# System Verification and Validation Plan for Recommender System

Seyed Ali Mousavi

April 15, 2024

## Revision History

| Date | Version | Notes |
|---|---|---|
| 2024-02-19 | 1.0 | Initial Release |
| 2024-03-02 | 2.0 | Second Draft |
| 2024-04-15 | 3.0 | Final Release |

# Contents

# 1 General Information

This section will provide the background and objectives for this document.

## 1.1 Summary

This project is building a recommendation system based on the MovieLens dataset, i.e. a Movie Recommender. Our movie recommender system utilizes a collaborative filtering method. Collaborative methods for recommender systems are methods that are based solely on the past interactions recorded between users and items to produce new recommendations. These interactions are stored in the "user-item interactions matrix". The rows in the matrix represent users indices and the columns represent item indices (movies, in our case). So the entry $ij$ of the matrix represents the rating of the user $i$ for the movie $j$ (an integer number between 1 to 5). The main idea that rules collaborative methods is that these past user-item interactions are sufficient to produce good enough recommendations. After receiving a specified user of interest for whom we want to make recommendations, the ANN or KNN method(please refer to the SRS document for definition and discussion) algorithm is executed to generate the top recommendations.

## 1.2 Objectives

The objectives outlined in this VnV are as follows:

- Validate the software's capability to produce good recommendations (according to performance metrics in subsequent sections).
- Verify the proper functioning of the Movie Recommender.
- Confirm the correctness and maintainability of the software

I avoid conducting usability testing for the Movie Recommender as this project lacks a defined user interface, and the required inputs are too straightforward to warrant user testing.

## 1.3 Relevant Documentation

The reader of this document will be referred to the other documents of this project (SRS MG, MIS) whenever it's needed.

## 2 Plan

This section outlines the approach for validating both the documentation and software of the Movie Recommender. The main elements scheduled for validation include the Software Requirements Specification (SRS), design, Verification and Validation (VnV) plan, and implementation.

### 2.1 Verification and Validation Team

The team members and their responsibilities are specified in Table 1.

| **Name** | **Document** | **Role** | **Description** |
|---|---|---|---|
| Seyed Ali Mousavi | All | Author | Generate and oversee document creation, develop the VnV plan, conduct VnV testing, and validate the implementation. |
| Dr. Spencer Smith | All | Instructor/Reviewer | Evaluate the documents for design and documentation style. |
| Nada Elmasry | All | Domain Expert Reviewer | Review all the documents. |
| Xinyu Ma | SRS | Secondary Reviewer | Review the SRS document |
| Valerie Vreugdenhil | VnV Plan | Secondary Reviewer | Review the VnV plan. |
| Gaofeng Zhou | MG + MIS | Secondary Reviewer | Review the MG and MIS document. |

Table 1: Verification and Validation Team

## 2.2 SRS Verification Plan

The SRS document will be verified in the following way:

1. The assigned members (Dr. Spencer Smith, Nada Elmasry, and Xinyu Ma) will conduct the initial review. This will involve a manual assessment using the provided SRS Checklist, along with additional checklists gathered from various sources. SRS Checklist and following additional checklist are gathered from here.

    - Do all requirements strictly pertain to requirements, and not to design or implementation solutions?
    - Has each requirement been uniquely and accurately identified?
    - Is each requirement verifiable by testing, demonstration, review, or analysis?
    - Do any requirements conflict with or duplicate other requirements?
    - Is each requirement in scope for the project?
    - Are all requirements written at a consistent and appropriate level of detail?
    - Do the requirements provide an adequate basis for design?

2. The reviewers will provide feedback to the author by creating an issue on GitHub.

3. Seyed Ali Mousavi, the author, is responsible for addressing the issues.

## 2.3 Design Verification Plan

The assigned members (Dr. Spencer Smith, Xinyu Ma, and Gaofeng Zhou) will conduct the initial review. This will involve a manual assessment using the provided checklist.

- Does the design meet all specified requirements outlined in the design specification document?
- Are all interfaces between design components specified clearly?
- Are all components of the design testable separately?

- Is the design scalable to accommodate future growth or changes in requirements?
- Are the design components separated as much as possible?
- Does each design component have a logical task?

## 2.4 Verification and Validation Plan Verification Plan

The specified members (Dr. Spencer Smith, Xinyu Ma, and Valerie Vreugdenhil) will perform the initial review, which will involve manually assessing the document using the following checklist.

- Does the VnV plan verify all the functional and non-functional requirements?
- Are all input and output pairs for test cases correct?
- Are roles and responsibilities clearly defined for individuals or teams involved in verification and validation activities?

