ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA TOÁN - TIN HỌC

ĐỒ ÁN CUỐI KỲ

Latent Semantic Analysis và Ứng dụng trong Information Retrieval

Môn học: Phương pháp số cho Khoa học dữ liệu
Giảng viên hướng dẫn: TS. Nguyễn Thị Hoài Thương
Nhóm: 6

STT	Họ và tên	MSSV	Mức độ hoàn thành (%)
1	Đặng Minh Phúc	22280064	100
2	Nguyễn Minh Hùng	21110301	100
3	Trương Quốc Trung	21110427	100
4	Lê Hồng Cát	21110249	100
5	Trương Minh Hoàng	22280034	100
6	Trần Nguyễn Trung Tuấn	22280101	100
7	Nguyễn Thuận Phát	22280062	100

Thành phố Hồ Chí Minh, tháng 07/2025

Lời nói đầu

Lời đầu tiên, nhóm tác giả xin chân thành cảm ơn Tiến sĩ Nguyễn Thị Hoài Thương vì đã giảng dạy học phần và dành thời gian góp ý cho đồ án của nhóm trong buổi thuyết trình. Những ý kiến phản biện và hướng dẫn chỉnh sửa từ cô đã giúp nhóm hoàn thiện báo cáo một cách rõ ràng và mạch lạc hơn. Qua quá trình này, nhóm đã học hỏi thêm nhiều kinh nghiệm quý báu trong việc trình bày và hoàn thiện sản phẩm học thuật, làm nền tảng cho các nghiên cứu sau này.

Trong khuôn khổ đồ án này, nhóm đã chủ động hiện thực hóa toàn bộ các phương pháp từ đầu, không sử dụng các thư viện có sẵn nhằm hiểu rõ hơn nguyên lý hoạt động cốt lõi của từng mô hình. Do không sử dụng các mô hình đã được huấn luyện trước (pretrained models), nhóm tập trung triển khai các phương pháp đơn giản nhưng hiệu quả, phù hợp với giới hạn về tài nguyên tính toán sẵn có. Vì lý do đó, nhóm không theo đuổi các hướng nghiên cứu yêu cầu năng lực xử lý cao như xử lý video hay các mô hình học sâu phức tạp sử dụng kiến trúc Transformer.

Mặc dù còn gặp nhiều hạn chế về tài nguyên tính toán, nhóm đã nỗ lực tiếp cận bài toán một cách có hệ thống, từ khâu khảo sát lý thuyết, triển khai và đánh giá thực nghiệm cho đến xây dựng một ứng dụng web có thể sử dụng trực tiếp bởi người dùng. Nhóm hy vọng rằng đồ án này không chỉ là kết quả của một quá trình học tập nghiêm túc, mà còn là tiền đề cho những nghiên cứu sâu hơn trong tương lai.

Một lần nữa, nhóm xin chân thành cảm ơn cô và kính chúc cô sức khỏe, thành công trong công tác giảng dạy và nghiên cứu.

Nhóm 6

Mục lục

1 Giới thiệu

Hình 1: Vấn đề đặt ra của bài toán

Truy xuất thông tin (Information Retrieval - IR) là một bài toán trung tâm trong lĩnh vực xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP) và khoa học máy tính. Mục tiêu chính của IR là tìm kiếm và truy xuất các tài liệu hoặc đoạn văn bản có liên quan đến một truy vấn đầu vào từ phía người dùng.

Trong bối cảnh lượng thông tin về phim ngày càng trở nên phong phú và đa dạng, bao gồm mô tả phim, đánh giá, thể loại, tóm tắt nội dung và nhận xét từ người dùng, nhu cầu tìm kiếm phim dựa trên những mô tả ngắn gọn, đôi khi không đầy đủ, ngày càng trở nên thiết thực. Ai trong chúng ta chắc hẳn cũng từng bắt gặp một đoạn trích hấp dẫn của một bộ phim nào đó nhưng lại không thể nhớ tên phim (Hình 1).

Từ thực tế này, nhóm tác giả đề xuất xây dựng một hệ thống hỗ trợ người dùng tìm kiếm tên phim dựa trên mô tả nội dung. Nhờ việc dữ liệu phim thường được lưu trữ dưới dạng văn bản, các kỹ thuật IR trở nên rất phù hợp để giải quyết bài toán này. Tuy nhiên, khác với IR truyền thống chỉ dựa trên từ khóa, hệ thống cần hiểu được ngữ nghĩa tiềm ẩn bên trong mô tả phim và truy vấn người dùng, điều này đòi hỏi những phương pháp xử lý ngôn ngữ hiệu quả hơn.

Một trong những phương pháp kinh điển nhưng vẫn mang lại hiệu quả nhất định trong việc khai thác ngữ nghĩa tiềm ẩn là Phân tích ngữ nghĩa tiềm ẩn (Latent Semantic Analysis - LSA), một kỹ thuật học không giám sát dựa trên nền tảng đại số tuyến tính.

Trong khuôn khổ bài báo cáo này, nhóm tác giả sẽ trình bày chi tiết nguyên lý hoạt động của phương pháp LSA, cách mà nó vận hành hiệu quả trong việc rút trích ngữ nghĩa từ văn bản bằng kỹ thuật phân rã ma trận. Hơn nữa, nhóm tác giả áp dụng LSA trong việc giải quyết bài toán truy vấn tên phim dựa trên mô tả nội dung, đồng thời so sánh hiệu quả của phương pháp này với các hướng tiếp cận hiện đại khác. Qua đó, báo cáo không chỉ giải quyết một bài toán cụ thể mà còn góp phần cung cấp cái nhìn tổng quan về các kỹ thuật biểu diễn văn bản trong lĩnh vực truy xuất thông tin.

Cấu trúc bài báo cáo bao gồm:

- Giới thiệu (Mục 1): Là phần dẫn dắt vào bài báo cáo của nhóm
- Cơ sơ lý thuyết (Mục 2): Các công thức toán học ẩn sau thuật toán
- Ý tưởng hình thành (Mục 3): Nguyên nhân và động lực để hình thành phương pháp

- Phương pháp (Mục 4): Bao gồm cách tiền xử lý dữ liệu ở Mục 4.1, các phương pháp giải quyết vấn đề theo hướng LSA ở Mục 4.2 và các phương pháp embedding khác hiện đại hơn ở Mục 4.3.
- Thực nghiệm (Mục 5): Nội dung về các nhóm tác giả thí nghiệm và triển khai thuật toán trong thực tế.
- Kết luận (Mục 6): Đánh giá phương pháp.

Để người đọc có thể hình dung trực quan hơn, nhóm cũng đã xây dựng một trang web trình bày dự án, bao gồm tóm tắt nội dung, kết quả thực nghiệm và video demo tại: https://phucdm04.github.io/Film-Searching/

2 Cơ sở lý thuyết

2.1 Ma trận term-document

Ma trận term-document là dạng biểu diễn ma trận của tập hợp tài liệu văn bản, trong đó mỗi hàng tương ứng với một từ (*term*), mỗi cột tương ứng với một tài liệu (*document*), và mỗi phần tử biểu thị trọng số của từ trong tài liệu tương ứng. Ký hiệu tổng quát của ma trận:

$$X \in \mathbb{R}^{n \times N}$$

- n: số lượng từ (kích thước từ vựng)
- N: số lượng tài liệu
- x_{ij}: là phần tử của X, độ đo liên quan giữa từ t_i và tài liệu d_j

Như vậy, mỗi hàng của ma trận term-document tương ứng với cách biểu diễn của mỗi từ, mỗi cột tương ứng với cách biểu diễn của mỗi tài liệu, và ma trận này có tính chất thưa. Một phiên bản chuyển vị của nó là ma trận document-term, với mỗi hàng tương ứng với số tài liệu, mỗi cột tương ứng với số từ vựng duy nhất.

2.2 Term Frequency - Inverse Document Frequence

Trong không gian vector ngữ liệu, mỗi tài liệu d_j được biểu diễn như một vector trong không gian $\mathbb{R}^n$, trong đó n là kích thước từ vựng duy nhất trong tập dữ liệu. Mỗi thành phần của vector này tương ứng với một từ t_i trong từ vựng, và trọng số của nó được tính bằng Term Frequency - Inverse Document Frequency (TF-IDF).

Cho tập văn bản $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$ và từ vựng $V = \{t_1, t_2, \ldots, t_n\}$, ta định nghĩa:

- Hàm tần suất chuẩn hóa - term frequency (TF):

$$\mathrm{TF}(t_i, d_j) = \frac{f_{t_i,d_j}}{\sum_{k=1}^{n} f_{t_k,d_j}} \tag{1}$$

- Hàm nghịch đảo tần suất tài liệu - inverse document frequency (IDF):

$$\mathrm{IDF}(t_i) = \log\left(\frac{N}{df(t_i)}\right), \quad df(t_i) = |\{d_j \in \mathcal{D} : t_i \in d_j\}| \tag{2}$$

- Trọng số TF-IDF:

$$w_{ij} = \text{TF-IDF}(t_i, d_j) = \text{TF}(t_i, d_j) \cdot \text{IDF}(t_i) \tag{3}$$

Toàn bộ tập dữ liệu được biểu diễn thành ma trận $A \in \mathbb{R}^{n \times N}$, trong đó hàng i là từ t_i, cột j là tài liệu d_j, và phần tử $a_{ij} = w_{ij}$.

2.3 Phân rã giá trị kỳ dị

Cho một ma trận thực $A \in \mathbb{R}^{m \times n}$, tồn tại phép phân rã giá trị kỳ dị - Singular Value Decomposition (SVD) [1]:

$$A = U\Sigma V^\top \tag{4}$$

- $U \in \mathbb{R}^{m \times m}$: ma trận trực chuẩn chứa các vector riêng trái (left singular vectors), là các vector riêng của $AA^\top$.
- $V \in \mathbb{R}^{n \times n}$: ma trận trực chuẩn chứa các vector riêng phải (right singular vectors), là các vector riêng của $A^\top A$.
- $\Sigma \in \mathbb{R}^{m \times n}$: ma trận đường chéo chứa các giá trị kỳ dị $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r > 0, r = \min(m, n)$, là căn bậc hai của các trị riêng không âm của cả $AA^\top$ và $A^\top A$.

Tuy nhiên, với các bài toán thực tế, chỉ các giá trị kỳ dị đầu tiên chiếm phần lớn thông tin của ma trận. Khi này, ta có thể xấp xỉ ma trận A bằng cách giữ lại $k < r$ thành phần lớn nhất:

$$A \approx A_k = U_k \Sigma_k V_k^\top \tag{5}$$

với $U_k \in \mathbb{R}^{m \times k}, \Sigma_k \in \mathbb{R}^{k \times k}, V_k \in \mathbb{R}^{n \times k}$. Phương pháp này được gọi là Phân rã kì dị bị cắt ngắn - Truncated SVD hay xấp xỉ hạng thấp. Phần trăm lượng thông tin mà A_k giữ lại được tính như sau:

$$\frac{\|A_k\|_F^2}{\|A\|_F^2} = \frac{\sum_{i=1}^{k} \lambda_i^2}{\sum_{i=1}^{r} \lambda_i^2} \tag{6}$$

Giá trị k có thể được điều chỉnh để giữ lại lượng thông tin mong muốn. Nhờ vậy, Truncated SVD trở thành một công cụ mạnh mẽ để giảm chiều dữ liệu, nén thông tin, và khử nhiễu trong các hệ thống học máy và khai phá dữ liệu.

2.4 Ma trận đồng xuất hiện (Co-occurrence Matrix)

Ma trận đồng xuất hiện là một biểu diễn phân bố tần suất, trong đó mỗi phần tử $C_{i,j}$ biểu diễn số lần từ w_i và w_j cùng xuất hiện trong một ngữ cảnh nhất định (thường là cửa sổ trượt quanh từ mục tiêu).

Giả sử có một tập từ vựng $V = \{w_1, w_2, \ldots, w_n\}$, ta xây dựng một ma trận $C \in \mathbb{R}^{n \times n}$, trong đó:

$$C_{i,j} = \text{số lần từ } w_i \text{ và } w_j \text{ xuất hiện trong cùng một ngữ cảnh}$$

Ngữ cảnh có thể là một câu, một tài liệu, một cửa sổ có một cửa sổ kích thước k từ,. Ngoài ra, có thể áp dụng trọng số cho các đồng xuất hiện, ví dụ: càng gần nhau thì trọng số càng lớn ($\frac{1}{|i-j|^{0.75}}$), để phản ánh đúng mức độ gắn kết ngữ nghĩa.

2.5 Pointwise Mutual Information (PMI)

Pointwise Mutual Information đo mức độ bất ngờ của việc hai từ cùng xuất hiện, so với giả định chúng độc lập:

$$PMI(w_i, w_j) = \log_2 \frac{P(w_i, w_j)}{P(w_i) \cdot P(w_j)} \tag{7}$$

- $P(w_i, w_j) = \frac{C_{i,j}}{N}$: xác suất đồng xuất hiện.
- $P(w_i) = \frac{\sum_j C_{i,j}}{N}$, $P(w_j) = \frac{\sum_i C_{i,j}}{N}$: xác suất biên.
- $N = \sum_{i,j} C_{i,j}$: tổng số lần đồng xuất hiện.

Giá trị PMI dương phản ánh mối liên hệ ngữ nghĩa mạnh. Tuy nhiên, nếu từ rất hiếm, PMI có thể rất lớn không thực tế. Đồng thời, các giá trị âm hoặc gần 0 thường không hữu ích trong học máy. Vì thế, ta dùng phiên bản không âm của PMI chính là Positive PMI. Phiên bản này giúp loại bỏ các liên hệ yếu hoặc tiêu cực bằng cách lấy giá trị dương của PMI:

$$PPMI(w_i, w_j) = \max(0,\ PMI(w_i, w_j)) = \max\left(0,\ \log 2 \frac{P(w_i, w_j)}{P(w_i) \cdot P(w_j)}\right) \tag{8}$$

Mục tiêu là giữ lại các cặp từ có khả năng đồng xuất hiện cao hơn kỳ vọng ngẫu nhiên, phản ánh liên kết ngữ nghĩa tích cực.

2.6 Khoảng cách Hellinger

Cho hai phân phối xác suất rời rạc $P = (p_1, p_2, \ldots, p_n)$ và $Q = (q_1, q_2, \ldots, q_n)$, khoảng cách Hellinger giữa chúng được định nghĩa như sau:

$$H(P, Q) = \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^{n} \left(\sqrt{p_i} - \sqrt{q_i}\right)^2} \tag{9}$$

Dễ thấy, Phương trình (9) có thể viết thành dưới dạng khoảng cách Euclidean

$$H(P, Q) = \frac{1}{\sqrt{2}} ||\bar{P} - \bar{Q}||_2 \approx ||\bar{P} - \bar{Q}||_2 \tag{10}$$

Với $\bar{P} = \sqrt{P} = \left(\sqrt{p_1}, \sqrt{p_2}, \ldots, \sqrt{p_n}\right)$, đây được gọi là biến đổi Hellinger. Nhờ vào biến đổi này, phép tính vừa giữa được tính chất toàn cục của khoảng cách Euclidean, vừa giữ được tính chất simplex của không gian văn bản.

2.7 Phân tích thành phần chính (Principal Component Analysis - PCA)

Giả sử ta có ma trận dữ liệu $X \in \mathbb{R}^{n \times d}$ đã được chuẩn hóa, trong đó n là số lượng điểm dữ liệu và d là số đặc trưng. Mỗi hàng của X là một vector dữ liệu d-chiều. Mục tiêu của PCA là tìm ra một tập hợp mới gồm k trục tọa độ trực giao, sao cho khi chiếu dữ liệu lên không gian mới này, ta giữ lại được càng nhiều phương sai của dữ liệu gốc càng tốt.

Trước hết, cần tìm ma trận hiệp phương sai thể hiện mức độ tương quan tuyến tính giữa các chiều dữ liệu, được tính bằng:

$$S = \frac{1}{n} X^\top \tilde{X} \tag{11}$$

Trong đó $S \in \mathbb{R}^{d \times d}$ là một ma trận đối xứng và bán xác định dương. Sau đó, thực hiện phân tích trị riêng trên ma trận S để tìm các vector riêng và trị riêng:

$$Sv_i = \lambda_i v_i, \quad i = 1, \ldots, d$$

Trong đó λ_i là trị riêng tương ứng với vector riêng v_i. Các trị riêng biểu thị phương sai dữ liệu khi chiếu lên trục v_i. Sắp xếp các trị riêng theo thứ tự giảm dần, và chọn k trị riêng lớn nhất và lấy các vector riêng tương ứng để tạo thành ma trận chiếu $W_k \in \mathbb{R}^{d \times k}$. Số chiều k có thể được chọn dựa trên tỷ lệ phương sai mong muốn giữ lại:

$$\text{Explained Variance Ratio} = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i}$$

Dữ liệu ban đầu sau khi được chiếu vào không gian k-chiều mới được tính bằng:

$$X_{\text{new}} = XW_k$$

Mỗi hàng của $X_{\text{new}} \in \mathbb{R}^{n \times k}$ là một biểu diễn rút gọn của điểm dữ liệu gốc trong không gian thành phần chính.

Một cách khác để thực hiện PCA hiệu quả hơn là sử dụng phân rã giá trị suy biến. Thay vì tính ma trận hiệp phương sai, ta áp dụng trực tiếp SVD lên ma trận dữ liệu X (sau khi đã chuẩn hóa), dữ liệu chiếu vào không gian mới có thể được tính bằng biểu thức

$$X_{\text{new}} = XV_k \tag{12}$$

Việc sử dụng SVD giúp PCA ổn định và chính xác hơn, đặc biệt khi số chiều đặc trưng d lớn hoặc dữ liệu phân tán. Đây cũng là phương pháp được sử dụng trong các thư viện học máy hiện đại.

2.8 Độ đo tương đồng Cosine (Cosine Similarity)

Độ đo tương đồng Cosine (Cosine Similarity) là một phương pháp phổ biến để đo lường mức độ tương đồng giữa hai vector trong không gian vector, thường được áp dụng trong các bài toán xử lý ngôn ngữ tự nhiên và truy xuất thông tin. Thay vì dựa vào độ lớn tuyệt đối, cosine tập trung vào góc giữa hai vector, phản ánh mức độ định hướng tương đối của chúng.

Công thức tính độ đo cosine giữa hai vector $\vec{A}$ và $\vec{B}$ được định nghĩa như sau:

$$CosineSim(\vec{A}, \vec{B}) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \cdot \|\vec{B}\|} \tag{13}$$

Trong đó:

- $\vec{A} \cdot \vec{B}$ là tích vô hướng (dot product) giữa hai vector,
- $\|\vec{A}\|$ và $\|\vec{B}\|$ là chuẩn (norm) của từng vector, tính theo công thức $\|\vec{X}\| = \sqrt{\sum_i x_i^2}$.

Giá trị của cosine similarity nằm trong khoảng $[-1, 1]$. Tuy nhiên, trong các bài toán xử lý văn bản, nơi vector biểu diễn thường có giá trị không âm, độ đo này thường dao động trong khoảng $[0, 1]$:

- Giá trị gần 1: hai vector gần như cùng hướng, mức độ tương đồng cao.
- Giá trị gần 0: hai vector gần như vuông góc, ít tương đồng.
- Giá trị âm (hiếm gặp trong NLP): thể hiện hai vector có hướng ngược nhau.

3 Ý tưởng hình thành

3.1 Thách thức

Phim ảnh là một trong những loại hình giải trí phổ biến và có sức lan tỏa mạnh mẽ nhất trong đời sống hiện đại. Với sự phát triển của các nền tảng trực tuyến như Netflix, YouTube, Disney+, hay các trang xem phim miễn phí, người dùng ngày càng có nhiều lựa chọn hơn để tiếp cận các tác phẩm điện ảnh đến từ nhiều quốc gia, thể loại và thời kỳ khác nhau. Tuy nhiên, chính sự phong phú này cũng khiến việc tìm kiếm một bộ phim phù hợp trở nên ngày càng khó khăn, đặc biệt là trong những tình huống mà người dùng không nhớ rõ thông tin cụ thể về bộ phim.

Trong thực tế, nhu cầu tìm kiếm và khám phá phim thông qua tên phim, diễn viên hoặc thể loại là khá phổ biến và đã được nhiều hệ thống hỗ trợ tốt. Tuy nhiên, một tình huống rất thường gặp là người dùng chỉ nhớ mang máng nội dung hoặc diễn biến của bộ phim - ví dụ như một cảnh hành động đặc biệt, tình tiết bất ngờ, hoặc mối quan hệ giữa các nhân vật - mà hoàn toàn không nhớ tên phim hay thông tin cụ thể nào khác. Trong những trường hợp như vậy, việc sử dụng các phương pháp tìm kiếm truyền thống sẽ không còn hiệu quả, và người dùng thường gặp khó khăn trong việc diễn đạt ký ức của mình sao cho khớp với dữ liệu hệ thống có sẵn.

Thách thức đặt ra là làm thế nào để xây dựng một hệ thống truy xuất thông tin có khả năng tìm kiếm phim dựa trên mô tả ngôn ngữ tự nhiên, tức là cho phép người dùng nhập vào một đoạn văn mô tả nội dung phim mà họ nhớ, và từ đó hệ thống sẽ đề xuất những phim có khả năng trùng khớp cao nhất. Điều này đòi hỏi hệ thống phải hiểu được ngữ nghĩa của văn bản đầu vào, xử lý được sự mơ hồ trong cách diễn đạt của người dùng, và ánh xạ chính xác đến các bộ phim tương ứng trong cơ sở dữ liệu.

Hơn nữa, một điểm cần đạt được là hệ thống cần được thiết kế sao cho gọn nhẹ, có thể triển khai trong môi trường hạn chế về tài nguyên tính toán - ví dụ như máy tính cá nhân hoặc server miễn phí - thay vì dựa vào các mô hình ngôn ngữ lớn (LLM) tiêu tốn nhiều chi phí hoặc các thuật toán học sâu phức tạp. Do đó, bài toán này không chỉ là một bài toán truy xuất thông tin thuần túy, mà còn là bài toán cân bằng giữa tính hiệu quả, chi phí và trải nghiệm người dùng trong điều kiện thực tế.

3.2 Giải pháp

Giải quyết bài toán tìm kiếm phim dựa trên mô tả ngôn ngữ tự nhiên không chỉ giúp cải thiện rõ rệt trải nghiệm người dùng mà còn mở ra cơ hội khai thác sâu hơn khả năng hiểu ngôn ngữ trong các ứng dụng thực tế. Đây là bước tiến cần thiết để hướng tới các hệ thống thông minh có khả năng giao tiếp tự nhiên hơn với con người. Đặc biệt, việc nghiên cứu các giải pháp đơn giản, hiệu quả, không phụ thuộc vào mô hình ngôn ngữ quy mô lớn còn mang ý nghĩa thực tiễn lớn, giúp hệ thống có thể dễ dàng triển khai trong các điều kiện hạn chế về tài nguyên hoặc phục vụ cho cộng đồng rộng lớn hơn với chi phí thấp. Vì vậy, nhóm tác giả đề xuất hệ thống truy xuất thông tin sử dụng thuật toán Latent Senmatic Analysis (LSA)

3.2.1 Vì sao là LSA?

Trước khi LSA được dùng rộng rãi [2], đã có nhiều kỹ thuật biểu diễn từ xuất hiện nhưng tồn tại một số điểm yếu cố hữu. Đầu tiên, các kỹ thuật biểu diễn ma trận từ như ma trận document-term [3] hoặc ma trận đồng xuất hiện [3] tạo ra một ma trận rất thưa do chỉ dựa trên tần suất xuất hiện. Điều này gây khó khăn cho các thuật toán học máy do số chiều quá lớn, dẫn đến hiện tượng curse of dimensionality (lời nguyền cao chiều) và khiến việc học các mối quan hệ ngữ nghĩa trở nên kém hiệu quả. Bên cạnh đó, việc lưu trữ các ma trận thưa như vậy cũng tiêu tốn nhiều bộ nhớ

Tiếp theo, các phương pháp biểu diễn truyền thống không phản ánh chính xác ngữ nghĩa và mối quan hệ giữa các từ. Chúng không nhận diện được các từ đồng nghĩa như *car* và *automobile*, đồng thời không xử lý được hiện tượng từ đa nghĩa như *back*, dẫn đến khó khăn trong việc đo lường độ tương đồng giữa từ hoặc văn bản. Một hạn chế rõ rệt của biểu diễn dựa trên tần suất là các từ phổ biến như *the*, *is*, *and*, *of* thường có trọng số rất cao, dù đóng góp rất ít về mặt ngữ nghĩa. Điều này gây nhiễu cho vector đặc trưng, làm giảm hiệu quả trong việc đo độ tương đồng, phân loại văn bản hoặc trích xuất thông tin.

