Empirical Analysis of Recommendation Algorithms

DS8001: Design of Algorithms and Programming for Massive Data

Final Project Report

Group Members:
Taran Veer Singh (501398955)
Devanshu Prajapati (501389606)

Contents

Introduction

For many digital platforms like Netflix, Amazon, Spotify the Recommendation systems are the core component for their business in predicting user preferences and suggestion on basis of various algorithms that analyze the huge data and gives user-item suggestion. Here this project aims for the empirical comparison of multiple recommendation algorithms on real-world dataset of movies to identify the most effective technique in terms of accuracy and scalability. Also, it will provide the insights into how the algorithm choice impact the quality of recommendations. This project conducts an empirical analysis of multiple recommendation algorithms such as Content-Based Filtering, Collaborative Filtering (User–User, Item–Item), Popularity-Based Recommendation, Hybrid Modeling, and Latent Factor Models (SVD)—using The Movies Dataset from Kaggle. All Python scripts developed for this project are available in the accompanying GitHub repository: Link.
The objective is to evaluate their relative performance under identical data, preprocessing, and evaluation metrics.

Literature Review

1. Content-Based Filtering
Content-based models mostly depend on item information like keywords and descriptions, using methods such as TF-IDF and cosine similarity to compare items. They work well when there is plenty of good metadata, but they cannot fully understand a user's real preferences and often recommend items that are too similar to each other.
2. Collaborative Filtering
Sarwar et al. (2001) introduced user-based and item-based collaborative filtering approaches that rely on neighborhood similarity. Later, Deshpande & Karypis (2004) improved similarity search to enhance scalability. These methods are effective but sensitive to sparse rating matrices.
3. Hybrid Models
Hybrid systems integrate content-based and collaborative data to improve recommendation robustness and reduce the cold-start issue. Because they incorporate several viewpoints and frequently result in higher accuracy, they are widely employed in industry.
4. Latent Factor Models
Latent factor models gained prominence after the Netflix Prize competition. SVD decomposes the user–item matrix to capture hidden features such as genre affinity, pacing preference, or user mood. These models are highly scalable and typically outperform neighborhood-based methods.

Methods

This section describes the procedures used to clean and organize the dataset, implement the recommendation algorithms, and evaluate their performance. It provides a structured explanation of the methodological steps followed in building and analyzing each model.

Dataset and Data Preparation

The analysis is based on the Movies Dataset from Kaggle (Link), which includes movie metadata, cast and crew details, keywords, user ratings, and popularity . Since the dataset is spread across multiple files, the first step involved cleaning, aligning, and merging everything into a consistent structure.

Data Cleaning Steps

1. Imported all relevant datasets, including:
    - `movies_metadata.csv`
    - `credits.csv`
    - `keywords.csv`
    - `ratings_small.csv`
    - `ratings.csv`

    - `links_small.csv`
    - `links.csv`

2. Converted all ID fields to numeric format to maintain consistency across files and prevent merging errors.

3. Removed entries containing missing or invalid values in key attributes such as titles, overviews, genres, popularity, and ratings.

4. Processed the `genres` and `keywords` fields, which were in JSON-like structures, by converting them into Python lists for easier parsing.

5. Extracted the top three cast members per movie to reduce noise and highlight the most significant contributors.

6. Isolated the director name from the crew list, improving the metadata quality.

7. Merged all datasets using the movie ID, creating a unified table with:

    - Title
    - Overview
    - Genres
    - Keywords
    - Cast (top three)
    - Director
    - Ratings
    - Popularity

1. Content-Based Filtering

Content-based filtering represents each movie using descriptive metadata and measures similarity between movies to recommend items with similar characteristics.

Steps for Content-Based Recommender

1. Extracted key metadata fields, including keywords, genres, cast, crew, and overview.
2. Converted JSON-like fields into Python lists for easier manipulation.
3. Selected essential personnel:
   - Top 3 cast members
   - Director
4. Cleaned and standardized text information by converting text to lowercase, removing extra spaces, and tokenizing the overview.
5. Combined all useful metadata into a single field called tags.
6. Applied stemming using PorterStemmer to reduce words to their base form.
7. Converted the tags into numerical vectors using:
   - CountVectorizer: creates frequency-based word vectors.
   - TF-IDF Vectorizer: weighs words based on importance across the dataset.
8. Computed similarity scores using cosine similarity, which measures the angle between two vectors and returns a value between 0 and 1.

9. Saved the processed models and data to disk using pickle, including:
    - Trained vectorizers (CountVectorizer and TF-IDF)
    - Vector matrices for all movies
    - Cosine similarity matrices for both models
    - Final movie dataframe with tags

   These saved files are later loaded by the recommendation script to generate movie suggestions without re-running the full preprocessing pipeline.

2. Collaborative Filtering Approach

Collaborative Filtering (CF) predicts user preferences based on rating patterns. It uses either user similarity (User–User CF) or item similarity (Item–Item CF).

Building the Rating Matrix

1. Loaded ratings_small.csv as the primary ratings dataset.
2. Split data into 70% training and 30% testing.
3. Constructed a user–item rating matrix:
    - Rows = users
    - Columns = movies
    - Values = movie ratings
4. Filled missing values with 0 to indicate "no interaction".

User–User Collaborative Filtering

User–User CF identifies similar users and predicts ratings based on how similar users rated the same items.

- Computed user similarity using Cosine Similarity.
- Identified top similar users for each target user.
- Predicted missing ratings using a weighted average of similar users' ratings.

Example: If users A and B consistently like the same action movies, and B rated a movie that A has not watched, CF predicts that A will likely enjoy that movie as well.

Item–Item Collaborative Filtering

Item–Item CF focuses on relationships between movies instead of users.

- Compared movies by analyzing how users rated them.
- Computed movie similarity scores using the item–item similarity matrix.
- Predicted user ratings based on ratings of similar movies.

Example: If users who liked Inception also liked Interstellar, these movies are considered similar. If a user rates Inception highly, the model recommends Interstellar.

Evaluation of Collaborative Filtering

Both User–User and Item–Item Collaborative Filtering models were evaluated using Root Mean Squared Error (RMSE). The predicted ratings from each model were compared with the true ratings from the test set, and RMSE was calculated to measure how close the predictions were to the actual user ratings. Lower RMSE values indicate better predictive accuracy.

