ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA CÔNG NGHỆ PHẦN MỀM

BÁO CÁO ĐỒ ÁN MÔN HỌC

ĐỒ ÁN CHUYÊN NGÀNH

ĐỀ TÀI

Nghiên cứu 1 số giải thuật khuyến nghị và ứng dụng vào hệ thống khuyến nghị phim điện ảnh

Giảng viên hướng dẫn: Nguyễn Tấn Toàn

Lớp: **SE112.K21.PMCL**

Nhóm thực hiện : **Phan Thanh Tùng – 16521399**

TP. Hồ Chí Minh, tháng 5 năm 2020

Contents

LỜI CẢM ƠN

Đề tài “Nghiên cứu 1 số giải thuật khuyến nghị và ứng dụng vào hệ thống khuyến nghị phim điện ảnh” là nội dung em chọn để thực hiện sau khi kết thúc môn học Đồ án chuyên ngành tại Trường Đại học Công nghệ thông tin – Đại học Quốc gia Thành phố Hồ Chí Minh.

Để hoàn thành quá trình tìm hiểu kiến thức và xây dựng ứng dụng, lời đầu tiên em xin chân thành cảm ơn sâu sắc đến Thầy Nguyễn Tấn Toàn thuộc Khoa Công nghệ phần mềm – Trường Đại học Công nghệ thông tin. Thầy đã trực tiếp giảng dạy, chỉ bảo và hướng dẫn chúng em trong suốt quá trình học tập và tìm hiểu để thực hiện đề tài này. Ngoài ra chúng em xin chân thành cảm ơn các Thầy, Cô trong Khoa Công nghệ phần mềm đã đóng góp những ý kiến quý báu để đề tài được hoàn thiện hơn.

Cuối cùng, xin cảm ơn những anh chị sinh viên khoá trên, bạn bè đã hỗ trợ, giúp đỡ nhóm trong suốt khoảng thời gian vừa qua!

PHẦN 1: TỔNG QUAN

1. Lý do chọn đề tài:

Ngày nay, nhu cầu xem phim của người dùng đang phát triển rất nhanh và trở thành món ăn tinh thần của mọi người nhất là trong khoảng thời gian cách ly xã hội vì dịch Covid19 hiện nay. Tuy nhiên, số lượng phim hiện có là một con số cực kỳ lớn, do đó việc để lựa chọn xem phim nào trở thành một quyết định hết sức khó khăn. Từ đó, em quyết định chọn nghiên cứu giải thuật khuyến nghị để ứng dụng nó vào hệ thống khuyến nghị phim điện ảnh, để giúp cho người dùng có được 1 công cụ gợi ý hữu ích, giúp cho việc lựa chọn phim để xem trở nên đơn giản và nhanh hơn.

2. Mục đích

Hệ thống là 1 trang web với hệ thống cung cấp 1 lượng lớn phim hiện hành với các thông tin chi tiết để người dùng có thể nắm được thông tin phim, cho phép người dùng chấm điểm phim (rating) cũng như để lại review, cho phép thêm phim vào danh sách yêu thích để từ các thông tin trên đưa ra những khuyến nghị các bộ phim tương tự mà người dùng có thể thích. Giao diện của website cần trực quan, dễ hiểu, dễ thao tác.

3. Môi trường phát triển

- **Python**

  Python là ngôn ngữ hướng đối tượng bậc cao, dùng để phát triển website và nhiều ứng dụng khác nhau. Mặc dù sở hữu cú pháp cực kì đơn giản và thanh lịch, tuy nhiên đây cũng là ngôn ngữ nổi tiếng về sự chặt chẽ, nhanh và mạnh

- **SQL**

  SQL là hệ quản trị cơ sở dữ liệu tự do nguồn mở phổ biến nhất thế giới và được các nhà pháy triển ưa chuộng trong quá trình phát triển ứng dụng nhờ tốc độ cao, ổn định và dễ sử dụng, có tính khả chuyển, hoạt động trên nhiều hệ điều hành.

- **Reactjs**

  Reactjs là 1 thư viện được sử dụng phổ biến nhất của javascript dùng để phát triển frontend của các website. Reactjs giúp cho nhà phát triển tạo ra UI một cách đơn giản và hiệu quả hơn nhà tính tái sử dụng, giúp cho code trở nên dễ đoán hơn và dễ để debug hơn

- **IDE: Visual Studio Code**

  Là một IDE mạnh mẽ chính thức dành cho phát triển các ứng dụng với nền tảng mọi loại ngôn ngữ khác nhau. Hỗ trợ nhiều hệ điều hành như Windows, Mac OS X và Linux, và là IDE được phát triển bởi Microsoft và là phiên bản gọn nhẹ hơn của Visual Studio.

4. Kế hoạch thực hiện đồ án

Thời gian thực hiện: 19/04/2020 – 14/06/2020
Sản phẩm dự kiến: 1 trang web với hệ thống cung cấp 1 lượng lớn phim hiện hành với các thông tin chi tiết để người dùng có thể nắm được thông tin phim, cho phép người dùng chấm điểm phim (rating) cũng như để lại review, cho phép thêm phim vào danh sách yêu thích để từ các thông tin trên đưa ra những khuyến nghị các bộ phim tương tự mà người dùng có thể thích. Giao diện của website cần trực quan, dễ hiểu, dễ thao tác.
Phạm vi: những người yêu thích xem phim và có nhu cầu đóng góp cũng như muốn nhận khuyến nghị cho những phim mới

<table>
<tr><td colspan="2">Công nghệ phát triển dự kiến:
Front end: Reactjs
Back end: Python
Hệ quản trị cơ sở dữ liệu: SQL</td></tr>
<tr><td colspan="2">Kế hoạch thực hiện:</td></tr>
<tr><td>Tuần 1
(19/04/2020 –
25/04/2020)</td><td>Tìm hiểu về các giải thuật khuyến nghị
- Tìm hiểu các giải thuật khuyến nghị hiện có cũng như những giải thuật phù hợp với đồ án để phát triển hệ thống
- Tìm hiểu về các trang khuyến nghị phim điện ảnh hiện có</td></tr>
<tr><td>Tuần 2
(26/04/2020 –
02/05/2020)</td><td>Thiết kế hệ thống
- Phân tích nghiệp vụ và yêu cầu phần mềm.
- Thiết kế hệ thống: các sơ đồ UML (use – case, activity diagram)
- Thiết kế cơ sở dữ liệu</td></tr>
<tr><td>Tuần 3
(03/04/2020 –
09/05/2020)</td><td>Coding chức năng hiển thị phim
- Cài đặt cơ sở dữ liệu
- Cài đặt các RESTful API có liên quan
- Cài đặt các ReactJS Component FrontEnd có liên quan: hiển thị top phim (điểm cao nhất, nhiều review nhất, ...), hiển thị phim theo thể loại phim, ...</td></tr>
<tr><td>Tuần 4
(10/05/2020 –</td><td>Coding chức năng tài khoản người dùng
- Cài đặt cơ sở dữ liệu
- Cài đặt các RESTful API có liên quan</td></tr>
</table>

16/05/2020)	- Cài đặt các ReactJS Component FrontEnd có liên quan: đăng nhập, đăng ký, quản lý thông tin cá nhân, thêm phim vào danh sách yêu thích, ...
Tuần 5 – Tuần 6 (17/05/2020 – 30/05/2020)	Sprint 5: Coding chức năng rating + review - Cài đặt cơ sở dữ liệu - Cài đặt các RESTful API có liên quan - Cài đặt các ReactJS Component FrontEnd có liên quan: hiển thị rating của phim, cho phép người dùng chấm điểm phim, cho phép người dùng review phim
Tuần 7 – Tuần 8 (31/05/2020 – 14/06/2020)	Cài đặt thuật toán khuyến nghị và hoàn thiện trang web - Cài đặt thuật toán bằng ngôn ngữ Python - Kiểm thử và hoàn thiện trang web

PHẦN 2: TÌM HIỂU VỀ CÁC HỆ THỐNG KHUYẾN NGHỊ

1. Tổng quan:

Hệ thống khuyến nghị (Recommend System) là 1 thành phần trong hệ thống thông tin. Mục đích của nó là hỗ trợ người dùng tìm kiếm được đúng thông tin cần thiết, dự đoán sở thích hay xếp hạng mà người dùng có thể dành cho một mục thông tin (item) nào đó mà họ chưa xem xét tới trong quá khứ.

Nội dung liên quan nói trên chính là các gợi ý, là kết quả được tính toán dựa trên việc thu thập dữ liệu về người dùng như khi mua hàng, khi đưa ra các đánh giá cá nhân. Việc thực hiện tính toán được xây dựng trên các thuật toán Học máy (Machine Learning), đưa ra các dự đoán tốt nhất về sản phẩm mà người dùng có thể thích, giúp tối ưu hóa doanh thu qua up-sale, cross-sale. Cải thiện trải nghiệm người dùng, tăng hiệu năng hoạt động bằng tự động hóa, biến khách hàng tiềm năng trở thành khách hàng thật.

VD: Hệ thống bán hàng của Amazon, họ sẽ dựa vào những gì người sử dụng đã mua trong quá khứ, những sản phẩm mà bạn đã bấm like, cho điểm hay những sản phẩm vẫn còn đang trong giỏ hàng của bạn. Vì thế, nếu bạn là một tín đồ công nghệ, Amazon sẽ gợi ý cho bạn rất nhiều sản phẩm công nghệ hay ho khác. Còn nữa này, ở phần people you my know của facebook, mặc dù có những người không có bạn chung nào, nhưng facebook vẫn gợi ý để bạn kết bạn, tất cả đó đều là recommendation system.

2. Hướng tiếp cận:

Các hệ thống khuyến nghị thường sử dụng nhiều thuật toán khác nhau, tuy nhiên, về cơ bản, ta có thể chia làm 2 nhóm lớn:

- Lọc cộng tác (Collaborative Filtering): hệ thống gợi ý items dựa trên sự tương quan giữa các users và/hoặc items. Có thể hiểu rằng ở hướng tiếp cận này, 1 item được gợi ý tới 1 user dựa trên những user có hành vi tương tự. Vd: 3 user A, B, C đều thích các bộ phim của Marvel. Ngoài ra, hệ thống biết rằng B và C còn thích xem các bộ phim khoa học viễn tưởng nhưng chưa có thông tin về việc liệu A có thích thể loại này hay không. Dựa trên thông tin của những user tương tự là B và C, hệ thống có thể dự đoán là A cũng thích các bộ phim thuộc thể loại trên và từ đó gợi ý các bộ phim thuộc thể loại này tới A.
- Lọc dựa vào nội dung (Content-base System): đánh giá đặc tính của item được gợi ý. Vd: 1 người xem rất nhiều phim về cảnh sát hình sự, vậy thì gợi ý những bộ phim trong CSDL có chung đặc tính hình sự tới user này, ví dụ như phim "Người phán xử". Cách tiếp cận này yêu cầu việc sắp xếp các item vào từng nhóm hoặc đi tìm đặc trưng của từng item. Tuy nhiên, có những item không có nhóm cụ thể và việc xác định nhóm hoặc đặc trưng của từng item đôi khi là bất khả thi

3. Các thành phần của 1 hệ thống khuyến nghị:

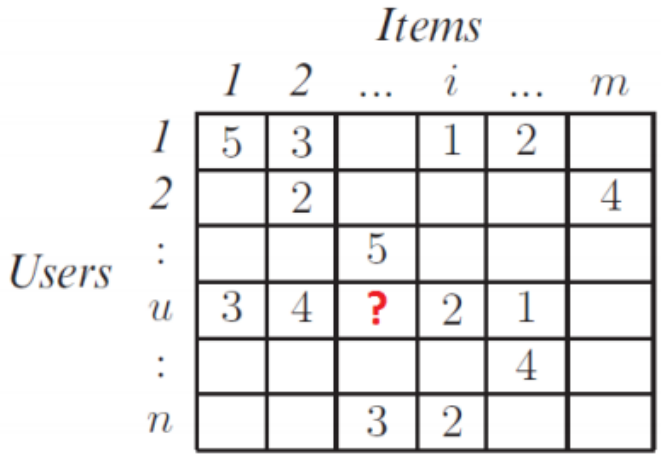
1. Dữ liệu:

Đầu tiên, ta cần có dữ liệu về users, items, feedback. Trong đó:

- Users: danh sách người dùng
- Items: danh sách sản phẩm, đối tượng của hệ thống. Ví dụ trong phạm vi của đồ án này là danh sách các bộ phim. Và mỗi item có thể kèm theo thông tin mô tả

- Feedback là lịch sử tương tác của user với mỗi item, có thể là đánh giá của user với mỗi item, số ratings, hoặc comment, việc user xem thông tin của item, like item, ...

2. Ma trận user-item (Utility Matrix):



Như đã đề cập, 2 thực thể chính trong hệ thống khuyến nghị là users và items. Mỗi user sẽ có mức độ quan tâm (degree of preference) tới từng item khác nhau. Mức độ quan tâm này ta tạm gọi là rating. Tập tất cả ratings, bao gồm các giá trị chưa biết cần được dự đoán tạo nên 1 ma trận gọi là utility matrix.