Cuối cùng, việc phát hiện các chủ đề tiềm ẩn trong một tập văn bản lớn là một thách thức đáng kể. Các phương pháp thời kỳ đầu thường dựa vào các luật thủ công, từ điển chủ đề, hoặc các quy tắc heuristic đơn giản. Tuy nhiên, cách tiếp cận này thiếu khả năng tổng quát hóa, khó áp dụng cho các tập dữ liệu lớn và đa dạng về nội dung.

Để khắc phục những hạn chế đó, LSA đã được đề xuất như một phương pháp có nền tảng toán học rõ ràng nhằm trích xuất và biểu diễn ý nghĩa ngữ cảnh của từ dựa trên các phép tính thống kê áp dụng trên tập văn bản lớn. Ý tưởng chính của LSA là việc một từ xuất hiện hay không xuất hiện trong các ngữ cảnh khác nhau sẽ tạo nên những ràng buộc tương hỗ, từ đó phản ánh mức độ tương đồng về ngữ nghĩa giữa các từ và cụm từ. Nhiều nghiên cứu đã chứng minh tính hiệu quả của LSA trong việc mô phỏng tri thức ngôn ngữ của con người: từ việc đạt điểm số tương đương với con người trong các bài kiểm tra từ vựng và kiến thức chuyên môn, cho đến khả năng mô phỏng cách con người phân loại từ, đánh giá mức độ liên quan giữa các từ hoặc đoạn văn. Ngoài ra, LSA còn có thể ước lượng độ mạch lạc của đoạn văn, khả năng tiếp thu nội dung của người học, cũng như đánh giá chất lượng tri thức trong văn bản.

3.2.2 Hướng tiếp cận thay thế

LSA tuy có thể đáp ứng được các thách thức Mục 3.1, nhưng nó vẫn là một phương pháp đã lâu đời và thiếu hiệu quả trong các hệ thống hiện đại khi mà vấn đề nguồn tài nguyên

được cải thiện. Sự phát triển mạnh mẽ của phần cứng hiện đại, đặc biệt là GPU, các mô hình học sâu trở nên khả thi và phổ biến hơn, cho phép xây dựng những biểu diễn ngôn ngữ giàu ngữ nghĩa hơn, học được trực tiếp từ dữ liệu mà không cần giả định tuyến tính hay ma trận thưa.

Vì vậy, để có cái nhìn toàn diện và đánh giá khách quan hơn giữa các thế hệ phương pháp, nhóm tác giả xin giới thiệu thêm một số hướng tiếp cận hiện đại hơn. Các phương pháp này không chỉ cải thiện hiệu quả biểu diễn ngữ nghĩa của từ vựng mà còn cho phép mô hình học được mối quan hệ phức tạp giữa các từ trong ngữ cảnh, từ đó nâng cao độ chính xác trong các bài toán truy xuất thông tin phim. Việc sử dụng các phương pháp thay thế để so sánh với LSA sẽ giúp làm rõ ưu điểm và hạn chế của từng phương pháp trong từng điều kiện cụ thể, từ đó đưa ra lựa chọn phù hợp hơn cho các hệ thống tìm kiếm thực tế.

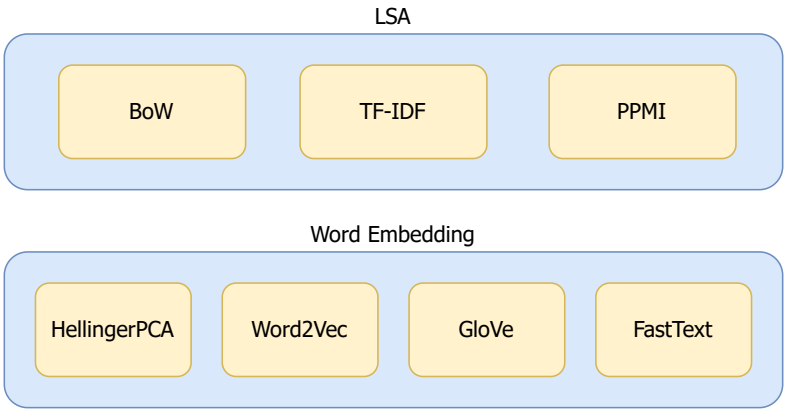
4 Phương pháp

Truy xuất thông tin là một bài toán lâu đời trong lĩnh vực xử lý ngôn ngữ tự nhiên, đóng vai trò quan trọng trong nhiều ứng dụng thực tế nhằm nâng cao trải nghiệm người dùng. Mục tiêu của hệ thống IR là trả về những kết quả có nội dung phù hợp nhất với nhu cầu tìm kiếm của người dùng.

Một hệ thống IR điển hình bao gồm các thành phần chính như: tập tài liệu (corpus), truy vấn (query), chỉ mục (index), và mô hình truy xuất (retrieval model). Quy trình hoạt động cơ bản bao gồm: xử lý câu truy vấn, ánh xạ truy vấn và tài liệu sang không gian nhúng, tính toán độ tương đồng (thường là Cosine), và xếp hạng kết quả. Các chỉ số đánh giá phổ biến của hệ IR bao gồm Precision, Recall, F1-score, MAP, MRR và nDCG, giúp đảm bảo kết quả truy xuất vừa chính xác vừa có thứ tự ưu tiên phù hợp.

Ở phần này, nhóm tác giả trình bày một số kỹ thuật tiền xử lý dữ liệu văn bản ở Mục 4.1. Cùng với đó là giới thiệu và so sánh hai hướng tiếp cận để xây dựng mô hình truy xuất, được minh họa trong Hình 2:

- LSA: là hướng tiếp cận chính, được trình bày chi tiết trong Mục 4.2, bao gồm các phương pháp truyền thống như BoW, TF-IDF và PPMI, nhằm khai thác mối quan hệ ngữ nghĩa tiềm ẩn giữa các từ và tài liệu thông qua ma trận xuất hiện và giảm chiều. Đối với hướng tiếp cận này, nhóm tác giả sẽ trình bày ví dụ minh hoạ, thuật toán, và cách triển khai code để có cái nhìn sâu sắc.
- Các kĩ thuật Word Embedding khác: là hướng tiếp cận có tiềm năng thay thế LSA, được trình bày trong Mục 4.3, gồm các phương pháp hiện đại hơn như Word2Vec, GloVe, FastText, HellingerPCA. Các kỹ thuật này có thể tận dụng ngữ cảnh phân phối của từ trong tập liệu lớn để tạo ra biểu diễn vector giàu ngữ nghĩa. Tương tự với hướng tiếp cận LSA, với từng phương pháp, ví dụ minh hoạ, thuật toán, cách triển khai code sẽ được trình bày, cùng với đó là vấn đề phương pháp này có thể giải quyết trong bài toán nhúng từ.



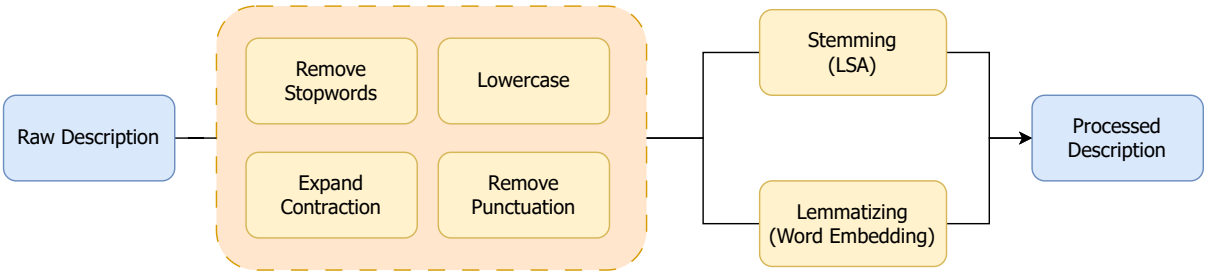
Hình 2: Tổng quan các phương pháp truy xuất thông tin

4.1 Tiền xử lý dữ liệu văn bản

Trong các bài toán NLP, dữ liệu văn bản thô thường chứa nhiều yếu tố gây nhiễu như từ viết hoa, dấu câu, từ dừng hoặc các hình thức viết tắt khác nhau. Nếu đưa trực tiếp dữ liệu này vào mô hình mà không xử lý trước, sẽ dẫn đến các vấn đề như: Tăng kích thước từ vựng không cần thiết; giảm hiệu quả truy xuất; sai lệch ngữ nghĩa trong quá trình vector hóa hoặc phân tích thống kê.

Tiền xử lý dữ liệu vì vậy là bước nền tảng không thể thiếu, giúp chuẩn hóa đầu vào theo cách nhất quán và tối ưu hơn cho việc học đặc trưng và phân tích. Các lợi ích chính bao gồm: Giảm chiều từ vựng; Tăng độ chính xác của mô hình; Tối ưu hiệu năng; Tạo tính nhất quán cho biểu diễn đặc trưng.

Tùy vào bài toán cụ thể, các bước xử lý có thể điều chỉnh linh hoạt. Dưới đây là ví dụ minh họa cho quá trình tiền xử lý được thực hiện trong bài toán của nhóm. Hình 3 tóm tắt quy trình xử lý từ đoạn mô tả thô đến khi trả về đoạn mô tả đã được xử lý.



Hình 3: Quy trình tiền xử lý dữ liệu

Các bước tiền xử lý.

1. Chuẩn bị tập corpus

```
corpus = [
    "A father and son bond over a stick up robbery.",
    "An old family foe threatens a young woman's Nashville dreams."
]
```

Bộ `corpus` sẽ được tái sử dụng ở các ví dụ sau để đồng nhất về mặt minh họa.

2. Chuyển về chữ thường (Lowercasing)

```
lowercase_corpus = [
    "a father and son bond over a stick up robbery",
    "an old family foe threatens a young woman's nashville dreams"
]
```

3. Mở rộng từ viết tắt (Expand contractions)
   Do văn bản không có từ viết tắt nên không thay đổi:

```
expanded_corpus = lowercase_corpus
```

4. Loại bỏ dấu câu (Remove punctuations)

```
punct_removed_corpus = [
    "a father and son bond over a stick up robbery",
    "an old family foe threatens a young womans nashville dreams"
]
```

5. Loại bỏ từ dừng (Remove stopwords)

```
stopwords_removed_corpus = [
    "father son bond stick robbery",
    "old family foe threatens young womans nashville dreams"
]
```

6. Lemmatizing và Stemming (tuỳ pipeline)

   (a) Lemmatization

```
lemmatized_corpus = [
    "father son bond stick robbery",
    "old family foe threatens young woman nashville dream"
]
```

   (b) Stemming

```
stemmed_corpus = [
    "father son bond stick robberi",
    "old famili foe threaten young woman nashvil dream"
]
```

Nhận xét

- **Stemming** (dùng trong `LSASVDPipeline`) đơn giản và nhanh, tuy nhiên có thể làm biến dạng từ (vd: *robbery* → *robberi*). Tuy vậy, LSA hoạt động dựa trên thống kê tần suất từ trong ma trận tài liệu–từ (document-term), nên vẫn có thể khai thác tốt các dạng rút gọn này mà không bị ảnh hưởng nhiều bởi ngữ pháp hay ngữ nghĩa.
- **Lemmatization** (dùng trong `WordEmbeddingPipeline`) giữ ngữ nghĩa tốt hơn, phù hợp với các mô hình embedding hiện đại (như Word2Vec, GloVe,...) vốn học dựa trên ngữ cảnh và quan hệ ngữ nghĩa giữa các từ. Việc dùng từ nguyên dạng giúp mô hình học được các mối quan hệ từ vựng chính xác hơn.
- Việc tách riêng hai pipeline giúp tối ưu hóa cho từng loại thuật toán, từ đó tăng hiệu quả tổng thể.

Triển khai bằng code Python.

- Khởi tạo tập văn bản đầu vào

```
corpus = [
    "A father and son bond over a stick up robbery.",
    "An old family foe threatens a young woman's Nashville dreams."
]
```

- Pipeline 1: Dành cho LSA (Stemming)

```
class LSASVDPipeline:
    """
    Preprocessing pipeline for LSA/SVD
    """

    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    html_pattern = re.compile(r'<[^>]+>')
    non_alpha_pattern = re.compile(r'[^a-zA-Z\s]')

    def __init__(
        self,
        extra_stopwords: Optional[Set[str]] = None
    ):
        self.stemmer = PorterStemmer()
        self.lemmatizer = WordNetLemmatizer()
        base_sw = set(stopwords.words('english'))
        self.stop_words = base_sw.union(extra_stopwords or set())

    def clean_text(self, text: str) -> str:
        if not isinstance(text, str):
            return ""
        # lowercase
        text = text.lower()
        # remove HTML
        text = self.html_pattern.sub('', text)
        # remove URLs
        text = self.url_pattern.sub('', text)
        # remove non-alphabetic
        text = self.non_alpha_pattern.sub(' ', text)
        # normalize spaces
        return re.sub(r'\s+', ' ', text).strip()

    def tokenize_filter(self, text: str) -> List[str]:
        tokens = word_tokenize(text)
        return [tok for tok in tokens if tok not in self.stop_words and len(tok) >
            2]

    def stem(self, tokens: List[str]) -> List[str]:
        return [self.stemmer.stem(tok) for tok in tokens]

    def lemmatize(self, tokens: List[str]) -> List[str]:
        return [self.lemmatizer.lemmatize(tok) for tok in tokens]

    def preprocess(self, text: str, use_stemming: bool = True) -> str:
        cleaned = self.clean_text(text)
        tokens = self.tokenize_filter(cleaned)
        processed = self.stem(tokens) if use_stemming else self.lemmatize(tokens)
        return ' '.join(processed)

    def batch(self, texts: List[str], use_stemming: bool = True) -> List[str]:
        return [self.preprocess(txt, use_stemming) for txt in texts]

lsa_preprocessor = LSASVDPipeline()
lsa_corpus = [lsa_preprocessor.preprocess(text) for text in corpus]
# Output: ['father son bond stick robberi', 'old famili foe threaten young woman
    nashvil dream']
```

- Pipeline 2: Dành cho các phương pháp embedding khác (Lemmatizing)

```
class WordEmbeddingPipeline:
```

```
    """
    Preprocessing pipeline for WordEmbedding (Word2Vec/GloVe/FastText)
    --> CORE: Lightweight preprocessing. Maintain context.
    """

    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    html_pattern = re.compile(r'<[^>]+>')
    punct_except_basic = re.compile(r'[^\w\s\.\!?]')

    def __init__(self, minimal_stopwords: Optional[Set[str]] = None):
        self.lemmatizer = WordNetLemmatizer()
        defaults = {"a", "an", "the", "and", "or", "but", "is", "are", "was", "were
            "}
        self.stop_words = defaults.union(minimal_stopwords or set())

    def expand_contractions(self, text: str) -> str:
        return contractions.fix(text)

    def clean_gentle(self, text: str) -> str:
        if not isinstance(text, str):
            return ""
        text = self.expand_contractions(text)
        text = text.lower()
        text = self.html_pattern.sub('', text)
        text = self.url_pattern.sub(' ', text)
        text = self.punct_except_basic.sub(' ', text)
        return re.sub(r'\s+', ' ', text).strip()

    def tokenize_sentences(self, text: str) -> List[List[str]]:
        sentences = sent_tokenize(text)
        result = []
        for sent in sentences:
            toks = word_tokenize(sent)
            filtered = [tok for tok in toks if tok not in self.stop_words and len(
                tok) > 1]
            if filtered:
                result.append(filtered)
        return result

    def lemmatize(self, tokens: List[str]) -> List[str]:
        return [self.lemmatizer.lemmatize(tok) for tok in tokens]

    def preprocess(self, text: str) -> List[List[str]]:
        cleaned = self.clean_gentle(text)
        sents = self.tokenize_sentences(cleaned)
        return [self.lemmatize(sent) for sent in sents]

    def preprocess_single_text(self, text: str) -> List[str]:
        sents = self.preprocess(text)
        return [tok for sent in sents for tok in sent]

    def flatten(self, text: str) -> str:
        return ' '.join(self.preprocess_single_text(text))

    def batch(self, texts: List[str]) -> List[List[List[str]]]:
        return [self.preprocess(txt) for txt in texts]


embed_preprocessor = WordEmbeddingPipeline()
wemb_corpus = [embed_preprocessor.preprocess(text) for text in corpus]
# Output: ['father son bond stick robbery', 'old family foe threaten young woman
    nashville dream']
```

4.2 Latent Semantic Analysis

Nguyên lý của LSA là ánh xạ ma trận biểu diễn từ (document-term hoặc term-document) sang một không gian mới giàu ngữ nghĩa tiềm ẩn nhờ vào phép xoay trục. Cốt lõi của phương pháp là áp dụng Truncated Singular Value Decomposition, nhằm tách ma trận

ban đầu thành ba thành phần chính phản ánh cấu trúc tiềm ẩn của ngôn ngữ.

Cho $X \in \mathbb{R}^{n \times N}$ là ma trận term-document, áp dụng Truncated SVD, ta có (14):

$$X \approx U_k \Sigma_K V_k^\top \tag{14}$$

- $U_k \in \mathbb{R}^{n \times k}$: ma trận biểu diễn các từ trong không gian $\mathcal{S}$, đã được lược bỏ các thành phần không quan trọng
- $V_k \in \mathbb{R}^{k \times N}$: ma trận biểu diễn các tài liệu trong không gian $\mathcal{S}$, đã được lược bỏ các thành phần không quan trọng
- $\Sigma_k \in \mathbb{R}^{k \times k}$: ma trận đường chéo chứa các giá trị suy biến, biểu diễn mức độ quan trọng của các thành phần trong không gian $\mathcal{S}$, và đã được lược bỏ các thành phần không quan trọng.

Sau khi tách thành ba ma trận, nếu muốn biểu diễn tài liệu hoặc từ bất kỳ dựa theo không gian ngữ nghĩa, thì chỉ cần thực hiện phép nhân ma trận.Có hai cách tiếp cận việc xoay trục này: xoay theo không gian từ vựng và xoay theo không gian văn bản.

Xoay theo không gian từ vựng tức là ánh xạ các từ sang một không gian ngữ nghĩa có chiều thấp, trong đó các từ có ngữ nghĩa tương tự sẽ có tọa độ gần nhau. Điều này giúp làm nổi bật mối liên hệ ngữ nghĩa giữa các từ, ngay cả khi chúng không cùng xuất hiện trong một ngữ cảnh cụ thể. Để ánh xạ b từ ngữ bất kỳ vào không gian ngữ nghĩa tiềm ẩn, ta biểu diễn chúng qua ma trận $\mathbf{t} \in \mathbb{R}^{b \times N}$. Khi đó, biểu diễn tương ứng là $\overline{\mathbf{t}}$ được xác định theo công thức:

$$\overline{\mathbf{t}} = \mathbf{t} V_k \tag{15}$$

Nếu ánh xạ văn bản gốc vào tập không gian từ, công thức (15) trở thành:

$$\overline{X} = X V_k = U_k \Sigma_k \tag{16}$$

Khi này, mỗi hàng của $\overline{X} \in \mathbb{R}^{n \times k}$ hoặc $\overline{\mathbf{t}} \in \mathbb{R}^{b \times k}$ tương ứng với mỗi biểu diễn từ trong không gian ngữ nghĩa. Với cách xoay theo không gian văn bản, các văn bản được biểu diễn dưới dạng vector trong không gian ngữ nghĩa mới. Việc thay đổi về thấp chiều này giúp phát hiện ra các chủ đề tiềm ẩn và mô hình hóa nội dung văn bản hiệu quả hơn, nhất là khi so sánh mức độ tương đồng giữa các tài liệu. Để ánh xạ a tài liệu bất kỳ vào không gian $\mathcal{S}$, trước tiên cần biểu diễn chúng dưới dạng ma trận term-document $\mathbf{d} \in \mathbb{R}^{n \times a}$. Sau đó, dùng công thức:

$$\tilde{\mathbf{d}} = U_k^\top \mathbf{d} \tag{17}$$

Nếu ánh xạ văn bản gốc vào tập không gian tài liệu, công thức (17) trở thành:

$$\tilde{X} = U_k^\top X = \Sigma_k V_k^\top \tag{18}$$

Khi này, mỗi cột của $\tilde{X} \in \mathbb{R}^{k \times N}$ hoặc $\tilde{\mathbf{d}} \in \mathbb{R}^{k \times a}$ tương ứng với mỗi biểu diễn tài liệu trong không gian ngữ nghĩa. Với trường hợp ma trận biểu diễn là document-term, $\mathbf{d}$ và $\mathbf{t}$ sẽ hoán đổi vài trò cho nhau, cụ thể:

$$\tilde{\mathbf{t}} = U_k^\top \mathbf{t}$$

$$\overline{\mathbf{d}} = \mathbf{d} V_k$$

Khi này, mỗi hàng $\overline{\mathbf{d}}$ chính là một biểu diễn tài liệu trong không gian tiềm ẩn, mỗi cột $\tilde{\mathbf{t}}$ chính là một biểu diễn từ ngữ trong không gian tiềm ẩn

Nếu trong trường hợp áp dụng Truncated SVD lên ma trận đồng xuất hiện thì sao? Do tính chất đối xứng của ma trận đồng xuất hiện, ma trận $X \in \mathbb{R}^{n \times n}$ được biểu diễn: $X \approx U_k \Sigma_k U_k^\top$. Khi này, biểu diễn của một từ $\mathbf{t}$ trong không gian ngữ nghĩa tiềm ẩn:

$$\overline{\mathbf{t}} = \mathbf{t} U_k = (U_k^\top \mathbf{t}^\top)^\top = \tilde{\mathbf{t}}^\top \tag{19}$$

Có thể thấy rằng việc biểu diễn một từ khi sử dụng ma trận đồng xuất hiện rất đơn giản, nhưng nếu muốn biểu diễn một tài liệu thì lại khó khăn vì ma trận đồng xuất hiện chỉ là biểu diễn của từ. Để giải quyết, kĩ thuật pooling ra đời bằng cách tổng hợp các vector từ ngữ thành một vector tài liệu. Một số kỹ thuật pooling phổ biến bao gồm:

- Mean pooling: Tính trung bình các vector từ trong tài liệu.
- Sum pooling: Cộng tất cả các vector từ lại với nhau.
- Max pooling: Lấy giá trị lớn nhất theo từng chiều.

Với ý tưởng của kĩ thuật pooling, ta hoàn toàn có thể biểu diễn tất cả các cấp bậc khác của văn bản chỉ nhờ vào đơn vị cơ bản là từ ngữ. Sau khi hiểu được cách hoạt động của LSA, câu hỏi đặt ra là: làm sao xây dựng được ma trận biểu diễn từ phù hợp để tiến hành phân tích ngữ nghĩa tiềm ẩn? Như đã giới thiệu ở Mục 3.2, một số dạng ma trận được sử dụng phổ biến là ma trận document-term, term-document hoặc co-occurrence matrix, tùy thuộc vào mục tiêu biểu diễn và ngữ cảnh ứng dụng. Nhóm tác giả xin giới thiệu ba kỹ thuật phổ biến để xây dựng nên ma trận này, bao gồm: Bag-of-Words (BoW), Term Frequency - Inverse Document Frequence (TF-IDF), và Positive Pointwise Mutual Information (PPMI).

