

ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG



SOICT

Báo Cáo Bài Tập Lớn

Đề tài: [Đề xuất game cho người dùng trên nền tảng Steam](#)

Học phần: [Học máy và Khai phá dữ liệu](#)

Mã học phần: IT3190 - Mã lớp: 157001

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

Nhóm sinh viên thực hiện:	Đặng Tiên Cường	20220020
	Bùi Mạnh Hưng	20224860
	Hoàng Quốc Hùng	20224857
	Trịnh Tuấn Thành	20225532
	Phạm Gia Hưng	20230036

Hà Nội, Ngày 6 tháng 9 năm 2025

Mục lục

1 Giới thiệu đề tài	4
1.1 Đặt vấn đề	4
1.2 Tổng quan về bài toán hệ gợi ý	5
1.3 Cách tiếp cận của nhóm chúng tôi	8
2 Tiền xử lý dữ liệu	9
2.1 Đường ống xử lý dữ liệu	9
2.1.1 Thu thập dữ liệu	9
2.1.2 Mô tả tổng quan dữ liệu	9
2.1.3 Phân tích khám phá dữ liệu (EDA)	11
2.1.4 Làm sạch dữ liệu (Data Cleaning)	17
2.1.5 Chuẩn bị dữ liệu (Data Preparation)	18
2.1.6 Kết luận	21
3 Cơ sở lý thuyết	22
3.1 Hướng tiếp cận Content-based Filtering	22
3.1.1 Xây dựng đặc trưng	22
3.1.2 Ưu điểm và Hạn chế	23
3.2 Hướng tiếp cận Collaborative Filtering	23
3.2.1 Neighbourhood Models	24
3.2.2 Matrix Factorization Models	25
3.2.3 Factorization Machine Models	35
3.2.4 Graph Neural Network Models	37
4 Các chiến lược đánh giá và độ đo áp dụng	47
4.1 Các độ đo đánh giá hệ gợi ý	47
4.1.1 Precision at k	47
4.1.2 Recall at k	47
4.1.3 F ₁ at k	47
4.1.4 NDCG at k	48
4.1.5 Hit Rate at k	48
4.2 Các chiến lược đánh giá	48
4.2.1 Full-corpus	48
4.2.2 Leave-one-last	49
5 Huấn luyện và đánh giá	51
5.1 Thuật toán huấn luyện Adam	51
5.2 Kết quả huấn luyện và đánh giá	52

Lời nói đầu

Trong kỷ nguyên bùng nổ thông tin, chúng ta đang phải đối mặt với một “biển” thông tin về sản phẩm và dịch vụ xuất hiện trên các nền tảng trực tuyến. Việc tìm kiếm nội dung phù hợp với nhu cầu cá nhân trở thành một thách thức không nhỏ. Trước thực trạng đó, **Hệ thống gợi ý (Recommendation System)** đã ra đời như một giải pháp công nghệ mang tính cá nhân hóa cao, giúp người dùng dễ dàng tiếp cận các lựa chọn phù hợp hơn với sở thích và nhu cầu của mình.

Chắc hẳn, chúng ta đã từng một lần bước vào một cửa hàng, mà “mặt tiền” trưng bày nhiều mặt hàng “bán chạy”, các mặt hàng này có thể phù hợp nhu cầu của đa số khách hàng, nhưng không phù hợp với nhu cầu của ta. Điều mà chúng ta thực sự muốn bây giờ là ngay lập tức thấy được những mặt hàng phù hợp, nhưng điều này trong thực tế là khó, chúng ta chỉ còn cách hỏi nhân viên cửa hàng hoặc “tự mò mẫm” trong một “mớ hỗn độn” các mặt hàng. Tuy nhiên, việc gợi ý mặt hàng phù hợp trên nền tảng trực tuyến là dễ dàng do sự phát triển mạnh mẽ của **Trí tuệ nhân tạo (AI)**, đặc biệt là **Học máy (Machine Learning)**, **Học sâu (Deep Learning)** trong những năm gần đây đã góp phần quan trọng trong việc nâng cao hiệu quả và độ chính xác của các hệ thống gợi ý, đặc biệt là trong môi trường trực tuyến – nơi dữ liệu người dùng và hành vi của họ được thu thập và xử lý liên tục. Nhờ đó, việc đề xuất sản phẩm phù hợp cho từng cá nhân trở nên khả thi và hiệu quả hơn bao giờ hết.

Nhiều tập đoàn công nghệ lớn trên thế giới như **Netflix**, **Amazon**, **Steam**,... đã ứng dụng mạnh mẽ hệ thống gợi ý vào hoạt động kinh doanh của mình. Những hệ thống này không chỉ nâng cao trải nghiệm người dùng mà còn góp phần cải thiện tỷ lệ chuyển đổi, tăng mức độ hài lòng của khách hàng và tối ưu hóa doanh thu một cách đáng kể. Thực ra, việc gợi ý sản phẩm cho người dùng cũng rất quen thuộc trong bối cảnh các quy trình nghiệp vụ ngày càng được số hóa như ngày nay, ví dụ như khi ta mua sách online trên ứng dụng Tiki chẳng hạn, ta thực hiện tìm kiếm tên quyển sách, sau đó cho quyển sách vào giỏ hàng thì ngay lập tức ở đâu đó trên màn hình ứng dụng, ta sẽ được Tiki gợi ý những quyển sách liên quan với quyển sách ta định mua, có thể là cùng một tác giả hoặc cùng một chủ đề. Ngoài ra, việc Facebook gợi ý kết bạn khi chúng ta lướt News Feed cũng là một tình huống dễ bắt gặp nhất.

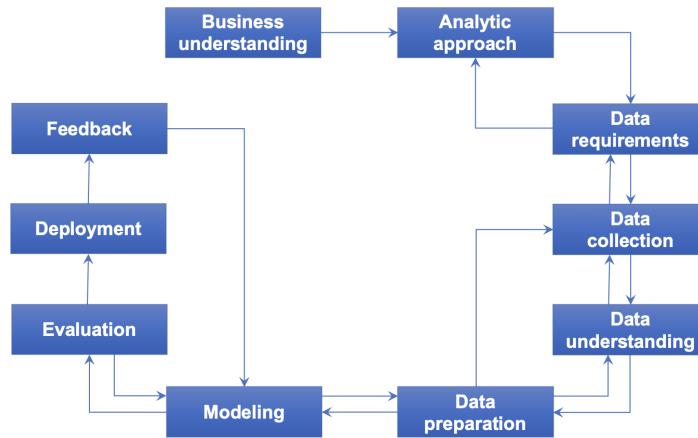
Vậy câu hỏi đặt ra là: Làm cách nào để một Hệ thống gợi ý có thể đưa ra những sản phẩm phù hợp cho các người dùng khác nhau? Đây cũng là nguyên nhân chính mà nhóm chúng tôi lựa chọn đề tài về **Hệ thống gợi ý**, tất cả thành viên đều đồng tình rằng nếu không có cơ hội tìm hiểu về chủ đề này thì việc gợi ý sản phẩm, gợi ý kết bạn trên các nền tảng trực tuyến là trong suốt, không có quá nhiều sự nhận thức rõ ràng. Do đó, việc tìm hiểu về chủ đề này giúp cả nhóm khám phá cơ chế ẩn sâu bên trong các **Hệ thống gợi ý** đã và đang thực hiện ngày nay, nhóm chúng tôi không còn thấy việc gợi ý sản phẩm như là một “**Black Box**” nữa.

Tất nhiên, đề tài về Hệ thống gợi ý là khá chung chung, do đó nhóm chúng tôi quyết định nghiên cứu một Case Study cụ thể hơn, chính là tiêu đề của báo cáo: **Đề xuất game cho người dùng trên nền tảng Steam**. Lý do sâu sa là vì một vài thành viên trong nhóm đã từng có rất nhiều kinh nghiệm chơi game trên nền tảng Steam và có thành tích chơi game đáng nể. Khi nhóm đã có kinh nghiệm thì việc hiểu bài toán **Business Understanding¹** cần giải quyết là vô cùng dễ dàng.

Trong toàn bộ quy trình làm bài tập lớn của học phần này, nhóm chúng tôi đã bám sát các bước xây dựng một hệ thống dựa trên học máy (Hình 1). Điều này thể hiện rõ ràng một cách tuần tự ở cấu trúc báo cáo. Bố cục của báo cáo bao gồm 04 chương:

- **Chương 1 - Giới thiệu đề tài.** Chương này trình bày cái nhìn tổng quan nhất về nghiệp vụ

¹Bước đầu tiên trong quy trình xây dựng hệ thống học máy



Hình 1: Quy trình xây dựng một hệ thống dựa trên học máy

(business), miền lĩnh vực mà nhóm cần giải quyết. Những thách thức tiêu biểu khi xây dựng hệ thống gợi ý Game gấp phải, cần phải giải quyết để đạt hệ thống được hiệu suất tốt nhất có thể. Sau khi đã hiểu nghiệp vụ (Business Understanding), nhóm chuyển nghiệp vụ cần giải quyết về bài toán trong miền lĩnh vực trong học máy đồng thời đặt ra **đòi hỏi về dữ liệu (Data Requirement)** và **nhu cầu cần thu thập đúng dữ liệu (Data Collection)**.

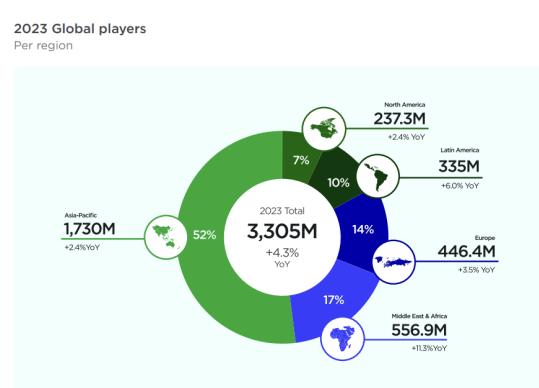
- **Chương 2 - Tiền xử lý dữ liệu.** Chương này trình bày về cách chúng tôi xử lý dữ liệu trước khi đưa vào mô hình học máy. Chúng tôi sẽ trình bày quy trình Tiền xử lý dữ liệu dưới dạng hai bước con:
 - **Bước 1 - Hiểu dữ liệu (Data Understanding):** Nhóm áp dụng các kỹ thuật trong Phân tích khám phá dữ liệu (EDA - Exploratory Data Analysis) như **Trực quan hóa dữ liệu (Data Visualization)**, **Phân tích Ma trận tương quan (Matrix Correlation)**,...
 - **Bước 2 - Chuẩn bị dữ liệu (Data Preparation):** Nhóm trình bày các chiến lược chia tập dữ liệu ban đầu thành các tập dữ liệu train/validate/test để phục vụ cho việc đánh giá hệ thống gợi ý một cách hiệu quả. Các chiến lược này được nhóm dày công tìm hiểu, đúc rút qua nhiều vòng lặp **Chuẩn bị dữ liệu (Data Preparation)**, **Mô hình hóa (Modeling)**, **Đánh giá mô hình (Evaluation)**.
- **Chương 3 - Cơ sở lý thuyết:** Chương này sẽ trình bày chi tiết về cơ sở lý thuyết về hai cách tiếp cận mà nhóm chúng tôi sẽ áp dụng sau một **giai đoạn phân tích (Analytic approach)**
 - **Ở hướng tiếp cận Content-based Filtering (Lọc theo nội dung):** Nhóm chúng tôi dựa trên metadata (tag) và sử dụng độ tương đồng Cosine để gợi ý cho người dùng
 - **Ở hướng tiếp cận Collaborative - Filtering (Lọc cộng tác):** Nhóm chúng tôi tiến hành tiếp cận bài toán một cách tổng quát đi từ các **mô hình cơ bản**, **truyền thống** trong lĩnh vực hệ gợi ý như **ItemKNN**, **MF** đến những **mô hình tiên tiến**, nổi bật ra đời ở những năm gần đây như **các mô hình dựa trên Mạng nơ-ron đồ thị (GNN-based Models)** như: **NGCF**, **LightGCN**.
- **Chương 4 - Chiến lược đánh giá và độ đo áp dụng:** Nhóm chúng tôi áp dụng hai chiến lược đánh giá phổ biến trong hệ thống gợi ý bao gồm 02 chiến lược Full-Corpus và Leave-One-Last-Item, hai chiến lược này quyết định cách chúng tôi chia tập dữ liệu để thực hiện huấn luyện và đánh giá mô hình.
- **Chương 5 - Huấn luyện và đánh giá mô hình:** Nhóm chúng tôi đề cập đến thuật toán huấn luyện mô hình và đánh giá mô hình với tính chính xác tham số tốt nhất.

Chương 1

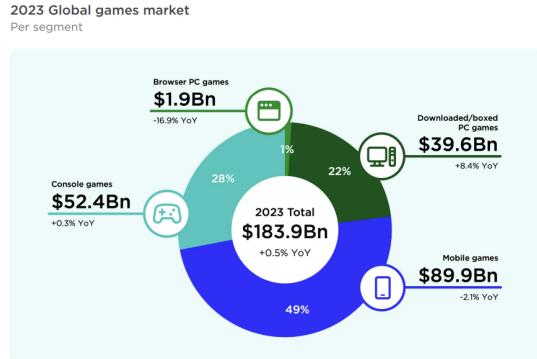
Giới thiệu đề tài

1.1 Đặt vấn đề

Trong những năm gần đây, thị trường trò chơi toàn cầu đã chứng kiến sự tăng trưởng mạnh mẽ cả về số lượng người chơi và doanh thu. Theo *Báo cáo Thị trường Trò chơi Toàn cầu 2023* của Newzoo, số lượng người chơi trên toàn thế giới đã đạt **3,31 tỷ người** vào năm 2023, và dự kiến sẽ tiếp tục tăng trưởng và đạt **3,68 tỷ người** vào năm 2026. Về doanh thu, thị trường trò chơi toàn cầu năm 2023 tạo ra **183,9 tỷ USD**, và dự kiến sẽ đạt **207,0 tỷ USD** vào năm 2026. Phân khúc PC, một trong những động lực tăng trưởng chính, đạt doanh thu **41,5 tỷ USD** trong năm 2023, nhờ sự đóng góp của các tựa game bom tấn như Hogwarts Legacy và Diablo IV. Trong bối cảnh này, Steam – nền tảng phân phối kỹ thuật số hàng đầu cho trò chơi trên PC – đã khẳng định vị thế của mình với **21% thị phần** trò chơi PC toàn cầu vào năm 2020, đạt doanh thu **33,9 tỷ USD** theo Newzoo.



Hình 1.1: Số lượng người chơi toàn cầu



Hình 1.2: Doanh thu của thị trường game toàn cầu

Steam, được phát triển bởi Valve Corporation từ năm 2003, không chỉ là một cửa hàng trực tuyến mà còn là một hệ sinh thái toàn diện cho người chơi và nhà phát triển. Với hơn **50.000 tựa game** và **132 triệu người dùng hoạt động hàng tháng** (tính đến năm 2023), Steam cung cấp một thư viện trò chơi phong phú, đã dạng thể loại từ các tựa game AAA đến các tựa game indie, hỗ trợ đa nền tảng (Windows, macOS, Linux). Tuy nhiên, sự đa dạng khổng lồ của thư viện game và số lượng người dùng lớn đã đặt ra nhiều thách thức trong việc cá nhân hóa trải nghiệm người dùng. Theo số liệu từ Steam vào năm 2014, **khoảng 37% trò chơi được mua nhưng chưa từng được chơi**, cho thấy người dùng gặp khó khăn trong việc tìm kiếm các tựa game phù hợp với sở thích của mình. Ngoài ra, một lỗi thuật toán vào năm 2018 đã khiến lưu lượng truy cập tập trung vào các tựa game phổ biến, làm giảm cơ hội hiển thị của các trò chơi nhỏ hơn, gây bất lợi cho các nhà phát triển độc lập và làm giảm sự đa dạng trong gợi ý.

Những vấn đề này đã tạo ra nhu cầu cấp thiết cho việc xây dựng một **Hệ thống gợi ý trò chơi** hiệu quả trên Steam, nhằm cung cấp các đề xuất cá nhân hóa, tăng cường trải nghiệm người dùng và thúc đẩy doanh thu. Tuy nhiên, việc phát triển **Hệ thống gợi ý** trên nền tảng Steam đối mặt với nhiều thách

thức, vấn đề tiêu biểu như sau:

- **Vấn đề 1 - Khởi đầu lạnh (Cold-Start):** Khi một nhà phát triển phát hành một **trò chơi mới (Item Cold-Start)** trên Steam thì chúng thường không đủ dữ liệu về người chơi (playtime, đánh giá), hay trường hợp người dùng mới (User Cold-Start) đăng ký tài khoản Steam, hệ thống không có đủ lịch sử để hiểu sở thích của họ. Mô hình dựa trên CF thường kém hiệu quả khi không thể tính toán được độ tương đồng giữa user-user và game-game.
- **Vấn đề 2 - Hiện tượng đuôi dài (Long-Tail Phenomenon):** Steam có hơn 50.000 tựa game nhưng khoảng 20% các Game phổ biến sẽ chiếm phần lớn tương tác (recommendation, số lượt chơi) hoặc số giờ chơi, 80% còn lại (các game ít phổ biến) thì nhận được một số ít tương tác cũng như số giờ chơi, tạo nên hiện tượng đuôi dài (long-tail), điều này dẫn đến ma trận tương tác user-item rất thưa, điều này gây khó khăn cho các mô hình học máy dựa trên CF như kNN, MF, FM,..., có thể tìm pattern chung giữa user-item.
- **Vấn đề 3 - Dữ liệu nhiễu (Noise Data):** Trên Steam, những tương tác “rác” từ tài khoản ít hoạt động (1-5 review) hay bot shilling rating không chỉ làm loãng ma trận user-item mà còn dẫn đến đề xuất sai lệch, khi mô hình học “thích” những pattern giả tạo thay vì sở thích thật. Bên cạnh đó, các item phụ như DLC hay bundle, vốn không phản ánh trải nghiệm chơi cốt lõi, cũng cần được tách biệt để tránh làm nhiễu tín hiệu chính. Việc thiếu cơ chế phát hiện và loại bỏ hiệu quả các nguồn noise này sẽ làm giảm đáng kể độ tin cậy của hệ gợi ý.

Chúng tôi sẽ trình bày chi tiết các cách giải quyết các thách thức trên ở các chương, phần sau của báo cáo. Một cách ngắn gọn:

- **Về vấn đề 1:**
 - **Với User Cold-Start,** đây là vấn đề thách thức lớn nên chúng tôi chỉ khắc phục được một phần vấn đề bằng cách gợi ý item phổ biến nhất cho User đó.
 - **Với Item Cold-Start,** nhóm chúng tôi sử dụng hướng tiếp cận **Content-based Filtering**, sử dụng cơ chế để gắn tag cho mỗi Item đó. Cụ thể chi tiết về hướng giải quyết sẽ được trình bày ở chương 3, phần 3.1.
- **Với vấn đề 2, vấn đề 3:** Nhóm chúng tôi thực hiện khắc phục vấn đề bằng cách lọc dữ liệu ban đầu để loại bỏ nhiễu, chúng tôi chỉ xét những user tích cực và các game chính, cụ thể chi tiết bước tiền xử lý dữ liệu này sẽ trình bày ở chương 2.

Nhóm chúng tôi đã thực hiện một quy trình phân tích có hệ thống, bắt đầu từ việc **hiểu rõ vấn đề nghiệp vụ (Business Understanding)**, sau đó **hiểu tổng quan về bài toán hệ gợi ý**, tiếp theo sau đó là **xác định và đặt ra nhu cầu về thu thập dữ liệu (Data Requirement & Data Collection)**. Phần tiếp theo sẽ trình bày tổng quan về bài toán hệ gợi ý.

1.2 Tổng quan về bài toán hệ gợi ý

Bản chất của bài toán hệ gợi ý là xếp hạng, trong đó mô hình sẽ tìm cách sắp xếp và đưa ra một nhóm nhỏ các sản phẩm phù hợp nhất trên tập hợp gồm rất nhiều các sản phẩm để gợi ý cho người dùng, dựa trên sở thích cá nhân của mỗi người và các ràng buộc liên quan.

Cụ thể, giả sử ta có một hàm đánh giá mức độ phù hợp giữa người dùng và sản phẩm:

$$f : \mathcal{U} \times \mathcal{I} \rightarrow \mathbb{R}$$

trong đó \mathcal{U} là tập người dùng, \mathcal{I} là tập sản phẩm. Mục tiêu của bài toán là học ra một quan hệ thứ tự toàn phần (hoặc xấp xỉ toàn phần) được cá nhân hóa cho mỗi người dùng $u \in \mathcal{U}$:

$$>_u \subseteq \mathcal{I} \times \mathcal{I}$$

sao cho:

$$i >_u j \iff f(u, i) > f(u, j)$$

Một quan hệ $>_u$ được gọi là **thứ tự toàn phần (total order)** trên tập \mathcal{I} nếu với mọi $i, j, k \in \mathcal{I}$, quan hệ này thỏa mãn các điều kiện sau:

- **Tính phản xạ (Irreflexivity):**

$$i \in \mathcal{I} \text{ sao cho } i >_u i$$

- **Tính bất đối xứng (Antisymmetry):**

$$i >_u j \Rightarrow \neg(j >_u i)$$

- **Tính bắc cầu (Transitivity):**

$$i >_u j \wedge j >_u k \Rightarrow i >_u k$$

- **Tính khả so sánh toàn phần (Totality):**

$$\forall i, j \in \mathcal{I}, i \neq j \Rightarrow ((i >_u j) \vee (j >_u i))$$

Từ quan hệ thứ tự cá nhân học được, hệ gợi ý sẽ sắp xếp các sản phẩm và đề xuất cho người dùng dựa trên thứ tự này. Hàm đánh giá f có thể được giả định là cố định từ trước, hoặc tùy vào mô hình mà có thể được học cùng quan hệ thứ tự $>_u$.

Để học được một quan hệ thứ tự toàn phần cho hệ gợi ý, có hai hướng tiếp cận chính: ***learn to predict*** và ***learn to rank***.

(1) **Learn to predict (pointwise approach):** Hướng tiếp cận này xem bài toán là dự đoán *điểm số* phản ánh mức độ người dùng ưa thích một sản phẩm, thông qua một hàm $f(u, i)$. Từ các điểm số này, mô hình có thể xếp hạng các sản phẩm theo thứ tự giảm dần của $f(u, i)$. Tùy vào dạng dữ liệu thu thập được mà bài toán có thể là *regression* hoặc *classification*. Việc huấn luyện thường được thực hiện thông qua tối ưu hóa một hàm mất mát dạng:

$$\min_{\theta} \sum_{(u, i, y_{ui}) \in \mathcal{D}} \mathcal{L}_{\text{pred}}(f_{\theta}(u, i), y_{ui}) + \lambda \mathcal{R}(\theta)$$

(2) **Learn to rank (pairwise hoặc listwise approach):** Hướng tiếp cận này không quan tâm đến giá trị tuyệt đối của $f(u, i)$, mà tập trung vào việc học trực tiếp quan hệ thứ tự giữa các sản phẩm, ví dụ như học để đảm bảo $f(u, i) > f(u, j)$ nếu i được ưu tiên hơn j . Nếu thu thập được dữ liệu có thứ tự toàn phần giữa các sản phẩm, mô hình sẽ tập trung tối ưu hóa thứ tự này cho cá nhân mỗi người dùng - hướng tiếp cận *listwise*. Còn nếu dữ liệu có được chỉ là tập thứ tự một phần (hữu hạn các cặp so sánh được), mô hình tìm cách tối ưu hóa khả năng mở rộng thứ tự này thành thứ tự toàn phần - hướng tiếp cận *pairwise*. Quá trình này thường được mô hình hóa dưới dạng bài toán tối ưu sau:

$$\min_{\theta} \sum_{(u, i, j) \in \mathcal{D}} \mathcal{L}_{\text{rank}}(f_{\theta}(u, i) - f_{\theta}(u, j)) + \lambda \mathcal{R}(\theta)$$

trong đó mỗi bộ ba (u, i, j) biểu thị rằng người dùng u thích sản phẩm i hơn j . Đây có thể xem là một dạng *self-supervised learning*, khi mô hình tự sinh ra nhãn huấn luyện từ dữ liệu tương tác chưa gán nhãn.

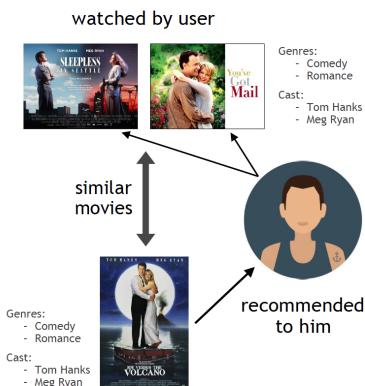
Dữ liệu đầu vào của dataset bao gồm 3 loại chính:

- **Thông tin của người dùng (user):** độ tuổi, giới tính, khu vực sinh sống, công việc,...
- **Thông tin của sản phẩm (item):** tiêu đề, thể loại, mô tả nội dung, hình ảnh minh họa,...
- **Lịch sử tương tác giữa user và item,** thường có dạng danh sách các bộ ba (user, item, interaction). Lịch sử này sẽ được chuyển về dạng ma trận tương tác $\mathbf{R} \in \mathbb{R}^{M \times N}$, trong đó mỗi cột ứng với lịch sử của mỗi user còn mỗi hàng ứng với các item.

Dựa vào loại dữ liệu được sử dụng để huấn luyện, các mô hình gợi ý có thể được chia thành ba nhóm chính:

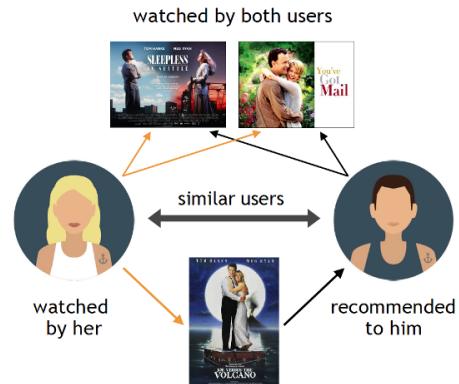
- **Content-based Filtering (Lọc theo nội dung):** Phương pháp này sử dụng thông tin nội dung của từng sản phẩm (item) để xây dựng đặc trưng riêng biệt cho từng item. Dựa vào hồ sơ sở thích cá nhân của mỗi người dùng (được hình thành từ lịch sử tương tác trước đó), hệ thống sẽ đề xuất các item có đặc điểm tương đồng với hồ sơ này. Ví dụ: nếu một người dùng thường xuyên xem các bộ phim thuộc thể loại hình sự, hệ thống có thể gợi ý phim *Người phán xử* vì nó chia sẻ đặc trưng thể loại *hình sự* với các phim mà người dùng đã xem.

Content-based Filtering



Hình 1.3: Content-based Filtering

Collaborative Filtering



Hình 1.4: Collaborative Filtering

- **Ưu điểm:** Khả năng cá nhân hóa cao, độc lập với các người dùng khác. Đặc biệt phòng tránh được tình huống *cold-start item*, khi có sản phẩm mới mà chưa có ai tương tác.
- **Nhược điểm:** Do chỉ dựa vào thông tin của từng người dùng nên không tận dụng được các xu hướng phổ biến trong toàn bộ hệ thống. Ngoài ra, với các nội dung phức tạp như hình ảnh hoặc âm thanh, việc xử lý và trích xuất đặc trưng có thể gặp nhiều khó khăn.
- **Collaborative Filtering (Lọc cộng tác):** Phương pháp này chủ yếu dựa vào lịch sử tương tác giữa người dùng và sản phẩm để đưa ra gợi ý, mà không cần quan tâm đến nội dung cụ thể của item.
 - **Ưu điểm:** Có khả năng khai thác các xu hướng chung trong cộng đồng người dùng. Đồng thời, không phụ thuộc vào đặc trưng nội dung của sản phẩm nên dễ dàng mở rộng cho nhiều loại item khác nhau.
 - **Nhược điểm:** Rất nhạy cảm với hiện tượng *cold-start* ở cả phía người dùng và sản phẩm, do phụ thuộc mạnh vào dữ liệu tương tác. Để khắc phục, thường cần những kỹ thuật kết hợp thêm thông tin phụ về người dùng và item vào mô hình.
- **Hybrid:** Sử dụng các chiến lược để kết hợp 2 loại mô hình nhằm khắc phục nhược điểm và tối đa hóa ưu điểm của mỗi loại.