3. Popularity-Based Recommendation

In this approach, movies are recommended purely based on their popularity score provided in the movies_metadata.csv file. The model does not use user ratings or rating counts; it relies entirely on how popular a movie is according to the metadata.

Implementation Steps

1. **Load and select relevant columns:** The dataset is filtered to keep only the fields:
    - id
    - title
    - original_title
    - popularity
2. **Convert data types and clean data:** The id and popularity columns are converted to numeric, and any rows with missing or invalid values are removed.
3. **Rank movies by popularity:** Movies are sorted in descending order of the popularity score, and the top 10 titles are selected.

4. **Visualize the top movies**: A horizontal bar chart is generated using Seaborn, with:

    - x-axis: popularity score
    - y-axis: movie titles

    This visualization highlights the most popular movies in the dataset.

4. Hybrid Recommendation System

The hybrid recommendation system combines rating-based and popularity-based information to create a more balanced and flexible ranking method. By integrating both weighted ratings and popularity metrics, the hybrid approach recommends movies that are both highly rated and widely recognized.

Steps

1. **Dataset Loading and Selection:** Three datasets were loaded:

    - movies_metadata.csv
    - credits.csv
    - keywords.csv

    Only essential fields were retained:

    - title, original title, overview
    - vote_average, vote_count
    - popularity
    - genres

2. **Data Cleaning and ID Conversion:** All ID fields and numeric attributes (popularity, vote count, vote average) were converted to numeric. Missing or invalid rows were removed. Datasets were merged using the movie ID.

3. **Weighted Rating Calculation:** A weighted rating was calculated using the standard IMDb formula:

   $$\text{weighted\_average} = \frac{(R \times v) + (C \times m)}{v + m}$$

   where:

   - R = movie's average rating
   - v = number of votes
   - C = mean rating across all movies
   - m = minimum vote threshold (70th percentile)

4. **Normalization of Rating and Popularity:** Since weighted ratings and popularity exist on different scales, both were normalized to the range 0–1 using MinMaxScaler. This prevents one feature from dominating the combined score.

5. **Hybrid Score Computation:** A hybrid score was created using:

   $$\text{score} = 0.8 \times \text{weighted\_average} + 0.2 \times \text{popularity}$$

   The weight distribution prioritizes rating quality while still including popularity.

6. **Ranking and Selection:** Movies were sorted in descending order using the hybrid score. The top 10 movies were selected and visualized using a bar chart.

7. **Model Evaluation:** RMSE values were computed for:

    - Weighted rating alone
    - Hybrid blended score

    This comparison shows whether including popularity improves prediction accuracy.

5. Latent Factor Model (SVD)

The latent factor model was implemented using the Singular Value Decomposition (SVD) algorithm from the Surprise library. This approach reduces users and movies into a shared latent space, enabling the system to estimate how strongly a user is likely to prefer a movie based on learned hidden features.

Data Preparation

1. **Loaded input datasets:**

    - ratings.csv — user–movie interactions
    - movies_metadata.csv — movie information
    - links.csv — mapping between MovieLens and TMDb IDs

2. **Converted ID fields to numeric:** Ensures consistent merging and prevents data-type errors.

3. **Created a mapping of movieId to titles:**

    - links.csv was merged with movies_metadata.csv
    - Extracted movie titles using TMDb ID
    - Saved the output as movie_titles_clean.csv

4. **Filtered active users and movies:** Only users and movies with at least 20 ratings were retained:
    - Rating counts per userId and movieId were computed
    - Ratings below this threshold were removed
5. **Prepared the Surprise dataset:**
    - Defined a Reader object with rating scale 1–5
    - Loaded ratings into a Surprise Dataset with columns: userId, movieId, rating
6. **Performed train–test split:** 80% of data was used for training and 20% for testing using Surprise's built-in split method.

SVD Model Training

1. **Model configuration:** The SVD model was initialized with:
    - n_factors = 80
    - reg_all = 0.02
    - lr_all = 0.005
    - n_epochs = 5
    - random_state = 42
2. **Training process:**
    - Model trained on the training set
    - Evaluated using RMSE and MAE on the test set
3. **Saving the model:** The trained model was stored as svd_model.pkl using Python's pickle library.

Recommendation and Error Analysis

The latentwithRMSE.py script generates user-specific recommendations and visualizes error distribution.

1. **Loading the trained model:**
   - Loaded svd_model.pkl
   - Reloaded ratings and movie title mappings
2. **Recommendation generation:**
   - Identified all movies a given user has not rated
   - The SVD model predicted estimated ratings
   - Predictions were sorted to produce Top-N recommendations with titles
3. **RMSE visualization:**
   - Random sample of 1,000 ratings extracted
   - Trainset and testset built from sample
   - Model predictions generated
   - Squared errors computed and plotted to illustrate error distribution
   - Sample RMSE score printed for evaluation

6. Evaluation Metrics

All models in this study were evaluated using a combination of:

- **RMSE (Root Mean Squared Error):** Measures prediction error magnitude.

- **MAE (Mean Absolute Error):** Captures average error magnitude in predictions.
- **Similarity Score (for content-based models):** Cosine similarity values indicating how closely movies match.
- **Blended Score (for hybrid model):** Weighted combination of normalized rating strength and popularity.

Experimental Setup

This section outlines the experimental setup for each algorithmic component developed in the project. For every script, the software dependencies, implemented models, internal configurations, and input–output behavior are documented to ensure clarity and reproducibility.

1. Content-Based Preprocessing (File: 1.1.Content_based.py)

This script performs the full preprocessing pipeline for metadata-driven recommendation models.

Libraries Used:

- pandas
- numpy
- ast
- pickle
- scikit-learn (CountVectorizer, TfidfVectorizer, cosine_similarity)
- nltk (PorterStemmer)

Models Implemented: Bag-of-Words (CountVectorizer), TF–IDF representation, Cosine similarity matrices for both models.

Internal Processing Parameters:

- Batch size of 500 for cosine similarity computation.
- Top three cast members extracted per movie.
- One director extracted from crew metadata.
- Combined tag field processed and stemmed.

