Empirical Analysis of Movie Recommendation Algorithms

Prepared for

Prof. Mucahit Cevik

Prepared by

Jason Yu, Jiho Choi

Toronto Metropolitan University

DS8001 – Fall 2025

Project Report

Image Source:https://www.deviantart.com/logofeveryt/art/The-Netflix-Login-Background-Canada-2024-1029429319

Contents

1 Introduction

1.1 Motivation and Problem Definition

Recommendation systems have become increasingly important in guiding user decision-making across various platforms, including streaming and e-commerce. This project aims to compare different recommendation algorithms in terms of their ability to predict user movie ratings. Both rating-based and content-based algorithms will be explored, and unique user cases will be identified with potential strategies to address them.

1.2 Dataset Description

This project uses the MovieLens dataset from GroupLens Research. The dataset contains approximately 100,000 ratings and 3,683 user-generated tags across 9,742 movies, rated by 610 users on a 0.5–5.0 star scale, with the following files:

- Rating.csv
- Movies.csv
- Tags.csv
- Links.cvs

There are two sizes of this dataset: the smaller dataset (100,000 ratings) will be used for model development and testing to speed up the process, and the larger dataset with 33,000,000 ratings will be used later to evaluate its performance and scalability.

1.3 Objectives

- Conduct a brief literature review on new recommendation methods developed after the Netflix Prize.
- Compare the performance of multiple recommendation algorithms
- Measure accuracy and scalability of each algorithm

2 Literature Review

The Netflix Prize highlighted explicit movie rating prediction as a core benchmark and emphasized the importance of strong baselines. The original Netflix system already relied on relatively simple statistical components, like the global average rating, per-movie average, and user- and movie-specific bias terms (Koren, Bell, & Volinsky, 2009).

Su and Khoshgoftaar (2009) surveyed collaborative filtering techniques and categorized them into memory-based, model-based, and hybrid algorithms. Memory-based CF is predicted based on neighbourhood models, such as user-based and item-based models. The user-based model focuses on the ratings of similar users, while the item-based model focuses on the ratings of similar items. Both calculate similarities, such as cosine similarity, and then predict based on those similar ratings. These methods are conceptually simple and interpretable and remain standard baselines for more advanced recommender systems.

In parallel, content-based recommendation uses item features instead of cross-user overlap. Google's content-based filtering tutorial describes representing items as feature vectors (e.g., genres or categories), constructing a user profile from previously interacted items, and scoring candidate items by the similarity between the user profile and item vectors (Google Developers, n.d.). IBM's overview similarly defines content-based filtering as matching a learned user preference profile to item profiles built from metadata such as tags or textual descriptions (IBM, n.d.). These sources emphasize that content-based methods are particularly useful when collaborative data is sparse or when recommending new or niche items, provided item features are informative.

3 Methodology

3.1 The Naive Approach

First, a very natural Approach is to just return thr global average score for a given movie. For each movie we compute the mean of all rating it recevied in the training dataset, and this value will become the prediction value for all user for that given movie. This approach works well when a movie that is considered very good or very bad by the general public. For example the movie "The Shawshank Redemption" is very popular and highly rated, so a new person watching it are likely to give it a high rating which is close to the existing global mean. Here is the formula:

$$\bar{r}_i = \frac{1}{n_i} \sum_{u \in U_i} r_{ui}$$

If we run the code we will get the following

3.2 The Naive Approach with Smoothing

While the Naive Approach or the global average Approach is easy to code and very intuitive, we notice one critical issue: we are assuming all movies as equally reliable, regardless of how many ratings they have received. For example, Movie A has 4 rating of 5.0 and Movie B have 3000 rating of 4.5. Under our Naive model, Movie A will always have a high rating of 5.0 and will always outrank movie B, which has 3000 ratings. This make the rating of movie A less creditable. So the prediction on movies with low number of rating in our dataset will be unstable or driven far away from its true rating.

We will attempet so solve this problem by introducting a smoothing parameter. We let k denote this parameter.

