TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

THÁI GIA BẢO - 52000014
NGUYỄN HOÀNG LONG - 51800577
NGUYỄN DUY KHÁNH - 51802085

BÁO CÁO GIỮA KỲ

NHẬP MÔN
XỬ LÝ NGÔN NGỮ TỰ NHIÊN

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

THÁI GIA BẢO - 52000014
NGUYỄN HOÀNG LONG - 51800577
NGUYỄN DUY KHÁNH - 51802085

BÁO CÁO GIỮA KỲ
NHẬP MÔN
XỬ LÝ NGÔN NGỮ TỰ NHIÊN

Người hướng dẫn
PGS.TS. Lê Anh Cường

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

LỜI CẢM ƠN

Đầu tiên, chúng em xin gửi lời cảm ơn chân thành đến Thầy Lê Anh Cường – Giảng viên khoa Công nghệ thông tin – Trường đại học Tôn Đức Thắng, đã hỗ trợ và giúp đỡ nhiệt tình trong quá trình thực hiện Dự án này.

Chúng em trân trọng cảm ơn Thầy Cô giảng viên Trường đại học Tôn Đức Thắng nói chung cũng như Thầy Cô giảng viên khoa Công nghệ thông tin nói riêng đã giảng dạy và truyền đạt nhiều kinh nghiệm quý trong suốt quá trình học tập tại trường.

Cuối cùng, xin cám ơn gia đình, bạn bè đã luôn động viên và đồng hành trong quá trình học tập cũng như quá trình thực hiện Dự án này.

Mặc dù rất cẩn thận trong quá trình thực hiện đồ án cũng như viết báo cáo nhưng cũng không tránh khỏi những thiếu sót. Chúng em rất mong nhận được sự góp ý từ các Thầy/Cô để đồ án được hoàn thiện hơn.

Xin chân thành cám ơn!

TP. Hồ Chí Minh, ngày 4 tháng 4 năm 2024

Tác giả

(Ký tên và ghi rõ họ tên)

Thái Gia Bảo

Nguyễn Hoàng Long

Nguyễn Duy Khánh

CÔNG TRÌNH ĐƯỢC HOÀN THÀNH
TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của PGS.TS. Lê Anh Cường. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong Dự án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung Dự án của mình. Trường Đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 4 tháng 4 năm 2024

Tác giả

(Ký tên và ghi rõ họ tên)

Thái Gia Bảo

Nguyễn Hoàng Long

Nguyễn Duy Khánh

TÓM TẮT

Báo cáo bắt đầu bằng cách giới thiệu lý do và mục tiêu lựa chọn chủ đề tài chính, sau đó đi sâu vào cơ sở lý thuyết với công việc tìm hiểu và so sánh các phương pháp Word2Vec như CBOW, Skip-gram và GloVe. Tiếp theo, báo cáo trình bày cách áp dụng các phương pháp này để thực hiện bài toán, bao gồm cài đặt thư viện, tiền xử lý dữ liệu, huấn luyện mô hình và vectơ tính toán biểu diễn trung bình của các câu hỏi. Sau đó, báo cáo đề cập đến công việc tìm câu tương tự nhất với một câu đầu vào, sử dụng cả phương pháp nhúng Word2Vec và GloVe. Cuối cùng, báo cáo tập trung vào việc xây dựng một văn bản phát hiện mô hình được gán nhãn, một ứng dụng thực tế của NLP và thử nghiệm mô hình để đánh giá hiệu suất.

Tổng thể, báo cáo này nỗ lực cung cấp một cái nhìn tổng quan về các phương pháp và công nghệ trong lĩnh vực NLP, đồng thời cung cấp một hướng dẫn thực hiện để xây dựng mô hình và ứng dụng chúng vào các công cụ toán toán các.

MỤC LỤC

Contents

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

DANH MỤC CÁC CHỮ VIẾT TẮT

NLP	Natural Language Processing
CBOW	Continuous Bag of Words
GloVe	Global Vectors for Word Representation
SVM	Support Vector Machine
MLP	Multilayer Perceptron

CHƯƠNG 1. MỞ ĐẦU VÀ TỔNG QUAN ĐỀ TÀI

1.1 Lý do chọn đề tài

Trong một thế giới mạng ngày càng phát triển, việc giám sát và lọc nội dung văn bản đóng vai trò quan trọng trong việc bảo vệ người dùng khỏi các loại thông điệp gây hại như quấy rối, lạm dụng, hoặc nội dung độc hại. Bằng cách nhận dạng và loại bỏ các nội dung không phù hợp, chúng ta tạo ra một môi trường trực tuyến lành mạnh và an toàn, nơi mà người dùng có thể tương tác và chia sẻ thông tin mà không phải lo lắng về sự xâm phạm hoặc tác động tiêu cực đến tâm lý của họ. Đồng thời, việc nhận dạng nội dung văn bản cũng đóng vai trò quan trọng trong việc bảo vệ danh tiếng và uy tín của các tổ chức và doanh nghiệp trực tuyến, giúp duy trì một hình ảnh tích cực trong cộng đồng mạng và thu hút sự tin tưởng từ khách hàng.

1.2 Mục tiêu thực hiện đề tài

Hiểu rõ về các phương pháp Word Embedding và Phân loại văn bản: Bằng cách nghiên cứu và áp dụng các phương pháp như CBOW, Skip-gram, FastText và các mô hình phân loại văn bản khác, chúng ta có cơ hội hiểu sâu hơn về cách máy tính hiểu và xử lý ngôn ngữ tự nhiên, từ việc biểu diễn từ vựng đến việc phân loại nội dung văn bản.

Xây dựng kỹ năng thực hành: Bằng cách thực hiện các bài tập và dự án liên quan đến Word Embedding và Phân loại văn bản, chúng ta có cơ hội áp dụng kiến thức lý thuyết vào thực tế và phát triển kỹ năng thực hành trong việc xử lý dữ liệu văn bản và xây dựng các mô hình NLP.

Góp phần vào sự phát triển của môi trường trực tuyến an toàn: Bằng việc nhận dạng và loại bỏ các nội dung không phù hợp, chúng ta góp phần vào việc tạo ra một môi trường trực tuyến an toàn và tích cực cho người dùng, giúp bảo vệ họ khỏi các loại nội dung gây hại và tạo điều kiện cho sự tương tác và chia sẻ thông tin tích cực trên mạng.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT VÀ THỰC NGHIỆM

2.1 Câu 1.1: Tìm hiểu các phương pháp Word2Vec

2.1.1 CBOW

CBOW (Continuous Bag of Words) là một phương pháp được sử dụng trong xử lý ngôn ngữ tự nhiên (NLP) để tạo ra các vectơ biểu diễn từ. Đây là một loại mạng nơ-ron nhận một chuỗi các từ làm đầu vào và đầu ra là một biểu diễn vectơ cho mỗi từ. Mục tiêu của CBOW là học một ánh xạ giữa các từ trong từ vựa và các vectơ tương ứng của chúng, sao cho các từ tương tự được ánh xạ vào các điểm gần nhau trong không gian vectơ.

Ý tưởng cơ bản đằng sau CBOW là dự đoán từ mục tiêu dựa trên bối cảnh của nó. Cho một chuỗi các từ, mô hình cố gắng dự đoán từ tiếp theo trong chuỗi. Mô hình được huấn luyện bằng cách sử dụng một tập dữ liệu văn bản lớn, và mục tiêu là làm giảm sự khác biệt giữa từ được dự đoán và từ thực tế.

Kiến trúc của một mô hình CBOW bao gồm một lớp đầu vào, một lớp ẩn và một lớp đầu ra. Lớp đầu vào nhận vào một chuỗi các từ, và lớp ẩn xử lý các từ đầu vào bằng một mạng nơ-ron. Lớp đầu ra tạo ra một biểu diễn vectơ cho mỗi từ trong từ vựa.

Quá trình huấn luyện cho CBOW bao gồm đưa chuỗi đầu vào vào mô hình, và điều chỉnh các trọng số và ngưỡng của mô hình để làm giảm sự khác biệt giữa từ được dự đoán và từ thực tế. Điều này được thực hiện bằng cách sử dụng backpropagation, một phương pháp được sử dụng trong mạng nơ-ron để tối ưu hóa các tham số của mô hình.

