



UNIVERSITÀ DEGLI STUDI DI MESSINA

DIPARTMENT:

Scienze matematiche e informatiche, scienze fisiche e scienze
della terra

DEGREE COURSE:

INFORMATICA

Software Development Project Report



Movie Recommendation System By Bot

By: Tien M. Nguyen

Guided by: Professor Distefano Salvatore

Academic year 2017-2018

Contents

I.	Overview	4
1.	Project Ideal	4
2.	Project Description	4
3.	Iterations Planning	4
II.	Software Processes	7
1.	What is SCRUM?	7
2.	Why use SCRUM?.....	9
3.	Used Technologies	9
III.	Sprint 1	11
1.	Sprint Planning.....	11
2.	Sprint Implementation.....	11
2.1	Use Cases Diagram	11
2.2	Requirements Analysis.....	12
2.3	Functional Requirements	13
2.4	Non-functional Requirements	14
2.5	The Use of Framework Django and ReactJS.....	14
2.6	Recommendation Systems.....	14
3.	Sprint Review	23
4.	Sprint Retrospective.....	24
IV.	Sprint 2	24
1.	Sprint Planning.....	24
2.	Sprint Implementation.....	24
2.1	Neighborhood-based collaborative filtering.....	24
2.2	User-User collaborative filtering.....	25
2.3	Item-Item collaborative filtering.....	31
2.4	Development in Python	33
2.5	About Collaborative Filtering	37
2.6	Backend Development	38
2.7	Frontend Development.....	39
3.	Sprint Review	44
4.	Sprint Retrospective.....	45
V.	Sprint 3	46

1.	Sprint Planning.....	46
2.	Sprint Implementation.....	46
2.1	Matrix factorization collaborative filtering.....	46
2.2	MFCF in Python	49
2.3	Review on MFCF.....	53
2.4	Advantage of MFCF	54
2.5	Import Data to Database.....	54
3.	Sprint Review	55
4.	Sprint Retrospective.....	55

I. Overview

1. Project Ideal

The Ideal of Project is a website where users can be suggested which movies they might like. They will also be able to rate on a 5-point scale the movies they have watched. The system will use a machine learning algorithm to determine which movies users will like based on data given. The more movies they rated, the more accurate the suggested movies would be.

2. Project Description

The final product will be a website where users can sign up, log in, find a movie to rate, and get recommendations for movies they might like.

There will be three main parts to be focused on in this project:

- Backend
- Frontend
- Movie Recommendation System

Details of the development will be covered in more detail in the Sprint sections below.

3. Iterations Planning

Backlog Refinement

MRSBB Team

Backlog

Analytics

+ New Work Item

View as Board


Column Options

<div><div></div><div></div></div>	Order	ID	Title	Assigned To	State	Tags
<div><div></div><div></div></div>	1	42	<div><div></div></div> Requirements Analysis	<div></div> MINH TIEN N...	<div></div> To Do	
	2	43	<div><div></div></div> Recommendation System Research	MINH TIEN N...	<div></div> To Do	
	3	52	<div><div></div></div> Content-based Recommendation System	MINH TIEN N...	<div></div> To Do	
	4	64	<div><div></div></div> Backend Development for Authenticate Management	MINH TIEN N...	<div></div> To Do	
	5	58	<div><div></div></div> Neighborhood-based Collaborative Filtering	MINH TIEN N...	<div></div> To Do	
	6	66	<div><div></div></div> Frontend Development Authenticate Management	MINH TIEN N...	<div></div> To Do	
	7	76	<div><div></div></div> Matrix Factorization Collaborative Filtering	MINH TIEN N...	<div></div> To Do	
	8	83	<div><div></div></div> Frontend for Movie Management	MINH TIEN N...	<div></div> To Do	
	9	82	<div><div></div></div> Backend for Movie Management	MINH TIEN N...	<div></div> To Do	
	10	86	<div><div></div></div> Apply MFCF to Backend	MINH TIEN N...	<div></div> To Do	
	11	84	<div><div></div></div> Backend for Rating Management	MINH TIEN N...	<div></div> To Do	
	12	85	<div><div></div></div> Frontend for Rating Management	MINH TIEN N...	<div></div> To Do	

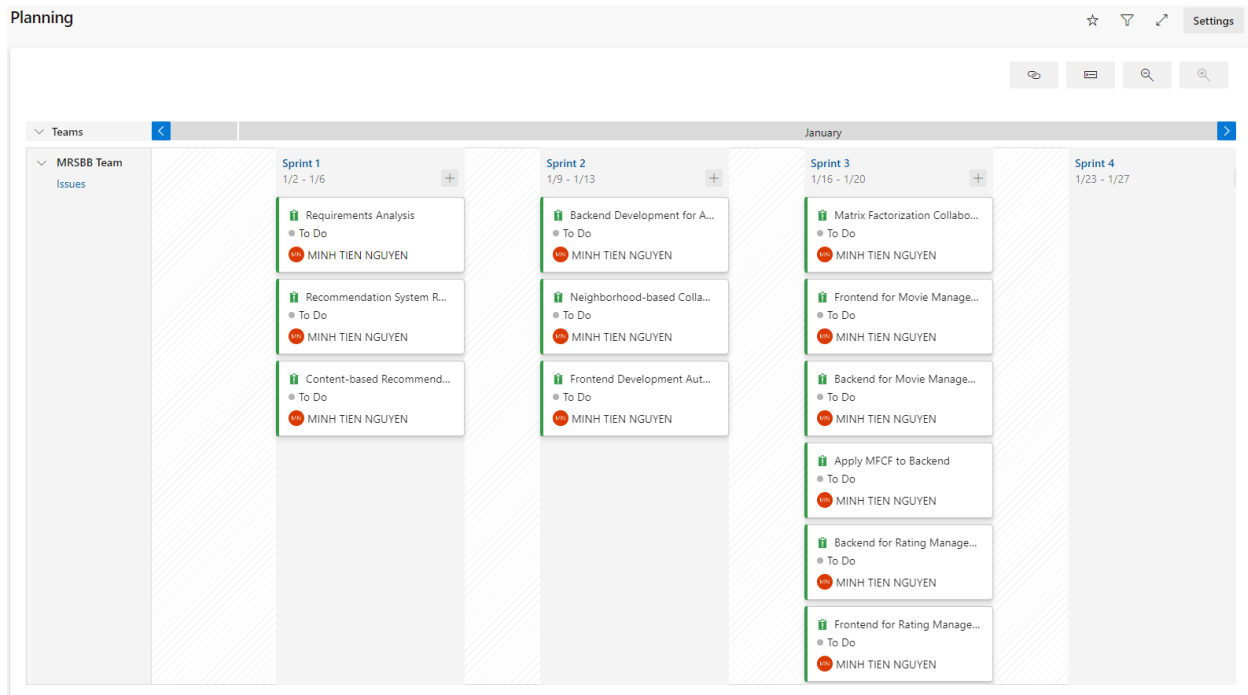
Sprint Planning

	Order	ID	Title	Assigned To	State	Tags
+	1	42	<div> <div></div> <div>Requirements Analysis</div> </div>	MINH TIEN N...	To Do	
		45	<div> <div></div> <div>Use Cases Diagram</div> </div>		Done	
		46	<div> <div></div> <div>Analyze Requiremens</div> </div>		Done	
		47	<div> <div></div> <div>Functional Requirements</div> </div>		Done	
		48	<div> <div></div> <div>Non-Functional Requirements</div> </div>		Done	
		49	<div> <div></div> <div>Decide What Framework Will Be Used</div> </div>		Done	
	2	43	<div> <div></div> <div>Recommendation System Research</div> </div>	MINH TIEN N...	To Do	
		50	<div> <div></div> <div>Research on Recommendation System</div> </div>		Done	
		51	<div> <div></div> <div>Analyze MovieLens 100k Data</div> </div>		Done	
	3	52	<div> <div></div> <div>Content-based Recommendation System</div> </div>	MINH TIEN N...	To Do	
		53	<div> <div></div> <div>Utility Matrix</div> </div>		Done	
		54	<div> <div></div> <div>Item Profile</div> </div>		Done	
		55	<div> <div></div> <div>Building Loss Function</div> </div>		Done	
		56	<div> <div></div> <div>Development in Python</div> </div>		Done	
		57	<div> <div></div> <div>Evaluate System</div> </div>		Done	
	4	64	<div> <div></div> <div>Backend Development for Authenticate Management</div> </div>	MINH TIEN N...	To Do	
		65	<div> <div></div> <div>Django and djoser configuration</div> </div>		Done	
	5	58	<div> <div></div> <div>Neighborhood-based Collaborative Filtering</div> </div>	MINH TIEN N...	To Do	
		59	<div> <div></div> <div>Similarity Determination Function</div> </div>		Done	
		60	<div> <div></div> <div>Fill the missing values in utility matrix</div> </div>		Done	
		61	<div> <div></div> <div>Item-Item Collaborative Filtering</div> </div>		Done	
		62	<div> <div></div> <div>Development in Python</div> </div>		Done	
		63	<div> <div></div> <div>Evaluate System</div> </div>		Done	

	Order	ID	Title	Assigned To	State	Tags
	6	66	<div> <div></div> <div>Frontend Development Authenticate Management</div> </div>	MINH TIEN N...	To Do	
		67	<div> <div></div> <div>Redux</div> </div>		Done	
		68	<div> <div></div> <div>Sequence Diagram</div> </div>		Done	
		69	<div> <div></div> <div>Logic of Authentication</div> </div>		Done	
		70	<div> <div></div> <div>Login Page</div> </div>		Done	
		71	<div> <div></div> <div>Signup Page</div> </div>		Done	
		72	<div> <div></div> <div>Activate Page</div> </div>		Done	
		73	<div> <div></div> <div>Home Page</div> </div>		Done	
		74	<div> <div></div> <div>Reset Password Page</div> </div>		Done	
		75	<div> <div></div> <div>ReactJS Design</div> </div>		Done	
	7	76	<div> <div></div> <div>Matrix Factorization Collaborative Filtering</div> </div>	MINH TIEN N...	To Do	
		77	<div> <div></div> <div>Advantage of MFCF</div> </div>		To Do	
		78	<div> <div></div> <div>Loss Function</div> </div>		To Do	
		79	<div> <div></div> <div>Update Equation as Gradient Descent Model</div> </div>		To Do	
		80	<div> <div></div> <div>Development in Python</div> </div>		To Do	
		81	<div> <div></div> <div>Review MFCF</div> </div>		To Do	
	8	83	<div> <div></div> <div>Frontend for Movie Management</div> </div>	MINH TIEN N...	To Do	
		92	<div> <div></div> <div>Search Page</div> </div>		To Do	
		93	<div> <div></div> <div>Suggest Page</div> </div>		To Do	
		94	<div> <div></div> <div>Movie Detail and Rating Page</div> </div>		To Do	
	9	82	<div> <div></div> <div>Backend for Movie Management</div> </div>	MINH TIEN N...	To Do	
		87	<div> <div></div> <div>Clean Data</div> </div>		To Do	
		88	<div> <div></div> <div>Import Cleaned Data to Database</div> </div>		To Do	
		91	<div> <div></div> <div>Build API for Frontend</div> </div>		To Do	

10	86	▼  Apply MFCF to Backend	MINH TIEN N...	● To Do
	89	✓ Advantage of MFCF model		● To Do
	90	✓ Save Model in Database		● To Do
	95	✓ Build API for Frontend		● To Do
11	84	▼  Backend for Rating Management	MINH TIEN N...	● To Do
	96	✓ Build API for Frontend		● To Do
12	85	▼  Frontend for Rating Management	MINH TIEN N...	● To Do
	97	✓ Rating Component in Movie Detail Page		● To Do

Planning



- Sprint 1 runs from 2nd Jan to 6th Jan (5 working days). 3 Issues with 12 Tasks.
- Sprint 2 runs from 9th Jan to 13th Jan (5 working days). 3 Issues with 15 Tasks.
- Sprint 3 runs from 16th Jan to 20th Jan (5 working days). 6 Issues with 16 Tasks.

II. Software Processes

There are many different software development processes to choose from, including the traditional Waterfall model, component-based software engineering, and evolutionary development approaches such as Agile. Among the Agile methodologies, I have chosen SCRUM for this project.

The Waterfall model follows a linear and sequential approach, with each stage of the development process completed before moving on to the next. This method can work well for projects with well-defined requirements and a stable environment.

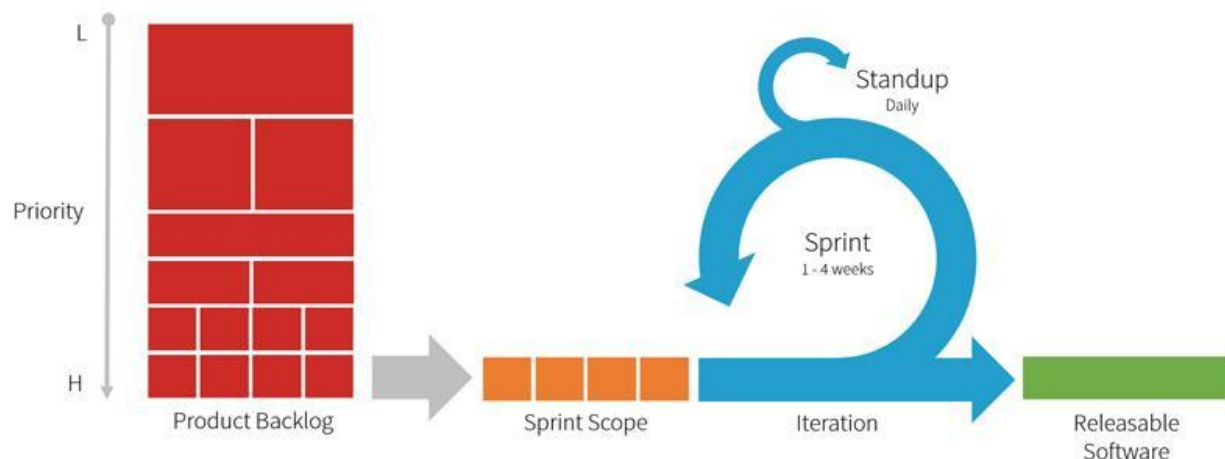
Component-based software engineering emphasizes the modular construction of software, using pre-existing components to build the final product. This approach can be useful for developing reusable components for multiple projects.