The formula is the following:

$$\hat{r}_i = \frac{\bar{r}_i \cdot n_i + k \cdot \mu}{n_i + k}$$

- n_i is number of rating for movie i
- $\bar{r}_i$ is average rating of movie i
- μ is global average rating
- k is the smoothing parameter

The idea of this algorithm is to use the global mean as a stabilizing factor. we compute a weighted average between movie's taring and the global mean of all movies. Movie with large number of ratings like movie B from the previous example will retain its rating and movie with small number of rating like movie A, will be pulled towards the global mean.

3.3 User-based Model

As explained in the literature review section, this model will adopt an idea from A survey of collaborative filtering techniques (Su and Khoshgoftaar, 2009). Construct a user-item rating matrix R from the training data; Mark missing entries as NaN. For each user u, compute the mean rating $\bar{r_u}$ and mean-center the ratings: $R'_{ui} = r_{ui} - \bar{r}_u$

Then we compute user-user cosine similarities between rows of R'. Prediction equation:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in N_u(i)} s(u,v)\ (r_{vi} - \bar{r}_v)}{\sum_{v \in N_u(i)} |s(u,v)|}$$

- $Nu(i)$ is the neighbour set
- $s(u,v)$ is cosine similarity.

3.4 Item-based Model

As explained in the literature review section, this model will adopt an idea from A survey of collaborative filtering techniques (Su and Khoshgoftaar, 2009). Item-based model uses the same structures as the user-based model, but with items as explained.

Construct a user-item rating matrix R from the training data; Mark missing entries as NaN. For each user u, compute the mean rating $\bar{r_u}$ and mean-center the ratings: $R''_{ui} = r_{ui} - \bar{r}_i$

Then we compute user-user cosine similarities between rows of R'. Prediction equation:

$$\hat{r}_{ui} = \bar{r}_i + \frac{\sum_{j \in N_i(u)} s(i,j)\ (r_{uj} - \bar{r}_j)}{\sum_{j \in N_i(u)} |s(i,j)|}$$

- $Nj(u)$ is the neighbour set - $s(i,j)$ is cosine similarity

3.5 Bias-Based Model

Even with smoothing naive approach, we are still making a impossible assumption on the user and movie, that they are uniformly equal. For example, some users tends to rate every movie higher than average, while some routinely scored films more harshly. Likewise, some movies tended to attract universally high ratings, while others were consistently judged more critically by the general audience, even after accounting for the number of ratings.

Consider the following more detailed senerial: User A give every movie a rating of 5, he/she comes accros, while User B give everything a 2, because he does not like watching movies that is longer than 1h. Even if both users watch the same movie with a smoothed average of 3.0, treating them identically misses an important reality: User A is likely to rate it above 3.0, and User B is likely to rate it below 3.0. Likewise, some movies like classic favorites, or universally praised films tend to attract higher ratings regardless of who watches them, while niche or polarizing movies may be consistently rated lower.

With this knowledge, we introduce bias-based model. We decomposes each predicted rating into three intuitive components: μ as the Global Average Rating of All Movies b_i as the bias of user i, in other words how much higher or lower the user tends to rate movies in general b_u as the bias on movie u, in other words how much higher or lower the movie tends to be rated than the global mean (Koren, Bell, & Volinsky, 2009)

The formula is the following:

$$\hat{r}_{ui} = \mu + b_i + b_u$$

3.6 Bias-Based + Matrix Factorization Model

From the bias based we see it capture biass very well. Improve the RMSE. Let's see another important observation. Users who gave high ratings to science-fiction films also tended to enjoy action movies. Meanwhile, users who preferred slow-paced dramas often disliked comedies. So we can notice each user have a "taste" on movies. From our bias model this feature is not yet been captured. Because in the bias model, we assume user have the same level of "taste" for all kind of movies.