Một ưu điểm của CBOW là nó có thể bắt được các phụ thuộc xa giữa các từ trong một chuỗi. Không giống như các phương pháp khác như skip-gram, chỉ xem xét ngữ cảnh ngay lập tức của một từ, CBOW có thể bắt được các mối quan hệ giữa các từ xa trong một chuỗi. Điều này làm cho nó đặc biệt hữu ích cho các nhiệm vụ như mô hình ngôn ngữ và phân loại văn bản.

Tuy nhiên, CBOW cũng có một số hạn chế. Một hạn chế là nó yêu cầu một lượng lớn dữ liệu huấn luyện để học ánh xạ giữa các từ và các vectơ của chúng. Ngoài ra, CBOW có thể tốn nhiều tài nguyên tính toán để huấn luyện, đặc biệt là đối với các từ vựa lớn. Cuối cùng, CBOW có thể không hoạt động tốt trên các từ không nằm trong từ vựa, đó là các từ không có trong dữ liệu huấn luyện.

Nhìn chung, CBOW là một phương pháp mạnh mẽ để tạo ra các vectơ biểu diễn từ, nhưng nó yêu cầu việc điều chỉnh cẩn thận các siêu tham số và lượng lớn dữ liệu huấn luyện để đạt được hiệu suất tốt.

Ý nghĩa:

- CBOW (Continuous Bag of Words) là một phương pháp trong Word2Vec sử dụng để tạo biểu diễn từ vựng dựa trên ngữ cảnh từ trong một câu.
- CBOW cố gắng dự đoán từ cụ thể trong một câu dựa trên các từ xung quanh nó.

Mô hình

- Trong mô hình CBOW, một tầng ẩn được sử dụng để biểu diễn các từ trong câu.
- Đầu vào của CBOW là một tập hợp các từ ngữ cảnh, và mục tiêu là dự đoán từ mục tiêu.
- Để làm điều này, CBOW sử dụng các vector one-hot encoding của các từ ngữ cảnh (được tổng hợp lại) làm đầu vào cho một mạng neural network. Tầng ẩn trong mạng này biểu diễn các vector từ của các từ ngữ cảnh.

Huấn luyện mô hình

- Quá trình huấn luyện CBOW bắt đầu bằng việc chia dữ liệu thành các cặp từ ngữ cảnh và từ mục tiêu.
- Mục tiêu của quá trình huấn luyện là tối ưu hóa mô hình sao cho khả năng dự đoán từ mục tiêu dựa trên từ ngữ cảnh được cải thiện.

- Thuật toán tối ưu hóa như stochastic gradient descent (SGD) hoặc các biến thể của nó được sử dụng để điều chỉnh các trọng số của mạng neural network sao cho hàm mất mát được giảm thiểu.

CBOW (Continuous Bag of Words) là một phương pháp để học các vectơ biểu diễn từ, đại diện cho từng từ dưới dạng các vectơ dày đặc bắt được ý nghĩa ngữ nghĩa của chúng. Ý tưởng cơ bản đằng sau CBOW là dự đoán từ ngữ cảnh dựa trên từ mục tiêu và các từ ngữ cảnh xung quanh.

Quá trình học các vectơ biểu diễn từ sử dụng CBOW có thể được phân thành ba bước:

1. Tiền xử lý: Bước này bao gồm việc tách từ trong dữ liệu văn bản, loại bỏ các từ dừng, cắt gốc hoặc đưa về dạng gốc, và chuyển đổi chúng thành chữ thường.

2. Huấn luyện: Trong bước này, chúng ta huấn luyện một mô hình mạng nơ-ron trên dữ liệu văn bản đã được tiền xử lý bằng cách sử dụng hàm mục tiêu CBOW. Hàm mục tiêu này nhận từ mục tiêu và các từ ngữ cảnh xung quanh làm đầu vào, và đầu ra là một vectơ xác suất đại diện cho xác suất của mỗi từ ngữ cảnh khi biết từ mục tiêu.

3. Đánh giá: Sau khi huấn luyện mô hình, chúng ta đánh giá hiệu suất của nó trên một tập dữ liệu thử để xem nó có thể tổng quát hóa tốt đối với dữ liệu mới, chưa nhìn thấy không.

Hàm mục tiêu CBOW được định nghĩa như sau:

- Giả sử chúng ta có một câu như "The quick brown fox jumps over the lazy dog." Chúng ta muốn dự đoán từ ngữ cảnh cho từ mục tiêu và các từ ngữ cảnh xung quanh. Hàm mục tiêu CBOW nhận từ mục tiêu và các từ ngữ cảnh xung quanh làm đầu vào, và đầu ra là một vectơ xác suất đại diện cho xác suất của mỗi từ ngữ cảnh khi biết từ mục tiêu.

Ví dụ, nếu chúng ta muốn dự đoán từ ngữ cảnh cho "fox" trong câu trên, chúng ta sẽ sử dụng các đầu vào sau:

* Từ mục tiêu: "fox"

* Các từ ngữ cảnh: ["quick", "brown", "jumps", "over", "lazy", "dog"]

Hàm mục tiêu CBOW sau đó sẽ đầu ra một vectơ xác suất đại diện cho xác suất của mỗi từ ngữ cảnh khi biết từ mục tiêu. Ví dụ, nếu chúng ta giả định rằng mô hình được huấn luyện trên một tập dữ liệu văn bản lớn, nó có thể đầu ra các xác suất sau:

* Xác suất của "quick" khi biết "fox": 0.25
* Xác suất của "brown" khi biết "fox": 0.30
* Xác suất của "jumps" khi biết "fox": 0.40
* Xác suất của "over" khi biết "fox": 0.15
* Xác suất của "lazy" khi biết "fox": 0.20
* Xác suất của "dog" khi biết "fox": 0.35

Các xác suất này sau đó có thể được sử dụng để tính toán vectơ biểu diễn cho "fox". Vectơ biểu diễn là một vectơ dày đặc bắt các ý nghĩa ngữ nghĩa của từ mục tiêu, và nó được học bằng cách tối đa hóa xác suất của các từ ngữ cảnh khi biết từ mục tiêu.

2.1.2 Skip-gram

Skip-gram là một phương pháp trong ngữ cảnh của các vectơ biểu diễn từ, đại diện cho các từ dưới dạng các vectơ dày đặc trong không gian đa chiều. Ý tưởng đằng sau skip-gram là dự đoán từ ngữ cảnh dựa trên từ mục tiêu, và ngược lại. Điều này được thực hiện bằng cách huấn luyện một mạng nơ-ron để dự đoán từ còn thiếu trong một câu dựa trên các từ đã biết trong câu.

Cách tiếp cận chính của skip-gram là ý thức chính là ý nghĩa của một từ không chỉ được xác định bởi các đặc điểm cá nhân của nó, mà còn bởi ngữ cảnh mà nó xuất hiện. Bằng cách học mối quan hệ giữa các từ và ngữ cảnh của chúng, skip-gram có thể nắm bắt được sự tinh tế của ngôn ngữ và tạo ra các vectơ biểu diễn từ chính xác hơn.

Mô hình skip-gram bao gồm hai thành phần chính: một bộ mã hóa và một bộ giải mã. Bộ mã hóa nhận vào một câu và đầu ra là một biểu diễn vectơ của mỗi từ trong câu. Bộ giải mã sau đó nhận biểu diễn vectơ này làm đầu vào và dự đoán từ còn thiếu trong câu.

Trong quá trình huấn luyện, mô hình được huấn luyện để làm giảm sự khác biệt giữa từ được dự đoán và từ thực tế. Điều này được thực hiện bằng cách sử dụng một hàm mất mát như hàm mất mát cross-entropy. Mô hình cũng được huấn luyện để khuyến khích việc dự đoán các từ tương tự với từ mục tiêu và để ngăn chặn việc dự đoán các từ không tương tự.

Skip-gram đã được chứng minh là hiệu quả trong việc nắm bắt ý nghĩa của các từ và cụm từ một cách mà các phương pháp khác như bag-of-words không thể. Nó cũng hiệu quả hơn các phương pháp khác, vì nó chỉ yêu cầu một lần duy nhất đi qua dữ liệu huấn luyện để học các vectơ biểu diễn từ.