Thuật toán

Algorithm 1 LSA: Biểu diễn tài liệu trong không gian ngữ nghĩa tiềm ẩn

```
procedure LSA(𝒟, d)
    Tạo ma trận biểu diễn từ X từ 𝒟
    Phân rã giá trị kỳ dị rút gọn (SVD) trên X để thu được ma trận U_k
    Tạo biểu diễn ban đầu d̂ cho tài liệu d
    Xoay trục: d̄ ← U_k d̂
    return d̄
end procedure
```

Code minh họa. Do sự phụ thuộc vào cách tạo nên ma trận biểu diễn, phần code minh họa sẽ được trình bày phía sau.

4.2.1 Bag-of-Words

Xuất phát từ nhu cầu mã hóa văn bản để hỗ trợ xử lý bằng các công cụ toán học và thuật toán, các biểu diễn như ma trận đồng xuất hiện và ma trận tài liệu-từ đã được phát triển. Trong bối cảnh này, mô hình Bag-of-Words (BoW) [3] ra đời, dựa trên nền tảng của ma trận đồng xuất hiện, với nguyên lý cốt lõi là biểu diễn tài liệu thông qua

việc thống kê tần suất xuất hiện của các từ trong một tập từ vựng chuẩn hóa. Trong mô hình này, mỗi từ được xem như một đặc trưng độc lập, và toàn bộ tài liệu được ánh xạ thành một vector tần suất dựa trên tập từ vựng. Mặc dù không xét đến ý nghĩa ngữ nghĩa của từ hay ngữ cảnh sử dụng, mô hình BoW vẫn cung cấp một biểu diễn đủ mạnh mẽ, cho phép các thuật toán học máy hoạt động hiệu quả trong nhiều tác vụ xử lý ngôn ngữ tự nhiên.

Quy trình xây dựng BoW.

1. Tiền xử lý văn bản
2. Tokenize văn bản thành từ đơn (theo whitespace hoặc hàm tùy chỉnh).
3. Xây dựng từ vựng:
   - Loại bỏ các từ hiếm, có tần suất thấp hơn một ngưỡng `min_freq`.
   - Giới hạn số đặc trưng tối đa bằng `max_features`. `max_features` là số từ vựng tối đa muốn giữ lại trong vocab. Tham số này giúp ngăn ngừa tình trạng ma trận quá thưa khi hoạt động với các từ hiếm, từ đó giảm thiểu chi phí tính toán.
4. Mã hóa từng văn bản thành vector BoW bằng cách đếm số lần các từ trong từ vựng xuất hiện. Lưu ý, nếu một văn bản mới được ghép từ hai văn bản trên, thì biểu diễn Bag-of-Words của nó sẽ là: BoW3 = BoW1 $\uplus$ BoW2

Ví dụ minh họa. Với `corpus` ban đầu được lấy từ bộ dữ liệu IMDB:

```
raw_corpus = [
  "A father and son bond over a stick up robbery.",
  "An old family foe threatens a young woman's Nashville dreams."
]
```

- **Bước 1: Chuẩn bị văn bản và tiền xử lý**

  Các phương pháp được áp dụng là biến đổi về chữ thường (không có ký tự chữ hoa); loại bỏ stopwords và các dấu câu; stemming. Kết quả được một `corpus` như sau:

  ```
  preprocessed_corpus = [
    "father son bond stick robberi",
    "old famili foe threaten young woman nashvil dream"
  ]
  ```

- **Bước 2: Token hóa từng câu dựa vào khoảng trắng**

  ```
  tokens1 = ["father", "son", "bond", "stick", "robberi"]
  tokens2 = ["old", "famili", "foe", "threaten", "young",
  "woman", "nashvil", "dream"]
  ```

  Mỗi văn bản được chia nhỏ thành danh sách các từ riêng biệt. Việc tách theo dấu cách giúp đơn giản hóa việc đếm tần suất từ.

- **Bước 3: Xây dựng tập từ vựng**

Sử dụng điều kiện `min_freq = 1`, `max_features = 10`, ta tính tần suất xuất hiện của mỗi từ trong toàn bộ corpus (`max_features` sẽ được chọn thông qua giai đoạn phân tích đối với bộ `corpus` lớn. Ở ví dụ này, tham số được chọn nhằm để minh họa để ví dụ, không có ý nghĩa toán học) :

`"father":`	1
`"son":`	1
`"bond":`	1
`"stick":`	1
`"robberi":`	1
`"old":`	1
`"famili":`	1
`...`	tất cả các từ khác đều có tần suất = 1

Sau đó, lấy tất cả từ thỏa mãn điều kiện và giới hạn số lượng theo `max_features`:

```
vocab = ["father", "son", "bond", "stick", "robberi", "old",
            "famili", "foe", "threaten", "young"]
```

- **Bước 4: Mã hóa vector BoW theo từ vựng đã chọn**

Đầu tiên, ta định nghĩa rõ:

`doc1: "father son bond stick robberi"`

`doc2: "old famili foe threaten young woman nashvil dream"`

Tập từ vựng đã xây dựng từ bước trước:

```
vocab = ["father", "son", "bond", "stick", "robberi", "old",
            "famili", "foe", "threaten", "young"]
```

Tiến hành mã hóa BoW cho từng văn bản theo từng bước như sau:

1. Duyệt qua từng tài liệu - `doc` (tức là từng câu trong `corpus` – tập văn bản).
2. Với mỗi `doc`, ta sẽ lần lượt duyệt qua từng từ trong tập từ vựng (vocab).
3. Ở mỗi bước, ta kiểm tra xem từ đó có xuất hiện trong `doc` hay không.
4. Nếu từ xuất hiện đúng một lần trong `doc`, ta gán giá trị 1 vào vị trí tương ứng trong vector BoW của `doc` đó. Nếu từ không xuất hiện, thì giữ nguyên giá trị 0 tại vị trí đó. Tương tự với những số lần xuất hiện khác.
5. Sau khi xử lý hết tất cả từ trong `vocab`, ta thu được một vector BoW – chính là biểu diễn dạng số của văn bản ban đầu.

Với `doc1`:

`"father"`	xuất hiện ⇒ 1
`"son"`	xuất hiện ⇒ 1
`"bond"`	xuất hiện ⇒ 1
`"stick"`	xuất hiện ⇒ 1
`"robberi"`	xuất hiện ⇒ 1
`"old"`	không xuất hiện ⇒ 0
`"famili"`	không xuất hiện ⇒ 0
`"foe"`	không xuất hiện ⇒ 0
`"threaten"`	không xuất hiện ⇒ 0
`"young"`	không xuất hiện ⇒ 0

BoW vector cho `doc1`:

```
BoW_doc1 = {  "father":1, "son":1, "bond":1, "stick":1, "robberi":1,
              "old":0, "famili":0, "foe":0, "threaten":0, "young":0 }
```

Tương tự với `doc2`:

`"father"`	không xuất hiện ⇒ 0
`"son"`	không xuất hiện ⇒ 0
`"bond"`	không xuất hiện ⇒ 0
`"stick"`	không xuất hiện ⇒ 0
`"robberi"`	không xuất hiện ⇒ 0
`"old"`	xuất hiện ⇒ 1
`"famili"`	xuất hiện ⇒ 1
`"foe"`	xuất hiện ⇒ 1
`"threaten"`	xuất hiện ⇒ 1
`"young"`	xuất hiện ⇒ 1

BoW vector cho doc2:

```
BoW_doc2 = {  "father":0, "son":0, "bond":0, "stick":0, "robberi":0,
              "old":1, "famili":1, "foe":1, "threaten":1, "young":1 }
```

Kết quả tổng hợp: `BoW` = `BoW_doc1` ⊎ `BoW_doc2`

```
BoW = {  "father":1, "son":1, "bond":1, "stick":1, "robberi":1,
         "old":1, "famili":1, "foe":1, "threaten":1, "young":1 }
```

Mỗi văn bản được chuyển thành vector số nguyên, biểu diễn số lần xuất hiện của mỗi từ trong tập từ vựng. Đây là bước cốt lõi trong mô hình BoW.

Cuối cùng, xuất ra ma trận BoW.

	father	son	bond	stick	robberi	old	famili	foe	threaten	young
doc1	1	1	1	1	1	0	0	0	0	0
doc2	0	0	0	0	0	1	1	1	1	1

Đây là kết quả cuối cùng sau khi áp dụng thuật toán BoW để xây dựng ma trận số (document-term matrix) theo phương pháp được nêu ở trên.

Thuật toán.

Algorithm 2 Xây dựng ma trận BoW từ tập tài liệu

procedure BUILDBOW(D, `min_freq`, `max_features`)
  Tạo bộ đếm tần suất từ trên toàn tập và khởi tạo ma trận không $X \in \mathbb{R}^{N \times F}$
  Lọc từ có `freq < min_freq`, giữ lại top `max_features`
  Gán chỉ số cho từ vựng $T = \{t_1, ..., t_F\}$
  for $j = 1$ **to** N **do**
    Đếm số lần mỗi từ $w \in T$ xuất hiện trong d_j
    $X_{j,\text{idx}(w)} \leftarrow$ số lần w trong d_j
  end for
  return X
end procedure

Code minh họa. Ở phần này, nhóm tác giả đã thiết kế thuật toán phân rã Truncated SVD vào chung với thuật toán ma trận BoW để tạo nên thuật toán LSA hoàn chỉnh.

Lớp `BagOfWords`: Lớp này xây dựng biểu diễn Bag-of-Words cho các tài liệu văn bản đã làm sạch. Các phương thức chính gồm:

- `__init__`: Khởi tạo với các tham số như tần suất từ tối thiểu (`min_word_freq`), số lượng đặc trưng tối đa (`max_features`), và bộ tách từ (`tokenizer`).
- `fit(documents)`:
  - Tách từ và đếm số lần xuất hiện của mỗi từ trong tập văn bản.
  - Lọc các từ theo tần suất tối thiểu và giới hạn số lượng đặc trưng.
  - Xây dựng từ điển ánh xạ từ $\rightarrow$ chỉ số.
- `transform(documents)`: Chuyển tập tài liệu thành ma trận BoW có kích thước $(n\_documents, n\_features)$.
- `transform_single(document)`: Tạo vector BoW cho một tài liệu đơn lẻ.
- `fit_transform(documents)`: Gộp hai bước `fit` và `transform`.
- `get_feature_names()`: Trả về danh sách các từ trong từ điển.
- `get_vocabulary_size()`: Trả về kích thước của từ điển.
- `get_word_frequency(word)`: Trả về số lần xuất hiện của một từ.
- `print_vocabulary_info(top_n)`: In thông tin thống kê về từ điển.
- `save_vocabulary(filepath)` và `load_vocabulary(filepath)`: Lưu và tải từ điển từ tệp văn bản.

```
class BagOfWords:
    def __init__(self,
                 min_word_freq: int = 1,
                 max_features: Optional[int] = None,
                 tokenizer: str = 'whitespace'):
        self.min_word_freq = min_word_freq
        self.max_features = max_features
        self.tokenizer = tokenizer

```

```
        # Vocabulary and mappings
        self.vocabulary = {}  # word -> index
        self.idx_to_word = {}  # index -> word
        self.word_counts = Counter()
        self.is_fitted = False

    def _tokenize(self, text: str) -> List[str]:
        if self.tokenizer == 'whitespace':
            # Split by whitespace and remove empty tokens
            return [token.strip() for token in text.split() if token.strip()]
        elif callable(self.tokenizer):
            # Use custom tokenizer function
            return self.tokenizer(text)
        else:
            raise ValueError("tokenizer must be 'whitespace' or a function")

    def fit(self, documents: List[str]) -> 'BagOfWords':
        # Reset
        self.word_counts = Counter()

        # Tokenize and count word frequencies
        for doc in documents:
            tokens = self._tokenize(doc)
            self.word_counts.update(tokens)

        # Filter words by min_word_freq
        filtered_words = {word: count for word, count in self.word_counts.items()
                          if count >= self.min_word_freq}

        # Sort by descending frequency
        sorted_words = sorted(filtered_words.items(), key=lambda x: x[1], reverse=True)

        # Limit number of features
        if self.max_features:
            sorted_words = sorted_words[:self.max_features]

        # Create vocabulary
        self.vocabulary = {word: idx for idx, (word, _) in enumerate(sorted_words)}
        self.idx_to_word = {idx: word for word, idx in self.vocabulary.items()}

        self.is_fitted = True
        return self

    def transform(self, documents: List[str]) -> np.ndarray:
        if not self.is_fitted:
            raise ValueError("Model has not been fit. Call fit() first.")

        n_docs = len(documents)
        n_features = len(self.vocabulary)

        # Initialize BoW matrix
        bow_matrix = np.zeros((n_docs, n_features), dtype=int)

        for doc_idx, doc in enumerate(documents):
            tokens = self._tokenize(doc)
            word_counts = Counter(tokens)

            for word, count in word_counts.items():
                if word in self.vocabulary:
                    word_idx = self.vocabulary[word]
                    bow_matrix[doc_idx, word_idx] = count

        return bow_matrix

    def transform_single(self, document: str) -> np.ndarray:
        """
        Convert a single document to BoW vector
        Args:
            document: A cleaned string
        Returns:
            BoW vector with shape (n_features,)
        """
```

```
        if not self.is_fitted:
            raise ValueError("Model has not been fit.")

        n_features = len(self.vocabulary)
        bow_vector = np.zeros(n_features, dtype=int)

        tokens = self._tokenize(document)
        word_counts = Counter(tokens)

        for word, count in word_counts.items():
            if word in self.vocabulary:
                word_idx = self.vocabulary[word]
                bow_vector[word_idx] = count

        return bow_vector

    def fit_transform(self, documents: List[str]) -> np.ndarray:
        """Fit and transform in one step"""
        return self.fit(documents).transform(documents)

    def get_feature_names(self) -> List[str]:
        """Return the list of words in the vocabulary"""
        if not self.is_fitted:
            raise ValueError("Model has not been fit.")
        return [self.idx_to_word[i] for i in range(len(self.vocabulary))]

    def get_vocabulary_size(self) -> int:
        """Return the size of the vocabulary"""
        return len(self.vocabulary)

    def get_word_frequency(self, word: str) -> int:
        """Return the frequency of a word in the corpus"""
        return self.word_counts.get(word, 0)

    def get_word_index(self, word: str) -> Optional[int]:
        """Return the index of a word in the vocabulary"""
        return self.vocabulary.get(word, None)

    def get_document_tokens(self, document: str) -> List[str]:
        """Return tokens of a document (for debugging)"""
        return self._tokenize(document)

    def print_vocabulary_info(self, top_n: int = 10):
        """Print information about the vocabulary"""
        if not self.is_fitted:
            print("Model has not been fit.")
            return

        print(f"Vocabulary size: {self.get_vocabulary_size()}")
        print(f"Total words in corpus: {sum(self.word_counts.values())}")
        print(f"Unique words in corpus: {len(self.word_counts)}")
        print(f"Top {top_n} most common words:")

        for i, word in enumerate(self.get_feature_names()[:top_n]):
            freq = self.get_word_frequency(word)
            print(f"  {i+1:2d}. '{word:15s}': {freq:4d}")

    def analyze_document(self, document: str):
        """Detailed analysis of a document"""
        if not self.is_fitted:
            raise ValueError("Model has not been fit.")

        tokens = self._tokenize(document)
        word_counts = Counter(tokens)
        bow_vector = self.transform_single(document)

        print(f"Document: '{document[:100]}...' (length: {len(document)})")
        print(f"Tokens: {tokens[:20]}... (total: {len(tokens)})")
        print(f"Unique words: {len(word_counts)}")
        print(f"Words in vocabulary: {sum(1 for w in word_counts if w in self.vocabulary
            )}")
        print(f"BoW vector sum: {bow_vector.sum()}")
```

```
153
154         # Top words in document
155         in_vocab_words = [(w, c) for w, c in word_counts.most_common(10)
156                           if w in self.vocabulary]
157         if in_vocab_words:
158             print("Top words in vocabulary:")
159             for word, count in in_vocab_words:
160                 idx = self.vocabulary[word]
161                 print(f"  '{word}': {count} (index: {idx})")
162
163     def save_vocabulary(self, filepath: str):
164         """Save vocabulary to file"""
165         if not self.is_fitted:
166             raise ValueError("Model has not been fit.")
167
168         with open(filepath, 'w', encoding='utf-8') as f:
169             for word, idx in self.vocabulary.items():
170                 freq = self.get_word_frequency(word)
171                 f.write(f"{word}\t{idx}\t{freq}\n")
172         print(f"Vocabulary has been saved to {filepath}")
173
174     def load_vocabulary(self, filepath: str):
175         """Load vocabulary from file"""
176         self.vocabulary = {}
177         self.idx_to_word = {}
178         self.word_counts = Counter()
179
180         with open(filepath, 'r', encoding='utf-8') as f:
181             for line in f:
182                 parts = line.strip().split('\t')
183                 if len(parts) == 3:
184                     word, idx, freq = parts[0], int(parts[1]), int(parts[2])
185                     self.vocabulary[word] = idx
186                     self.idx_to_word[idx] = word
187                     self.word_counts[word] = freq
188
189         self.is_fitted = True
190         print(f"Vocabulary has been loaded from {filepath}")
```

Lớp `SVDModel`: Lớp thực hiện giảm chiều bằng TruncatedSVD được cài đặt từ đầu bằng phân rã giá trị riêng. Các phương thức bao gồm:

- `__init__`: Khởi tạo với số chiều đầu ra mong muốn (`n_components`).
- `fit(X)`:
  - Tính ma trận hiệp phương sai $X^\top X$ và phân rã thành các trị riêng.
  - Lấy các thành phần chính (`U, S, Vt`) và tỷ lệ phương sai giải thích.
- `transform(X)`: Chiếu dữ liệu lên không gian giảm chiều.
- `fit_transform(X)`: Gộp hai bước `fit` và `transform`.
- `inverse_transform(X_transformed)`: Khôi phục dữ liệu từ không gian giảm chiều.

```
1  class SVDModel:
2      """
3      SVD class for dimensionality reduction of Bag of Words matrices
4      Built from scratch using eigenvalue decomposition
5      """
6      def __init__(self, n_components=100):
7          self.n_components = n_components
8          self.U = None
9          self.s = None  # singular values
10         self.Vt = None
11         self.mean_vec = None
12         self.is_fitted = False
13         self.explained_variance_ratio_ = None
```

```
14
15      def _svd_from_scratch(self, X):
16          """
17          SVD implementation using eigenvalue decomposition.
18          X = U * S * Vt
19          """
20          m, n = X.shape
21
22          # Compute covariance matrix
23          XtX = X.T @ X
24          eigenvals, V = np.linalg.eigh(XtX)
25
26          # Sort eigenvalues and corresponding eigenvectors in descending order
27          sorted_idx = np.argsort(eigenvals)[::-1]
28          eigenvals = eigenvals[sorted_idx]
29          V = V[:, sorted_idx]
30
31          # Remove non-positive eigenvalues
32          positive_mask = eigenvals > 1e-10
33          eigenvals = eigenvals[positive_mask]
34          V = V[:, positive_mask]
35
36          # Compute singular values and U
37          singular_values = np.sqrt(eigenvals)
38          r = len(singular_values)
39          U = np.zeros((m, r))
40          for i in range(r):
41              U[:, i] = (X @ V[:, i]) / singular_values[i]
42
43          # Normalize sign consistency
44          for i in range(r):
45              if U[0, i] < 0:
46                  U[:, i] *= -1
47                  V[:, i] *= -1
48
49          return U, singular_values, V.T
50
51      def fit(self, X):
52          """Fit SVD to input matrix X"""
53          X = np.asarray(X, dtype=np.float64)
54          self.mean_vec = np.mean(X, axis=0)
55          X_centered = X
56
57          self.U, self.s, self.Vt = self._svd_from_scratch(X_centered)
58
59          # Truncate to top n_components
60          if self.n_components < len(self.s):
61              self.U = self.U[:, :self.n_components]
62              self.s = self.s[:self.n_components]
63              self.Vt = self.Vt[:self.n_components, :]
64
65          total_var = np.sum(self.s ** 2)
66          self.explained_variance_ratio_ = (self.s ** 2) / total_var if total_var > 0 else
                np.zeros(len(self.s))
67          self.is_fitted = True
68          return self
69
70      def transform(self, X):
71          """Project input data X to reduced dimension space"""
72          if not self.is_fitted:
73              raise ValueError("SVD must be fitted before transform.")
74          X = np.asarray(X, dtype=np.float64)
75          X_centered = X
76          return X_centered @ self.Vt.T
77
78      def fit_transform(self, X):
79          """Fit and transform input data in one call"""
80          self.fit(X)
81          return self.transform(X)
82
83      def inverse_transform(self, X_transformed):
84          """Reconstruct data from reduced dimension"""
```

```
        if not self.is_fitted:
            raise ValueError("SVD must be fitted before inverse_transform.")
        X_transformed = np.asarray(X_transformed)
        return X_transformed @ self.Vt + self.mean_vec
```

Lớp `BOW_SVD_Embedding`: Đây là pipeline kết hợp giữa Bag-of-Words và SVD để sinh vector biểu diễn tài liệu. Các phương thức chính gồm:

- `__init__`: Khởi tạo pipeline với tham số tùy chỉnh cho BoW (`bow_args`) và SVD (`dim_reduc_args`).
- `fit(documents)`: Học từ điển và giảm chiều từ tập tài liệu đầu vào.
- `transform(documents)`: Chuyển đổi tài liệu sang vector rút gọn đã học.
- `fit_transform(documents)`: Gộp hai bước `fit` và `transform`.
- `get_feature_names()`: Trả về danh sách từ trong từ điển BoW.
- `get_vocabulary_size()`: Trả về kích thước từ điển BoW.

```
class BOW_SVD_Embedding:
    def __init__(
        self,
        bow_args: Optional[Dict] = None,
        dim_reduc_args: Optional[Dict] = None,
    ):
        self.bow_args = bow_args or {}
        self.dim_reduc_args = dim_reduc_args or {}

        self.bow = self._init_bow()
        self.dim_reduc = self._init_dim_reduc()
        self._is_fitted = False

    def _init_bow(self) -> BagOfWords:
        return BagOfWords(
            min_word_freq=self.bow_args.get("min_word_freq", 1),
            max_features=self.bow_args.get("max_features", None),
            tokenizer=self.bow_args.get("tokenizer", "whitespace")
        )

    def _init_dim_reduc(self) -> SVDModel:
        return SVDModel(
            n_components=self.dim_reduc_args.get("n_components", 100)
        )

    def fit(self, documents: Union[List[str], str]) -> "BOW_SVD_Embedding":
        texts = [documents] if isinstance(documents, str) else documents
        bow_matrix = self.bow.fit_transform(texts)
        self.dim_reduc.fit(bow_matrix)
        self._is_fitted = True
        return self

    def transform(self, documents: Union[List[str], str]) -> Union[np.ndarray, np.
        ndarray]:
        if not self._is_fitted:
            raise RuntimeError("Pipeline must be fit before calling transform.")

        is_single = isinstance(documents, str)
        texts = [documents] if is_single else documents

        bow_matrix = self.bow.transform(texts)
        reduced = self.dim_reduc.transform(bow_matrix)
        return reduced[0] if is_single else reduced

    def fit_transform(self, documents: Union[List[str], str]) -> Union[np.ndarray, np.
        ndarray]:
        texts = [documents] if isinstance(documents, str) else documents
        bow_matrix = self.bow.fit_transform(texts)
```

```
        reduced = self.dim_reduc.fit_transform(bow_matrix)
        self._is_fitted = True
        return reduced[0] if isinstance(documents, str) else reduced

    def get_feature_names(self) -> List[str]:
        return self.bow.get_feature_names()

    def get_vocabulary_size(self) -> int:
        return self.bow.get_vocabulary_size()
```

Triển khai code:

- Khởi tạo tập tài liệu:

```
    # Intialize corpus
    corpus = [
        "A father and son bond over a stick up robbery.",
        "An old family foe threatens a young woman's Nashville dreams."
    ]
```

- Tiền xử lí tập tài liệu:

```
    # Initialize preprocessor
    from preprocessing.preprocessing import LSASVDPipeline
    preprocessor = LSASVDPipeline()
    preprocessed_corpus = [preprocessor.preprocess(text) for text in corpus]
    print(preprocessed_corpus)
    # Result
    #['father son bond stick robberi',
    #'old famili foe threaten young woman nashvil dream']
```

- Khai báo model BoW đã chuẩn bị.

```
    from embedding.BoW import BagOfWords
    # Initialize bow model
    bow = BagOfWords(min_word_freq=1, max_features=10, tokenizer='whitespace')
```

- Huấn luyện trên tập corpus đã được tiền xử lí.

```
    # Train
    matrix = bow.fit_transform(preprocessed_corpus)
    print(matrix)
    # array([[1, 1, 1, 1, 1, 0, 0, 0, 0, 0],     #BoW doc1
    #        [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]])   #BoW doc2
```

- Áp dụng TruncatedSVD vào ma trận BoW.

```
    # Initialize SVD
    from embedding.BoW import SVDModel
    my_svd = SVDModel(n_components=2) # We set 2 components
    svd = TruncatedSVD(n_components=2)
    my_X = my_svd.fit_transform(matrix)      # Apply SVD
    print(f'SVD result: \n{my_X}') # Print results
    # SVD result:
    # [[ 0.      2.23606798]
    #  [-2.23606798  0.    ]]
```

4.2.2 TF-IDF

TF-IDF là một phương pháp cải tiến dựa trên BoW . Phương pháp này được thiết kế để đánh giá mức độ quan trọng của một từ trong một tài liệu cụ thể so với toàn bộ tập tài liệu. Về nguyên tắc, TF-IDF cho rằng một từ càng xuất hiện nhiều lần trong một tài liệu thì càng quan trọng với nội dung của tài liệu đó. Tuy nhiên, nếu từ đó lại cũng xuất

hiện phổ biến trong hầu hết các tài liệu, thì tầm quan trọng của nó nên bị giảm đi. Nhờ đó, TF-IDF làm nổi bật các từ mang tính đặc trưng riêng của từng tài liệu, đồng thời làm suy giảm vai trò của các từ phổ biến và ít mang tính phân biệt.