Agile development, of which SCRUM is one methodology, is a flexible and adaptive approach that emphasizes collaboration and continuous improvement. In Agile, work is divided into short cycles and delivered incrementally, with regular opportunities for reflection and adjustment.

Overall, SCRUM provides a suitable approach for complex and dynamic software development projects, which is why I have chosen it for this project.

SCRUM was chosen for this project because of its flexibility. I'm sure the project will change design a few times because I want to research and choose the most suitable technologies for the project through iterations both in terms of system design as well as data set research before completing the system.

1. What is SCRUM?



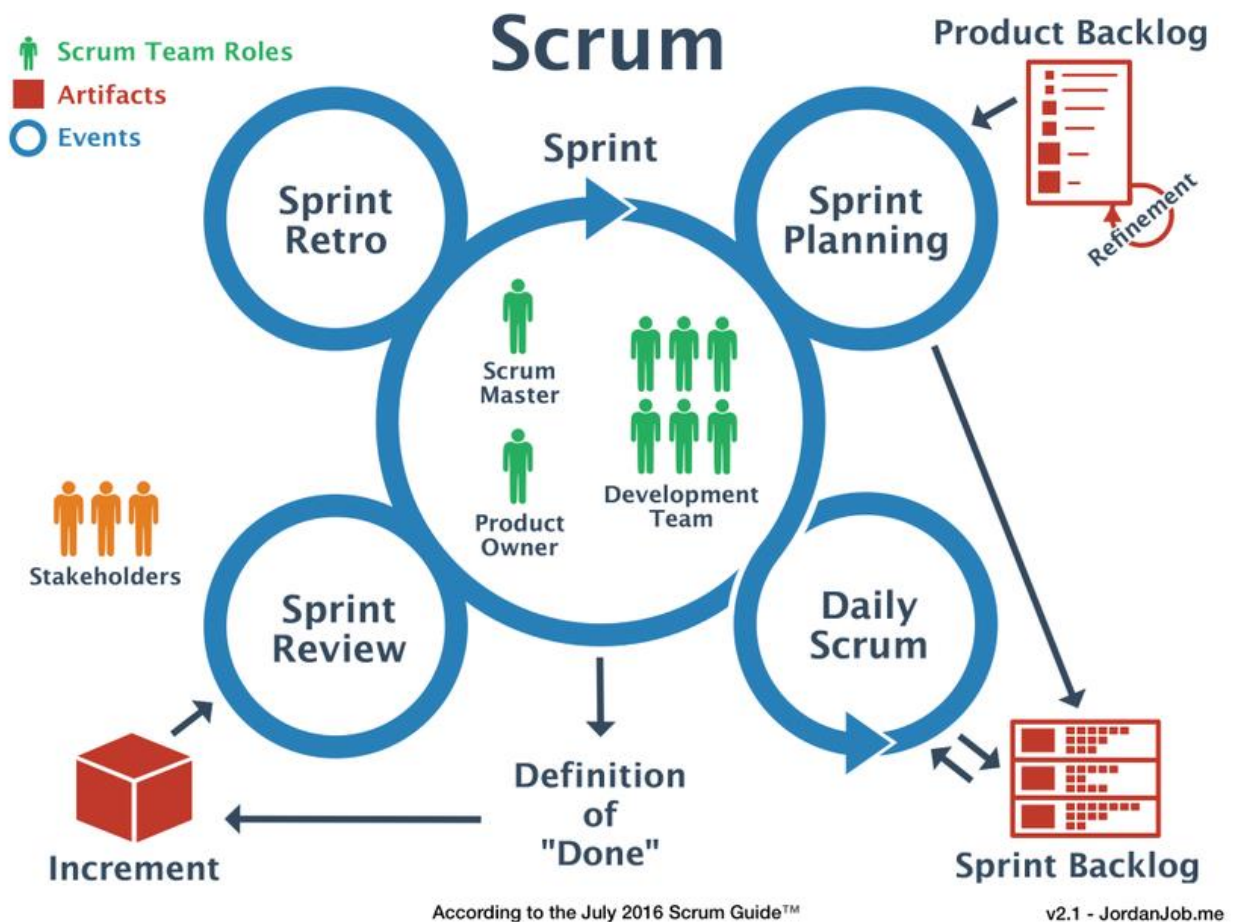
Many people often have confusion about SCRUM and Agile, and think they are the same type, which is not true. In fact, SCRUM is one of the most popular frameworks used to implement Agile.

And there are many other frameworks that can also be used to implement Agile like Kanban for example. However, why use SCRUM, what is special about SCRUM?

SCRUM is a process and management framework that helps solve complex problems, but still ensures efficiency, creativity and products created to achieve the highest value. SCRUM itself is a simple framework to help the most effective coordination among development team members when implementing complex products. With SCRUM, the product is built on a sequence of iterative processes (called a Sprint). Sprints take place regularly, each sprint is an opportunity to learn and adjust to achieve the best fit and results.

Referring to SCRUM are:

- Lightweight.
- Easy to understand.
- Difficult to manage and master.



When applying SCRUM, there are 4 important meetings (Meetings or Ceremonies) that make up the structure in each Sprint as follows:

1. Sprint Planning: The project team's planning meeting, to determine what needs to be done in the upcoming Sprint.
2. Sprint Implementation, Daily stand-up: Also known as “Daily SCRUM”, a small 15-minute meeting a day to exchange work between the development team.
3. Sprint Review: A sharing meeting where members point out what they have accomplished in that Sprint.
4. Sprint Retrospective: Evaluation, review of what has been done and not done of the current Sprint and propose action solutions for the next Sprint to be better and more complete.

A SCRUM development team will have slightly different components from the traditional Waterfall model, with the following 3 roles:

- Product Owner
- SCRUM Master
- Development Team

And because the SCRUM development team is cross-functional, the “Development Team” will include:

- Testers
- Designers
- Ops Engineers

2. Why use SCRUM?

Scrum's origins come from 1986's study of successful product development processes, all of which have such characteristics as new requirements and frequent and continuous requirements change, time to product delivery, and more. product to market noticeably short. In this study, the most productive way for the development team to work was compared with the game of rugby and called Scrum. Transparency, auditability, and adaptability are the three fundamental foundations of Scrum. And here are the reasons why you should use Scrum.

- Scrum allows for the freedom of implementation.
- Scrum is easy to learn and use.
- Scrum embraces change.
- Scrum reduces risk when building products.
- Scrum optimizes the efficiency and effort of the development team.
- Scrum allows customers to use products earlier.
- Scrum continuous improvement.

3. Used Technologies

- Git and GitHub

Git is a popular version control system for tracking changes in software code, supporting collaboration, branching, and merging. It is widely used in software development to manage and organize code changes efficiently. All code in this project can be accessed via: <https://github.com/iamtienng/MRSBB-SCRUM>

- Draw.io

Draw.io is a free, online diagramming tool for making flowcharts, UML diagrams, network diagrams, and more. It offers a simple interface, pre-made templates and shapes, collaboration, and integration with other tools. All diagrams in this project are drawn with Draw.io.

- Azure DevOps

Azure DevOps is a suite of development tools, services, and features offered by Microsoft for software development teams to plan, develop, test, and deliver software. It includes services for Agile project management, continuous integration and delivery, testing, and more.

- Django

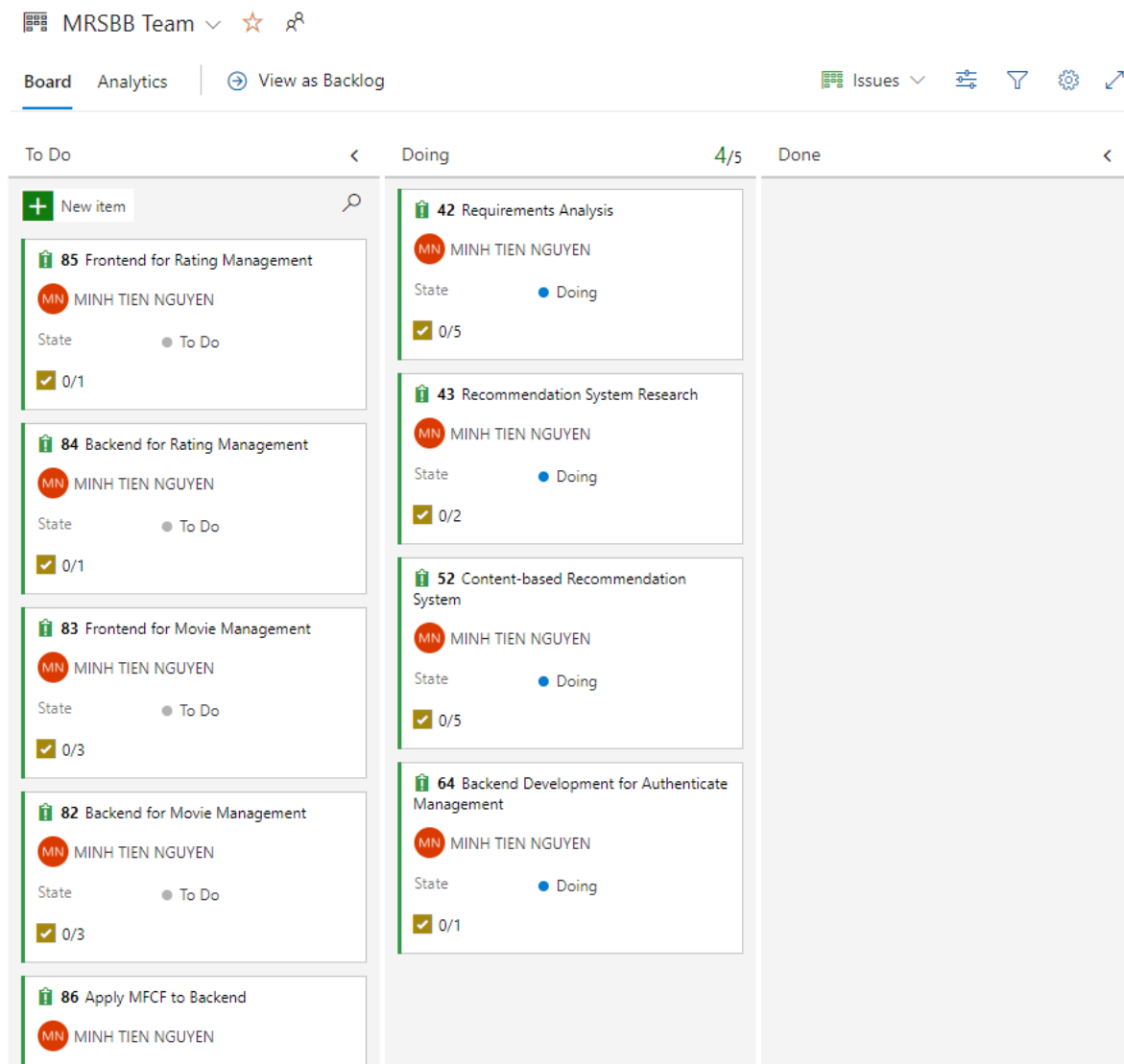
Django is a high-level Python web framework used for developing dynamic and scalable web applications. It provides a clean and efficient way to build and maintain applications, with features such as URL routing, template engine, object-relational mapping, and security features.

- ReactJS

React is a JavaScript library for building user interfaces, commonly used for developing single-page applications and mobile applications. It uses a component-based approach, allowing developers to build complex UIs by composing simple and reusable components. React also uses a virtual DOM for efficient updates and rendering, making it fast and efficient for dynamic applications.

