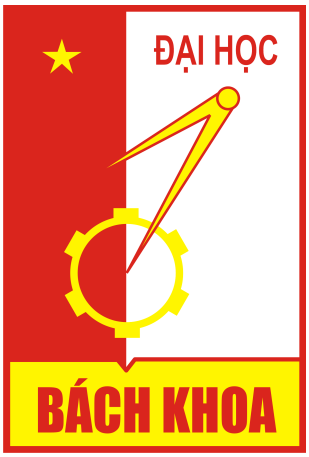
Đại học Bách khoa Hà Nội
Trường công nghệ Thông tin và Truyền thông



Bài tập lớn
Nhập môn Học máy và Khai phá dữ liệu
Đề tài: Sentiment Analysis

Giảng viên hướng dẫn: PGS. TS. Thân Quang Khoát

Nhóm sinh viên thực hiện:

Nguyễn Hồng Đăng	20220025
Ngô Duy Anh	20220069
Phùng Công Hiếu	20224848
Nguyễn Hoàng Xuân Sơn	20224896

Hà Nội, Ngày 20 tháng 5 năm 2025

Mục lục

Lời mở đầu

Trong bối cảnh thế giới đang chứng kiến những bước tiến nhảy vọt của khoa học và công nghệ, trí tuệ nhân tạo (AI) nổi lên như một trong những trụ cột then chốt, kiến tạo nên một kỷ nguyên công nghệ mới cho nhân loại. Và ở trung tâm của cuộc cách mạng này, không thể không nhắc đến *Học máy (Machine Learning - AI)* - một lĩnh vực nghiên cứu đầy "mê hoặc", nơi những thuật toán và mô hình toán học phức tạp được "huấn luyện" để có thể tự động học hỏi từ dữ liệu, nhận diện các thuộc tính ẩn trong dữ liệu, và đưa ra những dự đoán hoặc quyết định mà không cần đến sự can thiệp trực tiếp của con người.

Trong đó, học máy là lĩnh vực nghiên cứu và phát triển trong ngành khoa học máy tính, nhằm tập trung vào việc xây dựng các phương pháp và công nghệ giúp máy tính thực hiện những nhiệm vụ mà trước đây chỉ con người mới có khả năng thực hiện được. Mục tiêu chính của học máy là phát triển các hệ thống có khả năng tự học, tự hiểu, và thực hiện các nhiệm vụ thông minh mà không cần sự "giúp đỡ" từ con người.

Ứng dụng của học máy đa dạng và lan rộng sang nhiều lĩnh vực, bao gồm nhận diện giọng nói, xử lý ngôn ngữ tự nhiên, thị giác máy tính, ô tô tự lái, dự đoán và phân tích dữ liệu, chơi cờ với đối thủ người chơi, và nhiều lĩnh vực khác. Trong vài năm gần đây, lĩnh vực này đã chứng kiến sự phát triển mạnh mẽ không chỉ ở các quốc gia có nền kinh tế hàng đầu thế giới mà còn ở những quốc gia đang phát triển như Việt Nam. Sự ứng dụng của học máy hay trí tuệ nhân tạo đang ngày càng gia tăng, tạo ra những tác động tích cực trong xã hội.

Do đó, trong bài tập lớn của học phần "Nhập môn Học máy và Khai phá dữ liệu," nhóm chúng em đã quyết định chọn bài toán "Phân tích cảm xúc (Sentiment Analysis)" trên tập dữ liệu Stanford Sentiment Treebank. Chúng em đã tiến hành xây dựng và thử nghiệm một số phương pháp, kỹ thuật dựa trên các thuật toán và mô hình học máy (ML) và học sâu (DL) vào bài toán. Mục tiêu của chúng em là vận dụng những kiến thức đã học về học máy để giải quyết một bài toán mang tính ứng dụng cao trong việc hiểu ý kiến và thái độ của người dùng, đồng thời đánh giá hiệu quả của các phương pháp khác nhau trong việc "đọc vị" cảm xúc ẩn sau những dòng văn bản.

Báo cáo bao gồm 6 phần chính như sau:

- **Phần 1:** Giới thiệu bài toán
- **Phần 2:** Dữ liệu bài toán
- **Phần 3:** Mô hình đề xuất
- **Phần 4:** Kết quả thực nghiệm

- **Phần 5:** Kết luận
- **Phần 6:** Tài liệu tham khảo

.

1 Giới thiệu bài toán

Học máy - một nhánh quan trọng của trí tuệ nhân tạo, tập trung vào việc nghiên cứu và phát triển các phương pháp cho phép hệ thống tự động 'học' từ dữ liệu. Mục tiêu của học máy là xây dựng các kỹ thuật giúp máy móc giải quyết các nhiệm vụ cụ thể mà không cần được sự can thiệp trực tiếp của con người cho từng trường hợp. Một ví dụ điển hình là khả năng 'học' cách nhận diện và phân loại thư rác, tự động sắp xếp email vào các thư mục tương ứng. Ngày nay, học máy đã chứng minh được tính ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau, có thể kể đến như là công cụ tìm kiếm dữ liệu, hỗ trợ chẩn đoán y tế, phân tích thị trường chứng khoán,...

Khai phá dữ liệu (data mining) là một quá trình tính toán nhằm khám phá các thuộc tính tiềm ẩn trong các tập dữ liệu, trong đó có thể bao hàm các phương pháp kết hợp từ học máy, thống kê học,... Mục đích chính của khai phá dữ liệu là trích xuất thông tin có giá trị từ khối lượng lớn dữ liệu và chuyển đổi nó thành một định dạng dễ hiểu và hữu ích cho các ứng dụng thực tế.

Và một trong những nhóm bài toán phổ biến nhất, được các nhà nghiên cứu hay nhà khoa học ưa chuộng của học máy đó chính là phân loại đa lớp (multiclass classification). Với mục đích tìm hiểu, thử nghiệm, và so sánh các mô hình (model) và kỹ thuật (technique) phân loại đa lớp khác nhau, nhóm đã lựa chọn bài toán "Sentiment analysis" dựa trên bộ dữ liệu reviews-to-labels vô cùng nổi tiếng tới từ các nhà nghiên cứu ở Stanford, đó là bộ dữ liệu Stanford Sentiment Treebank (sẽ được trình bày kỹ hơn ở các phần sau).

Quá trình giải quyết bài toán sẽ được thực hiện qua các giai đoạn sau:

1. Chuẩn bị tập dữ liệu huấn luyện: Đây được coi là một trong những công đoạn quan trọng nhất, là đầu vào cho việc học để tìm ra mô hình giải quyết bài toán. Trong đó bao gồm những bước vô cùng quan trọng như tiền xử lý, làm sạch dữ liệu, lược bỏ những đặc trưng không tốt của dữ liệu hay xử lý các thông tin bị thiếu, tăng cường dữ liệu,v.v. Bước này cũng chuẩn bị dữ liệu để huấn luyện và đo đạc chất lượng mô hình, bao gồm tập huấn luyện (training) để huấn luyện mô hình, tập xác thực (validation) để kiểm định độ hiệu quả trong quá trình huấn luyện và tập kiểm tra để đo chất lượng của mô hình trong thực tế.

2. Xây dựng mô hình phân lớp (classifier model): xây dụng các mô hình xác định một tập các lớp dữ liệu. Ở đây chúng em xây dựng mô hình phân lớp từ các thuật toán học có giám giám sát như K-nearest Neighbor hay Naive Bayes,... hay các mô hình học sâu như LSTM, RNN nhằm phân lớp dữ liệu một cách tốt nhất.

3. Tiến hành các thực nghiệm để đánh giá sự hiệu quả của các mô hình, của các

phương pháp xử lý dữ liệu trên bộ dữ liệu validation.

2 Dữ liệu bài toán

2.1 Bộ dữ liệu

Bộ dữ liệu **Stanford Sentiment Treebank 2 (SST-2)** là một tập con của bộ dữ liệu Stanford Sentiment Treebank được phát triển bởi Socher et al. (2013), thuộc Đại học Stanford. Đây là một trong những bộ dữ liệu phổ biến nhất trong lĩnh vực xử lý ngôn ngữ tự nhiên (NLP), đặc biệt trong các bài toán phân tích cảm xúc (*sentiment analysis*).

SST-2 bao gồm các câu tiếng Anh ngắn được trích xuất từ các bài phê bình phim trên website *Rotten Tomatoes*. Mỗi câu được gán nhãn tương ứng với cảm xúc tích cực (*positive*) hoặc tiêu cực (*negative*), giúp giải quyết các bài toán phân loại nhị phân (*binary classification*).

Một số đặc điểm chính của bộ dữ liệu SST-2:

- **Ngôn ngữ:** Tiếng Anh
- **Số lượng mẫu huấn luyện:** Khoảng 67,000 câu
- **Tập kiểm tra:** Khoảng 872 câu
- **Số lượng nhãn:** Chỉ có 2 nhãn
  - **1**: Đại diện cho cảm xúc tích cực (*positive*).
  - **0**: Đại diện cho cảm xúc tiêu cực (*negative*).

Khác với phiên bản SST-1, dùng để phân loại cảm xúc thành 5 mức độ (rất tiêu cực đến rất tích cực), SST-2 chỉ giữ lại hai nhãn cơ bản nhằm đơn giản hóa bài toán, từ đó giúp mô hình dễ học hơn và đánh giá chính xác hơn trên nhiệm vụ phân loại nhị phân.

Bộ dữ liệu SST-2 hiện được tích hợp trong nhiều thư viện NLP phổ biến như *Hugging Face Transformers*, *TensorFlow Datasets*, v.v., và thường được sử dụng như một tập benchmark trong bộ tiêu chuẩn *GLUE (General Language Understanding Evaluation)*. Với tính chất rõ ràng, quy mô vừa phải và chất lượng nhãn cao, SST-2 là lựa chọn phù hợp cho các nghiên cứu và thử nghiệm trong bài toán phân tích cảm xúc.

Về bản chất, SST-2 gồm các **treebanks**, văn bản có kèm theo thông tin bổ sung về ngữ nghĩa của câu dưới dạng cây. Phương pháp xử lý ngôn ngữ tự nhiên sử dụng các treebank được sử dụng nhiều và tương đối hiệu quả trong thời kì đầu, nhưng có vấn đề lớn là quá trình bổ sung thêm ngữ nghĩa này phải được thực hiện một cách thủ công nên dần dần bị thay thế bởi các việc xử lý văn bản thô. Vì thế, các mô hình học máy và mạng nơ-ron được

trình bày sẽ chỉ sử dụng phần dữ liệu văn bản để đưa ra kết luận, mà không quan tâm đến các cây ngữ nghĩa trong treebank.

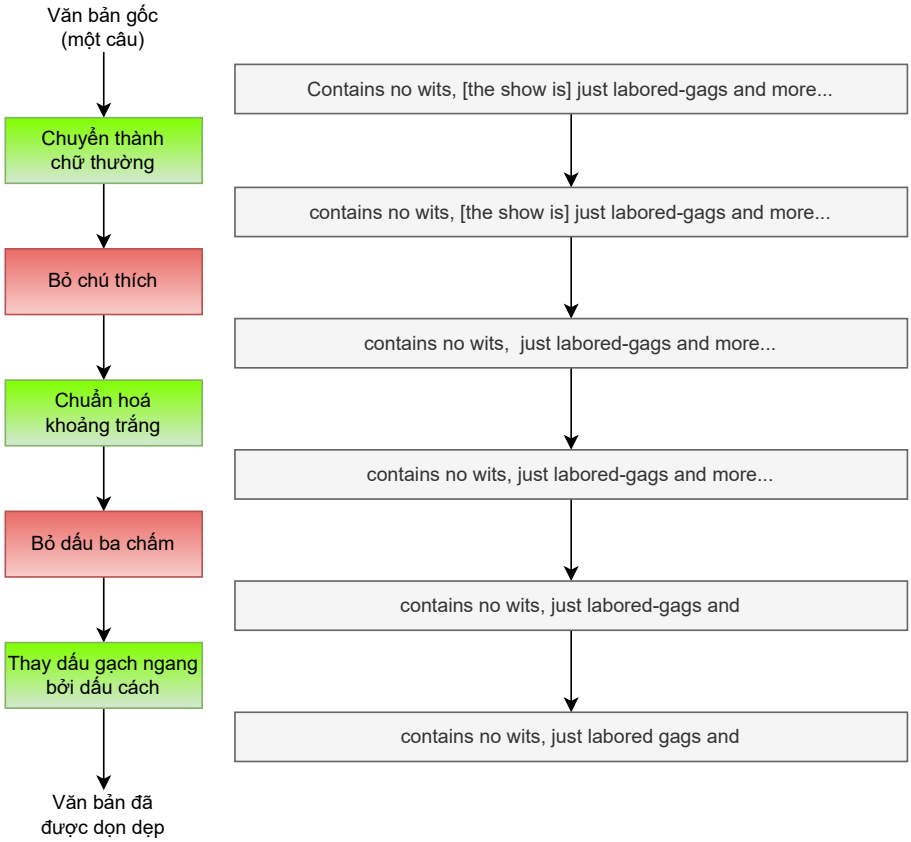
2.2 Tiền xử lý dữ liệu

Trong bài toán phân tích cảm xúc, chất lượng của bước tiền xử lý dữ liệu đóng vai trò then chốt đến hiệu quả huấn luyện và khả năng tổng quát của mô hình. Do dữ liệu đầu vào của hệ thống thuộc dạng văn bản thô, thường chứa nhiều yếu tố không liên quan, nhiễu và sự không nhất quán có thể ảnh hưởng đáng kể đến hiệu suất của các mô hình phân tích tình cảm, tiền xử lý dữ liệu văn bản là một bước thiết yếu để chuyển đổi văn bản thô thành một định dạng sạch hơn, có cấu trúc hơn và phù hợp hơn cho việc phân tích và mô hình hóa.

SST-2 là một tập dữ liệu phổ biến được sử dụng để phân loại tình cảm nhị phân, bao gồm các câu đơn trích từ các bài đánh giá phim, được gán nhãn là tích cực hoặc tiêu cực. Đánh giá phim được coi là một nguồn văn bản tương đối sạch so với các nguồn dữ liệu khác từ các dịch vụ mạng xã hội, và hơn nữa, SST-2 còn được những nhà nghiên cứu tạo ra nó clean lại tương đối kỹ lưỡng so với các bộ dữ liệu khác dùng trong nghiên cứu và thực nghiệm. Mỗi mẫu dữ liệu là một câu riêng biệt, không phải đoạn văn hoặc bài review dài, các review trung lập đều bị bỏ đi, chỉ giữ lại các câu có cảm xúc tích cực hoặc tiêu cực, có ít lỗi chính tả, ngữ pháp.

Tuy nhiên, việc áp dụng các bước tiền xử lý dữ liệu vẫn có khả năng nâng cao hiệu suất của mô hình một lượng tương đối đáng kể, cũng như hiệu năng của quá trình huấn luyện và thử nghiệm mô hình. Tiền xử lý không chỉ đơn thuần là làm sạch dữ liệu mà còn là quá trình chuyển đổi dữ liệu để làm nổi bật các đặc trưng thông tin nhất cho nhiệm vụ phân tích tình cảm cụ thể trên tập dữ liệu SST-2. Điều này bao gồm việc đưa ra các quyết định có khả năng ảnh hưởng đến khả năng của mô hình trong việc nắm bắt các sắc thái ngôn ngữ liên quan đến tình cảm. Ngoài ra, việc thực hiện các luật tiền xử lý và dọn dẹp dữ liệu giúp cho phương pháp có khả năng tổng quát hoá tốt hơn: phương pháp có thể vận hành tốt trên các bộ dữ liệu không được "gọn gàng" như SST-2.

2.2.1 Dọn dẹp câu



Hình 1: Các bước dọn dẹp câu được sử dụng

Các bước trong quá trình dọn dẹp dữ liệu được xây dựng theo quy trình chuẩn của các phương pháp dọn dẹp dữ liệu có dạng văn bản thô, được mô tả trong Hình 1.

Bước đầu tiên trong quy trình tiền xử lý là chuyển đổi tất cả ký tự trong văn bản đầu vào thành chữ thường. Mục đích của việc này là để chuẩn hóa văn bản bằng cách xử lý các từ có cùng ý nghĩa như nhau, bất kể cách viết hoa ban đầu của chúng. Ví dụ, "Movie" và "movie" sẽ được coi là cùng một từ. Thông qua bước này, số lượng từ khác nhau trong bộ dữ liệu được giảm đi tương đối đáng kể, giúp giảm độ phức tạp của dữ liệu và cải thiện độ chính xác của mô hình. Việc viết thường đảm bảo tính nhất quán và giảm số chiều dữ liệu, cho phép mô hình tập trung vào ý nghĩa ngữ nghĩa hơn là sự khác biệt về hình thức. Tuy nhiên, việc sử dụng chữ thường hay chữ hoa cũng có thể mang một sắc thái tình cảm nào đó, chẳng hạn như một người có viết hoa toàn bộ một từ nhằm nhấn mạnh vào ý nghĩa của từ đó.

Sau đó, bất kỳ nội dung nào được chứa trong dấu ngoặc vuông "[ ]" bị loại bỏ. Bước

này thường được sử dụng để loại bỏ các chú thích, siêu dữ liệu hoặc thông tin không cần thiết khác có thể xuất hiện trong văn bản nhưng không đóng góp vào việc phân tích tình cảm. Việc loại bỏ những yếu tố này giúp làm sạch văn bản và tập trung vào nội dung chính.

Để đảm bảo tính nhất quán và tránh các vấn đề có thể phát sinh do nhiều khoảng trắng giữa các từ, bước này sẽ loại bỏ mọi trường hợp có nhiều khoảng trắng liên tiếp và thay thế chúng bằng một khoảng trắng duy nhất. Điều này giúp chuẩn hóa cấu trúc của văn bản và tạo điều kiện thuận lợi cho các bước xử lý tiếp theo như token hóa.

Dấu ba chấm "..." thường được sử dụng để biểu thị sự lược bỏ hoặc bỏ lửng trong văn bản. Bước này sẽ loại bỏ dấu ba chấm và cả từ đứng ngay trước nó. Việc này nhằm mục đích loại bỏ các phần văn bản không hoàn chỉnh hoặc bị cắt xén có thể không mang lại nhiều giá trị cho việc phân tích tình cảm.

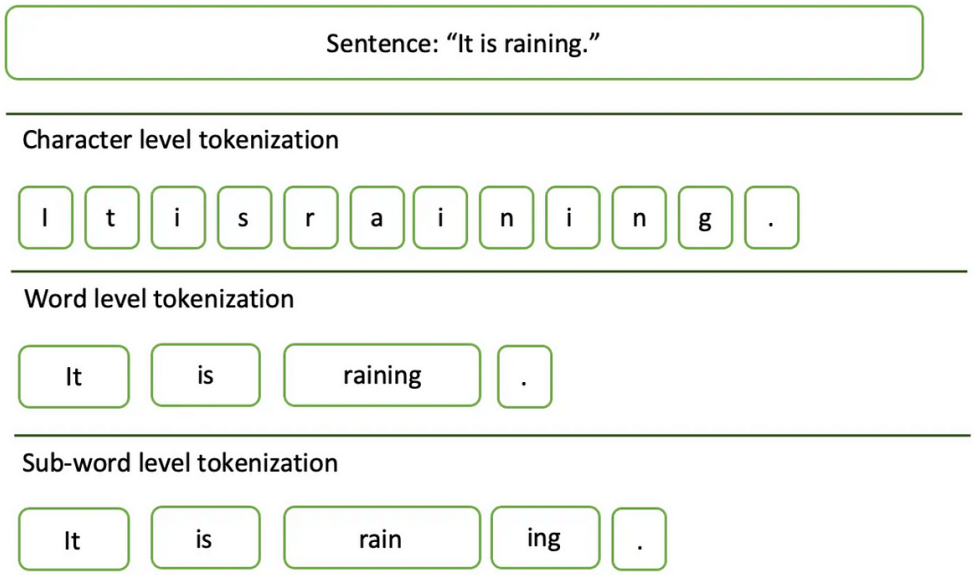
Trong nhiều trường hợp, dấu gạch ngang "-" có thể được sử dụng để nối các từ hoặc tạo thành các từ ghép. Để xử lý các trường hợp này một cách nhất quán, bước này sẽ thay thế dấu gạch ngang giữa các ký tự chữ và số bằng một khoảng trắng. Điều này giúp tách các từ ghép thành các từ riêng biệt, cho phép phân tích chúng một cách độc lập.

Dấu câu có thể thêm vào sự phức tạp không cần thiết cho phân tích văn bản và thường được loại bỏ trong quá trình tiền xử lý. Mặc dù trong một số trường hợp, dấu câu có thể mang thông tin về tình cảm (ví dụ: dấu chấm than biểu thị sự nhấn mạnh) , nhưng trong nhiều nhiệm vụ phân tích tình cảm, việc loại bỏ chúng giúp giảm nhiễu và chuẩn hóa dữ liệu.

2.2.2 Tách câu và token hoá

Sau khi loại bỏ các dấu câu và chú thích mang tính gây nhiễu trong văn bản, mỗi câu có thể được tách thành các đơn vị nhỏ hơn, gọi là tokens, thường là từ, cụm từ, hoặc thậm chí là các đơn vị ngữ nghĩa nhỏ hơn như subwords hoặc ký tự. Các tokens này là đầu vào cho hầu hết các mô hình học máy và deep learning được sử dụng để phân tích cảm xúc.