Đối với **Collaborative Filtering**, dữ liệu tương tác giữa user và item có thể được chia thành hai loại:

- **Explicit Feedback (Phản hồi tường minh):** Đây là dữ liệu yêu cầu người dùng chủ động đưa ra đánh giá cụ thể về item, ví dụ như cho điểm số (rating) từ 1 đến 5, hoặc like/dislike.
 - **Ưu điểm:** Là dữ liệu chất lượng cao, ít nhiễu hơn so với phản hồi ẩn do user thường ít khi quan tâm đến đánh giá, vì nó trực tiếp phản ánh ý kiến của người dùng.
 - **Nhược điểm:** Khó thu thập hơn, người dùng thường không có động lực để đánh giá tường minh cho mọi vật phẩm họ tương tác. Hoặc nếu có đánh giá thì giá thường đánh giá một cách cực đoan, hoặc rất tốt hoặc rất tệ - thay vì suy xét kỹ mức độ yêu thích của bản thân với item. Vì lí do này mà nhiều nền tảng chuyển sang kiểu đánh giá nhị phân (like/dislike) thay vì rating, nhưng chất lượng của feedback cũng giảm xuống
- **Implicit Feedback (Phản hồi tiềm ẩn):** Đây là dữ liệu không trực tiếp đến từ đánh giá của người dùng mà đến từ việc hệ thống thu thập dữ liệu về hành vi của người dùng, ví dụ như click vào sản phẩm, thời gian sử dụng, tìm kiếm từ khóa, lượt mua, thêm vào giỏ hàng,....
 - **Ưu điểm:** Dễ dàng thu thập với số lượng lớn, vì nó được ghi lại tự động từ các tương tác hàng ngày của người dùng.
 - **Nhược điểm:**

- * **Khó suy luận sở thích:** Sở thích của người dùng khó có thể suy luận ra từ *implicit feedback* do còn phụ thuộc vào rất nhiều yếu tố ngoài, vì vậy chất lượng kém hơn *explicit feedback*. Ví dụ, một người dùng xem một bộ phim nhiều lần có thể vì họ rất thích nó, hoặc vì họ đang nghiên cứu cho một dự án, hoặc đơn giản là họ để quên.
- * **Không có phản hồi tiêu cực rõ ràng:** Dối với *explicit feedback*, có sự phân biệt rõ ràng giữa rating = 0 (đánh giá tệ) và chưa rating (chưa tương tác với item). Còn đối với *implicit feedback* thì ranh giới này rất mơ hồ và khó phân biệt
- * **Độ nhiễu cao:** Độ nhiễu của *implicit feedback* cũng cao hơn nhiều so với *explicit feedback* do các yếu tố bên ngoài ảnh hưởng đến hành vi người dùng.

1.3 Cách tiếp cận của nhóm chúng tôi

Sau giai đoạn phân tích và xác định nhu cầu về dữ liệu, chúng tôi tiến hành thu thập dữ liệu, tập dữ liệu sẽ được giới thiệu ở phần đầu chương 2. Tiếp theo chúng tiến hành bước tiền xử lý, một bước quan trọng để hiểu dữ liệu (data understanding) và chuẩn bị dữ liệu (chuẩn hoá, làm sạch dữ liệu) để phục vụ cho các đầu vào của các mô hình học máy. Sau đó chúng tôi áp dụng cách tiếp cận theo 02 cách tiếp cận là Content-based Filtering và Collaborative-Filtering để huấn luyện mô hình gợi ý. Mục tiêu huấn luyện của chúng tôi là dạy cho mô hình học ra một danh sách top-K gợi ý chất lượng nhất có thể (để đánh giá chất lượng gợi ý chúng tôi sẽ trình bày các độ đo ở phần sau), và sau đó so sánh các phương pháp và đưa ra kết luận cuối cùng.

Chương 2

Tiền xử lý dữ liệu

Trong lĩnh vực khoa học dữ liệu và học máy, chất lượng của dữ liệu đầu vào là yếu tố then chốt quyết định đến độ chính xác và hiệu quả của mọi phân tích hay dự đoán. Quá trình chuẩn bị dữ liệu này được tổ chức một cách có hệ thống, thành một chuỗi các bước liên tiếp mà chúng ta thường gọi là một **"đường ống"** (**pipeline**) tiền xử lý dữ liệu. Mỗi giai đoạn trong đường ống đều đóng vai trò quan trọng trong việc chuyển hóa dữ liệu từ trạng thái ban đầu, dù đã được tinh chỉnh, thành một tập dữ liệu chất lượng cao, nhất quán, và sẵn sàng cung cấp những thông tin giá trị cho mô hình. Các phần tiếp theo trong báo cáo sẽ đi sâu vào giới thiệu các thư viện, công cụ được sử dụng; các bước thực hiện và cài đặt từng giai đoạn trong đường ống tiền xử lý dữ liệu mà nhóm chúng tôi đã xây dựng; từ việc hiểu rõ đặc điểm dữ liệu đã được cung cấp, đến việc làm sạch các vấn đề còn tiềm ẩn, chuẩn bị và biến đổi dữ liệu theo yêu cầu cụ thể.

Để đảm bảo tính liền mạch của chương, ta sẽ giới các thư viện được sử dụng ở mục sau; các công cụ cụ thể sẽ được trình bày tích hợp xuyên suốt báo cáo.

2.1 Đường ống xử lý dữ liệu

2.1.1 Thu thập dữ liệu

Bộ dữ liệu mà nhóm chúng tôi sử dụng trong bài tập lớn là **"Game Recommendations on Steam"**, bộ dữ liệu có thể truy cập online thông qua link: <https://tinyurl.com/neaky77f>. Mặc dù bộ dữ liệu đã được cung cấp sẵn từ nền tảng Kaggle, việc hiểu rõ nguồn gốc và quy trình thu thập dữ liệu ban đầu vẫn rất quan trọng. **Best practice** là luôn kiểm tra tài liệu mô tả bộ dữ liệu (metadata) để hiểu rõ các yếu tố trên, ngay cả khi dữ liệu đã được thu thập sẵn.

Ở đây, nguồn dữ liệu đến từ cơ sở dữ liệu của Steam (tất nhiên được xử lý để không để lộ thông tin riêng tư); giấy phép đi kèm là **CC0: Public Domain**, cho phép tái tạo và sử dụng công khai.

2.1.2 Mô tả tổng quan dữ liệu

Bộ dữ liệu **Game Recommendations on Steam** chứa **hơn 41 triệu** mẫu đề xuất của người dùng ở trên Steam Store (trang mua bán trò chơi, và các vật phẩm liên quan của Steam). Tất cả mẫu dữ liệu đã được làm sạch và tiền xử lý về dạng bảng (file csv) và dạng file json, với các tiêu mục dễ đọc. Bộ dữ liệu gồm ba file dữ liệu quan trọng:

- **games.csv** - file csv chứa thông tin về các trò chơi (hoặc các gói mở rộng), bao gồm đánh giá, giá bán bằng đô la Mỹ \$, ngày phát hành. Một số thông tin bổ sung không có dạng bảng về trò chơi, chẳng hạn như mô tả và thẻ, được lưu trong một tệp metadata;
- **users.csv** - bảng chứa thông tin công khai về hồ sơ người dùng: số lượng sản phẩm đã mua và số lượng đánh giá đã đăng;
- **recommendations.csv** - bảng chứa các đánh giá từ người dùng: liệu người dùng có đề xuất một sản phẩm hay không. Bảng này thể hiện mối quan hệ nhiều-nhiều giữa một thực thể trò chơi và một thực thể người dùng.

Các thông tin riêng của game còn được lưu vào một file **games_metadata.json**. Mỗi vật thể trong file lưu các khóa: app_id, description và tags. Để giới hạn nghiên cứu không chêch sang mảng NLP, ta sẽ không dùng file này.

Bộ dữ liệu không chứa bất kỳ thông tin cá nhân nào về người dùng trên nền tảng Steam. Quy trình tiền xử lý đã ánh xạ tất cả các ID người dùng sang ID mới.

Vì các file được dùng đều ở định dạng csv, ta sẽ dùng thư viện pandas để đọc file (thông qua `.read_csv()`), chuyển về xử lý đối tượng pandas.DataFrame. Đây là đối tượng rất dễ sử dụng, hỗ trợ nhiều phương thức xử lý đa dạng. Ta đọc tương ứng ba file games.csv thành game_df, users.csv thành user_df và recommendations.csv thành recom_df.

Ta xem xét tổng quan dữ liệu. Để rà soát dữ liệu Về kích thước bộ nhớ, ta sử dụng phương thức `.memory_usage()`; về kích cỡ, ta sử dụng trường `.shape`; về các cột, ta sử dụng phương thức `.describe()`

- **games.csv** - kích thước 4.86 MB, gồm 50 872 hàng, 13 cột, bao gồm các trường:
 - **app_id**: ID của phần mềm trên nền tảng (kiểu dữ liệu int64)
 - **title**: Tên của app (kiểu dữ liệu string)
 - **date_release**: Ngày phát hành phần mềm trên cửa hàng, theo khuôn dạng YY-MM-DD để phục vụ sắp xếp (kiểu dữ liệu string)
 - **win**: Phần mềm hỗ trợ hệ điều hành Windows không (kiểu dữ liệu bool)
 - **mac**: Phần mềm hỗ trợ hệ điều hành macOS không (kiểu dữ liệu bool)
 - **linux**: Phần mềm hỗ trợ hệ điều hành Linux không (kiểu dữ liệu bool)
 - **rating**: Xếp hạng độ tuổi của phần mềm (kiểu dữ liệu string)
 - **positive_ratio**: Tỷ lệ đánh giá tích cực của người dùng (kiểu dữ liệu int64)
 - **user_reviews**: Tổng số lượt đánh giá của người dùng (kiểu dữ liệu int64)
 - **price_final**: Giá cuối cùng của phần mềm sau khi áp dụng giảm giá (kiểu dữ liệu float64)
 - **price_original**: Giá gốc của phần mềm trước khi áp dụng giảm giá (kiểu dữ liệu float64)
 - **discount**: Phần trăm giảm giá được áp dụng (kiểu dữ liệu float64)
 - **steam_deck**: Phần mềm có hỗ trợ Steam Deck không (kiểu dữ liệu bool)
- **users.csv** - kích thước 193.07 MB, gồm 14 306 064 hàng, 3 cột; bao gồm các trường:
 - **user_id**: ID của người dùng (kiểu dữ liệu int64)
 - **products**: Số lượng sản phẩm mà người dùng đã sở hữu (kiểu dữ liệu int64)
 - **reviews**: Số lượng đánh giá mà người dùng đã thực hiện (kiểu dữ liệu int64)
- **recommendations.csv** - kích thước 2.02GB, gồm 41 154 794 hàng, 8 cột; bao gồm các trường:
 - **app_id**: ID của phần mềm được đánh giá (kiểu dữ liệu int64)
 - **helpful**: Số lượt đánh giá "hữu ích" cho bài đánh giá (kiểu dữ liệu int64)
 - **funny**: Số lượt đánh giá "hài hước" cho bài đánh giá (kiểu dữ liệu int64)
 - **date**: Ngày người dùng thực hiện đánh giá (kiểu dữ liệu string)
 - **is_recommended**: Trạng thái đề xuất của người dùng (True nếu đề xuất, False nếu không) (kiểu dữ liệu bool)
 - **hours**: Số giờ người dùng đã chơi phần mềm tại thời điểm đánh giá (kiểu dữ liệu float64)
 - **user_id**: ID của người dùng thực hiện đánh giá (kiểu dữ liệu int64)
 - **review_id**: ID của bài đánh giá (kiểu dữ liệu int64)

2.1.3 Phân tích khám phá dữ liệu (EDA)

Phát hiện dữ liệu bị thiếu (missing data)

Chúng tôi thực hiện thao tác kiểm tra xem có trường thông tin nào có chứa dữ liệu **null(None)** hay không thông qua phương thức **.isnull().any()**. Nếu kết quả trả về **false** thì có nghĩa cột tương ứng không chứa null. Sau khi thực hiện thao tác này có thể kết luận rằng **tất cả các cột không có giá trị null**.

app_id	False	user_id	False
title	False	products	False
date_release	False	reviews	False
win	False		
mac	False		
linux	False		
rating	False	app_id	False
positive_ratio	False	helpful	False
user_reviews	False	funny	False
price_final	False	date	False
price_original	False	is_recommended	False
discount	False	hours	False
steam_deck	False	user_id	False
		review_id	False

Hình 2.2: users.csv

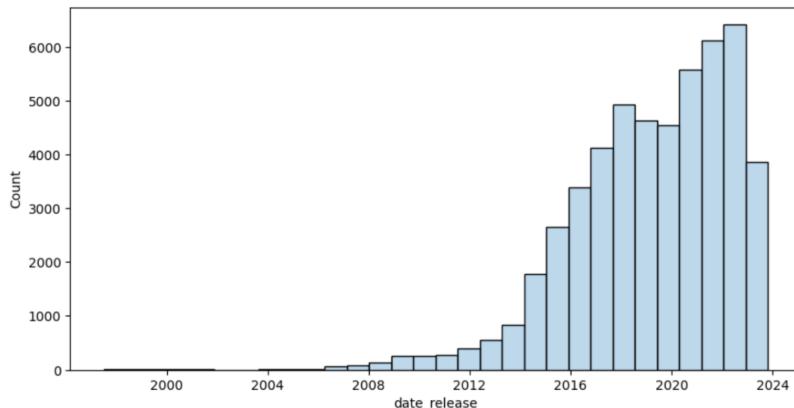
Hình 2.1: games.csv

Hình 2.3: recommendations.csv

Phân tích phân phối dữ liệu (data distribution)

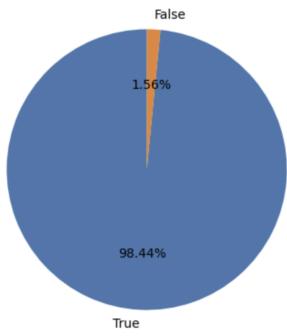
Ở thao tác này, chúng tôi chỉ quan tâm đến các trường có thông tin định lượng/ định tính. Các trường thông tin liên quan đến ID của người dùng, game không được xét đến.

- **games.csv:** Chúng tôi thực hiện các thao tác trực quan hóa trên các trường thông tin **date_release**, **win**, **mac**, **linux**, **rating**, **positive_ratio**, **user_reviews**
 - **Hình vẽ 2.4** cho thấy đa phần các game được ra mắt trong giai đoạn 2020 - 2023 (khả năng có sự tác động của đại dịch COVID-19).

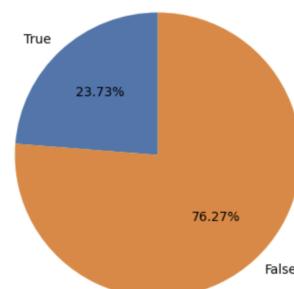


Hình 2.4: Phân phối ngày phát hành game

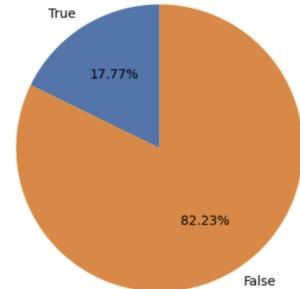
- Các hình vẽ 2.5, 2.6, 2.7 cho thấy gần như tất cả các game đều hỗ trợ hệ điều hành Windows, phần lớn các game không hỗ trợ hệ điều hành MacOS, Linux.



Hình 2.5: Tỉ lệ các game hỗ trợ Windows

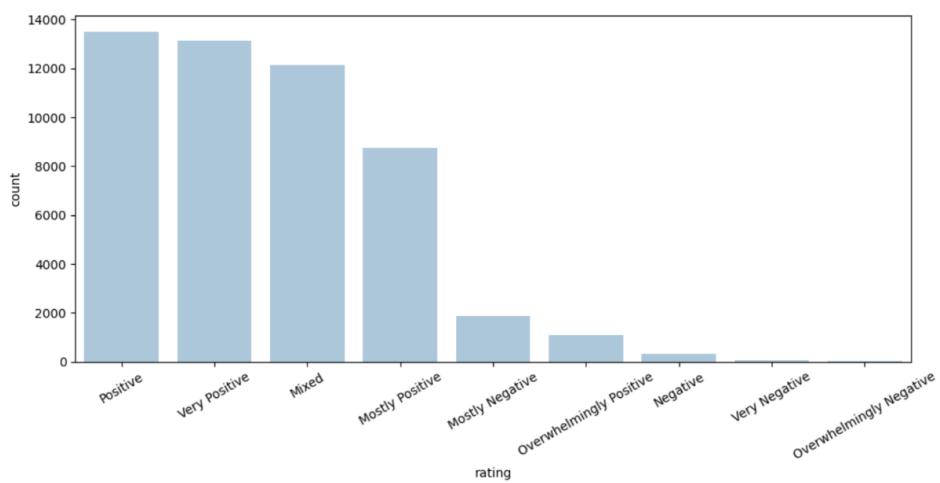


Hình 2.6: Tỉ lệ các game hỗ trợ MacOS

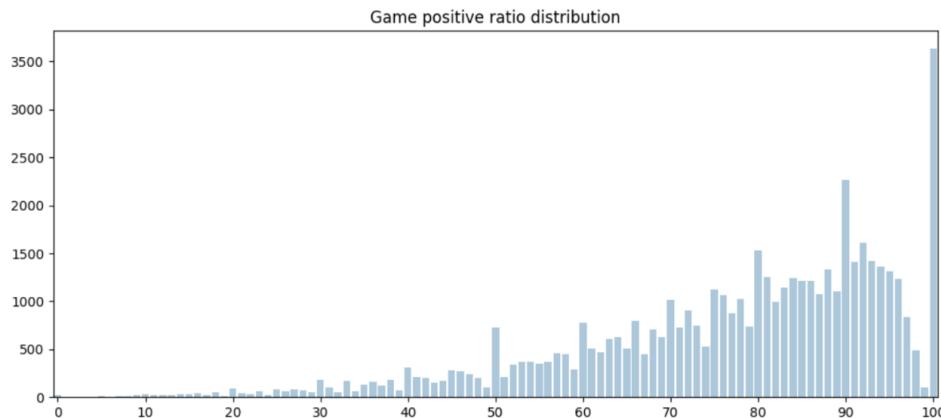


Hình 2.7: Tỉ lệ các game hỗ trợ Linux

- Các hình vẽ 2.8, 2.9 cho thấy phần lớn các loại rating hướng đến đánh giá tích cực, tỉ lệ đánh giá tích cực khá cao, nhảy vọt ở các tỉ lệ phần trăm như 70, 80, 90, 100.

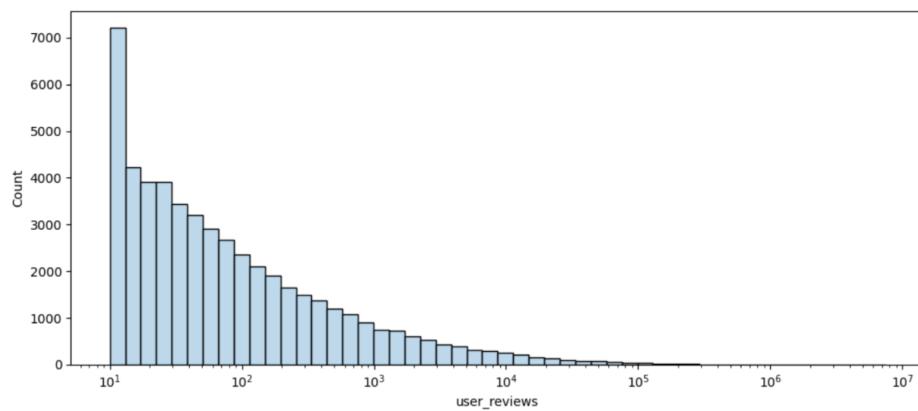


Hình 2.8: Phân phối đánh giá của người dùng



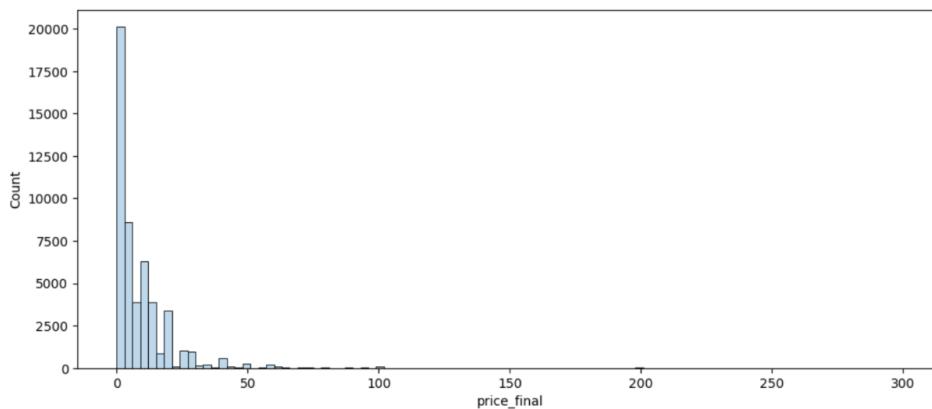
Hình 2.9: Phân phối tỉ lệ đánh giá của người dùng

- **Hình vẽ 2.10** cho thấy hầu hết các game được review rất ít, tuy nhiên một số ngoại lệ lại có được số review rất nhiều, lên đến gần 10 triệu. Khi biểu diễn phân phối áp dụng log-scale vẫn bị long-tailed, điều này cho thấy dữ liệu này tuân theo phân phối pareto (https://en.wikipedia.org/wiki/Pareto_distribution)

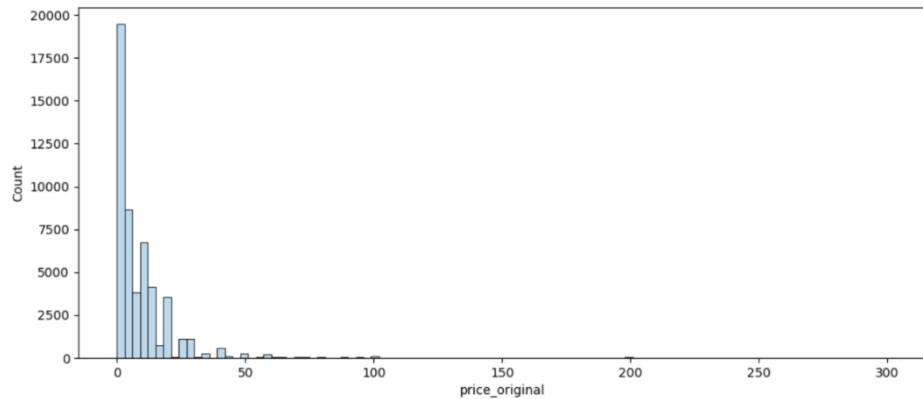


Hình 2.10: Phân phối số lượng đánh giá của người dùng

- **Các hình vẽ 2.11, 2.12** cho thấy phân phối giá bán game cuối cùng có những điểm ngoại lai rất xa. Các mức giá tròn chục, tròn trăm thường đông hơn. Phân phối giá bán game ban đầu (price_original) tương tự giá bán game cuối cùng (price_final).

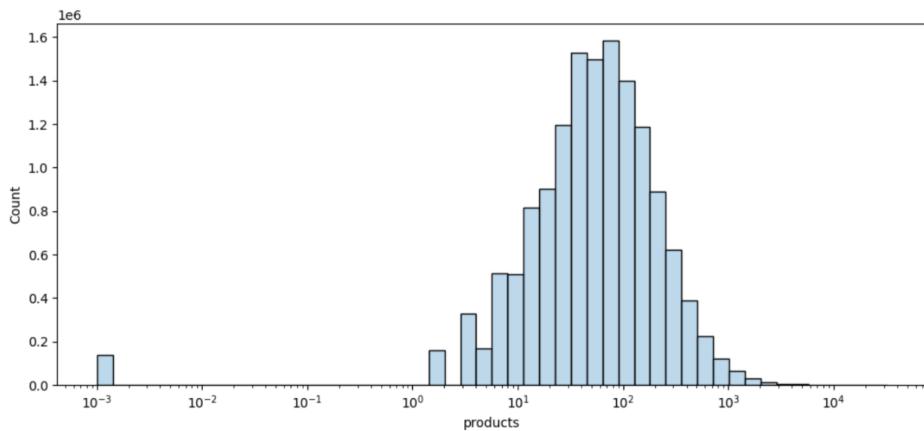


Hình 2.11: Phân phối giá bán game cuối cùng (price final)

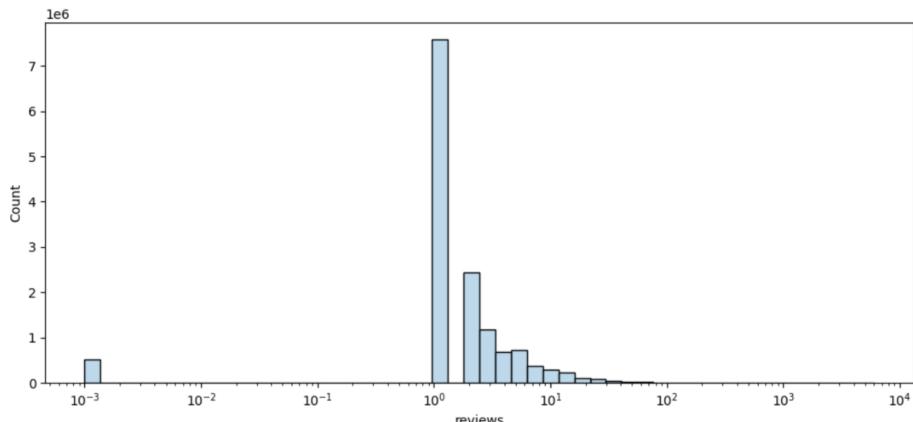


Hình 2.12: Phân phối giá bán game ban đầu (price original)

- **users.csv:** Chúng tôi thực hiện trực quan hóa trên các trường thông tin **products**, **reviews**. Các hình vẽ 2.13, 2.14 thì cũng thấy đa phần người dùng review rất ít, chỉ chơi tầm 10 - 200 game.