III. Sprint 1

1. Sprint Planning

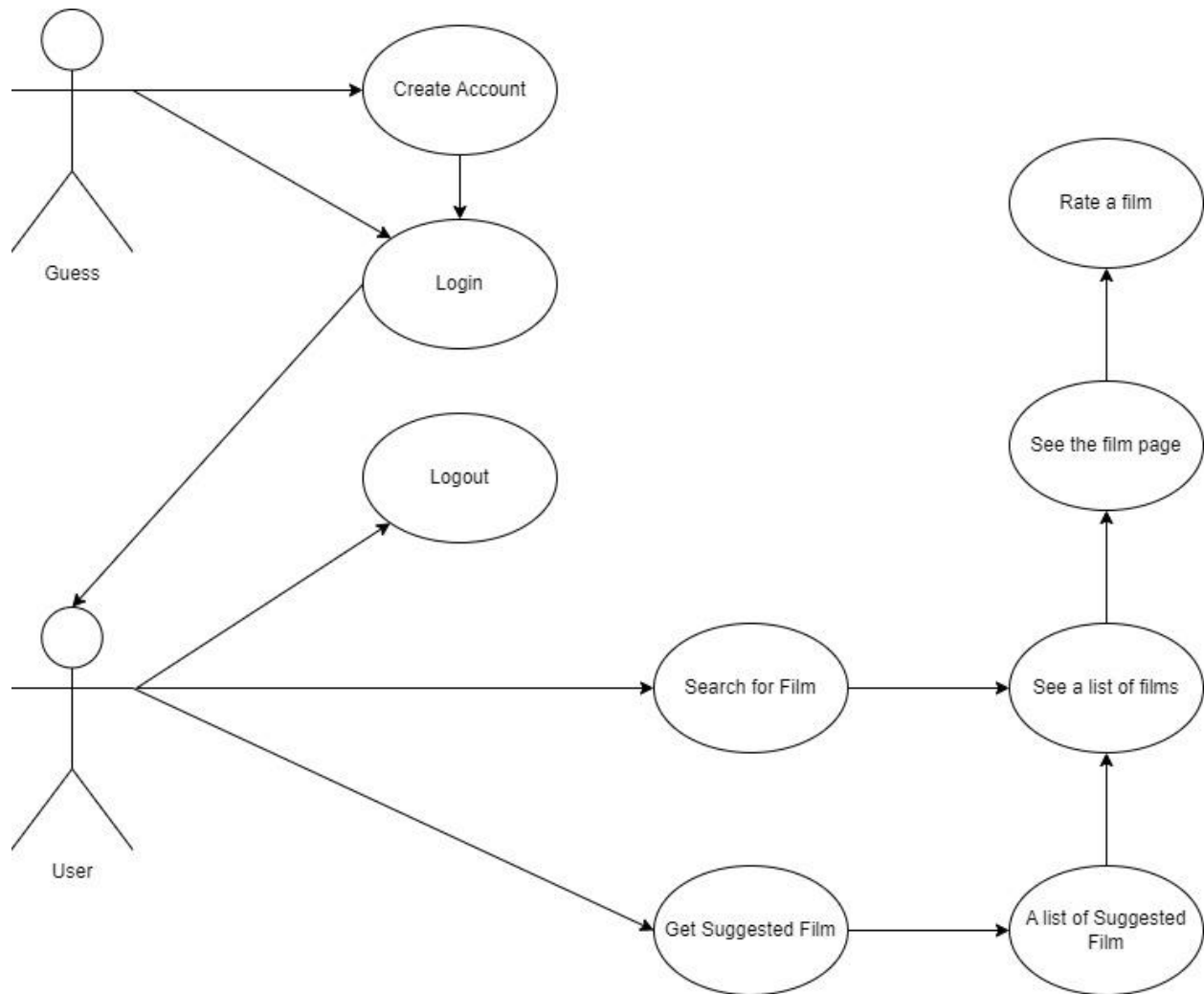


In the First Sprint, I want to focus on analyzing requirements, research on data and one recommendation system algorithm and finish authentication function if it is possible to. All the tasks shown above will be discussed in the next sections.

2. Sprint Implementation

2.1 Use Cases Diagram

The specifications for the first software prototype involve a thorough analysis of the customer requirements, which is important and requires careful attention as it determines whether the customer's needs goods are met or not. A "use case diagram" will be used to describe the relationship clearly and accurately between users and the actions they can take while using the software.



2.2 Requirements Analysis

In the initial iteration, I prioritized rapid development of the ideal solution and analyzed the data to identify the suitable machine learning models.

The software will allow the user to perform the various functions:

- Create Account
- Login
- Search for Film
- Get Suggested Film
- See a list of films
- See a specific film
- Rate a film
- Logout

The “Create Account” operation allows a new user or guess who has just had access to the software to be able to register and access the software functions. Registration is required since we need user rating film data.

The “Login” operation allows a user who already has access to the application to be able to reconnect to the system by entering the same data that he entered during the previous registration period.

The “Search for Film” operation will allow a logged in user to find the exact movie they want to rate.

The “Get Suggested Film” operation will allow a logged in user to get their might-like films by the machine learning algorithm.

The “See a list of films” operation will allow a logged in user to get a list after the operations “Search for Film” or “Get Suggested Film”.

The “Rate a Film” operation will follow after the “Search for Film” which will allow user to rate a film and save the rate in the database, maybe a function “Remove a Rate” will appear in the future.

The “Get Suggested Film” operation is the main operation which will allow a user to get their suggested films that they might like to depend on the machine learning model we develop.

Finally, the “Logout” operation will allow a logged in user to logout from system. Users will need to login again if they need to do any other action on the next connection.

2.3 Functional Requirements

Functional requirements refer to the product features or functions that must be implemented by the development team to enable users to carry out their tasks. It is important to make these requirements clear for both the development team and stakeholders. Typically, functional requirements describe the system's behavior in specific circumstances. Below are functional requirements for software:

FR.1: Authenticate Management:

- FR.1.1: Registration User
- FR.1.2: Verify User

FR.2: Movie Management:

- FR.2.1: Movie Searching
- FR.2.2: Movie Suggestion

FR.3: Rating Management:

- FR.3.1: Create Rating

2.4 Non-functional Requirements

Nonfunctional requirements, not related to the system functionality, rather define how the system should perform. Below are non-functional requirements for software:

NFR.1: Friendly Graphic User Interface.

NFR.2: Multiplatform compatible.

NFR.3: Fast response to user.

2.5 The Use of Framework Django and ReactJS

Django is a high-level Python web framework designed for rapid development of secure and maintainable web applications. It provides a full-stack framework with a set of tools and libraries for building dynamic websites and web applications. Django emphasizes reusability and "pluggability" of components, rapid development, and the principle of don't repeat yourself (DRY).

To rapidly demonstrate to the client how the system works, I chose to use the Django framework. Django provides a built-in system for Authenticate Management, and we won't need to worry about database design. Django is also capable of scaling well so if the client chooses this development direction, we can choose to continue using the Django framework.

React is a popular JavaScript library for building user interfaces, used for developing web and mobile applications. It uses a component-based approach, allowing developers to create complex UIs by combining simple, reusable components. React also utilizes a virtual DOM for efficient updates and rendering, making it fast and suitable for dynamic applications.

2.6 Recommendation Systems

We have probably encountered the following phenomena many times. Youtube automatically plays the clips related to the clip we are watching, or automatically suggest clips that we might like. When we buy an item on Amazon, the system will automatically suggest frequently used products bought together, or it knows what you might like based on your purchase history. Google displays ads for products related to the keyword you just searched on Google. Facebook suggests making friends. Netflix automatically suggests movies for users. And there are many other examples where the system has the ability to automatically suggest to users the products they may like. By advertising to that right audience, the effectiveness of marketing will also increase.

The algorithms behind these applications are machine learning algorithms collectively known as recommendation systems.

2.6.1 Content-based Recommendation System

Recommendation systems are a wide field of machine learning and are less old than classification or regression because the internet has only really exploded in the last 15-20 years. There are two main entities in a recommendation system: user and item. User is the user; item is the product,

such as movies, songs, books, clips, or other users in the friend recommendation problem. The main purpose of recommender systems is to predict a user's interest in a certain product, thereby having an appropriate recommendation strategy.

Recommendation systems are generally divided into two broad groups: Content-based system and Collaborative filtering. In the first prototype, we will research on Content-based system.

Content-based system: recommendation based on product characteristics. For example, if a user watches a lot of movies about criminal police, then recommend a movie in the database that shares criminal characteristics to this user, for example the movie Judge. This approach requires sorting products into groups or finding features of each product. However, there are products that do not have a specific grouping, and it is sometimes impossible to identify the group or feature of each product.

2.6.1.1 Utility Matrix

As mentioned, there are two main entities in recommendation systems: user and item. Each user will have a degree of preference for each item. This interest, if known in advance, is assigned a value for each user-item pair. Information about a user's interest in an item can be collected through a rating system (review and rating); or it may be based on the user clicking on the item's information on the website; Or it can be based on how long and how many times a user views an item's information. The examples in this section are all based on the rating system.

	A	B	C	D	E	F
Bohemian Rhapsody	5	5	0	0	1	?
I Love Rock 'N Roll	5	?	?	0	?	?
Hotel California	?	4	1	?	?	1
Twinkle, Twinkle Little Star	1	1	4	4	4	?
Happy and You Know It	1	0	5	?	?	?

Above is an example of utility matrix with song recommendation system. Songs (items) are rated by users on a scale from 0 to 5 stars. The '?' marks indicate that the data does not yet exist in the database. The Recommendation System needs to manually fill in these values.

With a rating system, a user's interest in an item is measured by the value the user has rated for that item, such as the number of stars out of a total of five stars. The set of all ratings, including the unknown values that need to be predicted, forms a matrix called the utility matrix. Consider the example shown above. In this example, there are six users A, B, C, D, E, F and five songs. The boxes with numbers represent that a user has rated a song with a rating from 0 (dislike) to 5 (like it very much). The boxes marked with a '?' correspond to cells with no data. The job of a recommendation system is to predict the value in these gray boxes, thereby giving suggestions to the user. Therefore, the recommendation system problem is sometimes referred to as a matrix completion problem.

In this simple example, it is easy to see that there are two different genres of music: the first three are rock music and the second two are children's music. From this data, we can also guess

that A, B like the Bolero genre, while C, D, E, F like Children's music. From there, a good system should suggest I Love Rock 'N Roll to B; Hotel California for A; Happy and You Know It for D, E, F. Assuming there are only these two types of music, when there is a new song, we just need to classify it into any genre, thereby giving suggestions to each user.

Usually, there are many users and items in the system, and each user usually only rates a very small number of items, even if there are users who do not rate any items. Therefore, the number of “?” cells of the utility matrix is usually very large, and the number of filled cells is a very small number.

The more cells are filled in, the better the accuracy of the system will be. Therefore, systems always encourage users to express their interest in items by rating those items. The evaluation of items, therefore, not only helps other users know the quality of that item, but also helps the system know the user's preferences, thereby having a reasonable advertising policy.

2.6.1.2 Building Item Profile

In content-based systems, i.e., based on the content of each item, we need to build a profile for each item. This profile is represented mathematically as a feature vector. In simple cases, this vector is directly extracted from the item. For example, consider the information of a song that can be used in the recommendation system:

1. Singer.
2. Composer.
3. Release Year.
4. Genre.

	A	B	C	D	E	F	Item's feature vectors
Bohemian Rhapsody	5	5	0	0	1	?	$x_1 = [0.91, 0.02]^T$
I Love Rock 'N Roll	5	?	?	0	?	?	$x_2 = [0.99, 0.11]^T$
Hotel California	?	4	1	?	?	1	$x_3 = [0.95, 0.05]^T$
Twinkle, Twinkle Little Star	1	1	4	4	4	?	$x_4 = [0.01, 0.99]^T$
Happy and You Know It	1	0	5	?	?	?	$x_5 = [0.03, 0.98]^T$
User's models	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	← need to optimize

Assume the feature vector for each product is given in the last column. For each user, we need to find a corresponding model θ_i such that the model obtained is the best.

There are many other characteristics of a song that can be used. Apart from Hard-to-define Category, all other elements can be clearly defined.

In the example in 6.1, we simplify the problem by constructing a two-dimensional feature vector for each song: the first dimension is the Bolero level, the second dimension is the Children's level of that song. Let the feature vectors for each song be x_1, x_2, x_3, x_4, x_5 . Assume the feature vectors (in columnar form) for each song are given in Figure 17.2. Here, we consider these vectors to be defined in some way.

Similarly, the behavior of each user can also be modeled as a set of parameters θ_i . The training data to build each model θ_u are pairs (item profile, rating) corresponding to the items that the user has rated. Filling in the missing values in the utility matrix is to predict the level of interest when applying the θ_u model to them. This output can be written as a function $f(\theta_u, x_i)$. The choice of the form of $f(\theta_u, x_i)$ depends on the problem. In the first iteration, we will be interested in the simplest form, the linear form.

2.6.1.3 Building Loss Function

Assume that the number of users is N , the number of items is M . Profile matrix $X = [x_1, x_2, \dots, x_M] \in R^{d \times M}$, and the utility matrix is $Y \in R^{M \times N}$. The component in the $m - th$ row, $n - th$ column of Y is the interest (here is the number of stars rated) of the $n - th$ user on the $m - th$ item that the system has collected. The Y matrix is missing a lot of components corresponding to the values that the system needs to predict. In addition, let R be the rated or not matrix representing whether a user has rated an item or not. Specifically, r_{mn} equals 1 if the $m - th$ item has been evaluated by the $n - th$ user, 0 otherwise.

Linear model

Suppose that we can find a model for each user, illustrated by a column vector of coefficients $w_n \in R^d$ and bias b_n such that a user's interest in an item can be calculated using a function linear:

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

Considering any $n - th$ user, if we consider the training set as the set of filled components of y_n (the $n - th$ column of the Y matrix), we can construct a loss function similar to ridge regression (linear). regression with l_2 regularization) as follows:

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

where, the second component is regularization and λ is a positive parameter; s_n is the number of items that the $n - th$ user has rated, which is the sum of the elements on the $n - th$ column of the matrix R , i.e., $s_n = \sum_{m=1}^M r_{mn}$. Note that regularization is not usually applied to bias b_n .

Since the loss function expression depends only on the items that have been evaluated, we can reduce it by making $\hat{y}_n \in R^{s_n}$ a sub-vector of y_n , constructed by extracting components with different signs. "?" in the $n - th$ column of Y . At the same time, let $\hat{X}_n \in R^{d \times s_n}$ be a submatrix of the feature matrix X , created by extracting the columns corresponding to the items evaluated by the user $n - th$. (See the example below for better understanding). Then, the loss function expression of the model for the $n - th$ user is abbreviated as:

$$L_n(w_n, b_n) = \frac{1}{2s_n} \|\hat{X}_n^T w_n + b_n e_n - \hat{y}_n\|_2^2 + \frac{\lambda}{2s_n} \|w_n\|_2^2$$

where e_n is the column vector with all components being 1. This is exactly the loss function of ridge regression. The pair of solutions w_n, b_n can be found through gradient descent algorithms. In this iteration, we will directly use the Ridge class in `sklearn.linear_model`. It is worth noting here that w_n is only determined if the $n - th$ user has rated at least one product.