Tổng quan, skip-gram là một phương pháp mạnh mẽ để học các vectơ biểu diễn từ đã được sử dụng rộng rãi trong các nhiệm vụ xử lý ngôn ngữ tự nhiên như mô hình ngôn ngữ, phân loại văn bản và dịch máy.

Ý nghĩa:

- Skip-gram là một phương pháp trong Word2Vec sử dụng để tạo biểu diễn từ vựng dựa trên từ mục tiêu và các từ ngữ cảnh xung quanh nó trong một câu.
- Mục tiêu của Skip-gram là dự đoán các từ ngữ cảnh từ một từ mục tiêu cụ thể.

Mô hình:

- Mô hình Skip-gram cũng sử dụng một tầng ẩn để biểu diễn các từ trong câu.
- Tuy nhiên, trong Skip-gram, từ mục tiêu được sử dụng để dự đoán các từ ngữ cảnh.

- Cụ thể, từ đầu vào của Skip-gram là một từ mục tiêu, và mục tiêu của mô hình là dự đoán các từ ngữ cảnh xung quanh từ mục tiêu đó.

Huấn luyện mô hình:

- Quá trình huấn luyện của Skip-gram được thực hiện bằng cách sử dụng dữ liệu huấn luyện để tối ưu hóa mô hình sao cho khả năng dự đoán các từ ngữ cảnh từ một từ mục tiêu được cải thiện.
- Tương tự như CBOW, thuật toán tối ưu hóa như stochastic gradient descent (SGD) hoặc các biến thể của nó được sử dụng để điều chỉnh các trọng số của mạng neural network sao cho hàm mất mát được giảm thiểu.

Skip-gram là một phương pháp khác để học các vectơ biểu diễn từ, tương tự như CBOW nhưng có một số sự khác biệt trong hàm mục tiêu và quá trình huấn luyện. Ý tưởng cơ bản đằng sau Skip-gram là dự đoán từ mục tiêu dựa trên các từ ngữ cảnh xung quanh, thay vì dự đoán các từ ngữ cảnh dựa trên từ mục tiêu.

Quá trình học các vectơ biểu diễn từ sử dụng Skip-gram có thể được chia thành ba bước:

1. Tiền xử lý: Bước này bao gồm việc tách từ trong dữ liệu văn bản, loại bỏ các từ dừng, cắt gốc hoặc đưa về dạng gốc, và chuyển đổi chúng thành chữ thường.

2. Huấn luyện: Trong bước này, chúng ta huấn luyện một mô hình mạng nơ-ron trên dữ liệu văn bản đã được tiền xử lý bằng cách sử dụng hàm mục tiêu Skip-gram. Hàm mục tiêu này nhận từ mục tiêu và các từ ngữ cảnh xung quanh làm đầu vào, và đầu ra là một vectơ xác suất đại diện cho xác suất của mỗi từ mục tiêu khi biết các từ ngữ cảnh.

3. Đánh giá: Sau khi huấn luyện mô hình, chúng ta đánh giá hiệu suất của nó trên một tập dữ liệu thử để xem nó có thể tổng quát hóa tốt đối với dữ liệu mới, chưa nhìn thấy không.

Hàm mục tiêu Skip-gram được định nghĩa như sau:

Giả sử chúng ta có một câu như "The quick brown fox jumps over the lazy dog." Chúng ta muốn dự đoán từ mục tiêu dựa trên các từ ngữ cảnh xung quanh. Hàm mục tiêu Skip-gram nhận từ mục tiêu và các từ ngữ cảnh xung quanh làm đầu vào, và đầu ra là một vectơ xác suất đại diện cho xác suất của mỗi từ mục tiêu khi biết các từ ngữ cảnh.

Ví dụ, nếu chúng ta muốn dự đoán từ mục tiêu cho "fox" trong câu trên, chúng ta sẽ sử dụng các đầu vào sau:

* Từ mục tiêu: "fox"

* Các từ ngữ cảnh: ["quick", "brown", "jumps", "over", "lazy", "dog"]

Hàm mục tiêu Skip-gram sau đó sẽ đầu ra một vectơ xác suất đại diện cho xác suất của mỗi từ mục tiêu khi biết các từ ngữ cảnh. Ví dụ, nếu chúng ta giả định rằng mô hình được huấn luyện trên một tập dữ liệu văn bản lớn, nó có thể đầu ra các xác suất sau:

* Xác suất của "fox" khi biết ["quick", "brown", "jumps", "over", "lazy", "dog"]: 0.35

* Xác suất của "dog" khi biết ["quick", "brown", "jumps", "over", "lazy", "fox"]: 0.20

* Xác suất của "lazy" khi biết ["quick", "brown", "jumps", "over", "fox", "dog"]: 0.15

* Xác suất của "jumps" khi biết ["quick", "brown", "fox", "over", "lazy", "dog"]: 0.40

* Xác suất của "over" khi biết ["quick", "brown", "fox", "lazy", "dog", "jumps"]: 0.15

* Xác suất của "quick" khi biết ["brown", "fox", "jumps", "over", "lazy", "dog"]: 0.25

Các xác suất này sau đó có thể được sử dụng để tính toán các vectơ biểu diễn từ mục tiêu. Vectơ biểu diễn là một vectơ dày đặc bắt các ý nghĩa ngữ nghĩa của từ mục tiêu, và nó được học bằng cách tối đa hóa xác suất của các từ mục tiêu khi biết các từ ngữ cảnh.

Tóm lại, CBOW và Skip-gram là hai phương pháp khác nhau để học các vectơ biểu diễn từ, với CBOW dự đoán các từ ngữ cảnh dựa trên từ mục tiêu và Skip-gram dự đoán từ mục tiêu dựa trên các từ ngữ cảnh xung quanh. Cả hai phương pháp đều sử dụng một mô hình

2.1.3 Glove

Lý thuyết đằng sau GloVe dựa trên ý tưởng rằng các từ có thể được biểu diễn dưới dạng vectơ trong không gian nhiều chiều. Điều này cho phép tính toán hiệu quả và có thể mở rộng các điểm tương đồng của từ, điều này rất quan trọng đối với nhiều tác vụ xử lý ngôn ngữ tự nhiên. Sự khác biệt chính giữa GloVe và các thuật toán nhúng từ khác như Word2Vec là nó sử dụng kỹ thuật phân tích nhân tử ma trận để tìm ra cách biểu diễn từ theo chiều thấp tốt nhất, thay vì sử dụng mạng thần kinh nông để tìm hiểu cách biểu diễn.

Mô hình được sử dụng trong GloVe là thuật toán nhân tố ma trận, là một loại kỹ thuật giảm kích thước. Ý tưởng cơ bản đằng sau thuật toán này là tìm hai vectơ chiều thấp hơn có thể nắm bắt thông tin quan trọng nhất về dữ liệu chiều cao ban đầu. Trong trường hợp của GloVe, dữ liệu gốc là ma trận xuất hiện đồng thời, trong đó mỗi từ được biểu diễn dưới dạng một vectơ nắm bắt ngữ cảnh của nó trong tài liệu hoặc kho văn bản. Mục tiêu là tìm ra hai vectơ có chiều thấp hơn có thể nắm bắt được thông tin quan trọng nhất về các từ và ngữ cảnh của chúng.

Ý nghĩa:

- GloVe kết hợp thông tin cục bộ như Continuous Bag of Words (CBOW) và thông tin toàn cục như Skip-gram.
- Giúp cho GloVe có khả năng biểu diễn không chỉ vị trí từ trong văn bản mà còn biểu diễn được mối quan hệ giữa các từ.

Mô hình:

- GloVe xây dựng một ma trận thống kê co-occurrence của các từ, tức là ma trận thể hiện tần suất xuất hiện của các cặp từ trong cùng một ngữ cảnh.
- Cụ thể, giá trị tại hàng i, cột j của ma trận này thể hiện số lần từ j xuất hiện gần với từ i.