Quá trình xây dựng TF-IDF. Để tạo nên ma trận TF-IDF, trước hết cần tạo một ma trận term-document hoặc document-term, sau đó áp dụng các biến đổi (1), (2), (3) để tạo ra ma trận, cụ thể:

1. Tiền xử lý dữ liệu
2. Tokenize văn bản thành các từ đơn
3. Giới hạn số đặc trưng nhờ vào `max_features` và xây dựng bộ từ vựng.
4. Xây dựng ma trận document-term BoW dựa trên bộ từ vựng. Sau đó:
   (a) Tính TF và IDF
   (b) Tạo ma tận TF-IDF

Ví dụ minh họa. Vì bước 1 và bước 2 khá giống với ở phương pháp BoW, nhóm tác giả sẽ bỏ qua mà tiếp tục ở bước 3.

- **Bước 3: Xây dựng tập từ vựng**

  `"father": 1, "son": 1, ..., "nashvil": 1, "dream": 1`

  Với số lượng đặc trưng tối đa giữ lại `max_features = 10`, các từ được giữ lại bao gồm:

  ```
  vocab = ["father", "son", "bond", "stick", "robberi", "old",
           "famili", "foe", "threaten", "young"]
  ```

- **Bước 4: Xây dựng ma trận TF-IDF**

  Đầu tiên, xây dựng ma trận document-term BoW.

	father	son	bond	stick	robberi	old	famili	foe	threaten	young
doc1	1	1	1	1	1	0	0	0	0	0
doc2	0	0	0	0	0	1	1	1	1	1

  Tiếp theo, tiến hành tính toán tần suất từ (TF).

$$f(\texttt{father, doc1}) = 1, \quad f(\texttt{father, doc2}) = 0$$

$$\Rightarrow \quad tf(\texttt{father, doc1}) = \frac{f(\texttt{father, doc1})}{f(\texttt{father, doc1}) + f(\texttt{father, doc2})} = \frac{1}{1+0} = 1$$

$$tf(\texttt{father, doc2}) = \frac{f(\texttt{father, doc2})}{f(\texttt{father, doc1}) + f(\texttt{father, doc2})} = \frac{0}{1+0} = 0$$

$$
\begin{aligned}
tf(\texttt{son, doc1}) = 1, \quad & tf(\texttt{son, doc2}) = 0 \\
tf(\texttt{bond, doc1}) = 1, \quad & tf(\texttt{bond, doc2}) = 0 \\
tf(\texttt{stick, doc1}) = 1, \quad & tf(\texttt{stick, doc2}) = 0 \\
tf(\texttt{robberi, doc1}) = 1, \quad & tf(\texttt{robberi, doc2}) = 0 \\
tf(\texttt{old, doc1}) = 0, \quad & tf(\texttt{old, doc2}) = 1 \\
tf(\texttt{famili, doc1}) = 0, \quad & tf(\texttt{famili, doc2}) = 1 \\
tf(\texttt{foe, doc1}) = 0, \quad & tf(\texttt{foe, doc2}) = 1 \\
tf(\texttt{threaten, doc1}) = 0, \quad & tf(\texttt{threaten, doc2}) = 1 \\
tf(\texttt{young, doc1}) = 0, \quad & tf(\texttt{young, doc2}) = 1
\end{aligned}
$$

Sau đó, ta tính toán nghịch đảo tài liệu (IDF).

$$
\begin{aligned}
df(\texttt{father}) &= |\{\texttt{doc} \in \texttt{corpus}, \texttt{father} \in \texttt{doc}\}| = 1 \\
\Rightarrow idf(\texttt{father}) &= \log\left(\frac{|\texttt{corpus}|}{df(\texttt{father})}\right) = \log\left(\frac{2}{1}\right) \approx 0.3
\end{aligned}
$$

$$
\begin{aligned}
&idf(\texttt{son}) = \log\left(\frac{2}{1}\right) \approx 0.3, \quad idf(\texttt{bond}) = \log\left(\frac{2}{1}\right) \approx 0.3 \\
&idf(\texttt{stick} = \log\left(\frac{2}{1}\right) \approx 0.3, \quad idf(\texttt{robberi}) = \log\left(\frac{2}{1}\right) \approx 0.3 \\
&idf(\texttt{old}) = \log\left(\frac{2}{1}\right) \approx 0.3, \quad idf(\texttt{famili}) = \log\left(\frac{2}{1}\right) \approx 0.3 \\
&idf(\texttt{foe}) = \log\left(\frac{2}{1}\right) \approx 0.3, \quad idf(\texttt{threaten}) = \log\left(\frac{2}{1}\right) \approx 0.3 \\
&idf(\texttt{young}) = \log\left(\frac{2}{1}\right) \approx 0.3
\end{aligned}
$$

Suy ra, TF-IDF được tính:

$$
\begin{aligned}
tfidf(\texttt{father, doc1}) &= tf(\texttt{father, doc1}) * idf(\texttt{father}) = 0.3 * 1 = 0.3 \\
tfidf(\texttt{father, doc2}) &= tf(\texttt{father, doc2}) * idf(\texttt{father}) = 0.3 * 0 = 0
\end{aligned}
$$

$$
\begin{aligned}
tfidf(\texttt{son, doc1}) = 0.3, \quad & tfidf(\texttt{son, doc2}) = 0 \\
tfidf(\texttt{bond, doc1}) = 0.3, \quad & tfidf(\texttt{bond, doc2}) = 0 \\
tfidf(\texttt{stick, doc1}) = 0.3, \quad & tfidf(\texttt{stick, doc2}) = 0 \\
tfidf(\texttt{robberi, doc1}) = 0.3, \quad & tfidf(\texttt{robberi, doc2}) = 0 \\
tfidf(\texttt{old, doc1}) = 0, \quad & tfidf(\texttt{old, doc2}) = 0.3 \\
tfidf(\texttt{famili, doc1}) = 0, \quad & tfidf(\texttt{famili, doc2}) = 0.3 \\
tfidf(\texttt{foe, doc1}) = 0, \quad & tfidf(\texttt{foe, doc2}) = 0.3 \\
tfidf(\texttt{threaten, doc1}) = 0, \quad & tfidf(\texttt{threaten, doc2}) = 0.3 \\
tfidf(\texttt{young, doc1}) = 0, \quad & tfidf(\texttt{young, doc2}) = 0.3
\end{aligned}
$$

Khi này, ta có ma trận TFIDF.

	father	son	bond	stick	robberi	old	famili	foe	threaten	young
doc1	0.3	0.3	0.3	0.3	0.3	0	0	0	0	0
doc2	0	0	0	0	0	0.3	0.3	0.3	0.3	0.3

Thuật toán.

Algorithm 3 Xây dựng ma trận TF-IDF từ tập tài liệu

procedure BUILDTFIDF(D)
    Tìm tập từ $T = \{t_1, ..., t_n\}$, $N = |D|$
    Tạo ma trận term-document tần suất cơ bản X
    for $i = 1$ **to** n **do**
        Tính $IDF(t_i)$
        for $j = 1$ **to** N **do**
            Tính $TF(t_i, d_j)$
            $X_{i,j} \leftarrow TF(t_i, d_j) \cdot IDF(t_i)$
        end for
    end for
    return X
end procedure

Code minh họa. Tương tự với phần BoW, thuật toán Truncated SVD cũng được trình bày chung thuật toán TFIDF tạo nên thuật toán LSA hoàn chỉnh.

Lớp `TruncatedSVD`: Lớp này khá giống với `SVDModel` ở thuật toán BoW, chỉ có thêm phần tự tìm số thành phần chính.

- `choose_n_components(threshold)`: Chọn số chiều tối ưu để giữ lại tỷ lệ phương sai mong muốn, mặc định là 95%.
- `plot_cumulative_variance()`: Vẽ đồ thị biểu diễn tỷ lệ phương sai tích lũy theo số chiều giữ lại.

```
class TruncatedSVD:
    # The below same as SVDModel class in BoW

    def choose_n_components(self, threshold=0.95):
        if not self.fitted:
            raise ValueError("PCA must be fitted first!")
        cum_var_ratio = np.cumsum(self.explained_variance_ratio_)

        n_components = np.searchsorted(cum_var_ratio, threshold) + 1
        self.n_components = n_components
        self.components_ = self.components_[:n_components]
        return n_components

    def plot_cumulative_variance(self, threshold = 0.95, n_component = None):
        if self.explained_variance_ratio_ is None:
            raise RuntimeError("Must fit first")

        cum_var = np.cumsum(self.explained_variance_ratio_)
        plt.figure(figsize=(20, 10))

        plt.plot(range(1, len(cum_var)+1), cum_var, linestyle='-')
        if threshold is not None:
```

```
            n_component = self.choose_n_components(threshold) if n_component is None
                else n_component
            plt.axvline(x=n_component, color='blue', linestyle='--', label=f'Selected
                Components = {n_component}')
            plt.axhline(y=threshold, color='red', linestyle='--', label=f'Remained
                Information = {threshold * 100}%')
        plt.xlabel("Number of Components")
        plt.ylabel("Cumulative Explained Variance Ratio")
        plt.title("Cumulative Explained Variance by PCA Components")
        plt.grid(True)
        plt.legend()
        plt.show()
```

Lớp `TEmbedder`: Lớp này xử lý đầu vào là các tài liệu văn bản và sinh ra biểu diễn vector rút gọn. Các phương thức bao gồm:

- `__init__`: Người dùng có thể tùy chỉnh số chiều đầu ra (`n_components`), số từ tối đa (`max_features`), và lựa chọn chuẩn hóa (`norm`).
- `fit(documents)`: Xây dựng từ điển (dạng `term` $\times$ `document`) và tính toán IDF cho từng từ. Sau đó, tính toán TF-IDF và chuẩn hóa, cuối cùng áp dụng `TruncatedSVD` để giảm chiều.
- `transform_doc(documents)`: Chuyển đổi danh sách tài liệu sang vector biểu diễn rút gọn.
- `find_best_n_components(threshold, plot)`: Dựa trên phương sai tích lũy để xác định số chiều tối ưu. Biến `plot` là biến nhị phân, dùng để thực hiện việc vẽ lượng thông tin dựa theo số chiều đầu ra.

```
class TEmbedder:
    def __init__(
        self,
        smooth_idf: bool = True,
        norm: Optional[str] = 'l2',
        n_components: Optional[int] = None,
        max_features: Optional[int] = None
    ):
        self.vocab: Dict[str, int] = {}
        self.idf: Dict[str, float] = {}
        self.smooth_idf = smooth_idf
        self.norm = norm  # 'l1', 'l2', or None
        self.max_features = max_features
        self.lsa = TruncatedSVD(n_components)
        self.doc_embeddings: List[np.ndarray] = []
        self.raw_documents: List[str] = []

    def _create_tfidf_matrix(self, documents: List[str]) -> np.ndarray:
        tfidf_matrix = np.zeros((len(documents), len(self.vocab)))
        for i, doc in enumerate(documents):
            tokens = doc.lower().split()
            tf = Counter(tokens)
            doc_len = len(tokens)
            print(tokens)

            for word in tf:
                if word in self.vocab:
                    tf_val = tf[word] / doc_len  #
                    idf_val = self.idf[word]
                    print(f"Word: {word}, TF: {tf_val}, IDF: {idf_val}")
                    tfidf_matrix[i, self.vocab[word]] = tf_val * idf_val

        if self.norm == 'l2':
            norms = np.linalg.norm(tfidf_matrix, axis=1, keepdims=True)
            norms[norms == 0] = 1
            tfidf_matrix = tfidf_matrix / norms
        elif self.norm == 'l1':
```

```
            norms = np.sum(np.abs(tfidf_matrix), axis=1, keepdims=True)
            norms[norms == 0] = 1
            tfidf_matrix = tfidf_matrix / norms
        elif self.norm is None:
            pass
        else:
            raise ValueError(f"Unsupported norm: {self.norm}")
        return tfidf_matrix.T #

    def fit(self, documents: List[str]) -> None:
        N = len(documents)
        df = Counter()

        for doc in documents:
            tokens = set(doc.lower().split())
            for token in tokens:
                df[token] += 1

        if self.max_features is not None:
            most_common = df.most_common(self.max_features)
            vocab_words = [word for word, _ in most_common]
        else:
            vocab_words = list(df.keys())

        self.vocab = {word: idx for idx, word in enumerate(vocab_words)}

        if self.smooth_idf:
            self.idf = {
                word: math.log((1 + N) / (1 + df[word])) + 1
                for word in self.vocab
            }
        else:
            self.idf = {
                word: math.log(N / df[word])
                for word in self.vocab
            }
        print("Self IDF",self.idf)

        tfidf_matrix = self._create_tfidf_matrix(documents)
        print(tfidf_matrix)
        self.lsa.fit(tfidf_matrix)

    def find_best_n_components(self, threshold: float = 0.95, plot: bool= True) -> int:
        best_n = self.lsa.choose_n_components(threshold)
        if plot:
            self.lsa.plot_cumulative_variance(threshold = threshold, n_component=best_n)
        return best_n

    def transform_doc(self, documents: List[str]) -> np.ndarray:
        tfidf_matrix = self._create_tfidf_matrix(documents)
        return self.lsa.transform(tfidf_matrix).T # each row is doc
```

Triển khai thuật toán:

- Tiền xử lý tập tài liệu

```
    # Intialize corpus
    corpus = [
        "A father and son bond over a stick up robbery.",
        "An old family foe threatens a young woman's Nashville dreams."
    ]
    # Initialize preprocessor
    from preprocessing.preprocessing import LSASVDPipeline
    preprocessor = LSASVDPipeline()
    preprocessed_corpus = [preprocessor.preprocess(text) for text in corpus]
    print(preprocessed_corpus)
    # Result
    #['father son bond stick robberi',
    #'old famili foe threaten young woman nashvil dream']
```

- Khai báo, huấn luyện mô hình, và biến đổi.

```
    from embedding.Tfidf import TEmbedder
    embedder = TEmbedder(max_features=10, n_components=None, smooth_idf=False, norm
        =None)
    embedder.fit(docs)
    # TFIDF matrix:
    #[[0.3010 0.3010 0.3010 0.3010 0.3010 0.    0.    0.    0.    0.   ]
    # [0.    0.    0.    0.    0.    0.3010 0.3010 0.3010 0.3010 0.3010]]

    X = embedder.transform_doc(docs)
    print(X.round(4)) # each row stand for each doc
    # result of X: reduce 10 dim to 2 dim (2 dim remains 95% from 10 dim)
    # [[-1.5499  0.    ]
    # [ 0.      -1.5499]]
```

4.2.3 PPMI

PPMI [4] cũng là một phương pháp cải tiến từ BoW. BoW có cách làm triển khai dễ hiểu, tuy nhiên các giá trị thô này thường không phản ánh chính xác mối quan hệ ngữ nghĩa - hai từ có thể xuất hiện nhiều lần mà không thực sự có liên hệ chặt chẽ, và ngược lại, những từ có quan hệ ngữ nghĩa sâu sắc đôi khi lại xuất hiện cùng nhau rất hiếm.

Để khắc phục hạn chế đó, chỉ số PPMI được đưa vào nhằm định lượng mức độ bất ngờ trong việc hai từ cùng xuất hiện, so với kỳ vọng nếu chúng độc lập về mặt thống kê. Áp dụng công thức (8) lên ma trận đồng xuất hiện, ta thu được được ma trận PPMI là một ma trận ngữ nghĩa, trong đó mỗi từ được ánh xạ thành một vector thể hiện mức độ liên kết của nó với toàn bộ các từ còn lại trong từ vựng. Khi này ma trận PPMI có thể thay thế cho ma trận TF-IDF trong việc thực hiện nhiệm vụ phân tích ngữ nghĩa tiềm ẩn.

Quy trình xây dựng ma trận PPMI Để tạo nên ma trận PPMI, cần thực hiện các bước sau:

1. Tiền xử lý dữ liệu.
2. Tokenize văn bản thành các từ đơn.
3. Giới hạn số đặc trưng nhờ vào `max_features` và xây dựng bộ từ vựng.
4. Xây dựng ma trận đồng xuất hiện.
5. Tính xác suất biên và xác suất đồng xuất hiện.
6. Tính giá trị PMI và PPMI, ta được ma trận PPMI.

Ví dụ minh họa. Bước 1, 2 và 3 tương tự với hai phương pháp trên

- **Bước 4: Xây dựng ma trận đồng xuất hiện** Ta xét các cặp từ xuất hiện cùng nhau trong cửa sổ trượt có kích thước ± 2. Ví dụ, với văn bản: `doc1 = ["father", "son", "bond", "stick", "robberi"]`:
  - Với từ `"father"` (vị trí 0), trong cửa sổ `[father, son, bond]`, ta lấy các cặp:
    - (`father`, `son`), (`father`, `bond`)
  - Với từ `"son"` (vị trí 1), cửa sổ `[father, son, bond, stick]`, tạo cặp:
    - (`son`, `father`), (`son`, `bond`), (`son`, `stick`)

– Với từ "bond" (vị trí 2), cửa sổ [father, son, bond, stick, robberi], tạo cặp:

  * (bond, father), (bond, son), (bond, stick), (bond, robberi)

– Với từ "stick" (vị trí 3), cửa sổ [son, bond, stick, robberi], tạo cặp:

  * (stick, son), (stick, bond), (stick, robberi)

– Với từ "robberi" (vị trí 4), cửa sổ [bond, stick, robberi], tạo cặp:

  * (robberi, bond), (robberi, stick)

Tổng hợp các cặp từ xuất hiện từ doc1 (không phân biệt thứ tự trái phải) gồm:

(father, son), (father, bond), (son, bond), (son, stick), (bond, stick), (bond, robberi), (stick, robberi)

Những cặp này sẽ được đếm vào ma trận đồng xuất hiện C. Ta thực hiện tương tự trên các từ con lại ở doc2. Kết quả ma trận đồng xuất hiện C như sau:

Bảng 1: Ma trận đồng xuất hiện C

	father	son	bond	stick	robberi	old	famili	foe	threaten	young
father	0	1	1	0	0	0	0	0	0	0
son	1	0	1	1	0	0	0	0	0	0
bond	1	1	0	1	1	0	0	0	0	0
stick	0	1	1	0	1	0	0	0	0	0
robberi	0	0	1	1	0	0	0	0	0	0
old	0	0	0	0	0	0	1	1	0	0
famili	0	0	0	0	0	1	0	1	1	0
foe	0	0	0	0	0	1	1	0	1	1
threaten	0	0	0	0	0	0	1	1	0	1
young	0	0	0	0	0	0	0	1	1	0

Tổng số lần đồng xuất hiện tính được từ bảng trên là $N = 28$.

- **Bước 5: Tính xác suất biên và xác suất đồng xuất hiện**

Tính xác suất biên:

$$P(\texttt{father}) = \frac{2}{28} = \frac{1}{14}, \quad P(\texttt{son}) = \frac{3}{28}, \quad P(\texttt{bond}) = \frac{1}{7}, \quad P(\texttt{stick}) = \frac{3}{28},$$

$$P(\texttt{robberi}) = \frac{1}{14}, \quad P(\texttt{old}) = \frac{1}{14}, \quad P(\texttt{famili}) = \frac{3}{28},$$

$$P(\texttt{foe}) = \frac{1}{7}, \quad P(\texttt{threaten}) = \frac{3}{28}, \quad P(\texttt{young}) = \frac{1}{14}$$

Tìm xác suất đồng xuất hiện:

Ví dụ:

$$P(\texttt{father}, \texttt{son}) = \frac{1}{28}, \quad P(\texttt{father}, \texttt{bond}) = \frac{1}{28},$$
$$P(\texttt{bond}, \texttt{stick}) = \frac{1}{28}, \quad P(\texttt{foe}, \texttt{young}) = \frac{1}{28}$$

- **Bước 6: Tính PMI và PPMI**

  PMI và PPMI được tính theo công thức:

$$PMI(w_i, w_j) = \log_2\left(\frac{P(w_i, w_j)}{P(w_i) \cdot P(w_j)}\right), \quad PPMI(w_i, w_j) = \max(0, PMI(w_i, w_j))$$

  Ví dụ 1: Tính PMI và PPMI giữa `father` và `son`

$$\begin{aligned} PMI(\texttt{father}, \texttt{son}) &= \log_2\left(\frac{P(\texttt{father}, \texttt{son})}{P(\texttt{father}) \cdot P(\texttt{son})}\right) \\ &= \log_2\left(\frac{1/28}{(1/14) \cdot (3/28)}\right) \\ &= \log_2(\frac{14}{3}) \approx 2.22 \end{aligned}$$

$$\begin{aligned} PPMI(\texttt{father}, \texttt{son}) &= \max(0, PMI(\texttt{father}, \texttt{son})) \\ &= \max(0, 2.22) = 2.22 \end{aligned}$$

  Ví dụ 2: Tính PMI và PPMI giữa `father` và `foe`

$$\begin{aligned} PMI(\texttt{father}, \texttt{foe}) &= \log_2\left(\frac{0}{(1/14) \cdot (1/7)}\right) \\ &= \log_2(0) = -\infty \end{aligned}$$

$$\begin{aligned} PPMI(\texttt{father}, \texttt{foe}) &= \max(0, PMI(\texttt{father}, \texttt{foe})) \\ &= \max(0, -\infty) = 0 \end{aligned}$$

- **Bước 7: Ma trận PPMI**

  Sau khi tính toán tất cả các giá trị PPMI, ta thu được bảng sau:

Bảng 2: Ma trận PPMI

	father	son	bond	stick	robberi	old	famili	foe	threaten	young
father	0	2.22	1.81	0	0	0	0	0	0	0
son	2.22	0	1.22	1.64	0	0	0	0	0	0
bond	1.81	1.22	0	1.22	1.81	0	0	0	0	0
stick	0	1.64	1.22	0	2.22	0	0	0	0	0
robberi	0	0	1.81	2.22	0	0	0	0	0	0
old	0	0	0	0	0	0	2.22	1.81	0	0
famili	0	0	0	0	0	2.22	0	1.22	1.64	0
foe	0	0	0	0	0	1.81	1.22	0	1.22	1.81
threaten	0	0	0	0	0	0	1.64	1.22	0	2.22
young	0	0	0	0	0	0	0	1.81	2.22	0

Thuật toán.