Ma trận này được xây dựng từ dữ liệu ở mục 1. Nhưng ma trận này có rất nhiều giá trị miss. Nhiệm vụ của hệ thống khuyến nghị là dựa vào các ô đã có giá trị trong ma trận trên (dữ liệu thu thập được trong quá khứ), thông qua mô hình đã được xây dựng, dự đoán giá trị của các ô còn trống (của user hiện hành), sau đó sắp xếp kết quả dự đoán (vd từ cao xuống thấp) và chọn ra Top-N items theo thứ tự rating giảm dần, từ đó gợi ý chúng cho người dùng

3. Ví dụ về Utility Matrix

	A	B	C	D	E	F
Mưa nửa đêm	5	5	0	0	1	?
Cỏ úa	5	?	?	0	?	?
Vùng lá me bay	?	4	1	?	?	1
Con cò bé bé	1	1	4	4	4	?
Em yêu trường em	1	0	5	?	?	?

Ta xét ví dụ về utility matrix với hệ thống gợi ý bài hát. Trong ví dụ này, có 6 users A, B, C, D, E, F và 5 bài hát. Các ô màu xanh thể hiện 1 user đã đánh giá 1 bài hát với ratings từ 0 (không thích) đến 5 (rất thích). Các ô có dấu '?' màu xám tương ứng với các ô chưa có dữ liệu. Công việc của 1 Recommendation Systems là dự đoán giá trị tại các ô màu xám này, từ đó đưa ra gợi ý cho người dùng. Recommendation Systems, vì vậy, đôi khi được coi là bài toán Matrix Completion (Hoàn thiện ma trận)

Trong ví dụ đơn giản này, dễ thấy có 2 thể loại nhạc khác nhau: 3 bài đầu là nhạc Bolero và 2 bài sau là nhạc thiếu nhi. Từ dữ liệu này, ta cũng có thể dự đoán được là A, B thích thể loại Bolero; C, D, E, F thích thể loại thiếu nhi. Từ đó 1 hệ thống tốt nên gợi ý 'Cỏ úa' cho B; 'Vùng lá me bay' cho A; 'Em yêu trường em' cho D, E, F. Giả sử chỉ có 2 thể loại nhạc này, khi có 1 bài hát mới, ta chỉ cần phân lớp nó vào thể loại nào, từ đó đưa ra gợi ý tới người dùng.

Thông thường, có rất nhiều users và items trong hệ thống, và mỗi user thường chỉ rate 1 số lượng rất nhỏ item, thậm chí có những user không rate item nào. Vì vậy, lượng ô màu xám của utility matrix trong cái bài toán đó thường rất lớn, và lượng các ô đã được điền là 1 số rất nhỏ.

Rõ ràng rằng càng nhiều ô được điền thì độ chính xác của hệ thống sẽ càng được cải thiện. Vì vậy, các hệ thống luôn luôn hỏi người dùng về sự quan tâm của họ tới sản phẩm, và muốn người dùng đánh giá càng nhiều sản phẩm càng tốt. Việc đánh giá các sản phẩm, vì thế, không những giúp các người dùng khác biết được chất lượng sản phẩm mà còn giúp hệ thống biết được sở thích của người dùng, qua đó có chính sách quảng cáo hợp lý.

4. Xây dựng Utility Matrix

Không có utility matrix, gần như rất khó để gợi ý sản phẩm tới người dùng, ngoài cách luôn luôn gợi ý những sản phẩm phổ biến nhất. Vì vậy trong Recommend System, việc xây dựng Utility Matrix là tối quan trọng. Tuy nhiên, việc xây dựng ma trận này thường có gặp nhiều khó khăn. Có 2 hướng tiếp cận phổ biến để xác định rating cho mỗi cặp user-item trong utility matrix:

1. Nhờ người dùng rate sản phẩm. Amazon luôn nhờ người dùng rate các sản phẩm của họ bằng cách gửi email nhắc nhở nhiều lần. Rất nhiều hệ thống khác cũng làm việc tương tự. Tuy nhiên, cách tiếp cận này có một vài hạn chế, vì thường thì người dùng ít khi rate sản phẩm. Và nếu có, đó có thể là những đánh giá thiên lệch bởi những người sẵn sàng rate.
2. Hướng tiếp cận thứ 2 là dựa trên hành vi của users. Rõ ràng, nếu một người mua 1 sản phẩm trên amazon, xem 1 clip trên youtube (có thể là nhiều lần), hay đọc một bài báo, thì có thể khẳng định rằng người dùng đó thích sản phẩm đó. Facebook cũng dựa trên việc bạn like những nội dung nào để hiển thị newsfeed của bạn những nội dung liên quan. Bạn càng đam mê Facebook, fb càng được lợi, thế nên nó luôn mang tới bạn những thông tin mà khả năng cao là bạn muốn đọc. Thường với cách này, ta chỉ xây dựng được một ma trận với các thành phần là 1 và 0, với 1 thể hiện người dùng thích sản phẩm, 0 thể hiện chưa có thông tin. Trong trường hợp này, 0 không có nghĩa thấp hơn

1, nó chỉ có nghĩa là người dùng chưa cấp thông tin. Chúng ta cũng có thể xây dựng ma trận với những giá trị cao hơn 1 thông qua thời gian hoặc số lượt mà người dùng xem 1 sản phẩm nào đó. Đôi khi, nút dislike cũng mang lại những lợi ích nhất định cho hệ thống, lúc này có thể gán giá trị tương ứng bằng -1 chẳng hạn

4. Content-based Recommendations:

1. Item Profiles:

Trong các hệ thống content-based, tức là dựa trên nội dung của mỗi item, chúng ta cần xây dựng một bộ hồ sơ (profile) cho mỗi item. Profile này được biểu diễn dưới dạng toán học là 1 feature vector. Trong những trường hợp đơn giản, feature vector được trực tiếp trích xuất từ item. Ví dụ, xem xét các features của 1 bài hát mà có thể được sử dụng trong các Recommend Systems:

1. Ca sĩ: Cùng là bài "Thành phố buồn" nhưng có người thích bản của Đan Nguyên, có người lại thích bản của Đàm Vĩnh Hưng
2. Nhạc sĩ sáng tác: Cùng là nhạc trẻ nhưng có người thích Phan Mạnh Quỳnh, người khác lại thích Sơn Tùng MTP
3. Năm sáng tác: Một số người thích nhạc xưa hơn nhạc hiện đại
4. Thể loại: Điều này là hiển nhiên rồi, những người khác nhau sẽ có sở thích thể loại nhạc khác nhau

Có rất nhiều yếu tố khác của 1 bài hát có thể sử dụng. Ngoại trừ thể loại khó định nghĩa, các yếu tố khác đều được xác định rõ ràng.

Trong ví dụ ở hình trên, chúng ta đơn giản hoá bài toán bằng việc xây dựng 1 feature vector 2 chiều cho mỗi bài hát: chiều thứ nhất là mức độ Bolero, chiều thứ 2 là mức độ thiếu nhi của bài hát đó. Đặt các feature vector cho

mỗi bài hát là x1, x2, x3, x4, x5. Giả sử các feature vector (ở dạng hàng) cho mỗi bài hát được cho trong hình dưới đây:

	A	B	C	D	E	F	item's feature vectors
Mưa nửa đêm	5	5	0	0	1	?	$\mathbf{x}_1 = [0.99, 0.02]$
Cỏ úa	5	?	?	0	?	?	$\mathbf{x}_2 = [0.91, 0.11]$
Vùng lá me bay	?	4	1	?	?	1	$\mathbf{x}_3 = [0.95, 0.05]$
Con cò bé bé	1	1	4	4	4	?	$\mathbf{x}_4 = [0.01, 0.99]$
Em yêu trường em	1	0	5	?	?	?	$\mathbf{x}_5 = [0.03, 0.98]$
User's models	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	$\leftarrow$ **need to optimize**

Bài toán đi tìm mô hình θ_i cho mỗi user có thể được coi là 1 bài toán hồi quy (Regression) trong trường hợp ratings là 1 dải giá trị, hoặc bài toán gán nhãn (Classification) trong trường hợp ratings là 1 vài trường hợp cụ thể, như like/dislike chẳng hạn. Dữ liệu training để xây dựng mỗi mô hình θ_i là các cặp (item, profile, ratings) tương ứng với các items mà user đó đã rate. Việc điền các giá trị còn thiếu trong ma trận chính là việc dự đoán đầu ra cho các item chưa được rate khi áp dụng mô hình θ_i lên chúng.

Việc lựa chọn mô hình Regression/Classification nào tuỳ thuộc vào ứng dụng

2. Xây dựng hàm mất mát

Giả sử số user là N, số item là M, utility matrix được mô tả bởi ma trận Y. Thành phần ở hàng thứ m, cột thứ n của Y là mức độ quan tâm (rating) của user thứ n lên sản phẩm thứ m mà hệ thống thu thập được. Ma trận Y bị khuyết rất nhiều thành phần tương ứng với giá trị mà hệ thống cần dự đoán. Thêm nữa, gọi R là mà trận 'rated or not' thể hiện việc 1 user đã rate 1 item hay chưa. Cụ thể r_{ij} = 1 nếu item thứ i đã được rate bởi user thứ j, bằng 0 trong trường hợp ngược lại.

Mô hình tuyến tính:

Giả sử rằng ta có thể tìm được 1 mô hình cho mỗi user, minh hoạ bởi vector cột hệ số w_i và bias b_n sao cho mức độ quan tâm của 1 user tới 1 item có thể tính được bằng 1 hàm tuyến tính:

$$y_{mn} = x_m w_n + b_n$$

(Chú ý rằng x_m là 1 vector hàng, w_n là 1 vector cột)

Xét 1 user thứ n bất kỳ, nếu t coi training set là tập hợp các thành phần đã được điền của y_n, ta có thể xây dựng hàm mất mát tương tự như Ridge Regression như sau:

$$\mathcal{L}_n = \frac{1}{2} \sum_{m:r_{mn}=1} (x_m w_n + b_n - y_{mn})^2 + \frac{\lambda}{2} \|w_n\|_2^2$$

Trong đó, thành phần thứ 2 là regularization term và λ là 1 tham số dương. Chú ý rằng regularization thường không được áp dụng lên bias b_n. Trong thực hành, trung bình cộng của lỗi thường được dùng, mất mát $\mathcal{L}_n$ được viết lại thành:

$$\mathcal{L}_n = \frac{1}{2s_n} \sum_{m:r_{mn}=1} (x_m w_n + b_n - y_{mn})^2 + \frac{\lambda}{2s_n} \|w_n\|_2^2$$

Trong đó s_n là số lượng các item mà user thứ n đã rate. Nói cách khác:

$$s_n = \sum_{m=1}^{M} r_{mn}$$

là tổng các phần tử trên cột thứ n của ma trận rated or not R

Vì biểu thức loss function chỉ phụ thuộc vào các item đã được rate, ta có thể rút gọn nó bằng cách đặt $\hat{y}_n$ là sub vector của y được xây dựng bằng cách trích các thành phần khác '?' ở cột thứ n, tức là đã được rate bởi user thứ n trong utility matrix Y. Đồng thời, đặt $\hat{X}_n$ là sub matrix của ma trận feature X, được tạo bằng cách trích các hàng tương ứng với các item đã được rate bởi

user thứ n. Khi đó biểu thức hàm mất mát của mô hình cho user thứ n được viết gọn thành:

$$\mathcal{L}_n = \frac{1}{2s_n} \sum_{m:r_{mn}=1} \left\| \hat{X}_m w_n + b_n e_n - \hat{y}_n \right\|_2^2 + \frac{\lambda}{2s_n} \|w_n\|_2^2$$

Trong đó, $e_n$là vector cột chứa s_n phần tử.

Đây chính xác là hàm mất mát của Ridge Regression. Cặp nghiệm w_n, b_n có thể được tìm qua Stochastic Gradient Descent (SGD), hoặc Mini-batch GD.

5. Collaborative Filtering:

Ở mục trước, ta đã tìm hiểu về 1 hệ thống khuyến nghị dựa trên đặc trưng của mỗi item. Đặc điểm của Content-base Recommendation là việc xây dựng mô hình cho mỗi user khác mà phụ thuộc vào profile của mỗi item. Việc này có lợi thế là tiết kiệm bộ nhớ và thời gian tính toán, đồng thời hệ thống có thể tận dụng các thông tin đặc trưng của item được mô tả trong description của mỗi item. Description có thể được xây dựng bởi nhà cung cấp hoặc được thu thập bằng cách yêu cầu user gắn tag cho item. Việc xây dựng feature vector cho mỗi item thường bao gồm các kỹ thuật Xử lý ngôn ngữ tự nhiên (Natural Language Processing – NLP).

