Assignment 2

ITI5212
Data Analysis for Semi Structured Data

MONASH University

Student Name: Robiatul Adawiyah Al-Qosh
Student ID: 34269193

Table of Contents

Introduction

Recommender systems play a crucial role in enhancing user experience by providing personalized suggestions based on user preferences and historical interactions (Raza et al., 2024). These systems are widely used in various domains such as e-commerce, streaming services, and social media platforms (Hasan et al., 2024). The primary objective of this project is to develop a recommender system capable of predicting user ratings for products in the Amazon product dataset. This task is part of the FIT5212 Data Analysis for Semi-Structured Data course, which involves the application of advanced machine learning techniques on semi-structured data.

The dataset used in this project consists of user-item interactions where users have explicitly rated products on a scale from 1 to 5. Additionally, the dataset includes product metadata such as user_id and product_id . The goal is to leverage collaborative filtering methods to predict user ratings for unseen products and evaluate the model's performance using Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) metrics.

Due to the large size of the dataset and computational constraints, collaborative filtering was selected as the primary approach, while content-based filtering was excluded after initial trials due to memory limitations. The final model's effectiveness is determined based on the RMSE score achieved on the Kaggle public leaderboard, where the predictions are submitted. A lower RMSE score indicates a more accurate model, reflecting the recommender system's ability to capture user preferences and deliver personalized product recommendations.

This report outlines the entire development process, from data exploration and preprocessing to model selection, evaluation, and discussion of the results. The comparison of multiple collaborative filtering models provides insights into the strengths and limitations of different approaches, ultimately guiding the selection of the best-performing model for the final submission.

Exploratory Data Analysis

The data was collected by crawling the Amazon website and contains product metadata and review information about 219,859 different products (Shareghi, 2025). It comprises 745,889 entries distributed across 7 columns. Each entry represents a user's interaction with a specific product, accompanied by metadata such as ratings and helpfulness votes. The columns include:

- user_id: A unique identifier assigned to each user, with a total of 2,000 distinct users.
- product_id: A unique identifier representing each product, with 201,325 unique products.
- product_name: The name of the product, with 178,037 unique product names.

- rating: The explicit rating given by the user on a scale of 1 to 5, with five possible unique values.
- votes: The number of votes received for the review, ranging from 0 to 348.
- helpful_votes: The number of helpful votes received, ranging from 0 to 296.
- ID: A unique identifier for each row, with 745,889 unique values.

No duplicate entries were identified during the initial data exploration, indicating the dataset's consistency. The dataset's memory usage is approximately 39.8 MB, which is manageable for further data processing.

The distribution of ratings reveals a significant imbalance towards higher ratings. The majority of the ratings are 5 stars (416,231 entries, 55.8%), followed by 4 stars (185,193 entries, 24.8%). Lower ratings are less frequent, with only 3 stars (79,870 entries, 10.7%), 2 stars (35,446 entries, 4.8%), and 1 star (29,149 entries, 3.9%). This skewed distribution suggests that users are more inclined to provide positive feedback, a common trend in user-generated product reviews.