2.6.2 Researching on MovieLens Data

The 100k MovieLens database (<https://goo.gl/BzHgtq>) was published in 1998 by GroupLens (<https://grouplens.org>). This database includes 100,000 (100k) reviews from 943 users for 1682 movies. We can also find similar databases with about 1M, 10M, 20M reviews.

After downloading and decompressing, we will get a lot of small files, we only need to care about the following files:

- **u.data:** Contains all 943 user reviews for 1682 movies. Each user rated at least 20 movies. Information about the time of evaluation is also given, but we do not use it.
- **ua.base, ua.test, ub.base, ub.test:** are two ways of dividing all data into two subsets, one for training, one for testing. We will test on ua.base and ua.test.
- **u.user:** Contains information about the user, including: id, age, gender, occupation, zipcode (region), as this information can also influence the preferences of users. However, we will not use this information, except for the id information to identify different users.
- **u.genre:** Contains the names of 19 movie genres. Genres include: unknown, Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western
- **u.item:** information about each movie. The first few lines of the file:

1	Toy Story (1995)	01-Jan-1995	http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)	0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2	GoldenEye (1995)	01-Jan-1995	http://us.imdb.com/M/title-exact?GoldenEye%20(1995)	0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
3	Four Rooms (1995)	01-Jan-1995	http://us.imdb.com/M/title-exact?Four%20Rooms%20(1995)	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
4	Get Shorty (1995)	01-Jan-1995	http://us.imdb.com/M/title-exact?Get%20Shorty%20(1995)	0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0
5	Copycat (1995)	01-Jan-1995	http://us.imdb.com/M/title-exact?Copycat%20(1995)	0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0

In each line, we will see the id of the movie, the name of the movie, the release date, the link on imdb, and the binary numbers 0, 1 at the end to indicate which of the 19 genres the movie

belongs to. u.genre. A movie can be of many different genres. Information about this category will be used to build item profiles.

```
import pandas as pd
import numpy as np

#Read user file:
u_cols=['user_id', 'age', 'sex', 'occupation', 'zip_code']
users=pd.read_csv('ml-100k/u.user',sep="|",names=u_cols,encoding='latin-1')
n_users = users.shape[0]
print('Number of users: ', n_users)

#Read ratings file:
r_cols=['user_id', 'movie_id', 'rating', 'unix_timestamp']

ratings_base = pd.read_csv('ml-100k/ua.base', sep='\t', names=r_cols,
encoding='latin-1')
ratings_test = pd.read_csv('ml-100k/ua.test', sep='\t', names=r_cols,
encoding='latin-1')

rate_train = ratings_base.to_numpy()
rate_test = ratings_test.to_numpy()

print('Number of training rates: ', rate_train.shape[0])
print('Number of test rates: ', rate_test.shape[0])
```

Result:

```
Number of users: 943
Number of training rates: 90570
Number of test rates: 9430
```

2.6.3 Building The Content-based Recommendation System

2.6.3.1 Building Item Profile

An important job in a content-based recommendation system is to build a profile for each item, that is, a feature vector for each item. First, we need to load all the information about the items into the item's variable:

```
#Reading items file:
i_cols = ['movie id', 'movie title', 'release date', 'video release date', 'IMDb URL', 'unknown', 'Action', 'Adventure', 'Animation', 'Children's', 'Comedy', 'Crime', 'Documentary', 'Drama', 'Fantasy', 'Film-Noir', 'Horror', 'Musical', 'Mystery', 'Romance', 'Sci-Fi', 'Thriller', 'War', 'Western']

items = pd.read_csv('ml-100k/u.item', sep='|', names=i_cols, encoding='latin-1')

n_items = items.shape[0]
print('Number of items: ', n_items)
```

Result:

```
Number of items: 1682
```

Since we're building our profile by movie genre, we'll only be interested in the 19 binary values at the end of each row:

```
# Build Item Profile with Movie Genres, they are the last 19 columns
X0 = np.asmatrix(items)
X_train_counts = X0[:, -19:]
```

Next, we display the first few rows of the rate_train matrix:

```
print(rate_train[:4, :])
```

Result:

```
[[ 1 1 5 874965758]
 [ 1 2 3 876893171]
 [ 1 3 4 878542960]
 [ 1 4 3 876893119]]
```

The first row is understood as the first user to rate the first movie 5 stars. The last column is a number indicating the evaluation time, we will ignore this parameter.

Next, we will construct the feature vector for each item based on the TF-IDF feature and movie genre matrix (<https://goo.gl/bpDdQ8>) in the sklearn library.

```
from sklearn.feature_extraction.text import TfidfTransformer
transformer = TfidfTransformer(smooth_idf=True, norm='l2')
X = transformer.fit_transform(X_train_counts.tolist()).toarray()
```

After this step, each row of X corresponds to the feature vector of a movie.

2.6.3.2 Find model for each user

For each user, we need to find out what movies that user has rated, and the value of those ratings.

```
def get_itemsRated_by_users(rate_matrix,user_id):
    """
    in each line of rate_matrix, we have infor: user_id, item_id, rating
    (scores), time_stamp
    we care about the first three values
    """
    y=rate_matrix[:,0] # all users
    # item indices rated by user_id
    # we need to +1 to user_id since in the rate_matrix, id starts from 1
    # while index in python starts from 0
    ids = np.where(y == user_id +1)[0]
    item_ids = rate_matrix[ids, 1] - 1 # index starts from 0
    scores = rate_matrix[ids, 2]
    return (item_ids, scores)
```

Now, we can find the Ridge Regression coefficients for each user:

```
from sklearn.linear_model import Ridge
from sklearn import linear_model
d=X.shape[1] #data dimension
W = np.zeros((d, n_users))
b = np.zeros((1, n_users))
for n in range(n_users):
    ids, scores = get_itemsRated_by_users(rate_train,n)
    if(len(ids)==0):
        W[:,n]=0
        b[0,n]=0
        continue
    clf=Ridge(alpha=0.01, fit_intercept=True)
    Xhat=X[ids,:]
    clf.fit(Xhat,scores)
    W[:,n]=clf.coef_
    b[0,n]=clf.intercept_
```

2.6.3.3 Predict User's Ratings

After calculating the W and b coefficients, the rating that each user rated each movie is predicted:

```
#predicted scores
Yhat=X.dot(W)+b
```

Here is an example with a user whose id is 30.

```

n = 30
np.set_printoptions(precision=2) # 2 digits after .
ids, scores = get_itemsRated_by_users(rate_test, n)
print('Rated movies ids : ', ids )
print('True ratings : ', scores)
print('Predicted ratings: ', Yhat[ids, n])

```

Result:

```

Rated movies ids : [134 301 320 483 492 497 503 681 704 885]
True ratings : [4 4 4 5 5 4 5 2 5 2]
Predicted ratings: [3.42 3.56 4.7 3.82 3.66 3.53 4.19 3.67 4.03 3.64]

```

2.6.3.4 Evaluate Content-based Recommendations System

To evaluate the model found, we will use the Root Mean Squared Error (RMSE), which is the square root of the mean squared error.

```

from math import sqrt
def evaluate(Yhat, rates, W, b):
    se=0
    cnt=0
    for n in range(n_users):
        ids,scores_truth=get_itemsRated_by_users(rates,n)
        scores_pred=Yhat[ids,n]
        e=scores_truth-scores_pred
        se+=np.linalg.norm(e)
        cnt+=e.size
    return sqrt(se/cnt)

print('RMSE for training: ', evaluate(Yhat, rate_train, W, b))
print('RMSE for test : ', evaluate(Yhat, rate_test, W, b))

```

```

RMSE for training: 0.27453050104193866
RMSE for test : 0.6113579566395051

```

2.6.4 Disadvantages of Content-based Recommendation System

The feature of the content-based recommendation system is that the model building for each user does not depend on other users but on the profiles of the items. This has the advantage of saving memory and computation time. This approach has two basic disadvantages.

1. When building a model for a single user, content-based systems do not take advantage of information from other users. This information is often very useful because the purchasing behavior of users is often grouped into a few simple groups. If it knows the

buying behavior of some users in the group, the system should be able to infer the behavior of the rest of the users.

2. It's not always possible to build a profile for each item.

These disadvantages can be addressed by a technique called collaborative filtering. Since I don't know if it's the optimal algorithm for the system or not, I decided to further study the recommendation algorithm in the next Sprint.

3. Sprint Review

After the first Sprint, most of the tasks set out have been completed. I have done well in setting up the Use Case Diagram. However, because I spent too much time learning about the recommendation system, I could not complete the authentication function.

The screenshot displays a Jira board for the 'MRSBB Team'. The board is organized into three columns: 'To Do', 'Doing', and 'Done'. The 'Doing' column is currently selected, showing 1 out of 5 tasks in progress. The tasks are categorized by color-coded icons: green for 'To Do', blue for 'Doing', and red for 'Done'.

Tasks in 'To Do' column:

- 85 Frontend for Rating Management (MINH TIEN NGUYEN, State: To Do, Progress: 0/1)
- 84 Backend for Rating Management (MINH TIEN NGUYEN, State: To Do, Progress: 0/1)
- 83 Frontend for Movie Management (MINH TIEN NGUYEN, State: To Do, Progress: 0/3)
- 82 Backend for Movie Management (MINH TIEN NGUYEN, State: To Do, Progress: 0/3)
- 86 Apply MFCF to Backend (MINH TIEN NGUYEN, State: To Do, Progress: 0/3)

Task in 'Doing' column:

- 64 Backend Development for Authenticate Management (MINH TIEN NGUYEN, State: Doing, Progress: 0/1)

Tasks in 'Done' column:

- 43 Recommendation System Research (MINH TIEN NGUYEN, State: Done, Progress: 2/2)
 - Research on Recommendation System
 - Analyze MovieLens 100k Data
- 42 Requirements Analysis (MINH TIEN NGUYEN, State: Done, Progress: 5/5)
 - Use Cases Diagram
 - Analyze Requirements
 - Functional Requirements
 - Non-Functional Requirements
 - Decide What Framework Will Be Used
- 52 Content-based Recommendation System (MINH TIEN NGUYEN, State: Done, Progress: 5/5)

4. Sprint Retrospective

Certainly, in the next Sprint the time spent on the recommendation system will be reduced because the background knowledge has been learned in this Sprint. I'll have to try to build the authentication function in the next Sprint.

IV. Sprint 2

1. Sprint Planning

The screenshot shows a Jira board for the "MRSBB Team". The board is set to "View as Backlog" and displays a Kanban workflow with three columns: "To Do", "Doing" (with 3/5 items), and "Done".

To Do Column:

- Swimlane 85 Frontend for Rating Management:** Task 85 (MINH TIEN NGUYEN) is in "To Do" state. Progress: 0/1.
- Swimlane 84 Backend for Rating Management:** Task 84 (MINH TIEN NGUYEN) is in "To Do" state. Progress: 0/1.
- Swimlane 83 Frontend for Movie Management:** Task 83 (MINH TIEN NGUYEN) is in "To Do" state. Progress: 0/3.
- Swimlane 82 Backend for Movie Management:** Task 82 (MINH TIEN NGUYEN) is in "To Do" state. Progress: 0/3.
- Swimlane 86 Apply MFCF to Backend:** Task 86 (MINH TIEN NGUYEN) is in "To Do" state. Progress: 0/3.

Doing Column (3/5 items):

- Swimlane 64 Backend Development for Authenticate Management:** Task 64 (MINH TIEN NGUYEN) is in "Doing" state. Progress: 0/1.
- Swimlane 66 Frontend Development Authenticate Management:** Task 66 (MINH TIEN NGUYEN) is in "Doing" state. Progress: 0/9.
- Swimlane 58 Neighborhood-based Collaborative Filtering:** Task 58 (MINH TIEN NGUYEN) is in "Doing" state. Progress: 0/5.

Done Column:

- Swimlane 43 Recommendation System Research:** Task 43 (MINH TIEN NGUYEN) is in "Done" state. Progress: 2/2. Sub-tasks: "Research on Recommendation System" and "Analyze MovieLens 100k Data".
- Swimlane 42 Requirements Analysis:** Task 42 (MINH TIEN NGUYEN) is in "Done" state. Progress: 5/5. Sub-tasks: "Use Cases Diagram", "Analyze Requirements", "Functional Requirements", "Non-Functional Requirements", and "Decide What Framework Will Be Used".
- Swimlane 52 Content-based Recommendation System:** Task 52 (MINH TIEN NGUYEN) is in "Done" state. Progress: 5/5.

2. Sprint Implementation

2.1 Neighborhood-based collaborative filtering

In the last Sprint, I researched a simple recommendation system based on feature vectors of each individual item. The feature of the content-based recommendation system is that the model building for each user does not depend on other users but on the profiles of the items. This has

the advantage of saving memory and computation time. This approach has two basic disadvantages. First, when building a model for a single user, content-based systems fail to take advantage of information from other users. This information is often very useful because the purchasing behavior of users is often grouped into a few simple groups. If it knows the buying behavior of some users in the group, the system should be able to infer the behavior of the rest of the users. Second, it is not always possible to build a profile for each item.

These disadvantages can be addressed by a technique called collaborative filtering.

The idea of NBCF is to determine a user's interest in an item based on the behavior of other users that closely resemble this user. Proximity between users can be determined by the degree of interest these users have in other items known to the system. For example, A and B both like the movie Criminal Police, and have rated this movie 5 stars. We already know that A also likes The Judge, so it's likely that B will also like this movie.