Tuy nhiên, cách làm này có 2 nhược điểm cơ bản. Thứ nhất, khi xây dựng mô hình cho 1 user, các hệ thống Content-base không tận dụng được thông tin từ các users khác. Những thông tin này thường rất hữu ích vì hành vi mua hàng của các user thường được nhóm thành 1 vài nhóm đơn giản, nếu biết hành vi của 1 vài user trong nhóm, hệ thống nên suy luận ra hành vi của những user còn lại. Thứ hai, không phải lúc nào chúng ta cũng có description cho mỗi item. Việc yêu cầu người dùng gắn tag còn khó khăn hơn vì không phải ai cũng sẵn sàng làm việc đó; hoặc có làm nhưng sẽ mang xu hướng cá nhân. Các thuật toán NLP cũng phức tạp hơn ở việc phải xử lý các từ gần nghĩa, viết tắt, sai chính tả, hoặc được viết ở các ngôn ngữ khác nhau.

Những nhược điểm trên sẽ được giải quyết bằng Collaborative Filtering (CF). Có 2 phương pháp CF, phương pháp đầu tiên có tên là Neighborhood-based Collaborative Filtering (NBCF), một phương pháp khác có tên là Matrix Factorization Collaborative Filtering. Khi chỉ nói Collaborative Filtering, chúng ta sẽ ngầm hiểu rằng phương pháp được sử dụng là neighborhood-based.

Ý TƯỞNG:

Ý tưởng cơ bản của CF là xác định mức độ quan tâm của 1 user tới 1 item dựa trên các user khác gần giống user này. Việc gần giống nhau giữa các user có thể được xác định thông qua rating của các user này tới các item khác nhau mà hệ thống đã biết. Ví dụ, A và B đều thích phim "Cảnh sát hình sự", tức đều rate bộ phim này 5 sao. Ta cũng biết A thích "Người phán xử", vậy khả năng cao B cũng thích bộ phim này.

Hai câu hỏi quan trọng nhất trong 1 hệ thống Neighborhood-based Collaborative Filtering là:

- Làm thế nào xác định được sự giống nhau giữa 2 user?
- Khi đã xác định được các user gần giống nhau (similar user) rồi, làm thế nào dự được được mức độ quan tâm của 1 user lên 1 item?

Việc xác định mức độ quan tâm của mỗi user tới 1 item dựa trên mức độ quan tâm của similar user tới item đó còn được gọi là User-user collaborative filtering. Có một hướng tiếp cận khác được cho là hiệu quả hơn là Item-item collaborative filtering. Trong hướng tiếp cận này, thay vì xác định similar user, hệ thống sẽ xác định similar item. Từ đó, hệ thống gợi ý những items gần giống với những item mà user có mức độ quan tâm cao.

1. User-user collaborative filtering:

1.1. Similarity functions

Công việc quan trọng nhất phải làm trước tiên trong User-user collaborative filtering là phải xác định được sự giống nhau (similarity) giữa 2 user. Dữ liệu duy nhất chúng ta có là utility matrix Y, vậy nên sự giống nhau này phải xác định dựa trên các cột tương ứng với 2 user trong ma trận này. Xét ví dụ trong hình sau:

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	5	5	2	0	1	?	?
i_1	3	?	?	0	?	?	?
i_2	?	4	1	?	?	1	2
i_3	2	2	3	4	4	?	4
i_4	2	0	4	?	?	?	5

Giả sử có các user từ u_0 đến u_6 và các item từ i_0 đến i_4, trong đó các số trong mỗi ô vuông thể hiện số sao mà mỗi user đã rate cho item với giá trị càng cao thể hiện mức độ quan tâm cao. Các dấu hỏi chấm là các giá trị mà hệ thống cần phải đi tìm. Đặt mức độ giống nhau của 2 user u_i, u_j là $sim(u_i, u_j)$.

Quan sát qua ta có thể nhận thấy là u_0, u_1 thích i_0, i_1, i_2 và không thích i_3, i_4 cho lắm. Điều ngược lại xảy ra ở các user còn lại. Vì vậy, 1 similarity function tốt cần đảm bảo: $sim(u_0, u_1) > sim(u_0, u_i), \forall i > 1$

Từ đó, để xác định mức độ quan tâm của u_0 lên i_2, chúng ta nên dựa trên hành vi của u_1 lên sản phẩm này. Vì u_1 thích i_2 nên hệ thống cần gợi ý i_2 cho u_0

Câu hỏi đặt ra là: hàm số similarity như thế nào là tốt? Để đo similarity giữa 2 user, cách thường làm là xây dựng feature vector cho mỗi user rồi áp dụng 1 hàm có khả năng đo similarity giữa 2 vector đó. Chú ý rằng việc feature vector này khác với việc xây dựng item profile như trong Content-based System. Các vector này được xây dựng trực tiếp dựa trên utility matrix chứ không dùng dữ liệu ngoài như item profile. Với mỗi user, thông tin duy nhất mà chúng ta biết là các rating mà user đó đã thực hiện, tức cột tương ứng vì mỗi user thường chỉ rate một số lượng rất nhỏ các item. Cách khắc phục là bằng cách nào đó, ta giúp hệ thống điền các giá trị này sao cho việc điền không làm ảnh hưởng nhiều tới sự giống nhau giữa 2 vector. Việc điền này

chỉ phục vụ cho việc tính similarity chứ không phải là suy luận ra giá trị cuối cùng.

Vậy mỗi dấu '?' nên được thay bằng giá trị nào để hạn chế việc sai lệch quá nhiều? Một lựa chọn có thể nghĩ tới là thay các dấu '?' bằng giá trị '0'. Điều này không thực sự tốt vì giá trị '0' tương ứng với mức độ quan tâm thấp nhất. Một giá trị an toàn hơn là 2.5 vì nó là trung bình cộng của 0, mức thấp nhất, và 5, mức cao nhất. Tuy nhiên, giá trị này có hạn chế đối với những user dễ tính hoặc khó tính. Với các user dễ tính, thích tương ứng với 5 sao, không thích có thể ít sao hơn 1 chút, 3 sao chẳng hạn. Việc chọn giá trị 2.5 sẽ khiến cho các item còn lại là negative đối với user đó. Điều ngược lại xảy ra với những user khó tính hơn khi chỉ cho 3 sao cho các item họ thích và ít sao hơn cho những item họ không thích. Để giải quyết vấn đề này, ta có thể chọn 1 giá trị khả dĩ hơn là trung bình cộng các rating mà user tương ứng đã thực hiện. Việc này tránh được việc user quá khó tính hoặc quá dễ tính, tức lúc nào cũng có những item mà 1 user thích hơn so với những item khác.\

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	5	5	2	0	1	?	?
i_1	4	?	?	0	?	2	?
i_2	?	4	1	?	?	1	1
i_3	2	2	3	4	4	?	4
i_4	2	0	4	?	?	?	5
$\bar{u}_j$	3.25	2.75	2.5	1.33	2.5	1.5	3.33

a) Original utility matrix $\mathbf{Y}$ and mean user ratings.

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	1.75	2.25	-0.5	-1.33	-1.5	0	0
i_1	0.75	0	0	-1.33	0	0.5	0
i_2	0	1.25	-1.5	0	0	-0.5	-2.33
i_3	-1.25	-0.75	0.5	2.67	1.5	0	0.67
i_4	-1.25	-2.75	1.5	0	0	0	1.67

b) Normalized utility matrix $\bar{\mathbf{Y}}$.

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
u_0	1	0.83	-0.58	-0.79	-0.82	0.2	-0.38
u_1	0.83	1	-0.87	-0.40	-0.55	-0.23	-0.71
u_2	-0.58	-0.87	1	0.27	0.32	0.47	0.96
u_3	-0.79	-0.40	0.27	1	0.87	-0.29	0.18
u_4	-0.82	-0.55	0.32	0.87	1	0	0.16
u_5	0.2	-0.23	0.47	-0.29	0	1	0.56
u_6	-0.38	-0.71	0.96	0.18	0.16	0.56	1

c) User similarity matrix $\mathbf{S}$.

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	1.75	2.25	-0.5	-1.33	-1.5	0.18	-0.63
i_1	0.75	0.48	-0.17	-1.33	-1.33	0.5	0.05
i_2	0.91	1.25	-1.5	-1.84	-1.78	-0.5	-2.33
i_3	-1.25	-0.75	0.5	2.67	1.5	0.59	0.67
i_4	-1.25	-2.75	1.5	1.57	1.56	1.59	1.67

d) $\hat{\mathbf{Y}}$

Predict normalized rating of u_1 on i_1 with $k = 2$

Users who rated i_1 : $\{u_0, u_3, u_5\}$

Corresponding similarities: {0.83, -0.40, -0.23}

$\Rightarrow$ most similar users: $\mathcal{N}(u_1, i_1) = \{u_0, u_5\}$

with **normalized ratings {0.75, 0.5}**

$$\Rightarrow \hat{y}_{i_1,u_1} = \frac{0.83*0.75+(-0.23)*0.5}{0.83+|-0.23|} \approx 0.48$$

e) Example

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	5	5	2	0	1	1.68	2.70
i_1	4	3.23	2.33	0	1.67	2	3.38
i_2	4.15	4	1	-0.5	0.71	1	1
i_3	2	2	3	4	4	2.10	4
i_4	2	0	4	2.9	4.06	3.10	5

f) Full $\mathbf{Y}$

Ví dụ mô tả User-user CF: a) utility matrix (UM) ban đầu. b) UM đã được chuẩn hoá. c) user similarity matrix. d) Dự đoán các (normalized) rating còn thiếu. e) Vd về cách dự đoán normalized rating của u_1 cho i_1. f) Dự đoán các (denormalized) rating còn thiếu.

Chuẩn hoá dữ liệu:

Hàng cuối cùng trong hình a) là giá trị trung bình của rating cho mỗi user. Giá trị cao tương ứng với các user dễ tính và ngược lại. Khi đó nếu tiếp tục trừ từ mỗi rating đi giá trị này và thay các giá trị chưa biết bằng 0, ta sẽ được ma trận chuẩn hoá như hình b). Vì sao bước chuẩn hoá này lại quan trọng?

- Việc trừ đi trung bình cộng của mỗi cột khiến trong mỗi cột có giá trị dương và âm. Những giá trị dương tương ứng với việc user thích item, những giá trị âm tương ứng với việc user không thích item. Những giá trị bằng 0 tương ứng với việc chưa xác định được liệu user có thích item hay không.
- Về mặt kỹ thuật, số chiều của utility matrix là rất lớn với hàng triệu user và item, nếu lưu toàn bộ giá trị trong 1 ma trận thì khả năng cao sẽ không đủ bộ nhớ. Ta thấy rằng số lượng rating biết trước thường là một số rất nhỏ so với kích thước của utility matrix, sẽ tốt hơn nếu chúng ta lưu ma trận này dưới dạng sparse matrix, tức là chỉ lưu các giá trị khác 0 và vị trí của chúng. Vì vậy, tốt hơn hết, các dấu '?' nên được thay bằng giá trị '0', tức chưa xác định liệu user có thích item hay không. Việc này không những tới ưu bộ nhớ mà việc tính toán similarity matrix sau này cũng hiệu quả hơn.

Sau khi đã chuẩn hoá dữ liệu như trên, một vài similarity function thường được sử dụng là:

Cosine Similarity:

Đây là hàm được sử dụng nhiều nhất, và nó cũng quen thuộc nhất vì nó giống với công thức tính cos của góc giữa 2 vector u_1, u_2.

$$\text{cosine\_similarity}(u_1, u_2) = \cos(u_1, u_2) = \frac{u_1 \cdot u_2}{\|u_1\| . \|u_2\|}$$

Trong đó, $u_{1,2}$ là vector tương ứng với user 1, 2 đã được chuẩn hoá như ở trên. Độ similarity của 2 vector là 1 số trong đoạn [-1, 1]. Giá trị bằng 1 thể hiện 2 vector hoàn toàn giống nhau. Hàm số cos của 1 góc bằng 1 nghĩa là góc giữa 2 vector bằng 0, tức 1 vector bằng tích của 1 số dương với vector còn lại. Giá trị cos = -1 thể hiện 2 vector này hoàn toàn trái ngược nhau. Điều này cũng hợp lý, tức hành vi của 2 user là hoàn toàn ngược nhau thì similarity của 2 vector đó là thấp nhất.