Quá trình token hóa giúp biến dữ liệu ngôn ngữ – vốn không có cấu trúc và khó xử lý trực tiếp – thành dạng có cấu trúc hơn, phù hợp để biểu diễn bằng vector số. Việc lựa chọn chiến lược token hóa phù hợp đóng vai trò quan trọng trong việc bảo toàn ngữ nghĩa và hiệu quả học của mô hình.



Hình 2: Ba phương pháp token hoá thông dụng trong Xử lý ngôn ngữ tự nhiên

Có nhiều phương pháp token hóa phổ biến, tùy thuộc vào ngôn ngữ và mô hình được sử dụng như được mô tả trong Hình 2.

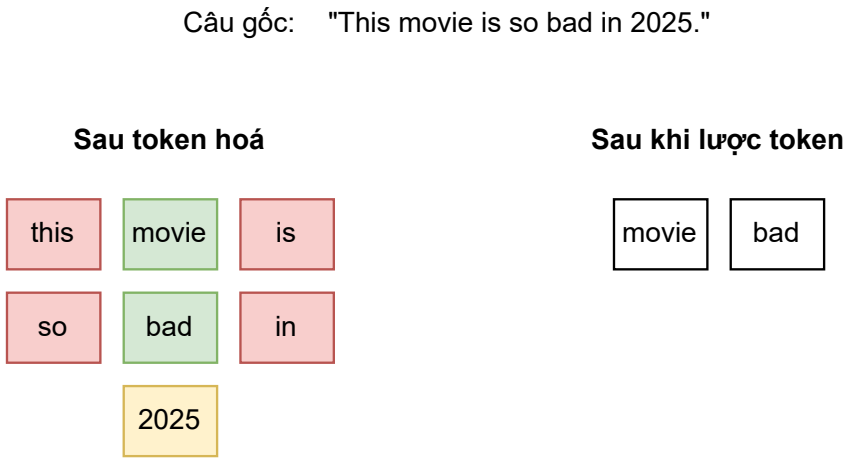
- **Token hóa theo từ (word-level)**: Đây là phương pháp token hoá ngây thơ và dễ thực hiện nhất. Để thực hiện token hoá theo từ, ta chỉ sử dụng các ký tự khoảng trống và coi đó là điểm tách giữa các token. Phương pháp này kém hiệu quả đối với các ngôn ngữ mà các từ không được phân cách bởi dấu cách, chẳng hạn như tiếng Việt hoặc tiếng Trung Quốc. Chẳng hạn, “học sinh” là một từ, nhưng nếu token hóa theo dấu cách, sẽ bị tách thành “học” và “sinh” – sai về mặt ngữ nghĩa. Tuy SST-2 là bộ dữ liệu chỉ gồm văn bản tiếng Anh, việc token hoá theo từ vẫn bộc lộ nhiều hạn chế, chẳng hạn như “they’re”, tuy là một từ trong tiếng Anh, thực chất mang ngữ nghĩa của “they are”, là hai từ phân biệt.
- **Token hóa ký tự (character-level)**: Ở một khía cạnh khác, khi xử lý với văn bản có nhiều lỗi chính tả phổ biến trên mạng, đôi khi các lỗi chính tả có thể làm kém tính hiệu quả của việc token hoá. Token hoá ký tự có khả năng chống lỗi tốt đối với các trường hợp người dùng gõ sai hoặc dùng nhiều từ lóng, nhưng có khả năng học ngữ nghĩa kém hơn đáng kể so với hai phương pháp kể trên. Ngoài ra, do mỗi token là một ký tự, nên phương pháp này tránh gặp phải vấn đề từ ngoài từ điển (OOV), song có số lượng token lớn hơn đáng kể.
- **Token hóa subword (subword-level)**: Token hóa theo subword là kỹ thuật phân tách văn bản thành các đơn vị nhỏ hơn từ (gọi là subword), thay vì tách hoàn toàn

theo từ (word-level) hoặc ký tự (character-level). Đây là một phương pháp cực kỳ phổ biến trong các mô hình NLP hiện đại như BERT, RoBERTa, GPT, PhoBERT,... vì nó kết hợp được ưu điểm của cả word-level và character-level.

Phương pháp token hoá được sử dụng trong đồ án là phương pháp token hoá subword dựa trên các biểu thức chính quy (regular expression) cải thiện từ phương pháp sử dụng trong việc xây dựng Penn Treebank. Phương pháp này được cung cấp bởi mô-đun nltk, một mô-đun xử lý ngôn ngữ tự nhiên thông dụng trong học thuật và nghiên cứu. Các từ ghép như "they're" được tách một cách chính xác thành các subword như "they" và "'re".

Sau khi tách một câu thành nhiều token, ta tiếp tục loại bỏ một số token thừa như sau:

- **Stop word**: các từ phổ biến trong một ngôn ngữ (ví dụ: "the", "a", "is") mà thường không mang nhiều ý nghĩa cho việc phân tích. Bước này loại bỏ các token xuất hiện trong một danh sách các stop word được xác định trước (thường là các stop word tiếng Anh). Việc loại bỏ stop word có thể giúp giảm kích thước dữ liệu và tập trung vào các từ quan trọng hơn đối với việc xác định tình cảm. Tuy nhiên, cần lưu ý rằng trong một số trường hợp, việc loại bỏ stop word có thể làm mất đi ngữ cảnh quan trọng.
- **Số**: Trong SST-2 có nhiều token chỉ chứa chữ số (các thông tin ngày, tháng, năm, tuổi tác, v.v.). Các chữ số thường không mang lại nhiều thông tin cho phân tích tình cảm, trừ khi chúng là một phần của một thực thể có tên hoặc một biểu thức cụ thể. Do đó, ta loại bỏ bất kỳ token nào chỉ chứa chữ số.
- **Token ngắn**: Các token quá ngắn (ví dụ: một hoặc hai ký tự) thường không mang nhiều ý nghĩa ngữ nghĩa và có thể là kết quả của lỗi token hóa hoặc các ký tự còn sót lại sau các bước làm sạch khác. Bước này loại bỏ bất kỳ token nào có độ dài nhỏ hơn hoặc bằng một ký tự, giúp tập trung vào các từ có ý nghĩa hơn.



Hình 3: Tác dụng của việc lược bỏ token sau token hoá. Các token đỏ là các stop words, còn token 2025 màu vàng chỉ gồm chữ số.

2.2.3 Vector hoá văn bản

Sau khi văn bản đã được chia thành các token rời rạc và phân biệt và các token thừa được loại bỏ, ta cần thêm một bước xử lý để chuyển dạng biểu diễn của các token thành dạng số để các mô hình học máy có thể dễ dàng xử lý và thực hiện. Quá trình này gọi là quá trình vector-hoá (vectorization process).

Phương pháp vector hoá đơn giản nhất có thể được thiết kế như sau. Đầu tiên, từ kết quả token hoá của các câu thành các tập token $S_1, S_2, ..., S_n$, ta xây dựng một ngữ vựng (vocabulary) V gồm tất cả các token xuất hiện:

$$V = \bigcup_{k=1}^{n} S_k.$$

Sau đó, với mỗi token $t_i \in S$, ta đếm số lần xuất hiện $c_{i,k}$ của t trong mỗi câu s_k. Khi đó, ta có thể vector-hoá câu s_k thành một vector nguyên $|S|$ chiều v_k như sau:

$$v_k = \begin{pmatrix} c_{1,k} \\ c_{2,k} \\ \cdots \\ c_{|S|,k} \end{pmatrix}.$$

Cuối cùng, các vector $v_1, v_2, ...v_n$ có thể được đưa vào một ma trận để dễ dàng kiểm

soát hơn. Một đặc điểm quan trọng của ma trận này là hầu hết các phần tử của nó là 0, do đó ma trận này được biểu diễn dưới dạng ma trận thưa trong máy tính. Các phương pháp lưu trữ thông dụng cho ma trận thưa gồm có dạng từ điển (Dictionary of keys, DOK), dạng danh sách (List of lists, LIL), danh sách toạ độ (Coordinate list, COO) và hàng/cột thưa thớt được nén (Compressed sparse row/column, CSR/CSC).

Với bộ dữ liệu SST-2 gồm hơn 67,000 câu, giả sử như ta ước tính số token phân biệt là 200,000, thì ma trận thu được sau quá trình vector hoá có thể chứa tới hàng tỉ phần tử, nghĩa là lượng bộ nhớ cần thiết để lưu trữ ma trận biểu diễn ngây thơ (dạng mảng hai chiều) có thể tốn đến hàng GB. Tệ hơn nữa, khi thực hiện tính toán trên ma trận này, kích thước lớn này cũng có thể làm chậm thời gian huấn luyện đi đáng kể. Khi đó, việc lưu ma trận dưới dạng ma trận thưa khi đó làm cải thiện hiệu năng lên đáng kể.

Phương pháp vector hoá sử dụng tần suất xuất hiện của từng token trên được hỗ trợ bởi thư viện sklearn với mô-đun CountVectorizer. Mô-đun này thực hiện lưu danh sách các token trong S trong bộ nhớ, và dựa vào đó thực hiện vector hoá các câu trong bộ dữ liệu.

Một cải thiện của CountVectorizer cho các bộ dữ liệu lớn là HashingVectorizer. Do việc lưu trữ các phần tử trong V có thể rất khó khăn khi $|V|$ lớn, HashingVectorizer sử dụng hàm băm để tránh phải lưu trữ các token, do đó có độ phức tạp bộ nhớ $O(1)$. Tuy nhiên, hàm băm luôn gặp phải vấn đề hash collision, hai token khác nhau và không liên quan đến nhau có thể được băm về cùng một giá trị. Ngoài ra, nếu CountVectorizer cho phép tìm ngược từ t_i từ chỉ số i, thì tính chất băm của HashingVectorizer làm việc tìm ngược từ từ kết quả băm một cách hiệu quả là một điều bất khả thi. Do dữ liệu của SST-2 không quá nhiều và vẫn có thể được lưu trữ hiệu quả trong bộ nhớ máy tính, HashingVectorizer không được sử dụng trong quá trình thực nghiệm của nhóm.

Một phương pháp vector hoá phổ biến khác là sử dụng **TF-IDF** (Term Frequency – Inverse Document Frequency). Phương pháp này không chỉ tính đến tần suất xuất hiện của từng token trong một câu (TF), mà còn điều chỉnh trọng số dựa trên mức độ phổ biến của token đó trong toàn bộ tập dữ liệu (IDF). Ý tưởng chính là các từ xuất hiện phổ biến trong hầu hết các câu (chẳng hạn như những từ stop words "the", "is", ...) sẽ mang ít thông tin đặc trưng và do đó nên có trọng số thấp hơn, trong khi những từ hiếm gặp nhưng đặc trưng cho nội dung sẽ được gán trọng số cao hơn.

Với một tập dữ liệu như đã mô tả trên, ta tính được **tần suất xuất hiện** (TF) của một token t_i trong câu s_k bằng cách chia số lần xuất hiện $c_{i,k}$ của t_i cho số token trong câu này:

$$\text{tf}(t_i, s_k) = \frac{c_{i,k}}{\sum_{t_j \in S} c_{j,k}}.$$

Sau đó, ta tính được **tần suất đảo ngược tài liệu** (IDF) của token t_i bằng công thức:

$$\text{idf}(t_i) = \log \frac{N}{1 + N(t_i)},$$

trong đó $N(t_i) = |s_k : t_i \in S_k|$ là số câu có chứa token t_i. Ta cộng thêm 1 ở mẫu số nhằm tránh tình trạng chia cho 0.

Khi đó, ta tính được giá trị TF-IDF bằng cách nhân giá trị TF và IDF tương ứng:

$$\text{tf-idf}(t_i, s_k) = \text{tf}(t_i, s_k) \cdot \text{idf}(t_i).$$

Thư viện sklearn cung cấp mô-đun TfidfVectorizer giúp tự động tính toán cả TF và IDF cho mỗi token trong tập huấn luyện, sau đó kết hợp hai thành phần này lại để tạo ra vector đặc trưng cho từng câu. Khác với CountVectorizer, TfidfVectorizer giúp giảm ảnh hưởng của các từ mang tính "nhiễu" (high-frequency noise words) trong mô hình học máy, đồng thời giữ lại những thông tin quan trọng mang tính đặc trưng cho văn bản.

Từ góc nhìn trực giác, TF giúp mô hình "tập trung" vào những token xuất hiện nhiều trong một câu cụ thể, trong khi IDF giúp "giảm nhiễu" từ những token xuất hiện quá phổ biến trên toàn tập dữ liệu. Việc kết hợp cả hai thành phần này giúp tăng khả năng phân biệt giữa các câu mang sắc thái cảm xúc khác nhau trong tập dữ liệu SST-2.

Trong thực nghiệm của nhóm, TfidfVectorizer được sử dụng như một lựa chọn thay thế cho CountVectorizer trong một số mô hình tuyến tính. Do dữ liệu SST-2 có kích thước vừa phải, việc tính toán và lưu trữ các trọng số TF-IDF có thể được thực hiện hiệu quả mà không gây ra vấn đề về hiệu năng hoặc bộ nhớ.

Ngoài hai phương pháp vector hoá sử dụng tần số các token như đã trình bày trên, một phương pháp vector hoá thông dụng trong các năm gần đây là phương pháp Embedding. Kỹ thuật embedding nhằm biểu diễn văn bản dưới dạng vector một cách ngữ nghĩa hơn, thay vì chỉ dựa vào tần suất từ, ánh xạ các từ trong văn bản thành các vector số thực có số chiều thấp, khắc phục nhược điểm của CountVectorizer và TF-IDF là không nắm bắt được ngữ nghĩa hay ngữ cảnh của từ.

Chẳng hạn, ta có thể đo được sự tương đồng giữa 2 từ có vector embedding $\mathbf{u}$ và $\mathbf{v}$

bằng các độ do similarity như độ đo cosine:

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}.$$

Giống như cách hai vector càng gần nhau thì cosine của góc giữa chúng gần nhau, hai từ có ngữ nghĩa liên quan đến nhau thì sẽ có độ đo cosine càng lớn và ngược lại. Ngoài ra, ta còn có thể thực hiện các phép toán đối với các vector embedding để tìm ra embedding của các từ mới: một ví dụ được sử dụng trong https://arxiv.org/abs/1301.3781 là khi tính

$$X = \text{embed}(\text{“biggest”}) + \text{embed}(\text{“big”}) + \text{embed}(\text{“small”}),$$

kết quả thu được là một vector gần với vector embedding của từ “smallest”.

Có thể thấy rằng, phương pháp embedding cho phép mô hình học được các đặc trưng ngữ nghĩa sâu sắc hơn từ văn bản, nhờ đó cải thiện hiệu suất của các mô hình downstream như phân loại, phân cụm, hay phân tích cảm xúc ta đang xét đến. Khác với các phương pháp vector hóa truyền thống như CountVectorizer hay TF-IDF vốn chỉ xem xét tần suất từ xuất hiện, embedding cho phép mô hình hiểu được các mối quan hệ như đồng nghĩa, phản nghĩa, hay vai trò từ vựng trong ngữ cảnh cụ thể.

Tuy nhiên, việc học ra embedding là một bài toán khó, vì đây là một bài toán học không giám sát (unsupervised learning). Cụ thể:

- Ta không có “đáp án đúng” về việc vector biểu diễn của một từ phải như thế nào.
- Dữ liệu đầu vào chỉ đơn thuần là một tập văn bản lớn, nơi các từ xuất hiện theo những ngữ cảnh tự nhiên.
- Không có hàm mất mát rõ ràng để đánh giá chất lượng embedding một cách trực tiếp, như trong các bài toán giám sát. Ta chỉ có thể đánh giá embedding qua các bài toán phụ như phân cụm từ, phân tích quan hệ ngữ nghĩa, hoặc hiệu suất khi áp dụng embedding vào các tác vụ cụ thể (ví dụ: phân loại cảm xúc, dịch máy, QA,...).

Do đó, việc tạo ra một embedding thực chất là thiết kế ra một bài toán phụ đòi hỏi embedding phải học được càng nhiều mối quan hệ giữa các từ nhất có thể, từ đó áp dụng embedding này trong các bài toán phụ khác. Bài toán phụ gốc cần phải được thiết kế một cách khéo léo để giúp embedding học được nhiều ngữ nghĩa nhất, và quan trọng hơn là phải phù hợp với mô hình unsupervised learning hiện tại: nếu ta dùng bài toán phụ là một bài toán phân loại cảm xúc thì cần có một bộ dữ liệu đã gán nhãn cảm xúc, vốn khó tạo ra hơn so với các bộ dữ liệu văn bản thô.

Word2Vec ra đời vào 2013, là một trong những mô hình tiên phong trong việc học biểu diễn từ dưới dạng vector liên tục. Để học được embedding, hai bài toán phụ được sử dụng, dẫn đến hai họ mô hình Word2Vec chính. Ý tưởng của cả hai mô hình là để học được embedding của một từ t_i, ta xét các từ xung quanh nó $C_i = \{t_{i-K}, t_{i-K+1}, \ldots, t_{i-1}, t_{i+1}, ...t_{i+K}\}$. Hai họ mô hình được phân loại theo mục tiêu:

- **Continuous Bag of Words (CBOW)**: tối đa hoá $\prod_i P(t_i|C_i)$, nghĩa là xác suất đoán đúng từ dựa trên những từ xung quanh. Mỗi ngữ cảnh C_i được embed thành vector v_i là trung bình các vector embedding của các từ trong C_i, và xác suất $P(t_i|C_i)$ tỉ lệ với độ đo tích vô hướng (sau khi chuẩn hoá dương bằng cách đưa qua hàm mũ):

$$v_i = \frac{1}{|C_i|} \sum_{t \in C_i} \text{embed}(t), P(t_i|C_i) = \frac{\exp(\text{embed}(t_i) \cdot v_i)}{\sum_{t \in V} \exp(\text{embed}(t) \cdot v_i)} \propto \exp(\text{embed}(t_i) \cdot v_i).$$

- **Skip-gram**: $\prod_i P(C_i|t_i)$, nghĩa là xác suất đoán đúng các từ xung quanh từ từ nằm giữa. Đặt $N_i = \{i-K, i-K+1, \ldots, i-1, i+1, ...i+K\}$ là chỉ số các từ ngữ cảnh của t_i, ta tính $P(C_i|t_i)$ thông qua giả thiết rằng các từ trong C_i độc lập có điều kiện với nhau:

$$P(C_i|t_i) = \prod_{j \in N_i} P(t_j|t_i),$$

  trong đó các xác suất $P(t_j|t_i)$ tính tương tự như trên mô hình CBOW, chỉ khác là ta bỏ qua bước tính trung bình (vì ngữ cảnh t_i coi như chỉ chứa một từ):

$$P(t_j|t_i) = \frac{\exp(\text{embed}(t_j) \cdot \text{embed}(t_i))}{\sum_{t \in V} \exp(\text{embed}(t) \cdot \text{embed}(t_i))} \propto \exp(\text{embed}(t_j) \cdot \text{embed}(t_i)).$$

GloVe (viết tắt của Global Vectors for Word Representation) là một mô hình embedding được giới thiệu bởi các nhà nghiên cứu tại Stanford vào năm 2014. Khác với Word2Vec -vốn học từ trực tiếp trên các cặp từ gần nhau, GloVe khai thác thông tin thống kê toàn cục từ toàn bộ tập văn bản bằng cách xây dựng một ma trận đồng xuất hiện (co-occurrence matrix).

Ý tưởng của GloVe là như sau: ước tính $P(t'|t)$ thông qua đếm từ, và qua đó học được embedding t_i và t_j mà qua mối quan hệ với các từ trung gian t'':

$$F(t, t', t'') = \frac{P(t''|t)}{P(t''|t')}.$$

Tỉ lệ này có thể được hiểu là độ sai khác ngữ nghĩa[1] tương đối với t''. Chẳng hạn, hai từ "ice" và "water" khác nhau ở nhiều phương diện: độ cứng, trạng thái, nhiệt độ, ..., nhưng ta chỉ chiếu lên ngữ cảnh của từ t'', chẳng hạn là "cold", thì thu được giá trị số có ý nghĩa là "ice" "cold" hơn "water" bao nhiêu. Trong embedding, phép đo sai khác ngữ nghĩa là trừ vector, còn phép chiếu là phép tích vô hướng, nên ta có thể viết $F = g((v - v') \cdot \tilde{v}'')$, trong đó v và v' lần lượt là embedding của t và t', còn $\tilde{v}''$ là vector embedding ngữ cảnh[2] của t''. Do g biến phép trừ thành chia, một sự lựa chọn tự nhiên là lấy $g = \exp$. Như vậy:

$$v \cdot \tilde{v}'' = \log P(t''|t) + \underbrace{v' \cdot \tilde{v}'' - \log P(t''|t')}_{\tilde{b}(t'')}.$$

Cuối cùng, ước tính $P(t''|t) = \frac{X(t'',t)}{X(t)}$, trong đó $X(t'', t)$ là số lần t'' xuất hiện trong ngữ cảnh của t và $X(t)$ là số lần xuất hiện của t, để đảm bảo tính đối xứng (có thể đảo vị trí vector embedding với vector embedding ngữ cảnh và công thức thu được vẫn đúng), ta thu được công thức $v \cdot \tilde{v}'' + b(t) + \tilde{b}(t'') = \log X(t'', t)$. Như vậy, GloVe xây dựng bài toán phụ dựa trên đẳng thức này, thực hiện huấn luyện sao cho tổng lỗi bình phương (có trọng số) ít nhất:

$$L = \sum_{v,v''} f(X(t,t''))(v \cdot \tilde{v}'' + b(t) + \tilde{b}(t'') - \log X(t'', t))^2,$$

trong đó trọng số $f(x) = \min\{1, \frac{x}{x_{max}}\}^\alpha, x_{max} = \max_{t,t''} X(t, t'')$.