The two most important questions in a neighborhood-based collaborative filtering system are:

1. How to determine the similarity between two users?
2. Once similar users have been identified, how can one predict a user's interest in an item?

Determining each user's interest in an item based on how similar users are to that item is called user-user collaborative filtering. Another approach that is said to be more productive is item-item collaborative filtering. In this approach, instead of determining similarity between users, the system determines similarity between items. From there, the system suggests items that are close to the items that the user has a high level of interest in.

2.2 User-User collaborative filtering

2.2.1 Similarity determination function

The most important job to do first in user-user collaborative filtering is to identify the similarity between two users. Assuming the only data we have is the utility matrix Y , then the similarity needs to be determined based on the columns corresponding to the two users in this matrix.

	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
-------	---	---	---	---	---	---	---

Suppose there are users from u_0 to u_6 and items from i_0 to i_4 where the numbers in each square represent the number of stars each user has rated the item with a higher value representing a higher level of interest. The question marks are the values the system needs to look for. Set the similarity of two users u_i, u_j to $sim(u_i, u_j)$. The first observation that can be noticed is that u_0, u_1 like i_0, i_1, i_2 and don't like i_3, i_4 very much. The opposite happens for the rest of the users. So, a good similarity function should ensure:

$$sim(u_0, u_1) > sim(u_0, u_i), \forall i > 1$$

To determine the interest of u_0 to i_2 , we should base u_1 's behavior on this item. Fortunately, u_1 already likes i_2 so the system needs to recommend i_2 to u_0 .

The question is, how should the similarity function be constructed? To measure similarity between two users, a common practice is to construct a feature vector for each user and then apply a function capable of measuring the similarity between those two vectors. Note that this feature vector construction is different from the item profile construction as in content-based recommendation systems. These vectors are built directly on the utility matrix and do not use additional external information such as item profiles. For each user, the only information we know is the ratings that user has made, i.e., the column corresponding to that user in the utility matrix. However, the difficulty is that these columns often have a lot of missing values because each user usually only evaluates a very small number of items. One workaround is to help the original system rough estimate these values so that filling doesn't affect the similarity between the two vectors much. This estimation is for similarity calculation only, not the result of the system to be estimated.

So, each '?' should be replaced by what value to limit the estimation bias? The first option that can be thought of is to replace the '?' with the value 0. This is not good because the value 0 corresponds to the lowest level of interest; Just because a user hasn't rated an item doesn't mean they don't care about the item at all. A safer value is 2.5 because it is the average of 0, the lowest, and 5, the highest. However, this value has limitations for easy or fastidious users. Easy users can rate three stars for items they don't like, conversely, difficult users can rate three stars for items they like. Mass replacement of missing elements by 2.5 in this case has not been effective. A more likely value for this is to estimate the missing elements as the average value that a user evaluates. This helps avoid a user being too fastidious or easygoing. And these estimates depend on each user.

Example of User-User Collaborative Filtering

a) Original utility matrix Y and mean user ratings.

	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

↓ ↓ ↓ ↓ ↓ ↓ ↓

\bar{u}_j	3.25	2.75	2.50	1.33	2.50	1.50	3.33
-------------	------	------	------	------	------	------	------

b) Normalized utility matrix \bar{Y}

	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

c) User Similarity Matrix S

	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

d) \hat{Y}

	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

e) Example: 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: $\{0.83, -0.40, -0.23\}$

→ most similar users: $N(u_0, u_i) = \{u_0, u_5\}$ with normalized ratings $\{0.75, 0.5\}$

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

f) Full Y

	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	2.1	4
i_4	2	0	4	2.9	4.06	3.10	5

The last row in Figure a) is the average of each user's ratings. High values correspond to easy-going users and vice versa. Then, if we continue to subtract this average value from each rating and replace the unknown values with 0, we will get a normalized utility matrix as shown in Figure b). Doing this has a few advantages:

- Subtracting the average from each column causes each column to have both positive and negative values. Items corresponding to positive values can be considered as items of greater interest to the user than those corresponding to negative values. Items with a value of 0 mainly correspond to the unknown interest of that user.
- Technically, the number of dimensions of the utility matrix is very large with millions of users and items, if all these values are stored in a matrix, there is a high chance that there will not be enough memory. Observe that since the number of known evaluations is usually a very small number compared to the size of the utility matrix, it would be better to store this matrix as a sparse matrix, i.e., store only other values. no and their location. Therefore, it is better to replace the '?' with the value '0', which is not yet determined whether the user likes the item or not. This not only optimizes memory, but also calculates similarity matrices later on more efficiently. Here, the element in the i-th row, j-th column of the similarity matrix is the similarity of the i-th and j-th users.

After the data has been normalized, the commonly used similarity function is cosine similarity:

$$\text{cosine}_{\text{similarity}}(u_1, u_2) = \cos(u_1, u_2) = \frac{u_1^T u_2}{\|u_1\|_2 \cdot \|u_2\|_2}$$

Where $u_{1,2}$ are vectors corresponding to user 1 and user 2 as above. There is a function in Python that serves to efficiently calculate this value, which we will see in the programming section.

The degree of similarity of two vectors is a real number in the interval $[-1, 1]$. A value of 1 represents two vectors that are completely similar. The cosine function of an angle equals 1 means that the angle between the two vectors is 0, that is, the two vectors have the same direction and the same direction. A cos value of -1 indicates that these two vectors are completely opposite, i.e., have the same direction but different flavor. This means that if the behavior of two users is completely opposite, the degree of similarity between the two vectors is the lowest.

An example of the cosine similarity of users (normalized) in b) is shown in c). Similarity matrix S is a symmetric matrix because cos is an even function, and if user A is the same as user B, the converse is also true. The blue cells on the diagonal are all cos of the angle between a vector and

itself, i.e., $\cos(0) = 1$. When calculating in the following steps, we do not need to care about this 1 value. Continuing to look at the row vectors corresponding to u_0 , u_1 , u_2 , we will see a few interesting things:

- u_0 is closer to u_1 and u_5 (the similarity is positive) than the rest of the users. The high similarity between u_0 and u_1 is understandable since both tend to care more about i_0 , i_1 , i_2 than the rest. The fact that u_0 is close to u_5 may at first seem absurd because u_5 undervalues the items that u_0 appreciates (a); however, looking at the normalized utility matrix in b), we see that this makes sense since the only item that both of these users have provided is i_1 with the corresponding values being positive.
- u_1 is close to u_0 and far from the rest of the users.
- u_2 is close to u_3 , u_4 , u_5 , u_6 and far from the rest of the users.

From this similarity matrix, we can group users into two groups (u_0 , u_1) and (u_2 , u_3 , u_4 , u_5 , u_6). Since this matrix S is small, we can easily observe this; When the number of users is larger, visual identification is not feasible.

It is important to note here that when the number of users is large, the matrix S is also very large and it is likely that there is not enough memory to store it, even if only more than half of the elements of the matrix is stored. this symmetry. For those cases, for each new user, we only need to calculate and save the result of a row of the similarity matrix, corresponding to the similarity between that user and the other users.

2.2.2 Fill in the missing values in the utility matrix

Predicting a user's predicted rating of an item based on these closest users is very similar to what we see in K-nearest neighbors (KNN) with a distance function of cosine similarity.

Similar to KNN, NBCF also uses information of k neighbors to make predictions. Of course, to gauge a user's interest in an item, we are only interested in the neighboring users who have rated the item. The value to be filled in is usually defined as the weighted average of the normalized ratings. There is a point to note, in KNN, the weights are determined based on the distance between two points, and these distances are non-negative numbers. In NBCF, weights are determined based on similarity between two users, these weights can be less than 0. The common formula used to predict the number of stars that user u rated item i is:

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

where $N(u, i)$ is the set of k users that are most similar, i.e., have the highest similarity of u who have evaluated i . d) shows filling in the missing values in the normalized utility matrix. The red background cells represent positive values, i.e., items that may be of interest to that user. Here, the threshold is taken to be 0, this threshold can be completely changed depending on whether we want to suggest more or less items.

An example of calculating the normalized rating of u_1 for i_1 is shown in Figure 18.2e with nearest neighbors $k = 2$. The steps are as follows.

1. Identify the users who rated i_1 , they are u_0 , u_3 , u_5 .
2. The similarity of u_1 with these users is $\{0.83, -0.40, -0.23\}$, respectively. The two ($k = 2$) maximum values are 0.83 and -0.23 for u_0 and u_5 respectively.
3. Determining the (normalized) evaluations of u_0 and u_5 for i_1 , we get two values of 0.75 and 0.5, respectively.
4. Predicted result:

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

Conversion of the normalized rating values to a scale of 5 can be done by adding the columns of the \hat{Y} matrix with the average rating value of each user as calculated in a). How the system decides which item to recommend for each user can be determined in a variety of ways. The system can sort unrated items by predicted rating descending, or it can just select items with a positive normalized predicted rating—corresponding to which this user is more likely to like.

2.3 Item-Item collaborative filtering

User-user CF has some limitations as follows:

- When the number of users is much larger than the number of items (which is often the case), the size of the similarity matrix is very large (each dimension of this matrix has the same number of key elements as the number of users). Storing a large matrix is often not feasible.
- The utility matrix Y is usually very sparse, i.e., only a small percentage of the elements are known. With a very large number of users compared to the number of items, many columns of this matrix have very few or even no non-zero elements because users are often lazy to evaluate items. Therefore, once that user changes the previous ratings or evaluates more items, the average of the ratings as well as the normalized vector corresponding to this user changes a lot. Accordingly, the similarity matrix calculation, which takes a lot of memory and time, also needs to be done again.

There is another approach, instead of finding similarity between users, we can find similarity between items. Since then, if a user likes an item, the system should suggest similar items to that user. This has several advantages:

- When the number of items is smaller than the number of users, the similarity matrix has a smaller size, which makes the storage and computation in later steps more efficient.
- Also assume that the number of items is less than the number of users. Since the total number of reviews is constant, the average number of items rated by a user will be less than the average number of users who have rated an item. In other words, if the utility matrix has fewer rows than columns, the average number of known elements in each row

will be more than the average number of known elements in each column. Accordingly, information about each item is more than information about each user, and the calculation of similarity between rows is also more reliable. Furthermore, the average value of each row also changes less with a few more reviews. As such, updating the similarity matrix can be done less frequently.

This second approach is called item-item collaborative filtering (item-item CF). When the number of items is less than the number of users, this method is preferred.

The procedure for predicting missing ratings is the same as in user-user CF, except that now we need to calculate the similarity between rows.

Example of Item-Item Collaborative Filtering:

a) Original utility matrix Y and mean item ratings.

	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	3	?	?	0	?	?	?	→ 2
i_2	?	4	1	?	?	1	2	→ 1.75
i_3	2	2	3	4	4	?	4	→ 3.17
i_4	2	0	4	?	?	?	5	→ 2.75

b) Normalized utility matrix \bar{Y}

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	2.4	2.4	-0.6	-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
-------	-------	-------	------	---	---	---	------

c) Item similarity matrix S

	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

d) Normalized utility matrix \bar{Y}

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	2.4	2.4	-0.6	-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	-1.2	-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

This result is slightly different from the result found by user-user CF in the last two columns corresponding to u_5 , u_6 . It seems that this result is more reasonable because from the utility matrix, we see that there are two groups of users who like two different groups of items. The first group is u_0 and u_1 ; The second group is the remaining users.

2.4 Development in Python

The collaborative filtering algorithm in this chapter is relatively simple and contains no optimization problems. We continue to use the 100k MovieLens database as in the previous chapter. Below is the code that demonstrates the uuCF class for user-user collaborative filtering.

There are two main methods of this class, `fit`—calculates a matrix similarity, and `predict`—predicts the number of stars a user will rate an item.

```
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from scipy import sparse
```

```

class CF(object):
    def __init__(self, Y_data, k, sim_func = cosine_similarity, uuCF = 1):
        self.uuCF = uuCF # user-user (1) or item-item (0) CF
        # Y_data is a 2d array of shape (n_users, 3), each row of Y_data has form
[user_id, item_id, rating]
        self.Y_data = Y_data if uuCF else Y_data[:, [1, 0, 2]]
        self.k = k # number of neighborhoods
        self.sim_func = sim_func # sim function, default cosine similarity
        self.Ybar_data = None # normalized data
        self.n_users = int(np.max(self.Y_data[:, 0])) + 1 # number of users
        self.n_items = int(np.max(self.Y_data[:, 1])) + 1 # number of items

    def normalize_Y(self):
        users = self.Y_data[:, 0] # all users - first col of the Y_data
        self.Ybar_data = self.Y_data.copy()
        self.mu = np.zeros((self.n_users,))
        for n in range(self.n_users):
            # row indices of ratings made by user n
            ids = np.where(users == n)[0].astype(np.int32)
            ratings = self.Y_data[ids, 2] # ratings made by user n
            self.mu[n] = np.mean(ratings) if ids.size > 0 else 0 # avoid zero
division
            self.Ybar_data[ids, 2] = (ratings - self.mu[n]).flatten()

        ## form the rating matrix as a sparse matrix
        data = np.asarray(self.Ybar_data[:, 2]).flatten()
        row = np.asarray(self.Ybar_data[:, 1]).flatten()
        col = np.asarray(self.Ybar_data[:, 0]).flatten()
        shape = np.asarray((int(self.n_items), int(self.n_users))).flatten()
        self.Ybar = sparse.coo_matrix((data, (row, col)), shape=shape).tocsr()

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

    def fit(self):
        self.normalize_Y()
        self.similarity()

    # predict the rating of user u for item i
    def __pred(self, u, i):
        # find item
        ids = np.where(self.Y_data[:, 1] == i)[0].astype(np.int32)
        # all users who rated i
        usersRated_i = (self.Y_data[ids, 0]).astype(np.int32)
        # similarity between u and usersRated_i