Algorithm 4 Xây dựng ma trận PPMI từ tập tài liệu

procedure BUILDPPMI(D)
    Xây dựng ma trận đồng xuất hiện $\mathbf{C} \in \mathbb{R}^{n \times n}$
    $N \leftarrow \sum_{i,j} \mathbf{C}_{i,j}$
    for $i = 1$ **to** n **do**
        $P(w_i) \leftarrow \sum_j \mathbf{C}_{i,j}/N$
        for $j = 1$ **to** n **do**
            if $\mathbf{C}_{i,j} > 0$ **then**
                $P(w_i, w_j) \leftarrow \mathbf{C}_{i,j}/N$
                $PMI \leftarrow \log_2 \left(\frac{P(w_i,w_j)}{P(w_i) \cdot P(w_j)} \right)$
                $PPMI(w_i, w_j) \leftarrow \max(0, PMI)$
            else
                $PPMI(w_i, w_j) \leftarrow 0$
            end if
        end for
    end for
    return $PPMI$
end procedure

Code minh họa.

Lớp `TruncatedSVD`: Tương tự với lớp cùng tên trong TF-IDF (Mục 4.2.2) nhưng có một chút khác biệt, lớp `TruncatedSVD` ở đây được định nghĩa lại cho phù hợp với PPMI khi sử dụng ma trận đầu vào là ma trận thưa để tối ưu hiệu suất.

Lớp `PPMIEmbedder`: Đây là lớp thực hiện xây dựng mô hình biểu diễn từ dựa trên PPMI và giảm chiều bằng TruncatedSVD. Lớp xử lý đầu vào là danh sách văn bản, đầu ra là biểu diễn từ và văn bản ở dạng vector. Các phương thức bao gồm:

- `__init__`: Khởi tạo mô hình với các tham số như kích thước cửa sổ ngữ cảnh (`window_size`), số lượng từ tối đa (`max_features`), số chiều giảm (`n_components`), tần suất tối thiểu để chọn từ (`min_count`) và số luồng xử lý song song (`n_jobs`).
- `fit(documents)`: Tiền xử lý văn bản và xây dựng từ điển ánh xạ từ → chỉ số. Sau đó, tính trọng số IDF tương ứng, xây dựng ma trận đồng xuất hiện dạng thưa. Cuối cùng, tính toán ma trận PPMI, áp dụng TruncatedSVD để giảm chiều và chuẩn hóa vector biểu diễn từ.
- `transform_docs(documents)`: Chuyển đổi danh sách tài liệu sang vector biểu diễn bằng cách lấy trung bình có trọng số IDF của các vector từ trong văn bản.
- `transform(document)`: Hỗ trợ suy diễn cho một tài liệu đơn hoặc danh sách tài liệu, trả về vector biểu diễn tương ứng.

```
1 class PPMIEmbedder:
2     def __init__(self, window_size=6, max_features=5000, n_components=300, min_count=2,
          n_jobs=4):
3         self.window_size = window_size
4         self.max_features = max_features
5         self.n_components = n_components
6         self.min_count = min_count
7         self.n_jobs = n_jobs
8
```

```
        self.vocab = {}
        self.idf = {}
        self.svd = TruncatedSVD(n_components)
        self.embeddings = None
        self.ppmi_sparse = None

    def _tokenize(self, doc: Union[str, List[str]]) -> List[str]:
        return doc.split() if isinstance(doc, str) else doc

    def _build_vocab(self, docs: List[Union[str, List[str]]]) -> None:
        counter = Counter()
        for doc in docs:
            counter.update(self._tokenize(doc))

        if self.min_count > 1:
            counter = Counter({w: c for w, c in counter.items() if c >= self.min_count})

        most_common = counter.most_common(self.max_features)
        self.vocab = {word: i for i, (word, _) in enumerate(most_common)}
        self._build_idf(docs)

    def _build_idf(self, docs: List[Union[str, List[str]]]) -> None:
        N = len(docs)
        df = Counter()
        for doc in docs:
            tokens = set(self._tokenize(doc))
            for token in tokens:
                if token in self.vocab:
                    df[token] += 1
        self.idf = {w: np.log((N + 1) / (df[w] + 1)) + 1 for w in self.vocab}

    def _build_cooc_worker(self, docs_chunk):
        cooc = defaultdict(float)
        for doc in docs_chunk:
            tokens = self._tokenize(doc)
            token_ids = [self.vocab[t] for t in tokens if t in self.vocab]
            for i, center in enumerate(token_ids):
                start = max(0, i - self.window_size)
                end = min(len(token_ids), i + self.window_size + 1)
                for j in range(start, end):
                    if i != j:
                        context = token_ids[j]
                        weight = 1.0
                        cooc[(center, context)] += weight
        return cooc

    def _merge_cooc(self, cooc_list):
        merged = defaultdict(float)
        for cooc in cooc_list:
            for key, value in cooc.items():
                merged[key] += value
        return merged

    def _build_cooc_sparse_parallel(self, docs):
        chunks = np.array_split(docs, self.n_jobs)
        with ThreadPoolExecutor(max_workers=self.n_jobs) as executor:
            results = list(tqdm(executor.map(self._build_cooc_worker, chunks), total=len
                (chunks), desc="Building Cooc"))
        merged = self._merge_cooc(results)
        rows, cols, data = zip(*[(i, j, v) for (i, j), v in merged.items()])
        size = len(self.vocab)
        return sparse.coo_matrix((data, (rows, cols)), shape=(size, size)).tocsr()

    def _calculate_ppmi(self, cooc_matrix: sparse.csr_matrix, eps=1e-12):
        total_sum = cooc_matrix.sum()
        row_sums = np.array(cooc_matrix.sum(axis=1)).flatten()
        col_sums = np.array(cooc_matrix.sum(axis=0)).flatten()

        coo = cooc_matrix.tocoo()
        p_ij = coo.data / total_sum
        p_i = row_sums[coo.row] / total_sum
        p_j = col_sums[coo.col] / total_sum
```

```
        pmi = np.log2((p_ij + eps) / (p_i * p_j + eps))

        ppmi = np.maximum(0, pmi)
        mask = ppmi > 0
        return sparse.coo_matrix(
            (ppmi[mask], (coo.row[mask], coo.col[mask])),
            shape=cooc_matrix.shape
        ).tocsr()

    def fit(self, docs: List[str]):
        self._build_vocab(docs)
        logging.info(f"Vocab size: {len(self.vocab)}")
        self.cooc_matrix = self._build_cooc_sparse_parallel(docs)
        self.ppmi_sparse = self._calculate_ppmi(self.cooc_matrix)
        self.embeddings = l2_normalize(self.svd.fit_transform(self.ppmi_sparse))

    def transform_docs(self, docs: List[str]) -> np.ndarray:
        dim = self.embeddings.shape[1]
        doc_vectors = np.zeros((len(docs), dim), dtype=np.float32)

        for idx, doc in enumerate(docs):
            tokens = self._tokenize(doc)
            weighted_sum = np.zeros(dim, dtype=np.float32)
            total_weight = 0.0
            for t in tokens:
                if t in self.vocab:
                    vec = self.embeddings[self.vocab[t]]
                    idf_weight = self.idf[t]
                    weighted_sum += vec * idf_weight
                    total_weight += idf_weight

            if total_weight > 0:
                doc_vectors[idx] = weighted_sum / total_weight

        return l2_normalize(doc_vectors)

    def transform(self, doc: Union[str, List[str]]) -> np.ndarray:
        if isinstance(doc, str):
            return self.transform_docs([doc])[0]
        return self.transform_docs(doc)
```

Triển khai code:

- Chuẩn bị văn bản đầu vào.

```
    # Raw corpus
    corpus = [
        "A father and son bond over a stick up robbery",
        "An old family foe threatens a young woman's Nashville dreams"
    ]
```

- Tiền xử lý văn bản:

```
    from preprocessing.preprocessing import LSASVDPipeline

    preprocessor = LSASVDPipeline()
    preprocessed_corpus = [preprocessor.preprocess(text) for text in raw_corpus]

    print("Preprocessed corpus:")
    for i, doc in enumerate(preprocessed_corpus, 1):
        print(f"doc{i} =", doc)

    #doc1 = father son bond stick robberi
    #doc2 = old famili foe threaten young woman nashvil dream
```

- Khởi tạo mô hình PPMI và huấn luyện.

```
    from embedding.ppmi import PPMIEmbedder
    ppmi = PPMIEmbedder(
```

```
        window_size=2, min_count=2, max_features=10, n_components=4, n_jobs=1
    )

    # Fit model on filtered corpus
    ppmi.fit(filtered_corpus)
```

- In ma trận đồng xuất hiện.

```
    vocab_order = ppmi.vocab.keys()
    word2idx = ppmi.vocab
    cooc = ppmi.cooc_matrix.toarray()

    print("\nCo-occurrence Matrix:")
    print("{:>12}".format("") + "".join(f"{w:>12}" for w in vocab_order))
    for w1 in vocab_order:
        i = word2idx[w1]
        row = [f"{cooc[i, word2idx[w2]]:.0f}" for w2 in vocab_order]
        print(f"{w1:>12}" + "".join(f"{v:>12}" for v in row))

    #Co-occurrence Matrix:
    #         father     son    bond   stick robberi    old famili    foe threaten   young
    # father       0       1       1       0       0       0       0       0       0       0
    #    son       1       0       1       1       0       0       0       0       0       0
    #   bond       1       1       0       1       1       0       0       0       0       0
    #  stick       0       1       1       0       1       0       0       0       0       0
    #robberi       0       0       1       1       0       0       0       0       0       0
    #    old       0       0       0       0       0       0       1       1       0       0
    # famili       0       0       0       0       0       1       0       1       1       0
    #    foe       0       0       0       0       0       1       1       0       1       1
    #threaten      0       0       0       0       0       0       1       1       0       1
    #  young       0       0       0       0       0       0       0       1       1       0
```

- Tính toán và in ma trận PPMI.

```
    # Print PPMI matrix
    PPMI = ppmi.ppmi_sparse.toarray()
    print("\nPPMI Matrix:")
    print("{:>8}".format("") + "".join(f"{w:>8}" for w in vocab_order))
    for w1 in vocab_order:
        i = word2idx[w1]
        row = []
        for w2 in vocab_order:
            j = word2idx[w2]
            value = PPMI[i, j]
            row.append(f"{value:.2f}" if value > 0 else "0")
        print(f"{w1:>8}" + "".join(f"{v:>8}" for v in row))

    # PPMI Matrix:
    #          father    son   bond  stick robberi    old famili   foe threate young
    #  father    0.00   2.22   1.81   0.00    0.00   0.00   0.00  0.00    0.00  0.00
    #     son    2.22   0.00   1.22   1.64    0.00   0.00   0.00  0.00    0.00  0.00
    #    bond    1.81   1.22   0.00   1.22    1.81   0.00   0.00  0.00    0.00  0.00
    #   stick    0.00   1.64   1.22   0.00    2.22   0.00   0.00  0.00    0.00  0.00
    # robberi    0.00   0.00   1.81   2.22    0.00   0.00   0.00  0.00    0.00  0.00
    #    old    0.00   0.00   0.00   0.00    0.00   0.00   2.22  1.81    0.00  0.00
    # famili    0.00   0.00   0.00   0.00    0.00   2.22   0.00  1.22    1.64  0.00
    #    foe    0.00   0.00   0.00   0.00    0.00   1.81   1.22  0.00    1.22  1.81
    # threate    0.00   0.00   0.00   0.00    0.00   0.00   1.64  1.22    0.00  2.22
    #   young    0.00   0.00   0.00   0.00    0.00   0.00   0.00  1.81    2.22  0.00
```

4.3 Các phương pháp embedding khác

Các phương pháp embedding khác về cơ bản vẫn là biểu diễn từ vựng thành ma trận số, giống với LSA. Tuy nhiên, nó vẫn có một số khác biệt như sử dụng mạng neuron để huấn luyện mô hình thay vì chỉ là biến đổi cơ bản. Nhóm tác giả giới thiệu tới bốn phương pháp, bao gồm: Word2Vec (Mục 4.3.1), GloVe (Mục 4.3.2), FastTest (Mục 4.3.3), và HellingerPCA (Mục 4.3.4).

4.3.1 Word2Vec

Word2Vec [5] là một kỹ thuật học biểu diễn từ được phát triển nhằm ánh xạ các từ ngữ trong ngôn ngữ tự nhiên thành các vector số thực trong không gian nhiều chiều, sao cho các mối quan hệ ngữ nghĩa và cú pháp giữa các từ được bảo tồn.

Đặt vấn đề. Giả sử ta có một kho văn bản lớn, mục tiêu là biểu diễn mỗi từ w bằng một vector số thực $\mathbf{v}_w \in \mathbb{R}^d$ sao cho các từ có ý nghĩa ngữ nghĩa gần nhau cũng có các vector gần nhau trong không gian.

Khác với các phương pháp truyền thống như PCA chỉ sử dụng thông tin thống kê tuyến tính, Word2Vec tận dụng thông tin ngữ cảnh cục bộ bằng cách học các mối quan hệ đồng xuất hiện giữa các từ trong cửa sổ ngữ cảnh.

Ý tưởng cốt lõi của Word2Vec là:

- Các từ xuất hiện trong ngữ cảnh tương tự sẽ có ý nghĩa gần nhau.
- Mối quan hệ ngữ nghĩa có thể được học thông qua việc tối ưu hóa sự tương đồng giữa các vector từ dựa trên ngữ cảnh.

Mô hình toán học. Trong biến thể đơn giản hóa của Word2Vec, ta sử dụng một ma trận embedding duy nhất $W \in \mathbb{R}^{V \times d}$, trong đó V là kích thước từ vựng và d là số chiều của vector embedding.

Mỗi từ w_i trong từ vựng được biểu diễn bởi dòng thứ i của ma trận W:

$$\mathbf{v}_{w_i} = W[i, :] \in \mathbb{R}^d \tag{20}$$

Với mỗi câu trong kho văn bản, ta xét cửa sổ ngữ cảnh có kích thước c. Đối với từ mục tiêu w_t tại vị trí t, các từ ngữ cảnh là những từ trong khoảng $[t-c, t+c]$ (loại trừ chính w_t).

Hàm mất mát. Thay vì sử dụng softmax phức tạp, mô hình đơn giản hóa này tối ưu hóa trực tiếp khoảng cách giữa vector từ mục tiêu và vector từ ngữ cảnh. Hàm mất mát cho một cặp (từ mục tiêu, từ ngữ cảnh) được định nghĩa là:

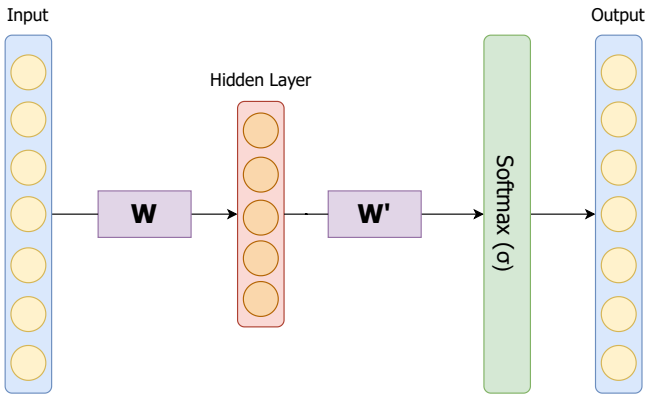
$$\mathcal{L}(w_t, w_c) = \frac{1}{2}\|\mathbf{v}_{w_t} - \mathbf{v}_{w_c}\|^2 \tag{21}$$

trong đó $\mathbf{v}_{w_t}$ và $\mathbf{v}_{w_c}$ lần lượt là vector embedding của từ mục tiêu và từ ngữ cảnh.

Tối ưu hóa. Để tối ưu hóa hàm mất mát (21), ta sử dụng gradient descent. Gradient của hàm mất mát theo vector embedding được tính như sau:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{v}_{w_t}} = \mathbf{v}_{w_t} - \mathbf{v}_{w_c} \tag{22}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{v}_{w_c}} = \mathbf{v}_{w_c} - \mathbf{v}_{w_t} \tag{23}$$



Hình 4: Cấu trúc đơn giản của Word2Vec với một ma trận embedding

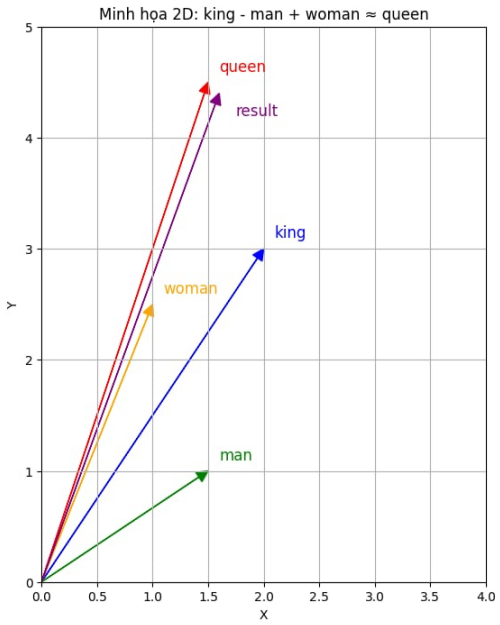
Quy tắc cập nhật với learning rate α:

$$\mathbf{v}_{w_t} \leftarrow \mathbf{v}_{w_t} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{v}_{w_t}} = \mathbf{v}_{w_t} - \alpha(\mathbf{v}_{w_t} - \mathbf{v}_{w_c}) \tag{24}$$

$$\mathbf{v}_{w_c} \leftarrow \mathbf{v}_{w_c} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{v}_{w_c}} = \mathbf{v}_{w_c} - \alpha(\mathbf{v}_{w_c} - \mathbf{v}_{w_t}) \tag{25}$$

Điều này có nghĩa là các vector của từ mục tiêu và từ ngữ cảnh sẽ được kéo gần nhau trong không gian embedding.

Biểu diễn trực quan. Sau khi huấn luyện, các vector từ học được có xu hướng mã hóa thông tin ngữ nghĩa thông qua quan hệ hình học. Một ví dụ kinh điển là phép toán: `king - man + woman` $\approx$ `queen`, cho thấy mô hình học được mối quan hệ giữa giới tính và chức vị trong không gian vector.



Hình 5: Minh hoạ trực quan cho quan hệ `king - man + woman` $\approx$ `queen`. Vectơ màu tím biểu diễn kết quả của phép tính này, gần trùng với vectơ `queen` (màu đỏ).

Quan sát từ Hình 5 cho thấy các mối quan hệ ngữ nghĩa có thể được ánh xạ thành phép tính hình học trong không gian vector học được.

Ví dụ minh họa. Với `corpus` ban đầu được lấy từ bộ dữ liệu IMDB:

```
raw_corpus = [
  "Dog barks at night.",   "Cat sleeps all day."
]
```

- **Bước 1: Sau bước tiền xử lý, ta thu được:**

  `tokens = ["dog", "barks", "night", "cat", "sleeps", "day"]`

- **Bước 2: Khởi tạo ma trận embedding**

  Giả sử embedding dimension $d = 3$. Ma trận $W \in \mathbb{R}^{6\times 3}$ như sau:

$$W = \begin{bmatrix} 0.01 & -0.02 & 0.03 \\ -0.01 & 0.02 & -0.01 \\ 0.02 & 0.01 & -0.02 \\ -0.03 & 0.01 & 0.02 \\ 0.01 & -0.03 & 0.01 \\ 0.00 & 0.02 & -0.03 \end{bmatrix}$$

  Với các dòng tương ứng với các từ: `["dog", "barks", "night", "cat", "sleeps", "day"]`.

- **Bước 3: Huấn luyện với cửa sổ ngữ cảnh**

  Xét câu: `["dog", "barks", "night"]`. Trong ví dụ này, ta chọn kích thước cửa sổ ngữ cảnh (window size) là 1, tức là chỉ xét các từ đứng liền trước và sau từ mục tiêu. Với từ mục tiêu `"barks"` (index 1), các từ ngữ cảnh là `"dog"` (index 0) và `"night"` (index 2).

  Cặp (barks, dog):

$$\begin{aligned} \mathbf{v}_{\text{barks}} &= [-0.01, 0.02, -0.01] \\ \mathbf{v}_{\text{dog}} &= [0.01, -0.02, 0.03] \\ \mathcal{L} &= \frac{1}{2}\|[-0.01, 0.02, -0.01] - [0.01, -0.02, 0.03]\|^2 \\ &= \frac{1}{2}\|[-0.02, 0.04, -0.04]\|^2 = 0.0018 \end{aligned}$$

  Cập nhật với learning rate $\alpha = 0.01$ (ví dụ minh họa, qua thực nghiệm ta sẽ chọn được learning rate tốt hơn):

$$\begin{aligned} \mathbf{v}_{\text{barks}} &\leftarrow \mathbf{v}_{\text{barks}} - \alpha(\mathbf{v}_{\text{barks}} - \mathbf{v}_{\text{dog}}) \\ &= [-0.01, 0.02, -0.01] - 0.01 \cdot [-0.02, 0.04, -0.04] \\ &= [-0.0098, 0.0196, -0.0096] \\ \mathbf{v}_{\text{dog}} &\leftarrow \mathbf{v}_{\text{dog}} - \alpha(\mathbf{v}_{\text{dog}} - \mathbf{v}_{\text{barks}}) \\ &= [0.0098, -0.0196, 0.0296] \end{aligned}$$

  Các cặp còn lại cũng được xử lý tương tự.

- **Bước 4: Kết quả**

  Sau nhiều epoch huấn luyện trên toàn bộ corpus, các vector biểu diễn từ có ngữ cảnh tương tự sẽ gần nhau hơn trong không gian.