Ví dụ về cosine_similarity của các user trong hình b) được cho trong hình c). Similarity matrix S là 1 ma trận đối xứng vì cos là 1 hàm chẵn, và nếu user A giống user B thì điều ngược lại cũng đúng. Các ô màu xanh trên đường chéo đều bằng 1 vì đó là cos của góc giữa 1 vector và chính nó, tức cos(0) = 1. Khi tính toán ở các bước sau, ta không cần quan tâm tới các giá trị 1 này. Tiếp tục quan sát các vector hàng tương ứng với u_0, u_1, u_2 ta sẽ thấy một vài điều thú vị:

- u_0 gần với u_1 và u_5 (độ giống nhau là dương) hơn các user còn lại. Việc similarity cao giữa u_0 và u_1 là dễ hiểu vì cả 2 đều có xu hướng quan tâm tới i_0, i_1, i_2 hơn cái item còn lại. Việc u_0 gần với u_5 thoạt đầu có vẻ vô lý vì u_5 đánh giá thấp các item mà u_0 đánh giá cao; tuy nhiên khi nhìn vào ma trận đã chuẩn hoá ở hình b), ta thấy rằng điều này là hợp lý. Vì item duy nhất mà cả 2 đã cung cấp thông tin là i_1 với các giá trị tương ứng đều là tích cực.
- u_1 gần với u_0 và xa các user còn lại
- u_2 gần với u_3, u_4, u_5, u_6 và xa các user còn lại

Từ similarity matrix này, ta có thể phân nhóm các user ra làm 2 nhóm (u_0, u_1) và (u_2, u_3, u_4, u_5, u_6). Vì ma trận S này nhỏ nên chúng ta có thể dễ dàng quan sát thấy điều này; khi số user lớn hơn, việc xác định bằng mắt thường là bất khả thi. Việc xây dựng thuật toán phân nhóm user (user clustering) là việc tối quan trọng

Pearson correlation:

Pearson correlation hay còn gọi là hệ số tương quan pearson, dùng để đo lường mức độ tương quan giữa 2 người dùng. Nguyên tắc cơ bản của thuật toán là đo lường sự phụ thuộc tuyến tính giữa 2 biến (hoặc người dùng). Do đó phân tích tương quan Pearson còn được gọi là phân tích hồi quy đơn giản.

Hệ số tương quan Pearson (r) nhận giá trị từ -1 đến +1. Khi r = 0 hoặc gần bằng 0, điều đó có nghĩa là 2 biến không có liên quan gì đến nhau. Khi r > 0 nghĩa là 2 biến tuyến tính dương, nghĩa là khi giá trị của biến này tăng thì giá trị của biến kia cũng tăng. Ngược lại khi r < 0 nghĩa là 2 biến tuyến tính âm, nghĩa là nếu giá trị của biến này tăng thì biến kia sẽ giảm và ngược lại.

Cho 2 biến số x, y từ n mẫu, hệ số tương quan Pearson được ước tính bằng công thức sau:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

1.2. Rating prediction

Việc xác định mức độ quan tâm của 1 user lên 1 item dựa trên các user gần nhất (neighbor user) tương tự phương pháp K-nearest neighbors (KNN). Tương tự như KNN, trong CF, missing rating cũng được xác định dựa trên

thông tin về k neighbor user. Tất nhiên chúng ta chỉ quan tâm tới các user đã rate item đang xét. Predicted rating thường được xác định là trung bình cộng trọng số của các rating đã chuẩn hoá. Có 1 điểm cần lưu ý, trong KNN, các trọng số được xác định dựa trên khoảng cách giữa 2 điểm, và các khoảng cách này là các số không âm. Trong khi đó, trong CF, các trọng số được xác định dựa trên similarity giữa 2 user, những trọng số này có thể nhỏ hơn 0 như trong hình c).

Công thức phổ biến được sử dụng để dự đoán rating của u cho i là:

$$\hat{y}_{i,u} = \frac{\sum_{u_j \in N(u,i)} \overline{y}_{i,u_j} sim(u, u_j)}{\sum_{u_j \in N(u,i)} \left| sim(u, u_j) \right|}$$

(ta sử dụng trị tuyệt đối để xử lý các số âm)

Trong đó $N(u, i)$ là tập hợp k user trong neighborhood (tức là có similarity cao nhất) của u và đã rate i.

Hình d) thể hiện việc điền các giá trị còn thiếu trong ma trận chuẩn hoá. Các ô màu nền đỏ thể hiện các giá trị dương, tức các item mà có thể user đó quan tâm. Ở đây, ta lấy ngưỡng bằng 0, ta cũng có thể hoàn toàn chọn các ngưỡng khác 0.

Một ví dụ về việc tính normalize rating của u_1 cho i_1 được cho trong hình e) với số nearest neighbors là k = 2. Các bước thực hiện như sau:

- Xác định các user đã rate i_1, đó là u_0, u_3, u_5.
- Xác định similarity của u_1 và các user này, ta nhận được 0.83, -0.40, -0.23. Hai (k=2) giá trị lớn nhất là 0.83 và -0.23 tương ứng với u_0 và u_5
- Xác định các normalized rating của u_0 và u_5 cho i_1, ta thu được 2 giá trị lần lượt là 0.75 và 0.5
- Dự đoán kết quả:

$$\hat{y}_{i_1,u_1} = \frac{0.83 \times 0.75 + (-0.23) \times 0.5}{|0.83| + |-0.23|} \approx 0.48$$

Việc hệ thống quyết định gợi ý item nào cho mỗi user có thể được xác định bằng nhiều cách khác nhau. Có thể sắp xếp các unrate item theo thứ tự từ lớn đến bé của các predicted rating, hoặc hỉ chọn các item có normalized predicted rating dương – tương ứng với việc user này có nhiều khả năng thích hơn.

2. Item-item collaborative filtering:

Một số hạn chế của User-user CF:

- Trên thực tế, số lượng user luôn lớn hơn số lượng item rất nhiều. Kéo theo đó là similarity matrix là rất lớn với số phần tử phải lưu giữ là hơn 1 nửa của bình phương số lượng user (chú ý rằng ma trận này là đối xứng). Việc này, như đã đề cập ở trên, khiến cho việc lưu trữ ma trận này trong nhiều trường hợp không khả thi.
- Ma trận utility Y thường là rất trống. Với số lượng rất lớn user so với số lượng item, rất nhiều cột của ma trận này sẽ nhiều ô trống, tức là chỉ có vài phần tử khác 0. Lý do đã đề cập là do user thường lười rate. Cũng chính vì việc này, một khi user đó thay đổi rating hoặc rate thêm item, trung bình cộng các rating cũng như vector chuẩn hoá tương ứng với user này thay đổi nhiều. Kéo theo đó, việc tính toán ma trận similarity, vốn tồn nhiều bộ nhớ và thời gian, cũng cần được thực hiện lại.

Ngược lại, nếu ta tính toán similarity giữa các item rồi recommend những item gần giống với item yêu thích của 1 user thì sẽ có những lợi ích sau:

- Vì số lượng item thường nhỏ hơn lượng user, similarity matrix trong trường hợp này cũng nhỏ hơn nhiều, thuận lợi cho việc lưu trữ và tính toán ở các bước sau

- Vì số lượng phần tử đã biết trong utility matrix là như nhau nhưng số hàng (item) ít hơn số cột (user) nên trung bình mỗi hàng của ma trận này sẽ có nhiều phần tử đã biết hơn số phần tử đã biết trong mỗi cột. Việc này cũng dễ hiểu vì mỗi item có thể được rate bởi nhiều user. Kéo theo đó, giá trị trung bình của mỗi hàng ít bị thay đổi hơn khi có thêm một vài rating. Như vậy, việc cập nhật ma trận similarity có thể được thực hiện ít thường xuyên hơn.

Cách tiếp cận thứ hai này được gọi là Item-item Collaborative Filtering. Hướng tiếp cận này cũng được sử dụng nhiều hơn trong thực tế.

Quy trình dự đoán missing rating cũng tương tự như trong User-user CF. Hình dưới đây mô tả quy trình này với ví dụ ở mục trước:

	u_0	u_1	u_2	u_3	u_4	u_5	u_6		
i_0	5	5	2	0	1	?	?	→	2.6
i_1	4	?	?	0	?	2	?	→	2
i_2	?	4	1	?	?	1	1	→	1.75
i_3	2	2	3	4	4	?	4	→	3.17
i_4	2	0	4	?	?	?	5	→	2.75

a) Original utility matrix $\mathbf{Y}$ and mean item ratings.

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	2.4	2.4	-.6	-2.6	-1.6	0	0
i_1	2	0	0	-2	0	0	0
i_2	0	2.25	-0.75	0	0	-0.75	-0.75
i_3	-1.17	-1.17	-0.17	0.83	0.83	0	0.83
i_4	-0.75	-2.75	1.25	0	0	0	2.25

b) Normalized utility matrix $\bar{\mathbf{Y}}$.

	i_0	i_1	i_2	i_3	i_4
i_0	1	0.77	0.49	-0.89	-0.52
i_1	0.77	1	0	-0.64	-0.14
i_2	0.49	0	1	-0.55	-0.88
i_3	-0.89	-0.64	-0.55	1	0.68
i_4	-0.52	-0.14	-0.88	0.68	1

c) Item similarity matrix $\mathbf{S}$.

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	2.4	2.4	-.6	-2.6	-1.6	-0.29	-1.52
i_1	2	2.4	-0.6	-2	-1.25	0	-2.25
i_2	2.4	2.25	-0.75	-2.6	-1.20	-0.75	-0.75
i_3	-1.17	-1.17	-0.17	0.83	0.83	0.34	0.83
i_4	-0.75	-2.75	1.25	1.03	1.16	0.65	2.25

d) Normalized utility matrix $\bar{\mathbf{Y}}$.

Ví dụ mô tả Item-item CF: a) UM ban đầu. b) UM đã được chuẩn hoá. c) User similarity matrix. d) Dự đoán các (normalized) rating còn thiếu.

Có 1 điểm thú vị trong similarity matrix ở hình c) là có các phần tử trong 2 hình ô vuông xanh và đỏ đều là các số không âm, các phần tử ngoài là các số âm. Việc này thể hiện rằng các items có thể được chia thành 2 nhóm rõ rệt với những item có similarity không âm vào 1 nhóm. Như vậy, 1 cách vô tình, chúng ta đã thực hiện việc item clusterig. Việc này sẽ giúp ích rất nhiều trong việc dự đoán.

Kết quả về việc chọn item nào để recommend cho mỗi user được thể hiện bởi các ô màu đỏ trong hình d). Kết quả này có khác 1 chút so với kết quả tìm được bởi User-user CF ở 2 cột cuối cùng tương ứng với u_5, u_6. Dường như kết quả này hợp lý hơn vì từ UM, có 2 nhóm user thích 2 nhóm item khác nhau.

*** Về mặt tính toán, Item-item CF có thể nhận được từ User-user CF bằng cách chuyển vị ma trận utility, và coi như items đang rate user. Sau khi tính ra kết quả cuối cùng, ta lại chuyển vị 1 lần nữa để thu được kết quả*

5*. Matrix Factorization Collaborative Filtering:

Ở mục 5, ta đã làm quen với 1 hướng tiếp cận trong CF dựa trên hành vi của user hoặc item lân cận. Tiếp theo, ta sẽ làm quen với hướng tiếp cận khác cho CF dựa trên Matrix Factorization (hoặc Matrix Decomposition), tức Phân tích ma trận thành nhân tử.

Trong Content-based RS, mỗi item được mô tả bằng 1 vector x được gọi là item profile. Trong phương pháp này, ta cần tìm 1 vector hệ số w tương ứng với mỗi user sao cho rating đã biết mà user dó cho item xấp xỉ: $y \approx xw$

Với cách làm trên, Utility Matrix Y, giả sử đã được điền hết, sẽ xấp xỉ với:

$$Y \approx \begin{bmatrix} x_1w_1 & x_1w_2 & \cdots & x_1w_N \\ x_2w_1 & x_2w_2 & & x_2w_N \\ \cdots & \cdots & \ddots & \vdots \\ x_Mw_1 & x_Mw_2 & \cdots & x_Mw_N \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_M \end{bmatrix} [w_1 \quad w_2 \quad \cdots \quad w_N] = XW$$

với M, N lần lượt là số item và số user.

Chú ý rằng, x được xây dựng dựa trên thông tin mô tả của item và quá trình xây dựng này độc lập với quá trình đi tìm hệ số phù hợp cho mỗi user. Như vậy, việc xây dựng item profile đóng vai trò rất quan trọng và có ảnh hưởng trực tiếp lên hiệu năng của mô hình. Thêm nữa, việc xây dựng từng mô hình riêng lẻ cho mỗi user dẫn đến kết quả chưa thực sự tốt vì không khai thác được đặc điểm của những user gần giống nhau.

PHẦN 3: CÀI ĐẶT THUẬT TOÁN

1. Tìm kiếm tập dữ liệu:

Để có thể thực hiện cài đặt thuật toán, sự cần thiết của tập dữ liệu (dataset) là tối quan trọng. Hiện nay có rất nhiều trang web chia sẻ tài nguyên, do đó việc tìm kiếm cho mình 1 tập dataset không phải là quá khó. Tuy nhiên, để tìm được 1 tập dataset có đầy đủ thông tin cần thiết cho kế hoạch phát triển ứng dụng thì không phải là đơn giản.