Trong thực nghiệm, nhóm sẽ sử dụng GloVe làm phương pháp embedding chủ yếu. Việc áp dụng Word2Vec có thể được xem xét trong tương lai.

2.3 Tăng cường dữ liệu

So với các bộ dữ liệu khác như IMDB Dataset (gồm 50,000 *đoạn* đánh giá phim trên trang web IMDb), bộ dữ liệu SST-2 tương đối nhỏ, chỉ chứa khoảng 67,000 *câu* ngắn (bao gồm cả train, dev và test). Do đó, mô hình dễ gặp phải hiện tượng overfitting và khó tổng quát hóa khi gặp những biểu đạt cảm xúc đa dạng trong thực tế. Để giải quyết vấn đề này, nhóm đã xem xét và áp dụng các kỹ thuật tăng cường dữ liệu (data augmentation) nhằm mở rộng tập huấn luyện mà không cần thu thập thêm dữ liệu thật.

Ngoài việc cải thiện chất lượng mô hình, tăng cường dữ liệu còn giúp hệ thống trở nên

[1] Độ sai khác ngữ nghĩa: hai từ t và t' khác nhau ở đâu (man vs. woman); Độ khác biệt: ý nghĩa hai từ t và t' không liên quan thế nào (dog vs. building). Hai khái niệm này dễ gây nhầm lẫn.

[2] Vector embedding ánh xạ một từ đến một điểm trong không gian vector embedding, còn vector embedding ngữ cảnh ánh xạ một từ đến một "hướng" đại diện cho sự ảnh hưởng của từ đó lên các từ xung quanh. Về mặt toán học, quan hệ giữa hai loại vector này giống như quan hệ giữa không gian vector và không gian đối ngẫu của nó.

mạnh mẽ hơn trước các biến thể ngôn ngữ trong thực tế, đặc biệt là trong môi trường đa dạng như mạng xã hội hoặc các ứng dụng chatbot. Việc tạo ra nhiều cách diễn đạt khác nhau cho cùng một ý nghĩa giúp mô hình học được những biểu hiện cảm xúc linh hoạt hơn, từ đó nâng cao khả năng phân loại đúng trong các trường hợp chưa từng gặp.

Kỹ thuật tăng cường dữ liệu xuất hiện từ khá sớm trong thị giác máy tính (computer vision), nơi mà các phép biến đổi như xoay ảnh, cắt, phóng to/thu nhỏ, lật ngang/lật dọc đã chứng minh hiệu quả rõ rệt trong việc cải thiện độ chính xác của các mô hình phân loại hình ảnh như CNN. Ví dụ điển hình là mô hình AlexNet (2012), khi sử dụng các phép biến đổi dữ liệu đơn giản nhưng đã giúp mạng học được những đặc trưng hình ảnh mạnh mẽ hơn và giành chiến thắng trong cuộc thi ImageNet.

Trong xử lý ngôn ngữ tự nhiên (nói chung) và bài toán phân biệt cảm xúc (nói riêng), tăng cường dữ liệu từng gặp nhiều thách thức vì tính nhạy cảm của ngôn ngữ – chỉ một thay đổi nhỏ về từ ngữ có thể làm sai lệch ý nghĩa hoặc nhãn. Tuy nhiên, với sự phát triển của các mô hình ngôn ngữ và công cụ xử lý ngữ nghĩa, các kỹ thuật tăng cường dữ liệu cho NLP ngày càng đa dạng và hiệu quả hơn. Đặc biệt trong các bài toán phân tích cảm xúc, nơi mà cách diễn đạt cảm xúc rất phong phú, tăng cường dữ liệu giúp mô hình hiểu được nhiều cách biểu đạt cảm xúc khác nhau, ngay cả khi dữ liệu huấn luyện còn hạn chế.

Các phương pháp tăng cường cơ bản trong xử lý ngôn ngữ tự nhiên gồm có:

- **Thay từ đồng nghĩa**: Một số từ (không phải stop word) trong câu được chọn ngẫu nhiên và thay bằng các từ đồng nghĩa, giúp tạo ra câu mới có cùng nghĩa nhưng khác bề mặt.

  Ví dụ: “this film is great” thành “this movie is amazing”. Hai từ “film” and “great“ được thay thế bởi từ đồng nghĩa với chúng, lần lượt là “movie” và “amazing”.

- **Chèn từ ngẫu nhiên**: Tìm một từ đồng nghĩa với một từ ngẫu nhiên (không phải stop word) trong câu và chèn vào vị trí ngẫu nhiên trong câu.

  Ví dụ: “this article will focus on summarizing data augmentation techniques in NLP” thành “This article will focus on **write-up** summarizing data augmentation techniques in NLP **methods**.” Hai từ được thêm vào, “write-up” và “methods” là từ đồng nghĩa của “article” và “techniques”.

- **Đảo từ ngẫu nhiên**: Đảo vị trí của hai từ ngẫu nhiên (không phải stop word) cho nhau.

  Ví dụ: “but I’m still hoping for the radiance” thành “but I’m still radiance for the hoping”. Hai từ “hoping” và “radiance” được đảo vị trí với nhau.

- **Xoá từ ngẫu nhiên**: Xoá ngẫu nhiên một số từ trong câu (trừ stop word), giúp mô hình học được tính khái quát hơn.

  Ví dụ: "she absolutely loves the performance" thành "she loves the performance". Từ "absolutely" bị xoá đi.

Bốn phương pháp cơ bản trên được tích hợp vào thành một phương pháp tăng cường Easy Data Augmentation (EDA). Với mỗi câu s gồm l từ, EDA thực hiện các phép thay từ đồng nghĩa, chèn từ ngẫu nhiên và đảo từ ngẫu nhiên $n = \alpha l$ lần, và xoá từ ngẫu nhiên với xác suất $p = \alpha$, trong đó α là một siêu tham số chọn trước. Ý tưởng của EDA là câu s càng dài thì càng bị ảnh hưởng bởi nhiễu, do đó số lần thực hiện các phép tăng cường cơ bản càng được tăng lên.

EDA sử dụng WordNet, một cơ sở dữ liệu ngữ nghĩa của tiếng Anh để tìm từ đồng nghĩa. WordNet tổ chức từ vựng dưới dạng các tập từ đồng nghĩa (synsets), trong đó mỗi synset biểu diễn một khái niệm và liên kết với các synset khác qua những quan hệ ngữ nghĩa như từ đồng nghĩa, trái nghĩa, bao hàm, phân loại, v.v. Điều này cho phép EDA không chỉ đơn giản tra từ đồng nghĩa một cách thủ công, mà còn khai thác được mối quan hệ ngữ nghĩa có hệ thống, đảm bảo rằng từ thay thế có ý nghĩa gần gũi và hợp ngữ cảnh hơn.

Tuy nhiên, do WordNet được xây dựng cho tiếng Anh và mang tính học thuật cao, một số từ vựng đời thường hoặc hiện đại (slang, informal language) có thể không được bao phủ đầy đủ. Ngoài ra, WordNet không xử lý ngữ cảnh cụ thể, nên trong một số trường hợp, từ được thay thế tuy đồng nghĩa về mặt định nghĩa, nhưng lại không phù hợp với ngữ cảnh trong câu. Vì vậy, khi áp dụng EDA trên các tập dữ liệu thực tế, đặc biệt là với các câu mang cảm xúc mạnh hoặc cấu trúc ngôn ngữ linh hoạt, việc kiểm soát chất lượng từ đồng nghĩa là rất quan trọng.

Ngoài ra, việc thay từ đồng nghĩa có thể được thực hiện bằng các mô hình embedding như Word2Vec, GloVe hoặc FastText. Trong các mô hình này, mỗi từ được biểu diễn bằng một vector liên tục trong không gian nhiều chiều, sao cho các từ có ngữ nghĩa tương đồng sẽ nằm gần nhau về mặt khoảng cách cosine. Dựa vào đặc tính này, ta có thể tìm từ đồng nghĩa bằng cách tìm các từ gần nhất trong không gian embedding so với một từ gốc.

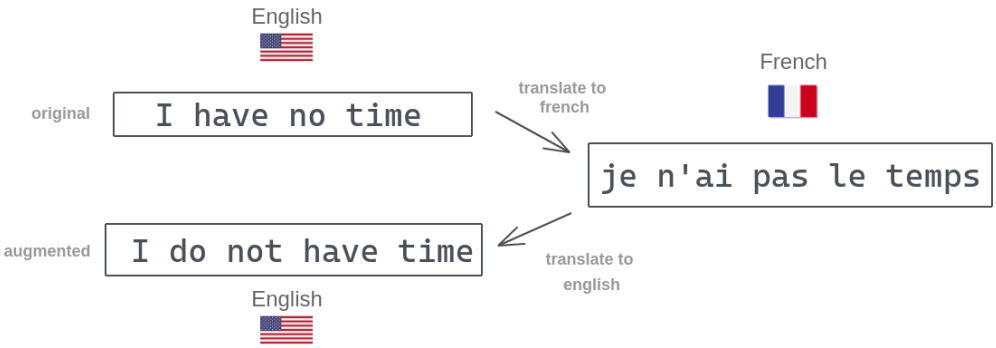
Phương pháp này có ưu điểm là không phụ thuộc vào từ điển ngữ nghĩa như WordNet, nên có thể áp dụng linh hoạt hơn, đặc biệt là với các ngôn ngữ ít tài nguyên hoặc với các từ không có trong WordNet. Ngoài ra, các mô hình như FastText còn cho phép sinh vector cho từ chưa từng thấy (out-of-vocabulary) thông qua đặc trưng ký tự, giúp mở rộng khả năng tìm từ đồng nghĩa cho những trường hợp hiếm gặp.

Tuy nhiên, việc sử dụng embedding để thay từ đồng nghĩa cũng tồn tại rủi ro. Các từ có khoảng cách gần nhau trong không gian vector chưa chắc đã tương đương về mặt ngữ nghĩa trong mọi ngữ cảnh. Ví dụ, hai từ "strong" và "weak" có thể xuất hiện trong những ngữ cảnh tương tự (ví dụ: "a _ argument"), khiến mô hình embedding đặt chúng gần nhau, dù chúng có ý nghĩa trái ngược hoàn toàn. Vì vậy, khi thay thế từ dựa trên embedding, cần có thêm cơ chế kiểm soát ngữ cảnh, ví dụ bằng cách kết hợp với mô hình ngôn ngữ như BERT để chọn từ phù hợp hơn với câu.

Một phương pháp khác tương đối thông dụng là **dịch ngược** (back translation): câu gốc được dịch sang một ngôn ngữ trung gian (thường là tiếng Anh hoặc tiếng Pháp), sau đó được dịch ngược trở lại về ngôn ngữ ban đầu. Kỹ thuật này thường sử dụng các hệ thống dịch máy như Google Translate, MarianMT hoặc các mô hình dịch học sâu dựa trên Transformer.

Phương pháp này có ưu điểm nổi bật là giữ được ngữ nghĩa tổng thể của câu trong khi thay đổi cấu trúc bề mặt, nhờ đó tạo ra những biến thể tự nhiên và hợp ngữ cảnh. Chẳng hạn, câu tiếng Việt "Bộ phim này rất cảm động" có thể được dịch sang tiếng Anh là "This movie is very touching", rồi dịch ngược lại thành "Bộ phim này rất xúc động". Dù có khác biệt về từ vựng, ý nghĩa vẫn được bảo toàn, giúp mô hình học được nhiều cách diễn đạt khác nhau cho cùng một nhãn cảm xúc.

So với các phương pháp như EDA, dịch ngược tạo ra dữ liệu chất lượng cao hơn, ít bị lỗi cú pháp, và tránh được việc thay nhầm từ làm sai lệch ý nghĩa câu. Tuy nhiên, nó cũng đòi hỏi tài nguyên tính toán nhiều hơn, đặc biệt nếu sử dụng mô hình dịch máy nội bộ thay vì API của các dịch vụ có sẵn. Ngoài ra, hiệu quả của phương pháp này còn phụ thuộc vào độ chính xác của hệ thống dịch, và có thể không hoạt động tốt với những câu ngắn, không rõ ngữ cảnh hoặc chứa nhiều từ không phổ biến. Tuy nhiên, do những hạn chế về tài nguyên, nhóm không sử dụng phương pháp này để sinh dữ liệu.



Hình 4: Ví dụ của phương pháp dịch ngược

3 Mô hình đề xuất

3.1 Mô hình học máy

3.1.1 K-nearest neighbors

K-nearest neighbor là một trong những thuật toán học có giám sát (supervised learning) đơn giản nhất (mà hiệu quả trong một vài trường hợp) trong Machine Learning. Khi training, thuật toán này không học một điều gì từ dữ liệu training (lý do thuật toán này được xếp vào loại lazy learning), mọi tính toán được thực hiện khi nó cần dự đoán kết quả của dữ liệu mới. K-nearest neighbor có thể áp dụng được vào cả hai loại của bài toán Supervised learning là Classification và Regression. KNN còn được gọi là một thuật toán Instance-based hay Memory-based learning.

Với KNN, trong bài toán Classification, label của một điểm dữ liệu mới được suy ra trực tiếp từ K điểm dữ liệu gần nhất trong training set. Label của một test data có thể được quyết định bằng major voting (bầu chọn theo số phiếu) giữa các điểm gần nhất, hoặc nó có thể được suy ra bằng cách đánh trọng số khác nhau cho mỗi trong các điểm gần nhất đó rồi suy ra label.

Một cách ngắn gọn, KNN là thuật toán đi tìm đầu ra của một điểm dữ liệu mới bằng cách chỉ dựa trên thông tin của K điểm dữ liệu trong training set gần nó nhất (K-lân cận), không quan tâm đến việc có một vài điểm dữ liệu trong những điểm gần nhất này là nhiễu.

Trong triển khai của KNN, để tăng tốc cho quá trình tìm các điểm láng giềng gần nhất, ta thường sử dụng các cấu trúc dữ liệu phân hoạch không gian (space partitioning data structure) như cây k-d (k-d tree), Ball tree, Vantage-point tree, v.v. Những loại cấu trúc dữ liệu này về mặt bản chất, chia không gian vector thành các miền con dựa trên các điểm dữ liệu huấn luyện, giúp tăng tốc thời gian thực hiện truy vấn, cải thiện hiệu năng của thuật toán đáng kể.

Khoảng cách trong không gian vector

Trong không gian một chiều, khoảng cách giữa hai điểm được xác định đơn giản bằng trị tuyệt đối của hiệu giá trị giữa chúng. Tuy nhiên, trong không gian nhiều chiều, khái niệm khoảng cách được mở rộng và có thể được định nghĩa thông qua nhiều hàm số khác nhau.

Trong bài toán KNN, việc lựa chọn hàm khoảng cách phù hợp đóng vai trò quan trọng để xác định các điểm láng giềng gần nhất. Một số khoảng cách thông dụng thường được sử dụng bao gồm:

- **Khoảng cách Euclid:** Là độ dài đường thẳng nối hai điểm, được tính bằng công thức:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Trong đó, $\mathbf{x}, \mathbf{y}$ là hai điểm trong không gian n chiều. Đây là lựa chọn phổ biến khi các đặc trưng có tính chất đồng nhất và liên tục.

- **Khoảng cách Manhattan:** Là tổng trị tuyệt đối của hiệu các tọa độ, được tính theo công thức:

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} |x_i - y_i|$$

Khoảng cách này hữu ích khi dữ liệu có dạng lưới hoặc các đặc trưng rời rạc không liên tục.

- **Khoảng cách Chebyshev:** Được tính bằng giá trị lớn nhất của hiệu tuyệt đối giữa các tọa độ tương ứng:

$$d(\mathbf{x}, \mathbf{y}) = \max_i |x_i - y_i|$$

Khoảng cách này phù hợp khi muốn đo sự khác biệt lớn nhất giữa các chiều đặc trưng.

- **Khoảng cách Minkowski:** Là trường hợp tổng quát của cả khoảng cách Euclid và Manhattan, được định nghĩa bởi:

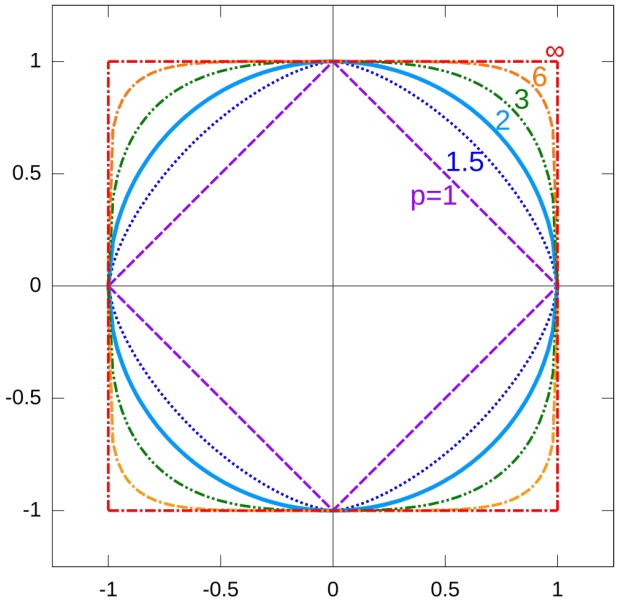
$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

Trong đó p là một tham số điều chỉnh. Khi $p = 1$, ta thu được khoảng cách Manhattan; khi $p = 2$, ta thu được khoảng cách Euclid. Ngoài ra, khi cho p hội tụ về ∞, ta cũng thu được khoảng cách Chebyshev.

Kiến trúc và nguyên lý của mô hình KNN

Đầu vào:

- Tập huấn luyện $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_r, y_r)\}$, trong đó:
  - $\mathbf{x}_i \in \mathbb{R}^n$: là vector đặc trưng đầu vào,
  - y_i: nhãn lớp tương ứng của mẫu thứ i.
- Tham số K: Số lượng láng giềng gần nhất cần xem xét, là một số nguyên dương,
- Hàm khoảng cách: Một hàm số để đo khoảng cách giữa hai điểm trong không gian đặc



Hình 5: Đường tròn đơn vị 2D (tập các điểm trên mặt phẳng cách gốc toạ độ một khoảng cách bằng 1) đối với các giá trị p tương ứng trong khoảng cách Chebyshev.

trưng.

Vòng lặp chính: Với tập điểm dữ liệu mới $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$

- Với mỗi $\mathbf{x} \in \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$:
  1. Tính $d(\mathbf{x}, \mathbf{x}_i)$ cho mọi $\mathbf{x}_i \in D$ (ví dụ: $d = \sqrt{\sum_{j=1}^{n}(x_j - x_{i,j})^2}$).
  2. Chọn K điểm dữ liệu gần $\mathbf{x}$ nhất ($N_k(\mathbf{x})$).
  3. Dự đoán kết quả:
     - Phân loại: Nhãn đa số trong $N_k(\mathbf{x})$.
     - Hồi quy: $\hat{y} = \frac{1}{k}\sum_{\mathbf{x}_i \in N_k(\mathbf{x})} y_i$.

Ứng dụng cho bài toán phân loại cảm xúc

Trong bài toán phân tích cảm xúc nhị phân (binary sentiment analysis), ta cần phân loại một đoạn văn bản thành một trong hai cực tính cảm xúc đối lập, thường là tích cực (positive) hoặc tiêu cực (negative). Để tận dụng các thuật toán học máy, văn bản đầu vào cần được biểu diễn dưới dạng vector đặc trưng số học qua phương pháp *Term Frequency-Inverse Document Frequency (TF-IDF)*. Sau khi quá trình trích xuất đặc trưng hoàn tất, thuật toán KNN được dùng để phân loại. Mỗi văn bản trong tập huấn luyện trở thành một vector $\mathbf{x}_i$ với nhãn $y_i \in \{0, 1\}$. Với một văn bản mới $\mathbf{x}$, KNN tìm K văn bản gần nhất trong tập huấn luyện và dự đoán nhãn dựa trên nhãn chiếm đa số của K láng giềng này.

Nhìn chung, KNN là một thuật toán theo hướng *lazy learning*, do không thực hiện huấn luyện rõ ràng mà chỉ đưa ra dự đoán tại thời điểm kiểm tra. Điều này khiến KNN không thực sự hiệu quả đối với bài toán phân loại cảm xúc, đặc biệt khi kích thước dữ liệu lớn. Trong phần tiếp theo, nhóm sẽ trình bày một số phương pháp có hiệu suất cao hơn và phù hợp hơn cho bài toán này.