## 2.5 Implementation Verification Plan

The Movie Recommender project will utilize the Python programming language for development purposes. Moreover, code walkthroughs involving domain expert Nada Elmasry will be conducted. Dynamic testing will include both system and unit level testing, elaborated further in sections 3 and 4.

## 2.6 Automated Testing and Verification Tools

- Unittest: The unittest library in Python is a built-in testing framework that allows you to write and run tests for your Python code. It provides a set of classes and methods to create and execute test cases, allowing you to verify that your code behaves as expected.
- Pytest: A popular testing framework for Python that simplifies the process of writing and executing test code. It allows developers to write simple and scalable test cases.

- GitHub: This platform will be used for version control and collaboration.
- GitHub CI workflow: This will automate regression tests and checks that Movie Recommender builds are passing before code is merged into a protected branch.

## 2.7 Software Validation Plan

Obtaining genuine data to validate the Movie Recommender requires a substantial number of real users, ideally exceeding 100, along with a considerable amount of feedback. However, given the limitations of this project, acquiring such extensive real-world data is not feasible.

# 3 System Test Description

This section will outline the tests aimed at verifying that the Movie Recommender fulfills the functional and non-functional criteria outlined in the SRS document.

## 3.1 Tests for Functional Requirements

The code for the following tests is in `recommendation_tests.py` file in the `test` folder of this project.

### 3.1.1 One Popular Movie

Suppose we have a user-item interaction matrix like below:

The final row represents the target user, whom we aim to provide recommendations for. It's evident from the user feedback, based on the ratings, that the last movie is the highest-rated, while all other movies are rated significantly lower. Therefore, it's clear that if the Movie Recommender is expected to generate a single recommendation, it should unequivocally be the last movie(test-id1). Additionally, we'll modify the ID of both the target user and the target movie (the popular one) to guarantee that the recommendation remains unaffected by their placement in the user-item interaction matrix.(test-id2)

$$\begin{array}{cc} & \textbf{MOVIES} \\ \textbf{USERS} & \begin{pmatrix} 1 & 1 & 1 & \dots & 1 & 5 \\ 1 & 1 & 1 & \dots & 1 & 5 \\ 1 & 1 & 1 & \dots & 1 & 5 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & 1 & \dots & 1 & 5 \\ 1 & 1 & ? & \dots & ? & ? \end{pmatrix} \end{array}$$

Figure 1: User-Item interaction Matrix for test-case 1

### One Popular Movie

1. test-one-popular-movie-id1

   Control: Manual

   Initial State: N/A

   Input: User-Item Interaction Matrix specified in the figure, in addition to the target user's index number

   Output: The last movie Id

   How the test will be performed: I manually construct such a user-item matrix and enter the target user id.

2. test-one-popular-movie-id2

   Control: Manual

   Initial State: N/A

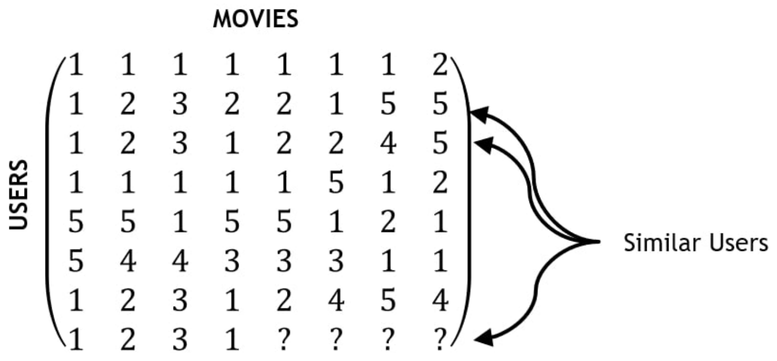
   Input: The modified User-Item Interaction Matrix (with the random location of the target user and target movie), in addition to the target user's index number. In this case I chose 7 for the target movie id and 5 for the target user id.

   Output: The target movie Id

How the test will be performed: I manually construct such a user-item matrix and enter the target user id.

### 3.1.2 Artificial Close Users

Suppose we have a user-item interaction matrix like below:



$$\begin{array}{c|c}
 & \text{MOVIES} \\
\hline
\text{USERS} & \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 \\
1 & 2 & 3 & 2 & 2 & 1 & 5 & 5 \\
1 & 2 & 3 & 1 & 2 & 2 & 4 & 5 \\
1 & 1 & 1 & 1 & 1 & 5 & 1 & 2 \\
5 & 5 & 1 & 5 & 5 & 1 & 2 & 1 \\
5 & 4 & 4 & 3 & 3 & 3 & 1 & 1 \\
1 & 2 & 3 & 1 & 2 & 4 & 5 & 4 \\
1 & 2 & 3 & 1 & ? & ? & ? & ?
\end{pmatrix}
\end{array}$$

Figure 2: User-Item interaction Matrix for test-case 2

Once more, the final row represents the target user, for whom we strive to offer recommendations. While it might not be immediately apparent, user2, user3, and the target user exhibit similar feedback vectors, indicating their proximity in terms of preferences. Conversely, the other users are noticeably distant from the target user. Consequently, we anticipate that the recommendation should include the favorite movies of user2 and user3, which happen to be the last movie.