2. Colaborative Filtering:

1. Tạo class CF:

- Khởi tạo class:

  Dữ liệu đầu vào cho hàm khởi tạo của class CF là ma trận utility Y_data được lưu dưới dạng ma trận 3 cột (user_id, movie_id, rating), k là số lượng các điểm lân cận được sử dụng để dự đoán kết quả. distance_func là hàm đo mức độ similarity giữa 2 vector, mặc định sẽ là cosine_similarity như đã giới thiệu ở phần trước.

```
class CF(object):
    def __init__(self, Y_data, k, dist_func = cosine_similarity):
        self.Y_data = Y_data
        self.k = k
        self.dist_func = dist_func
        self.Ybar_data = None

        # number of users and items. plus 1 cuz id start with 0
        self.n_users = int(np.max(self.Y_data[:, 0])) + 1
        self.n_items = int(np.max(self.Y_data[:, 1])) + 1
```

- Tính toán normalized UM và Similarity matrix:

```
def normalize_Y(self):
    users = self.Y_data[:, 0] # first col of matrix
    self.Ybar_data = self.Y_data.copy()
    self.mu = np.zeros((self.n_users,))

    for n in range(self.n_users):
        ids = np.where(users == n)[0].astype(np.int32)

        # get all item which user with ids already rated and rating values.
        item_ids = self.Y_data[ids, 1]
        ratings = self.Y_data[ids, 2]

        # take mean
        m = np.mean(ratings)
        if np.isnan(m):
            m = 0 # to avoid empty array an non value

        # store average rating of each user
        self.mu[n] = m

        # normalize
        self.Ybar_data[ids, 2] = ratings - self.mu[n]

    self.Ybar = sparse.coo_matrix((self.Ybar_data[:, 2],
                    (self.Ybar_data[:, 1], self.Ybar_data[:, 0])), (self.n_items, self.n_users))
    self.Ybar = self.Ybar.tocsr()

def similarity(self):
    self.S = self.dist_func(self.Ybar.T, self.Ybar.T)
```

- Thực hiện lại 2 hàm trên khi có thêm dữ liệu mới:

```
def refresh(self):
    """
    Normalize data and calculate similarity matrix again (after
    some few ratings added)
    """
    self.normalize_Y()
    self.similarity()

def fit(self):
    self.refresh()
```

- Dự đoán kết quả:

  Hàm __pred là hàm dự đoán item i cho user u. Để cho API được đơn giản, ta để hàm __pred là 1 hàm private, chỉ có thể truy cập được trong class CF; hàm pred sẽ là 1 hàm public, thứ tự tham số sẽ là (user, item).

```python
def __pred(self, u, i, normalized = 1):
    # find all user whos rated i
    ids = np.where(self.Y_data[:, 1] == i)[0].astype(np.int32)

    users_rated_i = (self.Y_data[ids, 0]).astype(np.int32)

    # find similarity btw the current user and others who already rated i
    sim = self.S[u, users_rated_i]

    # find the k most similar users
    a = np.argsort(sim)[-self.k:]

    # and the corresponding similarity lvl
    nearest_s = sim[a]

    r = self.Ybar[i, users_rated_i[a]]

    if normalized:
        # add a small number, for instance, 1e-8, to avoid dividing by 0
        return (r*nearest_s)[0]/(np.abs(nearest_s).sum() + 1e-8)

    return (r*nearest_s)[0]/(np.abs(nearest_s).sum() + 1e-8) + self.mu[u]

def pred(self, u, i, normalized = 1):
    """
    predict the rating of user u for item i (normalized)
    if you need the un
    """
    return self.__pred(u, i, normalized)
```

- Recommend: Tìm tất cả các item nên được gợi ý cho user u, sắp xếp theo thứ tự điểm rating predict từ cao đến thấp

```
def recommend(self, u, normalized = 1):
    ids = np.where(self.Y_data[:, 0] == u)[0]

    item_rated_by_u = self.Y_data[ids, 1].tolist()

    recommend_items = []

    for i in range(self.n_items):
        if i not in item_rated_by_u:
            rating = self.__pred(u, i)

            if rating > 0:
                recommend_items.append((i, rating))

    # sort list recommend from highest predicted rating to lowest
    recommend_items.sort(key=lambda tup: tup[1], reverse=True)

    return recommend_items
```

2. Đánh giá thuật toán:

Để có thể đánh giá độ chính xác của thuật toán, ta sử dụng RMSE (root mean squared error) để tính toán sai số toàn phương trung bình của thuật toán.

RMSE: tức sai số toàn phương trung bình, là trung bình của bình phương các sai số, tức là sự khác biệt giữa các ước lượng và những gì được đánh giá. RMSE là 1 hàm rủi ro, tương ứng với giá trị kỳ vọng của sự mất mát sai số bình phương. Sự khác biệt xảy ra là do ngẫu nhiên, hoặc vì các ước lượng không tính đến thông tin có thể cho ra 1 ước tính chính xác hơn.

MSE là moment bậc 2 (về nguồn gốc) của sai số, và do đó kết hợp cả 2 phương sai của ước lượng thiên vị của nó. Đối với 1 ước lượng không có thiên vị, MSE là phương sai của ước lượng. Cũng giống như các phương sai, MSE có cùng một đơn vị đo lường theo bình phương của số lượng được ước tính. Đối với đại lượng không có thiên vị, các RMSE là căn bậc 2 của phương sai, được gọi là độ lệch chuẩn.

Công thức tính RMSE:

Nếu $\hat{Y}$ là một vector của n trị dự báo, và Y là vector các trị quan sát được, tương ứng với ngõ vào của hàm số phát ra dự đoán, thì RMSE của phép dự đoán có thể được tính theo công thức:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}$$

Để có cái nhìn tổng quan về, ta tính toán thời gian thực thi của thuật toán để cho ra dự đoán và sai số RMSE trên 10 tập test với số lượng records khác nhau. Từ đó ta có bảng giá trị như hình bên dưới:

STT	Số lượng records	Thời gian chạy (s)	RMSE	Image
1	90570	0.44	0.73	Nums of records: 90570 Run time: 0.42991042137145996 RMSE = 0.7314707613553664
2	9430	0.32	0.51	Nums of records: 9430 Run time: 0.3192570209503174 RMSE = 0.5134675485682235
3	9430	0.32	0.52	Nums of records: 9430 Run time: 0.32213854789733887 RMSE = 0.5164081898054714
4	90570	0.44	0.73	Nums of records: 90570 Run time: 0.4447762966156006 RMSE = 0.7287720142970381
5	20000	0.35	0.61	Nums of records: 20000 Run time: 0.34912586212158203 RMSE = 0.6054163264669068
6	80000	0.51	0.71	Nums of records: 80000 Run time: 0.5115344524383545 RMSE = 0.713213562319097
7	20000	0.32	0.58	Nums of records: 20000 Run time: 0.32114386558532715 RMSE = 0.5782683353056038
8	80000	0.42	0.71	Nums of records: 80000 Run time: 0.42113184928894043 RMSE = 0.7100469487635904
9	20000	0.32	0.56	Nums of records: 20000 Run time: 0.3279609680175781 RMSE = 0.5634062622838398
10	80000	0.49	0.71	Nums of records: 80000 Run time: 0.4855682849884033 RMSE = 0.7139772938251607

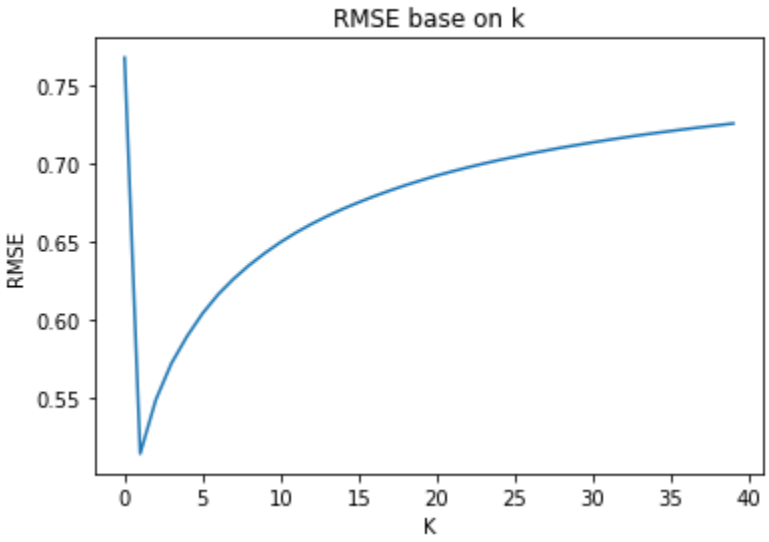
Từ bảng so sánh, ta có thể thấy giá trị sai số trung bình của thuật toán rơi vào khoảng 0.6. Đây là một con số không quá tệ cho 1 thuật toán bình thường, chưa áp dụng máy học.

3. Chọn giá trị k nearest:

Ở mục 1, khi giới thiệu về class CF, ta thấy ở tham số đầu vào cho class CF có nhận giá trị k (k nearest neighbor). Vậy làm sao để chọn giá trị k cho bài toán là một việc hết sức khó khăn.

Để chọn giá trị cho biến k, ta nhìn vào biểu đồ sự thay đổi giá trị RMSE dựa vào k như hình bên dưới:

```
Nums of records:  80000
```



Ta có thể thấy, RMSE đạt giá trị nhỏ nhất khi k = 1 và tăng dần khi giá trị k tăng. Khi k đạt ngưỡng 35 thì RMSE cũng tăng chậm lại và gần như là không khác biệt quá lớn. Tuy nhiên liệu chọn k = 1 có phải là lựa chọn đúng đắn.

Bởi vì hướng tiếp cận đang sử dụng là Colaborative Filtering, tức là sẽ đưa ra khuyến nghị nhờ vào các user có thể tương tự, do đó nếu ta chọn k quá nhỏ (k = 1) thì số lượng item được gợi ý sẽ là rất nhỏ và có thể không phù hợp với user đó. Do đó đối với bài toán đang giải quyết, em quyết định chọn k = 20, mặc dù sai số sẽ lớn hơn một chút, nhưng lượng user được đánh giá sẽ đủ lớn và từ đó sẽ cover được số lượng item lớn hơn => cho ra hiệu quả khuyến nghị tốt hơn.

3. Contentbase Filtering:

Về hướng tiếp cận, ta đã được đề cập đến ở mục 2. Trong mục này, ta sẽ tìm hiểu về ứng dụng của hướng tiếp cận này trong bài toán cụ thể: Hệ thống khuyến nghị phim điện ảnh.

Trong bài toán cụ thể này, em sẽ sử dụng ContentBase Filtering để từ 1 bộ phim người dùng đang quan tâm, chúng ta sẽ tìm được các bộ phim tương tự mà người dùng cũng có thể quan tâm. Trước khi đi vào chi tiết cài đặt thuật toán, ta cần tìm hiểu về 1 kỹ thuật mà ta áp dụng vào thuật toán, đó là Count Vectorizer.

1. Count Vectorizer:

Như đã đề cập, Contentbase Filtering đưa ra dự đoán dựa trên nội dung (content) của đối tượng mà ta muốn dự đoán mà không cần quan tâm đến thông tin của bất kể người dùng nào khác.

Do đó, để có thể chuyển đổi mỗi item, trong trường hợp này là mỗi bộ phim thành 1 vector thuộc tính, chứa các thuộc tính nổi bật của bộ phim, trong bài toán này chúng ta chọn thể loại, diễn viên, từ khoá và đạo diễn, người ta cần áp dụng kỹ thuật Count Vectorizer để đếm số lượng từ xuất hiện trong 1 mảng các thuộc tính khác.

```
from sklearn.feature_extraction.text import CountVectorizer
feature_vector = [
  'This is the first document.',
  'This document is the second document.',
  'And this is the third one.',
  'Is this the first document?',
]
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(feature_vector)
feature_names = vectorizer.get_feature_names()

print(feature_names)
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']

print(X.toarray())
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]
```

Từ ví dụ trong hình, ta có thể thấy lớp CountVectorizer đã đếm số lần xuất hiện của mỗi từ trong mảng feature_names. Nhìn vào mảng đầu tiên của ma trận X, ta nhận thấy mảng [0 1 1 1 0 0 1 0 1] tương ứng với câu This is the first document. Theo thứ mỗi phần tử ở mảng feature_names, ta được số lần xuất hiện của chúng là:

- and: 0
- document: 1
- first: 1
- is: 1
- one: 0
- second: 0
- the: 1
- third: 0
- this: 1

Mảng feature_names thường được gọi là 1 "túi từ vựng". Bằng cách đếm số lần xuất hiện của mỗi từ (lấy từ túi từ vựng) trên 1 thuộc tính của item, Count Vectorizer đã biến chúng thành ma trận các vector thuộc tính, được sử dụng để tính độ tương quan giữa các vector sau này, từ đó phán đoán các bộ phim và người dùng có thể thích.

2. So sánh với TFIDF:

Có nhiều cách để chuyển đổi các thông tin mà ta cần thành 1 ma trận thuộc tính như Count Vectorizer (đã đề cập ở trên), TF-IDF, Word Embeddings, Text/ NLP, ... Tuy nhiên có 2 cách làm khác đơn giản và phổ biến khi tiếp cận theo hướng Contentbase Filtering, đó là Count Vectorizer và TF-IDF. Vậy tại em lại chọn giải quyết bài toán bằng Count Vectorizer, ta hãy cùng xem một vài so sánh dưới đây để làm rõ vấn đề.