```

```

        # similarity between u and usersRated_i
        sim = self.S[u, usersRated_i].flatten()
        # find k most similarity users
        a = np.argsort(sim)[-self.k:]
        nearest_s = sim[a] # and the corresponding similarity values
        # the corresponding ratings
        r = self.Ybar[i, usersRated_i[a]]
        return (r*nearest_s).sum()/(np.abs(nearest_s).sum() + 1e-8) + self.mu[u]

    def pred(self, u, i):
        if self.uuCF: return self.__pred(u, i)
        return self.__pred(i, u)

    def evaluate_RMSE(self, rate_test):
        n_tests = rate_test.shape[0]
        SE = 0
        for n in range(n_tests):
            pred = self.pred(rate_test[n, 0], rate_test[n, 1])
            SE += (pred - rate_test[n, 2])**2

        RMSE = np.sqrt(SE/n_tests)
        return RMSE

```

Next, we apply it to MoviesLen 100k:

```

r_cols = ['user_id', 'movie_id', 'rating', 'unix_timestamp']

ratings_base = pd.read_csv('ml-100k/ua.base', sep='\t', names=r_cols,
encoding='latin-1')
ratings_test = pd.read_csv('ml-100k/ua.test', sep='\t', names=r_cols,
encoding='latin-1')

rate_train = np.asmatrix(ratings_base)
rate_test = np.asmatrix(ratings_test)

# indices start from 0
rate_train[:, :2] -= 1
rate_test[:, :2] -= 1

rs = CF(rate_train, k = 40, uuCF = 1)
rs.fit()
rs_i = CF(rate_train, k = 40, uuCF = 0)
rs_i.fit()

print('User-user CF, RMSE =', rs.evaluate_RMSE(rate_test))
print('Item-item CF, RMSE =', rs_i.evaluate_RMSE(rate_test))

```

Results:

```

User-user CF, RMSE = 0.9766140289287265
Item-item CF, RMSE = 0.9688460838682366

```

Thus, in this case item-item collaborative filtering gives better results, even if the number of items (1682) is larger than the number of users (943). For other problems, we should try both on a validation set and choose the method that gives better results. We can also replace the neighborhood size k with other values and compare the results.

2.5 About Collaborative Filtering

- CF is an item recommendation method with the main idea based on the behavior of other similar users on the same item. This inference is made based on a similarity matrix that measures the similarity between users.
- To compute the similarity matrix, we first need to normalize the data. The common method is mean offset, which subtracts ratings from the average value a user gives for items.
- The commonly used similarity function is cosine similarity.

- A similar approach is that instead of looking for users that are close to a user (user-user CF), we look for items that are close to a given item (item-item CF). In fact, item-item CF usually gives better results.

2.6 Backend Development

With the help of a library called djoser and the Django framework, creating an authentication system is extremely simple. With some djoser config, we just need to create two classes in the models.py file:

```
class UserAccount(AbstractBaseUser, PermissionsMixin):
    email = models.EmailField(max_length=255, unique=True)
    first_name = models.CharField(max_length=255)
    last_name = models.CharField(max_length=255)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)
    gender = models.CharField(max_length=30, blank=True, null=True)
    birth_date = models.DateField(blank=True, null=True)

    objects = UserAccountManager()

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['first_name', 'last_name', 'gender', 'birth_date']
```

```

class UserManager(AbstractUserManager):
    def create_user(self, email, password=None, **extra_fields):
        if not email:
            raise ValueError('Users must have an email address')

        email = self.normalize_email(email)
        user = self.model(email=email, **extra_fields)

        user.set_password(password)
        user.save()

        return user

    def create_superuser(self, email, password=None, **extra_fields):
        if not email:
            raise ValueError('Users must have an email address')

        email = self.normalize_email(email)
        user = self.model(email=email, **extra_fields)
        user.is_staff = True
        user.is_superuser = True
        user.is_active = True

        user.set_password(password)
        user.save()

        return user

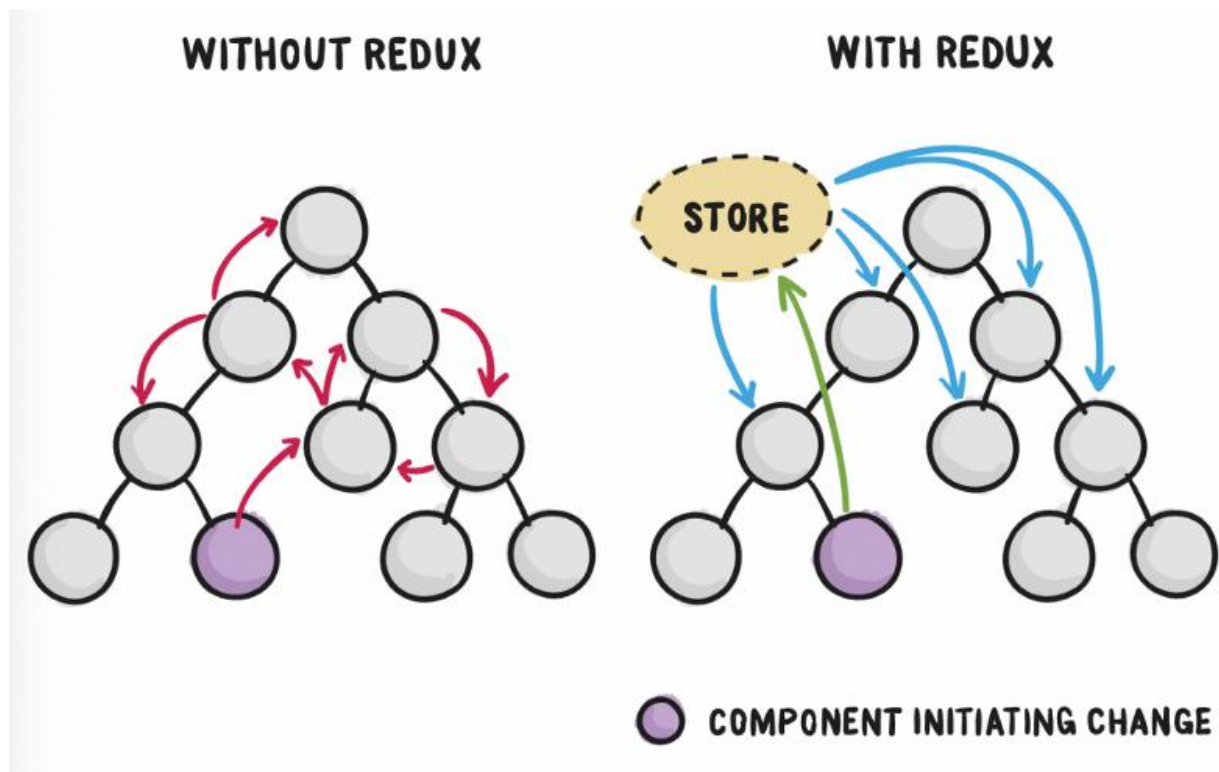
```

So, we've done the work on the server. This has helped a lot in saving time for the Frontend where the main user experience is delivered. We don't even need to worry about the database because Django already handles all of that for us.

2.7 Frontend Development

2.7.1 Redux

As the last Sprint, I decided to use ReactJS as the framework for rapid frontend development. And to get the most out of ReactJS, I use Redux to build the authentication system which requires handling many states of authenticating actions.



Redux is a predictable state management tool for JavaScript applications. It helps you write applications that work consistently, run in different environments (client, server, and native) and are easy to test. Redux was born inspired by the ideas of the Elm language and Facebook's Flux architecture. Therefore, Redux is often used in combination with React.

As the requirements for single-page applications using JavaScript become more and more complex, our code must manage more state. With Redux, the application state is kept in a place called the store and each component can access any state they want from this store.

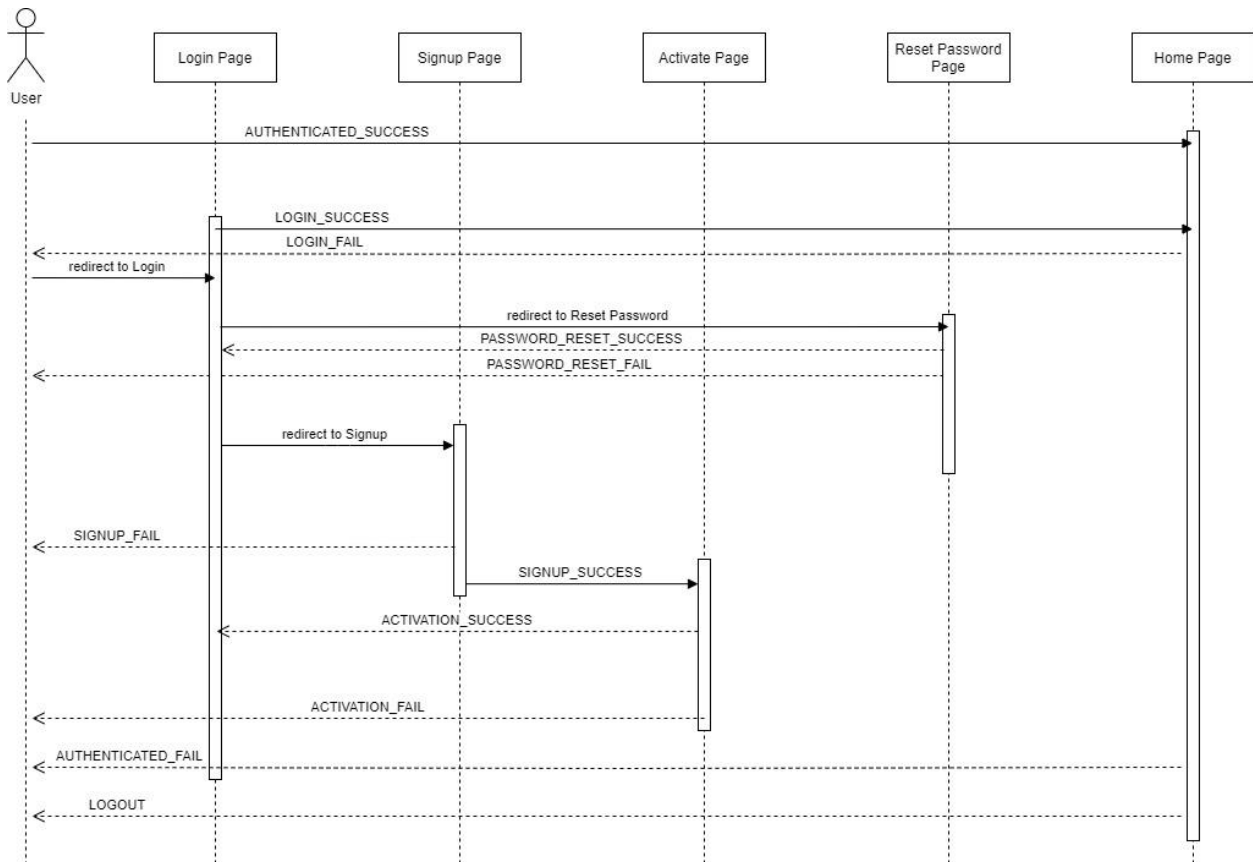
Redux components include:

- **Store:** Store is simply an object that holds all the global state of the application. But instead of saving states, it saves reducers, which will be discussed later.
- **Actions:** When we define actions, we declare the names of the actions in the application. For example, we have 1 state as counter and need 2 methods to increment and decrease the value of counter. Now we define 2 actions named 'INCREMENT' and 'DECREMENT' and that's it, the counter state change handling will be left to reducer.
- **Reducers:** A reducer is equivalent to a state but with descriptions of how the state will change when different actions are called. In the example we have a reducer as counter, it stores the state of the counter and checks whether the action has just been called INCREMENT or DECREMENT and returns the new state as state+1 or state-1 respectively.
- **Dispatchs:** When we need to use an action on a component, we call that action simply by using the dispatch method. For example: `dispatch(increment())`, `dispatch(decrement())`.

