INDIAN INSTITUTE OF TECHNOLOGY MADRAS ZANZIBAR

School of Science and Engineering

Z5007: Programming and Data Structures

M.Tech Data Science & Artificial Intelligence

Milestone 2

Recommendation System – Collaborative Filtering Recommender (Item-based)

Submitted by:

Student 1:	Surabhi Gudla ZDA25M001
Email:	zda25m001@iitmz.ac.in
Student 2:	Vineet Joshi ZDA25M007
Email:	zda25m007@iitmz.ac.in
Instructor:	Dr. Innocent Nyalala
Submission Date:	December 23, 2025

Contents

1. Introduction

Recommendation systems play a crucial role in modern digital platforms by helping users discover relevant content from large catalogs of items. Popular platforms such as Netflix, Amazon, and Spotify rely heavily on collaborative filtering techniques to personalize user experiences. Collaborative filtering leverages collective user behavior to infer preferences, making it one of the most widely adopted approaches in real-world systems.

This project focuses on implementing an Item-Based Collaborative Filtering (IBCF) recommendation system using fundamental data structures, as required by the course objectives. Instead of relying on high-level machine learning libraries, the system is built from scratch using hash tables, graphs, and priority queues to ensure conceptual clarity and algorithmic transparency.

This document reports the progress achieved up to Milestone-2, including completed components, implemented data structures, algorithms, challenges faced, and remaining work.

2. Progress Summary

At this stage, the core foundation of the recommendation system has been successfully implemented. The dataset is loaded and preprocessed, and user–item as well as item–user relationships are stored using hash tables. Item–item similarity computation using cosine similarity has been implemented, and the similarity relationships are stored in a sparse adjacency list representation.

Approximately 65–70% of the project has been completed at this milestone.

Completed Work:

- Loading and preprocessing of MovieLens dataset
- Train–test split (80% training, 20% testing)
- Construction of user–item and item–user hash tables
- Implementation of cosine similarity for item–item comparison
- Construction of a sparse item–item similarity graph
- Performance measurement of similarity graph construction
- Generation of meaningful intermediate outputs and sample results

Current Status of Components:

Data Loading & Preprocessing:	Completed
Hash Table Construction:	Completed
Item–Item Similarity Computation:	Completed
Sparse Similarity Graph:	Completed
Recommendation Generation:	Pending
Evaluation Metrics:	Pending
Code Modularization:	Pending

3. Dataset Description and Preprocessing

3.1 Dataset

The system uses the MovieLens (latest-small) dataset, which is widely used for academic research in recommendation systems. It consists of user ratings for movies along with movie metadata.

Files Used:
- ratings.csv – user ratings
- movies.csv – movie titles and genres

Key Statistics:
- Approximately 100,000 ratings
- Approximately 600 users
- Approximately 9,000 movies

3.2 Preprocessing Steps

The following preprocessing steps were applied:
1. CSV files were loaded using Pandas.
2. Missing values and duplicate rows were removed.
3. The ratings dataset was randomly split into 80% training data and 20% testing data.

4. Data Structures Implemented

4.1 Hash Tables (Python Dictionaries)

Two hash tables were implemented. The first maps each user to the movies they have rated along with the rating values. The second maps each movie to the users who rated it and their corresponding ratings. These structures allow constant-time lookup on average and efficiently handle sparse data.

4.2 Graph (Adjacency List Representation)

Item–item similarity relationships are stored as a graph using an adjacency list representation. Each movie is treated as a node, and edges connect movies that are similar based on cosine similarity scores. This representation avoids storing a full M×M similarity matrix and is well suited for sparse graphs.

4.3 Priority Queue (Heap)

A priority queue is used to retain only the top-K most similar neighbors per movie. This ensures sparsity and improves lookup efficiency during later recommendation stages.

5. Algorithms Implemented

5.1 Item-Based Collaborative Filtering

The system follows the item-based collaborative filtering paradigm, where similarities are computed between items instead of users. This approach is computationally efficient and widely used in large-scale recommender systems.

5.2 Cosine Similarity

Cosine similarity is used to measure similarity between two movies. Each movie is represented as a vector of ratings from users who rated it. Only overlapping users are considered during similarity computation.