1. Count Vectorizer chỉ đếm tần số từ: như đã giới thiệu trong mục trước, ta đã biết rằng Count Vectorizer chỉ đơn giản đếm tần số xuất hiện của 1 từ trong túi từ vựng trong vector.
2. TF -IDF: viết tắt của thuật ngữ term frequency – inverse document frequency, nó có nghĩa là trọng số của 1 từ trong văn bản thu được qua thống kê thể hiện mức độ quan trọng của từ này trong 1 văn bản, mà bản thân văn bản đang xét nằm trong 1 hợp các văn bản. Thuật toán này được sử dụng bởi vì trong ngôn ngữ luôn có các từ ngữ được xuất hiện thường xuyên hơn các từ khác.

   TF: Tần suất xuất hiện của 1 từ trong văn bản. Vì các văn bản có độ dài ngắn khác nhau nên một số từ có thể xuất hiện nhiều lần trong một văn bản dài hơn là một văn bản ngắn. Như vậy, term frequency thường được chia cho độ dài văn bản(tổng số từ trong một văn bản):

$$tf(t,d) = \frac{f(t,d)}{\max\{f(w,d) : w \in d\}}$$

   Trong đó:

- tf(t, d): tần suất xuất hiện của từ t trong văn bản d
- f(t, d): Số lần xuất hiện của từ t trong văn bản d
- max({f(w, d) : w ∈ d}): Số lần xuất hiện của từ có số lần xuất hiện nhiều nhất trong văn bản d

IDF: nghịch đảo tần suất của văn bản, giúp đánh giá tầm quan trọng của 1 từ. Khi tính toán TF, tất cả các từ được coi như có độ quan trọng là ngang nhau; tuy nhiên, một số từ như "is", "of" và "that" thường xuất hiện rất nhiều lần nhưng độ quan trọng là không cao. Như thế chúng ta cần giảm độ quan trọng của những từ này xuống:

$$idf(t, D) = log \frac{|D|}{|\{d \in D : t \in d\}|}$$

Trong đó:

- idf(t, D): giá trị idf của từ t trong tập văn bản
- |D|: Tổng số văn bản trong tập D
- |{d ∈ D : t ∈ d}|: thể hiện số văn bản trong tập D có chứa từ t

Từ đó, ta có công thức tính TF-IDF hoàn chỉnh như sau:

$$TF - IDF(t, d, D) = tf(t, d) \times idf(t, D)$$

Khi đó, những từ có giá trị TF-IDF cao là những từ xuất hiện nhiều trong văn bản này, và xuất hiện ít trong các văn bản khác. Việc này giúp lọc ra những từ phổ biến và giữ lại những từ có giá trị cao (từ khoá của văn bản đó)

Từ 1 vài điều cơ bản của 2 thuật toán này, ta chọn Count Vectorizer cho bài toán này. Bởi vì khi vector hoá các thuộc tính của item, mà cụ thể trong bài toán này là tên thể loại, tên diễn viên, tên đạo diễn, tên từ khoá; trừ 2 thuộc tính có thể có những stopword là tên phim và tên từ khoá, thì 2 thuộc tính còn lại đều chứa tên riêng của người. Từ đó, ta không muốn giảm đi sự quan trọng của diễn viên hay đạo diễn nào đó chỉ vì họ diễn trong nhiều bộ phim

khác nhau, điều này làm cho thuật toán có vẻ trở nên vô lý. Đó chính là lý do lớn nhất để ta chọn Count Vectorizer chứ không phải TF-IDF, mặc dù TF-IDF phổ biến và đánh giá tốt hơn.

3. Cài đặt thuật toán:

Để giảm độ phức tạp cho thuật toán, ta chỉ chọn ra 3 diễn viên có thuộc tính order nhỏ nhất (thuộc tính order đánh giá mức độ quan trọng của diễn viên đối với bộ phim), tức là 3 diễn viên chính của bộ phim, tên đạo diễn, thể loại và từ khoá

Do đó, từ mỗi thuộc tính của bộ phim, ta phải trích xuất ra và chỉ lấy 3 giá trị cần tính. Sau khi đã có được các thuộc tính như mong muốn từ mọi bộ phim có trong cơ sở dữ liệu và đọc nó bằng thư viện pandas; bước tiếp theo ta cần chuyển các dữ liệu đó về dạng chữ viết thường và bỏ hết khoảng trắng giữa các ký tự. Điều này là cần thiết để cho Count Vectorizer không tính mọi Johnny (Johnny Deep và Johnny Galecki) là giống nhau.

```
def clean_data(x):
    if isinstance(x, list):
        return [str.lower(i.replace(" ", "")) for i in x]
    else:
        #Check if director exists. If not, return empty string
        if isinstance(x, str):
            return str.lower(x.replace(" ", ""))
        else:
            return ''
```

Sau khi khai báo xong hàm, ta cần áp dụng đó cho mọi thuộc tính của bảng

```
# Declear which column applied clean_data
features = ['cast', 'keywords', 'director', 'genres']

for feat in features:      You, 3 days ago · similar movie
    movies[feat] = movies[feat].apply(clean_data)
```

Ta cần phải tạo ra 1 “soup” dữ liệu, 1 chuỗi ký tự chứa toàn bộ thông tin mà ta muốn đưa vào túi từ vựng

```
# Create our "metadata soup", a string that contain all the metadata to feed to vectorizer
def create_soup(x):
    return x['keywords'] + ' ' + x['cast'] + ' ' + x['director'] + ' ' + x['genres']

movies['soup'] = movies.apply(create_soup, axis=1)
```

```
count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(movies['soup'])      You, 3 days ago • simil
count_matrix = count_matrix.astype(np.float32)

# compute the cosine similarity matrix based on count_matrix
cosine_sim = cosine_similarity(count_matrix, count_matrix)
```

Sau khi đã có túi từ vựng, ta tính toán ma trận tương đồng giữa các vector dựa trên count_matrix. Ta đã có đủ những công cụ cần thiết để có thể cài đặt thuật toán khuyến nghị của ta. Ta sẽ đi theo những bước sau:

- Lấy giá trị index của bộ phim muốn được dự đoán
- Lấy danh sách điểm tương đồng của bộ phim đó với các bộ phim khác có trong cơ sở dữ liệu. Chuyển đổi nó thành 1 list tuples, với phần tử đầu tiên là vị trí của nó và phần tử thứ 2 là điểm tương đồng
- Sắp xếp danh sách mới tìm được ở trên dựa vào điểm tương đồng
- Lấy 10 phần tử đầu tiên của mảng sau khi được sắp xếp
- Trả về danh sách id của các phần tử trên

```python
# Function that takes in movie title as input and output most similar movies
def get_recommendations(title, cosine_sim=cosine_sim):
    # get index of movie that matches title
    idx = indices[title]

    # get the pairwise similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # get score of the 10 most similar movies
    sim_scores = sim_scores[1:6]

    # get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    return movies[['id','title']].iloc[movie_indices]
```

4. Demongraphic Filtering:

Để có thể thực hiện hướng tiếp cận này, ta cần một vài thông số sau:

- Số liệu về điểm hoặc rating của phim
- Chấm điểm cho mọi bộ phim
- Sắp xếp điểm và chọn ra những bộ phim tốt nhất

Ta có thể đơn giản lấy giá trị rating trung bình của 1 bộ phim để sắp xếp, tuy nhiên điều này không đủ công bằng cho mọi phim, bởi 1 phim có rating trung bình 4.9 được đánh giá bởi 1 người chưa chắc đã hay hơn 1 bộ phim có rating trung bình là 4.0 nhưng được đánh giá bởi 100 người. Do đó, ta sử dụng công thức đánh giá điểm của IMDB:

$$Weighted\ Rating\ (WR) = \left(\frac{v}{v+m}.R\right) + \left(\frac{m}{v+m}\right).C$$

Trong đó:

- v: tổng số lượng vote của bộ phim đó
- m: là số lượng votes tối thiểu để được đưa vào đánh giá
- R: điểm rating trung bình của bộ phim đó
- C: điểm rating trung bình của toàn bộ danh sách

Ta đã có sẵn 2 thông số là v(vote_count) và R(vote_average), C có thể được tính bởi công thức sau:

```
        C= df2['vote_average'].mean()
```

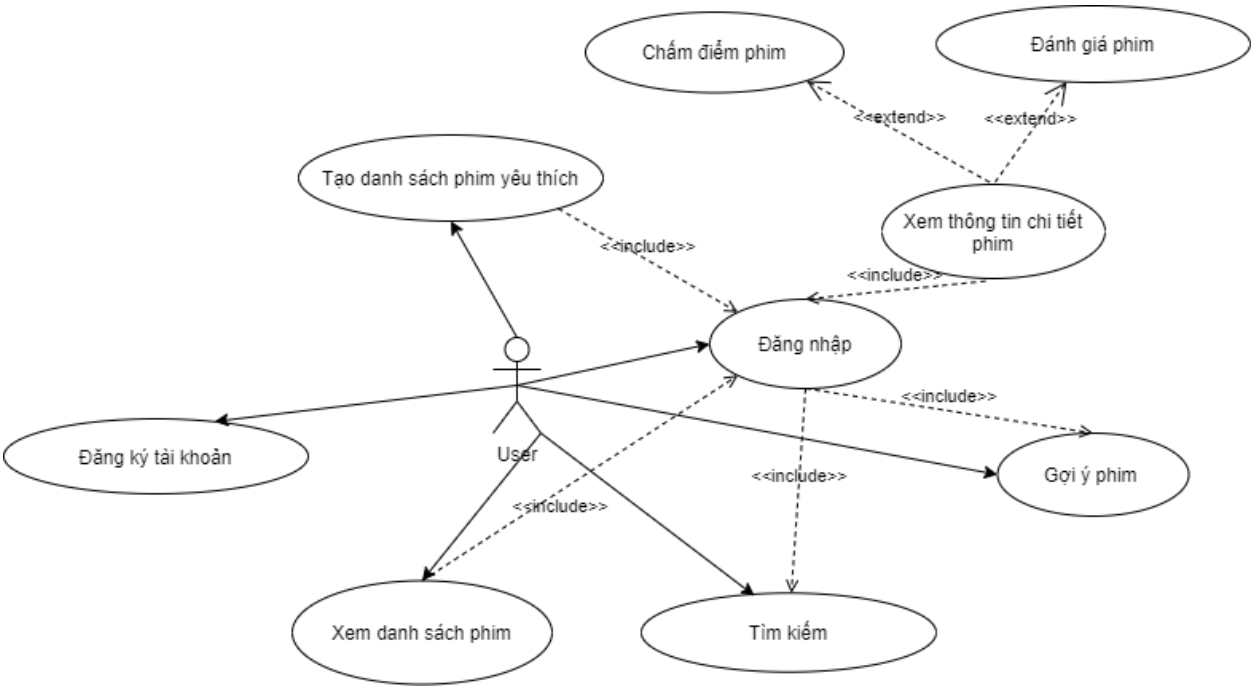
Cuối cùng, ta cần ước lượng giá trị m, số lượng vote trung bình để có thể được đưa vào danh sách đánh giá. Ở đây, ta sẽ lấy mốc 95% là mốc cutoff. Nói cách khác, 1 bộ phim để có thể được đưa vào danh sách đánh giá thì nó cần phải có số lượt đánh giá lớn hơn 95% bộ phim có trong CSDL.

```
        m= df2['vote_count'].quantile(0.95)
```

Sau khi có đầy đủ thông số, ta chỉ cần áp vào công thức và sắp xếp các bộ phim từ kết quả đã tính được.