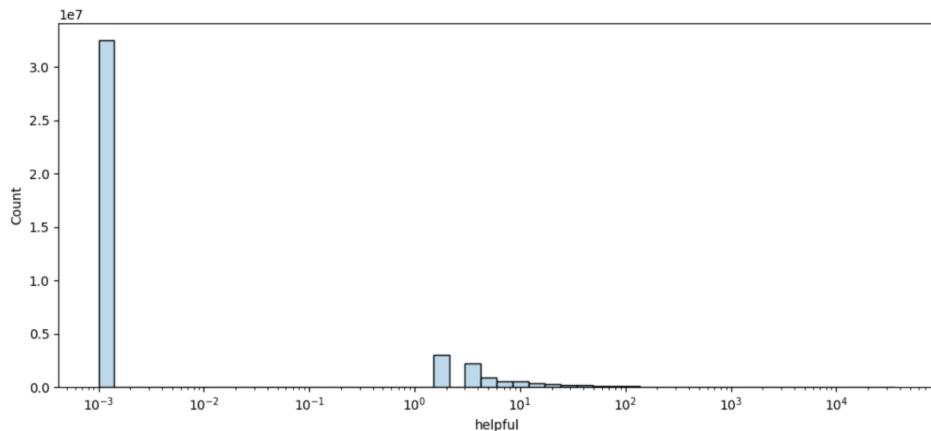
Hình 2.13: Phân phối số lượng sử dụng sản phẩm (chơi game) của người dùng



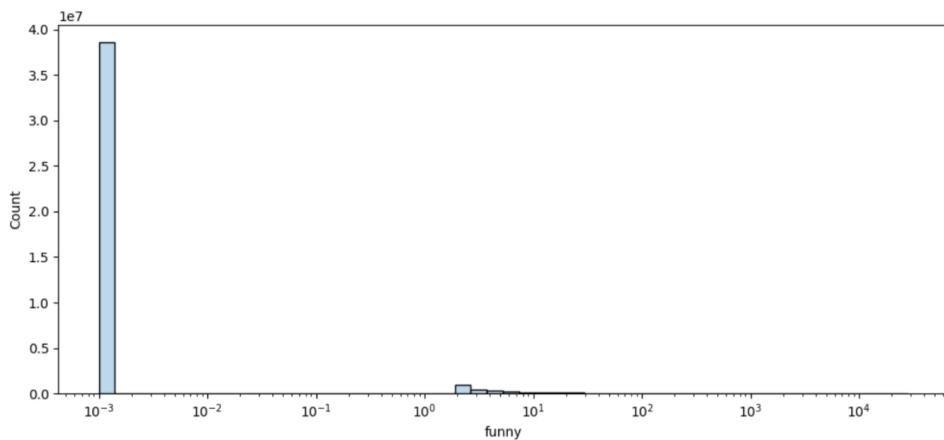
Hình 2.14: Phân phối số lượng đánh giá (reviews) của người dùng

- **recommendations.csv:** Chúng tôi thực hiện trực quan hóa trên các trường thông tin **helpful**, **funny**, **date**, **is_recommended**

- Các hình vẽ 2.15, 2.16 cho thấy hầu hết đánh giá (review) đều không có đánh giá helpful, funny (do quá ít đánh giá)

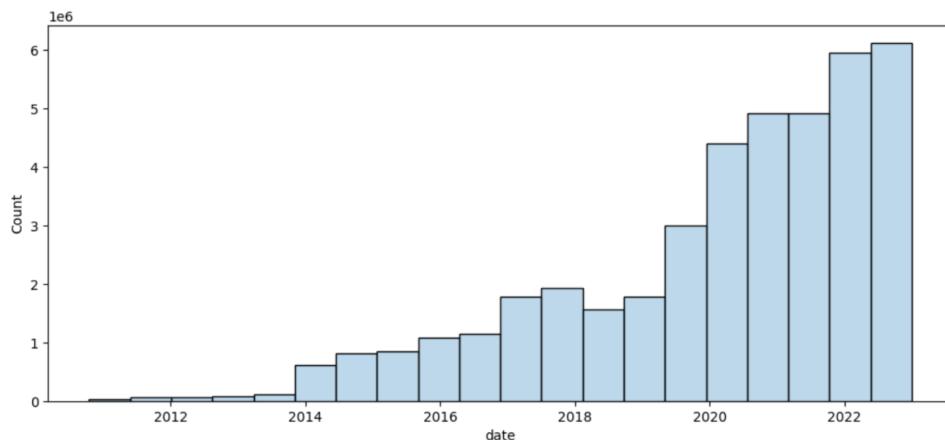


Hình 2.15: Phân phối đánh giá (review) là helpful



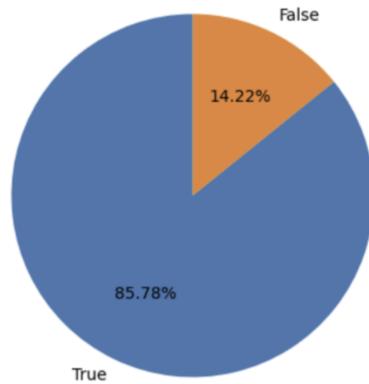
Hình 2.16: Phân phối đánh giá (review) là funny

- **Hình vẽ 2.17** cho thấy phân bố khá tương đồng với date _ release của games.csv, phản ánh sự phát triển của ngành game theo thời gian.



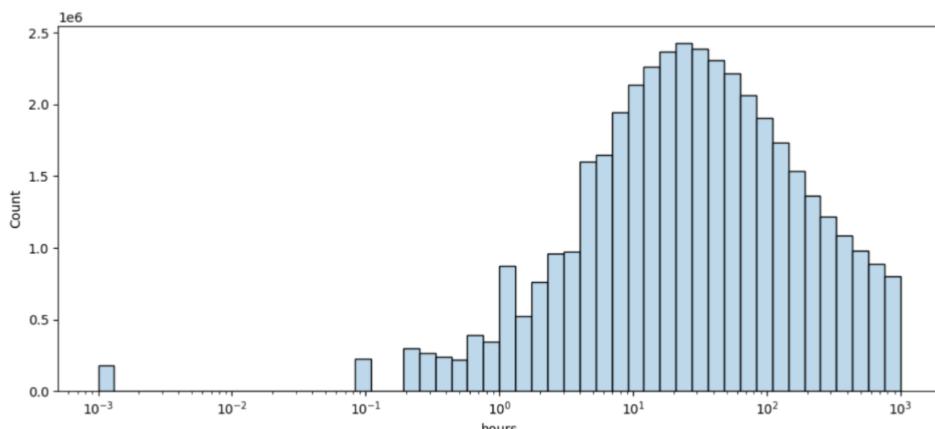
Hình 2.17: Phân phối ngày phát hành của game

- **Hình vẽ 2.18** cho thấy phần lớn các đánh giá là tích cực, đây là vấn đề tiềm ẩn về tình trạng mất cân bằng (imbalanced) các mẫu dữ liệu.



Hình 2.18: Tỉ lệ người dùng dè xuất game

- **Hình vẽ 2.19** cho thấy phân bố khá đồng đều khi ở log-scale. Phân phối này cho thấy thời gian chơi đã được cắt ở mức trần là 1000 giờ.

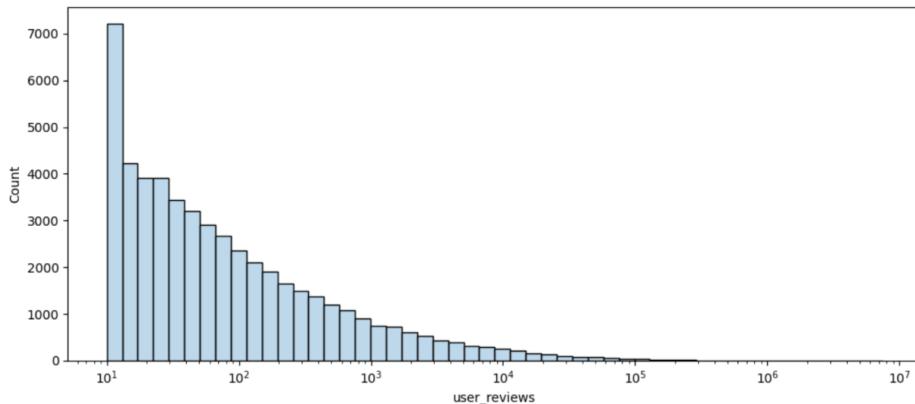


Hình 2.19: Phân phối số giờ chơi game của người dùng

Phát hiện ngoại lai

Chúng tôi thấy hầu hết các phân phối của các cột dữ liệu liên tục trong cả ba file đều nhất quán, không có sự nhiễu loạn lớn nào xuất hiện. Số ít ngoại lai còn sót lại gồm:

- Một số phần mềm có date_release dưới 2000
- Rất nhiều phần mềm có positive_ratio đạt 99-100 phần trăm. Ta sẽ xem xét phân bố về số lượt review của các phần mềm như vậy:



Điều này chứng tỏ các phần mềm này chẳng qua có số review rất ít, nên positive_ratio mới ở mức tuyệt đối như vậy. Số lượng review này có thể được làm giả dễ dàng bởi các shilling bot. Vậy nên, để loại bỏ khả năng trên, ta cần khử các phần mềm chưa thực sự nổi tiếng.

- Một số phần mềm có giá rất cao, hơn 100 đô la. Thậm chí có các phần mềm có giá đến 299 đô la. Tuy nhiên chỉ có 28 phần mềm có giá hơn 100 đô.
- Sự tương tác đánh giá ở recom_df (helpful, funny) có phân phối vô cùng lệch trái, với những điểm ngoại lai vô cùng hiếm và rất lớn về bên phải
- Một số không nhỏ những review thuộc về những người chưa từng dùng một giờ nào/dùng rất ít vào những phần mềm đó. Những đánh giá này sẽ có thể có thiên kiến.

2.1.4 Làm sạch dữ liệu (Data Cleaning)

Quá trình làm sạch dữ liệu nhằm mục đích xử lý các vấn đề về chất lượng dữ liệu như thiếu giá trị, ngoại lai; dữ liệu trùng lặp, không nhất quán, quá thừa. Điều này đảm bảo dữ liệu đầu vào là chính xác, màu mè, đáng tin cậy để phục vụ cho các bước phân tích và mô hình hóa tiếp theo.

2.1.4.1 Xử lý giá trị ngoại lai:

Từ những nhận xét về ngoại lai ở bước EDA, ta quyết định:

- Đối với game_df:
 - Khử những phần mềm quá cũ, dưới năm **2008** hoặc những phần mềm quá mới, từ năm 2024 trở đi.
 - Khử những phần mềm có user_reviews dưới **5000**
- Đối với user_df:
 - Khử những đánh giá bởi người dùng sử dụng phần mềm ít hơn **0.5** giờ (30 phút).

2.1.4.2 Xử lý dữ liệu trùng lặp:

Để đảm bảo chắc chắn không có dữ liệu trùng lặp, ta sử dụng phương thức `.drop_duplicates()` cho cả ba object game_df, user_df và recom_df.

2.1.4.3 Xử lý dữ liệu không nhất quán về logic

Các dữ liệu hạng mục và dữ liệu liên tục đều cho thấy sự tự nhiên. Vậy nên, ta nhắm tới các dữ liệu về thời gian. Ta cần đảm bảo rằng dữ liệu các cột về thời gian có đúng thứ tự nhân quả không. Ta sử dụng phương thức `.set_index` để tạo ra một dictionary ánh xạ mỗi app_id với release_date, sau đó chèn cột release_date vào recom_df bằng phương thức `.map()`. Cuối cùng, ta lọc để chỉ giữ lấy các hàng có ngày review được viết (cột 'date' của recom_df) sau release_date của app (cột 'date_release' mới được thêm vào).

2.1.4.4 Xử lý dữ liệu không có tương tác

Ta sẽ sử dụng phương thức `.isin(clt)` để lọc một nhãn nhị phân (boolean mask) biểu diễn một phần tử trong một pandas.Series này có nằm trong collection `clt` không. Từ đó, ta lọc chỉ lấy những phần tử thỏa mãn.

2.1.4.5 Xử lý dữ liệu thừa

Ta sử dụng phương thức `.value_counts()` để thu được một pandas.Series đếm tần suất của mỗi phần tử, từ đó lọc lấy các người dùng tương tác với ít nhất **20** phần mềm.

2.1.4.6 Xử lý định dạng dữ liệu

Bộ dữ liệu đã được định dạng rất tốt, các định dạng dữ liệu là nhất quán trong nội bộ mỗi đặc trưng. Mặc dù thế, các định dạng dữ liệu và đơn vị vẫn chưa đủ gần để có thể xử lý bởi các thuật toán. Ta tiến hành một số bước thay đổi đơn vị:

- Các cột dữ liệu thời gian: Đổi từ string về định dạng `pandas.DateTime`
- Các cột dữ liệu kiểu bool: Đổi về kiểu int32 (0 cho False và 1 cho True) thông qua phương thức `.astype(int)`

Sau các bước làm sạch dữ liệu, tập dữ liệu đã được cải thiện đáng kể về chất lượng. Đồng thời, số lượng bản ghi cũng giảm xuống đáng kể.

- `game_df`: 1721 bản.
- `user_df`: 104 238 bản.
- `recom_df`: 3 415 488 bản.

2.1.5 Chuẩn bị dữ liệu (Data Preparation)

Sau khi làm sạch, dữ liệu tiếp tục được biến đổi, thêm mới các đặc trưng để phù hợp với yêu cầu của các thuật toán học máy và các công cụ phân tích.

Ở trong các DataFrame, với mỗi phép biến đổi sẽ tạo nên một cột mới, không ghi đè các cột dữ liệu gốc, tên cột được thêm hậu tố `_tfdf`.

2.1.5.1 Biến đổi (Transformation)

Rất nhiều đặc trưng liên tục của bộ dữ liệu gấp phải tình trạng *long-tailed*. Vì thế, từ phân phối dữ liệu ở phần EDA, cộng thêm các khảo sát, các biến đổi chúng tôi quyết định sử dụng ba phép biến đổi:

- **Log-transform**. Sử dụng để xử lý phân phối mũ. Ta sử dụng `sklearn.preprocessing.FunctionTransformer`, nạp vào hàm `log1p` của numpy. Kí hiệu phép này là `log1p`
- **Box-Cox transform**. Sử dụng để xử lý các phân phối mạnh hơn cả phân phối mũ (như phân phối Pareto) Công cụ được dùng là `sklearn.preprocessing.PowerTransformer` với `method='box-cox'`. Kí hiệu phép này là `box-cox`
- **Yeo-Johnson transform**. Công dụng tương tự Box-cox, nhưng với đặc điểm rằng phép này 'nhẹ tay' hơn, đồng thời xử lý được các số không dương. Công cụ trong sklearn tương ứng là `sklearn.preprocessing.PowerTransformer` với `method='yeo-johnson'` Kí hiệu phép này là `Yeo-Johnson`

Nếu không dùng biến đổi nào, phép biến đổi được ký hiệu là '**giữ nguyên**'.

Các phép biến đổi được dùng ứng với mỗi đặc trưng là:

- **games.csv**
 - positive_ratio: **log1p**
 - user_reviews: **box-cox**
 - price_final/price_original: **yeo-johnson**
 - discount: **giữ nguyên**
- **users.csv**
 - products: **log1p**
 - reviews: **box-cox**
- **recommendations.csv**
 - helpful: **yeo-johnson**
 - funny: **yeo-johnson**
 - date: **giữ nguyên**
 - hours: **log1p**

2.1.5.2 Thu phóng dữ liệu: Normalization hay Standardization?

Sau khi đã biến đổi (transform) các cột dữ liệu, các thuộc tính vẫn phân bố ở các miền khác nhau, với độ lệch khác nhau. Khi này, hai phép biến đổi tuyến tính (scale) có thể được sử dụng là Normalization và Standardization (định nghĩa tiếng Việt xem ở phần Cơ sở lý thuyết). Nhưng ta cần chọn phép nào?. Ở bài toán RS, hầu như tất cả các đặc trưng liên tục (số giờ chơi, số lượng tương tác, số tiền đều không âm). Vì thế, Standardization có thể không phù hợp, vì nó khiến nhiều điểm dữ liệu âm. Biểu diễn như này sẽ làm mất đi tri thức cơ bản của bài toán.

Vì thế, phép biến đổi được dùng là Normalization. Về công cụ, ta sử dụng đối tượng **MinMaxScaler()** của thư viện **sklearn.preprocessing**.

2.1.5.3 Mã hóa đặc trưng hạng mục

Cột dữ liệu phân loại duy nhất của chúng ta là 'rating' của game_df. Đây là cột dữ liệu quan trọng. Để hiểu đúng về ý nghĩa của các đánh giá (chẳng hạn: 'Overwhelmingly Positive' khác gì 'Very Positive', 'Mixed' có nghĩa là gì), ta cần xem lại về quy tắc đánh giá của Steam.

Dựa trên tra cứu ở đường dẫn: https://www.reddit.com/r/Steam/comments/2rygk7/list_of_steam_user_review_rating_levels/?rdt=40287, ta thấy quy tắc của rating như sau:

- 95 - 100 | 500+ reviews | positive | overwhelming
- 85 - 100 | 50+ reviews | positive | very
- 80 - 100 | 1+ reviews | positive
- 70 - 79 | 1+ reviews | positive | mostly
- 40 - 69 | 1+ reviews | mixed
- 20 - 39 | 1+ reviews | negative | mostly
- 0 - 19 | 1+ reviews | negative
- 0 - 19 | 50+ reviews | negative | very
- 0 - 19 | 500+ reviews | negative | overwhelming

Đây là một quy tắc tuy phức tạp, nhưng rất công bằng, đảm bảo những phần mềm không được review nhiều sẽ không bị đẩy xuống đánh giá quá thấp hoặc quá cao. Để giữ tính thứ tự của đánh giá, ta sẽ chuyển nó về các nhãn số như sau:

- Overwhelming Negative: 0
- Very Negative: 1
- Negative: 2
- Mostly Negative: 3
- Mixed: 4
- Mostly Positive: 5
- Positive: 6
- Very Positive: 7
- Overwhelmingly Positive: 8

Ta sử dụng phương thức `.map()` để ánh xạ mỗi nhãn của thuộc tính này đến số tương ứng.

2.1.5.4 Rời rạc hóa

Đến bước này, ta thấy còn dữ liệu thời gian vẫn chưa được đưa về dạng số. Vì vậy, ta sẽ xây dựng các đặc trưng hạng mục(categorical) từ trường này thông qua kỹ thuật Phân nhóm theo khoảng (Binning). Ta sẽ phân nhóm theo năm/quý/tháng, thu được thêm ba trường mới. Về công cụ, ta sử dụng phương thức `.dt` để trả về object `DateTime`, `.year/.quarter/.month` cho năm/quý/tháng và `.astype('category')` để đưa về dạng số theo nhãn.

2.1.5.5 Tích hợp dữ liệu (Data Integration):

Các đặc trưng dữ liệu trong `game_df`, `user_df` vẫn tách biệt so với bảng tương tác `recom_df`. Để thuận tiện hơn cho các mô hình học máy, ta sẽ gộp ba bảng này thành một bảng thứ tư: `inter_df`. Bảng này có số bản ghi bằng với `recom_df`, nhưng sẽ có thêm các cột dữ liệu: từ `game_df` ứng với thuộc tính '`app_id`' và từ `user_df` ứng với thuộc tính '`user_id`'. Ta sử dụng phương thức `.merge()` của pandas để gộp hai bảng lại, thêm đối số `how='left'` để gộp dựa trên bảng gọi phương thức.

2.1.5.6 Lựa chọn đặc trưng:

Ta sử dụng mô hình `RandomForestClassifier` từ `sklearn.ensemble` để khớp (fit) dữ liệu (sau khi bỏ 4 cột thông tin tương tác) với cột '`is_recommended`'. Sau khi đã huấn luyện mô hình, ta gọi trường `.feature_importance_` để xem kết quả.

Kết quả độ quan trọng của các thuộc tính như sau (độ quan trọng là thang đo được chuẩn hóa, tổng các số đo bằng 1):

- `positive_ratio_tfd`: 0.5168
- `rating_tfd`: 0.15665
- `user_reviews_tfd`: 0.07515
- `year`: 0.04554
- `products_tfd`: 0.04257
- `reviews_tfd`: 0.04094
- `price_final_tfd`: 0.03231
- `month`: 0.02447
- `month_release`: 0.01546
- `year_release`: 0.01422
- `price_original_tfd`: 0.01305

- quarter: 0.0077
- quarter_release: 0.00547
- mac: 0.00528
- linux: 0.00345
- discount: 0.00091
- win: 1e-05
- steam_deck: 1e-05

Từ phân tích trên, nhóm tác giả quyết định bỏ các trường: '*win*', '*mac*', '*linux*', '*discount*', '*steam_deck*', '*price_original*', '*price_original_tfd*'.

2.1.5.7 Chia tập dữ liệu

Nhóm tác giả thống nhất chia tập dữ liệu theo tỉ lệ 60-20-20. Tức là 60% dành cho tập *train*, 20% giành cho tập *validation* và 20% giành cho tập *test*. Vì dữ liệu tương tác có tính ảnh hưởng bởi thời gian (time_series) rất cao, vậy nên nhóm tác giả chia các tập trên theo thứ tự thời gian. Tập *train* bao gồm các tương tác sớm nhất, rồi đến tập **validation**, cuối cùng là tập *test*.

2.1.5.8 Xuất dữ liệu về các file csv

Các DataFrame, bao gồm ba tập game_df, user_df, recom_df, tập tương tác tổng inter_df và ba tập dữ liệu chia theo thời gian recom_train, recom_val và recom_test được xuất trả lại dưới dạng file csv. Toàn bộ tập dữ liệu đã được xử lý, cùng với các tập dữ liệu được sử dụng riêng cho các mô hình, được lưu ở folder mã nguồn đi kèm với báo cáo này

2.1.6 Kết luận

Dựa trên các phân tích lý thuyết về quy trình tiền xử lý dữ liệu trong khoa học dữ liệu, bám sát theo yêu cầu bài toán Hệ thống gợi ý và các đặc thù của bộ dữ liệu được cung cấp, nhóm tác giả đã tiến hành triển khai các bước làm sạch và chuẩn bị dữ liệu một cách cẩn thận. Kết quả của giai đoạn tiền xử lý này là một tập dữ liệu đã được tinh chỉnh, sẵn sàng để được sử dụng hiệu quả trong việc xây dựng và đánh giá các mô hình Hệ thống gợi ý.

Chương 3

Cơ sở lý thuyết

3.1 Hướng tiếp cận Content-based Filtering

Phương pháp **Content-Based Filtering (CBF)** là một trong những kỹ thuật phổ biến được sử dụng trong hệ thống gợi ý (recommender system). Phương pháp này hoạt động dựa trên giả định cơ bản rằng: người dùng có xu hướng ưa thích những trò chơi (hoặc sản phẩm) có nội dung tương tự với những trò chơi mà họ đã yêu thích hoặc đánh giá cao trong quá khứ.

Trong mô hình này, mỗi trò chơi (game) sẽ được biểu diễn bằng một **vector đặc trưng (feature vector)** – đây là một biểu diễn định lượng mô tả các thuộc tính nội dung của trò chơi đó. Các thuộc tính này có thể bao gồm thể loại (genre), cơ chế chơi (gameplay mechanics), nhà phát hành, mức độ phổ biến, nền tảng hỗ trợ, nhãn phân loại (tags), v.v.

Vector đặc trưng của game có thể là nhị phân (biểu thị sự có mặt hay không của một đặc trưng) hoặc dạng số (phản ánh cường độ hoặc tần suất).

Tương tự, mỗi người dùng cũng sẽ được biểu diễn bằng một vector sở thích (user profile vector). Vector này được học dựa trên các trò chơi mà người dùng đã tương tác trong quá khứ, chẳng hạn như các game họ đã chơi, đã đánh giá, hoặc đã mua. Quá trình xây dựng vector sở thích thường bao gồm việc tổng hợp hoặc trung bình các vector đặc trưng của những game mà người dùng yêu thích, có thể có trọng số theo mức độ tương tác (ví dụ: số giờ chơi, mức đánh giá).

Khi hệ thống cần gợi ý game mới cho người dùng, nó sẽ tiến hành so sánh vector sở thích của người dùng với vector đặc trưng của từng game trong tập dữ liệu. Mức độ tương đồng giữa hai vector này phản ánh mức độ phù hợp giữa game đó và sở thích của người dùng.

Trong phương pháp sử dụng **cosine similarity**, ta đo độ tương đồng giữa profile của người dùng và profile của game thông qua góc giữa hai vector đặc trưng. Sau khi tính toán độ tương đồng giữa profile người dùng và các game, hệ thống sẽ xếp hạng các game theo thứ tự giảm dần của độ tương đồng và gợi ý cho người dùng những trò chơi có độ tương đồng cao nhất.

Tổng quát hóa, phương pháp **Content-Based Filtering** cung cấp một cơ chế trực tiếp và rõ ràng để đưa ra gợi ý cá nhân hóa, dựa trên việc phân tích nội dung của các trò chơi và sở thích đã biết của người dùng. Tuy nhiên, phương pháp này cũng có một số hạn chế như khả năng gợi ý bị giới hạn bởi phạm vi sở thích hiện tại của người dùng (không khuyến khích sự khám phá), hoặc khó xử lý khi thông tin nội dung game không đầy đủ hoặc không chuẩn hóa.

3.1.1 Xây dựng đặc trưng

Trong phần này, mỗi game có thể được mô tả bởi một loạt các đặc trưng nội dung (content features), ví dụ:

- Thể loại (Genres): Action, Adventure, Strategy, Indie, Simulation, RPG, v.v.

- Tính năng (Features): Single-player, Multi-player, Online PvP, Controller support, Steam Cloud, v.v.
- Nhà phát hành (Developer/Publisher).
- Dánh giá trung bình (User Reviews).
- Thời gian phát hành (Release Date).
- Tags người dùng gán (User tags).
- Hệ điều hành, giá cả, v.v.

Các thông tin này có thể được chuyển về dạng vector nhị phân hoặc dạng số thông qua các kỹ thuật one-hot encoding, TF-IDF. Khi đó, mỗi game được biểu diễn bởi một vector đặc trưng $x_{game} \in \mathbb{R}^d$.

Với mỗi user u , ta tạo một profile θ_u bằng cách trung bình các vector đặc trưng của các game họ đã yêu thích hoặc chơi nhiều:

$$\theta_u = \text{mean}(\{x_i \mid i \in I_u\})$$

I_u là tập các game mà user u đã tương tác tích cực (ví dụ: thời gian chơi nhiều, đánh giá tốt, v.v.).

Độ đẽ tương đồng: Cosine Similarity

Để dự đoán mức độ phù hợp giữa user u và game j , ta tính cosine similarity giữa θ_u và x_j :

$$\text{sim}(u, j) = \cos(\theta_u, x_j) = \frac{\theta_u \cdot x_j}{\|\theta_u\| \cdot \|x_j\|}$$

Giá trị cosine similarity nằm trong khoảng [0, 1] (nếu vector không âm), càng cao thì mức độ tương đồng càng lớn thì càng được khuyến nghị.

3.1.2 Ưu điểm và Hạn chế

Ưu điểm:

- Đơn giản, trực quan, dễ cài đặt.
- Không cần huấn luyện mô hình riêng biệt.
- Giải thích được gợi ý dựa trên đặc trưng nội dung game.
- Khắc phục cold-start cho game mới, miễn là có đủ metadata.

Hạn chế:

- Over-specialization: thường chỉ gợi ý game giống những game đã chơi, thiếu sự đa dạng.
- Không tận dụng collaborative signals, bỏ qua các game được người khác có thị hiếu tương tự đánh giá cao.
- Phụ thuộc vào chất lượng đặc trưng game, thông tin không đầy đủ hoặc không chính xác sẽ ảnh hưởng kết quả.

3.2 Hướng tiếp cận Collaborative Filtering

Trong nghiên cứu này, ngoài phương pháp ItemKNN, tất cả các mô hình áp dụng đều dựa trên **representation learning** trong Collaborative Filtering (CF). Phương pháp này tập trung vào việc học các biểu diễn tiềm ẩn (latent representations) cho người dùng (user) và sản phẩm (item) trực tiếp từ dữ liệu tương tác.