3.1.2 Support Vector Machine

Support Vector Machine (SVM) là một thuật toán supervised learning được ứng dụng rộng rãi trong cả bài toán phân loại (classification) và hồi quy (regression). Nguyên lý của SVM là xác định một siêu phẳng (hyperplane) tối ưu trong không gian đặc trưng (feature space) để phân tách các mẫu dữ liệu thuộc các lớp khác nhau. Tính tối ưu của siêu phẳng này được xác định bằng khoảng cách lề (margin), là khoảng cách vuông góc nhỏ nhất từ siêu phẳng đến các điểm dữ liệu gần nhất của mỗi lớp, được gọi là các vector hỗ trợ (support vectors). Mục tiêu của SVM là cực đại hóa khoảng cách lề này, qua đó tăng cường khả năng tổng quát hóa của mô hình trên dữ liệu chưa từng thấy.

Giả sử tập huấn luyện của ta gồm có các mẫu $(\mathbf{x_i}, y_i)$, $i = 1, 2, \dots r$, trong đó $\mathbf{x_i} \in \mathbb{R}^n$ là đặc trưng đầu vào, còn $y_i \in \{-1, 1\}$ là nhãn dữ liệu. Việc sử dụng nhãn -1 thay vì 0 chỉ giúp các công thức toán đơn giản hơn, không làm thay đổi bản chất bài toán.

Khoảng cách có dấu giữa một điểm $\mathbf{x}$ và một mặt phẳng $P : \mathbf{w}^T\mathbf{x} + b = 0$ có thể được tính bằng

$$d(\mathbf{x}, P) = \frac{\mathbf{w}^T\mathbf{x} + b}{\|\mathbf{w}\|_2},$$

trong đó dấu của khoảng cách xác định điểm huấn luyện nằm ở phía nào của nửa siêu phẳng. Ở đây, dấu âm hay dấu dương không có ý nghĩa gì đặc biệt (chẳng hạn như "dấu âm thì nằm trái siêu phẳng, dấu dương thì nằm ở bên phải") mà chỉ là một thước đo tương đối giữa các điểm dữ liệu với nhau. Việc đảo dấu $\mathbf{w}$ và b sẽ thu được siêu phẳng tương đương nhưng có ý nghĩa dấu bị đảo ngược.

Do khả năng đảo dấu trên, ta có thể coi nửa siêu phẳng dấu dương là không gian các vector đặc trưng có nhãn $y = 1$, còn siêu phẳng dấu âm là không gian các vector đặc trưng có nhãn $y = -1$. Khi đó, ta mong muốn tìm được một siêu phẳng có tính chất:

$$y_i \left(\mathbf{w}^T\mathbf{x}_i + b \right) > 0, \tag{1}$$

cho mọi điểm dữ liệu $(\mathbf{x_i}, y_i)$, $i = 1, 2, \dots r$.

Để thực hiện dự đoán nhãn cho điểm dữ liệu chưa biết $\mathbf{x}$, ta chỉ việc xét dấu của biểu thức $\mathbf{w}^T\mathbf{x}_i + b$: nếu dương thì nhãn là 1, nếu âm thì ngược lại. Giá trị biểu thức cũng có thể được sử dụng làm mức tự tin (confidence level) cho phép đoán.

Do hệ điều kiện 1 có thể có vô số nghiệm (tương ứng với vô số siêu phẳng phân biệt), ta cần một cơ chế để chọn ra siêu phẳng tốt nhất. Ý tưởng đơn giản nhất là sử dụng một độ đo "khoảng cách" giữa hai điểm dữ liệu huấn luyện khác nhãn gần nhất, coi đó làm khoảng cách giữa hai miền với hai nhãn tương ứng. Trong đó, "khoảng cách" được đo bằng cách nào đó mà phụ thuộc vào siêu phẳng tách: các metric Euclid, Manhattan không thể được sử dụng, mà thước đo khoảng cách phải dựa trực tiếp lên biểu thức quyết định $\mathbf{w}^T\mathbf{x} + b$. Ý tưởng đơn giản nhất là khoảng cách giữa hai điểm, chiếu xuống phương pháp tuyến của siêu phẳng

$$d_P(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{\|\mathbf{w}\|}\left|(\mathbf{w}^T\mathbf{x}_i + b) - (\mathbf{w}^T\mathbf{x}_j + b)\right| = \frac{1}{\|\mathbf{w}\|}\left|(\mathbf{w}^T(\mathbf{x}_i - \mathbf{x}_j)\right|,$$

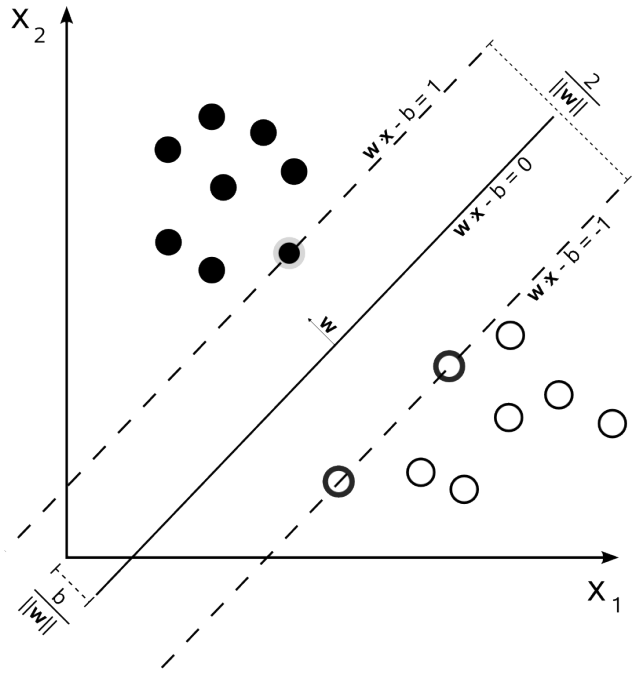
trong đó hai điểm dữ liệu $\mathbf{x}_i$ và $\mathbf{x}_j$ là điểm gần siêu phẳng P nhất đối với cả hai nhãn. Khoảng cách giữa chúng, chiếu theo phương pháp tuyến của P, là tổng khoảng cách từ mỗi điểm đến P, và do hai điểm nằm trên hai nửa siêu phẳng khác nhau, cũng là trị tuyệt đối của hiệu khoảng cách có dấu của chúng đến P.

Giá trị "khoảng cách" này được gọi là **bản lề** (margin)[3] của P. Chú ý rằng công thức này không phụ thuộc vào b, mà mặt khác, hai siêu phẳng $\mathbf{w}^T\mathbf{x} + b_1 = 0$ và $\mathbf{w}^T\mathbf{x} + b_2 = 0$, nếu cùng thoả mãn 1, thì siêu phẳng $\mathbf{w}^T\mathbf{x} + b' = 0$ cũng thoả mãn ràng buộc này, với mọi b' nằm giữa b_1 và b_2. Hơn nữa, tất cả các siêu phẳng này cùng cho một giá trị bản lề như nhau, nghĩa là phát biểu bải toán hiện tại không thể phân biệt và đánh giả được giữa các siêu phẳng này siêu phẳng nào là tốt nhất. Để siêu phẳng quyết định nằm xa các điểm quyết định nhất có thể[4] (vốn là định nghĩa gốc của khải niệm bản lề trong nhiều tài liệu), ta lấy b là giá trị sao cho siêu phẳng nằm chính giữa hai điểm dữ liệu quyết định $\mathbf{x}_i$ và $\mathbf{x}_j$ trên biểu thức bản lề.

Cuối cùng, để phát biểu bài toàn tương đương với các phát biểu thông dụng, ta cần đưa giả thuyết các điểm dữ liệu quyết định $\mathbf{x}_i$ và $\mathbf{x}_j$ là các điểm nằm gần siêu phẳng nhất

[3]Một khải niệm khác của bản lề là khoảng cách từ siêu phẳng quyết định đến điểm dữ liệu gần nhất (chiếu theo phương pháp tuyến siêu phẳng). Tuy hai khái niệm này gần tương đương (theo nghĩa tối ưu một cái đồng nghĩa với tối ưu cái còn lại), nhưng ý tưởng của khái niệm này hơi thiếu tự nhiên hơn so với khái niệm đề cập trên.

[4]Có thể trong một số tình huống ta sẽ không muốn thực hiện như thế này: chẳng hạn như dữ liệu lấy từ một univariate Gaussian mixture model gồm hai thành phần ứng với hai nhãn, nhưng một nhãn được lấy mẫu nhiều hơn nhãn còn lại. Khi đó, các điểm quyết định x_i và x_j có thể coi là các biến ngẫu nhiên nhưng có phân bố khác nhau (do một điểm là min hoặc max của nhiều điểm mẫu hơn điểm còn lại), và chọn siêu phẳng (suy biến thành còn một điểm) là điểm nằm chính giữa hai điểm quyết định này có thể không hiệu quả.



Hình 6: Siêu phẳng tách và hai siêu phẳng biên tương ứng. Trong phát biểu bài toán trên, ta có thể thấy rằng mọi siêu phẳng nằm giữa hai siêu phẳng biên đều thoả mãn ràng buộc và cho ra cùng một giá trị bản lề. Tuy nhiên, để tối thiểu khoảng cách từ siêu phẳng quyết định đến các điểm dữ liệu, ta chọn siêu phẳng nằm chính giữa hai siêu phẳng cực biên.

mà có hai nhãn khác nhau thành một ràng buộc. Do ta chọn b sao cho siêu phẳng cách đều hai điểm này, chẳng hạn bởi một khoảng d, ta có thể viết lại 1 thành:

$$y_i \left(\mathbf{w}^T \mathbf{x}_i + b \right) \geqslant d \|\mathbf{w}\|_2,$$

cho mọi điểm dữ liệu $(\mathbf{x_i}, y_i)$, $i = 1, 2, \ldots r$.

Cuối cùng, chuẩn hoá $d\|\mathbf{w}\|_2 = 1$ [5], ta có phát biểu cuối cùng thường thấy trong các tài liệu đề cập đến SVM:

$$\text{maximize } 2d = \frac{2}{\|\mathbf{w}\|_2}, \text{subject to } y_i \left(\mathbf{w}^T \mathbf{x}_i + b \right) \geqslant 1, \forall i = 1, 2, \ldots, r. \tag{2}$$

Tuy nhiên, dạng biểu diễn trên của bài toán có thể vô nghiệm. Điều này có thể xảy ra khi chẳng hạn dữ liệu huấn luyện có lỗi. Để giải quyết vấn đề này, ta thêm lỗi ξ_i cho từng điểm dữ liệu và nới lỏng ràng buộc thành $y_i \left(\mathbf{w}^T \mathbf{x}_i + b \right) \geqslant 1 - \xi_i$. Sau đó, ta tối thiểu hoá tổng lỗi cùng với tối đa hoá bản lề thông qua tối thiểu hoá hàm mục tiêu $\frac{1}{2}\|\mathbf{w}\|_2^2 + C \sum_{i=1}^{r} \xi_i^k$, trong đó C là hệ số phạt, cân bằng giữa hai mục tiêu của bài toán, còn k thường được lấy

[5] Nếu $d\|\mathbf{w}\|_2 \neq 1$ thì chia cả $\mathbf{w}$ và b cho một lượng tương ứng để thu được siêu phẳng tương đương có $d\|\mathbf{w}\|_2 = 1$

bằng 1 để làm đơn giản hoá tính toán. Mô hình này được gọi là mô hình SVM chuẩn. Ta thu được bài toán **Soft-margin SVM** tổng quát:

$$\text{maximize } \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{r}\xi_i^k, \text{subject to } y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \geqslant 1 - \xi_i, \forall i = 1, 2, \ldots, r. \tag{3}$$

Dưới giả thuyết rằng dữ liệu bị ảnh hưởng bởi nhiễu Gauss, thì ta có thể giả sử ξ_i là các giá trị lấy mẫu độc lập từ phân bố Gauss, và tối ưu log-likelihood sẽ tương đương với việc đặt $k = 2$ trong hàm mục tiêu nói trên. Mô hình này được gọi là mô hình Least-squares SVM (LS-SVM), tuy nhiên giống như cách ordinary least squares (OLS) kém hiệu quả với điểm ngoại lai hơn least absolute deviations (LAD), LS-SVM cũng kém hiệu quả hơn SVM chuẩn khi có nhiều điểm ngoại lai.

Mặc dù 3 là một bài toán Quadratic Programming, nghĩa là có thể được giải bằng các thư viện như CVXOPT, cách làm này rất kém hiệu quả khi số chiều lớn và không được sử dụng nhiều trong thực tế. Tuy nhiên, quan sát này không thực sự vô nghĩa: bài toán này lồi nên chắc chắn sẽ có nghiệm tối ưu toàn cục. Đề giải bài toán này, người ta thường giải bài toán đối ngẫu của nó. Thư viện LIBSVM, một thư viện thông dụng trong giải quyết các bài toán SVM, sử dụng phương pháp Sequential minimal optimization để giải quyết bài toán đối ngẫu, thực hiện tối ưu trên từng cặp nhân tử Lagrange vi phạm điều kiện Karush–Kuhn–Tucker (KKT) đến khi thuật toán hội tụ, trong đó, mỗi nhân tử Lagrange tương ứng với một điểm nằm trên một trong hai siêu phẳng biên.

Ban đầu, SVM được thiết kế để giải quyết các bài toán phân loại tuyến tính (linear classification). Mục tiêu của SVM là tìm kiếm một siêu phẳng tuyến tính (linear classifier) có khả năng phân chia rõ ràng hai tập hợp dữ liệu thuộc các lớp khác nhau.

Đối với các bài toán phân loại phi tuyến – tức là không thể tìm được siêu phẳng nào phân tách rõ ràng hai lớp dữ liệu, SVM sử hàm nhân (kernel function). Các hàm nhân đóng vai trò như một phép ánh xạ phi tuyến (non-linear mapping) dữ liệu từ không gian đầu vào ban đầu sang một không gian đặc trưng có số chiều cao hơn. Trong không gian mới này, dữ liệu có thể trở nên phân tách tuyến tính, cho phép tìm ra một siêu phẳng tuyến tính để thực hiện việc phân loại. Các hàm nhân phổ biến bao gồm hàm nhân tuyến tính(linear kernel), hàm nhân đa thức (polynomial kernel), hàm nhân Gaussian (Radial Basis Function - RBF kernel) và hàm nhân sigmoid.

Khi không sử dụng hàm nhân, ta gọi đó là SVM tuyến tính (linear SVM) – thực chất đây là trường hợp SVM sử dụng hàm nhân tuyến tính.

Ứng dụng SVM trong bài toán phân loại cảm xúc

Trong bài toán phân tích cảm xúc nhị phân (binary sentiment analysis), nhiệm vụ là phân loại một đoạn văn bản thành một trong hai cực tính cảm xúc đối lập, thường là tích cực (positive) hoặc tiêu cực (negative). Để tận dụng các thuật toán học máy, văn bản đầu vào cần được biểu diễn dưới dạng vector đặc trưng số học. Một phương pháp phổ biến và hiệu quả cho mục đích này là sử dụng Term Frequency-Inverse Document Frequency (TF-IDF). Sau khi quá trình trích xuất đặc trưng bằng TF-IDF hoàn tất, mô hình Linear Support Vector Machine (Linear SVM) được sử dụng để tìm kiếm một siêu phẳng tối ưu trong không gian đặc trưng đã được tạo ra.

Nhờ khả năng tìm kiếm siêu phẳng phân tách tối ưu và tập trung vào các vector hỗ trợ (support vectors) nằm gần ranh giới quyết định, Linear SVM thường đạt được hiệu suất phân loại chính xác và mạnh mẽ trong nhiều bài toán phân loại văn bản, bao gồm cả phân tích cảm xúc.

3.1.3 Logistic Regression

Trong lĩnh vực học máy và thống kê, Logistic Regression là một phương pháp phổ biến và hiệu quả để giải quyết các bài toán phân loại. Không giống như hồi quy tuyến tính dùng để dự đoán giá trị liên tục, Logistic Regression được thiết kế để ước lượng xác suất của một biến phân loại nhị phân – chẳng hạn như "có" hoặc "không", "thành công" hoặc "thất bại". Phương pháp này dựa trên hàm logistic (hay còn gọi là hàm sigmoid) để ánh xạ đầu ra của mô hình thành một giá trị nằm trong khoảng từ 0 đến 1, biểu diễn một điểm dữ liệu thuộc về một lớp cụ thể. Mặc dù trong tên có chứa từ regression, logistic regression thường được sử dụng nhiều hơn cho các bài toán classification.

Đầu ra dự đoán của mô hình Logistic Regression thường được biểu diễn dưới dạng:

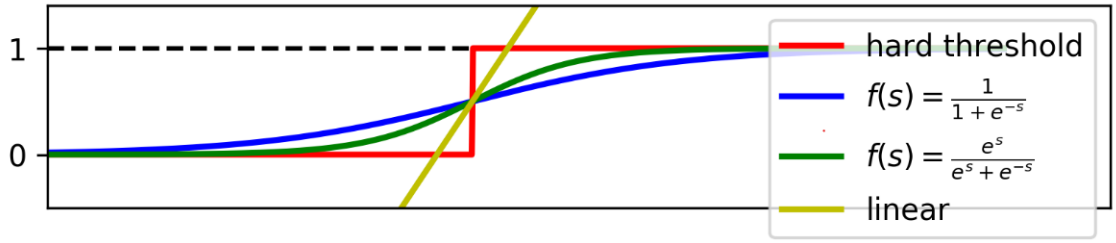
$$f(\mathbf{x}) = \theta(\mathbf{w}^T\mathbf{x} + b).$$

Trong đó, θ là một hàm logistic. Một số activation cho mô hình tuyến tính được cho trong hình 7.

Về mặt bản chất, hồi quy logistic đưa feature vector vào một mô hình hồi quy tuyến tính, sau đó đưa giá trị hồi quy tuyến tính này vào hàm logistic để chuyển thành giá trị xác suất.

Hàm kích hoạt $\theta(z)$ cần được chọn để thỏa mãn các tính chất sau:

- **Liên tục và bị chặn:** Hàm là một hàm số liên tục nhận giá trị thực và bị chặn trong khoảng $[0, 1]$. Điều này đảm bảo đầu ra có thể được biểu diễn dưới dạng xác suất.



Hình 7: Minh họa các activation function khác nhau

- **Tính chất phân tách xác suất:** Nếu coi điểm có tung độ $\theta(z) = 0.5$ là ngưỡng phân chia, thì:
  - Với các điểm $z \ll 0$, giá trị $\theta(z) \to 0$;
  - Với các điểm $z \gg 0$, giá trị $\theta(z) \to 1$.

  Ý tưởng của điều kiện này là như sau: z được coi là đại lượng quyết định điểm dữ liệu sẽ nằm nghiêng về class nào. Khi z quá nhỏ (hoặc quá lớn), điểm dữ liệu sẽ gần như xác định nằm trong class tương ứng.
- **Tính mượt:** Hàm phải trơn (smooth) và có đạo hàm ở mọi điểm. Tính chất này giúp quá trình tối ưu hóa (như sử dụng gradient descent) trở nên hiệu quả hơn.

Hàm sigmoid

Trong số các hàm số thỏa mãn ba tính chất đã nêu ở trên, hàm **sigmoid** là hàm được sử dụng phổ biến nhất, bởi vì nó bị chặn trong khoảng $(0, 1)$, rất phù hợp để biểu diễn xác suất trong các mô hình phân loại nhị phân. Hàm sigmoid được định nghĩa như sau:

$$f(s) = \frac{1}{1+e^{-s}} \triangleq \sigma(s)$$

Hàm sigmoid có một số tính chất giới hạn quan trọng, giúp nó thoả mãn tính chất phân tách xác suất nói trên:

$$\lim_{s\to-\infty} \sigma(s) = 0, \quad \lim_{s\to+\infty} \sigma(s) = 1.$$

Ngoài ra, hàm sigmoid có công thức đạo hàm rất đơn giản và hiệu quả cho việc tính toán trong huấn luyện mô hình:

$$\sigma'(s) = \frac{e^{-s}}{(1+e^{-s})^2} = \frac{1}{1+e^{-s}} \cdot \frac{e^{-s}}{1+e^{-s}} = \sigma(s)(1-\sigma(s)).$$

Chính nhờ công thức đạo hàm đơn giản này mà hàm sigmoid trở thành một lựa chọn rất phổ biến trong kiến trúc của các mô hình học máy.

Với mỗi điểm dữ liệu $\mathbf{x}$, ta tính giá trị hàm logistic $\hat{y} = \sigma(\mathbf{w}^T\mathbf{x} + b)$ và coi đây là xác suất điểm này thuộc lớp $y = 0$. Như vậy, lớp phân loại của điểm dữ liệu là một biến ngẫu nhiên Bernoulli Y có $p = \hat{y}$. Nhãn của điểm này, y, được coi là một giá trị lấy mẫu từ Y, có thể được dùng để đo **độ bất ngờ**[6] (suprisal, information content) khi thu được y bằng cách lấy mẫu từ Y:

$$I_Y(\hat{y}) = \begin{cases} -\log p, & \text{if } y = 1, \\ -\log(1-p), & \text{if } y = 0 \end{cases} = -[y \log \hat{y} + (1-y)\log(1-\hat{y})].$$

Do đại lượng này có tính cộng (đối với các điểm dữ liệu độc lập), ta có thể thiết lập công thức hàm mất mát bằng cách tính tổng độ bất ngờ của cả bộ dữ liệu.