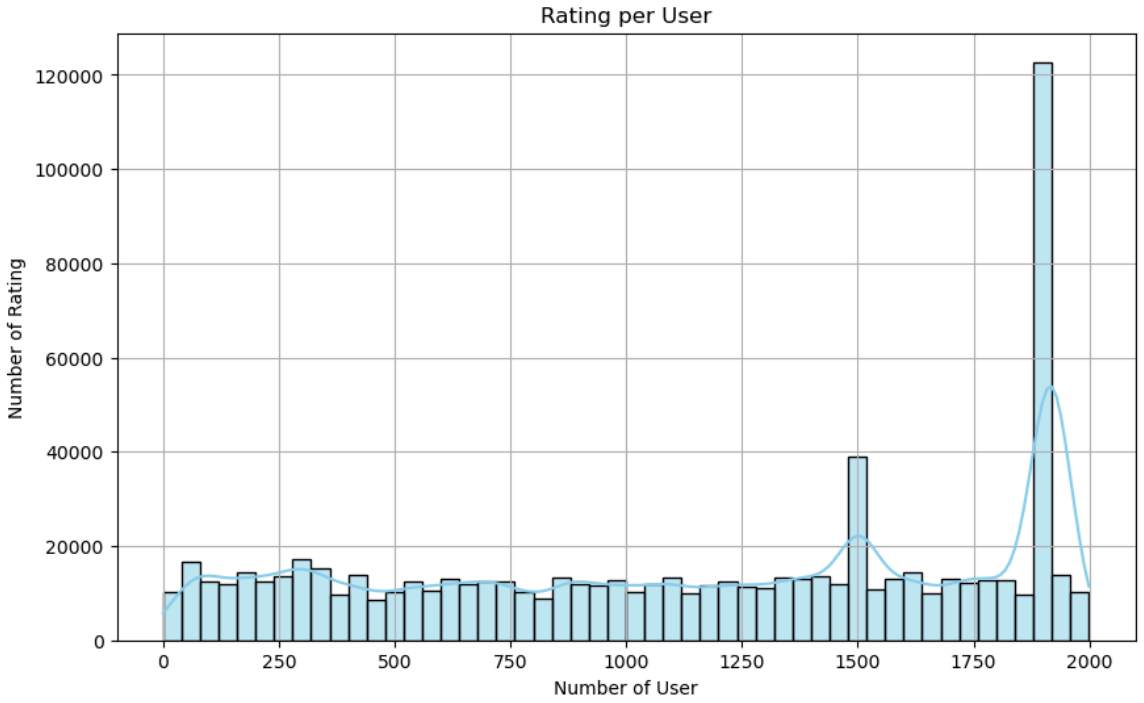

Figure 1. Rating per User Histogram

An important observation is that two user_id entries exhibit significantly higher activity compared to the rest of the users. These users have a disproportionately large number of interactions, making them power users who contribute a substantial portion of the dataset. Such users can heavily influence the model's predictions and must be considered during model evaluation to ensure the recommendations are not biased.

Data Cleaning

To address the imbalance caused by power users, a filtering process was applied to limit the maximum number of ratings per user. The number of ratings per user was first calculated, revealing that certain users had provided more than 20,000 ratings, far exceeding the typical number of interactions.

To prevent these users from dominating the dataset, the number of ratings per user was capped at 20,000 entries. This was achieved by grouping the dataset by user_id and applying a filtering function that retained only the first 20,000 entries for users exceeding this threshold. This step ensured that the dataset remained representative without excessive influence from outliers.

Following this filtering process, the dataset was prepared for the collaborative filtering model by converting it into the format required by the Surprise library. The rating_scale parameter was defined with a range of 1 to 5, reflecting the dataset's rating system. The dataset was then loaded into the Surprise library using the Dataset.load_from_df() method, which facilitated model training and evaluation in subsequent stages. Only three columns—user_id, product_id, and rating—were selected for this transformation, as these attributes are essential for collaborative filtering models. The exclusion of other columns helped reduce memory usage and ensured that the model focused solely on user-item interactions.

Methodology

Recommender systems are typically developed using two primary approaches: content-based filtering and collaborative filtering. Content-based filtering recommends items to users based on the similarity between item descriptions and the user's previous preferences. This approach relies heavily on item metadata such as product names, descriptions, or categories (Leskovec et al., 2020). However, the implementation of content-based filtering in this project was not feasible due to memory limitations encountered when computing cosine similarity on TF-IDF vectors. The process caused repeated system crashes, making it impractical to continue with this method.

Consequently, the project focused exclusively on collaborative filtering, which recommends items based on user-item interactions without requiring additional product information. Collaborative filtering leverages the patterns in user preferences to make predictions, assuming that users who agreed on past ratings are likely to agree on future ones (Aggarwal, 2016).