5.3 Sparsity Handling Strategies

To address sparsity and noise in the data, the following strategies were implemented:
- Minimum co-rating threshold
- Similarity thresholding (≥ 0.1)
- Retention of top-50 similar movies per item
- Sparse adjacency list representation

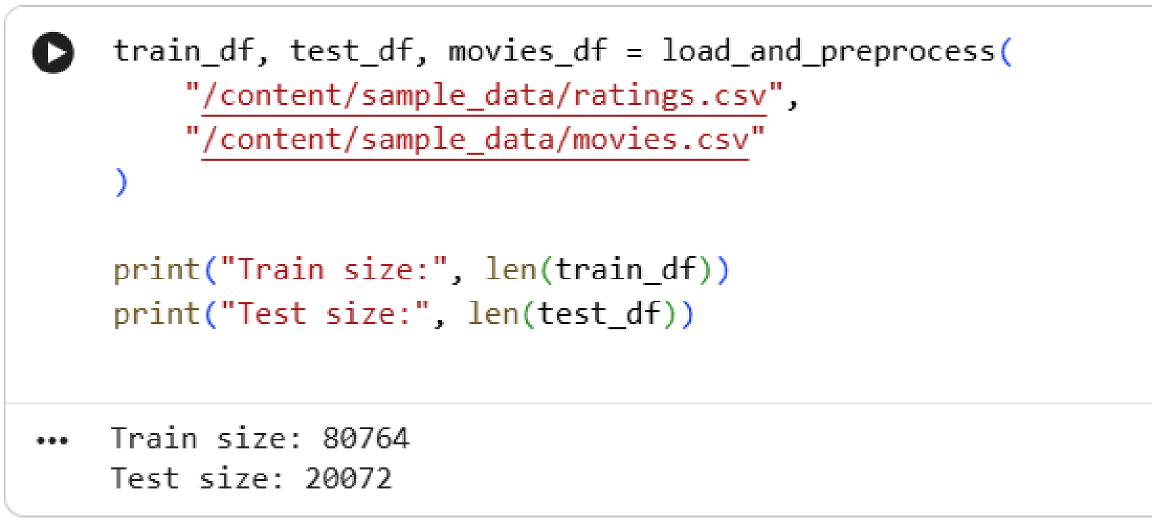
6. Time Complexity Analysis

Hash table construction has linear complexity with respect to the number of ratings. Item–item similarity computation is performed offline with complexity $O(M^2 \times U)$, where M is the number of movies and U is the average number of users per movie. Hash table lookups operate in $O(1)$ average time.

7. Sample Outputs and Verification

Sample outputs were generated to validate the implementation. These include dataset sizes after preprocessing, number of unique users and movies, sample user–movie mappings, sample movie–user mappings, example adjacency lists showing similar movies with similarity scores, and the time taken to build the similarity graph.

7.1 Dataset Loading and Train–Test Split:

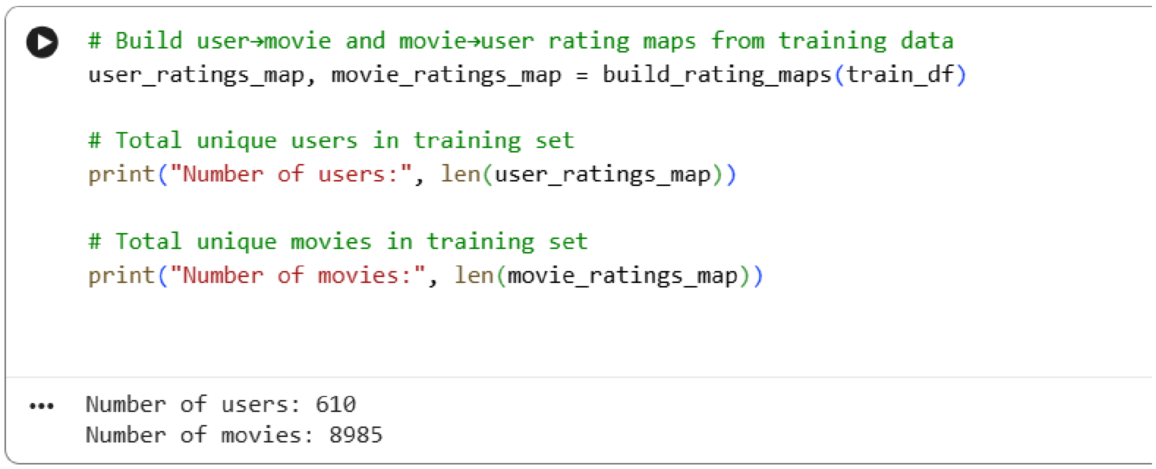


```
train_df, test_df, movies_df = load_and_preprocess(
    "/content/sample_data/ratings.csv",
    "/content/sample_data/movies.csv"
)

print("Train size:", len(train_df))
print("Test size:", len(test_df))
```

```
Train size: 80764
Test size: 20072
```