Để tối ưu dựa trên hàm mất mát này, ta có thể sử dụng các phương pháp tối ưu sử dụng đạo hàm:

$$\begin{aligned} dI_Y(\hat{y}) = -\left(\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}\right) d\hat{y} = \frac{\hat{y}-y}{\hat{y}(1-\hat{y})}\sigma'(\mathbf{w}^T\mathbf{x}+b)(\mathbf{x}^T d\mathbf{w} + db) \\ \Rightarrow \frac{\partial I_Y(\hat{y})}{\partial \mathbf{w}} = (\hat{y}-y)\mathbf{x}, \\ \frac{\partial I_Y(\hat{y})}{\partial b} = \hat{y} - y, \end{aligned}$$

trong đó ta sử dụng $\sigma'(\mathbf{w}^T\mathbf{x}+b) = \sigma(\mathbf{w}^T\mathbf{x}+b)(1-\sigma(\mathbf{w}^T\mathbf{x}+b)) = \hat{y}(1-\hat{y})$. Hàm sigmoid ở đây giúp cho công thức đạo hàm của ta gọn gàng hơn đáng kể. Với các công thức đạo hàm này, ta có thể sử dụng các thuật toán học theo đạo hàm (SGD, RMS, ...) để tối ưu mô hình.

3.1.4 Naive Bayes classifier

Naive Bayes là một phương pháp phân loại xác suất đơn giản, phổ biến trong các bài toán xử lý ngôn ngữ tự nhiên như phân loại văn bản, lọc thư rác, và phân tích cảm xúc. Ý tưởng chính của Naive Bayes dựa trên định lý Bayes, kết hợp với giả định ngây thơ (naive) rằng các đặc trưng (features) là độc lập có điều kiện (conditionally independent) với nhau khi biết nhãn lớp.

Khi áp dụng phương pháp phân lớp Naive Bayes, ta coi mỗi token là một đặc trưng của một câu. Ở đây, ta sử dụng mô hình bag-of-words: ta bỏ qua thông tin về thứ tự các

[6] Hoặc hiểu theo cách khác, đây cũng là tối ưu negative log-likelihood của Y.

token trong một câu, mà chỉ quan tâm đến câu gồm có những token nào, và một token xuất hiện bao nhiêu lần. Mô hình này rõ ràng là bộc lộ nhiều hạn chế vì tính mất mát thông tin, nhưng phương pháp này vẫn thực hiện tương đối tốt trong các ứng dụng thực tế.

Mặc dù đơn giản, Naive Bayes vẫn được sử dụng rộng rãi trong các hệ thống thực tế nhờ tính hiệu quả, tốc độ huấn luyện nhanh và khả năng hoạt động tốt trên các tập dữ liệu vừa và nhỏ. Naive Bayes được ứng dụng rộng rãi trong các lĩnh vực như lọc thư rác, phân loại tài liệu, phát hiện tin giả. Trong bài toán phân tích cảm xúc của nhóm, Naive Bayes thường được dùng như một baseline cho các phương pháp còn lại, giúp đánh giá hiệu quả tương đối của các mô hình phức tạp hơn.

Ý tưởng chính của Naive Bayes classifier dựa trên định lý Bayes trong lý thuyết xác suất, cho phép ta "đảo" các xác suất có điều kiện từ $P(A|B)$ sang $P(B|A)$ như sau:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}.$$

Giả sử ta có một câu s gồm các token $t_{\sigma(1)}, t_{\sigma(2)}, ..., t_{\sigma(m)}$ nằm trong tập ngữ vựng $V = \{t_1, t_2, ..., t_{|V|}\}$. Khi đó, ta có thể coi mỗi token $t_{\sigma(i)}$ được lấy mẫu từ một phân bố chỉ phụ thuộc vào phân loại cảm xúc của s. Nói một cách khác, ta giả dụ rằng hai loại câu ứng với cảm xúc tiêu cực và tích cực gồm có các phân bố token khác nhau: chẳng hạn như những câu có cảm xúc tích cực sẽ gồm nhiều từ như "like", "enjoy", "great", v.v., còn những câu có cảm xúc tiêu cực lại chứa những từ mang sắc thái cảm xúc tiêu cực, châm biếm như "hate", "sucks", "bad". Ta gọi hai phân bố này là T_+ và T_- lần lượt ứng với cảm xúc tích cực và tiêu cực,

Hai phân bố T_+ và T_- có thể được ước lượng từ tập dữ liệu huấn luyện. Ta tính số lần xuất hiện của mỗi token $t_i \in V$ trong các câu có cảm xúc tích cực và tiêu cực, đặt là c_i^+ và c_i^-, thì T_+ và T_- có thể được ước tính có các hàm phân bố xác suất như sau:

$$p_+(t_{\sigma(i)}) = P(T_+ = t_{\sigma(i)}) = \frac{c^+_{\sigma(i)}}{\sum_{j=1}^{|V|} c_j^+},$$
$$p_-(t_{\sigma(i)}) = P(T_- = t_{\sigma(i)}) = \frac{c^-_{\sigma(i)}}{\sum_{j=1}^{|V|} c_j^-}.$$

Ngoài ra, để tránh trường hợp xác suất bằng 0, ta thường thêm một lượng nhỏ $\alpha > 0$

vào các giá trị tần suất c_i^+:

$$p_+(t_{\sigma(i)}) = P(T_+ = t_{\sigma(i)}) = \frac{c^+_{\sigma(i)} + \alpha}{\sum_{j=1}^{|V|} c_j^+ + |V|\alpha},$$
$$p_-(t_{\sigma(i)}) = P(T_- = t_{\sigma(i)}) = \frac{c^-_{\sigma(i)} + \alpha}{\sum_{j=1}^{|V|} c_j^- + |V|\alpha}.$$

Lý do ta cần tránh các xác suất bằng 0 là do bộ dữ liệu huấn luyện có thể chưa đủ khả năng biểu diễn toàn bộ không gian phân bố của các biến T_+ và T_-, xác suất gặp phải một token t_i có thể nhỏ nhưng không bằng 0. α được gọi là tần suất giả (pseudocount), thường lấy giá trị là 1 (Laplace smoothing). Ngoài ra, một lý do khác ta tránh xác suất bằng 0 vì khi nhân xác suất: 0 nhân bất cứ xác suất nào cũng bằng 0, làm giá trị của các xác suất còn lại vô nghĩa.

Như vậy, giả sử như s là câu có cảm xúc tích cực, thì các token $t_{\sigma(1)}, t_{\sigma(2)}, ..., t_{\sigma(m)}$ trong câu này có thể coi là các token được lấy mẫu từ biến ngẫu nhiên T_+. Do đó, xác suất để tạo ra câu s bằng cách lấy mẫu độc lập các token từ biến ngẫu nhiên T_+, được gọi là *likelihood* của sự kiện s có mang tính tích cực, là

$$P(s = t_{\sigma(1)}t_{\sigma(2)}...t_{\sigma(m)}|s \text{ is positive}) = \prod_{i=1}^{m} p_+(t_{\sigma(i)}) = \prod_{i=1}^{m} \frac{c^+_{\sigma(i)} + \alpha}{\sum_{j=1}^{|V|} c_j^+ + |V|\alpha}.$$

Áp dụng định lý Bayes, ta tính được:

$$P(s \text{ is positive}|s = t_{\sigma(1)}t_{\sigma(2)}...t_{\sigma(m)}) = \frac{P(s = t_{\sigma(1)}t_{\sigma(2)}...t_{\sigma(m)}|s \text{ is positive})P(s \text{ is positive})}{P(s = t_{\sigma(1)}t_{\sigma(2)}...t_{\sigma(m)})}.$$

Một cách tương tự, ta cũng có,

$$P(s \text{ is negative}|s = t_{\sigma(1)}t_{\sigma(2)}...t_{\sigma(m)}) = \frac{P(s = t_{\sigma(1)}t_{\sigma(2)}...t_{\sigma(m)}|s \text{ is negative})P(s \text{ is negative})}{P(s = t_{\sigma(1)}t_{\sigma(2)}...t_{\sigma(m)})}.$$

Do xác suất ở mẫu số là bằng nhau, ta bỏ qua việc tính toán biểu thức này. Naive Bayes classifier sẽ chỉ tính hai biểu thức ở tử số và so sánh với nhau. Xác suất nào lớn hơn thì ta phân loại s vào lớp đó. Ở đây, hai xác suất $P(s \text{ is positive})$ và $P(s \text{ is negative})$, gọi là *prior*, có thể được ước lượng bằng cách đếm số lượng câu ứng với mỗi cảm xúc trong bộ dữ

liệu huấn luyện. Giả sử trong bộ dữ liệu huấn luyện có p câu tích cực và n câu tiêu cực, thì:

$$P(s \text{ is positive}) = \frac{p}{p+n}, P(s \text{ is negative}) = \frac{n}{p+n}.$$

Tổng kết lại, với câu $s = t_{\sigma(1)}t_{\sigma(2)}...t_{\sigma(m)}$, ta tính hai giá trị điểm ứng với hai lớp:

$$\text{score}_+(s) = \frac{p}{p+n}\prod_{i=1}^{m}\frac{c^+_{\sigma(i)} + \alpha}{\sum_{t_j \in S} c^+_j + |S|\alpha},$$
$$\text{score}_-(s) = \frac{n}{p+n}\prod_{i=1}^{m}\frac{c^-_{\sigma(i)} + \alpha}{\sum_{t_j \in S} c^-_j + |S|\alpha}.$$

Nếu $\text{score}_+(s) > \text{score}_-(s)$ thì phân loại s là có cảm xúc tích cực, nếu $\text{score}_+(s) < \text{score}_-(s)$ thì phân loại s là có cảm xúc tiêu cực.

Phương pháp trên được gọi là Multinomial Naive Bayes, vì nếu ta đặt x_i là số lần xuất hiện của token t_i trong s, thì thực chất ta tính các xác suất có điều kiện dựa trên giả định rằng vector tần suất $\mathbf{x} = (x_1, x_2, ..., x_{|V|})^T$ tuân theo phân phối đa thức (multinomial distribution) với các tham số $k = |V|$ và $n = m$. Công thức tính score trên có thể được viết lại thành:

$$\text{score}_+(s) = \frac{p}{p+n}\prod_{i=1}^{|V|}\left(\frac{c^+_i + \alpha}{\sum_{t_j \in S} c^+_j + |S|\alpha}\right)^{x_i},$$
$$\text{score}_-(s) = \frac{n}{p+n}\prod_{i=1}^{|V|}\left(\frac{c^-_i + \alpha}{\sum_{t_j \in S} c^-_j + |S|\alpha}\right)^{x_i}.$$

Nếu ta không quan tâm đến số lần xuất hiện của các token mà chỉ quan tâm đến việc token này có xuất hiện trong câu s hay không, thì ta sử dụng Bernoulli Naive Bayes classifier, với công thức tính score được thay đổi thành

$$\text{score}_+(s) = \frac{p}{p+n}\prod_{i=1}^{|V|}\frac{(d^+_i + \alpha)^{x_i}(p - d^+_i + \alpha)^{1-x_i}}{p + 2\alpha},$$
$$\text{score}_-(s) = \frac{n}{p+n}\prod_{i=1}^{|V|}\frac{(d^-_i + \alpha)^{x_i}(n - d^-_i + \alpha)^{1-x_i}}{n + 2\alpha},$$

trong đó d^+_i và d^-_i lần lượt là số câu có cảm xúc tích cực tiêu cực chứa token t_i, x_i là biến nhị phân: $x_i = 1$ nếu token t_i nằm trong câu s và $x_i = 0$ nếu ngược lại. Điểm khác biệt đáng kể ở đây là ta không chỉ xét các token $t_{\sigma(i)}$ có trong câu s, mà còn xét đến các token không

xuất hiện trong câu.

Ở đây, thay vì mô hình mỗi câu là một chuỗi quá trình lấy mẫu từ T_+ hoặc T_-, thì ta coi mỗi câu là một tập các token với kích thước không cố định. Việc token này có được thêm vào câu được quyết định độc lập đối với mỗi token, mỗi token được mô hình bởi một biến ngẫu nhiên Bernoulli.

Các công thức trên được biểu diễn dưới dạng tích các xác suất, nhưng trong máy tính, do hạn chế của việc lưu trữ số thực dấu phẩy động, người ta thường tính logarit của các giá trị trên nhằm tăng tính ổn định tính toán (numerical stability). Quá trình này làm thay đổi các phép nhân thành cộng, mũ thành nhân, có thể được dễ dàng thu được từ các công thức đã chứng minh trên.

Về mặt xác suất thì Naive Bayes là một mô hình xác suất, nhưng ta có thể cải tiến phương pháp này bằng cách sử dụng các chỉ số TF-IDF đã tính toán ở phần Tiền xử lý (sau khi được chuẩn hoá) thay cho các giá trị tần suất x_i trong việc tính score. Cách này làm mất đi ý nghĩa xác suất, song thực nghiệm đã chứng minh là cải thiện đáng kể hiệu suất của mô hình.

3.1.5 Decision Tree

Decision tree hay cây quyết định là một mô hình supervised learning, có thể được áp dụng vào cả hai bài toán classification và regression. Khác biệt với những phương pháp có tham số (parametric methods) chọn trước dạng hàm ánh xạ từ dữ liệu đến nhãn, Decision tree sử dụng cấu trúc cây để biểu diễn các hàm.

Mô hình sẽ nhận đầu vào là tập dữ liệu với các đặc trưng, và hoạt động bằng cách chia nhỏ tập dữ liệu thành các nhánh dựa trên các thuộc tính đặc trưng, cho đến khi đạt được các nhãn đầu ra cụ thể. Việc chia tách được quyết định dựa trên các tiêu chí như Gini impurity, Entropy (trong ID3, C4.5), hay mức giảm sai số (trong CART).

Trong bài toán phân tích cảm xúc (sentiment analysis), cụ thể là phân loại đánh giá phim thành hai nhãn tích cực (1) và tiêu cực (0), cây quyết định có thể được áp dụng như sau:

- Đầu tiên, văn bản được chuyển đổi thành dạng số bằng các kỹ thuật như Bag of Words hoặc TF-IDF.
- Sau đó, các đặc trưng này được sử dụng để huấn luyện mô hình cây quyết định.

Đối với bài toán sentiment analysis, đầu vào của Cây quyết định và các vector với thuộc tính liên tục, chính vì vậy ta cần phải sử dụng một biến thể của Decision tree là Classification

and Regression Tree (CART), bên cạnh đó nhóm sử dụng độ đo Gini Impurity để xác định thuộc tính phân loại tại mỗi bước.

1. Định nghĩa Gini impurity **Gini impurity** là một chỉ số đo mức độ không thuần khiết (impurity) của một tập dữ liệu. Nó phản ánh xác suất một mẫu bị phân loại sai nếu gán nhãn ngẫu nhiên theo phân phối lớp hiện tại.

Giả sử một nút chứa các mẫu thuộc k lớp, với p_i là xác suất một mẫu thuộc lớp i, thì:

$$\text{Gini}(t) = 1 - \sum_{i=1}^{k} p_i^2$$

Trong bài toán phân loại nhị phân (ví dụ: sentiment analysis với nhãn 0 và 1), công thức đơn giản thành:

$$\text{Gini}(t) = 2p(1-p)$$

2. Cách thuật toán CART sử dụng Gini impurity Thuật toán CART (Classification and Regression Tree) sử dụng Gini impurity như sau:

1. Với mỗi nút, xét tất cả các thuộc tính và các ngưỡng chia (nếu thuộc tính là liên tục).
2. Chia tập dữ liệu D thành hai nhánh con: D_{left} và D_{right}.
3. Tính Gini impurity có trọng số của phép chia:

$$\text{Gini}_{\text{split}} = \frac{|D_{\text{left}}|}{|D|} \cdot \text{Gini}(D_{\text{left}}) + \frac{|D_{\text{right}}|}{|D|} \cdot \text{Gini}(D_{\text{right}})$$

4. Chọn phép chia có $\text{Gini}_{\text{split}}$ nhỏ nhất để thực hiện.
5. Lặp lại quá trình cho đến khi đạt điều kiện dừng (ví dụ: đạt max depth, impurity bằng 0, số mẫu quá ít).

3. Nhận xét

- Gini impurity càng thấp thì dữ liệu tại nút càng "thuần khiết", tức là chứa chủ yếu một loại nhãn.
- CART luôn cố gắng chia sao cho Gini impurity của nhánh con giảm mạnh nhất.
- Đây là cách hiệu quả để phân loại trong các bài toán như sentiment analysis, khi dữ

liệu đầu vào là các giá trị liên tục (như TF-IDF).

3.1.6 Random Forest

Việc sử dụng cây quyết định có thể dẫn đến các rủi ro về Overfitting: Khi sử dụng một độ sâu tùy ý, một cây quyết định sẽ phân loại đúng hết các dữ liệu trong tập học và dẫn đến mô hình có thể dự đoán tệ trên tập kiểm thử.

Vì vậy Random Forests hay Rừng ngẫu nhiên thêm vào yếu tố ngẫu nhiên: Bằng cách sử dụng nhiều Cây quyết định và với mỗi Cây quyết định:

- Sử dụng dữ liệu ngẫu nhiên để xây dựng cây quyết định
- Lấy ngẫu nhiên các thuộc tính để đặt vào một đỉnh

Việc lựa chọn ngẫu nhiên như vậy sẽ tránh hiện tượng quá khớp trên từng cây, tuy nhiên lại dẫn đến kém khớp, hay nói cách khác thì mô hình có high bias. Tuy vậy, quyết định cuối cùng của Random Forests được tổng hợp từ nhiều cây quyết định, giúp cho mô hình có độ lệch và phương sai nhỏ, hay đầu ra là dự đoán tốt.

Thuật toán Random Forests Giả sử tập dữ liệu huấn luyện D gồm có n mẫu dữ liệu và mỗi dữ liệu có d thuộc tính. Chúng ta học K cây quyết định như sau:

1. Tạo K cây, mỗi cây được sinh ra như sau:
   (a) Xây dựng một tập con D_i bằng cách lấy ngẫu nhiên (có trùng lặp) từ $\boldsymbol{D}$.
   (b) Học cây thứ i từ D_i như sau:
   (c) Tại mỗi nút trong quá trình xây dựng cây:
      i. chọn ngẫu nhiên một tập con các thuộc tính
      ii. phân nhánh cây dựa trên tập thuộc tính đó.
   (d) Cây này sẽ được sinh ra với cỡ lớn nhất, không dùng cắt tỉa.
2. Mỗi phán đoán về sau thu được bằng cách lấy trung bình các phán đoán từ tất cả các cây.

Để xây dựng một tập con D_i từ D, chúng ta lấy ngẫu nhiên n dữ liệu từ tập dữ liệu huấn luyện với kĩ thuật Bootstrapping, hay còn gọi là lấy mẫu ngẫu nhiên có lặp lại (random sampling with replacement), có nghĩa là, chúng ta lấy mẫu được 1 mẫu dữ liệu thì không bỏ dữ liệu đấy ra mà vẫn giữ lại trong tập dữ liệu ban đầu, rồi tiếp tục lấy mẫu cho tới khi có đủ n mẫu dữ liệu. Khi dùng kĩ thuật này thì tập mới có thể có những mẫu dữ liệu bị trùng

nhau.

Random Forests là thuật toán dựa trên khái niệm Ensemble learning (Học kết hợp), tức là quá trình kết hợp nhiều bộ phân loại để giải quyết một vấn đề phức tạp để cải thiện hiệu suất mô hình. Random Forests là một thuật toán tuyệt vời ngay cả khi không điều chỉnh siêu tham số: Số lượng cây, số lượng mẫu huấn luyện trên cây, số lượng thuộc tính dùng để xây dựng cây... Tuy nhiên khi dùng thuật toán Random Forests, chúng ta nên chú ý đến các siêu tham số này.

3.2 Mô hình học sâu

3.2.1 Artificial Neural Network

Mạng neuron nhân tạo (Artificial Neural Network - ANN) là một mô hình tính toán lấy cảm hứng từ cấu trúc và chức năng của hệ thống thần kinh sinh học, đặc biệt là bộ não con người. ANN học cách thực hiện tác vụ nhất định bằng cách "học" những thuộc tính từ dữ liệu. Nó cũng cố gắng mô phỏng lại cách bộ não con người xử lý thông tin, cho phép nó có thể nhận diện các mẫu dữ liệu mới, phức tạp sau khi được huấn luyện trên bộ dữ liệu ban đầu.

Cụ thể, một mạng ANN bao gồm một tập hợp các đơn vị kết nối được gọi là các neuron (nút), được tổ chức thành các lớp. Các lớp phổ biến bao gồm:

- **Lớp đầu vào:** Nhận dữ liệu đầu vào ban đầu, chẳng hạn như pixel của một hình ảnh hoặc các từ trong một câu.
- **Lớp ẩn:** Thực hiện các phép tính trung gian để trích xuất các đặc trưng có ý nghĩa từ dữ liệu đầu vào.
- **Lớp đầu ra:** Tạo ra kết quả cuối cùng, chẳng hạn như phân loại một hình ảnh hoặc dự đoán giá trị.