PHẦN 4: THIẾT KẾ HỆ THỐNG

1. Sơ đồ Use case:



- Danh sách các tác nhân của hệ thống

STT	Tác nhân của hệ thống	Ý nghĩa
1	User	Người dùng của hệ thống,

- Danh sách use case:

STT	Usecase	Ý nghĩa
1	Đăng nhập	Người dùng đăng nhập vào hệ thống
2	Đăng ký	Người dùng đăng ký tài khoản cá nhân nhân để có thể đăng nhập vào hệ thống
3	Tìm kiếm	Người dùng nhập vào từ khoá để tìm kiếm phim hoặc diễn viên
4	Gợi ý phim	Người dùng khi tương tác với hệ thống sẽ nhận được danh sách gợi ý phim của hệ thống
5	Xem danh sách phim	Người dùng có thể được xem danh sách phim theo thể loại, các bộ phim được chấm điểm cao nhất hoặc đang thịnh hành
6	Xem thông chi tiết phim	Người dùng được xem mọi thông tin chi tiết của bộ phim lựa chọn
7	Chấm điểm phim	Người dùng được chấm điểm bộ phim bất kỳ
8	Đánh giá phim	Người dùng được để lại đánh giá của mình với bộ phim bất kỳ
9	Tạo danh sách yêu thích	Người dùng có thể tạo danh sách phim yêu thích để có thể truy cập vào bộ phim đó nhanh hơn

2. Đặc tả use case:

1. Use case đăng ký tài khoản

Tóm tắt	Mô tả cách người dùng đăng kí tài khoản thành viên
Tác nhân	User

Điều kiện tiên quyết	Không có
Hậu điều kiện	Nếu Use-case thành công, người dùng được đăng nhập trực tiếp vào hệ thóng
Dòng sự kiện chính	1. Người dùng truy cập vào trang web ký tài khoản 2. Hệ thống hiển thị form đăng ký 3. Người dùng nhập thông tin vào form đăng ký 4. Người dùng nhấn nút xác nhận đăng ký 5. Hệ thống xác thực thông tin đăng ký và đăng nhập vào hệ thống
Dòng sự kiện thay thế	
Dòng sự kiện ngoại lệ	5a. Hệ thống xác thực thông tin đăng ký không thành công và hiển thị thông báo 5a1. Người dùng sửa đổi lại thông tin cho hợp lệ Use Case tiếp tụ Use Case UC5 5a2. Người dùng thoát khỏi hệ thống Use Case dừng lại
Yêu cầu phi chức năng	

2. Use case đăng nhập

Tóm tắt	Mô tả cách người dùng đăng nhập tài khoản thành viên
Tác nhân	User

Điều kiện tiên quyết	Người dùng phải có tài khoản đăng nhập
Hậu điều kiện	Nếu Use-case thành công, người dùng có thể truy cập vào hệ thống
Dòng sự kiện chính	1. Người dùng truy cập vào trang web đăng nhập 2. Hệ thống hiển thị form đăng nhập 3. Người dùng nhập thông tin vào form đăng nhập 4. Người dùng nhấn nút xác nhận đăng nhập 5. Hệ thống xác thực thông tin đăng nhập thành công và chuyển tiếp người dùng vào trang web
Dòng sự kiện thay thế	
Dòng sự kiện ngoại lệ	5a. Hệ thống xác thực thông tin đăng nhập không thành công và hiển thị thông báo 5a1. Người dùng sửa đổi lại thông tin cho hợp lệ *Use Case tiếp tụ Use Case UC4* 5a2. Người dùng thoát khỏi hệ thống *Use Case dừng lại*
Yêu cầu phi chức năng	

3. Use case đăng nhập

Tóm tắt	Mô tả cách người dùng tìm kiếm phim trên hệ thống (theo tên)
Tác nhân	User
Điều kiện tiên quyết	Người dùng đã đăng nhập vào hệ thống
Hậu điều kiện	Nếu Use-case thành công, hệ thống sẽ trả về danh sách các bộ phim trùng khớp với kết quả tìm kiếm
Dòng sự kiện chính	1. Người dùng truy cập vào ô tìm kiếm 2. Người dùng nhập từ khóa tìm kiếm liên quan đế thông tin phim (tiêu đề, diễn viên…) 3. Hệ thống nhận từ khóa và trả về danh sách các phim trùng khớp hoặc gần giống với từ khóa tìm kiếm từ người dùng.
Dòng sự kiện thay thế	
Dòng sự kiện ngoại lệ	
Yêu cầu phi chức năng	

4. Use case đánh giá phim

Tóm tắt	Mô tả cách người dùng đánh giá 1 bộ phim trên hệ thống, mục đích để hệ thống quyết định có gợi ý những phim tương tự như vậy nữa hay không.
Tác nhân	User
Điều kiện tiên quyết	Người dùng đã đăng nhập vào hệ thống
Hậu điều kiện	Nếu Use-case thành công, bộ phim sẽ hiển thị số lượng sao bằng với đánh giá của người dùng đó.
Dòng sự kiện chính	1. Người dùng truy cập vào trang web xem phim. 2. Người dùng nhấn vào xem chi tiết 1 bộ phim mà mình quan tâm. 3. Người dùng đánh giá phim bằng cách chọn số lượng ngôi sao (lớn nhất là 5, bé nhất là 0) hiển thị ở dưới mỗi bộ phim. 4. Hệ thống ghi nhận đánh giá của người dùng, lưu vào cơ sở dữ liệu và hiển thị đúng số lượng sao nếu người dùng có nhấn vào xem phim đó trong lần kế tiếp.
Dòng sự kiện thay thế	
Dòng sự kiện ngoại lệ	

Yêu cầu phi chức năng	

5. Use case tạo danh sách phim yêu thích

Tóm tắt	Mô tả cách người dùng thêm 1 bộ phim vào danh sách yêu thích của mình.
Tác nhân	User
Điều kiện tiên quyết	Người dùng đã đăng nhập vào hệ thống
Hậu điều kiện	Nếu Use-case thành công, bộ phim sẽ được thêm vào mục danh sách yêu thích của người dùng
Dòng sự kiện chính	1. Người dùng truy cập vào trang web xem phim. 2. Người dùng rê chuột lên bộ phim mình thích. 3. Hệ thống hiển thị 1 biểu tượng trên bộ phim đó. 3a. Nếu biểu tượng có hình dấu cộng, có nghĩa là bộ phim vẫn chưa được thêm vào danh sách, người dùng nhấn vào dấu cộng. *Use case tiếp tục ở UC4a.* 3b. Nếu biểu tượng có hình dấu tick, có nghĩa là bộ phim đã được thêm vào danh sách, người dùng nhấn vào dấu tick *Use case tiếp tục ở UC4b.* 4a. Hệ thống ghi nhận thông tin phim được thêm, lưu vào cơ sở dữ liệu và hiển thị lên danh sách yêu thích của người dùng.

	4b. Hệ thống ghi nhận thông tin phim được thêm, xóa phim khỏi cơ sở dữ liệu và cập nhật lại danh sách yêu thích của người dùng.
Dòng sự kiện thay thế	
Dòng sự kiện ngoại lệ	
Yêu cầu phi chức năng	

6. Use case gợi ý phim

Tóm tắt	Mô tả cách hệ thống gợi ý phim cho người dùng dựa vào thông tin thu thập được từ việc người dùng tương tác với hệ thống.
Tác nhân	User, Hệ thống
Điều kiện tiên quyết	Người dùng đã đăng nhập vào hệ thống
Hậu điều kiện	Nếu Use-case thành công, hệ thống sẽ gợi ý danh sách các bộ phim phù hợp với sở thích của người dùng. Đối với người dùng chưa tương tác với hệ thống, các bộ phim phổ biến, nổi tiếng sẽ được đề xuất cho họ

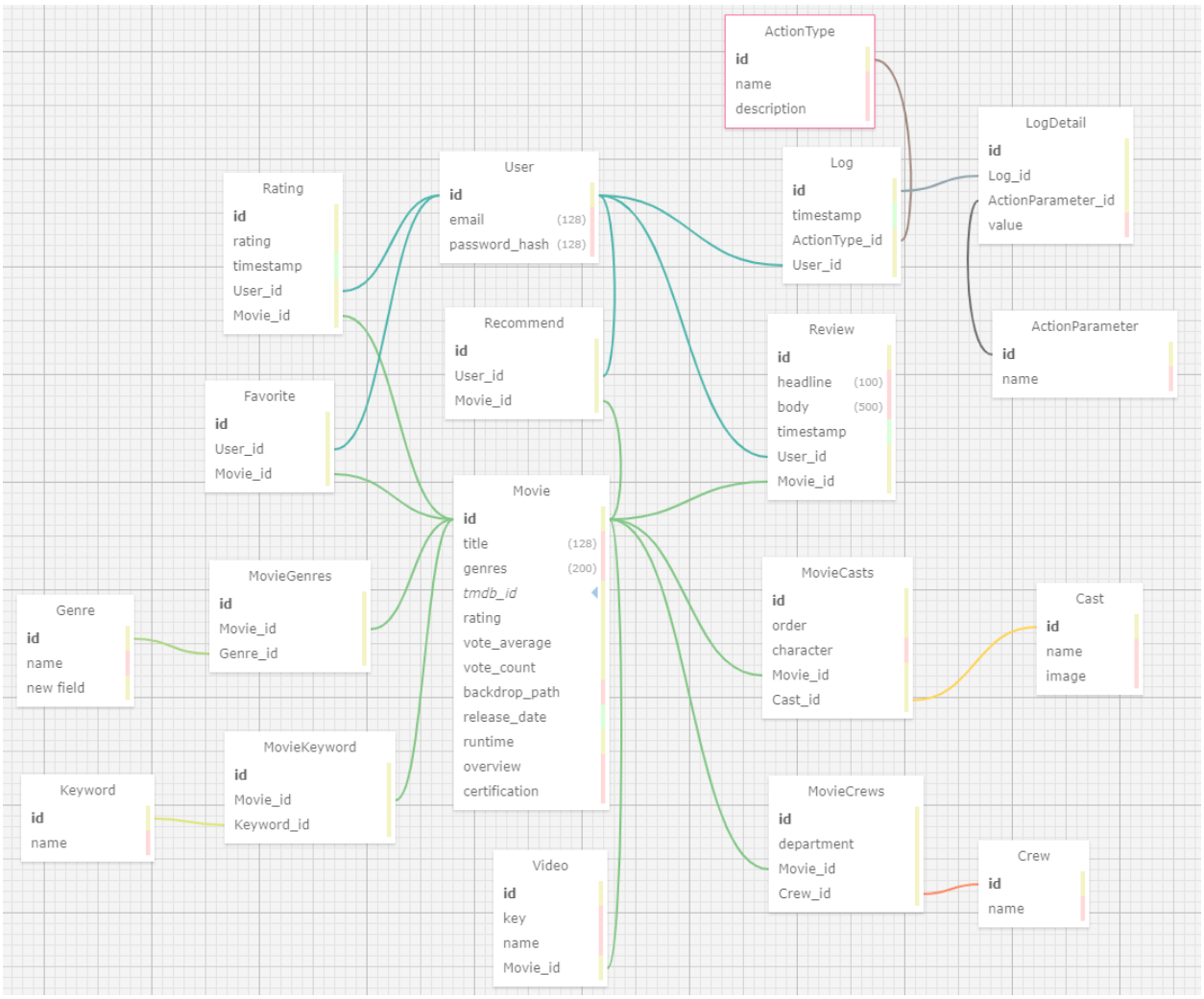
Dòng sự kiện chính	1. Người dùng truy cập vào trang web xem phim. 2. Người dùng nhấn vào xem chi tiết, hoặc đánh giá cho 1 bộ phim nào đó. 3. Hệ thống lưu lại thông tin các bộ phim được người dùng quan tâm, từ đó gợi ý cho các danh sách phim tương tự với phim người dùng thích, hoặc đưa ra danh sách phim của những người dùng khác có sở thích gần giống với người dùng hiện tại, đồng thời giảm khả năng đưa ra các phim giống phim mà người dùng không thích 4. Hệ thống hiện thị danh sách phim gợi ý lên trang
Dòng sự kiện thay thế	1a. Người dùng không tương tác với hệ thống (không chọn thể loại phim yêu thích khi mới tạo tài khoản, không nhấn vào xem phim, không đánh giá phim,…) 1a1. Hệ thống đưa ra danh sách các phim phổ biến, nổi tiếng, được nhiều người quan tâm. *Use case tiếp tục ở UC4*
Dòng sự kiện ngoại lệ	
Yêu cầu phi chức năng	

7. Use case xem chi tiết phim

Tóm tắt	Mô tả cách người dùng xem chi tiết của 1 bộ phim (tiêu đề, tác giả, tóm tắt, diễn viên, đánh giá).