During the initial stages, the Alternating Least Squares (ALS) algorithm from the Implicit library was considered as one of the collaborative filtering models. ALS is a matrix factorization technique commonly used for implicit feedback data, where user interactions are more indicative than explicit ratings (Koren et al., 2009). However, the implementation of

ALS faced several challenges due to environmental inconsistencies between Google Colab and Monash HPC. The same codebase produced different outcomes across environments, posing a significant risk of non-reproducible results. To avoid inconsistencies, ALS was ultimately excluded from the final model selection.

Instead, the project utilized collaborative filtering models available in the Surprise library, which is widely used for supervised learning-based recommender systems. This approach ensures consistent performance across different environments and simplifies model implementation.

Model Selection

Three collaborative filtering models were selected for comparison: K-Nearest Neighbors (KNN), Non-negative Matrix Factorization (NMF), and Singular Value Decomposition (SVD). These models were chosen based on their popularity and effectiveness in previous recommender system research.

K-Nearest Neighbors (KNN)

KNN is a memory-based collaborative filtering method that recommends items by identifying the most similar users or items. The model calculates the similarity between users or items based on their rating patterns and makes predictions by averaging the ratings of the nearest neighbors. The similarity metric can be computed using various distance measures such as cosine similarity or Pearson correlation. KNN is intuitive and easy to implement but tends to perform poorly on sparse datasets due to its reliance on direct similarity comparisons (Paul, 2023).

In this project, the application of the KNN model was validated using 5-fold cross-validation. The following results represent the KNN performance evaluation on the train dataset.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.9879	0.9897	0.9840	0.9859	0.9890	0.9873	0.0021
MAE (testset)	0.7099	0.7113	0.7070	0.7088	0.7116	0.7097	0.0017
Fit time	0.56	0.74	0.73	0.72	0.72	0.69	0.07
Test time	4.53	3.50	4.21	3.50	3.85	3.92	0.40

Figure 2. KNN Cross-Validation Results

Non-negative Matrix Factorization (NMF)

Non-negative Matrix Factorization is a model-based collaborative filtering method that decomposes the user-item rating matrix into two lower-dimensional non-negative matrices: user-feature matrix and item-feature matrix. The model optimizes these matrices by minimizing the reconstruction error between the original matrix and the product of the two decomposed matrices. NMF is particularly suitable for non-negative data such as ratings, as it produces interpretable latent features. The NMF objective function can be formulated as:

$$\begin{aligned} \text{Minimize} \quad & J = \tfrac{1}{2}||R - UV^T||^2 \\ & \text{subject to:} \\ & U \geq 0 \\ & V \geq 0 \end{aligned}$$

where R is the original user-item rating matrix, U is the user-feature matrix, V is the item-feature matrix, and $||.||^2$ denotes the Frobenius norm (Aggarwal, 2016).

The application of the NMF model was also validated using 5-fold cross-validation. The following results represent the NMFperformance evaluation on the train dataset.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	1.0337	1.0464	1.0440	1.0365	1.0410	1.0403	0.0047
MAE (testset)	0.7915	0.8063	0.8041	0.7952	0.8020	0.7998	0.0056
Fit time	23.76	24.09	25.07	24.67	23.88	24.29	0.50
Test time	1.36	1.10	0.94	0.92	0.95	1.05	0.16

Figure 3. NMF Cross-Validation Results

Singular Value Decomposition (SVD)

Singular Value Decomposition is another model-based collaborative filtering technique that factorizes the user-item rating matrix into three matrices: user-feature matrix, diagonal matrix of singular values, and item-feature matrix. SVD is widely used in recommender systems due to its ability to reduce data dimensionality and capture latent user preferences. Unlike NMF, SVD allows negative values, making it more flexible in capturing complex user-item interactions. The decomposition formula for SVD is:

$$\begin{aligned} R &\approx (Q_k \Sigma_k) P_k^T \\ &= UV^T \end{aligned}$$

where Q is the matrix of the top k left singular vectors, representing users and P^T is the transpose of the matrix containing the top k right singular vectors, representing items. While U is the user-feature matrix and V is the item-feature matrix (Aggarwal, 2016).