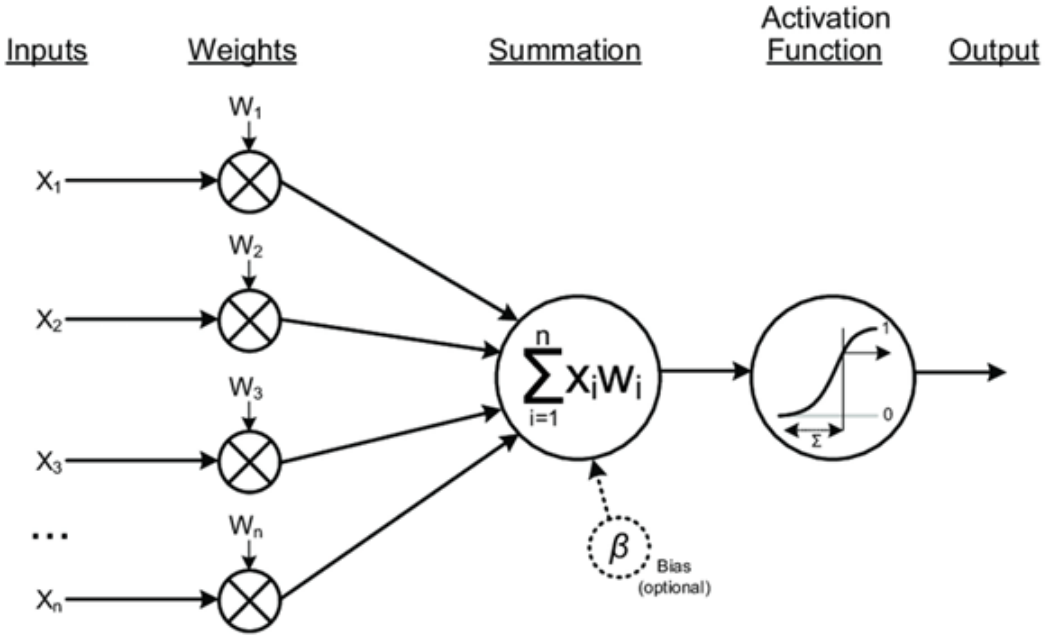
Mỗi kết nối giữa các neuron có một trọng số liên quan, biểu thị mức độ ảnh hưởng của một neuron đối với neuron khác. Trong quá trình huấn luyện, các trọng số này được điều chỉnh nhờ các thuật toán cụ thể để mạng có thể khớp với dữ liệu ban đầu, nhằm đưa ra các dự đoán chính xác hơn.

Một trong những đặc điểm quan trọng của ANN là khả năng xấp xỉ các hàm phức tạp. Cho một tập hợp các cặp giá trị đầu vào và đầu ra mẫu, ANN có thể học cách ánh xạ các đầu vào tới các đầu ra tương ứng. Điều này có nghĩa là ANN có thể khái quát hóa từ các ví dụ đã thấy để đưa ra dự đoán cho các đầu vào mới chưa từng gặp. Quá trình này được thực

hiện thông qua một thuật toán học, thuật toán phổ biến và kinh điển nhất là thuật toán lan truyền ngược (backpropagation).

Tín hiệu vào của mỗi một neuron (mỗi nốt trong mạng), trừ những nốt đầu tiên, là các nốt x_i, với $i = 1, m$. Ứng với mỗi nốt này là một trọng số w_i, $i = 1, m$. Ngoài ra còn có trọng số w_0. Net input được định nghĩa là một hàm dạng tuyến tính như sau:

$$\text{Net}(\mathbf{w}, \mathbf{x}) = w_0 + \sum_{i=1}^{m} x_i w_i$$



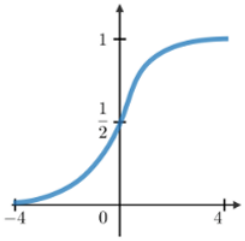
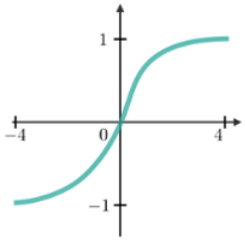
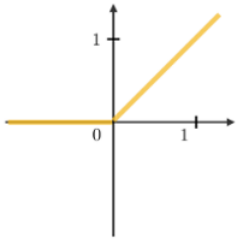
Hình 8: Minh họa tín hiệu vào và ra đến một neuron

Sau đó, Net output sẽ được đưa qua một hàm kích hoạt (activation function) $f(\cdot)$ để tạo thành giá trị đầu ra:

$$Out(\mathbf{w}, \mathbf{x}) = f(Net(\mathbf{w}, \mathbf{x}))$$

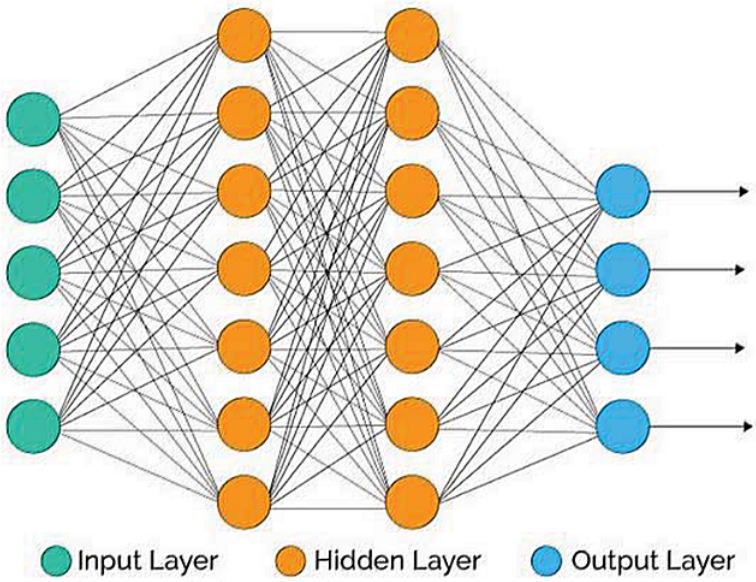
Một mạng neuron được gọi là kết nối đầy đủ (fully connected) nếu các đầu ra của một lớp kết nối với tất cả các neuron của lớp tiếp theo. Cấu trúc của một mạng neuron nhân tạo được thể hiện qua các đặc tính sau:

- Số đặc tính của tín hiệu vào và ra.
- Số lớp.

Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

Hình 9: Các hàm kích hoạt phổ biến

- Số neuron (số nút) ở mỗi lớp.
- Số lượng neuron kết nối với một neuron khác.
- Sự kết nối của các neuron trong cùng lớp hoặc giữa các lớp với nhau.



Hình 10: Các lớp trong mạng neuron nhân tạo

Mô hình của chúng em sẽ được huấn luyện với mục tiêu khớp với dữ liệu đầu vào và dự đoán tốt trên tập đánh giá, được hiện thực hóa thông qua việc tối ưu hóa hàm mất mát (loss function) như sau:

$$L(\mathbf{w}) = \frac{1}{|\mathbf{D}|} \sum_{x \in \mathbf{D}} \text{loss}(d_x, \text{out}(x))$$

Quá trình huấn luyện bao gồm việc tìm ra tập các tham số $\mathbf{w}$ sao cho hàm mất mát

$L(\mathbf{w})$ đạt giá trị nhỏ nhất có thể trên toàn bộ tập dữ liệu huấn luyện.

3.2.2 Recurrent Neural Network

Trong lĩnh vực học sâu, RNN nổi bật như một kiến trúc mạng nơ-ron được thiết kế đặc biệt để xử lý dữ liệu tuần tự, nơi mà thứ tự và sự phụ thuộc giữa các phần tử dữ liệu đóng vai trò then chốt. Khác với các mạng nơ-ron truyền thẳng (Feedforward Neural Networks) chỉ xử lý từng đầu vào một cách độc lập, RNN được trang bị "trí nhớ" cho phép chúng duy trì thông tin về các đầu vào trước đó để tác động đến việc xử lý các đầu vào hiện tại.

Chính khả năng này đã khiến RNN trở thành một công cụ mạnh mẽ cho các tác vụ liên quan đến chuỗi, chẳng hạn như xử lý ngôn ngữ tự nhiên (NLP), nhận dạng giọng nói, phân tích chuỗi thời gian, phân tích cảm xúc,...

Kiến trúc cơ bản của RNN:

Trong các mô hình RNN, đặc biệt là trong các bài toán xử lý ngôn ngữ tự nhiên (NLP), embedding input của từ đóng vai trò cực kỳ quan trọng. Thay vì trực tiếp đưa các từ vào mô hình (vì mô hình chỉ có thể xử lý thông tin dưới dạng số) dưới dạng chuỗi ký tự, chúng ta thường biểu diễn mỗi từ bằng một vectơ số thực, được gọi là "word embedding".

Trong RNN, một word/token sẽ được embedding hay mã hoá làm đầu vào cho RNN. Các vectơ embedding thu được từ lớp Embedding được đưa vào mô hình RNN. RNN sẽ trực tiếp xử lý chuỗi embedding này và sử dụng thông tin ngữ nghĩa của các từ để thực hiện tác vụ.

$$e_t = W_e x_t + b_e$$

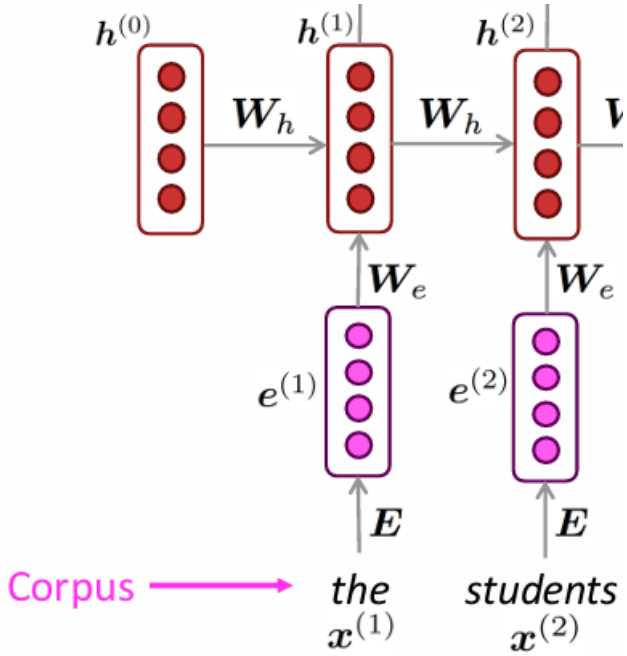
Một trạng thái ẩn (hidden state) trong RNN cơ bản nhận hai loại đầu vào: đầu vào hiện tại (x_t) tại bước thời gian t, và trạng thái ẩn từ bước thời gian trước đó (h_{t-1}). Bên trong RNN, đầu vào hiện tại và trạng thái ẩn trước đó được kết hợp thông qua các phép biến đổi tuyến tính (nhân với ma trận trọng số) và một hàm kích hoạt phi tuyến (ví dụ: sigmoid, tanh, ReLU). Kết quả của quá trình này là trạng thái ẩn hiện tại (h_t), chứa đựng thông tin đã được cập nhật từ đầu vào hiện tại và "ký ức" từ các bước thời gian trước. Trạng thái ẩn này sau đó được truyền sang bước thời gian tiếp theo và cũng có thể được sử dụng để đưa ra dự đoán hoặc kết quả đầu ra (y_t) tại bước thời gian hiện tại.

Công thức toán học cơ bản cho một trạng thái ẩn trong RNN có thể được biểu diễn như sau:

$$h_t = f(W_e e_t + W_h h_{t-1} + b_h)$$
$$y_t = g(W_y h_t + b_y)$$

Trong đó:

- e_t là vectơ embedding của đầu vào tại thời điểm t.
- h_{t-1} là trạng thái ẩn từ thời điểm $t-1$ (với h_0 thường được khởi tạo bằng vectơ không).
- W_h, W_e, W_y là các ma trận trọng số được học trong quá trình huấn luyện.
- b_h, b_y là các vectơ bias.
- f là hàm kích hoạt cho trạng thái ẩn (thường là tanh hoặc ReLU).
- g là hàm kích hoạt cho đầu ra (tùy thuộc vào tác vụ, ví dụ: softmax cho phân loại).



Hình 11: Lớp ẩn trong mạng RNN

Ứng dụng RNN trong phân tích cảm xúc:

Trong bài toán phân tích cảm xúc, một đoạn văn bản (ví dụ: một bình luận, một bài đánh giá sản phẩm) có thể được xem như một chuỗi các từ (hoặc các đơn vị nhỏ hơn như ký tự hoặc word embeddings). RNN có khả năng duyệt qua chuỗi này theo tuần tự, từ đầu đến cuối, và xây dựng một biểu diễn nội dung của toàn bộ văn bản trong trạng thái ẩn cuối cùng. Trạng thái ẩn này sau đó có thể được sử dụng làm đầu vào cho một lớp phân loại (ví

dụ: một lớp fully connected kết hợp với hàm softmax) để dự đoán nhãn cảm xúc (ví dụ: tích cực, tiêu cực, trung tính).

Huấn luyện mô hình RNN:

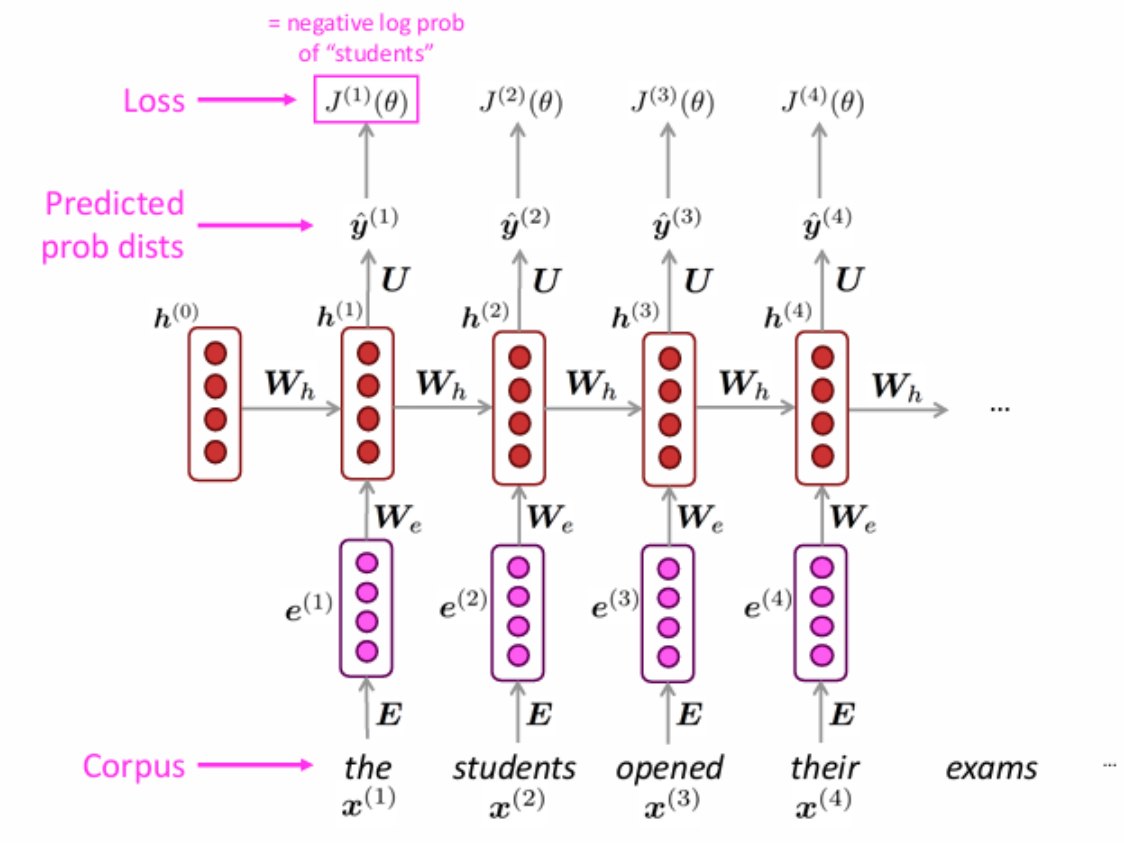
Hàm mất mát ở bước t là cross-entropy giữa phân phối xác suất dự đoán $\hat{y}^{(t)}$, và xác suất xuất hiện từ đúng tiếp theo $y^{(t)}$ (one-hot vector $x^{(t+1)}$):

$$J^{(t)}(\theta) = CE(\hat{y}^{(t)}, y^{(t)}) = -\sum_{w \in V} y_w^{(t)} \log \hat{y}_w^{(t)} = -\log \hat{y}_{x_{t+1}}^{(t)}$$

Sau đó, tính trung bình loss tại từng điểm dữ liệu để có được loss tổng thể cho toàn bộ tập huấn luyện.

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T}\sum_{t=1}^{T} -\log \hat{y}_{x_{t+1}}^{(t)}$$

Tương tự như ANN, quá trình huấn luyện bao gồm việc tìm ra tập các tham số sao cho hàm mất mát trên đạt giá trị nhỏ nhất có thể trên toàn bộ tập dữ liệu huấn luyện.

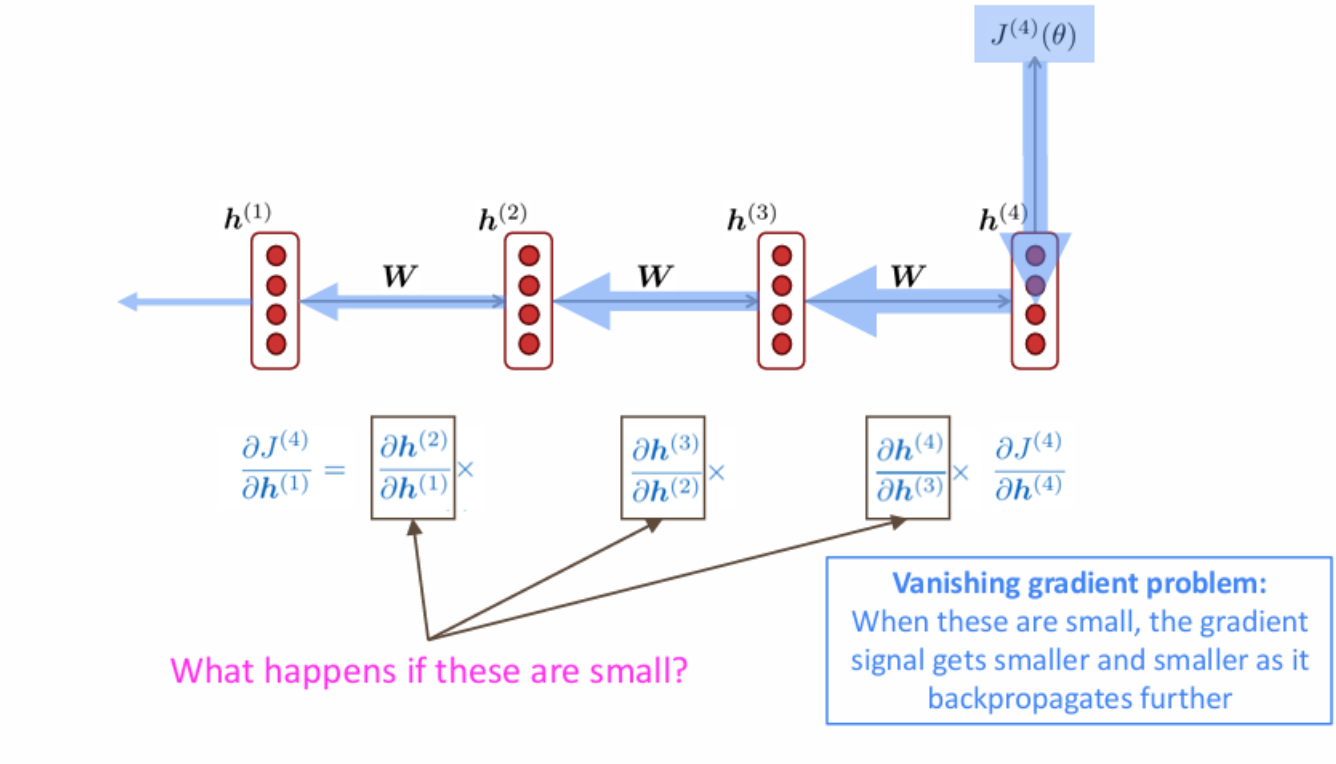


Hình 12: Kiến trúc mạng RNN

Tuy nhiên, RNN cũng có một vấn đề lớn làm quá trình training có thể diễn ra không

hiệu quả, hiện tượng đó được gọi là vanishing gradient.

Vấn đề này xảy ra khi huấn luyện các mạng RNN sâu, tức là các mạng có nhiều lớp. Trong quá trình huấn luyện, các tham số của mạng được cập nhật dựa trên gradient của hàm mất mát.



Hình 13: Hiện tượng Vanishing Gradient trong mạng RNN

Chính vì vấn đề vanishing gradient của RNN, các nhà nghiên cứu đã phát triển các kiến trúc và kỹ thuật khác để giảm thiểu hoặc giải quyết vấn đề này.

Một trong những cải tiến nhằm tránh tình trạng vanishing gradient đó chính là mô hình Long Short-Term Memory (LSTM), sẽ được trình bày ở phần sau.

3.2.3 Long-Short Term Memory

Như đã nói trên, RNN có hiện tượng vanishing gradient nếu như mạng có số lớp lớn. Chính vấn đề đó có thể làm cho mạng RNN bị mất thông tin từ những bước thời gian ở quá xa bước thời gian hiện tại, từ đó có thể dẫn tới mô hình không có hiệu suất tốt, không tận dụng được tối đa ngữ cảnh mình đang có.