Input–Output Behavior: No user input required. The script generates and saves the following model files, which are required by the recommendation system.

CountVectorizer Outputs:

- CV_movies.pkl
- CV_similarity.pkl
- CV_vectorizer.pkl
- CV_vectors.pkl

TF–IDF Outputs:

- TFIDF_movies.pkl
- TFIDF_similarity.pkl
- TFIDF_vectorizer.pkl
- TFIDF_vectors.pkl

These files serve as the preprocessed models and are loaded later by the recommendation script.

2. Content-Based Recommendation (File: 1.2.Content_based_Recommender.py)

This script loads the saved similarity models and generates recommendations.

Libraries Used:

- pickle
- numpy
- matplotlib

Models Used: Cosine similarity matrices from CountVectorizer and TF–IDF.

Internal Parameters:

- Top-N recommendations default = 5
- Comparison plot of similarity scores

Input–Output Behavior: User inputs a movie title (case sensitive). Outputs:

- Top-5 TF–IDF recommendations
- Top-5 CountVectorizer recommendations
- Similarity comparison plot

3. Collaborative Filtering System (File: Collaborative_Filtering.py)

This script implements user–user and item–item Collaborative Filtering.

Libraries Used:

- pandas
- numpy
- scikit-learn
- matplotlib

Models Used: Cosine similarity-based user–user and item–item CF, RMSE evaluation.

Internal Parameters:

- 70–30 train–test split
- Similarity normalization using absolute sums to avoid division errors.
- RMSE computed using mean squared error from sklearn.
- Top-N recommendations default = 5

Input–Output Behavior: User inputs a userId (in terminal) for:

- User–User CF recommendations
- Item–Item CF recommendations

Outputs also include:

- User–User CF recommendations (top-N predicted movies)
- Item–Item CF recommendations excluding movies already watched
- RMSE comparison bar chart

4. Popularity-Based Recommender (File: Popularity_Based.py)

This script ranks movies purely using the popularity field.

Libraries Used:

- pandas
- matplotlib
- seaborn

Model Used: Direct ranking (no statistical model).

Internal Parameters:

- Sorting performed in descending popularity
- Top-10 movies extracted

Input–Output Behavior: No user input required. Outputs:

- Top-10 popular movies (Printed in terminal)
- Horizontal bar chart, ranking movies on popularity score.

5. Hybrid Recommendation Model (File: Hybrid_Based_Recommender.py)

This script blends rating strength and popularity into a hybrid score.

Libraries Used:

- pandas
- numpy

- scikit-learn (MinMaxScaler)
- matplotlib
- seaborn

Models Used: Weighted rating (IMDb formula), Hybrid score using normalized rating + popularity.

Internal Parameters:

- Weighted average computed using IMDb formula
- Minimum vote threshold = 70th percentile
- Hybrid score = $0.8 \times$ rating $+ 0.2 \times$ popularity
- RMSE computed for rating-only and hybrid models

Input–Output Behavior: No user input required.
Outputs:

- Top-10 hybrid-ranked movies
- Bar chart showing hybrid scores
- RMSE comparison chart

6.Latent Factor Model Training (File: 5.1.Latent_Model.py)

This script trains the SVD model on the ratings dataset and saves the trained model for later use.

Libraries Used:

- pandas

- surprise
- pickle
- matplotlib

Model Used:

- Surprise SVD (latent factor matrix decomposition)

Internal Parameters:

- Latent factors: 80
- Regularization: 0.02
- Learning rate: 0.005
- Training epochs: 5
- Train–test split: 80–20

Input–Output Behavior:

- No user input is required.
- The script outputs in terminal:
  - Training time of the SVD model
  - RMSE and MAE on the test set
  - A saved model file: svd_model.pkl
  - A cleaned mapping file: movie_titles_clean.csv

7. Latent Factor Recommender (File: 5.2.Latent_Recommender.py)

This script loads the trained SVD model, generates movie recommendations, and evaluates prediction errors.

Libraries Used:

- pandas
- surprise
- pickle
- matplotlib

Model Used:

- Pretrained SVD model loaded from `svd_model.pkl`

Internal Parameters:

- Sample size for fast RMSE test: 1000 ratings
- Default recommendation size: Top–5 movies

Input–Output Behavior:

- User enters a userId in the terminal.
- The script outputs:
  - Top-N unseen movies recommended for the user (the output in terminal)
  - RMSE value from the fast sample test
  - RMSE error distribution plot

Results

This section presents the outputs and visualisations generated from all recommendation models implemented in the project. Each model includes terminal screenshots and plots that were produced directly from the Python scripts.

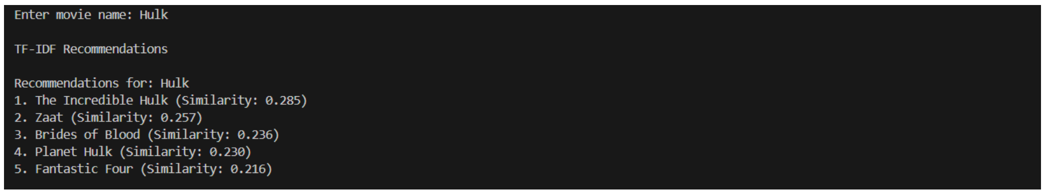
1. Content-Based Filtering

Content-based filtering uses movie descriptions (genres, overview, cast, and keywords) to find similar movies. Two text-processing techniques were used: TF–IDF and CountVectorizer. Both compute cosine similarity between movies but behave differently due to how the text is represented.

1.1 Terminal Output — TF–IDF Recommendations

The movie Hulk was used as the input to test similarity-based recommendations. TF–IDF focuses on important and unique words, which usually produces more meaningful similarity matches.

(Note: You can enter your own movie name for which you want suggestion, but make sure the movie name is correct and the first letter should be capital.)