#### Artificial Close Users

1. test-Artificial Close Users-id3

   Control: Manual

   Initial State: N/A

   Input: User-Item Interaction Matrix specified in the figure, in addition to the target user's index number

   Output: The Id of the last movie

How the test will be performed: I manually construct such a user-item matrix and enter the target user id.

## 3.2 Tests for Nonfunctional Requirements

### 3.2.1 Accuracy

RMSE (Root Mean Square Error) and MAE (Mean Absolute Error) are two common methods used to evaluate the accuracy of rating prediction in recommender systems. RMSE measures the square root of the average of the squared differences between the predicted ratings and the actual ratings and MAE measures the average of the absolute differences between the predicted ratings and the actual ratings.

#### Accuracy

1. test-accuracy-id4

   Type: Manual

   Initial State: N/A

   Input/Condition: The original (MovieLens) user-item interaction matrix.

   Output/Result: RMSE and MAE of the Movie Recommender for one of the users.

   How the test will be performed:

   I've selected the initial user (userId 1) to carry out the test. Given the dataset, we're aware that we possess ratings from this user for the first 50 movie IDs. Subsequently, we utilize the Movie Recommender to forecast the rating for each of these movies. (It's crucial to note that we have the actual ratings for assessing performance.) Afterwards, we present the corresponding RMSE and MAE values. To assess our RMSE and MAE, we'll generate a random list of predicted ratings ranging from 1 to 5. Subsequently, we'll calculate the RMSE and MAE for this dataset. We anticipate that Movie Recommender will exhibit considerably fewer errors in comparison.

### 3.3 Traceability Between Test Cases and Requirements

The following table shows which test cases are supporting which requirements.

| | R1 | R2 | R3 |
|---|---|---|---|
| test-id1 | | X | X |
| test-id2 | | X | X |
| test-id3 | | X | X |
| test-id4 | | | |

Table 2: Relation of Test Cases and Requirements.

## 4 Unit Test Description

Unit testing is a fundamental practice in software development for ensuring the functionality and reliability of individual components or units of code. I employed automated test cases from the `unittest` framework as my approach to implement unit testing. I endeavored to ensure that both the test code and the entire codebase are well-documented and feature meaningful names, facilitating ease of understanding for the reader.

### 4.1 Unit Testing Scope

My project doesn't include testing for `Numpy` and `Pandas` libraries as I used them in Movie Recommender, and `sklearn` library as I used it in one of my test cases.

### 4.2 Tests for Functional Requirements

Most of the verification will be through automated unit testing.

#### 4.2.1 Module Input Format

I used `'unittest'` module for constructing and running test cases. Please refer to the `Input_Format_module_test.py` in the `test` folder of the project to see and run the test code.

### 4.2.2 Module Movie Selector

I used `'unittest'` module for constructing and running test cases. Please refer to the `Movie_Selector_module_test.py` in the `test` folder of the project to see and run the test code.

### 4.2.3 Module KNN

I used `'pytest'` module for constructing and running test cases. Please refer to the `KNN_module_test_functional.py` in the `test` folder of the project to see and run the test code. (you need to write `pytest` in the terminal to run the test)

## 4.3 Tests for Nonfunctional Requirements

The performance of our KNN module needs to be evaluated independently. Please refer to the `KNN_module_test_non_functional.py` in the `test` folder of the project to see and run the test code.

### 4.3.1 Module KNN

1. test-id1

   Type: automatic

   Initial State: -

   Input/Condition: The iris dataset. You can find information about the iris dataset and access it through the following link: Iris Dataset

   Output/Result: the output of the KNN module must match the output of the knn module from the Scikit-Learn Library.

   How the test will be performed: We choose a training set from the iris dataset and utilize it to predict labels for the test set twice: once with our KNN module and another time with the KNN module imported from the Scikit-Learn library. The resulting predictions should be identical.

## 4.4 Traceability Between Test Cases and Modules

I have written code for testing all the modules involved in this project. Please refer to the `test` folder of the project to see all the tests. (`recommendation_tests.py` for test cases mentioned in Section 3)