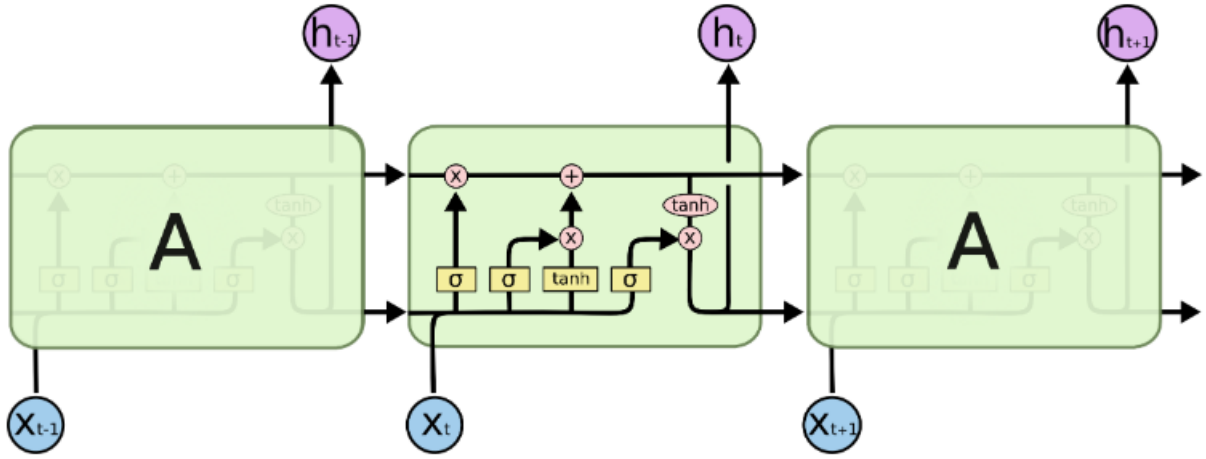
Lấy một ví dụ đơn giản như thế này. Đôi lúc ta chỉ cần xem lại thông tin vừa có thôi là đủ để biết được tình huống hiện tại. Ví dụ, ta có câu: "các đám mây trên bầu trời" thì ta chỉ cần đọc tới "các đám mây trên bầu" là đủ biết được chữ tiếp theo là "trời" rồi. Trong tình

huống này, khoảng cách tới thông tin có được cần để dự đoán là nhỏ, nên RNN hoàn toàn có thể học được.

Nhưng trong nhiều tình huống ta buộc phải sử dụng nhiều ngữ cảnh hơn để suy luận. Ví dụ, dự đoán chữ cuối cùng trong đoạn: "Tôi lớn lên ở Việt Nam .... Tôi nói tiếng Việt rất giỏi.". Rõ ràng là các thông tin gần ("Tôi nói giỏi") chỉ có phép ta biết được đằng sau nó sẽ là tên của một ngôn ngữ nào đó, còn không thể nào biết được đó là tiếng gì. Muốn biết là tiếng gì, thì ta cần phải có thêm ngữ cảnh "Tôi lớn lên ở Việt Nam" nữa mới có thể suy luận được. Rõ ràng là khoảng cách thông tin lúc này có thể đã khá xa rồi.

Và đây cũng chính là lý do, là động lực để người ta phát triển nên mô hình Long-Short Term Memory, hay còn gọi là LSTM.

Mạng bộ nhớ dài-ngắn (Long Short Term Memory networks), thường được gọi là LSTM - là một dạng đặc biệt của RNN, được cho rằng nó có khả năng học được các phụ thuộc xa. LSTM được giới thiệu bởi Hochreiter & Schmidhuber (1997), và sau đó đã được cải tiến và phổ biến bởi rất nhiều người trong ngành. Chúng hoạt động cực kì hiệu quả trên nhiều bài toán khác nhau nên dần đã trở nên phổ biến như hiện nay.

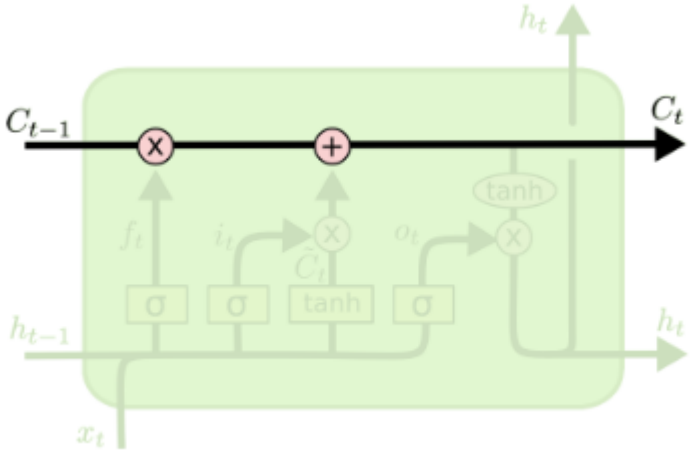


Hình 14: Kiến trúc cell trong LSTM.

LSTM cũng có kiến trúc dạng chuỗi như RNN, nhưng các mô-đun trong nó có cấu trúc khác với mạng RNN chuẩn. Thay vì chỉ có một tầng mạng nơ-ron, chúng có các tầng khác nhau ở mỗi cell nhằm tương tác với nhau một cách rất đặc biệt.

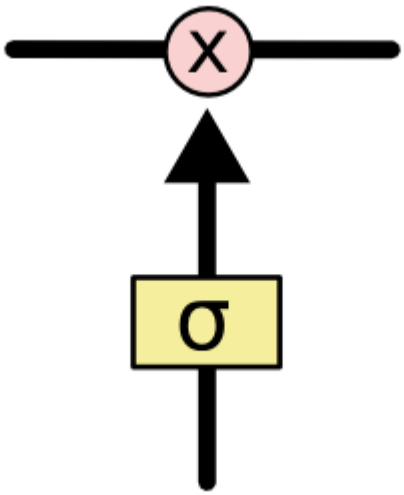
Chìa khóa của LSTM là trạng thái tế bào (cell state) - chính đường chạy thông ngang phía trên của sơ đồ hình vẽ.

Trạng thái tế bào là một dạng giống như băng truyền. Nó chạy xuyên suốt tất cả các mắt xích (các nút mạng) và chỉ tương tác tuyến tính đôi chút. Vì vậy mà các thông tin có thể dễ dàng truyền đi thông suốt mà không sợ bị thay đổi.



Hình 15: Cell state trong LSTM.

LSTM có khả năng bỏ đi hoặc thêm vào các thông tin cần thiết cho trạng thái tế báo, chúng được điều chỉnh bởi các nhóm được gọi là cổng (gate). Các cổng là nơi sàng lọc thông tin đi qua nó, chúng được kết hợp bởi một tầng mạng sigmoid và một phép nhân.



Hình 16: Gate trong LSTM.

Hàm sigmoid tạo ra các giá trị trong khoảng từ 0 đến 1, cho biết mức độ thông tin được phép đi qua. Giá trị 0 tương ứng với việc xoá hoàn toàn dòng thông tin, trong khi giá trị 1 cho phép thông tin truyền qua mà không bị cản trở. LSTM sử dụng 3 cơ chế điều khiển như vậy để điều chỉnh trạng thái của cell.

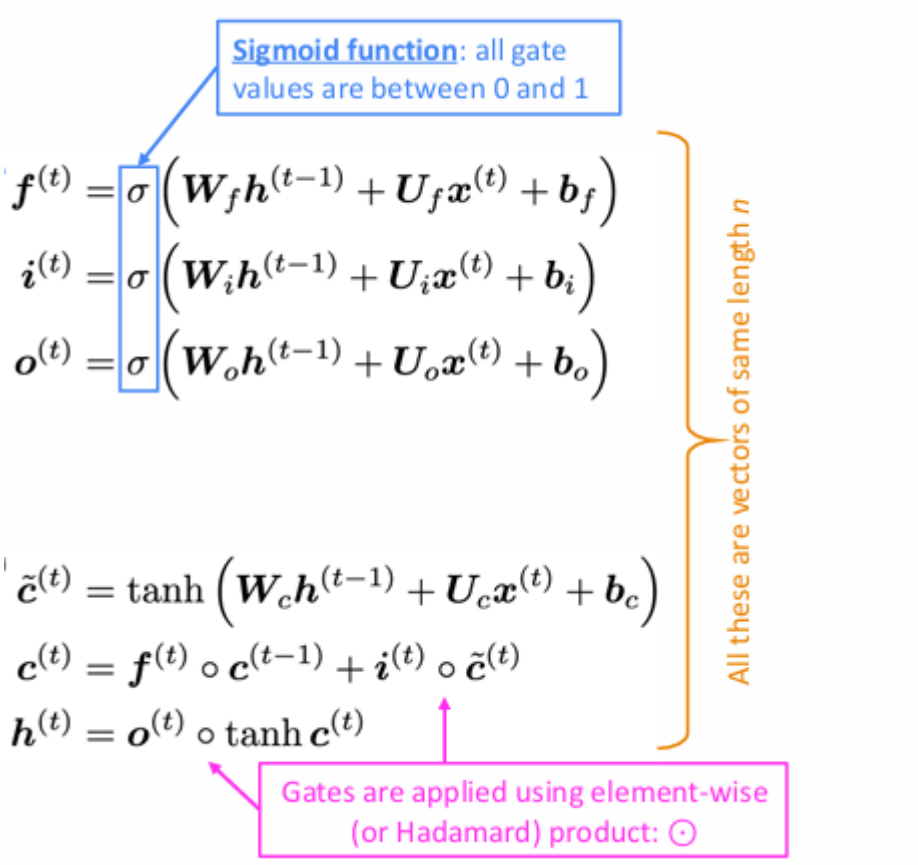
Bên trong LSTM

Bước đầu tiên của LSTM là quyết định xem thông tin nào cần giữ lại từ cell state. Quyết định này được đưa ra bởi một hàm sigmoid - gọi là forget gate layer. Nó sẽ lấy đầu

vào là h_{t-1} và x_t rồi đưa ra kết quả là một số trong khoảng $[0, 1]$ cho mỗi số trong trạng thái tế bào C_{t-1}. Đầu ra là 1 thể hiện rằng nó giữ toàn bộ thông tin lại, còn 0 chỉ rằng toàn bộ thông tin sẽ bị bỏ đi.

Bước tiếp theo là quyết định xem thông tin mới nào sẽ được lưu vào trạng thái tế bào. Việc này gồm 2 phần. Đầu tiên là sử dụng một tầng sigmoid được gọi là "tầng cổng vào" (input gate layer) để quyết định giá trị nào ta sẽ cập nhập. Tiếp theo đưa vector qua hàm *tanh* tạo ra một vector với giá trị mới $\tilde{C}_t$ cộng vào cho trạng thái. Trong bước tiếp theo, mô hình sẽ kết hợp 2 giá trị đó lại để cuối cùng cập nhập trạng thái mới cho cell.

Tiếp theo, cập nhập trạng thái tế bào cũ C_{t-1} thành trạng thái mới C_t. Ở đây, mô hình thực hiện sẽ nhân trạng thái cũ với f_t để bỏ đi những thông tin ta quyết định quên lúc trước. Sau đó cộng thêm $i_t * \tilde{C}_t$.



Hình 17: Các phép toán được thực hiện trong LSTM.

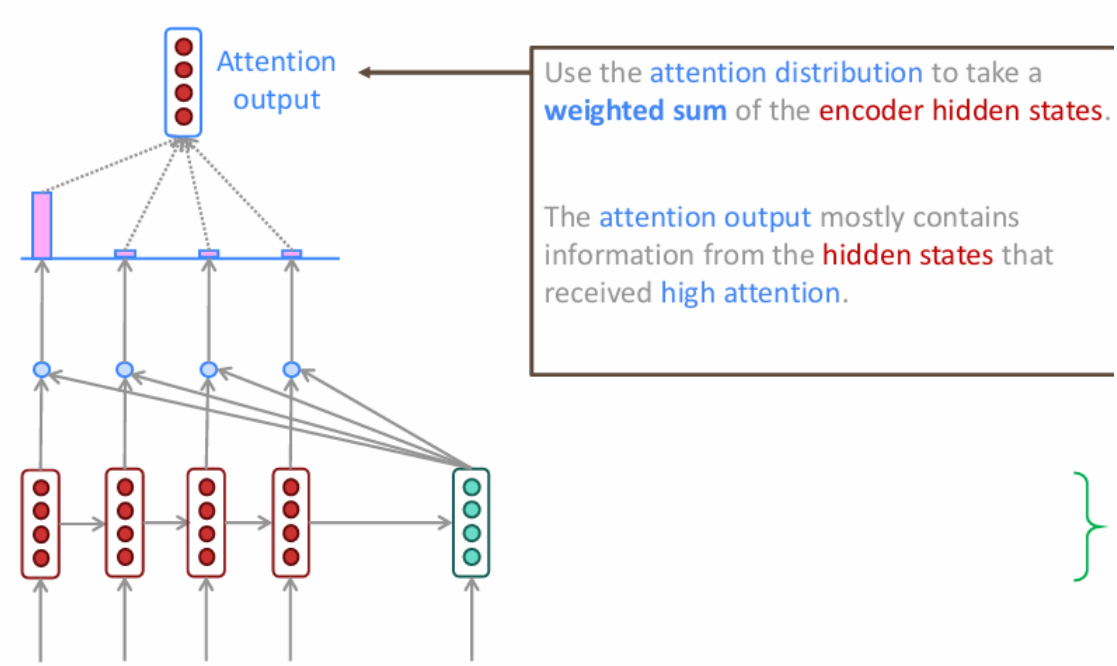
Cuối cùng, giá trị đầu ra sẽ dựa vào cell state, nhưng sẽ được tiếp tục sàng lọc. Đầu tiên, ta đưa qua hàm sigmoid để quyết định phần nào của trạng thái tế bào ta muốn đưa ra. Sau đó, ta đưa cell state qua một hàm *tanh* để co giá trị nó về khoảng $[-1, 1]$, và nhân nó với đầu ra của hàm sigmoid để được giá trị đầu ra mong muốn.

Attention

Sau khi có hidden state của tất cả điểm thời gian, một mạng MLP được áp dụng cho output của mô hình. Mô hình này ánh xạ mỗi trạng thái ẩn của nó thành một giá trị, được coi là score hay là mức độ "đóng góp" của trạng thái ở mỗi thời điểm đến kết quả đầu ra.

Các score sẽ lần lượt được chuyển qua hàm softmax để tạo ra các trọng số attention. Hàm softmax chuẩn hóa các giá trị thành một phân phối xác suất, trong đó mỗi trọng số nằm trong khoảng từ 0 đến 1 và tổng của tất cả các trọng số bằng 1.

Kết quả sau đó được tổng hợp theo chiều của ma trận đầu vào để tạo ra một vector duy nhất, chứa thông tin được chọn lọc từ các trạng thái ẩn của model. Vector này sẽ được sử dụng để dự đoán kết quả và thực hiện huấn luyện cho mô hình.



Hình 18: Attention trong Seq2Seq model.

3.2.4 Deep Learning Ensembler

Phương pháp ensemble là một kỹ thuật mạnh mẽ trong học máy, kết hợp dự đoán từ nhiều mô hình với mong muốn cải thiện độ chính xác và tính ổn định của kết quả cuối cùng. Ý tưởng cơ bản là các mô hình khác nhau có thể mắc những lỗi khác nhau, và bằng cách kết hợp dự đoán của chúng, những lỗi này được "kỳ vọng" có thể được giảm thiểu. Bởi vì theo như logic của con người chẳng hạn, càng nhiều người tham gia dự đoán thì khả năng dự đoán đúng có thể sẽ cao hơn.

Ở phương pháp này, chúng em chỉ thực hiện inference các điểm dữ liệu qua từng mô

hình. Sau đó dựa trên những dự đoán của từng mô hình, chúng em sẽ thực hiện phép toán cộng trung bình các điểm số dự đoán từ ba mô hình RNN, LSTM, và ANN, từ đó lấy làm kết quả dự đoán cuối cùng.

Quá trình của phương pháp Ensemble:

1. Thực hiện việc training cho các mô hình trên.
2. Lấy weight của từng mô hình ở các epoch có kết quả dự đoán tốt nhất trên tập Validation.
3. Thực hiện inference các điểm dữ liệu qua từng mô hình.
4. Lấy trung bình cộng điểm số dự đoán từ mỗi mô hình.
5. Thực hiện dự đoán.

Phép cộng trung bình đơn giản:

Đối với mỗi mẫu dữ liệu mới, chúng em lấy các vector điểm số đầu ra từ ba mô hình. Sau đó thực hiện kết quả dự đoán là 0 hay 1. Sau đó, chúng em thực hiện phép cộng các dự đoán này với nhau để thu được điểm số ensemble trung bình, cuối cùng làm tròn để lấy dự đoán cuối cùng.

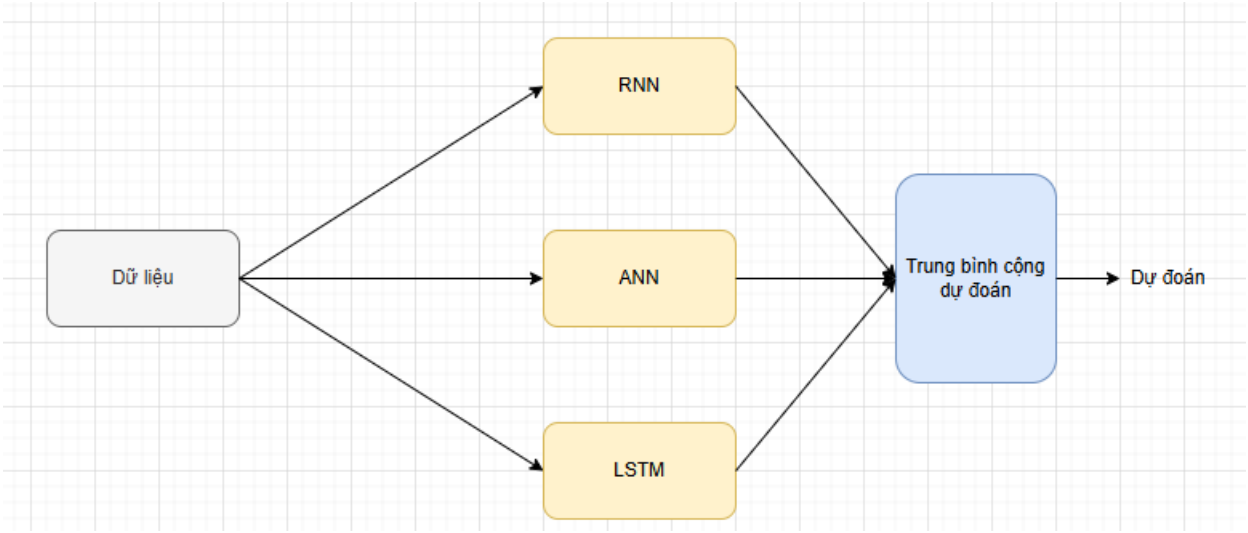
Ví dụ, giả sử chúng ta có ba mô hình dự đoán xác suất cho hai lớp cảm xúc (tiêu cực, tích cực) cho một đoạn văn bản như sau:

Mô hình RNN: $[0.9, 0.1]$ tức là label = 1.

Mô hình LSTM: $[0.3, 0.7]$ tức là label = 0.

Mô hình ANN: $[0.8, 0.2]$ tức là label = 1.

Vậy điểm số ensemble cuối cùng sẽ là $\frac{1+0+1}{3} = 0.66$, làm tròn sẽ được label = 1. Và đây sẽ là kết quả dự đoán cuối cùng.



Hình 19: Pipeline cho phương pháp ensemble.

4 Kết quả thực nghiệm

4.1 Cấu hình thực nghiệm và các mô hình

Trong phần thực nghiệm, chúng em tiến hành đánh giá hiệu quả của nhiều thuật toán học máy và học sâu khác nhau trên tập dữ liệu **Stanford Sentiment Treebank 2 (SST-2)**, cùng 3 bộ dữ liệu sau khi qua quá trình tiền xử lý dữ liệu cũng như tăng cường dữ liệu mà chúng em đã thực hiện. Dữ liệu bao gồm khoảng **67.000 câu huấn luyện và 872 câu kiểm tra**. Mọi mô hình được huấn luyện và đánh giá trên cùng một quy trình xử lý dữ liệu và sử dụng 2 nền tảng **Google Colab**, **Kaggle**.

Mô hình và Thiết lập. chúng em sử dụng các mô hình học máy gồm: *Naive Bayes, Logistic Regression, SVM, KNN, Decision Tree, Random Forest*, cùng với các mô hình học sâu: *Artificial Neural Network (ANN), Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM)* và một mô hình *ensemble* kết hợp *RNN–LSTM–ANN* bằng phương pháp hard voting.

Với phần xử lý dữ liệu, văn bản sau tiền xử lý được vector hóa bằng **TF-IDF** với giới hạn tối đa *10.000 từ đặc trưng* cho các mô hình học máy. Thông số cài đặt được chọn như sau: *KNN* dùng $k = 5$ với khoảng cách Euclidean, *SVM* sử dụng kernel tuyến tính với $C = 1.0$, *Naive Bayes* là MultinomialNB, *Logistic Regression* dùng solver 'liblinear', *Decision Tree* có độ sâu tối đa là 20, *Random Forest* gồm 100 cây quyết định.