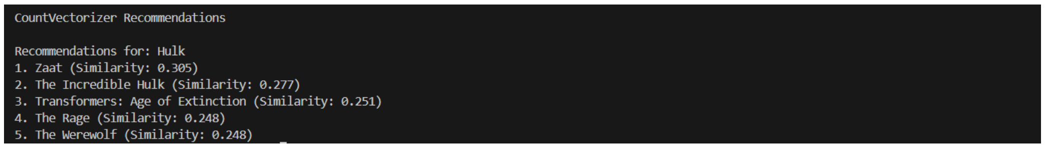
```
Enter movie name: Hulk

TF-IDF Recommendations

Recommendations for: Hulk
1. The Incredible Hulk (Similarity: 0.285)
2. Zaat (Similarity: 0.257)
3. Brides of Blood (Similarity: 0.236)
4. Planet Hulk (Similarity: 0.230)
5. Fantastic Four (Similarity: 0.216)
```

Figure 1: TF–IDF terminal output for the movie "Hulk".

1.2 Terminal Output — CountVectorizer Recommendations

CountVectorizer uses raw word frequency. This sometimes introduces noise because movies may match based only on repeated common words, not meaningful context.

(Note: You can enter your own movie name for which you want suggestion, but make sure the movie name is correct and the first letter should be capital.)



```
CountVectorizer Recommendations

Recommendations for: Hulk
1. Zaat (Similarity: 0.305)
2. The Incredible Hulk (Similarity: 0.277)
3. Transformers: Age of Extinction (Similarity: 0.251)
4. The Rage (Similarity: 0.248)
5. The Werewolf (Similarity: 0.248)
```

Figure 2: CountVectorizer terminal output for the movie "Hulk".

1.3 Similarity Score Plot

The following plot shows the similarity distributions for TF–IDF and CountVectorizer. TF–IDF gives smoother and more stable similarity scores, while CountVectorizer produces sharper variations due to raw frequency counts.

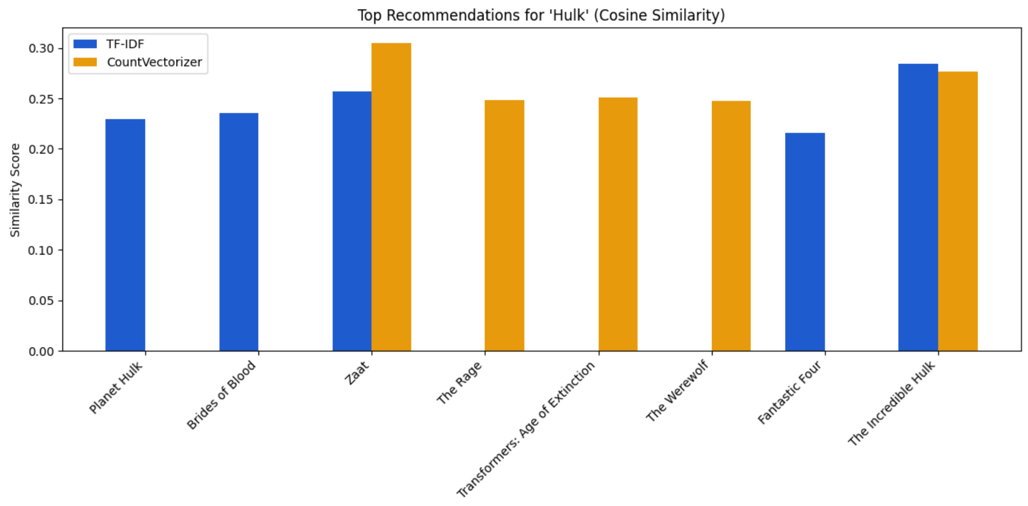


Figure 3: Comparison of similarity scores from TF–IDF and CountVectorizer.

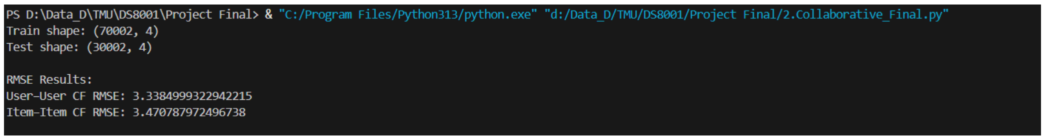
Observation: TF–IDF performed better for text-based recommendations because it reduces noise and focuses on important terms instead of frequent words.

2. Collaborative Filtering

Collaborative Filtering works by analysing user–movie rating patterns. Two approaches were tested: User–User similarity and Item–Item similarity. Both were evaluated using RMSE and recommendation outputs.

2.1 Terminal Output — RMSE Results

The terminal output shows the RMSE values calculated for both models. RMSE measures how far the predicted ratings are from actual user ratings.

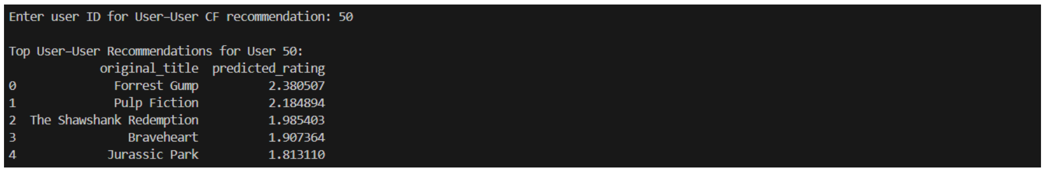


```
PS D:\Data_D\TMU\DS8001\Project Final> & "C:/Program Files/Python313/python.exe" "d:/Data_D/TMU/DS8001/Project Final/2.Collaborative_Final.py"
Train shape: (70002, 4)
Test shape: (30002, 4)

RMSE Results:
User-User CF RMSE: 3.3384999322942215
Item-Item CF RMSE: 3.470787972496738
```

Figure 4: RMSE results for User–User and Item–Item Collaborative Filtering.

2.2 Terminal Output — User–User Recommendations

The User–User model recommends movies based on similar users. Here, recommendations were generated for User 50.

