Verification and Validation Report: TTE RecSys

Yinying Huo

April 4, 2025

1 Revision History

Date	Version	Notes
Apr. 2 2025	1.0	First draft

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test
R	Requirement
NFR	Nonfunctional Requirement
TTE	Two Tower Embedding
RecSys	Recommendation System
ANN	Approximate Nearest Neighbor
SRS	Software Requirements Specification
FAISS	Facebook AI Similarity Search
DNN	Deep Neural Network

Contents

List of Tables

This document includes the results of VnV plan.

3 Functional Requirements Evaluation

The functional requirements are tested using both system tests and unit tests.

3.1 Dataset

To ensure the project receives a valid dataset for training:

Manually, a Jupyter Notebook (data_preprocessing.ipynb) is used to merge the raw data (three CSV files) into a single CSV file.

Then, the automated system test (test_data_validation.py) runs each time the dataset is updated before training starts to ensure that the input CSV file contains all the required columns.

Additionally, an automated unit test (test_data_processing.py) runs on each push to verify the correctness of the data processor, which generates new features for users and items.

All tests have passed successfully.

3.2 Model Training Convergence

The system test (test_model_convergence.py) checks whether the training loss decreases as training progresses.

This test has passed successfully.

3.3 Model storage

The system test (test_model_storage.py) runs each time model training is completed. The test ensures that the trained model and computed embeddings are correctly stored in the 'output' folder.

This test has passed successfully, confirming that:

- The user model (user_model.pth) is saved with the correct state dictionary format
- The item model (item_model.pth) is saved with the correct state dictionary format

- Item embeddings (item_embeddings.npy) are saved with the expected shape
- The ANN index is properly saved and can be loaded for inference

3.4 Embedding Generation

The unit test 'test_embedding_generation.py' verifies the correctness of embedding generation. This test has passed successfully, confirming that:

- The embedding generator correctly initializes with both user and item models
- Single user/item embeddings are generated with the expected dimensions
- All embeddings are properly normalized (unit norm)

4 Nonfunctional Requirements Evaluation

4.1 Usability

The usability of the system was evaluated using a survey as outlined in the VnV Plan.

4.2 Reliability

Reliability testing was conducted through both system tests and unit tests. The system demonstrated robust performance with the following results:

- All functional tests (test-id1, test-id2, test-id3) passed.
- Data validation tests showed 100% detection rate for improperly formatted inputs
- The system correctly handled edge cases such as:
  - Users with no location information
  - Extreme age values

- Recovery from invalid input was graceful, with appropriate error messages

The system meets the reliability requirement (NFR2) by performing as expected and handling edge cases appropriately.

4.3 Protability

The system's portability was tested across multiple operating systems as specified in NFR3:

5 Unit Testing

Unit tests were developed for five key modules. All unit tests were executed with pytest and integrated into the continuous integration pipeline.

5.1 Data Processing Module (M2)

The data processing module tests in 'test_data_processing.py' verify:

- Data loading functionality with both valid and invalid data files
- Data validation correctly identifies valid and invalid dataset formats
- Missing value handling ensures no NaN values
- Training data creation produces correctly structured input for the model
- Book mapping function correctly maps encoded IDs to book metadata

All tests passed successfully, indicating the data processing module functions as expected.

5.2 Embedding Generation Module (M4)

The embedding generation tests in 'test_embedding_generation.py' verify:

- Initialization with compatible and incompatible models
- User embedding generation for both single users

- Item embedding generation for both single items
- All generated embeddings are properly normalized
- Large batch handling for item embeddings

All tests passed successfully, confirming the embedding generation module meets its requirements.

5.3 Neural Network Module (M6)

The neural network tests in ‘test_neural_network.py‘ verify:

- Tower network initialization with correct architecture
- Forward pass produces normalized outputs of the expected dimension
- User tower creation function produces correct models
- Item tower creation function produces correct models
- Weight initialization produces non-zero weights with expected properties
- Different inputs produce different embeddings

All tests passed successfully, confirming the neural network module creates models with the expected behavior.

5.4 ANN Search Module (M7)

The ANN search tests in ‘test_ann_search.py‘ verify:

- Building a flat index with proper structure
- Exact match search correctly finds known vectors
- Approximate match search finds vectors similar to the query
- Multiple results retrieval with the correct number of candidates
- Error handling for empty or mismatched inputs
- Saving and loading indices preserves search functionality

All tests passed successfully, validating the ANN search module's ability to perform efficient vector similarity search.