Với các mô hình học sâu, chúng em sử dụng 2 loại Embedding layer khác nhau: học từ đầu embedding và sử dụng pretrained-embedding với kích thước vector là **chiều**. Tất cả mô

hình học sâu được huấn luyện với **optimizer Adam** trong **10 epoch**, không sử dụng early stopping.

Trong mô hình *ensemble*, chúng em huấn luyện riêng lẻ ba mô hình ANN, RNN và LSTM rồi kết hợp dự đoán bằng cách sử dụng kết quả voting từ các mô hình để tìm ra kết quả cuối cùng.

Tiền xử lý và Tăng cường dữ liệu. Quy trình tiền xử lý bao gồm chuyển văn bản về chữ thường, loại bỏ dấu câu và chú thích, chuẩn hóa khoảng trắng, token hóa theo từ, loại bỏ stop words, số và các token ngắn. Sau đó, chúng em áp dụng ba kỹ thuật tăng cường dữ liệu chính:

- **Easy Data Augmentation (EDA)**: thay từ đồng nghĩa, chèn từ ngẫu nhiên và đảo trật tự từ.
- **Embedding-based augmentation**: thay thế từ bằng các từ gần nhau trong không gian embedding.
- **WordNet-based augmentation**: sử dụng các từ đồng nghĩa từ WordNet.

Đánh giá. Tất cả mô hình được đánh giá theo bốn chỉ số: **Accuracy, Precision, Recall** và **F1-score** để đảm bảo phân tích toàn diện về hiệu suất phân loại.

4.2 Ảnh hưởng của các bước tiền xử lý và tăng cường dữ liệu

Sau khi áp dụng các kỹ thuật tiền xử lý văn bản như chuẩn hóa chữ, loại bỏ dấu câu, chuẩn hóa khoảng trắng, loại bỏ từ dừng, từ ngắn và token số, chúng em tiến hành huấn luyện các mô hình trên bộ dữ liệu gốc (SST-2) và các phiên bản mở rộng của nó thông qua ba phương pháp tăng cường dữ liệu: (i) **EDA (Easy Data Augmentation)** – gồm hoán vị, chèn từ, thay từ đồng nghĩa; (ii) **EDA kết hợp biểu diễn từ (embedding)** – dùng biểu diễn từ học được để thay thế từ ngữ trong câu; (iii) **EDA kết hợp WordNet** – thay từ đồng nghĩa dựa trên WordNet để đảm bảo tính ngữ nghĩa hơn.

Chúng em huấn luyện lại toàn bộ các mô hình với từng phiên bản dữ liệu để đánh giá ảnh hưởng của từng phương pháp tăng cường. Các mô hình được đánh giá trên bốn tiêu chí: *accuracy*, *precision*, *recall*, và *F1-score*.

Nhìn chung, các kỹ thuật tăng cường dữ liệu giúp cải thiện hiệu suất của hầu hết mô hình, đặc biệt rõ rệt đối với các mô hình học sâu như ANN, RNN, và LSTM. Ví dụ, với mô hình LSTM sử dụng embedding học sẵn, F1-score tăng từ 0.8272 (gốc) lên 0.84856 (EDA), và cao nhất ở mức 0.84605 (EDA + WordNet). Với mô hình RNN, điểm F1 tăng từ 0.8381 (gốc) lên 0.8539 (EDA + WordNet). Các mô hình đơn giản hơn như Naive Bayes, Logistic

Regression, hoặc SVM cũng có cải thiện nhưng không đáng kể.

Ngoài ra, chúng em quan sát thấy rằng: (1) **EDA kết hợp WordNet** là phương pháp hiệu quả nhất trên nhiều mô hình, thể hiện qua điểm F1 cao và ổn định; (2) Các mô hình tổng hợp như **ensemble của ANN-RNN-LSTM** cho kết quả tốt nhất với F1-score lên đến 0.8232 (EDA + WordNet); (3) Một số mô hình như KNN và KMeans không thể hiện hiệu suất tốt sau các kỹ thuật tăng cường dữ liệu, thậm chí còn có xu hướng giảm hiệu suất.

4.3 Chi tiết đánh giá

4.3.1 Thuật toán học sâu

1. Bộ dữ liệu gốc

model	accuracy	precision	recall	f1_score
ann_pretrained_embedding	0.8112	0.8179	0.8076	0.8127
rnn_pretrained_embedding	0.83984	0.8231	0.8537	0.8381
lstm_pretrained_embedding	0.83464	0.7795	0.8812	0.8272
DLEnsemble_pretrained_embedding	0.84352	0.7356	0.9071	0.8263
ann_random_embedding	0.8125	0.84746	0.76923	0.80645
rnn_random_embedding	0.76042	0.78771	0.72308	0.75401
lstm_random_embedding	0.79036	0.78987	0.8	0.7949
DLEnsemble_random_embedding	0.8268	0.5872	0.9124	0.7145

Nhìn chung, bảng kết quả cho thấy một xu hướng khá rõ ràng: việc sử dụng embedding được huấn luyện trước thường mang lại hiệu suất tốt hơn hoặc tương đương so với việc sử dụng embedding khởi tạo ngẫu nhiên trên hầu hết các kiến trúc mô hình.

Ngoài ra, việc sử dụng pretrained embedding cũng giúp các mô hình giảm được thời gian huấn luyện đi đáng kể mà vẫn có thể đạt được kết quả tốt do việc sử dụng knowledge transfer từ mô hình có sẵn. Bên cạnh đó, từ bản Voting Classifier đã mang lại kết quả tốt nhất trên độ đo accuracy và recall nhưng lại thể hiện kết quả rất tệ với độ đo precision.

2. Bộ dữ liệu gốc và dữ liệu từ PP EDA

- **LSTM + pretrained embedding** tiếp tục thể hiện hiệu suất vượt trội nhất với F1-score = 0.84856. Điều này cho thấy mô hình này học tốt từ các biến thể ngữ nghĩa do EDA tạo ra nhờ khả năng ghi nhớ ngữ cảnh dài hạn.
- **RNN + pretrained embedding** cũng hoạt động rất tốt với precision cao (0.86486), tuy recall thấp hơn một chút. Điều này phản ánh mô hình rất chắc chắn với các mẫu dự đoán nhưng có thể bỏ sót một số trường hợp cảm xúc.

model	accuracy	precision	recall	f1_score
ann_pretrained_embedding	0.81771	0.81888	0.82308	0.82097
rnn_pretrained_embedding	0.84375	0.86486	0.82051	0.84211
lstm_pretrained_embedding	0.84245	0.82885	0.86923	0.84856
DLEnsemble_pretrained_embedding	0.85416	0.75900	0.91080	0.82800
ann_random_embedding	0.81030	0.82140	0.81180	0.81450
rnn_random_embedding	0.77995	0.76755	0.81282	0.78954
lstm_random_embedding	0.81120	0.79951	0.83846	0.81852
DLEnsemble_random_embedding	0.83870	0.61720	0.92570	0.73180

- **DLEnsemble + pretrained embedding** đạt *recall* cao nhất (0.9108) nhưng precision thấp (0.759), cho thấy xu hướng dự đoán nhiều hơn, dễ xảy ra overgeneralization — mô hình dự đoán được đa dạng trường hợp nhưng không đảm bảo chính xác.
- **ANN + pretrained embedding** duy trì hiệu suất ổn định (F1-score = 0.82097), cho thấy mô hình đơn giản này vẫn có thể hưởng lợi từ dữ liệu tăng cường dù không mạnh về khai thác ngữ cảnh.
- **LSTM + random embedding** vẫn học tốt từ dữ liệu tăng cường (F1 = 0.81852), chứng minh khả năng tự học embedding từ các mẫu được đa dạng hóa ngữ nghĩa.
- **DLEnsemble + random embedding** tiếp tục có recall rất cao (0.9257), nhưng precision cực thấp (0.6172), khiến F1 giảm mạnh còn 0.7318 — một lần nữa nhấn mạnh vấn đề dự đoán quá rộng.
- **RNN + random embedding** và **ANN + random embedding** có hiệu suất trung bình (F1 $\approx$ 0.79–0.81), cho thấy mức độ học ổn định nhưng không nổi bật nếu không có embedding huấn luyện sẵn.

Kết luận: EDA giúp cải thiện hiệu quả của các mô hình học sâu, đặc biệt là khi kết hợp với embedding được huấn luyện sẵn. Tuy nhiên, một số mô hình như DLEnsemble có thể bị mất cân bằng precision/recall, cho thấy cần kiểm soát chất lượng dữ liệu tăng cường một cách chặt chẽ.

3. Bộ dữ liệu gốc và dữ liệu từ PP EDA, Embedding

Khi áp dụng phương pháp EDA để tăng cường dữ liệu và sử dụng embedding, các mô hình học sâu thể hiện những đặc điểm sau:

- **Nhìn chung, tất cả các mô hình đạt hiệu suất khá ổn định và cao**, đặc biệt là ở chỉ số *f1-score*, cho thấy kết hợp EDA, embedding cung cấp thông tin hữu ích cho quá trình học.

model	accuracy	precision	recall	f1_score
ann_pretrained_embedding	0.82292	0.84511	0.79744	0.82058
rnn_pretrained_embedding	0.84245	0.84224	0.84872	0.84547
lstm_pretrained_embedding	0.84766	0.88669	0.80256	0.84253
DLEnsemble_pretrained_embedding	0.85150	0.72820	0.91030	0.80910
ann_random_embedding	0.81250	0.83065	0.79231	0.81102
rnn_random_embedding	0.76042	0.72889	0.84103	0.78095
lstm_random_embedding	0.82161	0.82687	0.82051	0.82368
DLEnsemble_random_embedding	0.84244	0.64870	0.94050	0.76780

- **LSTM + pretrained embedding** có precision rất cao (0.88669), mặc dù recall thấp hơn một chút, dẫn đến F1 vẫn rất tốt (0.84253). Điều này cho thấy mô hình nhận diện cảm xúc một cách chính xác và ít nhầm lẫn hơn.
- **DLEnsemble + pretrained embedding** đạt recall cao nhất (0.9103), nhưng precision lại thấp (0.7282), kéo theo f1-score chỉ ở mức trung bình (0.8091). Mô hình có xu hướng dự đoán nhiều cảm xúc hơn, dễ bị overgeneralization.
- **RNN + pretrained embedding** có cân bằng rất tốt giữa precision và recall, dẫn đến f1-score cao nhất (0.84547). Điều này cho thấy mô hình học được các mẫu ngôn ngữ tổng quát tốt từ dữ liệu được tăng cường.
- **ANN + pretrained embedding** vẫn giữ hiệu suất ổn định, f1-score ở mức 0.82058 — cho thấy mô hình này vẫn hưởng lợi từ dữ liệu EDA dù không khai thác sâu ngữ cảnh.
- **Các mô hình dùng random embedding (không huấn luyện trước)** có hiệu suất thấp hơn một chút, nhưng vẫn khả quan. Trong đó:
  - **LSTM + random** đạt f1-score cao nhất (0.82368), cho thấy mô hình có khả năng tự học embedding tốt từ dữ liệu tăng cường.
  - **DLEnsemble + random** tiếp tục có recall rất cao (0.9405) nhưng precision lại thấp (0.6487), khiến f1-score chỉ đạt 0.7678 — tiếp tục cho thấy hiện tượng mất cân bằng giữa nhạy và chính xác.
  - **RNN + random** có recall cao nhưng precision thấp, f1-score trung bình (0.78095) — phù hợp với đặc tính của RNN nhạy cảm với thay đổi từ vựng do EDA gây ra.
  - **ANN + random** vẫn ổn định với f1-score là 0.81102 — không bị ảnh hưởng nhiều bởi EDA.

  Kết luận: Tăng cường dữ liệu bằng kết hợp EDA, embedding giúp cải thiện hiệu suất

của các mô hình học sâu, đặc biệt khi kết hợp với các mô hình có khả năng khai thác ngữ cảnh như RNN hoặc LSTM. Tuy nhiên, cần chú ý đến sự đánh đổi giữa *precision* và *recall*, nhất là trong các mô hình ensemble.

4. Bộ dữ liệu tổng hợp

model	accuracy	precision	recall	f1_score
ann_pretrained_embedding	0.82552	0.84930	0.81080	0.82850
rnn_pretrained_embedding	0.84896	0.83911	0.86923	0.85390
lstm_pretrained_embedding	0.84500	0.85370	0.83846	0.84605
DLEnsemble_pretrained_embedding	0.85937	0.74620	0.91800	0.82320
ann_random_embedding	0.81390	0.79013	0.85900	0.82350
rnn_random_embedding	0.75781	0.76020	0.76410	0.76215
lstm_random_embedding	0.81120	0.81330	0.81538	0.81434
DLEnsemble_random_embedding	0.82810	0.71280	0.92050	0.80350

- Các mô hình sử dụng **pretrained embedding** nhìn chung đạt hiệu suất cao hơn so với khi huấn luyện trên dữ liệu gốc. Đặc biệt:
  - **RNN + pretrained embedding** đạt F1-score cao nhất (0.8539), chứng tỏ mô hình này đã khai thác tốt ngữ nghĩa mở rộng từ WordNet và EDA.
  - **LSTM + pretrained embedding** vẫn giữ được độ ổn định cao (F1 = 0.84605), cho thấy khả năng học chuỗi ngữ nghĩa được tăng cường hiệu quả.
  - **ANN + pretrained embedding** có cải thiện nhẹ (F1 = 0.8285 so với 0.8127 ở dữ liệu gốc), thể hiện rằng ngay cả các mô hình đơn giản cũng được hưởng lợi từ tăng cường dữ liệu.
  - **DLEnsemble + pretrained embedding** tiếp tục có recall rất cao (0.918), tuy precision thấp (0.7462), dẫn đến F1 = 0.8232 — phù hợp với xu hướng overgeneralization đã thấy trước đó.
- Các mô hình sử dụng **random embedding** vẫn cải thiện so với dữ liệu gốc:
  - **ANN + random embedding** có F1-score = 0.8235 (so với 0.80645), chứng tỏ EDA + WordNet giúp tăng tính khái quát cho mô hình đơn giản.
  - **LSTM + random embedding** cải thiện nhẹ và ổn định (F1 = 0.81434 so với 0.7949).
  - **DLEnsemble + random embedding** có recall rất cao (0.9205), nhưng precision thấp dẫn đến F1 = 0.8035 — phản ánh mô hình vẫn dễ bị mất cân bằng giữa precision và recall.

– **RNN + random embedding** ít được cải thiện nhất, F1 chỉ đạt 0.76215.

Kết luận: Việc kết hợp nhiều kỹ thuật tăng cường dữ liệu (EDA, embedding, WordNet) giúp cải thiện rõ rệt hiệu suất các mô hình học sâu trong bài toán phân tích cảm xúc. Đặc biệt, các mô hình sử dụng embedding huấn luyện sẵn tận dụng được lợi ích tốt nhất từ dữ liệu mở rộng, tăng khả năng khái quát hóa và hiểu ngữ nghĩa sâu hơn.

4.3.2 Thuật toán học máy

Model	train_orig				train_orig_eda				train_orig_eda_embedding				train_orig_eda_embedding_wordnet			
	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1
Decision Tree	0.7259	0.7314	0.7297	0.7306	0.7362	0.7432	0.7365	0.7398	0.7294	0.7241	0.7568	0.7401	0.7248	0.7308	0.7275	0.7291
Random Forest	0.7913	0.7964	0.7928	0.7946	0.7787	0.7770	0.7928	0.7848	0.7775	0.7803	0.7838	0.7820	0.7867	0.7945	0.7838	0.7891
SVM	0.7993	0.7808	0.8423	**0.8104**	0.7947	0.7677	**0.8559**	0.8094	0.7924	0.7678	0.8491	0.8064	0.7833	0.7607	0.8387	0.7974
KNN	0.5791	0.6149	0.4640	0.5249	0.5975	0.6325	0.5000	0.5585	0.5917	0.6189	0.5158	0.5627	0.6273	0.6686	0.5315	0.5922
Logistic Regression	0.7890	0.7579	0.8604	0.8059	0.7993	0.7717	**0.8604**	**0.8136**	0.7959	0.7692	0.8559	**0.8102**	0.7959	0.7737	0.8468	**0.8086**
Multinomial NB	0.7833	0.7495	**0.8626**	0.8021	0.7775	0.7451	0.8559	0.7966	0.7821	0.7520	0.8536	0.7996	0.7810	0.7535	0.8468	0.7975
Bernoulli NB	0.7947	0.7766	0.8378	**0.8061**	0.7959	0.7451	0.8559	0.7966	0.7901	0.7771	0.8243	0.8000	0.7936	0.7797	0.8288	0.8035

- **Nhìn chung:**
  - Các mô hình có hiệu suất tốt nhất về F1-score trên toàn bộ dữ liệu là **Logistic Regression**, **SVM**, và **Bernoulli Naive Bayes**.
  - **Logistic Regression** duy trì hiệu suất cao và ổn định trên mọi bộ dữ liệu, đặc biệt F1-score luôn quanh mức **0.81**.
  - **SVM** có recall cao nhất (đến 0.8559 với EDA), tuy nhiên precision thấp hơn các mô hình khác, dẫn đến F1 giảm nhẹ khi thêm embedding/wordnet.
- **Hiệu ứng tăng cường dữ liệu:**
  - Đa số mô hình không cải thiện rõ ràng khi áp dụng embedding hoặc WordNet – ví dụ như **Random Forest** và **Decision Tree** không tăng đáng kể.
  - Một số mô hình bị **suy giảm nhẹ** khi có thêm embedding (như SVM và KNN), có thể do tăng cường làm tăng nhiễu cho mô hình không học sâu.
  - Các mô hình như **Naive Bayes** lại hoạt động ổn định hơn khi có dữ liệu tăng cường – điều này hợp lý vì mô hình này phụ thuộc phân phối xác suất từ dữ liệu đầu vào.
- **So sánh theo từng metric:**
  - **Precision cao nhất:** Random Forest (0.7964 gốc), Logistic Regression (0.7737 sau tăng cường).
  - **Recall cao nhất:** Logistic Regression (0.8604), Multinomial NB (0.8626).

Kết luận: Tăng cường dữ liệu có lợi với một số mô hình (như Logistic Regression), nhưng không phải tất cả. Việc chọn mô hình phù hợp với phương pháp tăng cường là rất quan trọng để đạt hiệu quả tối ưu.

5 Kết luận

Thông qua quá trình thực hiện bài tập lớn về Sentiment Analysis, nhóm chúng em đã thu được những kinh nghiệm quý báu trong việc tiếp cận và giải quyết một bài toán phân loại đa lớp đầy thú vị và mang tính ứng dụng cao. Chúng em đã dành thời gian để nghiên cứu sâu về bài toán, khám phá các phương pháp học máy và học sâu tiềm năng, và đặc biệt là học cách hiện thực hóa những kiến thức lý thuyết thông qua các công cụ và thư viện khác nhau.

Quá trình thực nghiệm đã cho phép chúng em đánh giá một cách khách quan hiệu quả của các mô hình khác nhau trong việc phân loại sắc thái cảm xúc của văn bản. Những kết quả thu được không chỉ củng cố sự hiểu biết của chúng em về ưu và nhược điểm của từng phương pháp, mà còn giúp chúng em nhận ra tầm quan trọng của việc lựa chọn mô hình phù hợp với đặc điểm của dữ liệu và yêu cầu cụ thể của bài toán.

Mặc dù đã đạt được những kết quả nhất định, chúng em cũng không tránh khỏi những khó khăn trong quá trình thực hiện. Thách thức mà nhóm gặp phải chủ yếu nằm ở giai đoạn tiền xử lý dữ liệu, một bước quan trọng có ảnh hưởng trực tiếp đến hiệu suất của các mô hình.

Và để hoàn thành được báo cáo này, nhóm chúng em xin gửi lời cảm ơn sâu sắc đến PGS. TS. Thân Quang Khoát vì những tiết dạy tuyệt vời và đáng quý của thầy ở môn "Nhập môn Học máy và Khai phá dữ liệu".

6 Tài liệu tham khảo

[1] Ethem Alpaydin. Introduction to machine learning. MIT press, 2015.

[2] Jiawei Han, Jian Pei, and Micheline Kamber. Data mining: concepts and techniques. Elsevier, 2011.

[3] Activation functions in neural network. https://studymachinelearning.com/activation-functions-in-neural-network

[4] Jeremy Jordan. Setting the learning rate of your neural network. https://www.jeremyjordan.me/learning-rate

[5] Stanford CS 224N - Natural Language Processing https://web.stanford.edu/class/cs224n/

[6] Zhang, L., Wang, S., & Liu, B. Deep learning for sentiment analysis: A survey. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery.

[7] Saif, H., Fernandez, M., Fernandez, V., & Cantador. Contextual sentiment analysis of social media posts. IEEE Transactions on Knowledge and Data Engineering.

[8] Liu B.. Sentiment analysis and opinion mining. Synthesis lectures on human language technologies.

[9] Marcus, M., Santorini, B. and Marcinkiewicz, M.A., 1993. Building a large annotated corpus of English: The Penn Treebank. Computational linguistics, 19(2), pp.313-330.