2.7.2 Redux in MRSBB

2.7.2.1 Sequence Diagram

Before the Redux development, I finished the sequence diagram as below:



2.7.2.2 Actions

There are 15 types of Authenticating States defined in the file actions/Types.js:

```
export const LOGIN_SUCCESS = "LOGIN_SUCCESS";
export const LOGIN_FAIL = "LOGIN_FAIL";
export const SIGNUP_SUCCESS = "SIGNUP_SUCCESS";
export const SIGNUP_FAIL = "SIGNUP_FAIL";
export const ACTIVATION_SUCCESS = "ACTIVATION_SUCCESS";
export const ACTIVATION_FAIL = "ACTIVATION_FAIL";
export const USER_LOADED_SUCCESS = "USER_LOADED_SUCCESS";
export const USER_LOADED_FAIL = "USER_LOADED_FAIL";
export const AUTHENTICATED_SUCCESS = "AUTHENTICATED_SUCCESS";
export const AUTHENTICATED_FAIL = "AUTHENTICATED_FAIL";
export const PASSWORD_RESET_SUCCESS = "PASSWORD_RESET_SUCCESS";
export const PASSWORD_RESET_FAIL = "PASSWORD_RESET_FAIL";
export const PASSWORD_RESET_CONFIRM_SUCCESS = "PASSWORD_RESET_CONFIRM_SUCCESS";
export const PASSWORD_RESET_CONFIRM_FAIL = "PASSWORD_RESET_CONFIRM_FAIL";
export const LOGOUT = "LOGOUT";
```

Actions corresponding are defined as functions with dispatch in the file actions/Actions.js, for example this is action checkAuthenticated:

```
export const checkAuthenticated = () => async (dispatch) => {
  if (localStorage.getItem("access")) {
    const config = {
      headers: {
        "Content-Type": "application/json",
        Accept: "application/json",
      },
    };

    const body = JSON.stringify({ token: localStorage.getItem("access") });

    try {
      const res = await axios.post(
        `${process.env.REACT_APP_API_URL}/auth/jwt/verify/`,
        body,
        config
      );

      if (res.data.code !== "token_not_valid") {
        dispatch({
          type: AUTHENTICATED_SUCCESS,
        });
      } else {
        dispatch({
          type: AUTHENTICATED_FAIL,
        });
      }
    } catch (err) {
      dispatch({
        type: AUTHENTICATED_FAIL,
      });
    }
  } else {
    dispatch({
      type: AUTHENTICATED_FAIL,
    });
  }
};
```

2.7.2.3 Reducers

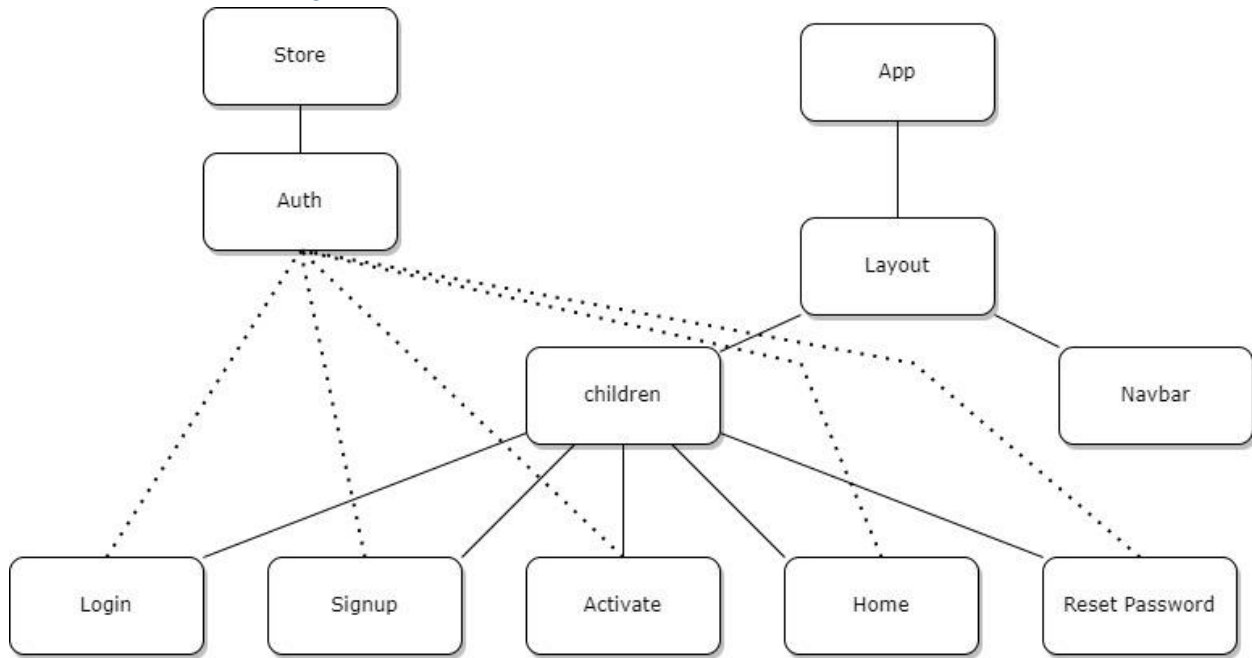
There are two important things in reducers/Auth.js:

```
const initialState = {
  access: localStorage.getItem("access"),
  refresh: localStorage.getItem("refresh"),
  isAuthenticated: null,
  user: null,
};
```

```
export default function (state = initialState, action) {
  const { type, payload } = action;

  switch (type) {
    case AUTHENTICATED_SUCCESS:
      return {
        ...state,
        isAuthenticated: true,
      };
    case LOGIN_SUCCESS:
      localStorage.setItem("access", payload.access);
      localStorage.setItem("refresh", payload.refresh);
      return {
        ...state,
        isAuthenticated: true,
        access: payload.access,
        refresh: payload.refresh,
      };
    case SIGNUP_SUCCESS:
      return {
        ...state,
        isAuthenticated: false,
      };
  }
};
```

2.7.3 ReactJS Design



Together with Redux and a simple system from ReactJS, I have completed the authentication function. I designed the frontend as the React component tree above.

And with Bootstrap, I finally finished creating a beautiful GUI that will be shown in Demo.

3. Sprint Review

So, what I planned for this week is done. I have finished tuning Django as well as created a GUI with the help of the ReactJS framework.

MRSBB Team

Board
Analytics
View as Backlog

Issues
Filter
Settings
Share

To Do
Doing
0/5
Done

+ New item

84 Backend for Rating Management

MN MINH TIEN NGUYEN

State
To Do

0/1

85 Frontend for Rating Management

MN MINH TIEN NGUYEN

State
To Do

0/1

83 Frontend for Movie Management

MN MINH TIEN NGUYEN

State
To Do

0/3

82 Backend for Movie Management

MN MINH TIEN NGUYEN

State
To Do

0/3

86 Apply MFCC to Backend

MN MINH TIEN NGUYEN

State
To Do

66 Frontend Development Authenticate Management

MN MINH TIEN NGUYEN

State
Done

9/9

64 Backend Development for Authenticate Management

MN MINH TIEN NGUYEN

State
Done

1/1

58 Neighborhood-based Collaborative Filtering

MN MINH TIEN NGUYEN

State
Done

5/5

43 Recommendation System Research

MN MINH TIEN NGUYEN

State
Done

2/2

+ Add Task

Research on Recommendation System

Analyze MovieLens 100k Data

4. Sprint Retrospective

In the next week, I will research another algorithm for the Recommendation System from which to conclude and choose the best option for this project.

Along with that is to apply that recommendation system model to actual data and hope to be able to complete the application soon.

45

V. Sprint 3

1. Sprint Planning

The screenshot shows a Jira board for the 'MRSBB Team' in 'Sprint 3'. The board is divided into three columns: 'To Do', 'Doing', and 'Done'. The 'Doing' column has a count of 6/5. Each task card includes a title, a Jira ID, the assignee (MINH TIEN NGUYEN), the state (Doing or Done), and a progress bar.

Column	Jira ID	Task Title	State	Progress
To Do	76	Matrix Factorization Collaborative Filtering	Doing	0/5
	84	Backend for Rating Management	Doing	0/1
	85	Frontend for Rating Management	Doing	0/1
	83	Frontend for Movie Management	Doing	0/3
	82	Backend for Movie Management	Doing	0/3
	86	Apply MFCF to Backend	Doing	0/3
Doing	66	Frontend Development Authenticate Management	Done	9/9
	64	Backend Development for Authenticate Management	Done	1/1
	58	Neighborhood-based Collaborative Filtering	Done	5/5
Done	43	Recommendation System Research	Done	2/2
	42	Requirements Analysis	Done	5/5

2. Sprint Implementation

2.1 Matrix factorization collaborative filtering

In the last Sprint, I researched a collaborative filtering (CF) method based on the behavior of neighboring users or items. In this Sprint, we will get acquainted with another approach for collaborative filtering based on the problem of matrix factorization or decomposition matrix. This method is called matrix factorization collaborative filtering (MFCF).

In content-based recommendation systems, each item is described by a vector x called the item profile. In that method, we need to find a coefficient vector w corresponding to each user such that the known rating that user gives the item is approximately equal to:

$$y \approx w^T x = x^T w$$

With this approach, the utility matrix Y , assuming it is completely filled, will approximate to:

$$Y \approx \begin{bmatrix} x_1^T w_1 & x_1^T w_2 & \dots & x_1^T w_N \\ x_2^T w_1 & x_2^T w_2 & \dots & x_2^T w_N \\ \dots & \dots & \dots & \dots \\ x_M^T w_1 & x_M^T w_2 & \dots & x_M^T w_N \end{bmatrix} = \begin{bmatrix} x_1^T \\ x_2^T \\ \dots \\ x_M^T \end{bmatrix} [w_1 \ w_2 \ \dots \ w_N] = X^T W$$

M: number of items.

N: number of users.

In content-based collaborative filtering, x is built based on only the item's descriptive information (TF-IDF), and this construction is independent of the matching coefficient for each user. The construction of the item profile plays a very important role and has a direct influence on the performance of the model. In addition, building each model individually for each user leads to poor results because it does not exploit the relationship between users.

Now, suppose that we do not need to pre-construct the x profile items, but that the feature vector for each of these items can be trained concurrently with each user's model (here, a coefficient vector). This means, the variables in the optimization problem are both X and W ; where, X is the matrix of the entire profile item, each column corresponds to an item, W is the matrix of the entire user model, each column corresponds to a user.

With this approach, we are trying to approximate the utility matrix $Y \in R^{M \times N}$ is equal to the product of two matrices $X \in R^{M \times K}$ and $W \in R^{K \times N}$. Usually, K is chosen to be a much smaller number than M , N . Then, both the X and W matrices have a rank not exceeding K .

$$Y \approx \hat{Y} = X^T W$$

N

 M Y

\approx

K

 M X^T

\times

N

 K W

User features

(Full) Utility matrix

Item features

The main idea behind matrix factorization for recommendation systems is that there exist latent features that describe the relationship between items and users.

User n 's rating on item m can be approximated by $y_{mn} \approx x_m^T w_n$. We can also add biases to this approximation and optimize those biases. Specifically:

$$y_{mn} \approx x_m^T w_n + b_m + d_n$$

b_m, d_n are the coefficients respectively for item m and user n .

Vector $b = [b_1, b_2, \dots, b_M]^T$ is the bias vector for items.

Vector $d = [d_1, d_2, \dots, d_N]^T$ is the bias vector for users.

Loss Function

$$\mathcal{L}(X, W, b, d) = \frac{1}{2s} \sum_{n=1}^N \sum_{m:r_{mn}=1} (x_m^T w_n + b_m + d_n - y_{mn})^2 + \frac{\lambda}{2} (\|X\|_F^2 + \|W\|_F^2)$$

$r_{mn} = 1$ if m -th item has been rated by n -th user.

s : number of ratings in dataset.

y_{mn} : rating of n -th user for m -th item.

$\|X\|_F$: Frobenius norm of X .

$\frac{1}{2s} \sum_{n=1}^N \sum_{m:r_{mn}=1} (x_m^T w_n + b_m + d_n - y_{mn})^2$ is the loss data, is the mean error of the model.

$\frac{\lambda}{2} (\|X\|_F^2 + \|W\|_F^2)$ is l_2 regularization loss, help avoid overfitting.

Simultaneous optimization of X, W, b, d is relatively complicated. Instead, the method used is to optimize one of the two pairs $(X, b), (W, d)$ in turn, while the other pair is fixed. This process is repeated until the loss function converges.

Optimizing the loss function

When the pair (X, b) is fixed, the pair (W, d) optimization problem can be decomposed into N subproblems:

$$\mathcal{L}_1(w_n, d_n) = \frac{1}{2s} \sum_{m:r_{mn}=1} (x_m^T w_n + b_m + d_n - y_{mn})^2 + \frac{\lambda}{2} (\|w_n\|_2^2)$$

Each problem can be optimized using gradient descent. Our important job is to calculate the derivatives of each of these small loss functions with respect to w_n, d_n . Since the expression in the sign Sigma depends only on the items that have been rated by the user in question (corresponding to $r_{mn} = 1$), we can simply by making \hat{X}_n a submatrix generated by the columns of X corresponds to the items that have been rated by user n , \hat{b}_n is the corresponding sub-bias vector, and \hat{y}_n is the corresponding rating. Then,

$$\mathcal{L}_1(w_n, d_n) = \frac{1}{2s} \|\hat{X}_n^T w_n + \hat{b}_n + d_n \mathbf{1} - \hat{y}_n\|^2 + \frac{\lambda}{2} (\|w_n\|_2^2)$$

With $\mathbf{1}$ is the vector with all elements equal to 1 and the appropriate size. Its derivative is

$$\nabla_{w_n} \mathcal{L}_1 = \frac{1}{S} \hat{X}_n (\hat{X}_n^T w_n + \hat{b}_n + d_n \mathbf{1} - \hat{y}_n) + \lambda w_n$$

$$\nabla_{b_n} \mathcal{L}_1 = \frac{1}{S} \mathbf{1}^T (\hat{X}_n^T w_n + \hat{b}_n + d_n \mathbf{1} - \hat{y}_n)$$

Update equation for w_n, d_n :

$$w_n \leftarrow w_n - \eta \left(\frac{1}{S} \hat{X}_n (\hat{X}_n^T w_n + \hat{b}_n + d_n \mathbf{1} - \hat{y}_n) + \lambda w_n \right)$$

$$d_n \leftarrow d_n - \eta \left(\frac{1}{S} \mathbf{1}^T (\hat{X}_n^T w_n + \hat{b}_n + d_n \mathbf{1} - \hat{y}_n) \right)$$

η is the learning rate.

