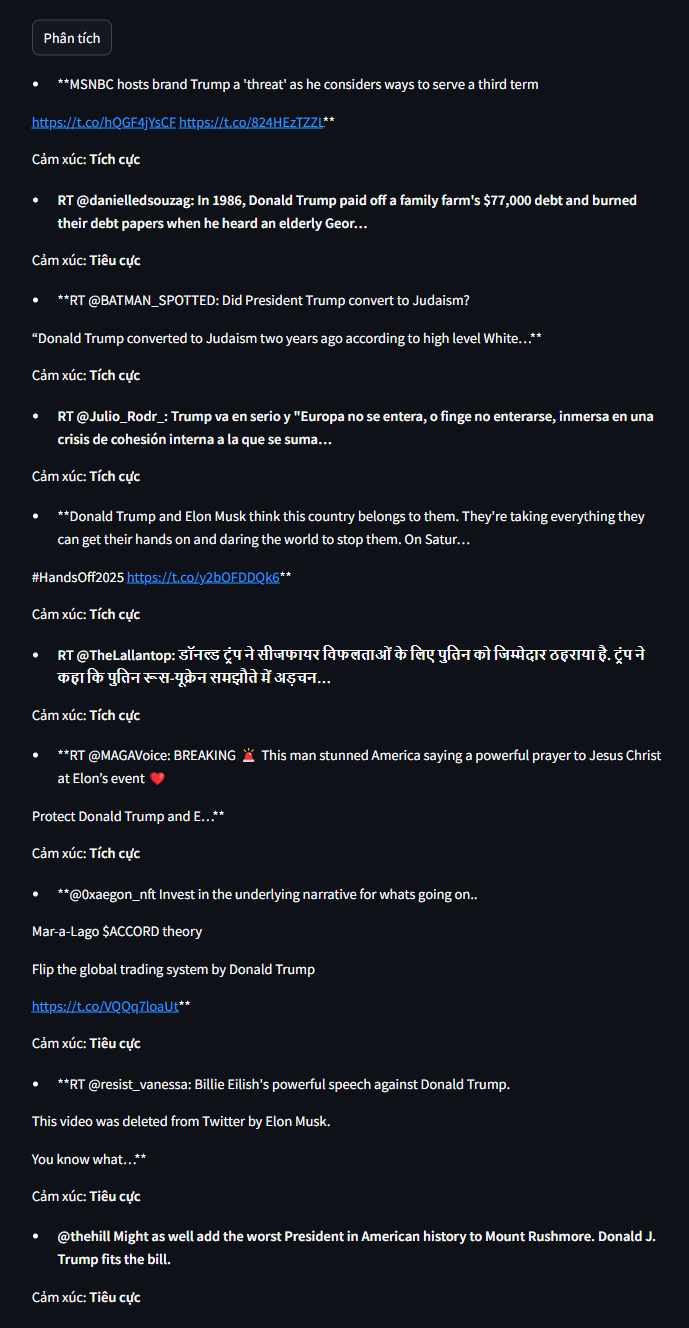
Hình 13.Mô hình Naive Bayes thực hiện dự đoán trên các tweet mới qua API X

### **3.2.2.Linear Support Vector Classification (LinearSVC)**

Thực hiện test dữ liệu mới với mô hình train và kết quả trả về như ảnh. Như có thế thấy dựa vào bài viết mô hình đã đánh giá được cảm xúc trong văn bản.



Hình 14.Mô hình LinearSVC thực hiện dự đoán trên dữ liệu mới



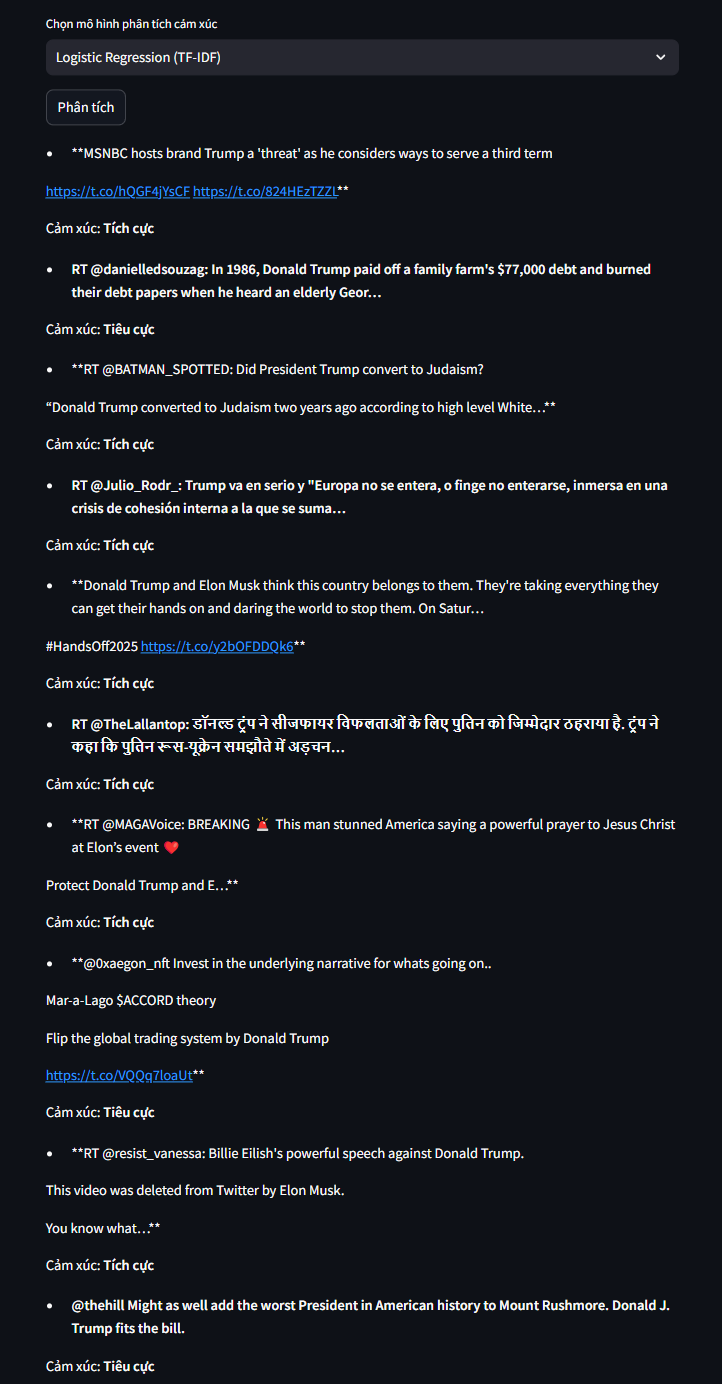
Hình 15.Mô hình LinearSVC thực hiện dự đoán trên các tweet mới qua API X

### **3.2.3. Logistic Regression**

Thực hiện test dữ liệu mới với mô hình train và kết quả trả về như ảnh. Như có thế thấy dựa vào bài viết mô hình đã đánh giá được cảm xúc trong văn bản.



Hình 16.Mô hình Logistic Regression thực hiện dự đoán trên dữ liệu mới



Hình 17.Mô hình Logistic Regression thực hiện dự đoán trên các tweet mới qua API X