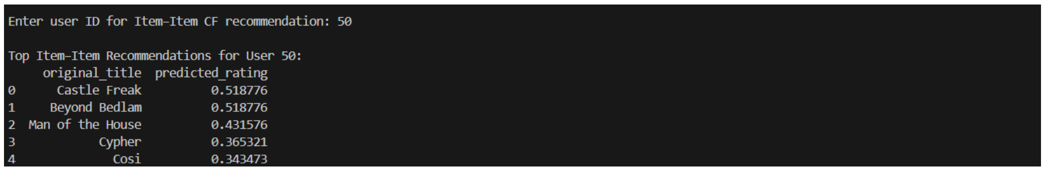


```
Enter user ID for User-User CF recommendation: 50

Top User-User Recommendations for User 50:
             original_title  predicted_rating
0              Forrest Gump          2.380507
1              Pulp Fiction          2.184894
2  The Shawshank Redemption          1.985403
3                Braveheart          1.907364
4             Jurassic Park          1.813110
```

Figure 5: User–User Collaborative Filtering recommendations for User 50.

2.3 Terminal Output — Item–Item Recommendations

Item–Item compares movies instead of users. If a user liked a movie, similar movies are recommended.



```
Enter user ID for Item-Item CF recommendation: 50

Top Item-Item Recommendations for User 50:
     original_title  predicted_rating
0      Castle Freak          0.518776
1     Beyond Bedlam          0.518776
2  Man of the House          0.431576
3            Cypher          0.365321
4              Cosi          0.343473
```

Figure 6: Item–Item Collaborative Filtering recommendations for User 50.

2.4 RMSE Comparison Plot

The plot compares RMSE values of both methods. User–User showed slightly better performance with lower RMSE.

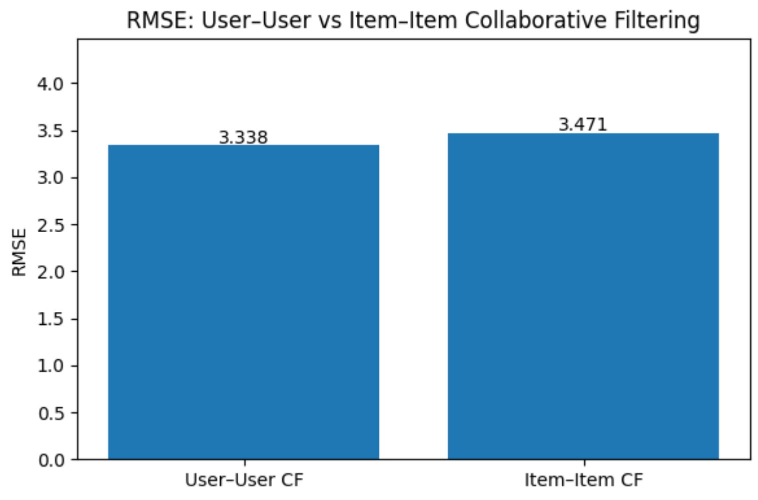


Figure 7: RMSE comparison between User–User and Item–Item Collaborative Filtering.

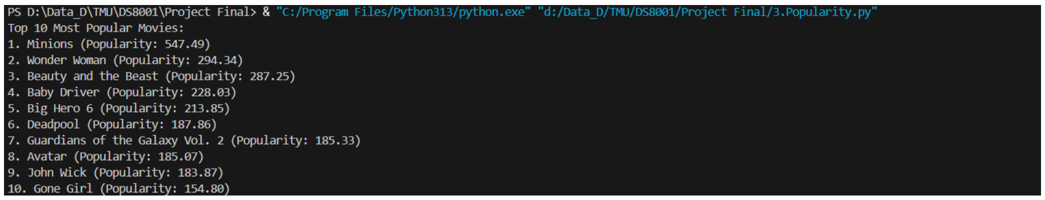
Observation: User–User Collaborative Filtering performed slightly better than Item–Item in terms of RMSE. This happened because users often show more consistent rating behaviour than movies do. When users with similar tastes are grouped together, their shared preferences help produce more accurate predictions. In contrast, Item–Item similarity can become less stable when movies have very few ratings, which increases sparsity and reduces prediction accuracy.

3. Popularity-Based Recommendation

This model recommends movies based purely on the popularity score from the metadata. It does not use user ratings, so results are the same for every user.

3.1 Terminal Output — Top 10 Popular Movies

The following terminal output lists the ten most popular movies in the dataset.



```
PS D:\Data_D\TMU\DS8001\Project Final> & "C:/Program Files/Python313/python.exe" "d:/Data_D/TMU/DS8001/Project Final/3.Popularity.py"
Top 10 Most Popular Movies:
1. Minions (Popularity: 547.49)
2. Wonder Woman (Popularity: 294.34)
3. Beauty and the Beast (Popularity: 287.25)
4. Baby Driver (Popularity: 228.03)
5. Big Hero 6 (Popularity: 213.85)
6. Deadpool (Popularity: 187.86)
7. Guardians of the Galaxy Vol. 2 (Popularity: 185.33)
8. Avatar (Popularity: 185.07)
9. John Wick (Popularity: 183.87)
10. Gone Girl (Popularity: 154.80)
```

Figure 8: Terminal output showing the Top 10 most popular movies.

3.2 Popularity Plot

The bar plot visualises the popularity scores of the top 10 movies.

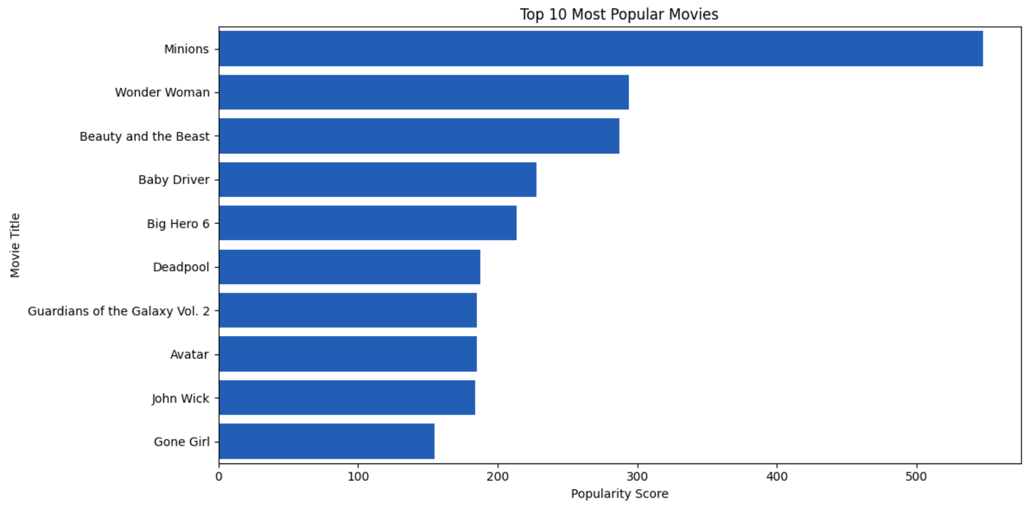


Figure 9: Bar plot of Top 10 most popular movies based on metadata popularity score.