Huấn luyện mô hình:

- Quá trình huấn luyện của GloVe là quá trình tối ưu hóa ma trận thống kê co-occurrence để biểu diễn chính xác các quan hệ giữa các từ.
- Trong quá trình này, GloVe cố gắng tối ưu hóa một hàm mất mát đặc biệt được thiết kế để đảm bảo rằng các vectơ biểu diễn của các từ thể hiện mối quan hệ co-occurrence của chúng

Để huấn luyện mô hình GloVe, chúng ta cần cung cấp cho nó một tập dữ liệu lớn gồm các tài liệu văn bản hoặc kho văn bản. Sau đó, chúng tôi sử dụng dữ liệu này để tính toán ma trận sự xuất hiện, ma trận này ghi lại tần suất của mỗi từ trong kho văn bản. Bước tiếp theo là phân tích ma trận đồng xuất hiện bằng thuật toán phân tích nhân tử ma trận, chẳng hạn như phân tích hệ số ma trận không âm (NMF) hoặc phân tách giá trị số ít (SVD). Điều này tạo ra hai vectơ chiều thấp hơn giúp nắm bắt thông tin quan trọng nhất về các từ và ngữ cảnh của chúng. Cuối cùng, chúng ta có thể sử dụng các vectơ này để tính toán các từ tương tự, được sử dụng trong nhiều tác vụ xử lý ngôn ngữ tự nhiên.

Ưu điểm của việc sử dụng GloVe so với các thuật toán nhúng từ khác như Word2Vec bao gồm:

* Nó hiệu quả hơn và có khả năng mở rộng hơn Word2Vec vì nó không yêu cầu nhiều bộ nhớ hoặc tài nguyên tính toán.

* Nó có thể xử lý các từ nằm ngoài từ vựng, là những từ không có trong dữ liệu huấn luyện nhưng vẫn liên quan đến nhiệm vụ hiện tại.

* Nó nắm bắt ý nghĩa và ngữ cảnh của các từ chính xác hơn các thuật toán nhúng từ khác vì nó tính đến các kiểu xuất hiện đồng thời giữa các từ trong tài liệu hoặc kho văn bản.

Nhìn chung, GloVe là một công cụ mạnh mẽ để biểu diễn các từ dưới dạng vectơ trong không gian nhiều chiều, có thể được sử dụng để tính toán hiệu quả và có thể mở rộng các điểm tương đồng của từ cũng như các tác vụ xử lý ngôn ngữ tự nhiên khác.

2.2 Câu 1.2: Áp dụng để thực hiện bài toán

Hãy ứng dụng để thực hiện bài toán sau:
Cho môt tập các câu gọi là S = {s1, s2, …, sn}

Nhập vào 1 câu X, hãy tìm câu si nào đó trong tập S mà tương tự với X nhiều nhất

2.2.1 Cài đặt thư viện

```
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
from nltk.corpus import stopwords
import nltk
nltk.download('wordnet')
```

2.2.2 Tiền xử lý câu

```
      # Preprocess a sentence
      def preprocess_sentence(sentence):
          tokens = word_tokenize(sentence.lower())  # Tokenize and
lowercase
          stop_words = set(stopwords.words('english'))
          tokens = [word for word in tokens if word.isalpha() and word
not in stop_words]  # Remove stopwords and punctuation
          lemmatizer = WordNetLemmatizer()
          tokens = [lemmatizer.lemmatize(word) for word in tokens]  #
Lemmatize
          return tokens
```

Hàm preprocess_sentence(sentence) được sử dụng để tiền xử lý một câu.

Câu được chuyển đổi thành chữ thường và được tách thành các từ (tokenize) bằng hàm word_tokenize() từ thư viện NLTK.

Các từ dừng (stopwords) trong tiếng Anh được loại bỏ.

Các từ chỉ chứa ký tự chữ cái và không phải là stopwords được giữ lại.

Các từ được lemmatize bằng cách sử dụng WordNetLemmatizer từ thư viện NLTK.

2.2.3 Sử dụng list comprehersion để tiền xử lý tất cả các câu S

```
      # Preprocess all sentences in S
      preprocessed_S = [preprocess_sentence(sentence) for sentence in
S]
```

Tất cả các câu trong danh sách S được tiền xử lý bằng cách sử dụng hàm preprocess_sentence(sentence) và kết quả được lưu trữ trong danh sách preprocessed_S.

2.2.4 Train model

```
# Train Word2Vec model with CBOW
model_cbow = Word2Vec(preprocessed_S, vector_size=100, window=5, min_count=1, workers=4, sg=0)

# Train Word2Vec model with Skip-gram
model_skipgram = Word2Vec(preprocessed_S, vector_size=100, window=5, min_count=1, workers=4, sg=1)
```

Hai mô hình Word2Vec được huấn luyện: một với kiểu huấn luyện Continuous Bag of Words (CBOW) và một với kiểu huấn luyện Skip-gram.

Mô hình được huấn luyện trên các câu đã được tiền xử lý từ danh sách preprocessed_S.

Cả hai mô hình đều có các tham số như kích thước vector là 100, cửa sổ là 5, tần suất xuất hiện tối thiểu của một từ trong từ vựng là 1, số lượng công nhân (workers) là 4.

2.2.5 Tính toán vector biểu diễn trung bình của một câu

```
# Function to compute average word embedding of a sentence
def average_word_embedding(sentence_tokens, model):
    embeddings = []
    for token in sentence_tokens:
        if token in model.wv:
            embeddings.append(model.wv[token])
    if embeddings:
        return np.mean(embeddings, axis=0)
    else:
        return None
```

Hàm average_word_embedding(sentence_tokens, model) tính toán vectơ nhúng trung bình của một câu.

Các từ trong câu được biểu diễn bằng vectơ nhúng từ trong mô hình đã được huấn luyện.

Trung bình của các vectơ nhúng của các từ trong câu được tính toán và trả về.

2.2.6 Tìm câu giống nhất với một câu đầu vào đã cho

```
    # Function to find most similar sentence using a given model
    def most_similar_sentence_with_model(X, S, model):
        X_tokens = preprocess_sentence(X)
        X_embedding = average_word_embedding(X_tokens, model)
        if X_embedding is None:
            return None
        max_similarity = -1
        most_similar_sentence = None
        for sentence in S:
            sentence_tokens = preprocess_sentence(sentence)
            sentence_embedding                                    =
average_word_embedding(sentence_tokens, model)
            if sentence_embedding is not None:
                similarity    =    cosine_similarity([X_embedding],
[sentence_embedding])[0][0]
                if similarity > max_similarity:
                    max_similarity = similarity
                    most_similar_sentence = sentence
        return most_similar_sentence, max_similarity
```

Hàm most_similar_sentence_with_model(X, S, model) tìm câu giống nhất với câu đầu vào X trong danh sách các câu S sử dụng một mô hình nhúng từ đã cho.

Câu đầu vào X được tiền xử lý và biểu diễn dưới dạng vectơ nhúng trung bình.

Đối với mỗi câu trong danh sách S, câu đó cũng được tiền xử lý và biểu diễn dưới dạng vectơ nhúng trung bình.

Độ tương đồng (similarity) giữa câu đầu vào và mỗi câu trong S được tính bằng độ đo cosine similarity.

Câu có độ tương đồng cao nhất và điểm tương ứng với độ tương đồng đó được trả về.

2.2.7 Tìm câu giống nhất với câu X đầu vào

```
      # Given sentence X
      X = "The crane"

      # Find most similar sentence using CBOW model
      most_similar_cbow,           similarity_score_cbow           =
most_similar_sentence_with_model(X, S, model_cbow)

      # Find most similar sentence using Skip-gram model
      most_similar_skipgram,       similarity_score_skipgram       =
most_similar_sentence_with_model(X, S, model_skipgram)

      print("Most similar sentence with CBOW:", most_similar_cbow)
      print("Similarity score with CBOW:", similarity_score_cbow)

      print("\nMost    similar    sentence    with    Skip-gram:",
most_similar_skipgram)
      print("Similarity       score       with       Skip-gram:",
similarity_score_skipgram)
```

Một câu đầu vào X được cung cấp. Hàm most_similar_sentence_with_model được sử dụng để tìm câu giống nhất với câu X sử dụng cả hai mô hình CBOW và Skip-gram. Kết quả được in ra màn hình, bao gồm câu giống nhất và điểm tương đồng tương ứng cho mỗi mô hình.