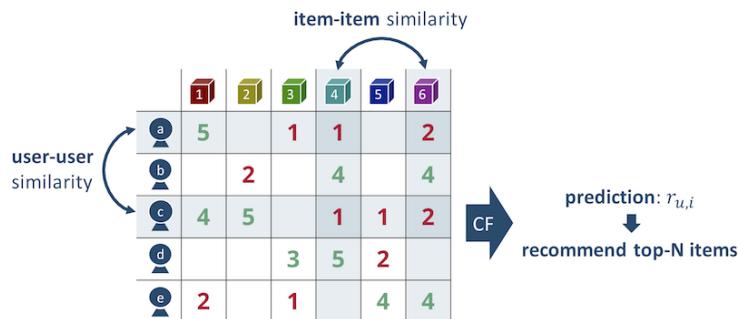
Mục tiêu chính của hướng tiếp cận là tự động trích xuất và học các **vector đặc trưng (embeddings)** cho user và item dựa trên lịch sử đánh giá (ratings), thay thế cho việc thiết kế đặc trưng thủ công. Các vector này cho phép ước lượng mức độ tương đồng giữa user và item, hỗ trợ việc dự đoán mức độ yêu thích hoặc tương tác trong các mô hình MF, NeuMF, NGCF, LightGCN,...

Theo đó, quy trình chung bao gồm:

1. Xây dựng ma trận tương tác user-item.
2. Thiết lập kiến trúc học biểu diễn (MF, NeuMF, NGCF, LightGCN,...)
3. Huấn luyện mô hình để học embeddings tối ưu dưới hàm mất mát phù hợp (BCE, BPR).
4. Sử dụng embeddings thu được để dự đoán rating hoặc xếp hạng đề xuất.

3.2.1 Neighbourhood Models

Một trong những mô hình đơn giản và trực quan nhất về mặt ý tưởng trong hệ gợi ý là lọc cộng tác dựa trên **K-Nearest Neighbors (KNN)**. Đây là một mô hình không cần huấn luyện (còn gọi là *lazy learning* hoặc *memory-based model*) khi toàn bộ quá trình suy luận được thực hiện trực tiếp từ dữ liệu quan sát ban đầu.



Hình 3.1: Neighbourhood-based Collaborative Filtering

Có 2 hướng triển khai chính cho phương pháp này:

(1) User-based: Phương pháp này dựa trên 2 giả định: (i) sở thích của mọi user không thay đổi theo thời gian, và (ii) các user có sở thích giống nhau thì trong tương lai cũng sẽ có sở thích giống nhau. Dự đoán độ phù hợp $f(u, i)$ giữa user u và item i được thực hiện như sau:

- Tìm tập hợp K user khác u đã từng tương tác với i và có lịch sử tương tác giống với u nhất.
- Tính $f(u, i)$ bằng cách tổng hợp đánh giá của tập user này trên i , thông qua trung bình có trọng số hoặc dựa trên quy tắc số đông.

(1) Item-based: Cũng tương tự với *User-based*, phương pháp này tìm kiếm sự tương đồng giữa các item thay vì giữa các user. Quá trình tính $f(u, i)$ như sau:

- Tìm tập hợp K item trong số những item người dùng u đã từng tương tác trong quá khứ mà giống với i nhất. Mỗi item được biểu diễn bởi đánh giá của tất cả user trên item này.
- Tính $f(u, i)$ dựa trên tổng hợp đánh giá của u trên tập K item.

Sự giống nhau giữa các đối tượng được tính bởi một độ đo tương đồng $sim(o_1, o_2)$, trong đó o_1, o_2 là các vector trong ma trận tương tác (cột nếu là user, hàng nếu là item). Để hạn chế sự ảnh hưởng từ số lượng tương tác của mỗi đối tượng mà chỉ tập trung vào xu hướng chung, độ đo tương đồng thường được sử dụng trong lọc cộng tác là *Cosine Similarity*.

$$sim(o_1, o_2) = \cos(\vec{o}_1, \vec{o}_2) = \frac{\vec{o}_1^\top \vec{o}_2}{\|\vec{o}_1\| \cdot \|\vec{o}_2\|} = \frac{\sum_k o_{1k} \cdot o_{2k}}{\sqrt{\sum_k o_{1k}^2} \cdot \sqrt{\sum_k o_{2k}^2}}$$

Mặc dù cả hai hướng triển khai User-based và Item-based đều khả thi trên lý thuyết, Item-based thường là lựa chọn hợp lý hơn trong nhiều tình huống thực tiễn.

Thứ nhất, user-based đang dựa trên giả thiết về sự bất biến của sở thích người dùng, nhưng trên thực tế điều này còn phụ thuộc vào lĩnh vực và loại sản phẩm mà hệ gợi ý đang áp dụng. Và trong phần

lớn trường hợp, giả thuyết này là không đúng khi sở thích cá nhân thường thay đổi theo thời gian và rất dễ dàng bị tác động bởi xu hướng chung của xã hội. Ngược lại, các đặc trưng riêng của sản phẩm thường không thay đổi hoặc ít được cập nhật, từ đó giúp mô hình item-based có sự khái quát ổn định hơn về mặt thời gian.

Thứ hai là xét về mặt chi phí tính toán và lưu trữ. Số lượng người dùng trong các nền tảng thường lớn hơn rất nhiều so với lượng sản phẩm, đặc biệt là trên các nền tảng quy mô lớn. Điều này khiến việc đánh giá độ phù hợp trên toàn bộ tập người dùng và sắp xếp là khó khăn hơn nhiều so với việc xét trên tập sản phẩm. Một hướng tiếp cận đơn giản để xử lý điều này là lưu trữ trước top K người dùng tương đồng, nhưng điều này sẽ phải đánh đổi về kích thước lưu trữ, nhất là khi K lớn.

Tổng hợp lại các lí do trên, item-based là lựa chọn phổ biến hơn so với user-based. Mô hình này được trình bày trong Thuật toán 1.

Algorithm 1 Item-based KNN for Rating Prediction

Require: Tập user \mathcal{U} , tập item \mathcal{I} , ma trận tương tác $\mathbf{R} \in \mathbb{R}^{|\mathcal{U}| \times |\mathcal{I}|}$, số lượng hàng xóm K

Ensure: Dự đoán độ phù hợp $f(u, i)$ với mỗi cặp (u, i)

- 1: **for all** $u \in \mathcal{U}$ **do**
- 2: **for all** $i \in \mathcal{I}$ **do**
- 3: Tính độ tương đồng giữa item i và từng item j mà user u đã từng tương tác:

$$\text{sim}(i, j) = \frac{\sum_{u \in \mathcal{U}} \mathbf{R}_{u,i} \cdot \mathbf{R}_{u,j}}{\sqrt{\sum_{u \in \mathcal{U}} \mathbf{R}_{u,i}^2} \cdot \sqrt{\sum_{u \in \mathcal{U}} \mathbf{R}_{u,j}^2}}$$

- 4: Lấy tập N_i^K gồm K item tương đồng nhất với i trong số các item j đã được user u tương tác
- 5: Dự đoán độ phù hợp $f(u, i)$ bằng trung bình có trọng số:

$$f(u, i) = \frac{\sum_{j \in N_i^K} \text{sim}(i, j) \cdot \mathbf{R}_{u,j}}{\sum_{j \in N_i^K} |\text{sim}(i, j)|}$$

- 6: **end for**
 - 7: **end for**
-

Mặc dù có cấu trúc đơn giản, phương pháp KNN vẫn cho thấy hiệu quả tương đối tốt trong nhiều trường hợp nhờ khả năng khai thác trực tiếp dữ liệu tương tác mà không cần huấn luyện phức tạp. Tuy vậy, nhược điểm lớn nhất của phương pháp này nằm ở độ phức tạp của bước suy diễn, đặc biệt khi kích thước tập người dùng hoặc sản phẩm trở nên quá lớn. Trong các nghiên cứu về hệ gợi ý, KNN vẫn thường được sử dụng như một phương pháp baseline đáng tin cậy nhờ tính trực quan và dễ triển khai.

3.2.2 Matrix Factorization Models

3.2.2.1 Matrix Factorization

Thay vì biểu diễn trực tiếp người dùng và sản phẩm bằng các vector lịch sử tương tác như trong KNN — vốn gặp hạn chế về chi phí tính toán và độ lớn của không gian vector tăng theo số lượng tương tác — phương pháp **Matrix Factorization (MF)** hướng đến việc ánh xạ các thực thể này vào một **không gian đặc trưng tiềm ẩn** (*latent feature space*). Trong không gian này, mỗi người dùng và sản phẩm được biểu diễn bởi một vector đặc trưng ẩn (còn gọi là *embedding*), với mỗi chiều đại diện cho một đặc trưng tiềm ẩn không quan sát được trực tiếp.

Nếu xem hàm tính độ phù hợp là tích vô hướng giữa 2 embedding của user và item (*embedding vectors*)

$$f(u, i) = u \cdot i = \sum_{k=1}^n u_k i_k$$

thì bài toán sẽ trở thành bài toán phân tích ma trận tương tác $\mathbf{R} \in \mathbb{R}^{M \times N}$ thành tích của 2 ma trận

$$\mathbf{R} \approx \hat{\mathbf{R}} = \mathbf{U}\mathbf{V}^\top$$

trong đó $\mathbf{U} \in \mathbb{R}^{M \times n}$ là ma trận embedding của người dùng và $\mathbf{V} \in \mathbb{R}^{N \times n}$ là ma trận embedding của sản phẩm. Mỗi cột của ma trận \mathbf{U}, \mathbf{V} được gọi là một *thuộc tính* (feature), đại diện cho một thuộc tính ẩn nào đó trong dữ liệu được nhúng thông qua các số trong cột. Chú ý rằng số thuộc tính của \mathbf{U} và \mathbf{V} là bằng nhau. Khi sử dụng phép nhân vô hướng, các phần tử cùng thuộc tính của hai ma trận sẽ nhân với nhau; tất cả tích này sẽ được cộng lại, thu về một dự đoán. Vì thế, các thuộc tính có sự tương đồng cao khi nhân lại sẽ ra kết quả lớn, các thuộc tính ít tương đồng sẽ triệt tiêu nhau. Điều này khá giống cơ chế hoạt động của độ **cosine similarity** đã được đề cập ở trên.

Nhìn chung, mỗi phần tử của Interaction matrix chính là một tổ hợp tuyến tính các features của cặp user-item đó. Đây là lí do phương pháp này được xếp vào loại collaborative filtering.

Xét mô hình Matrix Factorization trong trường hợp đơn giản nhất, khi đầu vào là dạng feedback với giá trị số thực liên tục — ví dụ như *explicit feedback* đánh giá theo thang điểm (1–5 sao), hoặc *implicit feedback* được lượng hóa thành các giá trị thực (số lần xem, thời gian xem, tần suất tương tác, v.v.). Khi đó, mục tiêu tối ưu là xấp xỉ càng gần càng tốt các giá trị trong ma trận \mathbf{R} . Hàm mất mát phổ biến để đo lường sai lệch giữa dự đoán và giá trị là hàm **Mean Squared Error (MSE)**.

$$\mathcal{L} = \frac{1}{|\mathcal{D}|} \sum_{(u,i) \in \mathcal{D}} (r_{ui} - \hat{r}_{ui})^2$$

Nhìn kĩ hơn, bài toán tối ưu trên **không phải** là bài toán quy hoạch lồi. Ngoài ra, cũng không tồn tại giải thuật chính xác được xây dựng ở thời điểm này. Vì thế, các phương pháp giải đều đi đến việc **xấp xỉ** lời giải dần qua các phương pháp lặp (iterative methods). Trong đó, hai giải thuật được dùng phổ biến là:

- **Gradient Descent:** Đây là phương pháp cơ bản để tìm cực tiểu của các bài toán *quy hoạch phi tuyến không ràng buộc*. Ý tưởng là cập nhật dần tham số theo hướng ngược chiều *gradient* của hàm mất mát, với một tốc độ học đã cho. Phương pháp này tuy đơn giản nhưng còn tiềm ẩn nhiều vấn đề như lựa chọn tốc độ học phù hợp, tính *gradient* khó khăn trên tập huấn luyện lớn hay dễ bị kẹt tại các điểm yên ngựa (*saddle point*) hoặc cực tiểu cục bộ. Để khắc phục những nhược điểm này, nhiều biến thể của Gradient Descent đã được phát triển và sử dụng rộng rãi, mà nổi tiếng nhất là **Stochastic Gradient Descent (SGD)** hay **Adaptive Moment Estimation (Adam)**.
- **Alternating Least Squares (ALS):** dựa vào đặc điểm song tuyến tính (*bilinear*) của hàm tích vô hướng $f(u, i) = u \cdot v$, tức là khi cố định u hoặc i thì hàm f sẽ trở thành hàm tuyến tính, và bài toán sẽ trở thành bài toán lồi dạng hồi quy tuyến tính đơn giản. Từ ý tưởng này, thuật toán ALS được thiết kế để luân phiên cố định 1 trong 2 ma trận \mathbf{U} hoặc \mathbf{V} và giải bài toán tối ưu đối với ma trận còn lại bằng phương pháp bình phương tối thiểu (*least squares*). Phương pháp này được trình bày trong Thuật toán 2.

Một yêu cầu tự nhiên khi phải sử dụng các giải thuật lặp là cần một thuật toán sinh nghiệm khởi đầu tốt. Một nghiệm \mathbf{U}, \mathbf{V} như vậy sẽ làm tăng hiệu quả của giải thuật, đồng thời đưa thuật toán hướng đến các điểm cực tiểu tốt hơn. Phương pháp khởi tạo được đề xuất là sử dụng **Phân rã trị suy biến (Singular Value Decomposition - SVD)** - [https://en.wikipedia.org/wiki/Singular_value_decomposition]. Về lý thuyết, đây là cách phân rã một ma trận \mathbf{M} kích thước bất kì, không nhất thiết vuông, thành tích của ba ma trận $\mathbf{U}, \Sigma, \mathbf{V}$. Trong đó Σ là ma trận đường chéo, có các phần tử là các *trị suy biến* (singular values). Các trị này thực chất chính là *trị riêng* (eigenvalues) của $\mathbf{H}\mathbf{H}^T$. Vì ma trận này là ma trận đối xứng, nên các trị này là không âm. Vì thế, ta có thể viết $\Sigma = \mathbf{s}\mathbf{s}^T$ với \mathbf{s} là ma trận đường chéo, có các phần tử là *căn bậc hai* của các trị suy biến tương ứng. Vì thế, khi ta đặt $\mathbf{U}^* = \mathbf{U}\mathbf{s}$ và $\mathbf{V}^* = \mathbf{s}\mathbf{V}$ thì ta có

$$\mathbf{M} = \mathbf{U}^*\mathbf{V}^*$$

là một phân tích của một ma trận dưới dạng tích của hai ma trận con. Khi ta lược bỏ các trị suy biến, chỉ giữ lại k giá trị (với k rất nhỏ so với kích thước hai cạnh của \mathbf{M}); đồng thời lược bỏ các hàng - cột tương ứng của hai ma trận \mathbf{U} và \mathbf{V} , thì ta có một biểu diễn $\hat{\mathbf{M}}$ xấp xỉ cho \mathbf{M} , gọi là **Truncated SVD**. Điểm mấu chốt của phân rã này nằm ở việc $\hat{\mathbf{M}}$ chính là nghiệm của bài toán tối ưu:

$$\begin{aligned} \min_{\mathbf{B}} \quad & \|\mathbf{R} - \mathbf{B}\|_F^2 \\ \text{s.t.} \quad & \text{rank}(\mathbf{B}) \leq k \end{aligned} \tag{17}$$

Ngoài ra, sai số của nghiệm này chính bằng tổng bình phương của các trị suy biến đã lược bỏ đi ở phía trên. Vì thế nên đây còn được gọi là **Best Low-rank Approximation** của \mathbf{M} .

Algorithm 2 Alternating Least Squares (ALS)

Require: Ma trận tương tác $\mathbf{R} \in \mathbb{R}^{M \times N}$, số chiều không gian ẩn k , số vòng lặp T , hệ số regularization λ

Ensure: Ma trận đặc trưng người dùng $\mathbf{U} \in \mathbb{R}^{M \times d}$ và sản phẩm $\mathbf{V} \in \mathbb{R}^{N \times d}$

- 1: Khởi tạo ngẫu nhiên \mathbf{U}, \mathbf{V}
- 2: **for** $t = 1$ **to** T **do**
- 3: **for** mỗi user $u = 1$ đến M **do**
- 4: Cập nhật \mathbf{u}_u theo công thức:

$$\mathbf{u}_u \leftarrow \arg \min_{\mathbf{u}} \sum_{i \in \mathcal{I}_u} (R_{ui} - \mathbf{u}^\top \mathbf{v}_i)^2 + \lambda \|\mathbf{u}\|^2$$

- 5: **end for**
 - 6: **for** mỗi item $i = 1$ đến N **do**
 - 7: Cập nhật \mathbf{v}_i theo công thức:
 - 8: **end for**
 - 9: **end for**
 - 10: **return** \mathbf{U}, \mathbf{V}
-

Mã giả thuật toán ALS

Vậy nếu ta chọn k là số thuộc tính của ma trận \mathbf{R} , thì Truncated SVD cho ta một phân tích $\hat{\mathbf{R}} = \mathbf{U}\mathbf{V}^T$ xấp xỉ \mathbf{R} . Tuy vậy, hàm mục tiêu chúng ta không xét trên toàn bộ ma trận \mathbf{R} (vì vốn ma trận này chúng ta chưa biết) mà chỉ xét trên các giá trị đã nhìn thấy trước đó. Vì vậy, ta sẽ tạm thay thế các giá trị chưa biết bằng giá trị thích hợp (chẳng hạn: trung bình cộng của các giá trị đã biết). Từ đó, ta có một nghiệm ban đầu cho thuật toán GD hoặc ALS. Thực nghiệm chứng tỏ nghiệm này có độ mất mát nhỏ hơn rất nhiều so với các phương pháp sinh ngẫu nhiên hoặc theo các heuristic khác. Đối với các mô hình sử dụng các hàm mất mát khác, thì Truncated SVD vẫn là sự lựa chọn tốt, dù cần một vài điều chỉnh nhất định.

Các biến thể của MF nhằm tăng hiệu quả bao gồm:

- **Nonnegative matrix factorization (NMF):** Ở biến thể này, ta chỉ cần thêm ràng buộc $\mathbf{U} \geq 0$ và $\mathbf{V} \geq 0$. Điều này để hàm ý rằng các phần tử của \mathbf{U} và \mathbf{V} đặc trưng cho các số liệu nào đó của các đặc trưng ẩn của dữ liệu ban đầu. Mà vì đây là bài toán Recommendation System nên hầu hết các dữ liệu gốc sẽ không âm. Đây là một cách ràng buộc các tri thức bổ sung thực tế vào bài toán, với mong muốn rằng kết quả của mô hình bám sát với cấu trúc dữ liệu hơn, từ đó tăng tính hiệu quả.
- **Thêm các biến thiên kiến** Thêm các vector giá trị (được gọi là bias) b_u, b_i ứng với mỗi user/item và một bias tổng μ (global bias). Các bias này sẽ tóm tắt những xu hướng chung, trung bình của mỗi user/item hoặc của toàn bộ dữ liệu. Ma trận dự đoán $\hat{\mathbf{R}}$ trở thành:

$$\hat{\mathbf{R}} = \mathbf{U}\mathbf{V}^T + b_u + b_i + \mu$$

- **Thêm các side features** Thêm các embedding của dữ liệu gốc (side feature). Ta có thể cộng ma trận U trở thành $(\mathbf{U} + \mathbf{J}_u \mathbf{E}_u)$ với \mathbf{J}_u là ma trận chứa các feature là các cột dữ liệu bổ sung từ tập dữ liệu đề ra, và \mathbf{E}_u là một ma trận nhúng (embedding matrix) học được. Dự đoán trở thành:

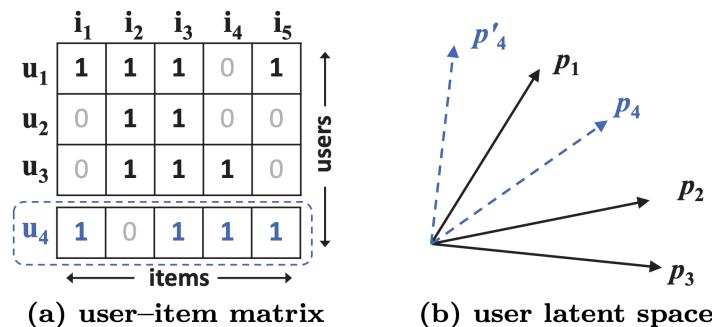
$$\hat{\mathbf{R}} = (\mathbf{U} + \mathbf{J}_u \mathbf{E}_u)(\mathbf{V} + \mathbf{J}_i \mathbf{E}_i)^T + b_u + b_i + \mu$$

Hạn chế của MF: Phương pháp MF hiện đang giả định các đặc trưng ẩn của embedding người dùng và sản phẩm là độc lập và tính ra giá trị tương tác bằng cách tổ hợp các chiều đó lại với cùng một trọng số. Điều này đồng nghĩa với việc mô hình MF chỉ có khả năng mô hình hóa các quan hệ tuyến tính giữa

các đặc trưng ẩn mà chưa xét được đến các quan hệ phi tuyến phức tạp.

Ví dụ trong Hình 3.2 chứng minh rằng hàm tích vô hướng có thể giới hạn chất lượng của MF, sử dụng *Cosine similarity* để đánh giá độ tương đồng giữa 2 embedding của người dùng.

Từ số liệu của ba dòng đầu tiên của hình (a) của ví dụ, ta sẽ tính được $s_{2,3}(0.66) > s_{1,2}(0.5) > s_{1,3}(0.4)$. Như vậy, quan hệ trên không gian của p_1, p_2, p_3 (p_u là vector ẩn của user u) có thể được vẽ như trong hình (b) của ví dụ. Xét user u_4 có $s_{4,1}(0.6) > s_{4,3}(0.4) > s_{4,2}(0.2)$, điều này có nghĩa là user u_4 tương đồng nhất với u_1 , tiếp theo là u_3 và cuối cùng là u_2 . Tuy nhiên, do mô hình MF đưa vector người dùng lên cùng một không gian ẩn, nên với MF, có 2 cách đặt vector của user u_4 thỏa mãn gần u_1 nhất như hình vẽ trên (2 vector p_4 và p'_4). Nhưng cả hai trường hợp này đều không thể thỏa mãn được tính chất thực tế là u_4 gần u_3 hơn u_2 . Từ đó, có thể thấy rằng MF không thể mô tả được độ đo Cosine hay chính là độ đo sự tương tự giữa các người dùng trong trường hợp này.



Hình 3.2: Ví dụ về hạn chế của MF

Qua ví dụ trên, ta có thể thấy được MF có một số giới hạn nhất định khi cố định sử dụng một hàm tích trong đơn giản để dự đoán cho một tương tác phức tạp giữa người dùng và sản phẩm. Một trong những cách khắc phục điều này đó là tăng số chiều k của embedding. Tuy nhiên, việc này sẽ làm mất tính tổng quát của mô hình, hay chính là vấn đề overfitting, đặc biệt là trong ma trận thừa.

3.2.2.2 Generalized Matrix Factorization (GMF)

Thoạt nhìn thì Matrix Factorization giống với 1 bài toán *unsupervised learning* - cụ thể là *dimensionality reduction* - hơn là *supervised learning*, khi mà đầu vào là 1 ma trận không nhãn và đầu ra là các vector embedding. Nhưng nhìn nhận dưới một góc độ tổng quát hơn, MF thực chất có thể được diễn giải như một bài toán *supervised learning* - cụ thể hơn là 1 biến thể của *linear regression*.

(1) Đầu vào (Input): Mỗi mẫu dữ liệu đầu vào có dạng $\{(u, i), r\}$ tương ứng với tương tác giữa 1 cặp chỉ số của user và item. Các chỉ số u và i được biểu diễn *one-hot encoding* thành 2 vector $e_u \in \mathbb{R}^N$ và $e_i \in \mathbb{R}^M$, trong đó N và M lần lượt là tổng số người dùng và sản phẩm. *One-hot encoding* là phương pháp biểu diễn một giá trị rời rạc dưới dạng vector nhị phân, trong đó chỉ có một phần tử bằng 1 tại vị trí tương ứng với giá trị đó, các phần tử còn lại bằng 0. Ví dụ, nếu có 5 người dùng, người dùng thứ 3 sẽ được biểu diễn bởi vector $e_3 = [0, 0, 1, 0, 0]^\top$.

Sau đó, các vector one-hot này được nhân với ma trận embedding — là các tham số được học trong quá trình huấn luyện — để thu được các vector đặc trưng (embedding vectors):

$$\mathbf{p}_u = \mathbf{P}^\top e_u, \quad \mathbf{q}_i = \mathbf{Q}^\top e_i$$

(2) Kết hợp (Combination): Quá trình kết hợp giữa một embedding user và một embedding item để trở thành 1 vector (hoặc tensor) duy nhất làm đầu vào cho các bước xử lý tiếp theo. Đối với MF kiểu truyền thống, toán tử kết hợp này là *element-wise product*, trong đó mỗi phần tử tương ứng của hai vector embedding được nhân với nhau để tạo thành vector đặc trưng tổng hợp.

$$z_1 = \phi(p_u, q_i) = p_u \odot q_i$$

Bên cạnh *element-wise product* còn nhiều cách khác để kết hợp 2 vector thành 1 vector hoặc tensor mới, điển hình như phép ghép nối (*concatenate*), phép tích ngoài (*outer product*) hay phép tích Kronecker.