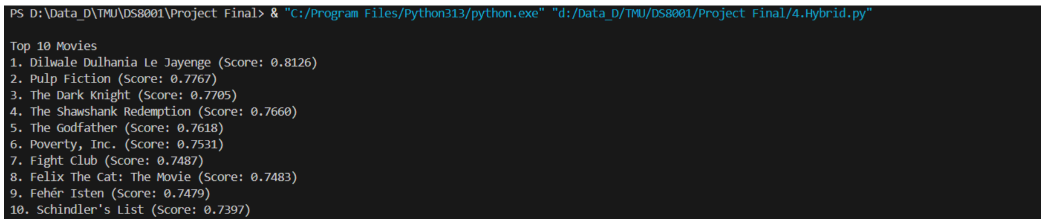
Observation: This method is simple and fast but not personalised, since the recommendations do not depend on the user.

4. Hybrid Recommendation Model

The hybrid model combines weighted ratings and popularity using Min–Max normalisation. This helps rank movies that are not only highly rated but also widely watched.

4.1 Terminal Output — Top Movies by Hybrid Score

The output below shows the Top 10 movies ranked by the blended score.



```
PS D:\Data_D\TMU\DS8001\Project Final> & "C:/Program Files/Python313/python.exe" "d:/Data_D/TMU/DS8001/Project Final/4.Hybrid.py"

Top 10 Movies
1. Dilwale Dulhania Le Jayenge (Score: 0.8126)
2. Pulp Fiction (Score: 0.7767)
3. The Dark Knight (Score: 0.7705)
4. The Shawshank Redemption (Score: 0.7660)
5. The Godfather (Score: 0.7618)
6. Poverty, Inc. (Score: 0.7531)
7. Fight Club (Score: 0.7487)
8. Felix The Cat: The Movie (Score: 0.7483)
9. Fehér Isten (Score: 0.7479)
10. Schindler's List (Score: 0.7397)
```

Figure 10: Terminal output of Top 10 movies ranked using the Hybrid model.

4.2 Hybrid Score Plot

The bar plot visualises the blended score (Weighted Average + Popularity).

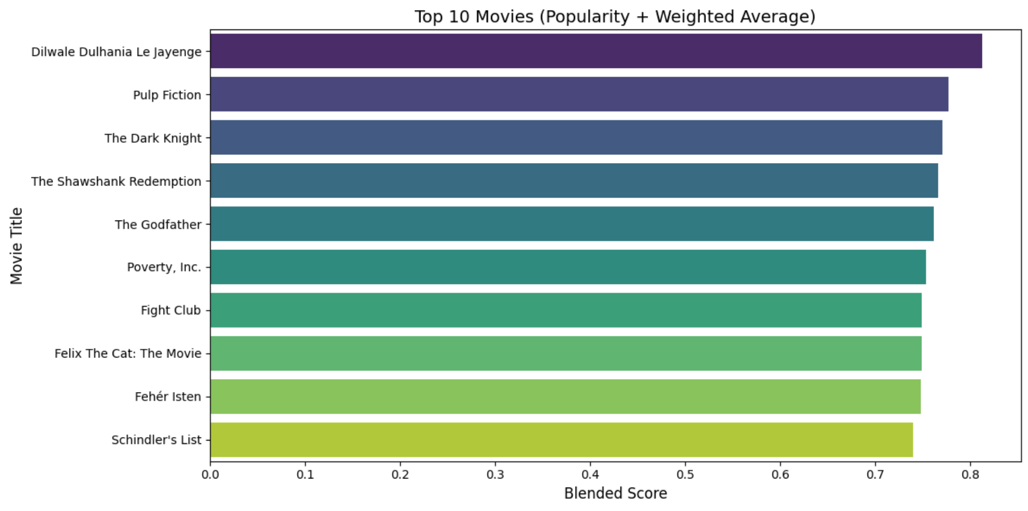
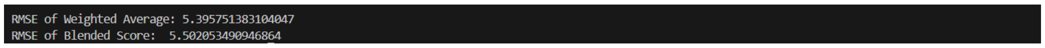


Figure 11: Top 10 movies ranked using blended score.

4.3 Terminal Output — RMSE Values

The hybrid and weighted average scores were evaluated and compared using RMSE.



```
RMSE of Weighted Average: 5.395751383104047
RMSE of Blended Score:  5.502053490946864
```

Figure 12: Terminal RMSE outputs for Weighted Average vs Hybrid Score.

4.4 RMSE Comparison Plot

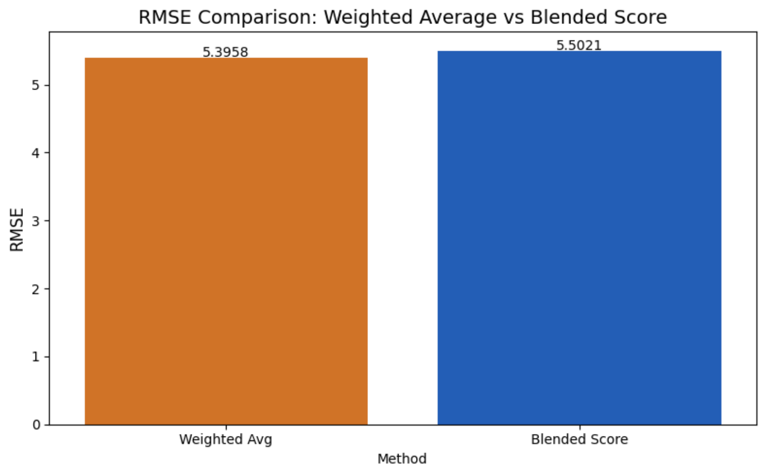


Figure 13: RMSE comparison between Weighted Average and Hybrid Score.