2.2.8 Tải các vector biểu diễn từ tập GloVe đã huấn luyện

```
# Load pre-trained GloVe word embeddings manually
def load_glove_embeddings(file_path):
    word_vectors = {}
    with open(file_path, 'r', encoding='utf-8') as f:
        for line in f:
            values = line.split()
            word = values[0]
            vector = np.asarray(values[1:], dtype='float32')
            word_vectors[word] = vector
    return word_vectors

# Load pre-trained GloVe word embeddings
glove_file = './glove.6B.100d.txt'  # Change path to the location of
your GloVe file
word_vectors = load_glove_embeddings(glove_file)
```

Đoạn code làm việc tải các word embedding từ file GloVe. File GloVe được tải từ địa chỉ `glove_file` và được lưu trữ trong một dictionary có tên là `word_vectors`.

Hàm `load_glove_embeddings` nhận vào một đường dẫn đến file GloVe, và trả về một dictionary chứa các word embedding. Hàm này sử dụng `open()` để mở file GloVe với chế độ đọc (`'r'`), sau đó sử dụng `split()` để tách các giá trị trong file theo khoảng trắng.

Mỗi khi một dòng được đọc, hàm này sẽ lấy ra từ vựng (word) và vector embedding của nó bằng cách sử dụng `values[0]` và `np.asarray(values[1:], dtype='float32')`. Vector embedding được lưu trữ trong dictionary với tên là từ vựng đó.

Tất cả các word embedding đã được lưu trữ trong dictionary `word_vectors` sau khi hàm `load_glove_embeddings` hoàn thành.

2.2.9 Tính toán vector biểu diễn trung bình

```
    # Function to compute average word embedding of a sentence using
GloVe embeddings
def average_word_embedding(sentence_tokens):
    embeddings = []
    for token in sentence_tokens:
        if token in word_vectors:
            embeddings.append(word_vectors[token])
    if embeddings:
        return np.mean(embeddings, axis=0)
    else:
        return None
```

Hàm `average_word_embedding` nhận vào một danh sách các từ vựng (tokens) của một câu ngữ, và trả về vector embedding trung bình của các từ vựng đó. Hàm này sử dụng `word_vectors` để lấy ra vector embedding của mỗi từ vựng, sau đó trả về vector embedding trung bình của tất cả các từ vựng.

2.2.10 Tìm câu giống nhất với câu đầu vào X

Hàm `most_similar_sentence` nhận vào một câu ngữ `X`, một danh sách các câu ngữ `S`, và trả về câu ngữ tương tự nhất với `X` trong `S`, cùng với độ tương tự (similarity score) giữa hai câu ngữ. Hàm này sử dụng hàm `average_word_embedding` để lấy ra vector embedding của `X`, sau đó so sánh với từng câu ngữ trong `S` và lấy ra độ tương tự giữa hai câu ngữ.

```
    # Function to find most similar sentence using GloVe embeddings
    def most_similar_sentence(X, S):
        X_tokens = preprocess_sentence(X)
        X_embedding = average_word_embedding(X_tokens)
        if X_embedding is None:
            return None
        max_similarity = -1
        most_similar_sentence = None
        for i, sentence in enumerate(S):
            sentence_tokens = preprocess_sentence(sentence)
            sentence_embedding                                    =
average_word_embedding(sentence_tokens)
            if sentence_embedding is not None:
                similarity    =    cosine_similarity([X_embedding],
[sentence_embedding])[0][0]
                if similarity > max_similarity:
                    max_similarity = similarity
                    most_similar_sentence = sentence
        return most_similar_sentence, max_similarity
```

2.2.11 Chạy thuật toán

```
# Given sentence X
    X = "The crane lifted"

    # Find most similar sentence using GloVe embeddings
    most_similar, similarity_score = most_similar_sentence(X, S)
    print("Most similar sentence:", most_similar)
    print("Similarity score:", similarity_score)
```

Tất cả các câu ngữ đã được so sánh và đánh giá độ tương tự, sau đó hàm này trả về câu ngữ tương tự nhất với `X`, cùng với độ tương tự (similarity score) giữa hai câu ngữ.

2.3 Câu 2. Xây dựng mô hình phát hiện text được gán nhãn

2.3.1 Cài đặt thư viện Underthesea

```
pip install underthesea
```

Underthesea là một thư viện hỗ trợ cho việc nghiên cứu và phát triển xử lý ngôn ngữ tự nhiên tiếng Việt. Nó cung cấp các API cực kỳ dễ dàng để áp dụng các mô hình pretrained NLP cho văn bản tiếng Việt, chẳng hạn như phân đoạn từ, gắn thẻ một phần giọng nói(PoS), nhận dạng thực thể có tên (NER), phân loại văn bản và phân tích cú pháp phụ thuộc.

2.3.2 Import các thư viện khác

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import GradientBoostingClassifier
import string
import unicodedata
import re
import underthesea
from collections import Counter
```

2.3.3 Đọc và tiền xử lý dữ liệu

- Đọc dữ liệu từ file: `facebook_comment_2k7.csv`.

```
# Đọc dữ liệu từ file CSV
data = pd.read_csv("facebook_comment_2k7.csv")
```

```
# Tiền xử lý dữ liệu văn bản tiếng Việt
def preprocess_text_vietnamese(text):
    # Chuyển văn bản về chữ thường
    text = text.lower()
    # Loại bỏ dấu câu
    text = text.translate(str.maketrans("", "", string.punctuation))
    # Loại bỏ dấu ký tự đặc biệt
    text = re.sub(r'[^\w\s]', '', text)
    # Tách từ tiếng Việt
    words = underthesea.word_tokenize(text)
    # Ghép từ lại thành câu
    text = ' '.join(words)
    return text
```

- Sử dụng hàm lower để làm cho tất cả các chữ trong văn bản thành chữ thường.
- Sử dụng str.maketrans(“”, “”, string.punctuation) tạo ra bảng dịch để ánh xạ mỗi ký tự thành ‘None’ từ đó sẽ loại bỏ chúng từ chuỗi. Tiếp theo sử dụng text.translate loại bỏ tất cả các ký tự dấu câu dựa trên bảng dịch đã tạo trước đó.
- Sử dụng re.sub để loại bỏ các dấu ký tự đặc biệt
- Sử dụng underthesea.word_tokenize(text) để tách các từ trong văn bản
- Sử dụng hàm join để ghép các từ thành câu.

2.3.4 Khởi tạo và huấn luyện mô hình

- Sử dụng các model như: Logistic Regression, Naive Bayes, Random Forest, SVM, KNeighbors, MLP, Gradient Boosting để huấn luyện mô hình cho bài toán.

Train models: vòng lặp for duyệt qua mỗi cặp (name, model) trong models.items trong đó name là tên của mô hình và model là pipeline liên kết với mô hình đó. Model.fit(data[‘text’], data[‘label’]) dùng để điều chỉnh mô hình cho dữ liệu văn bản text tương ứng với label.

