Project Title: System Verification and Validation Plan for Recommender System

Seyed Ali Mousavi

February 19, 2024

Revision History

Date	Version	Notes
2024-02-19	1.0	Initial Release

Contents

1 General Information

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 so-called "user-item interactions matrix". The rows in the matrix represent users indices and the columns represent items 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 (target user index) for whom we want to make recommendations, the ANN or KNN method 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 referenced to the other documents of this project(for instance, SRS MG, MIS, etc.) whenever it's needed.

2 Plan

This section will delineate the strategy for validating both the documentation and software of the Movie Recommender. The primary components slated for validation comprise 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.

Table 1: Verification and Validation Team

Name	Document	Role	Description
Seyed Ali Mousavi	All	Author	Create and manage all the documents, create the VnV plan, perform the VnV testing, verify the implementation.
Dr. Spencer Smith	All	Instructor/Reviewer	Review the documents, 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.

2.2 SRS Verification Plan

The SRS document will be verified in the following way:

1. Initial review will be performed by the assigned members (Dr. Spencer Smith, Nada Elmasry, and Xinyu Ma). For this, a manual review will be conducted using the provided SRS Checklist and following additional checklist gather from here.

   - Are all requirements actually requirements, not design or implementation solutions?
   - Is each requirement uniquely and correctly 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

Initial review will be performed by the assigned members (Dr. Spencer Smith, Xinyu Ma, and Gaofeng Zhou). For this, a manual review will be conducted using the following 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 designated members (Dr. Spencer Smith, Xinyu Ma, and Valerie Vreugdenhil) will conduct the initial review. This review will entail a manual assessment utilizing 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

Movie Recommender will be developed using the Python programming language. For static testing, we will use Flake8. Flake8 is a static code analysis tool for Python that checks code against coding style (PEP 8), detects syntax errors, and identifies potential bugs or problematic constructs. Additionally, we conduct code walkthroughs with domain expert (Nada Elmasry). For dynamic testing, we employ system and unit level testing, as explained in detail in sections 3 and 4.

2.6 Automated Testing and Verification Tools

- Pytest: A unit testing framework for Python that allows for easy and scalable testing of Python code.
- Flake8: A static code analysis tool for Python that checks code against coding style (PEP 8), detects syntax errors, and identifies potential bugs or problematic constructs.

- GitHub: This platform will be used for version control and collaboration.
- GitHub CI workflow: This will automate regression tests and checks that builds are passing before code is merged into a protected branch.
- Docker: A mechanism used to containerize applications, ensuring their installability and understandability throughout the project.

2.7 Software Validation Plan

No plans for Software Validation yet.

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 One Popular Movie

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

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

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

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.

One Popular Movie

1. test-popular movie-id1

   Control: Manual

   Initial State: N/A

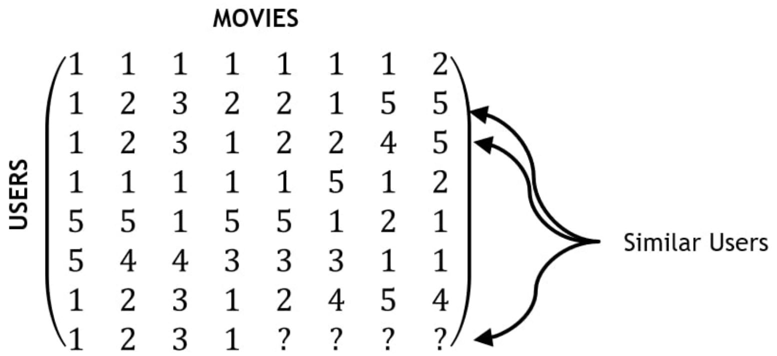
   Input: User-Item Interaction Matrix specified in the figure, in addition to the target user's index number

   Output: The Last Movie Id

   How test will be performed:

3.1.2 Artificial Closed Users

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



$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 \\ 1 & 2 & 3 & 2 & 2 & 1 & 5 & 5 \\ 1 & 2 & 3 & 1 & 2 & 2 & 4 & 5 \\ 1 & 1 & 1 & 1 & 1 & 5 & 1 & 2 \\ 5 & 5 & 1 & 5 & 5 & 1 & 2 & 1 \\ 5 & 4 & 4 & 3 & 3 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 & 2 & 4 & 5 & 4 \\ 1 & 2 & 3 & 1 & ? & ? & ? & ? \end{pmatrix}$$

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 movies.

Artificial Closed Users

1. test-Artificial Closed Users-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 Id of the last two movies

   How test will be performed:

3.2 Tests for Nonfunctional Requirements

3.2.1 Accuracy

Various metrics can gauge the performance of a Recommender system, with one commonly used metric being Recall@K. This metric measures how many items in the top K recommendations match those in the user's actual preferences, where K represents the number of recommendations provided to the user. For example, suppose we have rating feedback from a user for 50 movies, and we aim to calculate Recall@10. We divide the list of viewed movies into two sublists, one with 30 movies serving as the basis for generating new recommendations, and the other with 20 movies acting as the test set. Following the generation of recommendations, we compare them to the top 10 movies in the test list (as per the actual ratings). The higher the overlap between these lists, the better the performance of our recommender. Recall@10 is computed as the number of common movies (between the top 10 movies in the recommendation list and the top 10 movies according to the real ratings) divided by the number of recommendations, K (where $K = 10$).
After obtaining the Recall@10 for my Movie Recommender, I'll proceed to compare it with that of TFRS (TensorFlow Recommender System), an open-source Python library renowned for constructing cutting-edge recommenders. Currently, I deem a relative error of approximately 50 percent to be within an acceptable range for comparison purposes.

Accuracy

1. test-accuracy-1

   Type: Manual

   Initial State: N/A

   Input/Condition: a User-Item interaction matrix with removed 40 percent of ratings of the user of interest.

   Output/Result: Recall@10 for Movie Recommender and Recall@10 for the TFRS.

   How the test will be performed:

3.3 Traceability Between Test Cases and Requirements

Provide a table that shows which test cases are supporting which requirements.

4 Unit Test Description

This section should not be filled in until after the MIS (detailed design document) has been completed.

Reference your MIS (detailed design document) and explain your overall philosophy for test case selection.

To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here, you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests.

4.1 Unit Testing Scope

What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance.

4.2 Tests for Functional Requirements

Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section.

4.2.1 Module 1

Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected.

(a) test-id1

    Type: Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic

    Initial State:

    Input:

    Output: The expected result for the given inputs

    Test Case Derivation: Justify the expected value given in the Output field

    How test will be performed:

(b) test-id2

    Type: Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic

Initial State:

Input:

Output: The expected result for the given inputs

Test Case Derivation: Justify the expected value given in the Output field

How test will be performed:

(c) ...

4.2.2 Module 2

...

4.3 Tests for Nonfunctional Requirements

If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant.

These tests may involve collecting performance data from previously mentioned functional tests.

4.3.1 Module ?

(a) test-id1

Type: Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic

Initial State:

Input/Condition:

Output/Result:

How test will be performed:

(b) test-id2

Type: Functional, Dynamic, Manual, Static etc.

Initial State:
Input:
Output:
How test will be performed:

4.3.2 Module ?

...

4.4 Traceability Between Test Cases and Modules

Provide evidence that all of the modules have been considered.

References

5 Appendix

This is where you can place additional information.

5.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

5.2 Usability Survey Questions?

This is a section that would be appropriate for some projects.

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. Please answer the following questions:

Appendix — Reflection

This section is not required for CAS 741

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. Please answer the following questions:

(a) What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage etc. You should look to identify at least one item for each team member.

(b) For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?