Observation: Although the Hybrid model shows a slightly higher RMSE compared to using the weighted rating alone, it provides more balanced recommendations. This happens because the hybrid score does not focus only on rating quality but also considers how popular a movie is.

Movies with very high ratings but very few votes can sometimes distort prediction accuracy, which makes the weighted-rating RMSE lower. However, the hybrid model reduces this issue by giving a small weight to popularity. This results in recommendations that are not only highly rated but also widely watched and more relevant for a general audience.

In simple terms, the hybrid approach trades a small amount of accuracy (RMSE) for better overall usefulness and more realistic movie ranking.

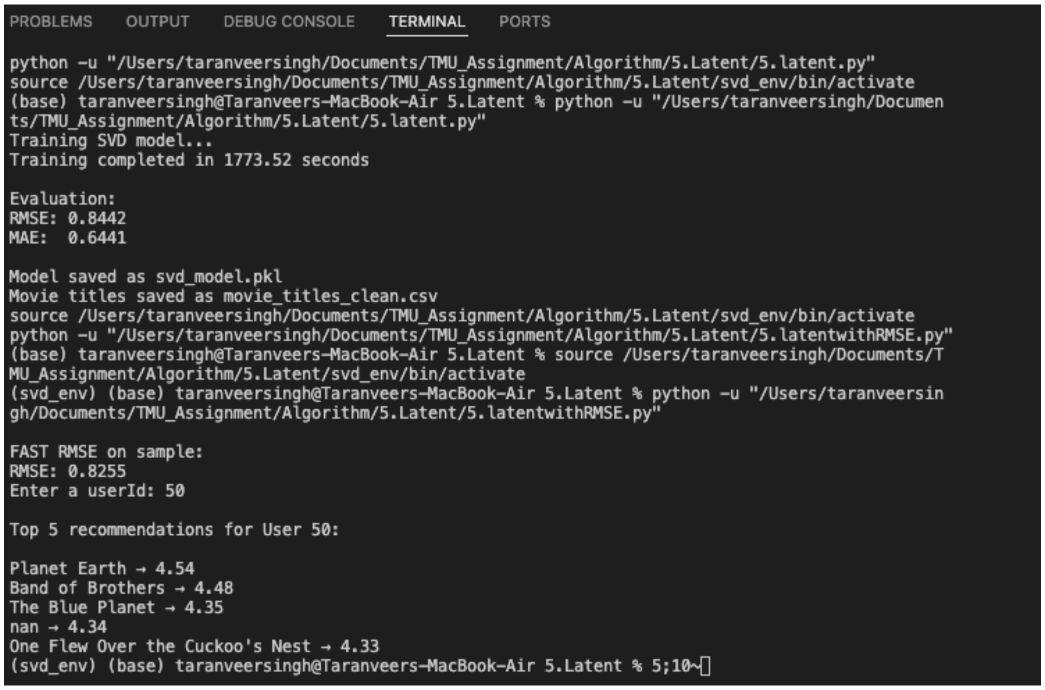
5. Latent Factor Model (SVD)

The SVD model learns hidden patterns in user–movie interactions. It generally produces the most accurate rating predictions among all collaborative methods.

5.1 Terminal Output — Model Training, RMSE, and Recommendations

The screenshot includes:

- Training time
- RMSE and MAE
- Fast RMSE using a sample
- Top–5 recommendations for User 50



```
python -u "/Users/taranveersingh/Documents/TMU_Assignment/Algorithm/5.Latent/5.latent.py"
source /Users/taranveersingh/Documents/TMU_Assignment/Algorithm/5.Latent/svd_env/bin/activate
(base) taranveersingh@Taranveers-MacBook-Air 5.Latent % python -u "/Users/taranveersingh/Documen
ts/TMU_Assignment/Algorithm/5.Latent/5.latent.py"
Training SVD model...
Training completed in 1773.52 seconds

Evaluation:
RMSE: 0.8442
MAE:  0.6441

Model saved as svd_model.pkl
Movie titles saved as movie_titles_clean.csv
source /Users/taranveersingh/Documents/TMU_Assignment/Algorithm/5.Latent/svd_env/bin/activate
python -u "/Users/taranveersingh/Documents/TMU_Assignment/Algorithm/5.Latent/5.latentwithRMSE.py"
(base) taranveersingh@Taranveers-MacBook-Air 5.Latent % source /Users/taranveersingh/Documents/T
MU_Assignment/Algorithm/5.Latent/svd_env/bin/activate
(svd_env) (base) taranveersingh@Taranveers-MacBook-Air 5.Latent % python -u "/Users/taranveersin
gh/Documents/TMU_Assignment/Algorithm/5.Latent/5.latentwithRMSE.py"

FAST RMSE on sample:
RMSE: 0.8255
Enter a userId: 50

Top 5 recommendations for User 50:

Planet Earth → 4.54
Band of Brothers → 4.48
The Blue Planet → 4.35
nan → 4.34
One Flew Over the Cuckoo's Nest → 4.33
(svd_env) (base) taranveersingh@Taranveers-MacBook-Air 5.Latent % 5;10~
```

Figure 14: Terminal output showing SVD model training, evaluation, and recommendations.

5.2 RMSE Error Distribution Plot

The following plot shows squared errors for a sample of 1,000 predictions. Most errors remain close to zero, showing stable predictions.