```
# Định nghĩa các mô hình học máy
models = {
    "Logistic Regression":
make_pipeline(TfidfVectorizer(preprocessor=preprocess_text_vietnames
e, max_features=10000), LogisticRegression()),
    "Naive Bayes":
make_pipeline(TfidfVectorizer(preprocessor=preprocess_text_vietnames
e, max_features=10000), MultinomialNB()),
    "Random Forest":
make_pipeline(TfidfVectorizer(preprocessor=preprocess_text_vietnames
e, max_features=10000), RandomForestClassifier()),
    "SVM":
make_pipeline(TfidfVectorizer(preprocessor=preprocess_text_vietnames
e, max_features=10000), SVC(kernel='linear')),
    "KNeighbors":
make_pipeline(TfidfVectorizer(preprocessor=preprocess_text_vietnames
e, max_features=10000), KNeighborsClassifier(n_neighbors=5)),
    "MLP":
make_pipeline(TfidfVectorizer(preprocessor=preprocess_text_vietnames
e, max_features=10000), MLPClassifier()),
    "Gradient Boosting":
make_pipeline(TfidfVectorizer(preprocessor=preprocess_text_vietnames
e, max_features=10000), GradientBoostingClassifier())
}
# Huấn luyện các mô hình
for name, model in models.items():
    model.fit(data['text'], data['label'])
```

2.3.4.1 Logistic Regression

Là 1 thuật toán phân loại được dùng để gán các đối tượng cho 1 tập hợp giá trị rời rạc (như 0, 1, 2, ...). Một ví dụ điển hình là phân loại Email, gồm có email công việc, email gia đình, email spam, phân loại cảm xúc, phân loại từ, dự đoán từ tiếp theo… Thuật toán trên dùng hàm sigmoid logistic để đưa ra đánh giá theo xác suất.

Công thức tính xác suất Logistic Regression:

$$P(y=1|x) = 1 / (1 + \exp(-(w^T * x + b)))$$

Trong đó:

- P(y=1|x) là xác suất của lớp tích cực
- x là vectơ đặc trưng đầu vào
- w là vectơ trọng số
- b là độ lệch (bias)
- exp() là hàm mũ với cơ số e

Hàm sigmoid là hàm liên tục và luôn đưa ra giá trị trong đoạn [0,1]

sigmoid(z) = 1 / (1 + exp(-z))

Trong đó:

- z là đầu vào của hàm sigmoid, có thể là một số thực bất kỳ hoặc một vectơ.
- exp(x) là hàm mũ với cơ số e (euler's number), có giá trị khoảng 2.71828.

Hàm sigmoid có một số đặc điểm quan trọng:

- Giá trị của hàm sigmoid luôn nằm trong khoảng từ 0 đến 1.
- Khi z tiến đến âm vô cùng, sigmoid tiến đến 0.
- Khi z tiến đến dương vô cùng, sigmoid tiến đến 1.
- Đối với z = 0, giá trị của sigmoid là 0.5, điều này thường được sử dụng như ngưỡng để quyết định lớp của một mẫu trong Logistic Regression.

2.3.4.2 Naïve Bayes

Là một thuật toán học máy phân loại dựa trên xác suất và phương pháp thống kê, được sử dụng rộng rãi trong các ứng dụng như phân loại văn bản, phát hiện thư rác, phân loại văn bản.

Đặc trưng đầu vào là các độc lập với nhau khi đã biết lớp dữ liệu. Mặc dù giả định này thường không đúng đối với các dữ liệu thực tế, nhưng trong nhiều trường hợp, thuật toán vẫn hoạt động tốt và có hiệu suất cao.

Công thức tính:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

Trong đó:

- P(y|X) gọi là posterior probability hay còn là xác suất của mục tiêu y với điều kiện đặc trưng X
- P(X|y) gọi là likelihood xác suất đặc trưng X khi đã biết mục tiêu y
- P(y) gọi là prior probability của mục tiêu y
- P(X) gọi là prior probability của đặc trưng X

X là vector các đặc trưng có thể viết dưới dạng X = (x_1, x_2, x_3…x_n)

Khi đó, đẳng thức Bayes trở thành:

$$P(y|x_1, ..., x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

Trong mô hình Naive Bayes, có hai giả thiết được đặt ra:

- Các đặc trưng đưa vào mô hình là độc lập với nhau. Tức là sự thay đổi giá trị của một đặc trưng không ảnh hưởng đến các đặc trưng còn lại.
- Các đặc trưng đưa vào mô hình có ảnh hưởng ngang nhau đối với đầu ra mục tiêu.

Khi đó, kết quả mục tiêu y để P(y|X) đạt cực đại trở thành:

$$y = argmax_y P(y) \prod_{i=1}^{n} P(x_i|y)$$

Chính vì hai giả thiết gần như không tồn tại trong thực tế trên, mô hình này mới được gọi là naive (ngây thơ). Tuy nhiên, chính sự đơn giản của nó với việc dự đoán rất nhanh kết quả đầu ra khiến nó được sử dụng rất nhiều trong thực tế trên những bộ dữ liệu lớn, đem lại kết quả khả quan. Một vài ứng dụng của Naive Bayes có thể kể đến như: lọc thư rác, phân loại văn bản, dự đoán sắc thái văn bản, …

2.3.4.3 Random Forest

Random Forest là một thuật toán học máy phổ biến được sử dụng cho các tác vụ phân loại và hồi quy. Nó là một phần của lớp các thuật toán gọi là ensemble learning, trong đó một số lượng lớn các mô hình được kết hợp lại để tạo ra một dự đoán cuối cùng. Random Forest là một loại ensemble learning dựa trên cấu trúc cây quyết định.

Ở bước huấn luyện thì mình sẽ xây dựng nhiều cây quyết định, các cây quyết định có thể khác nhau

- Xây dựng các cây quyết định (Decision Trees): Random Forest xây dựng một tập hợp các cây quyết định độc lập. Mỗi cây được xây dựng từ một mẫu ngẫu nhiên của dữ liệu huấn luyện và một phần của các đặc trưng.
- Phiên bản Bootstrap (Bootstrap Sampling): Mỗi cây được huấn luyện trên một tập con của dữ liệu huấn luyện được chọn ngẫu nhiên thông qua phương pháp Bootstrap sampling. Điều này đảm bảo rằng mỗi cây trong Random Forest được huấn luyện trên một tập dữ liệu khác nhau.
- Random Feature Selection: Khi xây dựng mỗi cây, chỉ một phần của tất cả các đặc trưng được sử dụng để chọn nút phân loại tốt nhất tại mỗi nút trong quá trình phát triển cây. Việc này giúp tăng tính đa dạng giữa các cây.

- Vận dụng Voting (Voting Process): Khi cần dự đoán lớp hoặc giá trị đầu ra cho một điểm dữ liệu mới, Random Forest tính toán dự đoán của từng cây và sau đó chọn lớp hoặc giá trị đầu ra được "bầu chọn" nhiều nhất (trong trường hợp phân loại) hoặc tính trung bình (trong trường hợp hồi quy).
- Ưu điểm của Random Forest bao gồm khả năng xử lý dữ liệu lớn, khả năng ứng phó với các đặc trưng nhiễu và các tương tác phức tạp giữa các đặc trưng, cũng như khả năng xác định độ quan trọng của các đặc trưng trong mô hình. Tuy nhiên, nhược điểm có thể bao gồm chi phí tính toán cao và khả năng overfitting đối với tập dữ liệu nhỏ

2.3.4.4 Support Vector Machine (SVM)

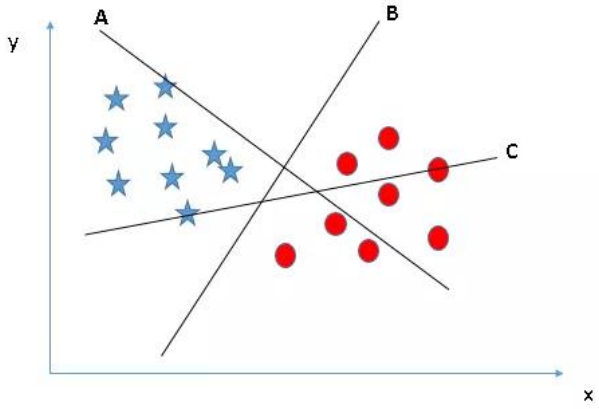
Là một thuật toán học máy phổ biến được sử dụng cho các tác vụ phân loại và hồi quy. SVM là một trong những thuật toán phân loại tuyến tính mạnh mẽ nhất và cũng có thể được mở rộng để xử lý các bài toán phi tuyến tính thông qua kỹ thuật "kernel trick".

Cơ bản, SVM tìm ra ranh giới phân chia (decision boundary) tốt nhất giữa các lớp dữ liệu bằng cách tìm ra siêu phẳng (hyperplane) phân chia lớp sao cho khoảng cách từ các điểm dữ liệu gần nhất của mỗi lớp đến siêu phẳng là lớn nhất. Các điểm dữ liệu này được gọi là các vector hỗ trợ (support vectors), từ đó xuất phát tên gọi "Support Vector Machine"

Cách tìm ra siêu phẳng (hyperplane) giữa các lớp:

Cách 1:

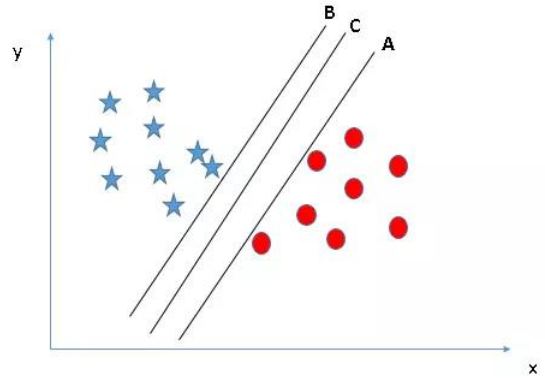
Ở đâu có 3 đường hyper-lane (A,B,C) vậy làm sao để chọn đúng cho 2 nhóm



Quy tắc số một để chọn 1 hyper-lane, chọn một hyper-lane để phân chia hai lớp tốt nhất. Trong ví dụ này chính là đường B.

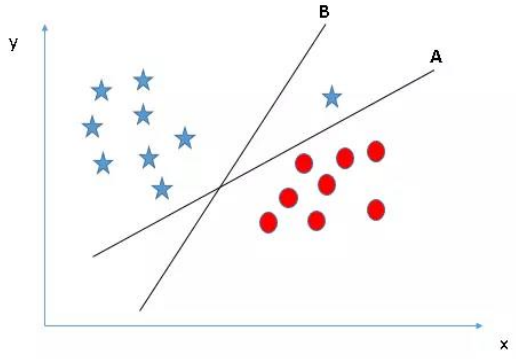
Cách 2:

Ở đây có 3 đường đều thỏa mãn theo quy tắc số một



Quy tắc thứ hai chính là xác định khoảng cách lớn nhất từ điểm gần nhất của một lớp nào đó đến đường hyper-lane. Khoảng cách này được gọi là "Margin", Hãy nhìn hình bên dưới, trong đấy có thể nhìn thấy khoảng cách margin lớn nhất đấy là đường C. Cần nhớ nếu chọn lầm hyper-lane có margin thấp hơn thì sau này khi dữ liệu tăng lên thì sẽ sinh ra nguy cơ cao về việc xác định nhầm lớp cho dữ liệu.