7.2 User–Item and Item–User Hash Table Sizes:



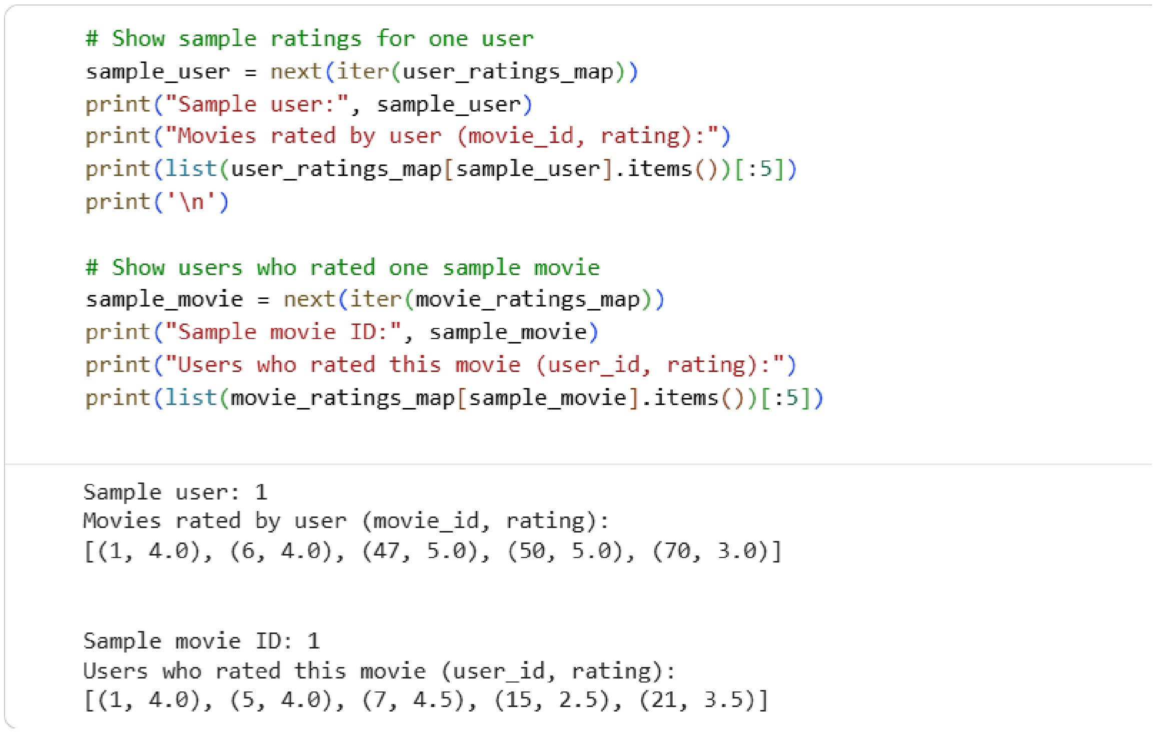
```
# Build user→movie and movie→user rating maps from training data
user_ratings_map, movie_ratings_map = build_rating_maps(train_df)

# Total unique users in training set
print("Number of users:", len(user_ratings_map))

# Total unique movies in training set
print("Number of movies:", len(movie_ratings_map))
```

```
Number of users: 610
Number of movies: 8985
```

7.3 Sample Item–Item Similarity Output



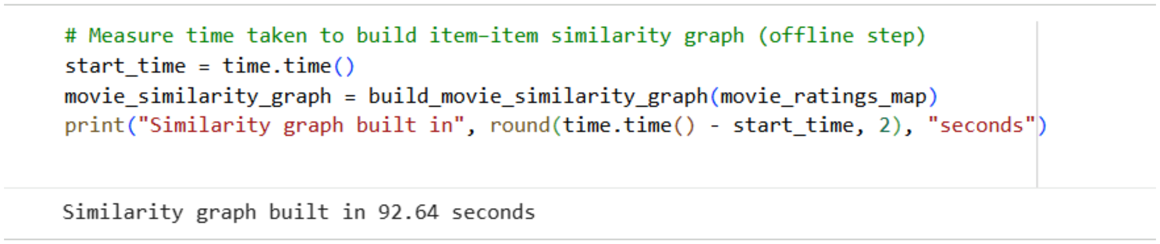
```python
# Show sample ratings for one user
sample_user = next(iter(user_ratings_map))
print("Sample user:", sample_user)
print("Movies rated by user (movie_id, rating):")
print(list(user_ratings_map[sample_user].items())[:5])
print('\n')

# Show users who rated one sample movie
sample_movie = next(iter(movie_ratings_map))
print("Sample movie ID:", sample_movie)
print("Users who rated this movie (user_id, rating):")
print(list(movie_ratings_map[sample_movie].items())[:5])
```

```
Sample user: 1
Movies rated by user (movie_id, rating):
[(1, 4.0), (6, 4.0), (47, 5.0), (50, 5.0), (70, 3.0)]


Sample movie ID: 1
Users who rated this movie (user_id, rating):
[(1, 4.0), (5, 4.0), (7, 4.5), (15, 2.5), (21, 3.5)]
```