Tác nhân	User
Điều kiện tiên quyết	Người dùng đã đăng nhập vào hệ thống
Hậu điều kiện	Nếu Use-case thành công, người dùng có thể xem chi tiết 1 bô phim mà họ quan tâm, đồng thời cũng có thể đánh giá phim
Dòng sự kiện chính	1. Người dùng truy cập vào trang web xem phim. 2. Người dùng lựa chọn bộ phim mà mình quan tâm trong danh sách phim được thể hiện trên trang chủ, nhấn vào biểu tượng mũi tên đi xuống, ngay dưới mỗi phim 3. Hệ thống hiển thị 1 tab mới gồm 3 tab nhỏ là “Tóm tắt”, “Diễn viên”, “Đánh giá”.
Dòng sự kiện thay thế	
Dòng sự kiện ngoại lệ	
Yêu cầu phi chức năng	

3. Sơ đồ lớp:



4. Kiến trúc xây dựng hệ thống:

Hệ thống được xây dựng theo cấu trúc 1 khối (monolith) gồm có 2 phần:

Client được xây dựng bằng Reactjs và Server được xây dựng bằng Flask (1 framework của Python)

1. Server:

Công nghệ sử dụng tổng quan:

- Framework: Flask

- Database: Sqlite
- Authentication: JWT

Flask Framework:

Ưu điểm:

- Tốc độ nhanh
- Hỗ trợ mọi loại CSDL từ relational db đến no sql.
- Độ phức tạp tối thiểu
- Chủ nghĩa tối giản tuyệt đối, dễ học dễ tiếp cận
- Không có ORM, dễ dàng kết nối với tiện ích mở rộng
- Trình gỡ lỗi được nhúng vào trình duyệt
- Code ngắn và đơn giản nhất trong số các framework của Python

Nhược điểm

- Tốn thời gian thực thi => Khó scale up hệ thống
- Cộng đồng ít => Tìm giải pháp cho vấn đề hay lỗi khó khăn hơn
- Không có login và authentication
- Khó để tạo migration hơn các framework

Sqlite:

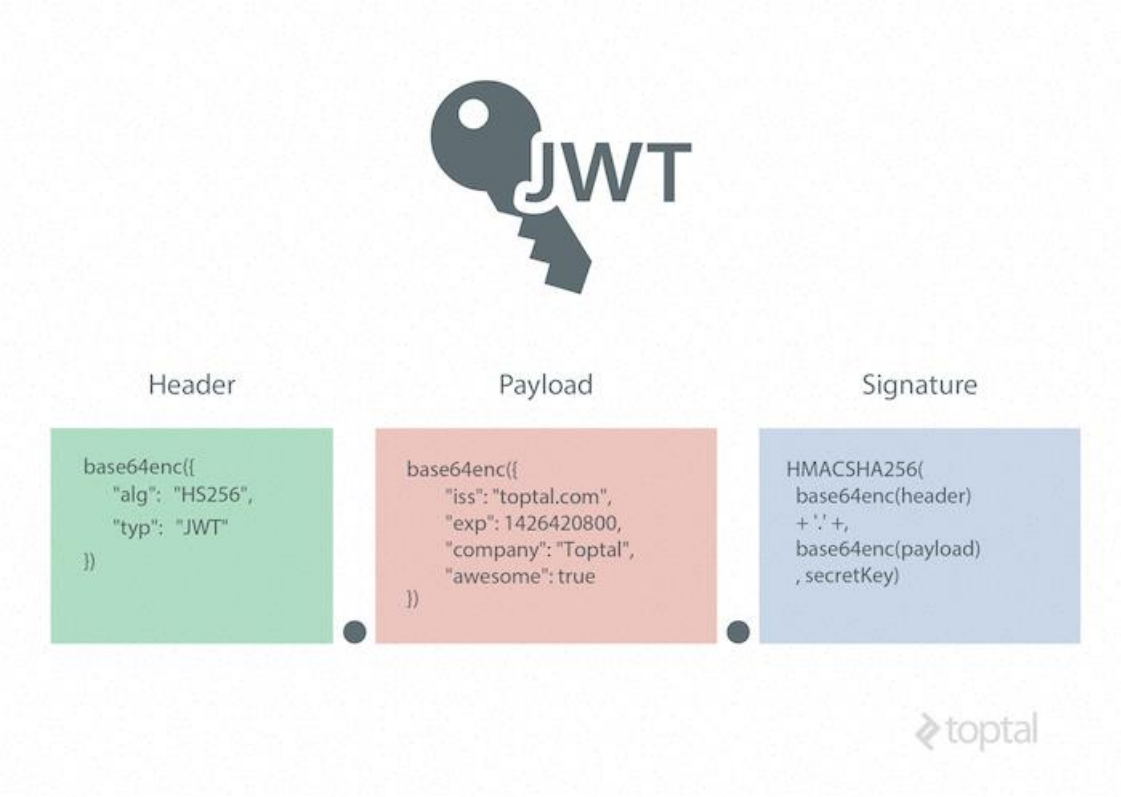
Sqlite là 1 thư viện phần mềm cho phép triển khai 1 SQL Db Engine mà không cần máy chủ, không cần cấu hình, khép kín và gọn nhẹ. Nó là 1 cơ sở dữ liệu, không cần cấu hình, khác với các ngôn ngữ relational db khác khi bạn phải tự cấu hình nó trong hệ thống của mình

Sqlite engine không phải 1 quy trình độc lập (standalone process) như các cơ sở dữ liệu khác, bạn có thể liên kết nó một cách tĩnh hoặc động tuỳ theo yêu cầu của bạn với hệ thống. Sqlite truy cập trực tiếp các file lưu trữ của nó

- SQLite không yêu cầu một quy trình hoặc hệ thống máy chủ riêng biệt để hoạt động.
- SQLite không cần cấu hình, có nghĩa là không cần thiết lập hoặc quản trị.
- Một cơ sở dữ liệu SQLite hoàn chỉnh được lưu trữ trong một file disk đa nền tảng (cross-platform disk file).
- SQLite rất nhỏ và trọng lượng nhẹ, dưới 400KiB được cấu hình đầy đủ hoặc dưới 250KiB với các tính năng tùy chọn bị bỏ qua.
- SQLite là khép kín (self-contained), có nghĩa là không có phụ thuộc bên ngoài.
- Các transaction trong SQLite hoàn toàn tuân thủ ACID, cho phép truy cập an toàn từ nhiều tiến trình (process) hoặc luồng (thread).
- SQLite hỗ trợ hầu hết các tính năng ngôn ngữ truy vấn (query language) được tìm thấy trong tiêu chuẩn SQL92 (SQL2).

JWT (json web token):

JWT là 1 chuẩn mở (RFC 7519) định nghĩa 1 cách nhỏ gọn và khép kín để truyền 1 cách an toàn thông tin giữa các bên dưới dạng đối tượng JSON. Thông tin này có thể được xác minh và đáng tin cậy bởi vì nó có chứa chữ ký số. JWT có thể được ký bằng 1 thuật toán bý mật (với thuật toán HMAC) hoặc 1 private key được sử dụng mã hoá RSA



Như hình trên, ta thấy JWT gồm 3 phần, được ngăn cách bởi dấu chấm (.)

1. Header: chứa kiểu dữ liệu và thuật toá sử dụng để mã hoá ra chuỗi JWT
2. Payload: chứa các thông tin mình muốn đặt trong chuỗi Token
3. Signature: Phần chữ ký số này sẽ tạo ra bằng cách mã hoá phần header, payload kèm theo 1 secret (private key)

Danh sách các routes

STT	Route	Phương thức HTTP	Ý nghĩa
1	/api/auth/register	POST	Đăng ký tài khoản
2	/api/auth/login	POST	Đăng nhập vào hệ thống

3	/api/auth/refresh-token	POST	Làm mới token cho người dùng
4	/api/auth/check-token	GET	Kiểm tra tính hợp lệ của token
5	/api/movies/<int:id>	GET	Lấy thông tin phim có id được yêu cầu
6	/api/movies/<int:id>/reviews	GET	Lấy toàn bộ đánh giá của phim có id được yêu cầu
7	/api/movies/<int:id>/similar	GET	Lấy danh sách phim tương tự phim có id được yêu cầu
8	/api/movies/<int:id>/rate	POST	Chấm điểm bộ phim có id được gửi lên
9	/api/movies/<int:id>/rate	DELETE	Xoá điểm của bộ phim với id được gửi lên
10	/api/movies/<int:id>/reviews	POST	Đánh giá bộ phim có id được gửi lên
11	/api/movies/<int:id>/review	GET	Lấy bình luận của bộ phim có id được gửi lên
12	/api/movies/<int:id>/favorites	POST	Thêm hoặc xoá bộ phim có id được gửi lên khỏi danh sách yêu thích
13	/api/movies/popular	GET	Lấy danh sách phim thịnh hành

14	/api/movies/top-rated	GET	Lấy danh sách phim được đánh giá cao nhất
15	/api/movies	GET	Tìm kiếm phim
16	/api/user/recommend	GET	Lấy danh sách phim gợi ý
17	/api/user/favorites	GET	Lấy danh sách phim yêu thích

2. Client:

Phần client bao gồm các trang có chức năng nhận, gửi dữ liệu cho phía backend và trả dữ liệu về cho người dùng, sử dụng Reactjs và Redux.

PHẦN 5: TỔNG KẾT

1. Kết luận:

Xây dựng hệ thống khuyến nghị là 1 bài toán khó. Tuy nhiên bên cạnh vấn đề khó khăn trong việc xây dựng thuật toán, chúng ta vẫn có thể tìm thấy nhiều điều thú vị. Chính những thú vị này là động lực thôi thúc chúng em tham gia nghiên cứu, góp phần giải quyết những vấn đề vướng mắc.

Trong khuôn khổ luận văn này, những vấn đề liên quan đến các bài toán cơ rút trích đặc trưng hay tính độ tương đồng đã được chúng em tìm hiểu rất công phu cả chiều rộng và chiều sâu vấn đề. Trên những cơ sở nghiên cứu đó, chúng em đã cài đặt thành công các thuật toán để đưa ra khuyến nghị với độ chính xác cao. Đây cũng chính là thành quả rất xứng đáng cho quá trình tìm em và thử nghiệm của chúng em trong suốt một học kỳ làm việc cật lực. Các kết quả nghiên cứu của chúng em đã mở ra nhiều hướng tiếp cận mới để giải quyết cho các bài toán liên quan đến xử lý ngôn ngữ tự nhiên mà kết quả hoàn toàn chấp nhận được.

Tuy nhiên, chúng em cũng muốn nhấn mạnh rằng, để có thể giải quyết tốt hơn nữa các bài toán liên quan đến việc xây dựng hệ thống khuyến nghị. Và điều quan trọng không kém khi sử dụng các phương pháp máy học là chúng ta phải xây dựng được một bộ ngữ liệu hoàn chỉnh và phải được công nhận. Có như thế, chúng ta mới có thể có cơ sở khẳng định cho các kết quả mà chúng ta đạt được ra với thế giới.

2. Hướng phát triển:

Trong bài toán xây dựng hệ thống khuyến nghị, điều quan trọng nhất vẫn là hệ thống có thể tự học hành vi người dùng để có thể cải thiện hiệu suất dự đoán cũng

như đưa ra những gợi ý chính xác hơn. Do đó, việc có mặt của máy học là không thể thiếu đối với hệ thống này.

Ngoài ra, việc số lượng item cũng như user lớn dẫn đến ảnh hưởng tới hiệu năng của server và khả năng phản hồi tới từng request của user. Do đó, việc tối ưu hoá các thuật toán và tăng khả năng chịu tải của server là điều thiết yếu.

Cuối cùng, việc thiết kế server 1 khối sẽ khiến khối lượng công việc của server phải làm trong cùng 1 khoảng thời gian là rất nhiều, dẫn tới server không thể cho ra kết quả một cách nhanh nhất có thể. Từ đó, việc tách hệ thống ra thành nhiều services hay còn gọi là microservices sẽ giúp cho hệ thống hoạt động tốt hơn

TÀI LIỆU THAM KHẢO

- Tham khảo về Content-base Filtering: https://machinelearningcoban.com/2017/05/17/contentbasedrecommendersys/
- Tham khảo về Collaborative Filtering: https://machinelearningcoban.com/2017/05/24/collaborativefiltering/ https://machinelearningcoban.com/2017/05/31/matrixfactorization/
- Một số tài liệu trên Viblo:
    - https://viblo.asia/p/he-thong-goi-y-va-cac-huong-tiep-can-jvEla6DN5kw
    - https://viblo.asia/p/toi-uu-hoa-ung-dung-react-voi-lazy-loading-YWOZr83w5Q0
    - https://viblo.asia/p/gioi-thieu-ve-matplotlib-mot-thu-vien-rat-huu-ich-cua-python-dung-de-ve-do-thi-yMnKMN6gZ7P
    - https://viblo.asia/p/reactjs-loading-skeleton-trong-reactjs-WAyK84kkKxX
    - https://viblo.asia/p/co-ban-ve-redux-form-ORNZqNONl0n
    - https://viblo.asia/p/co-ban-ve-react-router-v4-6J3ZgODMZmB
- Tham khảo về RMSE: https://vi.wikipedia.org/wiki/Sai_s%E1%BB%91_to%C3%A0n_ph%C6%B0%C6%A1ng_trung_b%C3%ACnh

- Tham khảo về React:
    - https://reactjs.org/docs/getting-started.html
    - https://redux-saga.js.org/docs/basics/
    - https://redux.js.org/introduction/getting-started

- Tham khảo về Flask:

  - Blog của Miguel Grinberg: https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world
  - https://flask-cors.readthedocs.io/en/latest/
  - https://flask-sqlalchemy.palletsprojects.com/en/2.x/
  - https://flask-script.readthedocs.io/
  - https://flask-migrate.readthedocs.io/en/latest/
  - https://flask-sqlalchemy.palletsprojects.com/en/2.x/models/
  - https://flask-jwt-extended.readthedocs.io/en/stable/api/#flask_jwt_extended.jwt_refresh_token_required

- Một số bài viết trên stackoverflow:
  - https://stackoverflow.com/questions/25594893/how-to-enable-cors-in-flask
  - https://stackoverflow.com/questions/21214270/how-to-schedule-a-function-to-run-every-hour-on-flask
  - https://stackoverflow.com/questions/33556572/paginate-a-list-of-items-in-python-flask
  - https://stackoverflow.com/questions/27900018/flask-sqlalchemy-query-join-relational-tables