Like the previous methods, the application of the SVD model was also validated using 5-fold cross-validation. The following results represent the SVD performance evaluation on the train dataset.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.9205	0.9150	0.9130	0.9180	0.9187	0.9171	0.0027
MAE (testset)	0.6817	0.6773	0.6771	0.6795	0.6827	0.6797	0.0023
Fit time	10.58	10.56	10.52	10.95	10.48	10.62	0.17
Test time	1.22	1.21	1.22	1.23	1.20	1.22	0.01

Figure 4. SVD Cross-Validation Results

Hyperparameter Determinations

Following the model selection phase, SVD demonstrated the best performance among the three models. Consequently, further tuning of SVD hyperparameters was conducted to optimize its predictive accuracy.

Hyperparameter tuning is the process of selecting the best combination of model parameters that yield the highest performance. In this project, GridSearchCV was used to perform an exhaustive search over a specified parameter grid. GridSearchCV is a hyperparameter optimization technique that systematically tests multiple combinations of hyperparameters and selects the combination that maximizes model performance based on cross-validation results (Shah, 2025).

The hyperparameters tuned in this study include: n_factors, which is the number of latent factors used in the matrix factorization.; n_epochs, which is the number of training iterations; lr_all, which is the learning rate applied to all parameters; and reg_all, which is the regularization term applied to all parameters.

The grid search process identified the optimal hyperparameters as:

- n_factors = 140
- n_epochs = 120
- lr_all = 0.085
- reg_all = 0.02

These hyperparameters were then applied to the final SVD model, which was subsequently validated using 5-fold cross-validation to ensure consistent performance.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.8299	0.8303	0.8355	0.8306	0.8295	0.8312	0.0022
MAE (testset)	0.6034	0.6028	0.6056	0.6030	0.6013	0.6032	0.0014
Fit time	64.73	65.45	64.43	66.87	64.64	65.23	0.89
Test time	0.85	0.89	0.93	1.35	0.91	0.99	0.18

Figure 5. Best SVD Cross-Validation Results

Model Training

Once the optimal hyperparameters were determined, the final model was trained on the entire dataset without splitting into training and testing subsets. The decision to utilize the full dataset was based on the objective of maximizing the model's exposure to the available data, thereby improving its ability to capture latent user preferences. This approach is commonly used when the primary goal is to generate recommendations rather than evaluate predictive performance on unseen data.

The training process was conducted using the Surprise library's build_full_trainset() method, which converts the entire dataset into a suitable format for model training. The

model was then fitted using the fit() method, allowing the SVD algorithm to learn latent factors from all available user-item interactions.

By leveraging the complete dataset, the model could generate recommendations with higher accuracy and better personalization. However, this approach comes at the cost of not being able to assess the model's generalization performance, which is typically evaluated through a separate validation set.

Test Predictions

After the model was trained, the final step was to generate predictions on the test dataset. The SVD model with the best hyperparameters was applied to predict user ratings for each user-product pair in the test set. The predictions were made by iterating over each entry in the test dataset and using the model's predict() method to estimate the rating.

The resulting predictions were then stored in a new CSV file as required by the competition guidelines. The CSV file contained two columns:

- ID: The unique identifier for each user-product pair.
- rating: The predicted rating generated by the model.

This file was subsequently submitted to the Kaggle competition page, where the RMSE score was calculated to evaluate the model's performance.