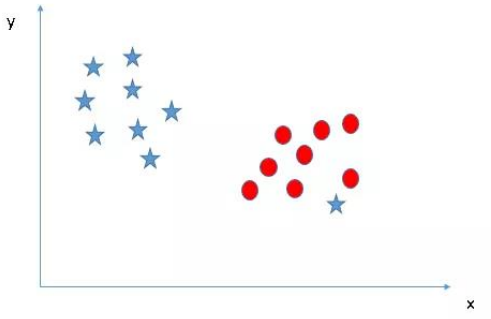
Cách 3:



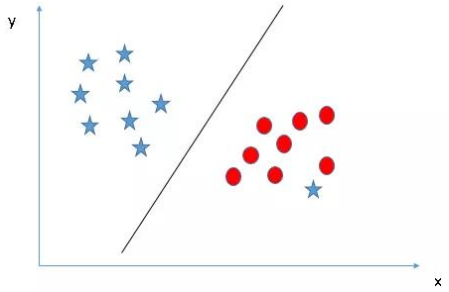
Có thể có một vài người sẽ chọn đường B bởi vì nó có margin cao hơn đường A, nhưng đấy sẽ không đúng bởi vì nguyên tắc đầu tiên sẽ là nguyên tắc số 1, chúng ta cần chọn hyper-lane để phân chia các lớp thành riêng biệt. Vì vậy đường A mới là lựa chọn chính xác.

Cách 4:

Tiếp theo hãy xem hình bên dưới, không thể chia thành hai lớp riêng biệt với 1 đường thẳng, để tạo 1 phần chỉ có các ngôi sao và một vùng chỉ chứa các điểm tròn.

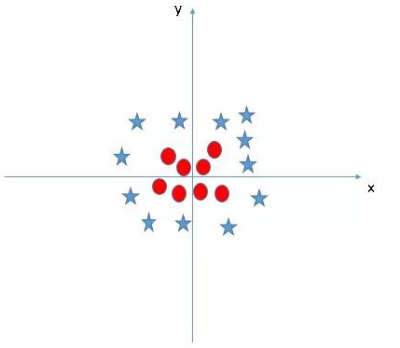


Ở đây sẽ chấp nhận, một ngôi sao ở bên ngoài cuối được xem như một ngôi sao phía ngoài hơn, SVM có tính năng cho phép bỏ qua các ngoại lệ và tìm ra hyper-lane có biên giới tối đa . Do đó có thể nói, SVM có khả năng mạnh trong việc chấp nhận ngoại lệ.

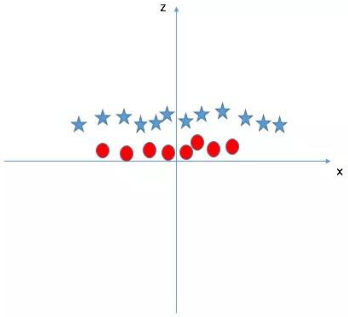


Cách 5:

Trong trường hợp dưới đây, không thể tìm ra 1 đường hyper-plane tương đối để chia các lớp, vậy làm thế nào để SVM phân tách dữ liệu thành hai lớp riêng biệt? Cho đến bây giờ chúng ta chỉ nhìn vào các đường tuyến tính hyperplane



SVM có thể giải quyết vấn đề này, Khá đơn giản, nó sẽ được giải quyết bằng việc thêm một tính năng, Ở đây chúng ta sẽ thêm tính năng z = x^2+ y^2. Bây giờ dữ liệu sẽ được biến đổi theo trục x và z như sau



Trong sơ đồ trên, các điểm cần xem xét là: • Tất cả dữ liệu trên trục z sẽ là số dương vì nó là tổng bình phương x và y • Trên biểu đồ các điểm tròn đỏ xuất hiện gần trục x và y hơn vì thế z sẽ nhỏ hơn => nằm gần trục x hơn trong đồ thị (z,x) Trong SVM, rất dễ dàng để có một siêu phẳng tuyến tính (linear hyperplane) để chia thành hai lớp, Nhưng một câu hỏi sẽ nảy sinh đấy là, chúng ta có cần phải thêm một tính năng phân chia này bằng tay hay không. Không, bởi vì SVM có một kỹ thuật được gọi là kernel trick (kỹ thuật hạt nhân), đây là tính năng có không gian đầu vào có chiều sâu thấm và biến đổi nó thành không gian có chiều cao hơn, tức là nó không phân chia các vấn đề thành các vấn đề riêng biệt, các tính năng này được gọi là kernel. Nói một cách đơn giản nó thực hiện một số biết đổi dữ liệu phức tạp, sau đó tìm ra quá trình tách dữ liệu dựa trên các nhãn hoặc đầu ra mà chúng ra đã xác định trước.

2.3.4.5 K-Nearest Neigbors (KNeighbors)

Là một thuật toán học máy đơn giản nhưng mạnh mẽ được sử dụng chủ yếu cho các vấn đề phân loại và hồi quy. Thuật toán này dựa trên ý tưởng rằng các điểm dữ liệu có cùng đặc điểm sẽ được gán vào cùng một lớp hoặc có cùng giá trị dự đoán

Cách hoạt động của KNeighbors như sau:

- Xây dựng mô hình: Trong quá trình huấn luyện, thuật toán chỉ đơn giản là lưu trữ toàn bộ dữ liệu huấn luyện.
- Dự đoán: Khi cần dự đoán lớp hoặc giá trị đầu ra cho một điểm dữ liệu mới, thuật toán tính toán khoảng cách giữa điểm dữ liệu mới và tất cả các điểm dữ liệu trong tập huấn luyện.
- Chọn K hàng xóm gần nhất: KNeighbors sau đó chọn ra K điểm dữ liệu gần nhất (kết quả của việc tính toán khoảng cách) và sử dụng thông tin từ chúng để dự đoán lớp hoặc giá trị đầu ra của điểm dữ liệu mới. Đối với phân loại, KNeighbors thường sử dụng phương pháp voting để quyết định lớp của điểm dữ liệu mới dựa trên lớp của các điểm hàng xóm. Đối với hồi quy, KNeighbors

thường sử dụng trung bình hoặc trung vị của các giá trị đầu ra của các điểm hàng xóm.

- Đánh giá kết quả: Sau khi dự đoán, thuật toán có thể được đánh giá bằng cách so sánh kết quả dự đoán với nhãn thực tế (đối với bài toán phân loại) hoặc giá trị thực tế (đối với bài toán hồi quy) của các điểm dữ liệu trong tập kiểm tra.

Ưu điểm của KNeighbors bao gồm tính đơn giản, dễ dàng triển khai và hiệu suất tốt trong một số trường hợp đặc biệt, đặc biệt là khi dữ liệu không gian thấp hoặc có cấu trúc đều đặn. Tuy nhiên, KNeighbors cũng có nhược điểm là tăng chi phí tính toán khi kích thước dữ liệu tăng lên và cần phải lưu trữ toàn bộ tập dữ liệu huấn luyện.

2.3.4.6 Multilayer Perceptron (MLP)

Là một mô hình mạng nơ-ron nhân tạo đa tầng, được sử dụng rộng rãi trong lĩnh vực học sâu (deep learning) cho các tác vụ như phân loại, hồi quy và dự đoán. MLP được xây dựng trên cơ sở của mạng nơ-ron nhân tạo đơn giản nhất, gồm các tầng (layers) liên kết với nhau bởi các trọng số (weights), và sử dụng hàm kích hoạt phi tuyến (non-linear activation function) để tạo ra sự phi tuyến tính và khả năng học được của mô hình.

Các thành phần chính của MLP bao gồm:

- Input Layer (Tầng Đầu Vào): Tầng này chứa các nút đại diện cho các biến đầu vào của mô hình. Số lượng nút trong tầng này phụ thuộc vào số lượng đặc trưng trong dữ liệu đầu vào.
- Hidden Layers (Tầng Ẩn): MLP có thể có một hoặc nhiều tầng ẩn. Mỗi tầng ẩn chứa một số lượng nút (neurons) được kết nối với các nút từ tầng trước và tầng sau nó. Các tầng ẩn giúp mô hình học được các biểu diễn phức tạp của dữ liệu thông qua việc học các tương tác phi tuyến giữa các đặc trưng.

- Output Layer (Tầng Đầu Ra): Tầng này chứa các nút đại diện cho các biến đầu ra của mô hình. Số lượng nút trong tầng này phụ thuộc vào số lượng lớp hoặc số lượng giá trị đầu ra trong bài toán.
- Weights (Trọng Số): Mỗi kết nối giữa các nút trong các tầng được điều chỉnh bằng các trọng số, đại diện cho mức độ quan trọng của mỗi kết nối trong quá trình học.
- Activation Function (Hàm Kích Hoạt): Hàm kích hoạt được áp dụng cho đầu ra của mỗi nút trong các tầng ẩn, tạo ra sự phi tuyến tính và khả năng học được của mô hình.
- Training Algorithm (Thuật Toán Huấn Luyện): MLP thường được huấn luyện thông qua các phương pháp như lan truyền ngược (backpropagation) kết hợp với các phương pháp tối ưu hóa như Gradient Descent để điều chỉnh trọng số và giảm thiểu hàm mất mát.

MLP là một mô hình linh hoạt có thể áp dụng cho nhiều loại dữ liệu và tác vụ khác nhau, nhưng đồng thời cũng đòi hỏi một lượng lớn dữ liệu huấn luyện và thời gian tính toán lớn khi có số lượng lớn các tham số.

2.3.4.7 Gradient Boosting

Là một kỹ thuật học máy thuộc lớp ensemble learning, trong đó nhiều mô hình yếu (weak learners) được kết hợp lại để tạo thành một mô hình mạnh hơn. Kỹ thuật này tập trung vào việc xây dựng các mô hình theo cách tuần tự, mỗi mô hình mới cố gắng cải thiện sai số của mô hình trước đó.

- Boosting là một quá trình tuần tự, không thể xử lí song song, do đó, thời gian train mô hình có thể tương đối lâu.
- Sau mỗi vòng lặp, Boosting có khả năng làm giảm error theo cấp số nhân.
- Boosting sẽ hoạt động tốt nếu base learner của nó không quá phức tạp cũng như error không thay đổi quá nhanh.
- Boosting giúp làm giảm giá trị bias cho các model base learner.

Gradient Boosting xây dựng thuật toán nhằm giải quyết bài toán tối ưu

$$\min_{c_n=1:N, w_n=1:N} L(y, \sum_{n=1}^{N} c_n w_n))$$

Trong đó:

- L: giá trị loss function
- y: label
- c_n : trọng số của weak learner lần thứ n
- w_n : weak learner thứ n

Thay vì cố gắng quét tìm tất cả các giá trị c_n , w_n để tìm nghiệm tối ưu toàn cục - một công việc tốn nhiều thời gian và tài nguyên, chúng ta sẽ cố gắng tìm các giá trị nghiệm cục bộ sau khi thêm mỗi một mô hình mới vào chuỗi mô hình với mong muốn dần đi đến nghiệm toàn cục.

Gradient Boosting thường cho hiệu suất tốt trên nhiều loại dữ liệu và có thể được sử dụng cho cả các vấn đề phân loại và hồi quy. Tuy nhiên, nó có thể đòi hỏi thời gian tính toán lớn và cần phải điều chỉnh một số siêu tham số quan trọng như số lượng mô hình yếu và học rate.

2.3.5 Khởi tạo hàm dự đoán mô hình