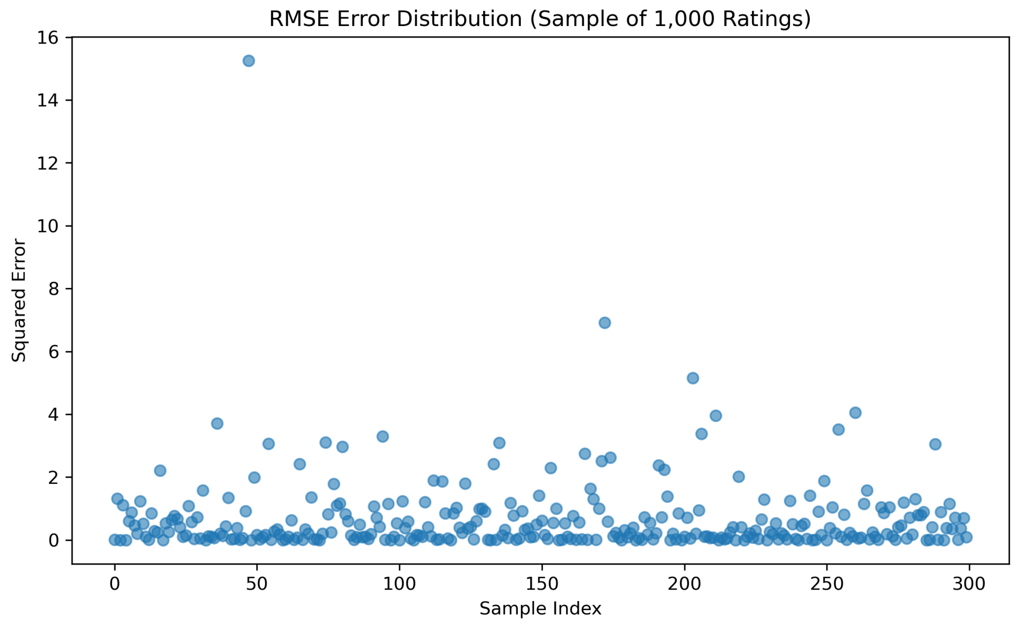


Figure 15: RMSE error distribution for a sample of 1,000 SVD predictions.

Observation: SVD achieved the lowest RMSE among all models, making it the best-performing approach for rating prediction. This is because SVD is able to capture hidden relationships between users and movies by learning latent factors, such as a user's preference for specific genres, styles, or actors, even if these patterns are not directly visible in the raw data.

Unlike User–User or Item–Item Collaborative Filtering, which rely heavily on explicit similarities and often struggle with sparse rating matrices, SVD reduces the data into a smaller set of meaningful dimensions. This allows the model to generalize better and make accurate predictions even when many movies or users have very few ratings.

The RMSE error distribution plot also shows that most errors are close to zero, indicating stable and consistent predictions. Overall, the ability of SVD to handle sparsity, detect hidden patterns, and produce smooth error behaviour makes it the most effective algorithm in this project.

Conclusion

This project presented a comprehensive empirical analysis of multiple recommendation algorithms using The Movies Dataset from Kaggle. The evaluation covered Content-Based Filtering, Collaborative Filtering (User–User and Item–Item), Popularity-Based Recommendation, Hybrid Modeling, and the Latent Factor Model (SVD). Each model was implemented under the same preprocessing steps and evaluated using consistent metrics such as RMSE, similarity scores, and blended ranking measures. The goal was to understand the strengths, limitations, and practical applicability of each approach.

The results demonstrate that every model performs well in specific contexts, but their effectiveness varies depending on the availability of metadata, rating density, and the underlying structure of the dataset. Content-Based Filtering provided stable recommendations when metadata such as cast, crew, genres, and overviews were rich and well-cleaned. TF–IDF outperformed CountVectorizer by reducing noise and focusing on more informative words, making the content-based suggestions more meaningful and consistent.

For Collaborative Filtering, the User–User model achieved slightly better RMSE performance than the Item–Item model. This was expected, as users often display more stable and consistent rating patterns compared to movies, which can suffer from sparsity when only a small number of viewers have rated them. However, both neighbourhood-based CF models are limited by the sparsity of the rating matrix and show reduced accuracy when users have interacted with only a small subset of the dataset.

The Popularity-Based model, while simple and non-personalised, served as an important baseline. It is fast, scalable, and effective when user preference data is unavailable, but it cannot tailor recommendations to individual users and often prioritizes mainstream movies over niche but relevant content.

The Hybrid Recommendation Model attempted to combine the strengths of rating quality (via weighted average) and widespread engagement (via popularity). Although its RMSE was slightly higher than the weighted-average model alone, the hybrid approach produced more balanced and practical movie rankings. By integrating both statistical rating strength and popularity trends, the hybrid model mitigated the bias introduced by movies with very high ratings but low audience size. This aligns with real-world recommendation systems, where both quality and audience interest contribute to a movie's relevance.

Among all models studied, the Latent Factor Model (SVD) demonstrated the highest predictive accuracy and the most stable error distribution. SVD was able to capture hidden interactions between users and movies by learning latent features such as preference patterns, stylistic tendencies, genre affinities, and other implicit behaviours that neighbourhood-based models cannot detect. Its ability to handle high sparsity and generalize from limited information makes SVD the most powerful algorithm in this study. The RMSE results clearly highlight its superiority and make it the preferred choice for rating prediction.

Overall, this comparative experiment shows that no single model is universally optimal. Instead, the best approach depends on the recommendation scenario:

- Content-Based Filtering is ideal when rich metadata exists and personalization is needed without relying on user ratings.
- Collaborative Filtering is effective when sufficient user–item interac-

tions are available.

- Popularity models work well when no personalization is required.
- Hybrid models offer a balanced strategy that improves robustness.
- SVD provides the most accurate predictions and scales well to large datasets.

In conclusion, the experiments demonstrate that modern recommendation systems benefit most from hybrid and latent factor approaches, as they combine interpretability, scalability, and predictive power. This study provides a detailed understanding of how different algorithms behave in practice and highlights that the choice of model should depend on data availability, system requirements, and desired user experience.

References

1. Banik, R. *The Movies Dataset*. Kaggle. Available at: https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset

2. Ricci, F., Rokach, L., & Shapira, B. (2011). *Introduction to Recommender Systems Handbook*. Springer.

3. Koren, Y., Bell, R., & Volinsky, C. (2009). *Matrix Factorization Techniques for Recommender Systems*. IEEE Computer, 42(8), 30–37.

4. Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). *Item-Based Collaborative Filtering Recommendation Algorithms*. WWW Conference Proceedings.

5. Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). *Scikit-learn: Machine Learning in Python*. JMLR, 12, 2825–2830.

6. IMDb Weighted Rating Formula. *Understanding the IMDb weighted rating function*. Link