(3) Tổng hợp (Aggregation): Tổng hợp tuyến tính các phần tử của vector z thu được từ bước kết hợp lại để cho ra 1 giá trị vô hướng duy nhất — chính là dự đoán \hat{y}_{ui} cho cặp người dùng u và sản phẩm i . Ở *linear regression*, bước tổng hợp này bao gồm trọng số và bias học được trong quá trình huấn luyện, nhưng ở MF cơ bản thì tất cả trọng số được cho bằng 1 và không có bias.

$$\hat{y}_{ui} = \mathbf{h}^\top (\mathbf{p}_u \odot \mathbf{q}_i), \quad \mathbf{h} = [1, 1, 1, \dots, 1] \text{ nếu là MF truyền thống}$$

(4) Kích hoạt (Activation): Bước này là không bắt buộc, nhưng nếu muốn mở rộng mô hình MF thành một dạng *Perceptron* để đưa vào các mạng neuron học sâu thì có thể gắn thêm hàm kích hoạt vào biểu thức ở bước tổng hợp.

$$\hat{y}_{ui} = a_{out}(\mathbf{h}^\top (p_u \odot q_i))$$

(5) Huấn luyện (Training): Với đầu vào đến từ bước 2, có thể đưa dữ liệu này vào để huấn luyện với nhiều loại hàm loss khác nhau tùy thuộc vào mục đích ban đầu là hồi quy, phân loại hay xếp hạng. Từ đó có thể thấy rằng mô hình Matrix Factorization kiểu cơ bản có nhiều điểm tương đồng với *linear regression*. Dựa trên quan sát này, một hướng mở rộng tự nhiên là bổ sung trọng số riêng và độ dời (bias) cho từng chiều trong vector đặc trưng ẩn (embedding). Biến thể này được gọi là *Generalized Matrix Factorization* (GMF), trong đó quá trình tổng hợp không còn là phép cộng đơn giản mà trở thành một tổ hợp tuyến tính có tham số:

$$\hat{R} = \mu + \mathbf{b}_u \mathbf{1}_m^\top + \mathbf{1}_n \mathbf{b}_i^\top + \left[\sum_{k=1}^K w_k \cdot \mathbf{u}_k \mathbf{v}_k^\top \right]$$

Quay lại với Matrix Factorization kiểu. Ở phần trước, chúng ta mới chỉ đề cập đến MF với dữ liệu dạng số thực - explicit feedback dạng rating 1-5 sao hoặc implicit feedback lượng hóa thành giá trị thực. Nhưng trên thực tế, việc thu thập được dữ liệu *explicit feedback* là tương đối khan hiếm. Thay vào đó, các hệ gợi ý chủ yếu phải sử dụng *implicit feedback*. Và cũng vì đặc thù nhiễu cao và không trực tiếp phản ánh mức độ yêu thích của *implicit feedback* mà các hệ gợi ý thường chuyển sang dạng nhị phân: đã tương tác (1) hoặc chưa tương tác (0). Với kiểu phản hồi này, ta cần một hàm mất mát phù hợp hơn MSE, vốn không phản ánh đúng bản chất của bài toán phân loại nhị phân. Nhờ vào framework tổng quát được đề xuất ở trên, có rất nhiều sự lựa chọn thay thế cho hàm mất mát có thể áp dụng:

(1) Binary Cross-Entropy Loss (BCE): Đây là một hàm mất mát được sử dụng rất phổ biến trong các bài toán phân loại nhị phân. Trong đó, giả định rằng 2 lớp dữ liệu $\{0, 1\}$ tuân theo phân phối Bernoulli với hàm dự đoán $\hat{y} = f(u \odot i)$, tức là

$$P(y | \hat{y}) = \hat{y}^y (1 - \hat{y})^{1-y}$$

khi đó bài toán học mô hình sẽ trở thành bài toán cực đại hóa khả năng dự đoán của hàm f (Maximum Likelihood Estimation) trên toàn bộ dữ liệu huấn luyện.

$$\max_{\theta} \prod_{i=1}^n P(y_i | x_i; \theta) = \max_{\theta} \prod_{i=1}^n \hat{y}_i^{y_i} (1 - \hat{y}_i)^{1-y_i}$$

Lấy logarithm phủ định của hàm này ta được công thức *Binary Cross-Entropy Loss*:

$$\mathcal{L}_{\text{BCE}} = - \sum_{i=1}^n [y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)]$$

Để thuận tiện cho việc tính gradient và đảm bảo đầu ra $\hat{y} \in (0, 1)$, hàm dự đoán $f(u, i)$ thường được chọn là hàm sigmoid của tích vô hướng giữa embedding của user và item $f(u, i) = \frac{1}{1+e^{-u \cdot i}}$:

$$\frac{\partial \mathcal{L}_{\text{BCE}}}{\partial z_i} = \frac{\partial \mathcal{L}_{\text{BCE}}}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial z_i} = \left(-\frac{y_i}{\hat{y}_i} + \frac{1 - y_i}{1 - \hat{y}_i} \right) \cdot \hat{y}_i (1 - \hat{y}_i) = \hat{y}_i - y_i$$

(2) Hinge Loss (Soft-margin SVM Loss): Nếu nhìn nhận bài toán dưới góc độ phân loại nhị phân thì một trong những mô hình học máy đơn giản mà mạnh mẽ về nền tảng lý thuyết toán học đằng sau

nó là Support Vector Machine. Trong đó, mục tiêu của bài toán là tìm một siêu phẳng (*hyperplane*) tối đa hóa khoảng cách đến các điểm dữ liệu gần nhất của mỗi lớp (*margin*). Do không phải lúc nào dữ liệu cũng là khả phân tách tuyến tính (*linear separable*), có thể cho phép tồn tại một mức độ sai lệch nhất định đối với những điểm không thể phân tách hoàn toàn. Cụ thể, hàm mục tiêu của bài toán là

$$\begin{aligned} \min_{\mathbf{w}, b, \xi} \quad & \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i \\ \text{s.t.} \quad & y_i(\mathbf{w}^\top \hat{y}_i + b) \geq 1 - \xi_i, \quad \forall i = 1, \dots, n \\ & \xi_i \geq 0, \quad \forall i = 1, \dots, n \end{aligned}$$

Trong đó:

- \mathbf{w} và b là tham số của siêu phẳng,
- ξ_i là biến trượt cho phép điểm dữ liệu i vi phạm margin,
- $C > 0$ là hệ số điều chỉnh giữa độ lớn margin và mức độ chấp nhận sai số.

Bài toán có thể viết lại dưới dạng:

$$\mathcal{L}_{\text{SVM}} = \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \max(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b))$$

Dây chính là tổng của *Hinge Loss* và *L2 Regularization*. Tuy *Hinge Loss* không liên tục tại điểm $y_i(\mathbf{w}^\top \mathbf{x}_i + b) = 1$, ta vẫn có thể áp dụng các thuật toán huấn luyện dựa trên gradient bằng cách sử dụng subgradient:

$$\frac{\partial \mathcal{L}_{\text{hinge}}}{\partial \mathbf{w}} = \begin{cases} \mathbf{0} & \text{nếu } y \cdot \mathbf{w}^\top \mathbf{x} \geq 1 \\ -y\mathbf{x} & \text{nếu } y \cdot \mathbf{w}^\top \mathbf{x} < 1 \end{cases}$$

Cuối cùng, sau khi thu được siêu phẳng phân loại $y_i(\mathbf{w}^\top \hat{y}_i + b) = 1$ ta có thể sử dụng khoảng cách từ mỗi điểm dữ liệu (u, i) đến siêu phẳng này làm thứ tự toàn phần $>_u$ cho tập sản phẩm.

(3) Bayesian Personalization Ranking Loss (BPR): Khác với hai hàm mất mát theo góc độ classification trên, BPR lại là một hàm mất mát theo góc độ ranking. Hàm mất mát này không sử dụng trực tiếp nhãn nhị phân của dữ liệu mà xây dựng từng cặp item đầu vào giữa 1 item có nhãn bằng 1 (positive item) và 1 item có nhãn bằng 0 (negative item) - do đó có thể xem là *self-supervised learning*.

Ý tưởng của hàm loss này là giả định rằng mọi item mà người dùng đã tương tác (positive) luôn vượt trội hơn các item mà người dùng chưa tương tác (negative). Khi đó, dù dữ liệu thu được là tưởng minh hay tiềm ẩn, ta luôn có thể thu được một tập thứ tự một phần (*partial order*) của các item. Và mục tiêu của BPR là cực đại hóa khả năng mở rộng tập thứ tự này thành tập thứ tự toàn phần.

$$\max_{\Theta} \prod_{u \in U} P(>_u | \Theta) = \max_{\Theta} \prod_{(u, i, j) \in U \times I \times I} P(i >_u j | \Theta)^{\delta_{(u, i, j) \in D_S}} \cdot (1 - P(i >_u j | \Theta))^{\delta_{(u, j, i) \in D_S}}$$

Với δ là hàm thuộc tính (indicator function):

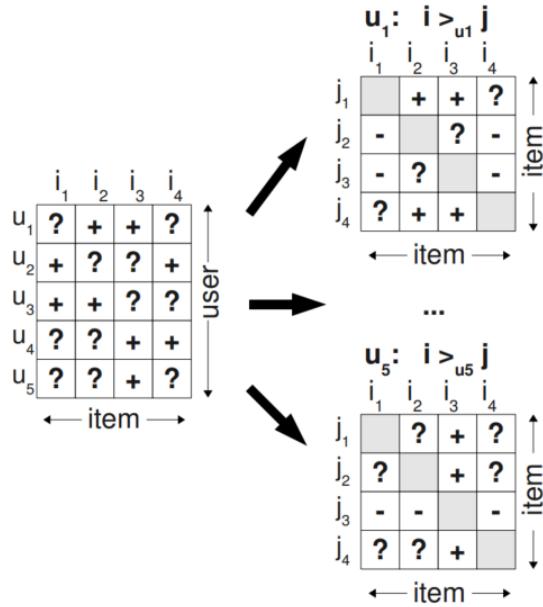
$$\delta(u, i, j) = \begin{cases} 1 & \text{nếu } u \text{ thích } i \text{ hơn } j \\ 0 & \text{còn lại} \end{cases}$$

Vì tính *khả so sánh toàn phần* và *phi đối xứng*, ta có thể đơn giản công thức về:

$$\max_{\Theta} \prod_{u \in U} P(>_u | \Theta) = \max_{\Theta} \prod_{(w, i, j) \in \mathcal{D}_S} p(i >_w j | \Theta) \tag{3.1}$$

Hiện tại quan hệ thứ tự của chúng ta mới chỉ là một phần nên lúc này một quan hệ $i >_w j$ bất kì:

$$p(i >_w j | \Theta) = \begin{cases} 1 & \text{nếu } i >_w j \\ 0 & \text{còn lại} \end{cases}$$



Hình 3.3: Bayesian Personalized Ranking

Ta cần tổng quát hóa quan hệ thứ tự này thành quan hệ thứ tự toàn phần. Chọn $p(i >_w j \mid \Theta) = \sigma(\hat{y}_{u,i} - \hat{y}_{u,j})$, thay vào biểu thức (3.1) và lấy logarith phủ định ta thu được công thức *BPR Loss*:

$$\begin{aligned}\mathcal{L}_{\text{BPR}} &= -\ln p(\Theta \mid >_u) \\ &= -\sum_{(u,i,j) \in \mathcal{D}_S} \ln \sigma(\hat{y}_{u,i} - \hat{y}_{u,j}) - \lambda_\Theta \|\Theta\|^2 \\ &= -\sum_{(u,i,j) \in \mathcal{D}_S} \ln \sigma(\hat{y}_{u,i,j}) - \lambda_\Theta \|\Theta\|^2\end{aligned}$$

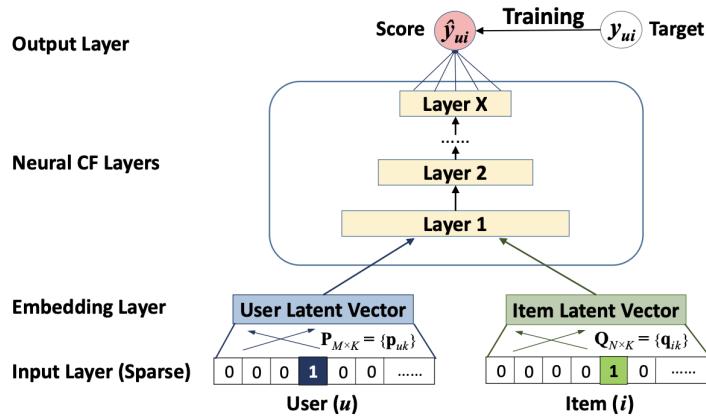
Vì số lượng cặp positive - negative là rất lớn, thông thường khi tính *BPR Loss* chỉ sample một số lượng nhỏ negative item cho mỗi positive item, điển hình là tỉ lệ 1 : 1, 1 : 3 hay 1 : 10. Ngoài ra, hàm mất mát này vốn được thiết kế dành riêng cho *implicit feedback*, tuy nhiên cũng có thể tùy chỉnh cách định nghĩa negative item và chiến lược negative sampling cho phù hợp.

3.2.2.3 Multi-Layer Perceptron

Như đã đề cập ở những phần trước, MF tồn tại nhiều hạn chế khi chỉ học được các quan hệ tuyến tính giữa người dùng và sản phẩm mà chưa xét đến các quan hệ phi tuyến phức tạp. Nhờ vào framework tổng quát đã được đề xuất với một góc nhìn khái quát hơn, ta có thể phát triển MF thành một mô hình Multi-Layer Perceptron (MLP) có khả năng khai thác các đặc trưng ẩn theo hướng này và thực hiện nó bằng mạng neuron.

Kiến trúc của mô hình được chia thành nhiều tầng, đầu ra tầng trước là đầu vào của tầng sau.

- **Tầng input:** Chúng ta sẽ id của user và item biểu diễn user và item dưới dạng one-hot vector.
- **Tầng embedding:** Đây thực chất là một tầng kết nối đầy đủ với đầu vào là vector one-hot của tầng input. Qua tầng này, user và item sẽ được biểu diễn dưới dạng 1 vector dày đặc với số chiều thấp hơn nhiều so với one-hot vector. Ta có thể coi các vector này chính biểu diễn của các thuộc tính ẩn của user và item.
- **Các tầng neuron:** Đây là một mạng nơ-ron kết nối đầy đủ nhiều tầng với đầu vào là một vector - kết quả của phép nối vector nhúng của user với item và đầu ra là một vector.



Hình 3.4: Kiến trúc mô hình của phương pháp MLP

- **Tầng output:** Với đầu vào vector output của tầng trước, ta sẽ sử dụng một tầng kết nối đầy đủ, thu được đầu ra là tương tác giữa user và item.

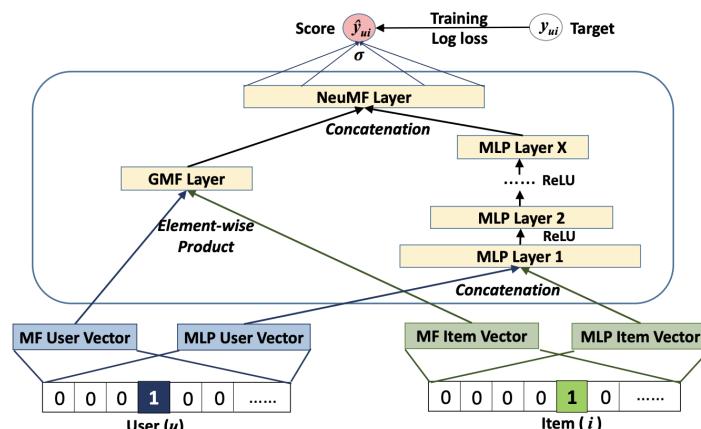
Ký hiệu p_u và q_i lần lượt là các vector đầu ra của tầng embedding với đầu vào là user u và item i . Từ đó ta định nghĩa model của phương pháp MLP như sau:

$$\begin{aligned} z_1 &= \phi(p_u, q_i) = \begin{bmatrix} p_u \\ q_i \end{bmatrix} \\ \phi_2(z_1) &= a_2(W_2^T z_1 + b_2) \\ &\dots\dots \\ \phi_L(z_{L-1}) &= a_L(W_L^T z_{L-1} + b_L) \\ \hat{y}_{ui} &= \sigma(h^T \phi_L(z_{L-1})) \end{aligned}$$

trong đó, W_i , b_i và a_i lần lượt là ma trận weight, vector bias và activation function của lớp neuron thứ i ; σ là hàm sigmoid.

Tương tác người dùng chỉ có thể nhận một trong hai giá trị 0 hoặc 1. Ta nhận được đầu ra của model nằm trong khoảng (0, 1) nhờ hàm activate của lớp cuối cùng. Giá trị của đầu ra thể hiện độ liên quan giữa người dùng và sản phẩm. Điểm càng cao thì độ liên quan càng cao. Để học tham số của mô hình, ta sử dụng hàm BCE loss.

3.2.2.4 Neural Matrix Factorization (NeuMF) - Sự kết hợp của MLP và GMF



Hình 3.5: Kiến trúc mô hình của phương pháp NeuMF

Ta có thể thấy rằng phương pháp GMF cho ta một mô hình tuyến tính, còn MLP cho ta một mô hình phi tuyến. Vậy tại sao ta không thử kết hợp mô hình của hai phương pháp này lại với nhau với hy vọng nó sẽ tận dụng được ưu điểm của các mô hình thành phần? Từ đó, phương pháp NeuMF ra đời như là sự kết hợp của tuyến tính trong GMF và phi tuyến trong MLP.

Ta sẽ chia kiến trúc của mô hình của phương pháp NeuMF thành 2 phần: GMF và MLP.

- **Tầng input và tầng embedding:** Tương tự với tầng input và tầng embedding của MLP. Tuy nhiên, với NeuMF ta sẽ sử dụng các vector embedding riêng cho mỗi phần GMF và MLP.
- **Với GMF,** ta có đầu vào là vector embedding của user và item, sau đó thực hiện phép element-wise product và thu được đầu ra là một vector với kích thước bằng với vector embedding của user và item.
- **Với MLP,** ta thực hiện tương tự theo kiến trúc mạng ở phần [**4.2**](#) với đầu ra thu được là một vector.
- **Tầng output:** Ta thực hiện phép nối vector đầu ra của 2 phần lại với nhau thu được một vector, sau đó cho nó đi qua một lớp kết nối đầy đủ thu được đầu ra là dự đoán tương tác của user với item.

Ký hiệu p_u^G, q_i^G và p_u^M, q_i^M lần lượt là các vector đầu ra của tầng embedding với đầu vào là user u và item i ứng với mỗi phần GMF và MLP. Từ đó ta định nghĩa model của phương pháp NeuMF như sau:

$$\begin{aligned}\phi^{GMF} &= p_u^G \odot q_i^G \\ \phi^{MLP} &= a_L(W_L^T(a_{L-1}(\dots a_2(W_2^T \begin{bmatrix} p_u^M \\ q_i^M \end{bmatrix} + b_2) \dots)) + b_L) \\ \hat{y}_{ui} &= \sigma(h^T \begin{bmatrix} \phi^{GMF} \\ \phi^{MLP} \end{bmatrix})\end{aligned}$$

trong đó, W_i , b_i và a_i lần lượt là ma trận weight, vector bias và activation function của lớp neuron thứ i ; σ là hàm sigmoid.

3.2.2.5 Convolutional Neural Collaborative Filtering

Các mô hình MF, MLP, NeuMF được trình bày trước đó chủ yếu thực hiện bước *Combination* trong framework tổng quát bằng 2 toán tử chính là phép ghép nối (*concatenate*) và phép nhân từng phần tử (*element-wise product*). Các lựa chọn này dẫn đến một hạn chế cố hữu là không thể học được tương quan chéo giữa các đặc trưng ẩn của embedding. Với *element-wise product*, ngay từ ban đầu chúng ta đã phải giả định rằng các feature của embedding là hoàn toàn độc lập, trong khi với *concatenate* để học được embedding có tương quan chéo tốt cần một kiến trúc mạng rất sâu - dẫn đến chi phí tính toán lớn và tiềm ẩn nguy cơ về *vanishing gradient*.

Để khắc phục điều này, một lựa chọn về toán tử kết hợp thay thế được đưa ra là phép tích trong (*outer product*). Toán tử này sẽ kết hợp embedding của user và item thành 1 ma trận, trong đó mỗi phần tử không nằm trên đường chéo sẽ thể hiện 1 cặp feature ẩn.

$$\begin{aligned}\mathbf{p}_u &= \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}, \quad \mathbf{q}_i = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} \\ \mathbf{Z}_{ui} = \mathbf{p}_u \mathbf{q}_i^\top &= \begin{bmatrix} p_1 q_1 & p_1 q_2 & p_1 q_3 \\ p_2 q_1 & p_2 q_2 & p_2 q_3 \\ p_3 q_1 & p_3 q_2 & p_3 q_3 \end{bmatrix}\end{aligned}$$

Đối với kết quả của phép kết hợp này, có 2 hướng xử lí bước tiếp theo

- Flatten ma trận này - tương đương với phép nhân Kronecker - và đưa vào một mạng MLP. Cách này có nhược điểm là cần học quá nhiều tham số. Lấy ví dụ với đầu vào embedding kích thước $k = 64$, vector tổng hợp sẽ có kích thước $64 \times 64 = 4096$. Nếu dựa theo kiến trúc phổ biến của MLP trong bài toán lọc cộng tác thì lớp ẩn đầu tiên sẽ có số neuron bằng một nửa đầu vào \Rightarrow Tổng cộng

lớp đầu tiên có đến $4096 \times 2048 = 8388608$ tham số - một con số khổng lồ kể cả xét về mặt tính toán hay lưu trữ, Việc có nhiều tham số thế này cũng khiến cho bước tinh chỉnh siêu tham số gấp nhiều khó khăn hơn.

- Một cách tiếp cận mới mẻ hơn, chuyên xử lý đầu vào có dạng ma trận là *Convolution Neural Network*

Convolution Neural Network (CNN) là một kiến trúc mạng neuron được thiết kế ban đầu để xử lý dữ liệu có cấu trúc ma trận, tiêu biểu là ảnh. Mục tiêu của CNN là khai thác các tính chất cục bộ không gian như cạnh, góc, hoặc kết cấu trong hình ảnh, từ đó giúp mô hình hiểu được nội dung hình ảnh ở mức độ trừu tượng hơn.

Khác với mạng neuron truyền thống, trong đó mỗi tầng bao gồm các *neuron* kết nối đầy đủ, thì ở CNN, mỗi tầng bao gồm một tập hợp các **ma trận kernel** thực hiện **phép tích chập** trên đầu vào. Kết quả cho ra là một ma trận mới gọi là **feature map**.

Giả sử đầu vào là một ma trận $\mathbf{X} \in \mathbb{R}^{H \times W}$, và kernel là một ma trận $\mathbf{K} \in \mathbb{R}^{k \times k}$. Phép chập tại vị trí (i, j) được tính như sau:

$$(\mathbf{X} * \mathbf{K})_{i,j} = \sum_{m=1}^k \sum_{n=1}^k \mathbf{X}_{i+m-1, j+n-1} \cdot \mathbf{K}_{m,n}$$

Sau khi áp dụng phép chập, ta sử dụng một hàm kích hoạt phi tuyến, chẳng hạn như ReLU:

$$\mathbf{Y}_{i,j} = \sigma((\mathbf{X} * \mathbf{K})_{i,j} + b), \quad \sigma(x) = \max(0, x)$$

Trong đó b là hệ số bias đi kèm với mỗi kernel.

Kích thước đầu ra: Kích thước đầu ra sau phép tích chập phụ thuộc vào:

- Kích thước kernel: $k \times k$
- Stride s : bước nhảy khi kernel trượt
- Padding p : số lượng pixel được thêm vào biên của đầu vào

Kích thước đầu ra $H_{\text{out}} \times W_{\text{out}}$ được tính theo công thức:

$$H_{\text{out}} = \left\lfloor \frac{H + 2p - k}{s} \right\rfloor + 1, \quad W_{\text{out}} = \left\lfloor \frac{W + 2p - k}{s} \right\rfloor + 1$$

Tùy vào việc chọn stride và padding, ma trận đầu ra có thể giữ nguyên kích thước hoặc bị thu nhỏ.

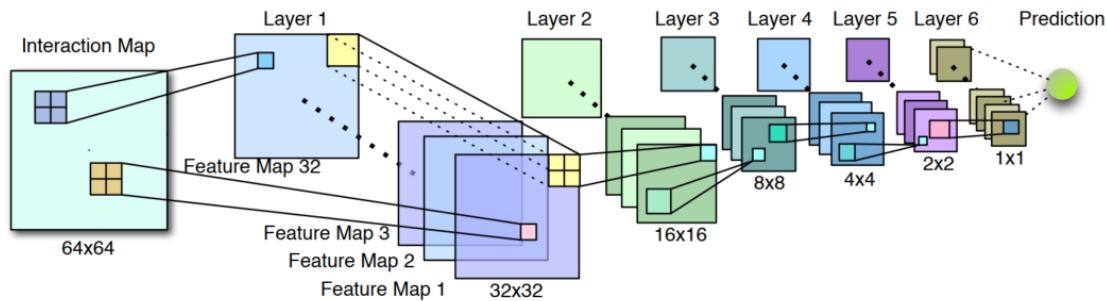
Kết hợp Outer product và CNN - ConvNCF: Dựa trên ý tưởng biểu diễn tương tác giữa người dùng và sản phẩm bằng ma trận outer product, ConvNCF tận dụng mạng CNN để trích xuất các đặc trưng phi tuyến từ bản đồ tương tác này. Mỗi lớp CNN học các mẫu cục bộ và dần mở rộng phạm vi tương quan giữa các chiều embedding thông qua việc xếp chồng nhiều tầng.

Kiến trúc cơ bản của ConvNCF gồm nhiều lớp tích chập liên tiếp với số lượng kernel cố định, trong khi kích thước feature map giảm một nửa sau mỗi lớp bằng cách sử dụng stride = 2 hoặc kết hợp với các lớp pooling. Với embedding đầu vào có kích thước $k = 2^n$, ma trận tương tác ban đầu có kích thước $2^n \times 2^n$, và sau n lớp tích chập, đầu ra còn lại là một feature map 1×1 , được đưa qua một lớp fully-connected để dự đoán xác suất tương tác. Thiết kế này giúp mô hình mở rộng khả năng học các tương quan chéo bậc cao một cách hiệu quả mà không cần đến số lượng tham số quá lớn như MLP.

Khả năng học tương quan chéo bậc cao của ConvNCF: Trong ma trận tương tác \mathbf{Z} , mỗi phần tử $z_{i,j}$ biểu diễn tương quan bậc hai giữa chiều thứ i của embedding người dùng và chiều thứ j của embedding sản phẩm:

$$z_{i,j} = \mathbf{p}_i \cdot \mathbf{q}_j$$

Tiếp theo, mỗi lớp ẩn l trong mạng CNN sẽ học các *mối tương quan cục bộ* trong vùng 2×2 từ tầng trước đó $l - 1$.



Hình 3.6: Kiến trúc cơ bản của ConvNCF với đầu vào là ma trận Outer product 64×64 , 6 hidden layers, mỗi lớp 32 feature map, stride = 2, kernel size = 2

Ví dụ, một phần tử $z_{x,y,c}^1$ trong lớp ẩn đầu tiên sẽ phụ thuộc vào 4 giá trị trong tầng trước (tức ma trận tương tác ban đầu \mathbf{Z}):

$$[z_{2x,2y}^0, z_{2x,2y+1}^0, z_{2x+1,2y}^0, z_{2x+1,2y+1}^0]$$

Điều này có nghĩa là $z_{x,y}^1$ đã học được **tương quan bậc 4** giữa các chiều embedding:

$$\{2x, 2x + 1, 2y, 2y + 1\}$$

Tổng quát hơn, một phần tử tại lớp ẩn thứ l sẽ học các tương quan trong một vùng $2^l \times 2^l$ của ma trận tương tác ban đầu \mathbf{Z} . Khi đó, mỗi phần tử sẽ nắm bắt tương quan bậc 2^{l+1} giữa các chiều embedding. Như vậy, một phần tử trong lớp cuối cùng có thể nắm bắt được tương quan toàn cục giữa tất cả các chiều embedding.

3.2.3 Factorization Machine Models

3.2.3.1 FM

Ta quay lại về mô hình *Matrix Factorization* (MF) một chút. Ở MF, bản chất là ta đi nhúng biểu diễn của không gian dữ liệu user và không gian dữ liệu item vào hai ma trận \mathbf{U}, \mathbf{V} phân biệt. Nhưng liệu ta có cần hai không gian không. Thế nào nếu ta nhúng toàn bộ không gian user/item, cộng với các thuộc tính lề (side features) vào cùng một ma trận và đi tìm biểu diễn chung của chúng. Đây là ý tưởng của mô hình **Factorization Machine - FM**

Mô hình này sẽ nhúng tất cả dữ liệu về một không gian số chiều \mathbf{K} biểu diễn bởi ma trận \mathbf{V} . Mỗi cột của \mathbf{V} là một vector embedding định danh với một tính chất nào đó của dữ liệu. Ta có một cột \mathbf{v}_{u_1} đại diện cho người dùng 1, một cột $\mathbf{v}_{\text{female}}$ đại diện cho các người dùng nữ, một cột \mathbf{v}_{2017} đại diện cho dữ liệu của năm 2017. Số cột của \mathbf{V} là số thuộc tính được nhúng vào. Chiều dài của mỗi vector quyết định lượng thông tin mà toàn bộ ma trận có thể lưu trữ. Khi tăng chiều dài này, mô hình có khả năng nắm bắt nhiều thông tin, cấu trúc của dữ liệu hơn, đồng thời làm tăng độ phức tạp của mô hình.