```
# Định nghĩa hàm dự đoán nhãn
def predict_labels(input_sentence):
    # Dự đoán nhãn của câu nhập vào bằng các mô hình đã huấn luyện
    predictions = {name: model.predict([input_sentence])[0] for
name, model in models.items()}

    # Đếm số lượng dự đoán của từng nhãn
    label_counter = Counter(predictions.values())

    # Lấy nhãn xuất hiện nhiều nhất (có ít nhất 4 lần dự đoán)
    most_common_labels = [label for label, count in
label_counter.items() if count >= 4]

    # Nếu không có nhãn nào được dự đoán nhiều lần, trả về "normal"
    if not most_common_labels:
        return ["normal"]
    return most_common_labels
```

Hàm này lặp qua từng mô hình trong các mô hình danh sách và dự đoán nhãn cho câu đầu vào bằng cách sử dụng từng mô hình.

Kết quả mong đợi của mỗi mô hình được lưu vào dự đoán từ điển với tên của mô hình là khóa.

Sau đó, hàm đếm số lượng dự kiến của từng nhãn bằng cách sử dụng Counter.

Hàm giữ lại các nhãn xuất hiện ít nhất 4 lần (biểu thị bởi biến nhất_common_labels) và trả về danh sách các nhãn đó.

Nếu không có nhãn nào được mong đợi nhiều lần (tức là Most_common_labels trống), hàm sẽ trả về danh sách chứa một phần tử là chuỗi "bình thường".

2.3.6 Thử nghiệm mô hình

```
# Nhập câu từ người dùng
input_sentence = input("Nhập câu: ")

# Dự đoán nhãn cho câu nhập vào
predicted_labels =
predict_labels(preprocess_text_vietnamese(input_sentence))
print("Dự đoán nhãn:", predicted_labels)
```

Kết quả thử nghiệm

```
Nhập câu: hôm nay tôi đi học
Dự đoán nhãn: ["['normal']"]
```

```
Nhập câu: chúng mày bị điên à
Dự đoán nhãn: ["['hate_speech']"]
```

```
Nhập câu: Shinhanbank há»— trá»£ khoá°£n vay theo lÆ°Æ¡ngðŸ‘‰Vay theo
bá°£o hiá»ƒm nhÃ¢n thá» ðŸ‘‰khoá°£n vay lÃªn
Dự đoán nhãn: ["['dangerous_content']"]
```

TÀI LIỆU THAM KHẢO

Tiếng Việt

Lê Minh, T., & Phạm Thị Ngọc, A. (2020). Xử lý ngôn ngữ tự nhiên. NXB Đại học Quốc gia Hà Nội.

Tiếng Anh

Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 1532–1543.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. In Advances in Neural Information Processing Systems (pp. 3111–3119)

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533–536.

Lewis, D. D. (1998). Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval. In European Conference on Machine Learning (pp. 4-15). Springer, Berlin, Heidelberg

Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5–32.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3), 273–297.