Thuật toán. Để huấn luyện mô hình Word2Vec đơn giản hóa, ta sẽ thực hiện như sau:

Algorithm 5 Thuật toán huấn luyện mô hình Word2Vec đơn giản hóa

Require: Tập câu $S = \{s_1, s_2, \ldots, s_N\}$, kích thước cửa sổ ngữ cảnh c, số chiều vector d, tốc độ học α, số vòng lặp T
Ensure: Ma trận embedding $W \in \mathbb{R}^{V \times d}$

Khởi tạo ngẫu nhiên ma trận embedding $W \in \mathbb{R}^{V \times d}$
Xây dựng từ điển ánh xạ word2idx : word $\to \{0, 1, \ldots, V-1\}$
for $t = 1$ **to** T **do**
  for mỗi câu $s \in S$ **do**
    for mỗi từ mục tiêu w_{target} tại vị trí i trong câu s **do**
      target_idx $\leftarrow$ word2idx$[w_{\text{target}}]$
      start $\leftarrow \max(0, i - c)$
      end $\leftarrow \min(|s|, i + c + 1)$
      for mỗi vị trí ngữ cảnh j trong khoảng [start, end] **do**
        if $j \neq i$ và $s[j] \in$ word2idx **then**
          context_idx $\leftarrow$ word2idx$[s[j]]$
          Tính sai số: error $= W[\text{target\_idx}, :] - W[\text{context\_idx}, :]$
          Cập nhật vector embedding:

$$W[\text{target\_idx}, :] \leftarrow W[\text{target\_idx}, :] - \alpha \cdot \text{error}$$
$$W[\text{context\_idx}, :] \leftarrow W[\text{context\_idx}, :] + \alpha \cdot \text{error}$$

        end if
      end for
    end for
  end for
end for
return W

Code minh họa. Đoạn mã sau hiện thực lớp `Word2Vec`, một cài đặt đơn giản của mô hình Word2Vec sử dụng cách cập nhật vector theo lỗi giữa từ trung tâm và từ ngữ cảnh, không sử dụng mạng nơ-ron hay negative sampling. Mô hình bao gồm các thành phần khởi tạo, huấn luyện, lưu/tải mô hình, và truy xuất vector từ.

Các phương thức chính gồm:

- `build_vocab`: xây dựng từ điển ánh xạ giữa từ và chỉ số từ danh sách các câu huấn luyện.
- `fit`: huấn luyện mô hình bằng cách điều chỉnh embedding sao cho vector của từ trung tâm gần với vector của từ ngữ cảnh.
- `get_vector`: trả về vector biểu diễn cho một từ bất kỳ nếu từ đó nằm trong từ điển.

```python
class Word2Vec:
    def __init__(self, embedding_dim=500, window_size=2, learning_rate=0.01):
        self.embedding_dim = embedding_dim
        self.window_size = window_size
        self.learning_rate = learning_rate
        self.word2idx = {}
        self.idx2word = {}
        self.W = None

    def build_vocab(self, sentences):
        vocab = set(word for sent in sentences for word in sent)
        self.word2idx = {w: i for i, w in enumerate(vocab)}
        self.idx2word = {i: w for w, i in self.word2idx.items()}
        vocab_size = len(self.word2idx)
        self.W = np.random.uniform(-0.01, 0.01, (vocab_size, self.embedding_dim))

    def fit(self, sentences, epochs=1):
        if not self.word2idx:
            self.build_vocab(sentences)

        for _ in range(epochs):
            for sent in sentences:
                for idx, target in enumerate(sent):
                    if target not in self.word2idx:
                        continue
                    target_idx = self.word2idx[target]
                    start = max(0, idx - self.window_size)
                    end = min(len(sent), idx + self.window_size + 1)
                    for context_pos in range(start, end):
                        if context_pos == idx:
                            continue
                        context_word = sent[context_pos]
                        if context_word not in self.word2idx:
                            continue
                        context_idx = self.word2idx[context_word]
                        error = self.W[target_idx] - self.W[context_idx]
                        self.W[target_idx] -= self.learning_rate * error
                        self.W[context_idx] += self.learning_rate * error

    def get_vector(self, word):
        if word in self.word2idx:
            return self.W[self.word2idx[word]]
        else:
            return None
```

4.3.2 GloVe

GloVe (Global Vectors for Word Representation) [6] là một thuật toán học không giám sát để thu được biểu diễn vector cho từ, dựa trên thống kê đồng xuất hiện toàn cục trên kho văn bản.

Đặt vấn đề. Giả sử ta có một kho văn bản lớn, mục tiêu là biểu diễn mỗi từ i bằng một vector số thực $\mathbf{w}_i \in \mathbb{R}^d$ sao cho các từ có ý nghĩa ngữ nghĩa gần nhau cũng có các vector gần nhau trong không gian.

Khác với các mô hình chỉ sử dụng thông tin ngữ cảnh cục bộ (như Word2Vec ở Mục 4.3.1), GloVe tận dụng thông tin toàn cục bằng cách xây dựng ma trận đồng xuất hiện X, trong đó mỗi phần tử X_{ij} là số lần từ j xuất hiện trong ngữ cảnh của từ i.

Ý tưởng cốt lõi của GloVe là:

- Các mối quan hệ ngữ nghĩa có thể được mã hóa thông qua **tỷ lệ** giữa các xác suất đồng xuất hiện.

- Ví dụ: từ "solid" thường xuất hiện cùng "ice" hơn là "steam", nên tỷ lệ $P(\text{solid} \mid \text{ice})/P(\text{solid} \mid \text{steam})$ sẽ lớn, phản ánh quan hệ ngữ nghĩa "ice" liên quan đến "solid" hơn là "steam".

Mô hình toán học. GloVe tìm cách xấp xỉ logarit của tần suất đồng xuất hiện qua tích vô hướng giữa các vector:

$$\mathbf{w}_i^\top \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j \approx \log(X_{ij}) \tag{26}$$

Hàm mất mát. Từ mục tiêu Phương trình (26), ta xây dựng hàm mất mát bình phương có trọng số:

$$J = \sum_{i,j=1}^{V} f(X_{ij}) \left(\mathbf{w}_i^\top \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log X_{ij} \right)^2 \tag{27}$$

trong đó $f(X_{ij})$ là hàm trọng số nhằm giới hạn ảnh hưởng của các cặp từ quá phổ biến hoặc quá hiếm:

$$f(x) = \begin{cases} \left(\dfrac{x}{x_{\max}} \right)^\alpha & \text{nếu } x < x_{\max} \\ 1 & \text{ngược lại} \end{cases} \tag{28}$$

Các tham số thường dùng: $\alpha = \frac{3}{4}$, $x_{\max} = 100$.

Tối ưu bằng AdaGrad. Để tối ưu hàm mất mát Phương trình (27), GloVe sử dụng AdaGrad [7], một phương pháp tối ưu thích nghi. Mỗi tham số θ_t được cập nhật như sau:

$$G_t := G_{t-1} + g_t \odot g_t \tag{29}$$

$$\theta_{t+1} := \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t \tag{30}$$

trong đó g_t là gradient tại thời điểm t, và $\odot$ là phép nhân từng phần tử.

Khi xét riêng từng cặp (i, j) sao cho $X_{ij} > 0$, ta có sai số:

$$\text{loss} = \mathbf{w}_i^\top \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log X_{ij} \tag{31}$$

Gradient theo từng tham số được tính như sau:

$$g_{\mathbf{w}_i} = f(X_{ij}) \cdot \text{loss} \cdot \tilde{\mathbf{w}}_j \tag{32}$$

$$g_{\tilde{\mathbf{w}}_j} = f(X_{ij}) \cdot \text{loss} \cdot \mathbf{w}_i \tag{33}$$

$$g_{b_i} = f(X_{ij}) \cdot \text{loss} \tag{34}$$

$$g_{\tilde{b}_j} = f(X_{ij}) \cdot \text{loss} \tag{35}$$

Các gradient trên được sử dụng trong cập nhật AdaGrad ở (29) và (30).

Ví dụ minh họa. Với `corpus` ban đầu được lấy từ bộ dữ liệu IMDB:

```
raw_corpus = [
  "A father and son bond over a stick up robbery.",
  "An old family foe threatens a young woman's Nashville dreams."
]
```

- **Bước 1: Sau bước tiền xử lý, ta thu được:**

```
tokens = ["father", "son", "bond", "stick", "robbery", "old",
 "family", "foe", "threatens", "young", "woman", "nashville",
                              "dream"]
```

- **Bước 2: Xây dựng ma trận đồng xuất hiện X:**

  Ở bước này, nhóm tác giả xây dựng ma trận đồng xuất hiện X bằng cách đếm số lần các cặp từ cùng xuất hiện trong một cửa sổ ngữ cảnh, để dễ hình dung ta sẽ ví dụ nó có kích thước là 2 (tức là mỗi từ được xem là xuất hiện cùng ngữ cảnh với các từ cách nó tối đa 2 vị trí về trước hoặc sau trong câu gốc). Trong thực tế, kích thước này được chọn thông qua thực nghiệm.

  Bảng dưới đây minh họa các mối quan hệ đồng xuất hiện giữa bốn từ: `father`, `son`, `bond`, và `stick`.

	`father`	`son`	`bond`	`stick`
`father`	0	1	1	0
`son`	1	0	1	1
`bond`	1	1	0	1
`stick`	0	1	1	0

- **Bước 3: Áp dụng AdaGrad để tối ưu các phương trình từ (27) đến (35).**

  Kết quả thu được là vector biểu diễn mỗi từ, ví dụ như sau:

```
w_father = [0.21, -0.45, 0.11, ..., -0.82]
w_son = [0.25, -0.39, 0.15, ..., -0.76]
w_bond = [0.23, -0.42, 0.13, ..., -0.79]
```

  Các vector trên gần nhau trong không gian, phản ánh rằng các từ như "`father`", "`son`" và "`bond`" có liên hệ ngữ nghĩa gần gũi.

Thuật toán. Để huấn luyện mô hình GloVe, ta sẽ thực hiện như sau:

Algorithm 6 Thuật toán huấn luyện mô hình GloVe

Require: Ma trận đồng xuất hiện $X \in \mathbb{R}^{V \times V}$, số mũ điều chỉnh trọng số α, ngưỡng tần suất đồng xuất hiện $x_{\max}$, số chiều vector d, số vòng lặp T

Ensure: Vector từ $\mathbf{w}_i$, vector ngữ cảnh $\tilde{\mathbf{w}}_j$, bias b_i, $\tilde{b}_j$ với $i, j = 1, \ldots, V$

Khởi tạo ngẫu nhiên $\mathbf{w}_i, \tilde{\mathbf{w}}_j \in \mathbb{R}^d$, $b_i, \tilde{b}_j \in \mathbb{R}$

for $t = 1$ **to** T **do**

  for mỗi cặp (i, j) sao cho $X_{ij} > 0$ **do**

    Tính trọng số:

$$f(X_{ij}) = \begin{cases} \left(\frac{X_{ij}}{x_{\max}}\right)^{\alpha} & \text{nếu } X_{ij} < x_{\max} \\ 1 & \text{ngược lại} \end{cases}$$

    Tính sai số:

$$\text{loss} = \left(\mathbf{w}_i^{\top} \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log X_{ij}\right)$$

    Tính gradient và cập nhật các tham số (dùng SGD hoặc AdaGrad):

$$\begin{aligned} \mathbf{w}_i &\leftarrow \mathbf{w}_i - \eta \cdot f(X_{ij}) \cdot \text{loss} \cdot \tilde{\mathbf{w}}_j \\ \tilde{\mathbf{w}}_j &\leftarrow \tilde{\mathbf{w}}_j - \eta \cdot f(X_{ij}) \cdot \text{loss} \cdot \mathbf{w}_i \\ b_i &\leftarrow b_i - \eta \cdot f(X_{ij}) \cdot \text{loss} \\ \tilde{b}_j &\leftarrow \tilde{b}_j - \eta \cdot f(X_{ij}) \cdot \text{loss} \end{aligned}$$

  end for

end for

return $\{\mathbf{w}_i + \tilde{\mathbf{w}}_i\}_{i=1}^{V}$

Code minh họa. Đoạn mã sau hiện thực lớp `GloVe`, một cài đặt đầy đủ của mô hình GloVe, bao gồm các thành phần khởi tạo, huấn luyện, lưu/tải mô hình, và ánh xạ văn bản đầu vào thành vector biểu diễn.

Các phương thức chính gồm:

- `fit`: huấn luyện mô hình từ dữ liệu đồng xuất hiện, sử dụng tối ưu hóa AdaGrad.
- `get_embeddings`: trả về vector biểu diễn cuối cùng cho từng từ, bằng tổng của vector từ chính và vector ngữ cảnh tương ứng.
- `from_pretrained`: nạp mô hình đã huấn luyện từ tệp `.pkl` được lưu trước đó, bao gồm từ điển và ma trận embedding.
- `encode`: ánh xạ một văn bản đầu vào thành vector biểu diễn duy nhất bằng cách trung bình các vector từ có trong văn bản.
- `weighting_func`: hiện thực hàm trọng số $f(x)$ nhằm điều chỉnh ảnh hưởng của các cặp từ theo tần suất đồng xuất hiện, giúp mô hình ổn định hơn.

```
class GloVe:
    def __init__(self, vocab_size, embedding_dim=50, x_max=100, alpha=0.75,
        learning_rate=0.05, epochs=100):
        self.vocab_size = vocab_size
        self.embedding_dim = embedding_dim
        self.x_max = x_max
```

```
        self.alpha = alpha
        self.lr = learning_rate
        self.epochs = epochs
        self.word2id = {}
        self.id2word = {}

        # Embedding vectors and biases
        self.W = np.random.randn(vocab_size, embedding_dim) / np.sqrt(vocab_size)
        self.W_context = np.random.randn(vocab_size, embedding_dim) / np.sqrt(vocab_size
            )
        self.b = np.zeros(vocab_size)
        self.b_context = np.zeros(vocab_size)

        # For AdaGrad
        self.gradsq_W = np.ones((vocab_size, embedding_dim))
        self.gradsq_W_context = np.ones((vocab_size, embedding_dim))
        self.gradsq_b = np.ones(vocab_size)
        self.gradsq_b_context = np.ones(vocab_size)

    def weighting_func(self, x):
        if x < self.x_max:
            return (x / self.x_max) ** self.alpha
        else:
            return 1

    def fit(self, cooccur_data):
        # cooccur_data: list of tuples (i, j, x_ij)
        for epoch in range(self.epochs):
            total_loss = 0
            for i, j, x_ij in cooccur_data:
                w_i = self.W[i]
                w_j = self.W_context[j]
                b_i = self.b[i]
                b_j = self.b_context[j]

                weight = self.weighting_func(x_ij)
                inner_prod = np.dot(w_i, w_j)
                cost = inner_prod + b_i + b_j - np.log(x_ij)
                loss = weight * (cost ** 2)
                total_loss += 0.5 * loss

                grad = weight * cost

                # Gradients
                grad_w_i = grad * w_j
                grad_w_j = grad * w_i
                grad_b_i = grad
                grad_b_j = grad

                # AdaGrad update
                self.W[i] -= (self.lr / np.sqrt(self.gradsq_W[i])) * grad_w_i
                self.W_context[j] -= (self.lr / np.sqrt(self.gradsq_W_context[j])) *
                    grad_w_j
                self.b[i] -= (self.lr / np.sqrt(self.gradsq_b[i])) * grad_b_i
                self.b_context[j] -= (self.lr / np.sqrt(self.gradsq_b_context[j])) *
                    grad_b_j

                # Update gradsq
                self.gradsq_W[i] += grad_w_i ** 2
                self.gradsq_W_context[j] += grad_w_j ** 2
                self.gradsq_b[i] += grad_b_i ** 2
                self.gradsq_b_context[j] += grad_b_j ** 2

            print(f"Epoch {epoch+1}/{self.epochs} loss: {total_loss:.4f}")

    def get_embeddings(self):
        # Return sum embeddings of word and context word
        return self.W + self.W_context

    @staticmethod
    def from_pretrained(pickle_path):
        with open(pickle_path, "rb") as f:
```

```
            data = pickle.load(f)

        word2id = data["word2id"]
        id2word = data["id2word"]
        embeddings = data["embeddings"]
        vocab_size = len(word2id)

        model = GloVe(vocab_size=vocab_size, embedding_dim=embeddings.shape[1])
        model.W = embeddings / 2  # assume W + W_context = embeddings
        model.W_context = embeddings / 2
        model.word2id = word2id
        model.id2word = id2word
        model.embeddings = embeddings
        return model

    def encode(self, text):
        tokens = [w.lower() for w in text if isinstance(w, str)]
        vectors = [self.embeddings[self.word2id[w]] for w in tokens if w in self.word2id
            ]
        if vectors:
            return np.mean(vectors, axis=0)
        else:
            return np.zeros(self.embeddings.shape[1])
```

4.3.3 FastText

Đặt vấn đề. FastText [8], được phát triển bởi nhóm nghiên cứu tại Facebook AI Research, là một kỹ thuật học biểu diễn từ tiên tiến, mở rộng từ mô hình Word2Vec. Khác với Word2Vec, vốn coi mỗi từ như một đơn vị độc lập, FastText phân tích từ thành tập hợp các n-gram ký tự (subword), cho phép mô hình học được thông tin hình thái học (morphological information) và xử lý hiệu quả các từ hiếm, từ ngoài từ vựng (out-of-vocabulary – OOV), cũng như các ngôn ngữ có cấu trúc hình thái học phức tạp như tiếng Việt, tiếng Đức, tiếng Ả Rập, hoặc tiếng Thổ Nhĩ Kỳ.

Ý tưởng cốt lõi của FastText là ý nghĩa và cấu trúc của từ không chỉ nằm ở từ như một đơn vị hoàn chỉnh mà còn ở các thành phần bên trong, chẳng hạn như tiền tố, hậu tố, hoặc gốc từ. Ví dụ, trong tiếng Việt, các từ như "chơi", "đang chơi", "người chơi", "sự chơi đùa" chia sẻ gốc từ "chơi". FastText học được mối liên hệ hình thái học giữa chúng bằng cách phân tích từ thành các n-gram ký tự, giúp cải thiện khả năng biểu diễn các từ hiếm và từ OOV.

Hướng tiếp cận. FastText dựa trên giả thuyết phân phối của Firth, nhưng mở rộng bằng cách kết hợp thông tin subword. Ví dụ, trong tiếng Việt, từ "người chơi" và "cầu thủ" có thể xuất hiện trong ngữ cảnh tương tự (liên quan đến thể thao), nhưng nhờ n-gram, FastText còn học được rằng "người chơi" và "người học" chia sẻ thành phần "người", phản ánh mối liên hệ hình thái học.

FastText sử dụng n-gram ký tự với độ dài từ $n_{\min}$ đến $n_{\max}$, các siêu tham số này sẽ được chọn thông qua thực nghiệm. Ta sẽ ví dụ với các giá trị thường được chọn là $n_{\min} = 3$ vả $n_{\max} = 6$) như sau:

- Với từ "where" và $n = 3$: `<wh, whe, her, ere, re>`, trong đó < và > biểu thị ranh giới từ.
- Với từ "playing" và $n = 3$: `<pl, pla, lay, ayi, yin, ing, ng>`.
- Trong tiếng Việt, với từ "người chơi": `<ng, ngu, guo, uoi, oi , ch, cho, hoi, oi>`.

Phương pháp này cho phép FastText học được cấu trúc hình thái học, chẳng hạn như tiền tố ("un-", "đang-"), hậu tố ("-ing", "-er"), hoặc gốc từ ("play", "chơi"). Đối với từ OOV, FastText tạo vector bằng cách tổng hợp các vector của các n-gram có trong từ đó. Ví dụ, từ "playful" (nếu không có trong từ vựng huấn luyện) được biểu diễn dựa trên các n-gram như `pla, lay, ayf, yfu, ful`.

Mô hình toán học. Giả sử từ vựng có kích thước V. Mỗi từ w được biểu diễn bằng tập hợp các n-gram ký tự $\mathcal{G}_w = \{g_1, g_2, \ldots, g_{|\mathcal{G}_w|}\}$ với độ dài từ $n_{\min}$ đến $n_{\max}$. Mỗi n-gram g_i được ánh xạ thành một vector $\mathbf{z}_{g_i} \in \mathbb{R}^d$, với d là số chiều của không gian biểu diễn (thường từ 100 đến 300). Vector biểu diễn của từ w được tính bằng trung bình cộng của các vector n-gram:

$$\mathbf{v}_w = \frac{1}{|\mathcal{G}_w|} \sum_{g \in \mathcal{G}_w} \mathbf{z}_g \tag{36}$$

FastText sử dụng kiến trúc Skip-gram, dự đoán các từ ngữ cảnh xung quanh từ trung tâm w_t trong một cửa sổ ngữ cảnh có kích thước c:

$$P(w_o \mid w_t) = \frac{\exp(\mathbf{v}_{w_t}^\top \mathbf{v}'_{w_o})}{\sum_{w' \in V} \exp(\mathbf{v}_{w_t}^\top \mathbf{v}'_{w'})} \tag{37}$$

trong đó $\mathbf{v}'_{w_o} \in \mathbb{R}^d$ là vector ngữ cảnh của từ w_o, và V là tập từ vựng.

Hàm mất mát. Để giảm chi phí tính toán khi từ vựng lớn, FastText sử dụng negative sampling. Hàm mất mát cho một cặp từ trung tâm w_t và từ ngữ cảnh w_o là:

$$\mathcal{L} = -\log \sigma\left(\mathbf{v}'^\top_{w_o} \mathbf{v}_{w_t}\right) - \sum_{i=1}^{k} \log \sigma\left(-\mathbf{v}'^\top_{w_i^-} \mathbf{v}_{w_t}\right) \tag{38}$$

trong đó:

- $\mathbf{v}_{w_t} \in \mathbb{R}^d$: vector biểu diễn từ trung tâm, tính từ (36).
- $\mathbf{v}'_{w_o} \in \mathbb{R}^d$: vector biểu diễn từ ngữ cảnh đúng.
- $\mathbf{v}'_{w_i^-} \in \mathbb{R}^d$: vector của các từ âm, chọn ngẫu nhiên theo phân phối xác suất tỷ lệ nghịch với tần suất từ.
- $\sigma(x) = \frac{1}{1+\exp(-x)}$: hàm sigmoid.
- k: số lượng negative samples, thường từ 5 đến 20.

Hàm mất mát được tối ưu hóa bằng Stochastic Gradient Descent (SGD). Gradient được tính như sau:

$$g_{\mathbf{z}_g} = \left[\sigma(\mathbf{v}_{w_t}^\top \mathbf{v}'_{w_o}) - 1\right] \mathbf{v}'_{w_o} + \sum_{i=1}^{k} \sigma(\mathbf{v}_{w_t}^\top \mathbf{v}'_{w_i^-}) \mathbf{v}'_{w_i^-}, \quad \forall g \in \mathcal{G}_{w_t} \tag{39}$$

$$g_{\mathbf{v}'_{w_o}} = \left[\sigma(\mathbf{v}_{w_t}^\top \mathbf{v}'_{w_o}) - 1\right] \mathbf{v}_{w_t} \tag{40}$$

$$g_{\mathbf{v}'_{w_i^-}} = \sigma(\mathbf{v}_{w_t}^\top \mathbf{v}'_{w_i^-}) \mathbf{v}_{w_t} \tag{41}$$

Cập nhật tham số bằng SGD:

$$\theta_{t+1} = \theta_t - \eta \cdot g_t \tag{42}$$

Ngoài Skip-gram, FastText cũng hỗ trợ kiến trúc CBOW, nhưng Skip-gram thường được sử dụng do hiệu quả cao hơn trong việc học biểu diễn ngữ nghĩa.

Ví dụ minh họa. Xét kho văn bản:

```
raw_corpus = [
  "A father and son bond over a stick up robbery.",
  "An old family foe threatens a young woman's Nashville dreams."
]
```

- **Bước 1: Tiền xử lý, thu được:**

  Áp dụng các bước tiền xử lý như chuyển về chữ thường, loại bỏ dấu câu và dừng từ (stopwords), sau đó tách từ. Kết quả thu được danh sách token đầu vào:

  ```
  tokens = ["father", "son", "bond", "stick", "robbery", "old",
  "family", "foe", "threatens", "young", "woman", "nashville",
  "dream"]
  ```

- **Bước 2: Tạo n-gram ký tự cho từng từ:**

  Với FastText, mỗi từ được biểu diễn không chỉ bởi chính nó mà còn bởi các n-gram ký tự con, nhằm xử lý tốt hơn với từ hiếm hoặc từ chưa thấy. Để dễ hình dung, ta giả sử $n = 3$ (n-gram 3 ký tự), dưới đây là một số ví dụ trong trường hợp này, với ký hiệu `< >` bao quanh từ thể hiện ký tự kết thúc hoặc bắt đầu của từ:

  - Từ "father": `<fa, fat, ath, the, her, er>`.
  - Từ "son": `<so, son, on>`.