Với mỗi bản ghi tương tác user \mathbf{u} - item \mathbf{i} , ta định nghĩa một vector one-hot x_{ui} , trong đó một phần tử bằng 1 chỉ khi dữ liệu của cột tương ứng trong \mathbf{V} được chọn vào trong hàng tử tương tác. Ta sẽ thêm một vector bias w cho mỗi cột của \mathbf{V} và một bias vô hướng toàn cục μ . Dự đoán của chúng ta cho mỗi giá trị tương tác là:

$$\hat{r}_{ui} = \mu + \mathbf{w}^T \mathbf{x} + \sum_i \sum_{j>i} \mathbf{v}_i^T \mathbf{v}_j x_i x_j$$

Ta có thể coi \mathbf{x} như là một **mã gen**, quy định tất cả các thuộc tính của dữ liệu được sử dụng để rút ra dự đoán \hat{r}_{ui} . Hạng tử $\mathbf{w}^T \mathbf{x}$ là tổng tất cả bias ứng với dữ liệu được chọn. Hạng tử $\sum_i \sum_{j>i} \mathbf{v}_i^T \mathbf{v}_j x_i x_j$ chính là tổng tất cả tích vô hướng của từng cặp cột dữ liệu được chọn. Khi nhìn ở khía cạnh **Feature Engineering**, điều này như thể ta đang *tạo thủ công thuộc tính tương tác*, là tích giữa mỗi cặp feature được chọn, xong rồi kết hợp hết kết quả lại. Nếu như \mathbf{V} chỉ chứa thông tin định danh user/item, đồng

thì vector \mathbf{x} chỉ có hai phần tử bằng 1 ứng với cặp user u - item i ở trên, thì công thức trên sẽ thu gọn thành:

$$\hat{\mathbf{r}}_{ui} = \mu + w_u + w_i + \mathbf{v}_u * \mathbf{v}_i$$

Đây chính là công thức dự đoán của mô hình Matrix Factorization. Điều này chứng tỏ rằng FM chính là mở rộng của MF đến **bậc hai**, khi ta xét đến tương tác theo cặp của các thuộc tính dữ liệu.

Một lưu ý nhỏ rằng tổng $\sum_i \sum_{j>i} \mathbf{v}_i^T \mathbf{v}_j x_i x_j$ có thể được viết lại một cách gọn gàng như sau:

$$\begin{aligned} 2 \sum_{i=1}^d \sum_{j=i+1}^d \mathbf{v}_i^T \mathbf{v}_j x_i x_j &= \left(\sum_{i=1}^d \sum_{j=1}^d \mathbf{v}_i^T \mathbf{v}_j x_i x_j - \sum_{i=1}^d \mathbf{v}_i^T \mathbf{v}_i x_i x_i \right) \\ &= \frac{1}{2} \left(\sum_{i=1}^d \sum_{j=1}^d \sum_{l=1}^k v_{i,l} v_{j,l} x_i x_j - \sum_{i=1}^d \sum_{l=1}^k v_{i,l} v_{i,l} x_i x_i \right) \\ &= \frac{1}{2} \sum_{l=1}^k \left(\sum_{i=1}^d v_{i,l} x_i \sum_{j=1}^d v_{j,l} x_j - \sum_{i=1}^d (v_{i,l} x_i)^2 \right) \\ &= \sum_{l=1}^k \left(\left(\sum_{i=1}^d v_{i,l} x_i \right)^2 - \sum_{i=1}^d (v_{i,l} x_i)^2 \right) \\ &= \sum_{l=1}^k \left(\sum_{i=1}^d v_{i,l} x_i \right)^2 - \sum_{l=1}^k \sum_{i=1}^d (v_{i,l} x_i)^2 \end{aligned}$$

Rút gọn lại, ta có: $\sum_i \sum_{j>i} \mathbf{v}_i^T \mathbf{v}_j x_i x_j = \frac{1}{2} \left(\|\mathbf{V}^T \mathbf{x}\|^2 - \sum_{i=1}^n x_i^2 \|\mathbf{v}_i\|^2 \right)$. Công thức này giúp ta tính toán đạo hàm hiệu quả trong bước huấn luyện mô hình

3.2.3.2 Neural Factorization Machine (NFM)

Người ta chỉ được ra rằng mô hình của phương pháp FM là một mô hình tuyến tính đối với tham số. Vì vậy FM có thể bị hạn chế về khả năng biểu diễn để mô hình hóa các dữ liệu thực tế vốn có cấu trúc phức tạp. Từ đó, NFM ra đời với hy vọng nâng cao khả năng biểu diễn của FM bằng việc sử dụng mạng neuron. Cụ thể, trong NFM, thay vì nối các vector embedding rồi cho qua mạng neuron như ở một số phương pháp khác, ta sẽ sử dụng mạng neuron để mô hình hóa phần tương tác đặc trưng bậc hai, từ đó mô hình tổng thể có thể học được các tương tác bậc cao hơn.

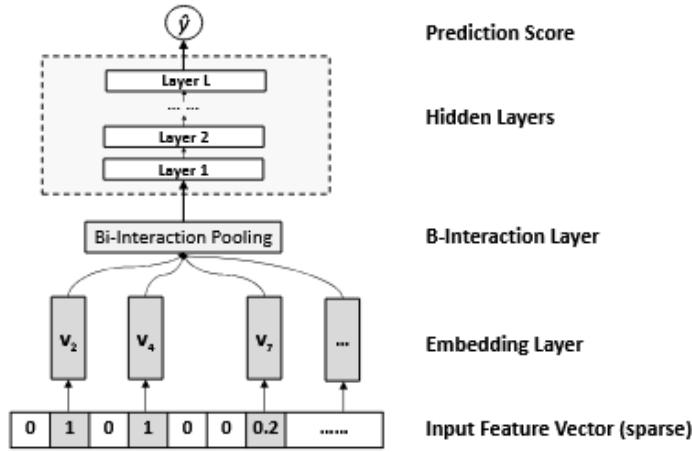
Giống với FM, phương pháp NFM cho phép chúng ta làm việc với bất kỳ vector đầu vào giá trị thực nào. Cho $x \in \mathbb{R}^n$ là đầu vào, với $x_i = 0$ thể hiện đầu vào không có đặc trưng i , mô hình của phương pháp NFM được định nghĩa như sau:

$$\hat{y}_{NFM}(x) = w_o + \sum_{i=1}^n w_i x_i + f(x)$$

Ta có thể thấy rằng hai số hạng đầu giống với FM, còn số hạng thứ ba sẽ được biểu diễn theo kiến trúc như hình 3.7.

- **Tầng embedding:** Mỗi đặc trưng đầu vào sẽ được ánh xạ tới một vector ma trận bởi một lớp kết nối đầy đủ.
- **Tầng B-Interaction:** Với đầu vào là các vector embedding của tầng trước, ta sẽ thu được một vector biểu diễn tương tác đặc trưng bậc hai.

$$f_{BI}(V_x) = \sum_{i=1}^n \sum_{j=i+1}^n x_i v_i \odot x_j v_j$$



Hình 3.7: Kiến trúc mô hình của phương pháp NeuMF

trong đó, V_x là tập các vector embedding của đầu vào, v_i là vector embedding tương ứng với đặc trưng x_i và \odot là phép nhân từng phần tử. Ta có thể đơn giản hóa phương trình trên về dạng dưới đây.

$$f_{BI}(V_x) = \frac{1}{2} \left[\left(\sum_{i=1}^n x_i v_i \right)^2 - \sum_{i=1}^n (x_i v_i)^2 \right]$$

trong đó, v^2 là $v \odot v$.

- **Các tầng neuron:** Với đầu vào là vector biểu diễn tương tác đặc trưng bậc hai, ta sẽ cho vector này qua một số tầng neuron, từ đó thu được một vector biểu diễn được các tương tác đặc trưng bậc cao hơn.

$$\begin{aligned} z_1 &= \sigma_1(W_1 f_{BI}(V_x) + b_1) \\ z_2 &= \sigma_2(W_2 z_1 + b_2) \\ &\dots \\ z_L &= \sigma_L(W_L z_{L-1} + b_L) \end{aligned}$$

trong đó, L là số lượng các tầng neuron, W_l , b_l và σ_l lần lượt là ma trận weight, vector bias và hàm activation của tầng neuron thứ l .

- **Tầng đầu ra:** Là một lớp kết nối đầy đủ với đầu vào là vector biểu diễn tương tác đặc trưng bậc cao và đầu ra là một giá trị.

$$f(x) = h^T z_L$$

trong đó, h là vector weight của lớp kết nối đầy đủ.

Tóm lại, ta có mô hình của phương pháp NFM.

$$\hat{y}_{NFM}(x) = w_o + \sum_{i=1}^n w_i x_i + h^T \sigma_L(W_L(\dots \sigma_1(W_1 f_{BI}(V_x) + b_1) \dots) + b_L)$$

Ta có thể thấy rằng FM là một trường hợp đặc biệt của NFM khi ta không sử dụng các tầng neuron và vector weight h không đổi bằng $(1, \dots, 1)$.

3.2.4 Graph Neural Network Models

3.2.4.1 Tổng quan về GNN (Overview)

Mạng nơ-ron đồ thị (GNN - Graph Neural Network), là một khung tổng quát để định nghĩa mô hình **Mạng nơ-ron sâu (deep neural network)** cho cấu trúc **dữ liệu đồ thị (graph data)**. Ý tưởng

cốt lõi của GNN là **tạo ra các biểu diễn ẩn (embedding)** của các nút sao cho chúng thực sự phụ thuộc vào **cấu trúc đồ thị**, cũng như bất kỳ **thông tin đặc trưng (feature information)** nào mà chúng ta có.

Mạng nơ-ron đồ thị (GNN) kết hợp thành tựu của hai lĩnh vực:

- **Mạng nơ-ron tích chập (CNN - Convolutional Neural Network)**, CNN vốn rất hiệu quả trong việc trích xuất đặc trưng (feature extraction) cục bộ trên dữ liệu dạng Euclid như ảnh, văn bản, tuy nhiên với dữ liệu dạng non-Euclid như đồ thị (graph), CNN cần phải được thiết kế một cách tổng quát hơn để xử lý được dữ liệu có kích thước không cố định (non-fixed size). Đó là lý do mà GNN ra đời để khắc phục vấn đề này:
 - **GNN lấy cảm hứng từ phép tích chập (convolution)**: Nếu như CNN dùng một bộ lọc (filter) nhỏ “quét” qua từng vùng cục bộ (local receptive field) và chia sẻ cùng một tập trọng số (weight sharing) thì GNN mở rộng khái niệm này sang dữ liệu dạng đồ thị: Mỗi nút thu thập thông tin từ các nút hàng xóm trong một “vùng lân cận” không định hình, rồi áp dụng một phép biến đổi tuyến tính chung cho tất cả các cạnh. Kết quả là chúng ta có được cơ chế **message passing (lan truyền thông điệp)** với đầy đủ các đặc tính cục bộ, chia sẻ trọng số và bắt biến khi đổi trật tự hàng xóm (permutation invariance).
 - **Xây dựng biểu diễn đa tầng (hierarchical representations)**: Nếu như kiến trúc CNN bao gồm nhiều tầng như: **Convolution Layer, Pooling Layer, Fully-Connected Layer,...** được xếp chồng lên nhau để học đặc trưng tiềm ẩn (latent feature) thì GNN cũng xếp chồng nhiều lớp lan truyền và tổng hợp thông điệp (**Message Propagation & Aggregation Layer**) để mô hình hiểu sâu hơn về biểu diễn của các nút, cũng như cấu trúc toàn cục của đồ thị.
- **Học biểu diễn đồ thị (GRL - Graph Representation Learning)**, mục tiêu GRL là tổng quát hóa vector biểu diễn ẩn có số chiều thấp (low-dimensional embedding vector) cho nút, cạnh,...
 - **Phương pháp Embedding không giám sát (Unsupervised Embedding)**: Trước khi GNN ra đời, các thuật toán như **DeepWalk, node2vec** dùng **random walk** kết hợp **Skip-Gram** để học Embedding cho mỗi nút, bảo toàn thứ tự và khoảng cách lân cận trong đồ thị. GNN kế thừa mục tiêu này nhưng cho phép tối ưu trực tiếp biểu diễn (end-to-end) theo downstream task (ví dụ: dự đoán liên kết user-item).
 - **Thiết kế hàm mất mát và sampling thông minh**: GRL đã phát triển các chiến lược lấy mẫu (negative sampling, importance sampling) và loss functions (ví dụ: KL divergence) để học Embedding hiệu quả trên đồ thị lớn. GNN tận dụng những kỹ thuật này để huấn luyện message-passing layer một cách ổn định, giảm nhiễu từ các cạnh không liên quan nhiều.

Ba cấp độ tác vụ chính trong GNN: Trong học sâu trên đồ thị, các bài toán thường được phân loại theo cấp độ của đối tượng dự đoán, gồm ba nhóm chính:

- **Node-Level Tasks**: Dự đoán nhãn hoặc đặc trưng cho từng nút trong đồ thị. Ví dụ phổ biến là **node classification**, như phân loại người dùng theo hành vi, dự đoán độ tuổi, vai trò trong mạng xã hội,...
- **Edge-Level Tasks**: Dự đoán đặc trưng hoặc xác suất tồn tại của các cạnh giữa hai nút. Một ví dụ tiêu biểu là **link prediction** – được sử dụng rộng rãi trong hệ gợi ý, với mục tiêu dự đoán xem liệu user u có khả năng tương tác với item i trong tương lai hay không. Ngoài ra còn có các bài toán như **edge classification** (dự đoán loại quan hệ giữa hai node).
- **Graph-Level Tasks**: Dự đoán nhãn hoặc thuộc tính của toàn bộ đồ thị. Ví dụ: Phân loại cấu trúc phân tử trong hóa học (toxic / non-toxic), chẩn đoán bệnh từ đồ thị tín hiệu não,...

3.2.4.2 Quy chuẩn thiết kế của GNN

Khi thiết kế các mạng nơ-ron sâu cho dữ liệu đồ thị, một trong những thách thức cốt lõi là xử lý bản chất phi Euclidean và không có thứ tự tự nhiên của dữ liệu đồ thị, khác biệt hoàn toàn so với dữ liệu lưới truyền thống như hình ảnh hay chuỗi. Theo thuật ngữ toán học (mathematical term), chúng ta cần phải thiết kế một hàm f nhận đầu vào là **ma trận kè A (biểu diễn cấu trúc đồ thị)** và **ma trận đặc trưng nút X (chứa thông tin của các nút)**, sao cho hàm này thỏa mãn hai tính chất cơ bản:

- **Tính bất biến hoán vị (Permutation Invariance):** Đầu ra của hàm hoặc mô hình không thay đổi khi thứ tự các nút đầu vào bị hoán vị. Điều này quan trọng cho *các tác vụ cấp đồ thị (graph-level tasks)*.

$$f(A, X) = f(PAP^T, PX)$$

với P là **ma trận hoán vị (permutation matrix)**. Tính bất biến hoán vị (permutation invariance) này là điều kiện cần thiết cho các **phép toán tổng hợp (aggregation functions)** thông tin lảng giềng trong GNN. Ví dụ, tổng của các biểu diễn lảng giềng không thay đổi bất kể thứ tự mà các lảng giềng đó được xem xét.

- **Tính đồng biến hoán vị (Permutation Equivariance):** Nếu đầu vào bị hoán vị, đầu ra của hàm hoặc mô hình cũng sẽ bị hoán vị theo cùng cách đó. Điều này cần thiết cho *các tác vụ cấp nút (node-level tasks) hoặc các tác vụ cấp cạnh (edge-level tasks)*.

$$f(PAP^T, PX) = Pf(A, X)$$

với P là **ma trận hoán vị (permutation matrix)**. Tính đồng biến hoán vị (permutation equivariance) này đảm bảo các lớp GNN cập nhật biểu diễn nút một cách nhất quán, dù thứ tự nút đầu vào thay đổi.

3.2.4.3 Vì sao GNN được coi là phương pháp tiên tiến nhất (state-of-the-art)?

Mạng nơ-ron đồ thị (GNN) áp dụng cho bài toán hệ gợi ý là một hướng tiếp cận tương đối mới gần đây so với các phương pháp đã đề cập ở các mục trước và nhanh chóng trở thành hướng tiếp cận **tiên tiến, hàng đầu (state-of-the-art)**, điều này có thể lý giải ở 03 khía cạnh chính như sau:

- **Dữ liệu cấu trúc (Structured data):** Dữ liệu từ các nền tảng trực tuyến rất đa dạng (tương tác user-item, hồ sơ người dùng, thuộc tính sản phẩm, v.v.). GNN giúp biểu diễn toàn bộ dữ liệu dưới dạng đồ thị với các node và cạnh, từ đó khai thác đầy đủ thông tin và học được embedding chất lượng cao cho người dùng, sản phẩm và các đặc trưng liên quan.
- **Kết nối bậc cao (High-order connectivity):** Các mô hình truyền thống thường chỉ khai thác mối quan hệ bậc 1 giữa user-item, trong khi GNN có khả năng lan truyền và tổng hợp thông tin từ các lân cận nhiều bậc (multi-hop neighbors), từ đó tận dụng hiệu ứng lọc cộng tác một cách đầy đủ và nâng cao hiệu suất của hệ gợi ý.
- **Tín hiệu giám sát (Supervision signal):** GNN có thể xử lý tốt các tình huống thiếu hụt **tín hiệu giám sát (supervised signal)** bằng cách tận dụng các hành vi không mục tiêu (như tìm kiếm, thêm vào giỏ) thông qua **hoc bán giám sát (self-supervised learning)**. Ngoài ra, việc thiết kế các tác vụ phụ trợ trên đồ thị còn giúp tận dụng thêm thông tin và cải thiện khả năng học của mô hình.

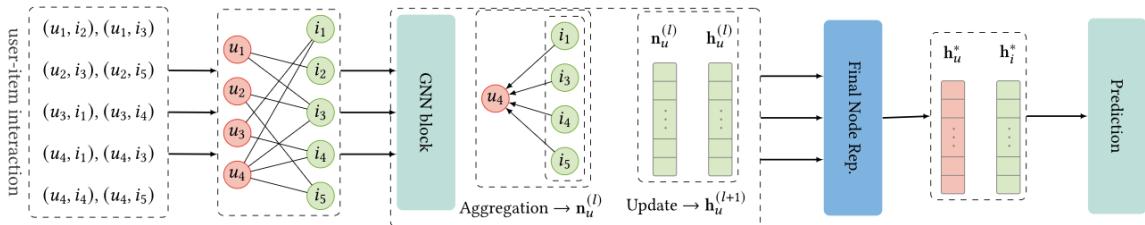
3.2.4.4 GNN Framework cho bài toán gợi ý dựa lọc cộng tác (CF)

Quy trình ở hình vẽ 3.8 có thể hiểu đơn giản như sau: Đầu vào của mô hình là cấu trúc **đồ thị hai phía user-item (user-item bipartite graph)**, sau khi đưa vào khối Message Passing Block thì **mô hình sẽ học các biểu diễn ẩn (embedding) của các nút (user, item) trên đồ thị**, đầu ra của mô hình là **biểu diễn ẩn cuối cùng (final representation)** của các nút phục vụ cho việc dự đoán. Chi tiết về quy trình như sau:

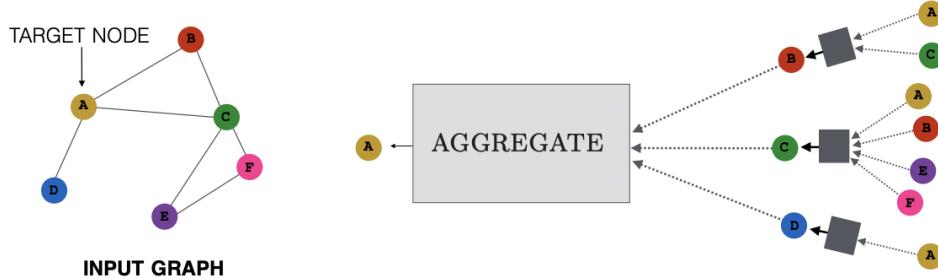
- **Xây dựng đồ thị (Graph Construction):** Khác với các mô hình gợi ý truyền thống – nơi dữ liệu đầu vào thường được biểu diễn dưới dạng bảng hoặc **ma trận tương tác người dùng – sản phẩm (user-item interaction matrix)**, các mô hình GNN cho bài toán gợi ý dựa trên lọc cộng tác (**Collaborative Filtering – CF**) yêu cầu biểu diễn dữ liệu dưới dạng **đồ thị hai phía (bipartite graph)**. Cụ thể, đồ thị được định nghĩa là $G = (U \cup V, E)$, trong đó:

- U là tập các nút người dùng (users), V là tập các nút sản phẩm (items). Trong đồ thị hai phía, không có cạnh nào giữa hai nút trong tập người dùng U hoặc hai nút trong tập sản phẩm V .
- $E \subseteq U \times V$ là tập các cạnh, mỗi cạnh $(u, v) \in E$ biểu thị một tương tác giữa người dùng u và item v (ví dụ: click, mua hàng, đánh giá, xem...).

Cấu trúc đồ thị này là nền tảng để mô hình GNN khai thác **tín hiệu cộng tác (collaborative signal)** thông qua các bước lan truyền thông tin giữa các node lân cận.



Hình 3.8: Framework chung cho các mô hình GNN áp dụng cho bài toán hệ gợi ý dựa trên **lọc cộng tác (CF)**.



Hình 3.9: Mô tả trực quan về cơ chế **lan truyền thông điệp (Message Passing Mechanism)** bên trong **Message Passing Block**. Mô hình sẽ tổng hợp các thông điệp từ các nút láng giềng gần của nút A (cụ thể là B, C và D). Các thông điệp từ những nút láng giềng này lại được tạo ra dựa trên thông tin được tổng hợp từ các nút lân cận của chính chúng, và quá trình này tiếp tục như vậy.

- **Khối lan truyền thông điệp (Message Passing Block):** Sau khi xây dựng đồ thị hai phía user-item, dữ liệu này được đưa vào **khối lan truyền thông điệp (message passing block)** nhằm học biểu diễn ẩn (embedding) cho từng node thông qua **các lớp lan truyền thông điệp (message passing layers)**. Mỗi lớp bao gồm ba bước chính:
 - **Xây dựng thông điệp (Message Construction):** Tạo thông điệp từ node lân cận, thường dựa trên embedding hiện tại và trọng số cạnh (nếu có).
 - **Tổng hợp thông điệp (Message Aggregation):** Tổng hợp thông điệp từ các node lân cận (sử dụng tổng, trung bình hoặc attention).
 - **Cập nhật thông tin (Information Update):** Cập nhật embedding mới của node trung tâm dựa trên embedding hiện tại và thông điệp tổng hợp.

Hình 3.9 là hình ảnh trực quan hóa về cơ chế lan truyền thông điệp thực hiện, quá trình này lặp lại qua nhiều tầng cho phép nút đích (target node) thu nhận thông tin từ các lân cận bậc cao hơn (multi-hop). Các mô hình GNN khác nhau như **GraphSAGE**, **GAT**, **GCN**,... sẽ có cách thiết kế các thao tác **Message Construction**, **Message Aggregation**, **Information Update** khác nhau.

- **Biểu diễn cuối cùng của nút (Final Node Representation):** Sau L lớp lan truyền, embedding cuối cùng của mỗi node (người dùng hoặc sản phẩm) sẽ được sử dụng trong các tác vụ downstream như tính điểm gợi ý, tính độ tương đồng, hoặc xếp hạng sản phẩm. Tùy theo mô hình, biểu diễn này có thể là embedding từ lớp cuối hoặc tổ hợp từ nhiều lớp (concatenation/sum).

Trong phạm vi của học phần, nhóm chúng tôi chỉ tập chung áp dụng các mô hình GNN dựa trên thiết kế của **Mạng tích chập đồ thị (GCN - Graph Neural Network)**. Phần tiếp theo sẽ trình bày tổng quan về GCN và mối quan hệ của GCN với các biến thể của nó được thiết kế dành riêng cho bài toán hệ gợi ý dựa trên **lọc cộng tác (CF - Collaborative Filtering)**.

3.2.4.5 Mối liên hệ giữa GCN và các mô hình hệ gợi ý

Graph Convolutional Network (GCN) được giới thiệu lần đầu bởi *Kipf và Welling (2016)* [24] như một phương pháp tổng quát hóa **phép tích chập (convolution)** cho dữ liệu đồ thị phi cấu trúc, nhằm

giải quyết bài toán phân loại nút (Node Classification) trên đồ thị đồng nhất (Homogeneous Graph).

Công thức tổng quát của GCN: GCN học biểu diễn (embedding) cho mỗi nút thông qua cơ chế lan truyền theo tầng (layer) như đã đề cập. Công thức cập nhật embedding cho nút v tại tầng $l + 1$ được định nghĩa như sau:

$$\mathbf{H}^{(l+1)} = \sigma \left(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^{(l)} \mathbf{W}^{(l)} \right)$$

Trong đó:

- $\mathbf{A} \in \mathbb{R}^{n \times n}$: ma trận kè của đồ thị (adjacency matrix), n là số node.
- $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$: ma trận kè có thêm self-loop.
- $\tilde{\mathbf{D}}$: ma trận bậc tương ứng với $\tilde{\mathbf{A}}$, tức $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$.
- $\mathbf{H}^{(l)} \in \mathbb{R}^{n \times d_l}$: ma trận embedding tại tầng l , với d_l là số chiều embedding.
- $\mathbf{W}^{(l)} \in \mathbb{R}^{d_l \times d_{l+1}}$: trọng số học được của tầng l .
- $\sigma(\cdot)$: hàm kích hoạt (thường dùng ReLU).

Giải thích hệ số chuẩn hoá đối xứng: Trong công thức GCN gốc, ta dùng ma trận $\tilde{A} = A + I$ (có self-loop) và $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. Phép nhân $\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ gọi là *normalized adjacency*, hay *symmetric normalization*.

- **Nguồn gốc từ spectral graph theory:** Normalized Laplacian định nghĩa là $L = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$. Kipf và Welling lấy xấp xỉ bậc nhất của bộ lọc phỏ trên đồ thị (ChebNet) dẫn đến phép nhân $\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$.
- **Ôn định giá trị eigen:** Tất cả eigenvalue của $\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ nằm trong khoảng $[-1, 1]$, giúp tránh vanishing/exploding gradient khi xấp xỉ nhiều tầng GCN.
- **Bất biến theo hoán vị (Permutation Invariance):** Chuẩn hoá đối xứng bảo đảm mỗi cạnh (u, v) đóng góp đồng nhất, không phụ thuộc thứ tự node.
- **Ôn định biên độ thông điệp:** Node có degree quá lớn hoặc quá nhỏ sẽ không chi phối quá mức tổng thông điệp, giúp embedding cân bằng giữa các node.

GCN là một trong những kiến trúc GNN đầu tiên mang tính tổng quát, với cơ chế **lan truyền thông điệp (message passing)** và **tổng hợp lân cận (neighborhood aggregation)** đơn giản nhưng hiệu quả. Tại mỗi tầng, biểu diễn (embedding) của một nút được cập nhật bằng cách tổng hợp biểu diễn của các nút láng giềng gần, sau đó áp dụng một hàm kích hoạt phi tuyến. Việc chia sẻ trọng số và bất biến hoán vị (permutation invariance) giúp mô hình học được đặc trưng từ cấu trúc đồ thị một cách hiệu quả, đồng thời tránh overfitting nhờ số lượng tham số nhỏ.

Chính những ưu điểm về tính ổn định, khả năng lan truyền cục bộ và bất biến hoán vị đã giúp GCN nhanh chóng trở thành nền tảng cho rất nhiều biến thể trong đa dạng các lĩnh vực, trong đó nổi bật nhất là bài toán hệ gợi ý dựa trên **lọc cộng tác (Collaborative Filtering – CF)**. Tuy nhiên, GCN gốc chưa được tối ưu cho bài toán này, do đó nhiều mô hình GCN-based CF ra đời như các mô hình: NGCF, LightGCN, UltraGCN,... để tận dụng tối ưu hơn GCN. Các phần sau của báo cáo trình bày sẽ trình bày cụ thể về các biến thể này.