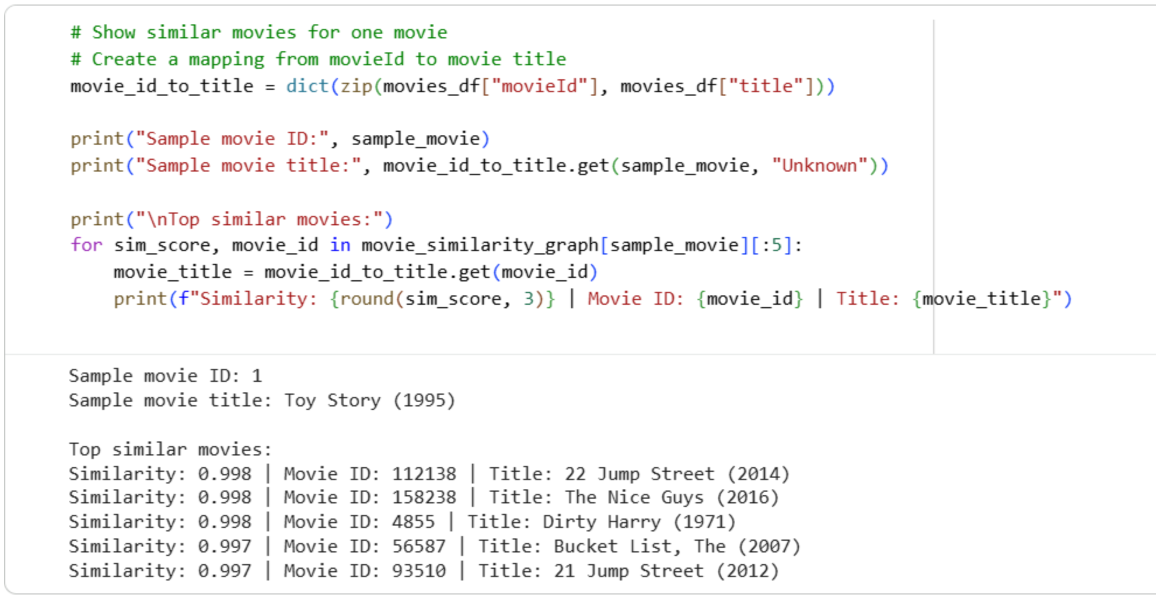
7.4 Similarity Graph Construction Time



```python
# Measure time taken to build item-item similarity graph (offline step)
start_time = time.time()
movie_similarity_graph = build_movie_similarity_graph(movie_ratings_map)
print("Similarity graph built in", round(time.time() - start_time, 2), "seconds")
```

```
Similarity graph built in 92.64 seconds
```

7.5 Sample Item–Item Similarity Output



```
# Show similar movies for one movie
# Create a mapping from movieId to movie title
movie_id_to_title = dict(zip(movies_df["movieId"], movies_df["title"]))

print("Sample movie ID:", sample_movie)
print("Sample movie title:", movie_id_to_title.get(sample_movie, "Unknown"))

print("\nTop similar movies:")
for sim_score, movie_id in movie_similarity_graph[sample_movie][:5]:
    movie_title = movie_id_to_title.get(movie_id)
    print(f"Similarity: {round(sim_score, 3)} | Movie ID: {movie_id} | Title: {movie_title}")
```

```
Sample movie ID: 1
Sample movie title: Toy Story (1995)

Top similar movies:
Similarity: 0.998 | Movie ID: 112138 | Title: 22 Jump Street (2014)
Similarity: 0.998 | Movie ID: 158238 | Title: The Nice Guys (2016)
Similarity: 0.998 | Movie ID: 4855 | Title: Dirty Harry (1971)
Similarity: 0.997 | Movie ID: 56587 | Title: Bucket List, The (2007)
Similarity: 0.997 | Movie ID: 93510 | Title: 21 Jump Street (2012)
```

8. Challenges and Solutions

The primary challenge encountered was handling sparsity in the user–item matrix. This was addressed by computing similarities only for overlapping users, filtering weak similarities, and storing only the top-K neighbors per movie. These strategies reduced noise and improved computational efficiency.

No major blockers exist at the current stage of the project.

9. Remaining Work and Updated Timeline

Remaining tasks include implementing rating prediction and top-N recommendation generation, handling cold-start users, evaluating the system using Precision@K, Recall@K, and NDCG@K, restructuring the code into a class-based modular design, and performing final benchmarking and documentation.

Based on the current progress and system stability, the team is confident of completing the remaining tasks before the final submission deadline.

10. Conclusion

This milestone demonstrates a solid foundation for the item-based collaborative filtering recommendation system. Core data structures and algorithms have been implemented successfully, and the system produces meaningful intermediate outputs. The project is on track for successful completion.