  Những n-gram này sẽ được ánh xạ sang các vector, và vector của một từ là tổng trung bình của vector chính từ đó và các vector n-gram thành phần.

- **Bước 3: Tạo các cặp từ trung tâm và ngữ cảnh:**

  FastText sử dụng cơ chế Skip-Gram tương tự Word2Vec để dự đoán các từ ngữ cảnh dựa trên từ trung tâm. Để dễ hình dung, ta giả sử lấy kích thước cửa sổ ngữ cảnh là 2, ta có:

  - Câu: "father and son bond" $\Rightarrow$ chuỗi từ quan trọng là: `["father", "son", "bond"]`.
  - Từ trung tâm: "father" $\rightarrow$ ngữ cảnh: ["son"].
  - Từ trung tâm: "son" $\rightarrow$ ngữ cảnh: ["father", "bond"].
  - Từ trung tâm: "bond" $\rightarrow$ ngữ cảnh: ["son"].

  Mỗi cặp (từ trung tâm, từ ngữ cảnh) sẽ được dùng để cập nhật vector thông qua hàm mất mát.

- **Bước 4: Tối ưu hàm mất mát (38) bằng Stochastic Gradient Descent.**

Để tối ưu hàm mất mát, ta sử dụng các gradient từ các phương trình (39), (40) và (41). Các cặp (từ trung tâm, ngữ cảnh) được dùng để tối ưu hàm mất mát Skip-Gram với Negative Sampling. Quá trình học diễn ra qua nhiều epoch, sử dụng thuật toán SGD và tính đạo hàm theo các phương trình đã nêu.

- **Bước 5: Vector biểu diễn thu được:**

  Sau khi huấn luyện, mỗi từ sẽ có một vector biểu diễn riêng. Một số ví dụ:

```
w_father = [0.18, -0.50, 0.09, ..., -0.85]
w_son = [0.22, -0.47, 0.12, ..., -0.80]
w_bond = [0.20, -0.48, 0.10, ..., -0.82]
```

  Các vector này gần nhau trong không gian biểu diễn, phản ánh rằng "`father`", "`son`" và "`bond`" có mối liên hệ ngữ nghĩa gần gũi.

Thuật toán. Để huấn luyện mô hình FastText, ta sẽ thực hiện như sau:

Algorithm 7 Thuật toán huấn luyện mô hình FastText

Require: Kho văn bản, kích thước cửa sổ ngữ cảnh C, số n-gram $n_{\min}$ đến $n_{\max}$, số chiều vector d, số vòng lặp T, số mẫu âm k, tốc độ học η

Ensure: Vector n-gram $\mathbf{z}_g$, vector ngữ cảnh $\mathbf{v}'_c$ với $g \in \mathcal{G}$, $c \in V$

Tiền xử lý kho văn bản, thu được tập từ vựng V và tập n-gram $\mathcal{G}$
Khởi tạo ngẫu nhiên $\mathbf{z}_g, \mathbf{v}'_c \in \mathbb{R}^d$ cho mọi $g \in \mathcal{G}$, $c \in V$
for $t = 1$ **to** T **do**
  for mỗi từ trung tâm w_t trong kho văn bản **do**
    Tính vector từ trung tâm: $\mathbf{v}_{w_t} = \frac{1}{|\mathcal{G}_{w_t}|} \sum_{g \in \mathcal{G}_{w_t}} \mathbf{z}_g$
    for mỗi từ ngữ cảnh w_o trong cửa sổ kích thước C **do**
      Lấy mẫu k từ âm $w_1^-, w_2^-, \ldots, w_k^-$ từ phân phối P_n
      Tính hàm mất mát: $\mathcal{L} = -\log \sigma\left(\mathbf{v}'^{\top}_{w_o} \mathbf{v}_{w_t}\right) - \sum_{i=1}^{k} \log \sigma\left(-\mathbf{v}'^{\top}_{w_i^-} \mathbf{v}_{w_t}\right)$
      Tính gradient:

$$
\begin{aligned}
g_{\mathbf{z}_g} &= \left[\sigma(\mathbf{v}_{w_t}^{\top} \mathbf{v}'_{w_o}) - 1\right] \mathbf{v}'_{w_o} + \sum_{i=1}^{k} \sigma(\mathbf{v}_{w_t}^{\top} \mathbf{v}'_{w_i^-}) \mathbf{v}'_{w_i^-}, \quad \forall g \in \mathcal{G}_{w_t} \\
g_{\mathbf{v}'_{w_o}} &= \left[\sigma(\mathbf{v}_{w_t}^{\top} \mathbf{v}'_{w_o}) - 1\right] \mathbf{v}_{w_t} \\
g_{\mathbf{v}'_{w_i^-}} &= \sigma(\mathbf{v}_{w_t}^{\top} \mathbf{v}'_{w_i^-}) \mathbf{v}_{w_t}
\end{aligned}
$$

      Cập nhật tham số bằng SGD: $\theta \leftarrow \theta - \eta \cdot g_\theta \quad$ với $\theta \in \{\mathbf{z}_g, \mathbf{v}'_{w_o}, \mathbf{v}'_{w_i^-}\}$
  end for
end for
return $\{\mathbf{v}_{w_i} = \frac{1}{|\mathcal{G}_{w_i}|} \sum_{g \in \mathcal{G}_{w_i}} \mathbf{z}_g\}_{i=1}^{V}$

Code minh họa. Đoạn mã sau hiện thực lớp `FastText`, một cài đặt đơn giản của mô hình FastText, bao gồm các thành phần khởi tạo, xây dựng tập từ vựng, sinh cặp huấn luyện, huấn luyện mô hình và truy xuất vector biểu diễn. Các phương thức chính gồm:

- `build_vocab`: xây dựng từ điển và ánh xạ chỉ số từ dữ liệu văn bản, lọc theo tần suất tối thiểu (min_count).
- `generate_training_pairs`: sinh ra các cặp (center, context) từ cửa sổ ngữ cảnh, dùng làm dữ liệu huấn luyện.
- `train`: huấn luyện mô hình với thuật toán Skip-gram kết hợp Negative Sampling, cập nhật vector đầu vào và đầu ra qua nhiều epoch.
- `sigmoid`: hàm kích hoạt sử dụng trong tính toán xác suất, được ổn định bằng clipping giá trị đầu vào và đầu ra.
- `get_vector`: trả về vector biểu diễn cuối cùng cho một từ nhất định, hoặc vector 0 nếu từ không có trong từ điển.

```
class FastText:
    def __init__(self, vector_size=50, window_size=2, epochs=5, lr=0.01, min_count=1,
        neg_samples=5):
        self.vector_size = vector_size
        self.window_size = window_size
        self.epochs = epochs
        self.lr = lr
        self.min_count = min_count
        self.neg_samples = neg_samples
        self.word2idx = {}
        self.idx2word = {}
        self.vocab = []
        self.input_vectors = None
        self.output_vectors = None

    def build_vocab(self, corpus: List[List[str]]):
        if not corpus or not any(corpus):
            raise ValueError("Corpus is empty or contains no valid sentences.")
        word_freq = Counter(w for sentence in corpus for w in sentence)
        self.vocab = [w for w, c in word_freq.items() if c >= self.min_count]
        if not self.vocab:
            raise ValueError("Vocabulary is empty after applying min_count.")
        self.word2idx = {w: i for i, w in enumerate(self.vocab)}
        self.idx2word = {i: w for w, i in self.word2idx.items()}

    def generate_training_pairs(self, corpus: List[List[str]]):
        pairs = []
        for sentence in corpus:
            for i, center in enumerate(sentence):
                if center not in self.word2idx:
                    continue
                for j in range(max(0, i - self.window_size), min(len(sentence), i + self
                    .window_size + 1)):
                    if i != j and sentence[j] in self.word2idx:
                        pairs.append((self.word2idx[center], self.word2idx[sentence[j]])
                            )
        if not pairs:
            raise ValueError("No valid training pairs generated.")
        return pairs

    def sigmoid(self, x):
        x = np.clip(x, -500, 500)
        return np.clip(1 / (1 + np.exp(-x)), 1e-15, 1 - 1e-15)

    def train(self, corpus: List[List[str]]):
        self.build_vocab(corpus)
        vocab_size = len(self.vocab)
        self.input_vectors = np.random.uniform(-0.01, 0.01, (vocab_size, self.
            vector_size))
        self.output_vectors = np.random.uniform(-0.01, 0.01, (vocab_size, self.
            vector_size))
        training_pairs = self.generate_training_pairs(corpus)
        print(f"Generated {len(training_pairs):,} training pairs.")
        losses = []
```

```
        for epoch in tqdm(range(self.epochs), desc="Training FastText"):
            np.random.shuffle(training_pairs)
            loss = 0
            for center, context in training_pairs:
                v_in = self.input_vectors[center]
                v_out = self.output_vectors[context]
                score = self.sigmoid(np.dot(v_in, v_out))
                grad = self.lr * (1 - score)
                self.input_vectors[center] += grad * v_out
                self.output_vectors[context] += grad * v_in
                # Normalize vectors
                self.input_vectors[center] /= np.linalg.norm(self.input_vectors[center])
                    + 1e-10
                self.output_vectors[context] /= np.linalg.norm(self.output_vectors[
                    context]) + 1e-10
                # Negative sampling
                neg_indices = np.random.choice(vocab_size, self.neg_samples)
                for neg in neg_indices:
                    v_neg = self.output_vectors[neg]
                    score_neg = self.sigmoid(np.dot(v_in, v_neg))
                    grad_neg = self.lr * score_neg
                    self.input_vectors[center] -= grad_neg * v_neg
                    self.output_vectors[neg] -= grad_neg * v_in
                    self.input_vectors[center] /= np.linalg.norm(self.input_vectors[
                        center]) + 1e-10
                    self.output_vectors[neg] /= np.linalg.norm(self.output_vectors[neg])
                        + 1e-10
                    loss += -np.log(1 - score_neg + 1e-15)
                loss += -np.log(score + 1e-15)
            avg_loss = loss / len(training_pairs)
            losses.append(avg_loss)
            print(f"Epoch {epoch+1}/{self.epochs}, Loss: {avg_loss:.4f}")
        return losses

    def get_vector(self, word: str) -> np.ndarray:
        if word in self.word2idx:
            return self.input_vectors[self.word2idx[word]]
        print(f"Warning: Word '{word}' not in vocabulary, returning zero vector.")
        return np.zeros(self.vector_size)
```

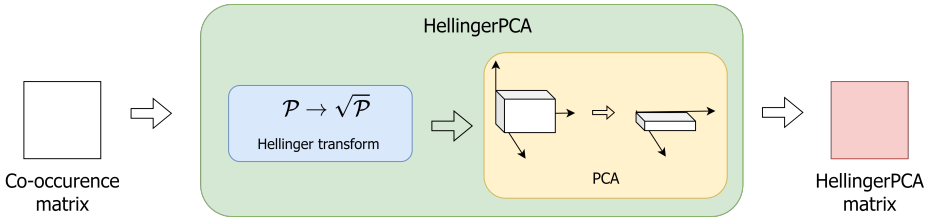
4.3.4 HellingerPCA

HellingerPCA [9] là một phương pháp đơn giản, có phần giống LSA, được tạo ra để đọ sức với các thuật toán sử dụng kiến thức học sâu như FastText, GloVe, Word2Vec.

Đặt vấn đề. Trong các biểu diễn truyền thống, thường cho ra các vector thưa, có tổng bằng 1, và tồn tại trong một không gian simplex. Tuy nhiên, khi áp dụng một số thuật toán gom cụm như KMeans hoặc PCA, kết quả thu được thường không phản ánh đúng mức độ tương đồng về ngữ nghĩa giữa các văn bản, do khoảng cách Euclidean không phù hợp trong không gian simplex.

Từ vấn đề trên, HellingerPCA được đề xuất như một giải pháp hiệu quả và dễ hiểu. Phương pháp này kết hợp biến đổi Hellinger, giúp đưa các vector phân phối xác suất từ không gian simplex sang không gian Euclidean với phân tích thành phần chính (PCA), nhằm rút trích đặc trưng toàn cục có phương sai lớn nhất. HellingerPCA vừa đảm bảo tính tương thích với các thuật toán tuyến tính, vừa giữ lại được nhiều thông tin ngữ nghĩa quan trọng trong dữ liệu. Điều đáng chú ý là, dù đơn giản, phương pháp này có thể cạnh tranh đáng kể với các phương pháp học sâu trong một số bài toán cụ thể.

Xây dựng ma trận HellingerPCA



Hình 6: Quá trình xây dựng ma trận HellingerPCA

1. Tiền xử lý dữ liệu
2. Tokenize văn bản thành các từ đơn
3. Giới hạn số đặc trưng nhờ vào `max_features` và xây dựng bộ từ vựng.
4. Xây dựng ma trận co-occurence dựa theo cửa sổ ngữ cảnh. Sau đó:
   (a) Áp dụng biến đổi Hellinger
   (b) Giảm chiều bằng PCA

Ví dụ minh họa.

- **Bước 1 và bước 2: Tiền xử lý văn bản và token hóa:**

  Thực hiện xong có được bộ từ vựng:

  ```
  tokens = ["father", "son", "bond", "stick", "robbery", "old",
   "family", "foe", "threatens", "young", "woman", "nashville",
                              "dream"]
  ```

- **Bước 3: Giữ lại số từ vựng là `max_feature = 4`**

  ```
  tokens = ["father", "son", "bond", "stick"]
  ```

- **Bước 4: Xây dựng ma trận co-occurences**

  Với cửa sổ ngữ cảnh là 2, ta có ma trận:

	father	son	bond	stick
father	0	1	1	0
son	1	0	1	1
bond	1	1	0	1
stick	0	1	1	0

  Tiếp theo, tính xác suất xuất hiện

$$P(\texttt{father}|\texttt{father}) = \frac{f(\texttt{father}|\texttt{father})}{\sum_i^{\texttt{max\_features}} f(\texttt{father}|\texttt{tokens}_i)} = 0$$

	father	son	bond	stick
father	0	1/2	1/2	0
son	1/3	0	1/3	1/3
bond	1/3	1/3	0	1/3
stick	0	1/2	1/2	0

Khi này áp dụng biến đổi Hellinger, được

	father	son	bond	stick
father	0	0.71	0.71	0
son	0.58	0	0.58	0.58
bond	0.58	0.58	0	0.58
stick	0	0.71	0.71	0

Cuối cùng, áp dụng PCA với số thành phần chính là 2, ta được

father	−0.5042	−2.13e-17
son	0.5042	−0.4082
bond	0.5042	0.4082
stick	−0.5042	−2.13e-17

Thuật toán.

Algorithm 8 Tính biểu diễn từ bằng HellingerPCA từ tập tài liệu

procedure HellingerPCAEmbed(D, `window_size`, `max_features`, k)
  Tiền xử lý và tách mỗi văn bản $d \in D$ thành chuỗi từ đơn
  Xây dựng bộ từ vựng V từ top `max_features` từ phổ biến nhất
  Tạo ma trận đồng xuất hiện $X \in \mathbb{R}^{|V|\times|V|}$ dựa vào `window_size`
  Áp dụng biến đổi Hellinger: $X_{ij} \leftarrow \sqrt{\frac{X_{ij}}{\sum_k X_{ik}}}$
  Thực hiện PCA để giảm chiều X thành $X' \in \mathbb{R}^{|V|\times k}$
  return Ma trận embedding X'
end procedure

Code minh họa.

Lớp `HPEmbedder`: Đây là lớp thực hiện phương pháp HellingerPCA, một kết hợp giữa biến đổi Hellinger và phân tích thành phần chính (PCA) áp dụng trên ma trận đồng xuất hiện. Các phương thức chính bao gồm:

- `__init__(n_components, window_size, max_features)`: Khởi tạo lớp với số chiều PCA mong muốn, kích thước cửa sổ đồng xuất hiện và số lượng đặc trưng tối đa.
- `fit(docs)`: Huấn luyện mô hình với tập văn bản đầu vào. Các bước bao gồm:
  - Chuẩn hóa văn bản thành các token.
  - Xây dựng ma trận đồng xuất hiện (co-occurrence).

  - Áp dụng biến đổi Hellinger.
  - Thực hiện PCA để thu được vector biểu diễn từ vựng.

- `transform_word(docs)`: Trả về ma trận nhúng (embedding) của các từ trong tập văn bản đầu vào, sử dụng từ điển đã học.
- `transform_doc(docs)`: Biểu diễn từng văn bản bằng trung bình các vector từ trong văn bản đó. Đầu vào có thể là chuỗi hoặc list từ.
- `find_best_n_components(threshold, plot)`: Xác định số thành phần chính cần giữ lại để đảm bảo tỷ lệ phương sai tích lũy lớn hơn ngưỡng `threshold`. Có thể vẽ biểu đồ minh họa.

```
class HPEmbedder:
    def __init__(self, n_components: int = None, window_size: int = 2, max_features: int = None):
        self.n_components = n_components
        self.window_size = window_size
        self.max_features = max_features
        self.vocab = None
        self.pca = PCA(n_components=n_components)
        self.embeddings = None

    def _build_cooccurrence_matrix(self, docs: List[Union[str, List[str]]], vocab: dict = None):
        tokenized_docs = []
        for doc in docs:
            if isinstance(doc, str):
                tokenized_docs.append(doc.split())
            elif isinstance(doc, list):
                tokenized_docs.append(doc)
            else:
                raise ValueError("Each document must be either a string or a list of tokens.")

        if vocab is None:
            word_counts = Counter()
            for tokens in tokenized_docs:
                word_counts.update(tokens)

            if self.max_features is not None:
                most_common = word_counts.most_common(self.max_features)
                vocab = {word: idx for idx, (word, _) in enumerate(most_common)}
            else:
                vocab = {word: idx for idx, word in enumerate(word_counts.keys())}
        else:
            vocab = vocab.copy()

        vocab_size = len(vocab)
        cooc_matrix = dok_matrix((vocab_size, vocab_size), dtype=np.float32)

        for tokens in tqdm(tokenized_docs, desc="Building co-occurrence matrix..."):
            for i, word in enumerate(tokens):
                if word not in vocab:
                    continue
                word_idx = vocab[word]
                start = max(0, i - self.window_size)
                end = min(len(tokens), i + self.window_size + 1)
                for j in range(start, end):
                    if i != j and tokens[j] in vocab:
                        neighbor_idx = vocab[tokens[j]]
                        cooc_matrix[word_idx, neighbor_idx] += 1

        self.vocab = vocab
        cooc_matrix = cooc_matrix.toarray()
        return cooc_matrix / cooc_matrix.sum(axis=1, keepdims=True)

    def _hellinger_transform(self, X: np.ndarray) -> np.ndarray:
```

```
        X = X.astype(np.float64)
        X_sum = X.sum(axis=1, keepdims=True)
        X_sum[X_sum == 0] = 1
        X_norm = X / X_sum
        return np.sqrt(X_norm)

    def fit(self, docs: List[str]):
        cooc_matrix = self._build_cooccurrence_matrix(docs)
        X_hel = self._hellinger_transform(cooc_matrix)
        self.embeddings = self.pca.fit_transform(X_hel)

    def find_best_n_components(self, threshold: float = 0.95, plot: bool= True) -> int:
        best_n = self.pca.choose_n_components(threshold)
        if plot:
            self.pca.plot_cumulative_variance(threshold = threshold, n_component=best_n)
        return best_n
    # Embed words
    def transform_word(self, docs: List[str]) -> np.ndarray:
        if self.vocab is None:
            raise ValueError("Model must be fitted")
        cooc_matrix = self._build_cooccurrence_matrix(docs, self.vocab)
        X_hel = self._hellinger_transform(cooc_matrix)
        return self.pca.transform(X_hel)

    # embed doc
    def transform_doc(self, docs: List[Union[str, List[str]]]) -> np.ndarray:
        if self.vocab is None or self.embeddings is None:
            raise ValueError("Model must be fitted before calling transform_docs.")
        doc_embeddings = []
        dim = self.embeddings.shape[1]

        for doc in docs:
            if isinstance(doc, str):
                tokens = doc.split()
            elif isinstance(doc, list):
                tokens = doc
            else:
                raise ValueError("Each document must be a string or a list of tokens.")

            word_vectors = [
                self.embeddings[self.vocab[word]]
                for word in tokens
                if word in self.vocab
            ]
            if word_vectors:
                doc_embeddings.append(np.mean(word_vectors, axis=0))
            else:
                doc_embeddings.append(np.zeros(dim))

        return np.array(doc_embeddings)
```

Lớp `PCA`: Lớp thực hiện phân tích thành phần chính bằng SVD, được sử dụng nội bộ trong `HPEmbedder`.

- `fit(X)`: Chuẩn hóa dữ liệu và tính toán ma trận thành phần chính bằng SVD.
- `transform(X)`: Chiếu dữ liệu vào không gian thành phần chính.
- `fit_transform(X)`: Kết hợp `fit` và `transform`.
- `inverse_transform(X_transformed)`: Khôi phục xấp xỉ dữ liệu từ không gian PCA về không gian ban đầu.
- `choose_n_components(threshold)`: Chọn số thành phần chính sao cho tỷ lệ phương sai tích lũy lớn hơn ngưỡng `threshold`.
- `plot_cumulative_variance(threshold)`: Vẽ đồ thị tỷ lệ phương sai tích lũy theo

số chiều giữ lại.

```
class PCA:
    def __init__(self, n_components):
        self.n_components = n_components
        self.components = None
        self.mean = None
        self.explained_variance_ratio_ = None
        self.fitted = False

    def fit(self, X):
        self.fitted = True
        self.mean = np.mean(X, axis=0)
        X_centered = X - self.mean

        U, S, Vt = np.linalg.svd(X_centered, full_matrices=False)
        total_var = np.sum(S**2)
        explained_var = (S**2) / total_var

        self.explained_variance_ratio_ = explained_var[:self.n_components]

        self.components = Vt[:self.n_components]

    def transform(self, X):
        X_centered = X - self.mean
        return np.dot(X_centered, self.components.T)

    def fit_transform(self, X):
        self.fit(X)
        return self.transform(X)

    def inverse_transform(self, X_transformed):
        return np.dot(X_transformed, self.components) + self.mean

    def choose_n_components(self, threshold=0.95):
        if not self.fitted:
            raise ValueError("PCA must be fitted first!")
        cum_var_ratio = np.cumsum(self.explained_variance_ratio_)

        n_components = np.searchsorted(cum_var_ratio, threshold) + 1
        self.n_components = n_components
        self.components = self.components[:n_components]
        return n_components

    def plot_cumulative_variance(self, threshold = 0.95, n_component = None):
        if self.explained_variance_ratio_ is None:
            raise RuntimeError("Must fit first")

        cum_var = np.cumsum(self.explained_variance_ratio_)
        plt.figure(figsize=(20, 10))

        plt.plot(range(1, len(cum_var)+1), cum_var, linestyle='-')
        if threshold is not None:
            n_component = self.choose_n_components(threshold) if n_component is None
                else n_component
            plt.axvline(x=n_component, color='blue', linestyle='--', label=f'Selected
                Components = {n_component}')
            plt.axhline(y=threshold, color='red', linestyle='--', label=f'Remained
                Information = {threshold * 100}%')
        plt.xlabel("Number of Components")
        plt.ylabel("Cumulative Explained Variance Ratio")
        plt.title("Cumulative Explained Variance by PCA Components")
        plt.grid(True)
        plt.legend()
        plt.show()
```

Triển khai thuật toán:

- Tiền xử lý tập tài liệu

```
    # Intialize corpus
```

```
    corpus = [
        "A father and son bond over a stick up robbery.",
        "An old family foe threatens a young woman's Nashville dreams."
    ]
    # Initialize preprocessor
    from preprocessing.preprocessing import LSASVDPipeline
    preprocessor = WordEmbPipeline()
    preprocessed_corpus = [preprocessor.preprocess(text) for text in corpus]
    print(preprocessed_corpus)
    # Result
    #['father son bond stick robbery',
    #'old famili foe threatens young woman nashville dream']
```

- Khai báo, huấn luyện mô hình, và biến đổi.

```
    from embedding.HellingerPCA import HEmbedder
    embedder = HPEmbedder(max_features=4, n_components=2)
    embedder.fit(docs)

    # Hellinger matrix:
    #[[0.         0.5        0.5        0.        ]
    # [0.33333334 0.         0.33333334 0.33333334]
    # [0.33333334 0.33333334 0.         0.33333334]
    # [0.         0.5        0.5        0.        ]]

    print(embedder.transform_word(docs)) # word embedding
    #[[-5.04191618e-01 -2.12811137e-17]
    # [ 5.04191618e-01 -4.08248290e-01]
    # [ 5.04191618e-01  4.08248290e-01]
    # [-5.04191618e-01 -2.12811137e-17]]
```

5 Thực nghiệm

Nhằm đánh giá trực quan và so sánh hiệu quả giữa các phương pháp đã đề xuất trong thiết lập ở Mục 5.1, nhóm tác giả tiến hành thực nghiệm trên tập số liệu cụ thể (Mục 5.2), đồng thời xây dựng một trang web minh hoạ để người dùng có thể dễ dàng quan sát và tương tác với kết quả (Mục 5.3). Việc kết hợp giữa phân tích định lượng và trình bày trực quan giúp làm nổi bật ưu điểm của từng phương pháp, đồng thời hỗ trợ người đọc có cái nhìn toàn diện và trực quan hơn về hiệu quả biểu diễn cũng như khả năng phân biệt của các mô hình.