5.5 Vector Operations Module (M8)

The vector operations tests in ‘test_vector_operations.py‘ verify:

- Basic dot product calculation with different vector formats
- Handling of empty vectors
- Orthogonal, same-direction, and opposite-direction vectors
- Handling of large and small values
- Error detection for dimension mismatches

All tests passed successfully, confirming the vector operations module correctly implements mathematical operations needed for similarity calculations.

6 Changes Due to Testing

Throughout the development process, many minor bugs and formatting issues in the document were fixed.

The major change was due to the poor performance (accuracy) of the DNN model, which resulted from the limited data. The quality of the recommendation results was poor because the dataset contained very few user features. Additionally, many items received ratings from only a single user, and the item features were also limited. This led to embedding collapse, meaning the model generated nearly identical embeddings for different inputs.

As a result, the system recommended very similar books to each different user.

After applying techniques such as feature engineering and negative sampling, the issue improved but still persisted. Due to time constraints, I was unable to implement more advanced techniques. Therefore, I had to remove the performance requirements.

7 Automated Testing

The project utilizes GitHub Actions for automated testing and continuous integration. The automated testing workflow follows a structured trigger-based approach:

- All unit tests run automatically whenever there is a change to any file in the `src` directory
- The dataset validation test (`test_data_validation.py`) executes whenever there are changes to the dataset files
- When dataset changes are detected and validation passes, the model retraining process is triggered automatically
- After model retraining completes, the model convergence test (`test_model_convergence.py`) and model storage test (`test_model_storage.py`) run sequentially to verify the new model

This automation pipeline ensures that code changes are continuously validated, data integrity is maintained, and the model quality is verified after each training cycle.

8 Trace to Requirements

The following table shows the traceability between tests and functional requirements:

The test coverage demonstrates that each functional requirement is verified by at least one test, ensuring comprehensive validation of system functionality.

For non-functional requirements:

- NFR1 (Usability): Verified through user surveys
- NFR2 (Reliability): Verified through system and unit tests.
- NFR3 (Portability): Verified by manually installing and running the system on different computers.

Test ID	R1	R2	R3	R4	R5	R6
test-id1 (Dataset Validation)	X					
test-id2 (Model Convergence)		X		X	X	
test-id3 (Model Storage)			X			
test-data-processing	X					
test-embedding-generation				X		
test-neural-network		X				
test-ann-search					X	X
test-vector-operations					X	X

Table 1: Traceability Matrix Between Tests and Requirements

9 Trace to Modules

The following table shows the traceability between tests and modules:

Test ID	M1	M2	M3	M4	M5	M6	M7	M8
test-id1 (Dataset Validation)		X						
test-id2 (Model Convergence)			X			X		
test-id3 (Model Storage)	X							
test-data-processing		X						
test-embedding-generation				X	X			
test-neural-network						X		
test-ann-search					X		X	
test-vector-operations					X			X

Table 2: Traceability Matrix Between Tests and Modules

Each module in the system is covered by at least one test, providing confidence in the correctness of individual components as well as their integration.

10 Code Coverage Metrics

Code coverage metrics were collected using pytest-cov during the automated testing process. The following table summarizes the coverage results for unit tests:

Module	**statements**	**missing**	**excluded**	**coverage**
modules/data_processing.py	119	17	0	86%
modules/embedding_generation.py	58	0	0	100%
modules/neural_network.py	28	0	0	100%
modules/ann_search.py	82	17	0	79%
modules/vector_operations.py	14	5	0	64%
Overall	301	39	0	87%

Table 3: Code Coverage

There is no unit testing for `main.py`, `model_training.py`, `recommendation.py`, `system_interface.py`, and `user_interface.py` due to limited time.

However, the modules `model_training.py`, `recommendation.py`, and `system_interface.py` are inherently tested through system tests and by users running the system using `user_interface.py`.

The code coverage report is also available as an artifact for download in GitHub Actions after each run of all unit tests.

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Reflection.

1. What went well while writing this deliverable?

2. What pain points did you experience during this deliverable, and how did you resolve them?

3. Which parts of this document stemmed from speaking to your client(s) or a proxy (e.g. your peers)? Which ones were not, and why?

4. In what ways was the Verification and Validation (VnV) Plan different from the activities that were actually conducted for VnV? If there were differences, what changes required the modification in the plan? Why did these changes occur? Would you be able to anticipate these changes in future projects? If there weren't any differences, how was your team able to clearly predict a feasible amount of effort and the right tasks needed to build the evidence that demonstrates the required quality? (It is expected that most teams will have had to deviate from their original VnV Plan.)