Kaggle Competitions

As part of the assignment, predictions generated by the model were submitted to the Kaggle competition page for external evaluation. Throughout the project, a total of 79 submissions were made since the release of Assignment 2. The lowest RMSE score achieved was 0.87594, while the highest RMSE score recorded was 2.83968. These submissions were part of the trial-and-error process involved in developing and tuning the recommender system.

Ultimately, two submission files were selected in Kaggle for evaluation. The first file represents the latest submission generated by the model using the best hyperparameters, yielding an RMSE score of **0.87694**. This file was submitted to both Kaggle and Moodle, aligning with the final version of the submitted Jupyter Notebook code. The second file contains predictions from the SVD model with the lowest RMSE score of 0.87594 achieved during the experimentation process. However, this score could not be replicated in the final submission due to differences in the computing environment. The final experiment was conducted on Google Colab, while the best previous result was obtained using Monash HPC, which was likely under heavy load and unable to connect at the time.

At the time this report was written, the best-performing model ranked second place on the Kaggle leaderboard, although the ranking remains subject to change as more submissions are made by other participants.

Summary

In this project, a collaborative filtering-based recommender system was successfully developed to predict user ratings for products in the Amazon product dataset. The implementation involved comparing three different collaborative filtering methods: K-Nearest Neighbors (KNN), Non-negative Matrix Factorization (NMF), and Singular Value Decomposition (SVD). After evaluating the models through 5-fold cross-validation, SVD was selected as the final model due to its superior predictive performance.

Hyperparameter tuning using GridSearchCV further optimized the SVD model, yielding the best combination of parameters for the dataset. The final model was trained on the entire dataset and used to generate predictions for the test set. The resulting predictions were submitted to the Kaggle competition page, where the model achieved a lowest RMSE score of 0.87594, placing second on the leaderboard at the time this report was written.

The iterative trial-and-error process, along with continuous hyperparameter tuning, played a critical role in improving the model's accuracy. This project highlights the importance of model selection, hyperparameter optimization, and performance evaluation in building effective recommender systems. Further improvements could involve experimenting with deep learning-based collaborative filtering methods or hybrid recommender systems to achieve even better performance.

References

Monash University. (2025). *Week 4: Recommender System* [Lecture slides]. Moodle. https://learning.monash.edu/course/view.php?id=25414&section=20

Raza, S., Rahman, M., Kamawal, S., Toroghi, A., Raval, A., Navah, F., & Kazemeini, A. (2024). *A Comprehensive Review of Recommender Systems: Transitioning from Theory to Practice*. https://doi.org/10.48550/arXiv.2407.13699

Hasan, E., Rahman, M., Ding, C., Huang, J. X., & Raza, S. (2024). *Review-based Recommender Systems: A Survey of Approaches, Challenges and Future Perspectives*. https://doi.org/10.48550/arXiv.2405.05562

Shareghi, E. (2025). *ITI5212 Rec Sys 2025: Recommender System Challenge for ITI5212* [Competition page]. Kaggle. https://www.kaggle.com/competitions/iti-5212-rec-sys-2025

Leskovec, J., Rajaraman, A., & Ullman, J. D. (2020). *Mining of Massive Datasets* (3rd ed.). Cambridge University Press. Retrieved from http://www.mmds.org/ (Chapter 9: Recommender Systems)

Aggarwal, C. C. (2016). *Recommender Systems: The Textbook*. Springer. https://doi.org/10.1007/978-3-319-29659-3

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix Factorization Techniques for Recommender Systems. Computer, 42(8), 30–37. https://doi.org/10.1109/MC.2009.263

Paul. (2023, January 19). *Collaborative Filtering Recommenders*. Medium. https://medium.com/@paul0/collaborative-filtering-25d181107082

Shah, R. (2025, February 15). *Tune Hyperparameters with GridSearchCV*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/06/tune-hyperparameters-with-gridsearchcv/

Acknowledgement

I acknowledge the use of AI assistance, specifically ChatGPT, for paraphrasing and refining grammar to enhance the clarity and coherence of this report.