5.1 Thiết lập thực nghiệm

Dữ liệu. Bộ dữ liệu được thu thập từ IMDB thông qua API, với gần 10.000 mẫu dữ liệu. Tập dữ liệu này được sử dụng làm tập huấn luyện, trong khi tập kiểm tra được trích ngẫu nhiên từ tập huấn luyện với kích thước hơn 2000 dữ liệu. Tập kiểm tra sau đó được áp dụng kĩ thuật sinh văn bản [10] từ mô hình ngôn ngữ lớn Gemma3 - 27b để tạo ra các câu truy vấn mang phong cách tự nhiên, gần với cách diễn đạt của con người.

Chỉ số đánh giá. Để đánh giá hiệu quả của hệ thống, nhóm tác giả sử dụng các chỉ số phổ biến trong lĩnh vực truy hồi thông tin và ngôn ngữ tự nhiên.

- Precision@K (P@K), Recall@K (R@K) F1@K [11]: phản ánh mức độ cân bằng giữa dự đoán đúng và sai, rất hữu ích trong bài toán phân loại không cân bằng.

$$\text{Precision@K} = \frac{\text{Số lượng kết quả đúng trong top-K}}{K} \tag{43}$$

$$\text{Recall@K} = \frac{\text{Số lượng kết quả đúng trong top-K}}{\text{Tổng số kết quả đúng}} \tag{44}$$

$$\text{F1@K} = 2 \times \frac{\text{Precision@K} \times \text{Recall@K}}{\text{Precision@K} + \text{Recall@K}} \tag{45}$$

- MRR (Mean Reciprocal Rank) [12]: Trung bình nghịch đảo thứ hạng của câu trả lời đúng đầu tiên trong danh sách kết quả.

$$\text{MRR} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{\text{rank}_i} \tag{46}$$

- MAP (Mean Average Precision) [13]: Trung bình của Precision tại mỗi vị trí có một mục đúng, qua nhiều truy vấn. MAP đánh giá toàn bộ danh sách kết quả.

$$\text{MAP} = \frac{1}{|Q|} \sum_{q \in Q} AP(q) \tag{47}$$

$$AP(q) = \frac{1}{\text{số lượng kết quả đúng}} \sum_{k=1}^{n} \text{Precision@}k \cdot \text{rel}(k) \tag{48}$$

  Trong đó rel(k) là 1 nếu đúng ở vị trí k, hoặc là 0.

- nDCG@k (Normalized Discounted Cumulative Gain) [14]: Là chỉ số đánh giá có trọng số cho các mục đúng theo thứ tự sắp xếp, với các mục đúng ở vị trí đầu có giá trị cao hơn. Được chuẩn hóa để dễ so sánh giữa các truy vấn.

$$\text{nDCG@K} = \frac{\text{DCG@K}}{\text{IDCG@K}} \tag{49}$$

  DCG@K = $\sum_{i=1}^{K} \frac{2^{\text{rel}_i}-1}{\log_2(i+1)}$, IDCG@K là DCG với các kết quả sắp xếp theo đúng độ liên quan giảm dần.

- Thời gian truy vấn trung bình của query: Là chỉ số đánh giá UX (user experience)

Trong số các chỉ số trên, các giá trị như P@K, R@K, F1@K, MRR, MAP và nDCG@K càng cao càng thể hiện hiệu quả mô hình tốt hơn, trong khi thời gian truy vấn càng thấp càng tốt để đảm bảo trải nghiệm người dùng.

5.2 Kết quả

Sau khi thực nghiệm, ta có Bảng 3 thể hiện kết quả cuối cùng của từng phương pháp. Ba phương pháp cơ bản gồm BoW, TF-IDF, và PPMI được xem như các baseline để đối chiếu hiệu quả với các kỹ thuật nâng cao. Trong số này, BoW đạt kết quả tốt nhất với Precision@10 (P@10) là 0.081 và Recall@10 (R@10) lên đến 0.800 - cao hơn đáng kể so với TF-IDF và PPMI. Đồng thời, BoW cũng có MRR và MAP cao nhất trong nhóm baseline (0.699 và 0.698), cho thấy khả năng truy xuất tài liệu đúng thứ tự khá hiệu quả dù biểu diễn còn thô. TF-IDF - vốn phổ biến nhờ việc gán trọng số theo tầm quan trọng của từ - lại có độ chính xác thấp hơn (P@10 = 0.072), và toàn bộ chỉ số truy xuất đều kém hơn BoW, ngoại trừ thời gian xử lý có phần nhanh hơn. PPMI, dù có lợi thế về khả năng nắm bắt thông tin đồng xuất hiện, lại là phương pháp có hiệu suất thấp nhất trong

Method	P@10	R@10	F1@10	MRR	MAP	nDCG@10	Time
BoW	0.081	0.800	0.146	0.699	0.698	0.723	0.246
TF-IDF	0.072	0.714	0.130	0.562	0.561	0.598	0.236
PPMI	0.051	0.503	0.092	0.441	0.441	0.456	<u>0.212</u>
HellingerPCA	0.038	0.381	0.069	0.265	0.265	0.293	0.230
Word2Vec	0.048	0.482	0.09	0.446	0.446	0.456	0.246
GloVe	**0.107**	**0.816**	**0.196**	**0.784**	**0.783**	<u>0.734</u>	0.224
FastText	<u>0.102</u>	<u>0.796</u>	<u>0.166</u>	<u>0.754</u>	<u>0.753</u>	**0.764**	**0.143**

Bảng 3: Kết quả thực nghiệm. Kết quả tốt nhất được bôi đậm, kết quả tốt nhì được gạch chân.

nhóm, với F1@10 chỉ 0.092 và nDCG@10 thấp (0.456), cho thấy cách tiếp cận này khó cạnh tranh nếu không có bước giảm chiều hoặc lọc nhiễu đi kèm.

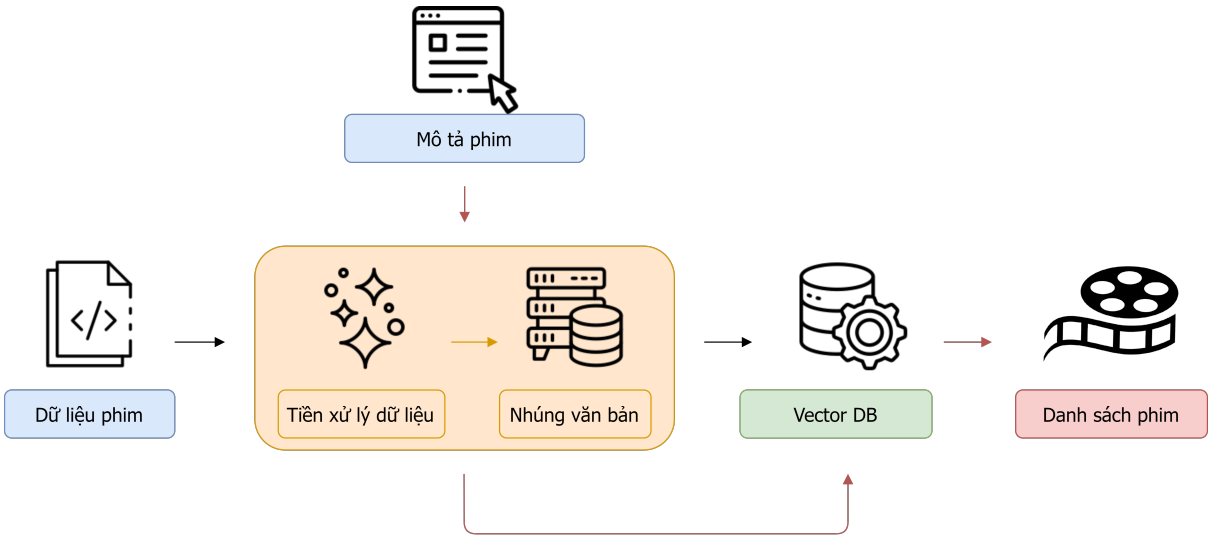
Ngược lại, các phương pháp thay thế như Word2Vec, GloVe, FastText, và biến thể HellingerPCA cho thấy hiệu quả vượt trội về mặt ngữ nghĩa. GloVe là phương pháp nổi bật nhất trong toàn bộ bảng, với P@10 đạt 0.107, F1@10 cao nhất là 0.196, và các chỉ số MRR, MAP đều trên 0.78 - chứng minh khả năng học tốt mối quan hệ ngữ nghĩa toàn cục từ văn bản. FastText tuy có Precision thấp hơn một chút (0.100), nhưng lại có thời gian xử lý nhanh nhất (0.143s), đồng thời vẫn duy trì hiệu suất tốt với nDCG@10 cao nhất (0.764), phù hợp với các hệ thống cần thời gian phản hồi nhanh. Word2Vec và HellingerPCA có kết quả thấp hơn, với HellingerPCA tỏ ra chưa hiệu quả khi chỉ đạt F1@10 là 0.069 và MRR là 0.265, cho thấy phương pháp giảm chiều bằng Hellinger + PCA có thể cần thêm tinh chỉnh để đạt được biểu diễn ngữ nghĩa ổn định. Nhìn chung, các mô hình embedding hiện đại đều vượt trội so với baseline truyền thống, không chỉ về độ chính xác mà còn ở khả năng sắp xếp tài liệu theo mức độ liên quan ngữ nghĩa trong bài toán truy xuất.

5.3 Xây dựng website ứng dụng LSA vào hệ thống truy xuất phim phim dựa trên mô tả

Nhóm tác giả đã phát triển một ứng dụng web với pipeline được mình họa ở Hình 7. Ứng dụng web này cho phép người dùng nhập một đoạn mô tả nội dung phim (bằng tiếng Anh) sau đó lựa chọn mô hình và nhận lại danh sách các phim có nội dung tương đồng nhất với mô tả đó. Mục tiêu chính là giúp người dùng tìm lại phim khi họ chỉ nhớ nội dung mà không nhớ tên phim, diễn viên hay năm phát hành thông qua các mô hình.

Chức năng chính của ứng dụng. Ứng dụng cho phép người dùng nhập mô tả nội dung hoặc thể loại phim mong muốn trực tiếp và chọn mô hình trên giao diện web. Sau khi tiếp nhận văn bản đầu vào, hệ thống sẽ xử lý mô tả này và ánh xạ nó thành vector nhúng thông qua các mô hình biểu diễn ngôn ngữ đã huấn luyện. Quá trình truy vấn được thực hiện trên cơ sở dữ liệu vector sử dụng công cụ tìm kiếm tương đồng, từ đó trả về danh sách các bộ phim có nội dung gần giống nhất. Kết quả hiển thị bao gồm tên phim, hình ảnh poster phim và các thông tin của phim trong cơ sở dữ liệu.

So sánh các thuật toán biểu diễn văn bản. Ứng dụng được thiết kế với khả năng tích hợp và so sánh nhiều kỹ thuật biểu diễn văn bản khác nhau, bao gồm TF-IDF,



Hình 7: Pipeline xử lý dữ liệu và truy vấn tìm kiếm phim

Word2Vec, FastText, PPMI, GloVe, Bag-of-Words (BoW) và HellingerPCA. Khi người dùng nhập mô tả phim,và chọn mô hình mong muốn hệ thống sẽ lần lượt ánh xạ mô tả này thành vector nhúng theo mô hình đã chọn, sau đó thực hiện truy vấn tìm kiếm trên cơ sở dữ liệu tương ứng để thu về danh sách các phim tương tự. Điều này cho phép hiển thị và so sánh trực quan kết quả giữa các mô hình khác nhau, hỗ trợ người dùng hoặc nhà nghiên cứu dễ dàng đánh giá hiệu quả, mức độ phù hợp và độ chính xác của từng phương pháp biểu diễn ngữ nghĩa.

Giao diện web đơn giản, dễ sử dụng. Giao diện của ứng dụng được xây dựng bằng Flask với thiết kế tối giản, trực quan, giúp người dùng dễ dàng nhập mô tả nội dung phim (hoặc dùng hệ thống nhận diện giọng nói tiếng Anh để nhập) và lựa chọn mô hình biểu diễn ngữ nghĩa để thực hiện tìm kiếm. Sau khi truy vấn được xử lý, hệ thống hiển thị kết quả một cách rõ ràng và trực quan, bao gồm tên phim, thông tin phim và hình ảnh poster phim đi kèm. Giao diện được tối ưu nhằm phục vụ mục đích thử nghiệm, so sánh mô hình cũng như hỗ trợ nghiên cứu trong lĩnh vực truy xuất thông tin.

Pipeline xử lý dữ liệu rõ ràng, có thể mở rộng. Pipeline của hệ thống được thiết kế mạch lạc và dễ dàng mở rộng. Đầu tiên, mô tả phim được tiền xử lý bao gồm các bước làm sạch dữ liệu, tách từ và chuẩn hoá văn bản. Sau đó, các mô hình biểu diễn ngữ nghĩa như TF-IDF, Word2Vec, PPMI hay GloVe sẽ được sử dụng để chuyển văn bản thành các vector đặc trưng. Những vector này sẽ được lưu trữ vào cơ sở dữ liệu vector Qdrant nhằm phục vụ việc truy vấn nhanh chóng. Khi người dùng nhập một mô tả mới, hệ thống sẽ chuyển đổi mô tả này thành vector tương ứng, truy vấn lên Qdrant để tìm ra top-k phim có vector gần nhất về mặt ngữ nghĩa sử dụng độ tương đồng Cosine. Nhờ thiết kế linh hoạt, pipeline này cho phép tích hợp thêm các mô hình mới hoặc mở rộng sang các tập dữ liệu lớn hơn mà không làm ảnh hưởng đến hiệu năng tổng thể. ư

6 Kết luận

LSA là phương pháp cổ điển nhưng là nền tảng vững chắc trong lĩnh vực xử lý ngôn ngữ tự nhiên, đặc biệt là trong các hệ thống truy hồi thông tin. Một trong những ưu điểm lớn của LSA là khả năng phát hiện mối quan hệ ngữ nghĩa giữa các từ và tài liệu, nhờ đó xử lý tốt hiện tượng đồng nghĩa. Việc giảm chiều dữ liệu thông qua SVD cũng giúp loại bỏ nhiễu, tiết kiệm chi phí tính toán và lưu trữ mà vẫn giữ lại được những yếu tố cốt lõi về ngữ nghĩa. Ngoài ra, LSA không yêu cầu lượng dữ liệu lớn, điều này đặc biệt hữu ích trong các hệ thống IR ở những lĩnh vực có tài nguyên hạn chế như y tế, pháp lý hay thư viện số. Việc triển khai LSA cũng khá đơn giản nhờ nền tảng toán học rõ ràng, dễ kiểm soát và dễ diễn giải.

Tuy nhiên, LSA cũng mang nhiều hạn chế. Mô hình không giải quyết được vấn đề đa nghĩa, ví dụ, từ *iron* trong *iron man* và *iron the clothe* sẽ có cùng biểu diễn vector, dẫn đến mất thông tin ngữ cảnh quan trọng trong IR. Bên cạnh đó, LSA rất phụ thuộc vào bước tiền xử lý, đặc biệt với tiếng Việt hoặc tiếng Đức. Về mặt tính toán, việc phân rã SVD trên các ma trận lớn và thưa cũng đòi hỏi tài nguyên cao. Cuối cùng, việc lựa chọn số chiều giữ lại sau SVD là một bài toán đánh đổi: nếu giữ quá ít sẽ làm mất thông tin quan trọng, còn nếu giữ quá nhiều sẽ dẫn đến lời nguyền chiều cao, làm giảm độ chính xác của các phép đo tương đồng hoặc khoảng cách - vốn là cốt lõi trong IR.

Những khuyết điểm của LSA có thể được cải thiện bằng các phương pháp hiện đại hơn là Word2Vec, GloVe, FastText, hoặc HellingerPCA. Word2Vec có khả năng học được các mối quan hệ ngữ nghĩa và cú pháp giữa các từ thông qua ngữ cảnh. GloVe tận dụng thống kê toàn cục của văn bản, giúp mô hình học được các đặc trưng ngữ nghĩa ổn định ngay cả khi không có ngữ cảnh gần. Trong khi đó, FastText cải tiến hơn bằng cách biểu diễn từ dưới dạng các n-gram ký tự, cho phép mô hình học được cấu trúc hình thái và tạo vector cho cả những từ chưa từng xuất hiện trong tập huấn luyện (OOV). HellingerPCA mặc dù không đạt được kết quả cao trong bài toán truy xuất phim, nhưng vẫn còn có tiềm năng lớn trong các bài toán NLP nhờ kiến trúc đơn giản.

Tóm lại, LSA là một phương pháp hiệu quả, dễ triển khai và vẫn rất phù hợp cho các hệ thống truy hồi thông tin yêu cầu xử lý ngữ nghĩa nhưng không có điều kiện triển khai các mô hình học sâu hiện đại. Từ Bảng 3 cho thấy, với ứng dụng tìm kiếm phim dựa trên mô tả có lượng dữ liệu không quá lớn (gần 10.000 mẫu dữ liệu), LSA vẫn hoạt động vô cùng tốt và vẫn có thể cạnh tranh với các phương pháp hiện đại hơn. Với sự cân bằng giữa tính ngữ nghĩa, chi phí và độ diễn giải, LSA vẫn là một công cụ đáng tin cậy cho nhiều ứng dụng IR hiện nay.

Thông qua quá trình thực hiện, nhóm tác giả đã tự mình cài đặt các thuật toán nhúng văn bản từ "cổ đại" đến hiện đại, cùng với đó xây dựng website tìm kiếm phim. Nhờ vậy, nhóm đã nắm rõ được cách hoạt động của từng thuật toán, cùng với đó là cách xây dựng và triển khai sản phẩm trong thực tế. Bên cạnh kiến thức về mô hình nhúng như Bag-of-Words, HellingerPCA, hay Word2Vec, nhóm còn trau dồi kỹ năng lập trình hướng đối tượng, tối ưu hiệu suất xử lý, và quản lý mã nguồn hiệu quả. Việc kết hợp các mô hình nhúng với hệ thống truy vấn thực tế cũng giúp nhóm hiểu rõ hơn về pipeline xử lý dữ liệu, từ tiền xử lý, biểu diễn văn bản đến đánh giá độ tương đồng và gợi ý kết quả phù hợp với nhu cầu người dùng.

Đặc biệt, việc xây dựng giao diện và triển khai website cho phép nhóm tiếp cận với các công cụ phát triển phần mềm thực tế và các framework front-end. Qua đó, nhóm nhận

thức được tầm quan trọng của việc kết hợp chặt chẽ giữa thuật toán và trải nghiệm người dùng trong các hệ thống AI ứng dụng. Cuối cùng, dự án không chỉ giúp nhóm củng cố lý thuyết và kỹ thuật lập trình, mà còn rèn luyện tinh thần làm việc nhóm, khả năng giải quyết vấn đề và thích nghi với những thách thức trong quá trình triển khai sản phẩm thực tế.

References

[1] C. Eckart and G. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, no. 3, pp. 211–218, 1936. DOI: 10.1007/BF02288367.

[2] T. K. Landauer and S. T. Dumais, "A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge.," *Psychological Review*, vol. 104, pp. 211–240, 1997. [Online]. Available: https://api.semanticscholar.org/CorpusID:1144461.

[3] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Commun. ACM*, vol. 18, no. 11, pp. 613–620, Nov. 1975, ISSN: 0001-0782. DOI: 10.1145/361219.361220. [Online]. Available: https://doi.org/10.1145/361219.361220.

[4] K. W. Church and P. Hanks, "Word association norms, mutual information, and lexicography," *Computational Linguistics*, vol. 16, no. 1, pp. 22–29, 1990. [Online]. Available: https://aclanthology.org/J90-1003/.

[5] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems*, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, Eds., vol. 26, Curran Associates, Inc., 2013, pp. 1–8. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

[6] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, A. Moschitti, B. Pang, and W. Daelemans, Eds., Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. [Online]. Available: https://aclanthology.org/D14-1162/.

[7] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, no. null, pp. 2121–2159, Jul. 2011, ISSN: 1532-4435.

[8] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the association for computational linguistics*, vol. 5, pp. 135–146, 2017.

[9] R. Lebret and R. Collobert, "Word embeddings through hellinger PCA," in *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, S. Wintner, S. Goldwater, and S. Riezler, Eds., Gothenburg, Sweden: Association for Computational Linguistics, Apr. 2014, pp. 482–490. DOI: 10.3115/v1/E14-1051. [Online]. Available: https://aclanthology.org/E14-1051/.

[10] H. A. Rahmani, N. Craswell, E. Yilmaz, B. Mitra, and D. Campos, "Synthetic test collections for retrieval evaluation," in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '24, Washington DC, USA: Association for Computing Machinery, 2024, pp. 2647–2651, ISBN: 9798400704314. DOI: 10.1145/3626772.3657942. [Online]. Available: https://doi.org/10.1145/3626772.3657942.

[11] C. J. V. Rijsbergen, *Information Retrieval*, 2nd. USA: Butterworth-Heinemann, 1979, pp. 114–119, ISBN: 0408709294.

[12] K. Sparck Jones, "A statistical interpretation of term specificity and its application in retrieval," en, *Journal of documentation*, vol. 28, no. 1, pp. 11–21, 1972.

[13] C. Buckley and E. M. Voorhees, "Evaluating evaluation measure stability," in *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '00, Athens, Greece: Association for Computing Machinery, 2000, pp. 33–40, ISBN: 1581132263. DOI: 10.1145/345508.345543. [Online]. Available: https://doi.org/10.1145/345508.345543.

[14] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of ir techniques," *ACM Trans. Inf. Syst.*, vol. 20, no. 4, pp. 422–446, Oct. 2002, ISSN: 1046-8188. DOI: 10.1145/582415.582418. [Online]. Available: https://doi.org/10.1145/582415.582418.