3.2.4.6 Công thức tổng quát GCN-based CF

Với mỗi nút u bất kỳ ở tầng l , ta định nghĩa công thức chung cho 03 thao tác chính trong cơ chế **lan truyền thông điệp (message passing)** như sau:

$$\text{MESSAGE: } \mathbf{m}_{u \leftarrow i}^{(l)} = \text{MSG}^{(l)}(\mathbf{h}_u^{(l)}, \mathbf{h}_i^{(l)}, e_{ui}) \quad (3.2)$$

$$\text{AGGREGATE: } \mathbf{n}_u^{(l)} = \text{AGG}^{(l)}\left(\{\mathbf{m}_{u \leftarrow i}^{(l)} \mid i \in \mathcal{N}_u\}\right) \quad (3.3)$$

$$\text{UPDATE: } \mathbf{h}_u^{(l+1)} = \text{UPD}^{(l)}(\mathbf{h}_u^{(l)}, \mathbf{n}_u^{(l)}) \quad (3.4)$$

Trong đó:

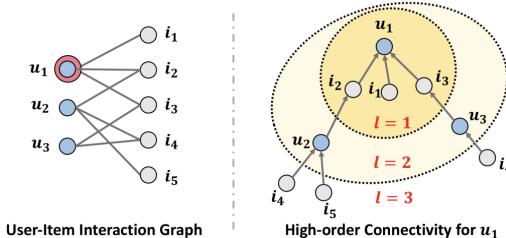
- $\mathbf{h}_u^{(l)} \in \mathbb{R}^d$ là biểu diễn ẩn (embedding) của node u tại layer l .
- $\mathbf{h}_i^{(l)} \in \mathbb{R}^d$ là biểu diễn ẩn (embedding) của node i tại layer l .
- $\mathcal{N}_u = \{i \mid (u, i) \in E\}$ là tập láng giềng của u .
- e_{ui} là trọng số hoặc đặc trưng cạnh (u, i) (nếu có).
- $\mathbf{m}_{u \leftarrow i}^{(l)}$ là thông điệp (message) được xây dựng từ láng giềng $i \rightarrow u$
- $\mathbf{n}_u^{(l)}$ là embedding đã được tổng hợp (aggregated representation) của các message đến u .
- $\text{MSG}^{(l)}, \text{AGG}^{(l)}, \text{UPD}^{(l)}$ lần lượt là các hàm **Message Construction**, **Message Aggregation** và **Information Update** ở layer l . Các hàm này sẽ được tùy chỉnh theo một thiết kế của một mô hình cụ thể.

Ở các phần sau khi đi chi tiết vào cách triển khai của thẻ 03 thao tác trên của từng mô hình, để diễn giải một cách mạch lạc hơn theo hình vẽ kiến trúc mô hình, chúng tôi sẽ dùng cả các ký hiệu khác nhau để trình bày về biểu diễn ẩn (embedding) của một nút v bất kỳ ở tầng thứ l , hai ký hiệu \mathbf{h}_v^l và \mathbf{e}_v^l có ý nghĩa tương tự nhau.

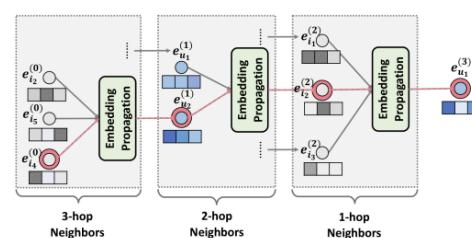
3.2.4.7 NGCF

NGCF (Neural Graph Collaborative Filtering) là mô hình tiên phong kế thừa kiến trúc GCN cho bài toán hệ gợi ý dựa trên **lọc cộng tác (CF)**, kiến trúc của NGCF thể hiện ở hình vẽ 3.12 bao gồm 03 thành phần chính:

- **Lớp Embedding (Embedding Layer):** Lớp này đóng vai trò khởi tạo embedding ban đầu cho tất cả các nút (user, item).
- **Các lớp lan truyền embedding (Embedding Propagation Layers):** Các lớp này đóng vai trò tinh chỉnh (huấn luyện) embedding bằng cơ chế lan truyền thông điệp (message passing).
- **Lớp dự đoán (Prediction Layer):** Lớp này đóng vai trò kết hợp các embedding từ nhiều lớp lan truyền embedding khác nhau, và đưa ra dự đoán (predict) về độ tương đồng của user u và item i



Hình 3.10: Minh họa cách mà nút u_1 (nút người dùng mục tiêu) cần cung cấp gợi ý, có thể khai thác thông tin kết nối bậc cao (high-order connectivity) để cập nhật biểu diễn ẩn (embedding) của chính nó.



Hình 3.11: Minh họa quá trình lan truyền nhúng (embedding propagation) bậc ba cho nút người dùng mục tiêu u_1 .

Lớp Embedding (Embedding Layer)

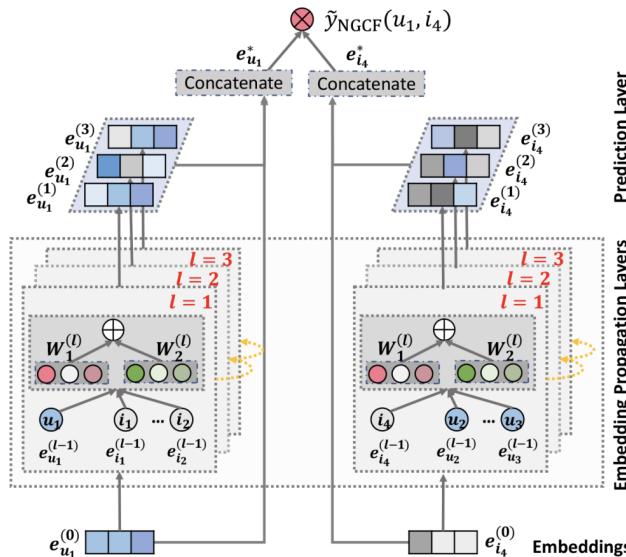
Mỗi user u và mỗi item i được khởi tạo bằng một véc-tơ nhúng (embedding) có kích thước d : $\mathbf{e}_u \in \mathbb{R}^d$, $\mathbf{e}_i \in \mathbb{R}^d$. Tất cả các embedding được lưu trong một bảng tra cứu (lookup table) $\mathbf{E} \in \mathbb{R}^{(N+M) \times d}$:

$$\mathbf{E} = \left[\underbrace{\mathbf{e}_{u_1}, \mathbf{e}_{u_2}, \dots, \mathbf{e}_{u_N}}_{\text{user embeddings}} \mid \underbrace{\mathbf{e}_{i_1}, \mathbf{e}_{i_2}, \dots, \mathbf{e}_{i_M}}_{\text{item embeddings}} \right].$$

Bảng tra cứu (lookup table) sẽ được tinh chỉnh dần dần qua các **Lớp lan truyền embedding (Embedding Propagation Layer)**.

Các lớp lan truyền embedding (Embedding Propagation Layers)

Dây là thành phần quan trọng nhất của NGCF. Thiết kế của các lớp này kế thừa và mở rộng kiến trúc của GCN ở 03 khía cạnh chính, thể hiện ở 03 thao tác **MESSAGE**, **AGGREGATE**, **UPDATE** như đã đề cập bên trên:



Hình 3.12: Minh họa kiến trúc tổng thể của mô hình NGCF (Các đường thông tin biểu thị luồng thông tin). Biểu diễn ẩn (embedding) của user u_1 (trái) và item i_4 (phải) được tinh chỉnh qua nhiều **Lớp lan truyền nhúng (embedding propagation layer)**, đầu ra của chúng được nối lại để đưa ra dự đoán cuối cùng.

- NGCF kế thừa thành phần *biến đổi tuyến tính* $\mathbf{W}_1^{(l)} \mathbf{h}_u^{(l)}$ trong GCN, đồng thời kết hợp *thêm thành phần tương tác phi tuyến* giữa user và item thông qua phép nhân Hadamard ($\mathbf{h}_u^{(l)} \odot \mathbf{h}_v^{(l)}$). Điều này giúp mô hình hóa tốt hơn sự ảnh hưởng hai chiều trong tương tác giữa user và item.

$$\text{MESSAGE : } \mathbf{m}_{u \leftarrow i}^{(l)} = \mathbf{W}_1^{(l)} \mathbf{h}_u^{(l)} + \mathbf{W}_2^{(l)} (\mathbf{h}_u^{(l)} \odot \mathbf{h}_i^{(l)})$$

Trong đó: $\mathbf{W}_1^{(l)}, \mathbf{W}_2^{(l)}$: Ma trận trọng số có thể huấn luyện ở lớp l , với kích thước $\mathbb{R}^{d_l \times d_{l-1}}$, trong đó d_l là kích thước embedding ở lớp l .

- $\mathbf{W}_1^{(l)}$: *Biến đổi tuyến tính* embedding của nút nguồn $\mathbf{h}_u^{(l)}$.
- $\mathbf{W}_2^{(l)}$: Biến đổi phép nhân Hadamard $\mathbf{h}_u^{(l)} \odot \mathbf{h}_v^{(l)}$, giúp nắm bắt các *tương tác phi tuyến* giữa embedding của người dùng và mặt hàng.

- NGCF kế thừa GCN, vẫn dùng **tổng chuẩn hoá đối xứng** (symmetrical normalization sum) để tổng hợp thông điệp từ các láng giềng.

$$\text{AGGREGATE : } \mathbf{n}_u^{(l)} = \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u| |\mathcal{N}_i|}} \mathbf{m}_{u \leftarrow i}^{(l)}$$

Trong đó **hệ số chuẩn hoá đối xứng** $\frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}}$, đơn giản là giúp ổn định về việc xây dựng thông điệp, tránh tình huống những nút có bậc (degree) cao (nhiều láng giềng) “áp đảo” việc xây dựng thông điệp chung.

- NGCF kế thừa GCN, vẫn dùng hàm kích hoạt phi tuyến $\sigma(\cdot)$ để nâng cao khả năng biểu diễn. **Hàm kích hoạt (activation function)** thường dùng trong mô hình NGCF là **ReLU** hoặc **LeakyReLU**.

$$\text{UPDATE : } \mathbf{h}_u^{(l+1)} = \sigma(\mathbf{n}_u^{(l)})$$

Lớp dự đoán (Prediction Layer): Lớp này thực hiện hai thao tác chính sau đây:

- **Biểu diễn cuối cùng (Final Representation):** Embedding cuối cùng của mỗi nút (node) được tạo bằng cách nối tất cả các embedding từ các lớp:

$$\begin{aligned} \mathbf{e}_u^* &= \mathbf{e}_u^{(1)} \| \mathbf{e}_u^{(2)} \| \dots \| \mathbf{e}_u^{(l)} \\ \mathbf{e}_i^* &= \mathbf{e}_i^{(1)} \| \mathbf{e}_i^{(2)} \| \dots \| \mathbf{e}_i^{(l)} \end{aligned}$$

- **Đưa ra dự đoán (Prediction):** Dự đoán độ yêu thích giữa user u và item i được tính bằng tích vô hướng giữa hai embedding cuối:

$$\hat{y}(u, i) = \mathbf{e}_u^{*\top} \cdot \mathbf{e}_i^*$$

Quy tắc lan truyền dưới dạng ma trận (Matrix–Form Propagation Rule): Toàn bộ cơ chế lan truyền thông điệp qua 1 lớp, với 03 thao tác chính ở mỗi tầng có thể được viết lại dưới dạng một công thức ma trận duy nhất:

$$\mathbf{E}^{(l)} = \text{LeakyReLU}\left((\mathcal{L} + \mathbf{I}) \mathbf{E}^{(l-1)} \mathbf{W}_1^{(l)} + \mathcal{L} \mathbf{E}^{(l-1)} \odot \mathbf{E}^{(l-1)} \mathbf{W}_2^{(l)}\right),$$

trong đó:

- $\mathbf{E}^{(l)} \in \mathbb{R}^{(N+M) \times d_l}$ là ma trận embedding của tất cả users và items sau l bước lan truyền.
- $\mathbf{W}_1^{(l)}, \mathbf{W}_2^{(l)} \in \mathbb{R}^{d_{l-1} \times d_l}$ là các ma trận trọng số (tuyến tính và phi tuyến) ở tầng l .
- $\mathcal{L} = \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$ là normalized adjacency matrix của đồ thị hai phía user-item, với

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{R} \\ \mathbf{R}^\top & \mathbf{0} \end{bmatrix}, \quad \mathbf{D}_{ii} = \sum_j A_{ij}$$

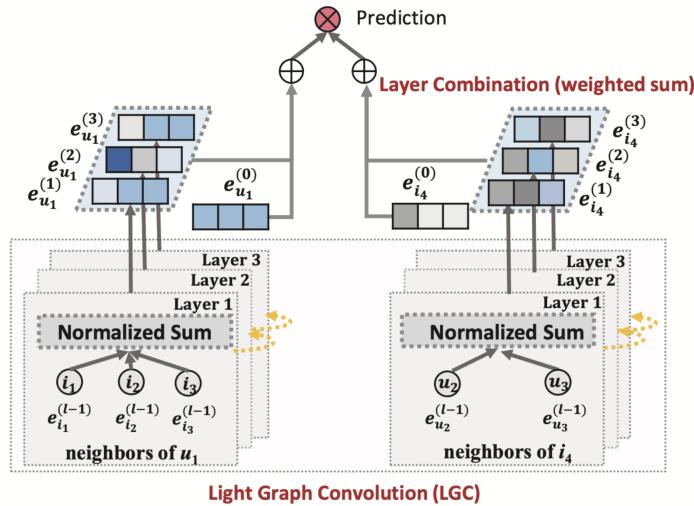
- \mathbf{I} là ma trận đơn vị, tương đương self-loop.
- \odot là phép nhân Hadamard giữa hai ma trận cùng kích thước, biểu diễn tương tác phi tuyến giữa embedding user và item.
- LeakyReLU(\cdot) là hàm kích hoạt phi tuyến, giúp NGCF học được các biểu diễn phức tạp hơn.

3.2.4.8 LightGCN

LightGCN (Light Graph Convolutional Network) là phiên bản nhẹ, gọn và đơn giản hơn rất nhiều so với mô hình NGCF truyền thống, LightGCN cải tiến NGCF bằng cách chỉ giữ lại đúng thành phần **neighborhood aggregation**, trong khi loại bỏ hoàn toàn **tự kết nối (self-loops)**, **biến đổi đặc trưng tuyến tính (feature transformation)**, **biến đổi phi tuyến (non-linear activation)**.

Do bản chất LightGCN là cải tiến của NGCF, hình vẽ 3.13 thể hiện kiến trúc của LightGCN vẫn gồm 03 thành phần chính giống NGCF, Paper của Xlanghe dùng tên gọi khác cho các Lớp lan truyền Embedding là **Light Graph Convolution (LGC)**, nhưng bản chất chúng là tương tự nhau.

Do đó ở phần này, chúng tôi sẽ chỉ đưa ra sự khác biệt của LightGCN, và sự khác biệt lớn nhất nằm ở 03 thao tác chính: **MESSAGE**, **AGGREGATE**, **UPDATE** trong cơ chế **lan truyền thông điệp (message passing mechanism)** ở các **tầng lan truyền thông điệp (message passing layer)**



Hình 3.13: Minh họa kiến trúc tổng thể của mô hình LightGCN. Trong LGC của LightGCN (tương ứng với Embedding Propagation Layer của NGCF), chỉ phép tính tổng chuẩn hóa (normalized sum) của các embedding láng giềng được thực hiện để truyền tới lớp tiếp theo, điều này làm đơn giản hóa đáng kể các lớp tích chập GCN. Trong phần Kết hợp lớp (Layer Combination), các embedding từ mỗi lớp để tổng hợp để thu được biểu diễn cuối cùng.

- **Loại bỏ kết nối vòng (self-loops)**, tức là mỗi bước xây dựng thông điệp (message construction), mỗi nút trên đồ thị chỉ nhận thông điệp từ các nút lân cận để phục vụ cho việc cập nhật vector biểu diễn ẩn của chính mình.

$$\text{MESSAGE : } \mathbf{m}_{u \leftarrow i}^{(l)} = \mathbf{h}_i^{(l)}$$

- **Loại bỏ biến đổi đặc trưng (feature transformation)**, tức là ở mỗi bước xây dựng thông điệp không đi kèm với phép nhân ma trận đặc trưng.

$$\text{AGGREGATE : } \mathbf{n}_u^{(l)} = \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u| |\mathcal{N}_i|}} \mathbf{m}_{u \leftarrow i}^{(l)}$$

- **Loại bỏ biến đổi phi tuyến (non-linear activation)**, tức là sau khi hoàn thành thu thập thông điệp (message aggregation), thì thông điệp này chính là biểu diễn ẩn (embedding) của nút mục tiêu (target node).

$$\text{UPDATE : } \mathbf{h}_u^{(l+1)} = \mathbf{n}_u^{(l)}$$

Ba thao tác MESSAGE, AGGREGATE, UPDATE bên trên có thể gộp lại thành một công thức duy nhất, ở mỗi lớp lan truyền thông điệp như sau:

$$\mathbf{h}_u^{(l+1)} = \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u| |\mathcal{N}_i|}} \mathbf{h}_i^{(l)}$$

Final Representation: Sự khác biệt thứ hai của LightGCN so với NGCF nằm ở **lớp dự đoán (Prediction Layer)**, cách kết hợp để thu được **biểu diễn cuối cùng (final representation)** trong LightGCN là sử dụng **phép tổng có trọng số (weighted sum)** embedding học được từ L lớp lan truyền:

$$\mathbf{h}_u^* = \sum_{l=0}^L \alpha_l \mathbf{h}_u^{(l)}, \quad \alpha_l = \frac{1}{L+1},$$

Prediction: Sau khi có được **biểu diễn cuối cùng (final embedding)**, mô hình đưa ra dự đoán user u và item i bằng phép tính tích vô hướng, tương tự giống như NGCF:

$$\hat{y}(u, i) = \mathbf{e}_u^{*\top} \mathbf{e}_i^*.$$

Matrix-Form Propagation Rule in LightGCN: Embedding đầu ra cuối cùng được tính bằng trung bình có trọng số của các embedding từ tất cả các tầng lan truyền:

$$\mathbf{E}^* = \alpha_0 \mathbf{E}^{(0)} + \alpha_1 \mathbf{E}^{(1)} + \cdots + \alpha_L \mathbf{E}^{(L)} = \sum_{k=0}^L \alpha_k \tilde{\mathbf{A}}^L \mathbf{E}^{(0)},$$

trong đó:

- $\mathbf{E}^{(0)} \in \mathbb{R}^{(N+M) \times d_0}$ là bảng embedding (look-up table) khởi tạo ban đầu.
- $\mathbf{E}^{(l)} \in \mathbb{R}^{(N+M) \times d_l}$ là ma trận embedding của tất cả users và items sau l bước lan truyền.
- $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$ là ma trận kè được chuẩn hoá đối xứng.
- α_l là trọng số kết hợp (thường lấy $\alpha_l = \frac{1}{L+1}$).

Chương 4

Các chiến lược đánh giá và độ đo áp dụng

4.1 Các độ đo đánh giá gợi ý

Trong các hệ thống gợi ý, các độ đo đánh giá mô hình thường tập trung vào nhóm các sản phẩm nằm trong **top- k kết quả được đề xuất** cho mỗi người dùng. Giá trị của k sẽ thay đổi tùy thuộc vào yêu cầu cụ thể của từng nền tảng ứng dụng, tuy nhiên trong các nghiên cứu học thuật, các giá trị phổ biến thường là $k = 10$ hoặc $k = 20$.

Để đánh giá mô hình trong top- k , mỗi sản phẩm trong danh sách gợi ý sẽ được gán nhãn nhị phân:

- Một item được xem là có liên quan (*relevant*, nhãn 1) nếu người dùng đã thực sự tương tác tích cực với item đó (ví dụ: mua, xem, đánh giá cao...).
- Ngược lại, item được xem là không liên quan (*irrelevant*, nhãn 0) nếu người dùng chưa từng tương tác, hoặc có hành vi phản hồi tiêu cực đối với sản phẩm đó.

Trong báo cáo này, chúng tôi sử dụng 5 độ đo phổ biến nhất để đánh giá hiệu quả của các mô hình hệ gợi ý, bao gồm: Precision@ k , Recall@ k , F1@ k , Hit Rate@ k , và NDCG@ k . Các độ đo này đều được tính trên tập top 10 item mà mô hình gợi ý cho mỗi người dùng, phản ánh khả năng đưa ra những gợi ý chính xác và đúng thứ tự ưu tiên.

4.1.1 Precision at k

Độ đo tính chính xác của đề xuất, xác định tỉ lệ phần *relevant* được đề xuất trên tổng số item được đề xuất

$$\text{Precision}@k = \frac{TP}{TP + FP} = \frac{|\text{relevant items recommended}|}{|k|}$$

Ý nghĩa: Precision cao cho thấy ít đề xuất “rác” (false positives).

4.1.2 Recall at k

Thước đo tính đầy đủ, xác định tỉ lệ phần *relevant* được đề xuất trên tổng số item *relevant* trên toàn bộ dataset

$$\text{Recall}@k = \frac{TP}{TP + FN} = \frac{|\text{relevant items recommended}|}{|\text{all relevant items}|}$$

Ý nghĩa: Recall cao cho thấy mô hình không bỏ sót nhiều item quan trọng (false negatives).

4.1.3 F₁ at k

Là trung bình điều hoà giữa Precision và Recall:

$$F1@k = \frac{2 \times \text{Precision}@k \times \text{Recall}@k}{\text{Precision}@k + \text{Recall}@k}.$$

Ý nghĩa: Kết hợp cả hai khía cạnh chính xác và đầy đủ của đề xuất.

4.1.4 NDCG at k

Normalized Discounted Cumulative Gain — thước đo độ hữu dụng của đề xuất giảm dần theo vị trí:

$$\text{DCG}@k = \sum_{i=1}^k \frac{2^{r_i} - 1}{\log_2(i + 1)}, \quad \text{NDCG}@k = \frac{\text{DCG}@k}{\text{IDCG}@k},$$

trong đó $r_i = 1$ nếu item thứ i trong đề xuất đúng, ngược lại 0; IDCG là DCG lý tưởng (khi sắp xếp tất cả item đúng lên đầu). Ý nghĩa: NDCG cao khi các item đúng được xếp ở thứ hạng cao.

4.1.5 Hit Rate at k

Kiểm tra xem trong đề xuất có ít nhất một item đúng hay không:

$$\text{HitRate}@k = \begin{cases} 1, & |\text{relevant items recommended}| \geq 1, \\ 0, & \text{ngược lại.} \end{cases}$$

Ý nghĩa: Một độ đo nhẹ hơn Precision và Recall, thích hợp khi dataset có độ thưa rất lớn

4.2 Các chiến lược đánh giá

Dữ liệu trong bộ Steam Games chứa yếu tố thời gian thông qua các cột *date* và *date_released*. Đây là yếu tố mang tính quyết định trong việc đánh giá mô hình một cách khách quan và thực tế. Nếu chia dữ liệu một cách ngẫu nhiên như các chiến lược phổ biến hiện nay (Hold-out, Cross-validation, Bootstrap Sampling), sẽ tiềm ẩn nguy cơ xảy ra hiện tượng rò rỉ dữ liệu thời gian (time leakage).

Time leakage là hiện tượng xảy ra khi thông tin từ tương lai (so với thời điểm dự đoán) vô tình xuất hiện trong tập huấn luyện. Điều này khiến mô hình học được các mẫu có liên hệ chặt chẽ với các sự kiện tương lai mà đáng lý ra không thể biết tại thời điểm dự đoán. Hậu quả là mô hình đạt hiệu suất rất cao trong giai đoạn đánh giá, nhưng lại không có khả năng tổng quát hóa khi triển khai trên dữ liệu thực tế – dẫn đến đánh giá sai lệch và tiềm ẩn rủi ro nghiêm trọng khi ứng dụng.

Lấy ví dụ cụ thể trong lĩnh vực hệ thống gợi ý, với phương pháp cơ bản là gợi ý theo độ phổ biến (popularity-based recommendation). Giả sử ta muốn dự đoán sản phẩm phù hợp cho người dùng tại thời điểm đầu tháng 4. Một item cụ thể trở nên cực kỳ phổ biến vào tháng 5 nhờ vào một xu hướng trên mạng xã hội. Nếu một phần dữ liệu tháng 5 bị trộn lẫn vào tập huấn luyện, mô hình sẽ học được rằng item đó có độ phổ biến cao và gợi ý cho người dùng, mặc dù tại thời điểm tháng 4 item này vẫn chưa có sức hút. Điều này cho thấy mô hình đã vô tình sử dụng thông tin từ tương lai để đưa ra quyết định – một vi phạm nghiêm trọng nguyên tắc nhân quả trong học máy.

Do đó, khi xử lý các bộ dữ liệu có yếu tố thời gian, bắt buộc phải chia tập dữ liệu theo trực thời gian nhằm đảm bảo tính khách quan, đúng ngữ cảnh và tránh hoàn toàn hiện tượng time leakage. Trong lĩnh vực hệ thống gợi ý, hai chiến lược đánh giá phổ biến tuân theo nguyên tắc này là **Full-corpus evaluation** và **Leave-one-last**, sẽ được trình bày chi tiết ở phần tiếp theo.

4.2.1 Full-corpus

Một chiến lược đánh giá mô hình vừa tự nhiên về mặt ý tưởng, vừa gần sát thực tế triển khai là phương pháp **Full-corpus Evaluation**, còn được gọi là **Temporal Hold-out Split**. Trong phương pháp này, một mốc thời gian cố định được chọn để phân chia dữ liệu cho toàn bộ người dùng. Cụ thể, tất cả các tương tác xảy ra trước thời điểm này sẽ được sử dụng để huấn luyện mô hình, còn các tương tác sau thời điểm đó sẽ dùng để đánh giá.

Khi thực hiện đánh giá, mô hình sẽ đưa ra gợi ý cho mỗi người dùng bằng cách xếp hạng toàn bộ các item có trong tập dữ liệu, ngoại trừ những item mà người dùng đó đã tương tác trước đó.

- Ưu điểm:** Phương pháp này phản ánh trung thực môi trường triển khai thực tế, nơi mà tại thời điểm dự đoán, hệ thống chỉ có thể tiếp cận thông tin từ quá khứ và cần đưa ra dự đoán cho tương lai. Do đó, nó đảm bảo không xảy ra rò rỉ thông tin theo thời gian và mô phỏng đúng quy trình sử dụng trong các hệ thống gợi ý thời gian thực.

- Hạn chế:** Tuy nhiên, chiến lược này tồn tại hai nhược điểm chính khiến nó không phổ biến bằng phương pháp *Leave-one-last*:

- Thứ nhất, việc đánh giá dựa trên toàn bộ tập item trong mỗi lần gợi ý khiến chi phí tính toán trở nên rất lớn, đặc biệt khi dataset có số lượng item lớn.

- Thứ hai, phương pháp này thường làm gia tăng các trường hợp *cold-start* (user hoặc item chỉ mới xuất hiện trong tập test). Dù đây là hiện tượng phổ biến trong thực tế, nhưng nó yêu cầu các mô hình phải có chiến lược xử lý cold-start kèm theo – điều mà các phương pháp collaborative filtering thường không hỗ trợ. Khi số lượng cold-start lớn, hiệu suất gợi ý của mô hình có thể không phản ánh đúng năng lực tổng thể mà bị ảnh hưởng nhiều bởi chiến lược xử lý kèm theo.