Similarly, each column x_m of X , which is the feature vector for each item, and b_m can be found by optimizing the problem:

$$\mathcal{L}_2(x_m, b_m) = \frac{1}{2S} \sum_{n:r_{mn}=1} (w_n^T x_m + d_n + b_m - y_{mn})^2 + \frac{\lambda}{2} (\|x_m\|_2^2)$$

Let \hat{W}_m be the matrix created with the columns of W corresponding to the users who have rated item m , \hat{d}_m the corresponding bias sub-vector, and \hat{y}_m the corresponding rating vector. The problem becomes:

$$\mathcal{L}_2(x_m, b_m) = \frac{1}{2S} \|\hat{W}_m^T x_m + \hat{d}_m + b_m \mathbf{1} - \hat{y}_m\|^2 + \frac{\lambda}{2} (\|x_m\|_2^2)$$

Update equation for x_m, b_m :

$$x_m \leftarrow x_m - \eta \left(\frac{1}{S} \hat{W}_m (\hat{W}_m^T x_m + \hat{d}_m + b_m \mathbf{1} - \hat{y}_m) + \lambda x_m \right)$$

$$b_m \leftarrow b_m - \eta \left(\frac{1}{S} \mathbf{1}^T (\hat{W}_m^T x_m + \hat{d}_m + b_m \mathbf{1} - \hat{y}_m) \right)$$

2.2 MFCF in Python

First, we will write an MF class that performs the optimization of variables with a utility matrix given in the form of `Y_data` just like with NBCF. First, we declare some necessary libraries and initialize the MF class:

```

import json
import pandas as pd
import numpy as np

class MF(object):
    def __init__(self, Y, K, lam = 0.1, Xinit = None, Winit = None,
learning_rate = 0.5, max_iter = 10000, print_every = 100):
        self.Y = Y          # represents the utility matrix
        self.K = K          # number of features
        self.lam = lam      # regularization parameter
        self.learning_rate = learning_rate # for gradient descent
        self.max_iter = max_iter          # maximum number of iterations
        self.print_every = print_every    # print loss after each a few iters
        self.n_users = int(np.max(Y[:, 0])) + 1
        self.users_ids = np.unique(np.asarray(Y[:,0].reshape(Y[:,0].shape[0])))
        self.n_items = int(np.max(Y[:, 1])) + 1
        self.items_ids = np.unique(np.asarray(Y[:,1].reshape(Y[:,1].shape[0])))
        self.n_ratings = Y.shape[0]
        self.X = np.random.randn(self.n_items, K) if Xinit is None else Xinit
        self.W = np.random.randn(K, self.n_users) if Winit is None else Winit
        self.b = np.random.randn(self.n_items) # item biases
        self.d = np.random.randn(self.n_users) # user biases

```

Next, we write loss, updateXb, updateWd methods for class MF:

```

# return current loss value
def loss(self):
    L = 0
    for i in range(self.n_ratings):
        # user_id, item_id, rating
        n, m, rating = int(self.Y[i, 0]), int(self.Y[i, 1]), self.Y[i, 2]
        L += 0.5*(self.X[m].dot(self.W[:,n]) + self.b[m] + self.d[n] - rating)**2
    L /= self.n_ratings
    # regularization, don't ever forget this
    return L + 0.5*self.lam*(np.sum(self.X**2) + np.sum(self.W**2))

```

```

def updateWd(self):
    for n in self.users_ids:
        # get all items rated by user n, and the corresponding ratings
        ids = np.where(self.Y[:, 0] == n)[0]
        item_ids, ratings = self.Y[ids, 1].astype(np.int32), self.Y[ids, 2]
        Xn, bn = self.X[item_ids], self.b[item_ids]
        for i in range(30): # 30 iteration for each sub problem
            wn = self.W[:, n]
            error = Xn.dot(wn) + bn + self.d[n] - ratings
            grad_wn = Xn.T.dot(error)/self.n_ratings + self.lam*wn
            grad_dn = np.sum(error)/self.n_ratings
            # gradient descent
            self.W[:, n] -= np.array(self.learning_rate*grad_wn.reshape(-1))[0]
            self.d[n] -= self.learning_rate*grad_dn

```

```

def updateXb(self):
    for m in self.items_ids:
        ids = np.where(self.Y[:, 1] == m)[0] # row indices of items m
        user_ids, ratings = self.Y[ids, 0].astype(np.int32), self.Y[ids, 2]
        Wm, dm = self.W[:, user_ids], self.d[user_ids]
        Wm = Wm.reshape(Wm.shape[0], Wm.shape[1])
        for i in range(30): # 30 iteration for each sub problem
            xm = self.X[m]
            error = Wm.T.dot(xm).reshape(-1,1) + self.b[m] + dm - ratings
            grad_xm = Wm.dot(error)/self.n_ratings + (self.lam*xm).reshape(-1,1)
            grad_bm = np.sum(error)/self.n_ratings
            # gradient descent
            self.X[m] -= np.array((self.learning_rate*grad_xm).T)[0]
            self.b[m] -= self.learning_rate*grad_bm

```

The next part is the main optimization process of MF (fit), predicting the new rating (pred) and evaluating the model quality by root-mean-square error (evaluate_RMSE).

```
def fit(self):
    for it in range(self.max_iter):
        self.updateXb()
        self.updateWd()
        if (it+1) % self.print_every == 0:
            rsme_train = self.evaluate_RMSE(self.Y)
            # print("iter = ",it+1 ,", loss = "+self.loss()+", RMSE train =
            ",rsme_train)
            print(it+1)
            print(self.loss())
            print(rsme_train)
            print(self.evaluate_RMSE(rate_test))
```

```
def pred(self, u, i):
    # predict the rating of user u for item i
    u, i = int(u), int(i)
    try:
        pred = self.X[i, :].dot(self.W[:, u]) + self.b[i] + self.d[u]
    except:
        return 0
    return max(0, min(5, pred))

def evaluate_RMSE(self, rate_test):
    n_tests = rate_test.shape[0]
    SE = 0
    for n in range(n_tests):
        pred = self.pred(rate_test[n, 0], rate_test[n, 1])
        SE += (pred - rate_test[n, 2])**2

    RMSE = np.sqrt(SE/n_tests)
    return RMSE
```

At this point, we have built in the MF class with the necessary methods. Next, we check the model quality when it is applied to the 100k MoviesLens dataset.

```

r_cols = ['user_id', 'item_id', 'rating', 'unix_timestamp']
ratings_base = pd.read_csv('ml-100k/ua.base', sep='\t', names=r_cols)
ratings_test = pd.read_csv('ml-100k/ua.test', sep='\t', names=r_cols)

rate_train = np.asmatrix(ratings_base)
rate_test = np.asmatrix(ratings_test)

# indices start from 0
rate_train[:, :2] -= 1
rate_test[:, :2] -= 1

rs = MF(rate_train, K = 50, lam = .01, print_every = 5, learning_rate =
50, max_iter = 30)
rs.fit()

```

Result:

```

5 0.9813242949856356 0.9576538777513495 1.0334281333969082
10 0.9517149245758885 0.927083129976136 0.9825093531726448
15 0.9439737011472505 0.9189954890904248 0.971302106084739
20 0.9407339857967102 0.9156333436753306 0.9675119236760502
25 0.9390274877027561 0.9138814638591751 0.9658114878351727
30 0.938001148320693 0.9128350514306869 0.9648862591718131

```

The obtained RMSE is 0.9648, better than the NBCF in the previous chapter (0.9688).

2.3 Review on MFCF

- Nonnegative matrix factorization. When the data is not normalized, they all carry non-negative values. Even in the case that the rating range contains negative values, we only need to add a reasonable value to the utility matrix to get ratings that are non-negative. At that time, another matrix factorization method with additional constraints is also widely used and highly effective in the recommendation system, which is nonnegative matrix factorization (NMF), ie the analysis of the performance matrix of matrices with elements non-negative.
- Through matrix factorization, users and items are linked together by hidden features. The association of each user and item to each hidden feature is measured by the corresponding component in their feature vector, the larger the value representing the greater the relevance of the user or item to that hidden feature. Intuitively, the relevance of a user or item to a hidden feature should be a non-negative number with a value of 0 indicating non-relevance. Furthermore, each user and item is only associated with certain

hidden features. Therefore, the feature vectors for user and item should be non-negative vectors and have lots of zero values. These solutions can be obtained by adding non-negative constraints to the components of X and W . This is the origin of the idea and name of nonnegative matrix factorization.

- Incremental matrix factorization. As mentioned, the prediction time of a recommendation system using matrix factorization is very fast, but the training time is quite long with large data sets. In fact, the utility matrix changes constantly because there are more users, items as well as new ratings or users want to change their rating, so the model parameters must also be regularly updated. This means that we must continue to perform the training process, which takes a lot of time. This is partially solved by incremental matrix factorization. The word incremental can be understood as a small adjustment to the data.

2.4 Advantage of MFCF

Matrix factorization-based collaborative filtering algorithms can be easily saved in a database by storing the learned latent factor matrices. These matrices can be saved as tables or arrays in a relational database, or as collections in a NoSQL database. It is important to consider the size of the matrices, as well as the memory and storage limitations of the database, when determining the best method for saving the results. In some cases, it may be necessary to reduce the dimensionality of the latent factor matrices to reduce memory usage and speed up the storage and retrieval process.

From the Development in Python section above, we need to save these 4 vectors to database:

- X for items
- W for users
- b for item biases
- d for user biases

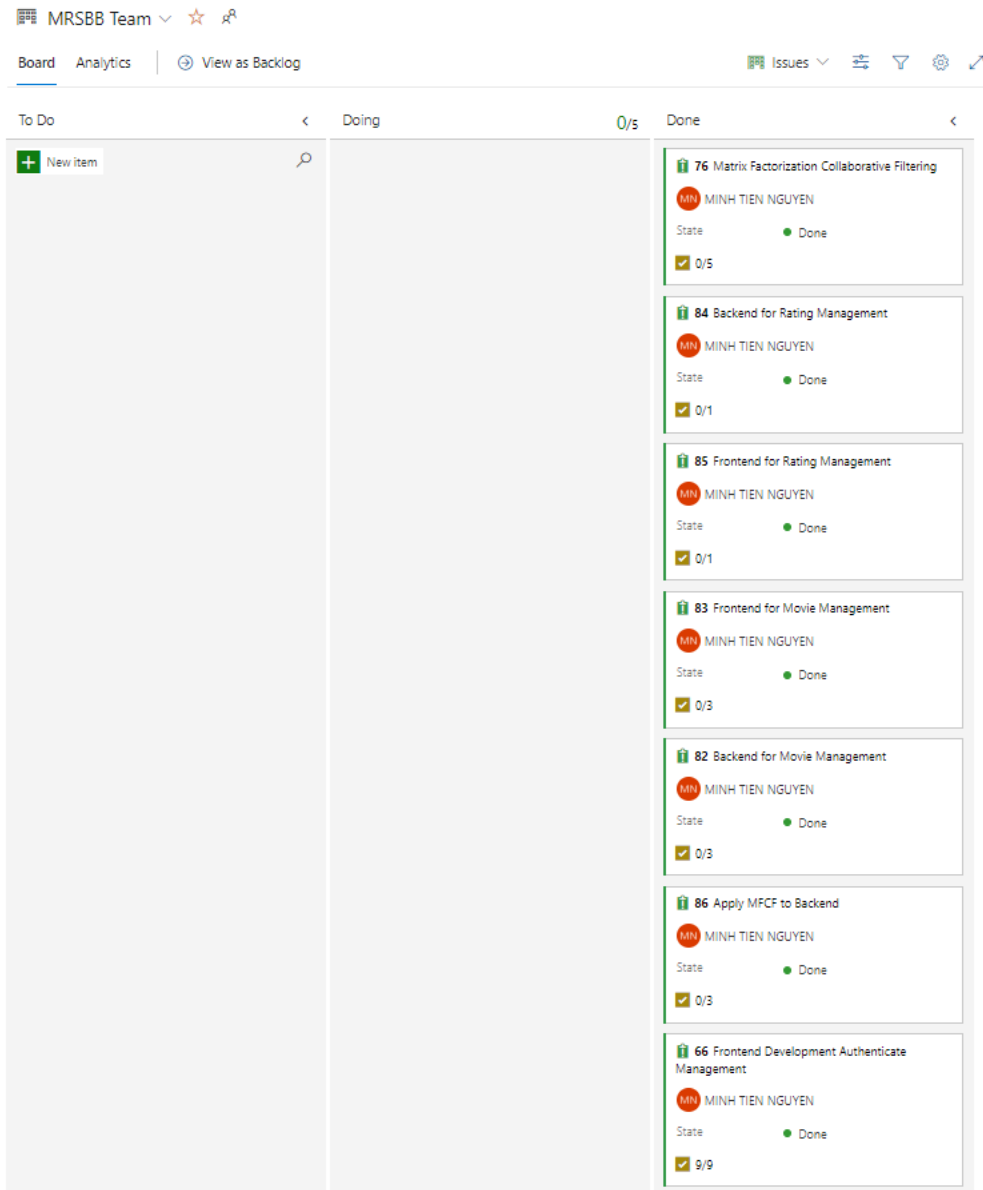
2.5 Import Data to Database

All the code to clean data is in the file `MovieLens/loader.ipynb`

After that, we get 6 files that need to be imported to the database in the data folder. We import to PostgreSQL with this command:

```
\copy "MRSBB_rating" FROM '/ratings.csv' csv header;  
\copy "MRSBB_movie" FROM '/indexmoviesWithPosters.csv' csv header;  
\copy "MRSBB_xmodels" FROM '/XXI.csv' csv header;  
\copy "MRSBB_wtmodels" FROM '/WWI.csv' csv header;  
\copy "MRSBB_bmodels" FROM '/bbI.csv' csv header;  
\copy "MRSBB_dmodels" FROM '/ddI.csv' csv header;
```

3. Sprint Review



The software is basically fully operational with the functions that I have analyzed in the Requirements Analysis section.

4. Sprint Retrospective

Perhaps if there are more requests from customers in the future, the Backend will have to change the framework to be able to customize it more deeply as well as master the logic thereby limiting security issues even though Django is supported by the community quite well. good but maybe one day there will be a zero-day vulnerability.