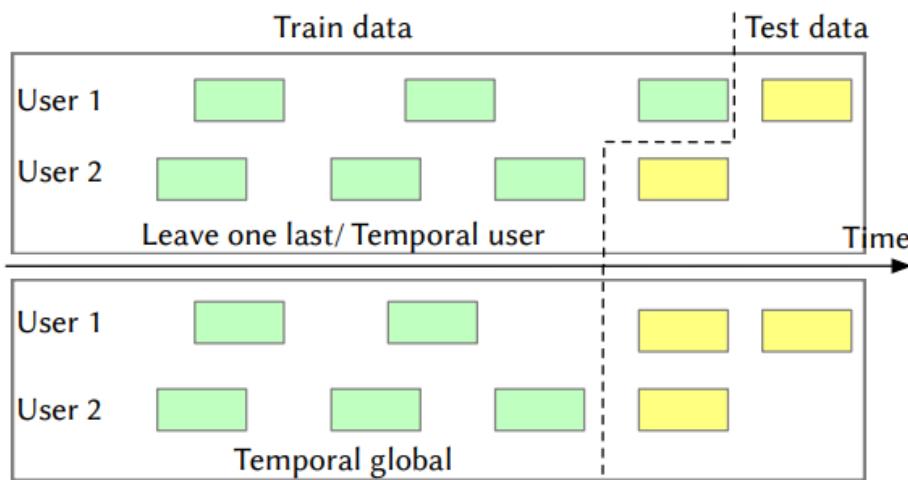
Tuy nhiên, với việc tập game trong bộ dữ liệu Steam Games có quy mô vừa phải (khoảng 4,600 trò chơi), nhược điểm liên quan đến chi phí tính toán của phương pháp Full-corpus Evaluation là không quá nghiêm trọng. Tổng số lượng item đủ nhỏ để việc đánh giá toàn bộ vẫn khả thi về mặt tài nguyên và thời gian xử lý. Do đó, chúng tôi hoàn toàn có thể áp dụng chiến lược đánh giá này nhằm đảm bảo tính khách quan, sát thực tế triển khai, đồng thời phản ánh đúng năng lực tổng quát hóa của các mô hình gợi ý.

4.2.2 Leave-one-last

Chiến lược đánh giá được sử dụng phổ biến nhất trong các nghiên cứu hệ gợi ý hiện nay là *Leave-one-last*. Đúng như tên gọi, phương pháp này giữ lại một tương tác tích cực cuối cùng của mỗi người dùng để đưa vào tập kiểm tra (test), trong khi toàn bộ các tương tác còn lại được sử dụng để huấn luyện mô hình.

Trong quá trình đánh giá, hệ thống sẽ thực hiện như sau: với mỗi người dùng, tương tác cuối cùng được xem là mục tiêu gợi ý cần dự đoán. Sau đó, item này sẽ được trộn lẫn với N item ngẫu nhiên mà người dùng đó chưa từng tương tác trước đó (nhưng xuất hiện trong tập huấn luyện). Tập hợp gồm $N + 1$ item này sẽ được đưa vào mô hình để tính điểm và xếp hạng, từ đó đánh giá khả năng của mô hình trong việc phát hiện đúng item mục tiêu.

Giá trị của N có thể được lựa chọn tùy theo quy mô của tập dữ liệu — phổ biến nhất là $N = 100$, nhưng cũng có thể là 200, 500 hoặc 1000 trong các thiết lập khác nhau. Chiến lược này cho phép kiểm tra khả năng phân biệt (*discriminative ability*) của mô hình trong không gian gợi ý thực tế, nơi mà số lượng item tiềm năng thường rất lớn.



Hình 4.1: So sánh 2 chiến lược đánh giá Full-corpus và Leave-one-last

- **Ưu điểm:** Phương pháp này giải quyết hiệu quả vấn đề chi phí tính toán cao của chiến lược *Full-corpus* bằng cách chỉ đánh giá trên một tập hợp con các item (gồm 1 item mục tiêu và N item nhiễu). Điều này giúp giảm đáng kể độ phức tạp tính toán, đặc biệt với các tập dữ liệu có số lượng item lớn. Ngoài ra, do chỉ giữ lại một tương tác cuối cùng để đánh giá, gần như toàn bộ dataset còn lại đều có thể sử dụng cho huấn luyện, từ đó mô hình được tiếp cận tối đa lượng dữ liệu sẵn có để học các đặc trưng người dùng và sản phẩm.

- **Hạn chế:**

- Một nhược điểm quan trọng của phương pháp *Leave-one-last* là không đảm bảo phân tách dữ liệu theo một mốc thời gian cố định trên toàn bộ người dùng. Thời điểm tương tác cuối cùng của mỗi user là khác nhau, dẫn đến nguy cơ **Time leakage** – khi mà dữ liệu từ tương lai (so với tương tác cuối cùng của một số user) có thể vô tình xuất hiện trong tập huấn luyện của user khác. Tuy nhiên, mức độ ảnh hưởng của hiện tượng này còn phụ thuộc vào đặc trưng của dataset, ví dụ như việc có tồn tại một bộ phận lớn người dùng không còn hoạt động trong giai đoạn cuối của tập dữ liệu hay không - và thông thường là không quá lớn vì khoảng thời gian trong test-set là không quá lớn.
- Nhược điểm thứ hai là các chỉ số phổ biến như Precision@k và Recall@k sẽ mất đi phần lớn ý nghĩa trong ngữ cảnh này, do chỉ có duy nhất một item mục tiêu được coi là “relevant” trong quá trình đánh giá. Khi đó, $\text{Precision}@k = \text{HitRate}@k / k$, còn $\text{Recall}@k = \text{HitRate}@k$. Vì vậy, trong chiến lược này, các chỉ số như Hit Rate và NDCG thường được sử dụng thay thế để phản ánh chính xác hơn hiệu suất mô hình.

Trong báo cáo này, chúng tôi sẽ đồng thời thực hiện đánh giá các mô hình trên cả 2 chiến lược này, nhằm so sánh và xác định sự ảnh hưởng đến từ các nhược điểm của mỗi phương pháp đến kết quả đánh giá.

Chương 5

Huấn luyện và đánh giá

5.1 Thuật toán huấn luyện Adam

Thuật toán Adam (Adaptive Moment Estimation) là một phương pháp tối ưu hóa mạnh mẽ và hiệu quả thường được sử dụng rộng rãi trong các mô hình học sâu. Đây là một cải tiến dựa trên các phương pháp tối ưu hóa trước đó như AdaGrad và RMSProp, đồng thời kết hợp cả ý tưởng của Momentum nhằm tăng tốc độ hội tụ và ổn định quá trình huấn luyện mô hình. Trong thuật toán Adam, mỗi tham số sẽ được cập nhật với hệ số học tập riêng biệt, điều này giúp cho việc tối ưu hóa trở nên linh hoạt hơn và không yêu cầu người dùng phải tinh chỉnh nhiều siêu tham số như trong các phương pháp truyền thống.

Công thức cập nhật trọng số trong Adam bao gồm nhiều bước quan trọng. Trước tiên, tại mỗi bước lặp t , ta tính gradient của hàm mất mát theo các tham số hiện tại:

$$g_t = \nabla J(\theta_t)$$

Tiếp theo, thuật toán ước lượng mô-men bậc nhất, tức là trung bình trượt mũ của gradient, theo công thức:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

Đồng thời, mô-men bậc hai, hay còn gọi là phương sai trượt mũ của gradient, cũng được cập nhật như sau:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Tuy nhiên, vì các giá trị ban đầu của m_t và v_t đều được khởi tạo bằng 0, nên ở những bước đầu tiên, các giá trị này có xu hướng bị lệch thấp so với thực tế. Để khắc phục vấn đề này, Adam áp dụng kỹ thuật hiệu chỉnh thiên lệch (bias correction):

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Sau khi có các giá trị mô-men đã được hiệu chỉnh, tham số của mô hình sẽ được cập nhật theo công thức sau:

$$\theta_{t+1} = \theta_t - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Trong đó, α là hệ số học tập (learning rate), β_1 và β_2 là các hệ số suy giảm mũ (thường được chọn lần lượt là 0.9 và 0.999), ϵ là một hằng số rất nhỏ (khoảng 10^{-8}) để tránh chia cho 0.

Ưu điểm nổi bật của thuật toán Adam là khả năng tự động điều chỉnh hệ số học tập cho từng tham số riêng biệt, nhờ đó giúp mô hình hội tụ nhanh hơn và ổn định hơn mà không đòi hỏi quá nhiều công sức trong việc lựa chọn và điều chỉnh thủ công hệ số học tập ban đầu. Ngoài ra, Adam còn tích hợp cả kỹ thuật Momentum thông qua mô-men bậc nhất và kỹ thuật chuẩn hóa gradient theo hướng tương tự RMSProp, giúp giảm dao động và tránh bị mắc kẹt tại các vùng phẳng hoặc cực trị địa phương. Thuật toán này đặc biệt hiệu quả khi áp dụng trên các mô hình có số lượng tham số lớn và dữ liệu đa dạng, phức tạp như mạng nơ-ron sâu.

So với thuật toán Gradient Descent truyền thống, Adam thể hiện rõ sự vượt trội về nhiều mặt. Trong Gradient Descent, hệ số học tập là cố định và nếu không được chọn phù hợp, mô hình có thể gặp khó khăn trong việc hội tụ hoặc mất rất nhiều thời gian để đạt đến nghiệm tối ưu. Ngược lại, Adam tự động thích nghi hệ số học tập theo từng bước huấn luyện và theo từng chiều của không gian tham số, mang lại tốc độ hội tụ nhanh hơn đáng kể. Đồng thời, nhờ vào việc sử dụng mô-men bậc nhất và bậc hai, Adam có khả năng điều chỉnh hướng di chuyển trong không gian tìm kiếm một cách linh hoạt hơn, giúp tránh xa các điểm cực trị địa phương hoặc vùng có độ dốc thấp. Nhờ những lợi thế này, Adam ngày càng được ứng dụng phổ biến trong các mô hình học máy hiện đại, đặc biệt là trong lĩnh vực trí tuệ nhân tạo và xử lý ngôn ngữ tự nhiên.

5.2 Kết quả huấn luyện và đánh giá

Bảng 5.1: Kết quả đánh giá mô hình trên Full Corpus và Leave-one-last

Mô hình	Full Corpus					Leave-one-last	
	P@10	R@10	F1@10	NDCG@10	HR@10	NDCG@10	HR@10
Content-based model	-	-	-	-	-	0.0003	0.0005
Neighborhood-based model							
ItemKNN	0.0027	0.0023	0.0025	0.0030	0.0233	0.0933	0.1422
Matrix Factorization-based models							
MF-Logistic	0.0014	0.0017	0.0015	0.0017	0.0142	0.0876	0.1815
MF-BPR	0.0111	0.0123	0.0117	0.0143	0.0927	0.1614	0.3151
MF-SVM	0.0110	0.0125	0.0117	0.0144	0.0920	0.1598	0.3034
GMF	0.0109	0.0120	0.0114	0.0129	0.0911	0.1061	0.3093
Deep MF-based models							
MLP	0.0120	0.0131	0.0125	0.0151	0.0982	0.1687	0.3142
NeuMF	0.0127	0.0140	0.0133	0.0161	0.1029	0.1655	0.3146
ConvNCF	0.0030	0.0033	0.0031	0.0031	0.0199	0.0816	0.1698
GNN-based models							
NGCF	0.0232	0.0267	0.0248	0.0299	0.1759	0.2728	0.5190
LightGCN	0.0246	0.0284	0.0264	0.0316	0.1855	0.2714	0.5185

Content-based Filtering so với Collaborative Filtering: Nhìn chung, kết quả của phương pháp Content-based Filtering thấp hơn đáng kể so với các phương pháp thuộc hướng tiếp cận Collaborative Filtering. Điều này phản ánh hạn chế trong việc khai thác đặc trưng nội dung văn bản (như mô tả game, thể loại, tag) để biểu diễn chính xác sở thích người dùng. Các phương pháp dựa trên nội dung thường phụ thuộc nhiều vào chất lượng và độ phong phú của các đặc trưng mô tả. Bên cạnh đó, Content-based Filtering không tận dụng được thông tin tương tác giữa người dùng với nhau, vốn là thế mạnh của Collaborative Filtering. Do đó, hệ thống gợi ý chỉ dựa trên nội dung thường gặp khó khăn trong việc nắm bắt các mối quan hệ ngầm giữa người dùng và game, dẫn đến hiệu suất thấp hơn đáng kể so với các phương pháp khác.

Các hàm mất mát cho Matrix Factorization: Chúng tôi tiến hành so sánh ba hàm mất mát phổ biến được sử dụng trong các mô hình phân rã ma trận (Matrix Factorization), bao gồm: **Binary Cross-Entropy Loss**, **Hinge Loss**, và **Bayesian Personalized Ranking Loss**. Kết quả thực nghiệm cho thấy **BPR** đạt hiệu suất tổng thể cao nhất, vượt trội hơn đáng kể so với hai hàm mất mát còn lại ở hầu hết các chỉ số đánh giá như HitRate và NDCG. Điều này đến từ việc BPR được thiết kế đặc biệt cho bài toán xếp hạng: thay vì học xác suất tương tác tuyệt đối, nó học thứ tự ưu tiên giữa các item, bằng

cách tối ưu hoá khoảng cách giữa các cặp item dương và âm. Ngoài ra, mặc dù vốn được thiết kế cho các dataset có implicit negative feedback, nhờ tận dụng thêm cả những explicit negative feedback trong dữ liệu khiến chất lượng mỗi cặp được huấn luyện là chắc chắn hơn. BCE Loss cho ra kết quả tương đối kém, phần nào kém hơn cả phương pháp dùng làm baseline là KNN - đến từ việc hàm mất mát này không được tối ưu cho nhiệm vụ xếp hạng và phiên bản MF gốc chưa có trọng số (đã cải thiện đáng kể khi nâng cấp lên GMF, dù vẫn kém hơn 2 hàm mất mát còn lại)

MLP, NeuMF và ConvNCF: Về kết quả kém hơn rất nhiều so với 2 mô hình còn lại của ConvNCF, có thể giải thích qua 2 ý chính: do nhóm chưa tìm được kiến trúc phù hợp với dữ liệu, và do tính chất về tương quan chéo giữa các đặc trưng ẩn của embedding là không quá quan trọng trong dataset này.

Các mô hình GNN-based so với phần còn lại: Dúng như nhiều nghiên cứu, bài khảo sát trình, các mô hình GCN cho hệ gợi ý dựa trên lọc cộng tác thực sự là một cách tiếp cận tiến tiến (state-of-the-art) nhất trong hệ gợi ý khi cho ra kết quả vượt xa các mô hình còn lại như **MF-based models**, **DeepMF-based models**.

- Tuy nhiên nhìn thật kỹ ở chiến lược đánh giá *Full-corpus*, **LightGCN** vượt trội hoàn so với NGCF, điều này phản ánh bản chất ưu điểm của chiến lược, tức là phản ánh tốt môi trường chiến khai thác tế, không bị ảnh hưởng bởi yếu tố ngẫu nhiên giống như chiến lược Leave One Last.
- Còn với chiến lược đánh giá *Leave One Last*, có thể bị ảnh hưởng bởi yếu tố ngẫu nhiên trong việc lấy mẫu. Vì chiến lược lấy mẫu **tuân theo phân phối đều (uniformly negative sampling)**, các **mẫu âm dễ (easy negative)** (nằm xa mẫu dương trong không gian ẩn) có thể được lấy nhiều hơn khi đánh giá NGCF. Điều này khiến cho NGCF dễ dàng phân biệt các **mẫu âm dễ** này và đưa ra kết quả NDCG@10, Hitrate@10 tốt hơn so với LightGCN. **Giải pháp có thể áp dụng:** Chạy thêm nhiều lần đánh giá NGCF, LightGCN trên các độ đo và tính trung bình để có được kết luận đúng đắn hơn.

Phụ lục

Các công cụ, thư viện sử dụng

Để thực hiện quá trình tiền xử lý dữ liệu cho bộ dữ liệu **Game Recommendations on Steam**, các thư viện và công cụ sau đây trong hệ sinh thái Python đã được sử dụng:

- **Pandas:**

- *Mục đích:* Là thư viện chính cho việc thao tác và phân tích dữ liệu dạng bảng.
- *Ứng dụng cụ thể:* Đọc các tệp CSV (“games.csv”, “users.csv”, “recommendations.csv”) và JSON (nếu có metadata) vào cấu trúc DataFrame; thực hiện các thao tác làm sạch dữ liệu như xử lý giá trị thiếu, loại bỏ trùng lặp; biến đổi kiểu dữ liệu; hợp nhất (merge/join) các DataFrame từ các tệp khác nhau; thực hiện các phép tính tổng hợp và nhóm dữ liệu.

- **NumPy:**

- *Mục đích:* Cung cấp hỗ trợ cho các mảng đa chiều và các phép toán số học hiệu suất cao.
- *Ứng dụng cụ thể:* Thường được sử dụng ngầm bởi Pandas cho các thao tác tính toán trên cột dữ liệu số. Có thể được sử dụng trực tiếp cho các phép biến đổi toán học phức tạp hoặc khi cần tối ưu hóa hiệu suất.

- **Scikit-learn (sklearn):**

- *Mục đích:* Thư viện học máy toàn diện, cung cấp nhiều công cụ hữu ích cho tiền xử lý.
- *Ứng dụng cụ thể:*
 - * `sklearn.preprocessing`: Bao gồm các công cụ để chuẩn hóa/tiêu chuẩn hóa dữ liệu (ví dụ: `MinMaxScaler`, `StandardScaler`), mã hóa đặc trưng hạng mục (ví dụ: `OneHotEncoder`, `LabelEncoder`).

- **Matplotlib & Seaborn:**

- *Mục đích:* Các thư viện trực quan hóa dữ liệu.
- *Ứng dụng cụ thể:* Tạo biểu đồ (histogram, box plot, scatter plot, heat matrix) để khám phá phân phối dữ liệu, xác định các giá trị ngoại lai, hiểu mối quan hệ giữa các biến, và kiểm tra kết quả của các bước tiền xử lý.

Ở giai đoạn **huấn luyện và đánh giá mô hình**, chúng tôi sử dụng thư viện **PyTorch & Pytorch Geometric**

- *Mục đích:* Cài đặt, huấn luyện và đánh giá các mô hình **học biểu diễn ẩn (latent representation learning)**.
- *Ứng dụng cụ thể:* Sử dụng các module của PyTorch để:
 - Xây dựng kiến trúc mạng cho các mô hình MF, NeuMF, NGCF, LightGCN,...
 - Tạo lớp `Dataset` và `DataLoader` để xử lý batch và negative sampling.
 - Định nghĩa hàm mất mát (BCE, BPR) và thuật toán tối ưu Adam cho việc huấn luyện.
 - Tận dụng GPU để tăng tốc quá trình huấn luyện và đánh giá hiệu năng mô hình.

Lời cảm ơn

Đây là dòng cuối cùng được viết vào báo cáo, thật sự rất khó để diễn tả cả một quá trình làm bài tập lớn chỉ với vọn vẹn vài từ, khi nhìn lại cả một chặng hành trình không dài, cũng không ngắn thì quả thật đây là một hành trình thật ý nghĩa.

Chúng tôi nhận được nhận rất nhiều từ học phần này, không chỉ về kiến thức, kỹ năng làm việc nhóm, giải quyết vấn đề mà học phần này còn giúp chúng tôi tìm thấy được niềm hăng say trong công việc. Và để có được nền tảng tốt hơn trong quá trình làm bài tập lớn, chúng tôi muốn gửi lời cảm ơn tới thầy **PGS.TS Thân Quang Khoát**, mỗi bài giảng của thầy từng tuần (offline trên lớp) và online trên Youtube, dù không nhiều, nhưng đó là những lượng kiến thức quý giá giúp chúng tôi hạ cánh an toàn ở học phần này.

Báo cáo này là kết tinh của cả một chặng đường dài cùng nhau khám phá, cùng nhau học hỏi, cùng nhau chịu đựng, cùng nhau vượt qua. Tính đến thời điểm báo cáo kết quả bài tập lớn, chúng tôi đã thực hiện 15 buổi họp, trung bình 1 buổi/tuần và vô số những giờ làm việc (đọc paper, lập trình, huấn luyện và đánh giá mô hình) không thể đếm đếm, cống hiến tận tụy vì mục tiêu chung.

Chúng tôi luôn có nhu cầu, đòi hỏi để làm ra những thứ tuyệt vời, dẫu biết rằng mọi thứ khó có thể hoàn hảo, nhưng mọi thứ có thể tuyệt vời hơn nếu dành nhiều thời gian hơn. Nhìn chung, mọi người trong team đều có tinh thần ý thức, trách nhiệm cao. Là một người đại diện của nhóm, tôi thật sự ghi nhận những đóng góp tiêu biểu của một số thành viên:

- **Tuấn Thành** được coi là **kiến trúc sư trưởng** của bài tập lớn, người thiết kế mọi giải pháp cho mọi vấn đề mà nhóm chúng tôi gặp phải trong thời gian làm bài tập lớn.
- **Mạnh Hưng** là **người tiên phong về công nghệ**, người luôn có những đề xuất về các công cụ giúp cả nhóm nâng cao hiệu quả làm việc.
- **Gia Hưng** là **người đáng tin cậy**, là người luôn đưa ra những giải pháp sáng tạo, đột biến trong các vấn đề chung của nhóm.

Tài liệu tham khảo

- [1] Newzoo. (2024). *Global Games Market Report* (Free Version, Updated May 2024). Newzoo. <https://tinyurl.com/58vf4w6m>
- [2] Batra, S., Sharma, V., Sun, Y., Wang, X., & Wang, Y. (2023, May 3). *Steam Recommendation System*. arXiv preprint arXiv:2305.04890. <https://arxiv.org/abs/2305.04890>
- [3] Recommendation Systems (Chapter 9 from Mining of Massive Datasets) Leskovec, J., Rajaraman, A., & Ullman, J. D. (2014). *Recommendation Systems*. In *Mining of Massive Datasets*. Cambridge University Press. <http://infolab.stanford.edu/~ullman/mmds/ch9.pdf>
- [4] Leskovec, J. (2021). *Recommender Systems and Collaborative Filtering*. CS124, Stanford University. <https://web.stanford.edu/class/cs124/lec/collaborativefiltering21.pdf>
- [5] Leskovec, J., Rajaraman, A., & Ullman, J. D. (2014). *Recommendation Systems*. In *Mining of Massive Datasets*. Cambridge University Press. <http://infolab.stanford.edu/~ullman/mmds/ch9.pdf>
- [6] Tiep, V. H. (2017, May 17). *Bài 23: Content-based Recommendation Systems*. Machine Learning cơ bản. <https://tinyurl.com/5x4ecx9f>
- [7] Tiep, V. H. (2017, May 24). *Bài 24: Neighborhood-Based Collaborative Filtering*. Machine Learning cơ bản. <https://tinyurl.com/4y7bsfa9>
- [8] Tiep, V. H. (2017, May 31). *Bài 25: Matrix Factorization Collaborative Filtering*. Machine Learning cơ bản. <https://tinyurl.com/mt3jxx6t>
- [9] Tiep, V. H. (2017, Jun 7). *Bài 26: Singular Value Decomposition*. Machine Learning cơ bản. <https://tinyurl.com/y3eb5h3m>
- [10] Tiep, V. H. (2021). *Factorization machine*. Machine Learning cơ bản. <https://tinyurl.com/rw4kwjat>
- [11] Le, J. (2020, February 24). *Recommendation System Series Part 4: The 7 Variants of MF For Collaborative Filtering. Towards Data Science*. <https://tinyurl.com/ypc7pa32>
- [12] Srebro, N., Rennie, J., & Jaakkola, T. (2004). *Maximum-Margin Matrix Factorization*. Advances in Neural Information Processing Systems, 17. https://proceedings.neurips.cc/paper_files/paper/2004/file/e0688d13958a19e087e123148555e4b4-Paper.pdf
- [13] Rendle, S. (2012, May 11). *BPR: Bayesian Personalized Ranking from Implicit Feedback*. arXiv preprint arXiv:1205.2618. <https://arxiv.org/pdf/1205.2618>
- [14] Wu, H., Liu, X., Ma, C., & Tang, R. (2022). *Adapting Triplet Importance of Implicit Feedback for Personalized Recommendation*. arXiv preprint arXiv:2208.01709. <https://arxiv.org/pdf/2208.01709>
- [15] Ma, H., Xie, R., Meng, L., Feng, F., Du, X., Sun, X., Kang, Z., & Meng, X. (2024). *Negative Sampling in Recommendation: A Survey and Future Directions*. arXiv preprint arXiv:2409.07237. <https://arxiv.org/pdf/2409.07237>
- [16] He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017, August 26). *Neural Collaborative Filtering*. arXiv preprint arXiv:1708.05031. <https://arxiv.org/abs/1708.05031>

- [17] He, X., & Chua, T.-S. (2017). *Neural Factorization Machines for Sparse Predictive Analytics*. In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (pp. 355–364). ACM. <https://arxiv.org/pdf/1708.05027>
- [18] He, X., Du, X., Wang, X., Tian, F., Tang, J., & Chua, T.-S. (2018, August 12). *Outer Product-based Neural Collaborative Filtering*. arXiv preprint arXiv:1808.03912. <https://arxiv.org/abs/1808.03912>
- [19] Wu, S., Sun, F., Zhang, W., Xie, X., & Cui, B. (2022, December 3). *Graph Neural Networks in Recommender Systems: A Survey*. ACM Computing Surveys, 55(5), 97. <https://doi.org/10.1145/3535101>
- [20] Gao, C., Zheng, Y., Li, N., Li, Y., Qin, Y., Piao, J., Quan, Y., Chang, J., Jin, D., He, X., & Li, Y. (2023, March 3). *A Survey of Graph Neural Networks for Recommender Systems: Challenges, Methods, and Directions*. ACM Transactions on Recommender Systems, 1(1), 3. <https://dl.acm.org/doi/10.1145/3568022>
- [21] Hamilton, W. L. (2020, September). *The Graph Neural Network Model*. In Graph Representation Learning Book. Retrieved from https://www.cs.mcgill.ca/~wlh/grl_book/
- [22] Hamilton, W. L. (2020, September). *Graph Neural Networks in Practice*. In Graph Representation Learning Book. Retrieved from https://www.cs.mcgill.ca/~wlh/grl_book/
- [23] O’Shea, K., & Nash, R. (2015). *An introduction to convolutional neural networks*. arXiv preprint arXiv:1511.08458. <https://arxiv.org/pdf/1511.08458>
- [24] Kipf, T. N., & Welling, M. (2017). *Semi-supervised classification with graph convolutional networks*. International Conference on Learning Representations (ICLR). <https://arxiv.org/pdf/1609.02907>
- [25] Wang, X., He, X., Wang, M., Feng, F., & Chua, T.-S. (2020). *Neural Graph Collaborative Filtering*. arXiv preprint arXiv:1905.08108 <https://arxiv.org/pdf/2002.0212>
- [26] He, X., Deng, K., Li, Y., & Zhang, Y. (2020). LightGCN: *Simplifying and Powering Graph Convolution Network for Recommendation*. arXiv preprint arXiv:2002.02126. <https://arxiv.org/pdf/2002.02126>
- [27] Meng, Z., McCreadie, R., Macdonald, C., & Ounis, I. (2020, July 26). *Exploring Data Splitting Strategies for the Evaluation of Recommendation Models*. arXiv preprint arXiv:2007.13237. <https://arxiv.org/pdf/2007.13237>
- [28] Campos, P. G., Díez, F. J., & Cantador, I. (2012). *Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols*. User Modeling and User-Adapted Interaction, 22(4-5), 421-455. <https://doi.org/10.1007/s11257-012-9136-x>
- [29] Kula, P., Wróbel, Ł., & Szymański, J. (2020). *Modeling Online Behavior in Recommender Systems: The Importance of Temporal Context*. arXiv preprint arXiv:2007.13237. <https://arxiv.org/pdf/2009.08978>
- [30] Cerqueira, V., Torgo, L., & Mozetic, I. (2019). *Evaluating time series forecasting models*. arXiv preprint arXiv:1905.11744. <https://arxiv.org/pdf/1905.11744>