The challenge here is we do not have the exact information on what kind of movie each user liked. We have only on our hand the ratings, this motivated us to explore matrix factorization. A technique famously central to the Netflix Prize–winning solution. Matrix factorization assumes that there are a small number of hidden dimensions that describe user preferences and movie characteristics. The formula is the following:

$$\hat{r}_{ui} = \mu + b_i + b_u + {p_u}^T q_i$$

Here p_u captures user $u's$ preference strengths across all latent dimensions, and q_i captures how strongly movie i expresses each latent dimension . The dot product ${p_u}^T q_i$ aligns users with movies they are predisposed to enjoy.

Here we also introduce 4 new tuning parameters:

- n_factors:Determines how many hidden "taste dimensions" the model learns for users and movies.
- n_epochs:Controls how many times the model goes through the training data to improve its parameters.
- lr: Sets how large each update step is when adjusting the model during training.
- reg: Prevents overfitting by shrinking extreme parameter values and keeping the model stable.

3.7 Tuning

For the tuning process, we applied a systematic tuning grid for each algorithm requires tuning, evaluating a range of possible parameter values rather than relying on a single preset choice. Each parameter combination was tested using consistent evaluation procedures, and the resulting RMSE values were recorded. The parameter setting that produced the lowest RMSE was then selected as the optimal configuration for that algorithm. This approach ensures that tuning is both data-driven and objective, allowing each model to operate at its best possible performance level before final comparison.

4 Experimental Setup

4.1 Preparation of the Dataset

For the data preparation, we first loaded the four core MovieLens datasets:ratings, movies, tags, and links. Then examined their structure to understand the available variables and confirm data completeness. We then converted all timestamp fields into readable datetime values to make temporal patterns easier to analyze. After checking for missing values across all columns, we merged the ratings and movies tables using movieId to create a single, consolidated dataset containing user ratings alongside movie titles and genre information. We also extracted and listed all unique genres by splitting the genre strings into individual labels. Once the dataset was fully cleaned, inspected, and merged, we saved the final version as Master_small.csv and Master_large.csv for use in the modeling and analysis stages that follow.

4.2 Testing Method

The dataset will be split into 80% training and 20% testing subsets, and implemented using Root Mean Square Error (RMSE) as the primary evaluation metric.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

4.3 Testing accuracy and Run time efficency

For the testing stage, we evaluated all six algorithms using both Master_small and Master_large to compare performance across different dataset sizes. In addition to full-dataset testing, we conducted a scaling experiment using Master_large by selecting the first 100,000, 390,000, 680,000, 970,000, 1,260,000, 1,550,000, 1,840,000, 2,130,000, 2,420,000, 2,710,000, and 3,000,000 rows. These progressively larger subsets allowed us to systematically measure how increasing data volume affects both prediction accuracy and runtime efficiency. This setup provides a clear understanding of each algorithm's scalability, computational cost, and robustness as the dataset grows.

5 Results

5.1 Accuracy

The algrithom's RMSE performed on the small dataset is the following:

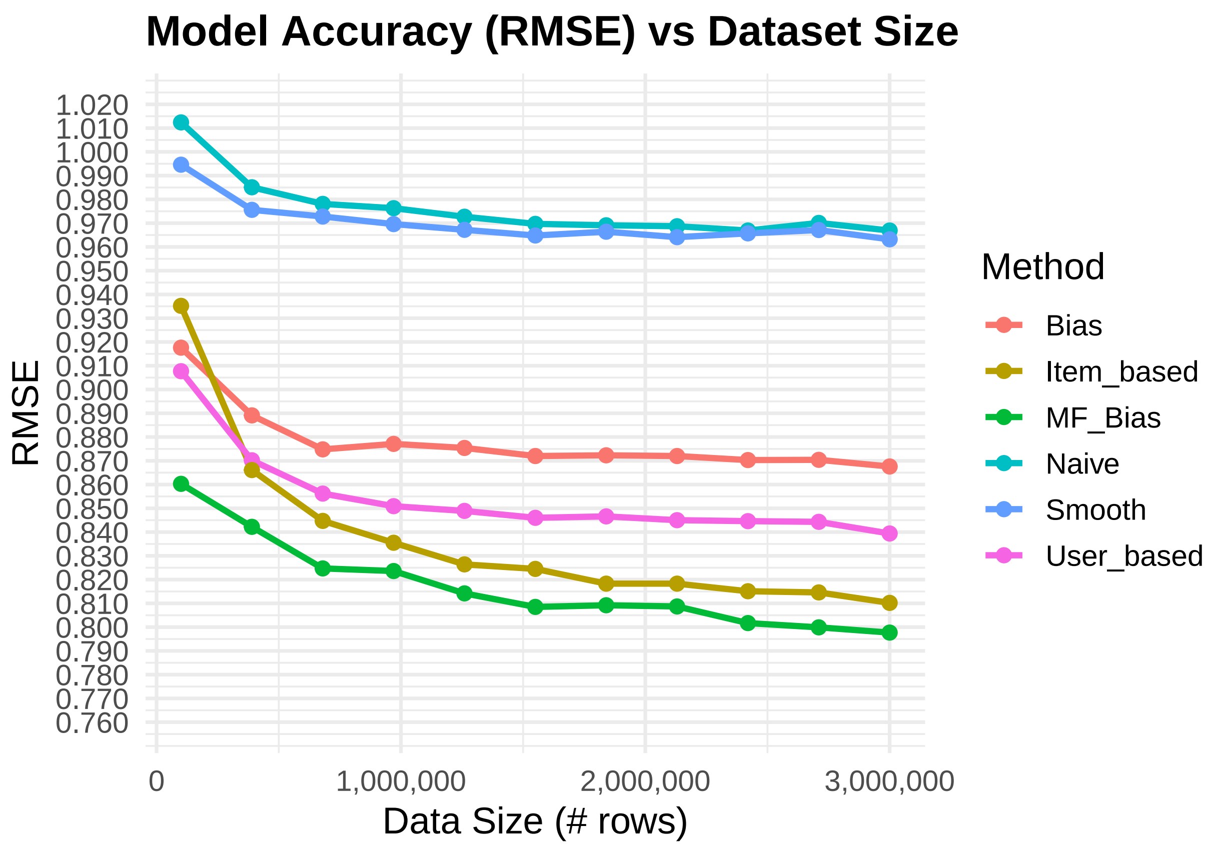
Method	RMSE
Naive	0.9689
Smoothing	0.9541
Bias	0.8892
User Based	0.8612
Item Based	0.8994
MF+Bias	0.8438

On the small dataset, the models show a clear ranking in accuracy. The naive and smoothing models perform the worst because they only use movie averages and cannot capture user differences. The bias model improves results by accounting for user and movie tendencies. User-based and item-based collaborative filtering perform even better because they use similarity between users or movies. Matrix Factorization with Bias achieves the best RMSE, showing that learning latent factors provides the most accurate predictions even with limited data.

Let's see how dataset size effect the performance of the algrithem

Data_Size	Naive	Smooth	User_based	Item_based	Bias	MF_Bias
100000	1.0124	0.9946	0.9077	0.9352	0.9176	0.8603
390000	0.9851	0.9756	0.8702	0.8661	0.8891	0.8422
680000	0.9781	0.9728	0.8562	0.8447	0.8748	0.8247
970000	0.9763	0.9696	0.8509	0.8355	0.8771	0.8236
1260000	0.9727	0.9672	0.8489	0.8264	0.8754	0.8142
1550000	0.9697	0.9648	0.8460	0.8245	0.8720	0.8085

Data_Size	Naive	Smooth	User_based	Item_based	Bias	MF_Bias
1840000	0.9691	0.9664	0.8466	0.8183	0.8723	0.8092
2130000	0.9687	0.9641	0.8450	0.8183	0.8720	0.8087
2420000	0.9669	0.9657	0.8446	0.8151	0.8703	0.8017
2710000	0.9701	0.9671	0.8443	0.8146	0.8704	0.7999
3000000	0.9669	0.9632	0.8394	0.8102	0.8676	0.7977
33000000	0.9655	0.9648	NA	NA	0.8652	0.7732

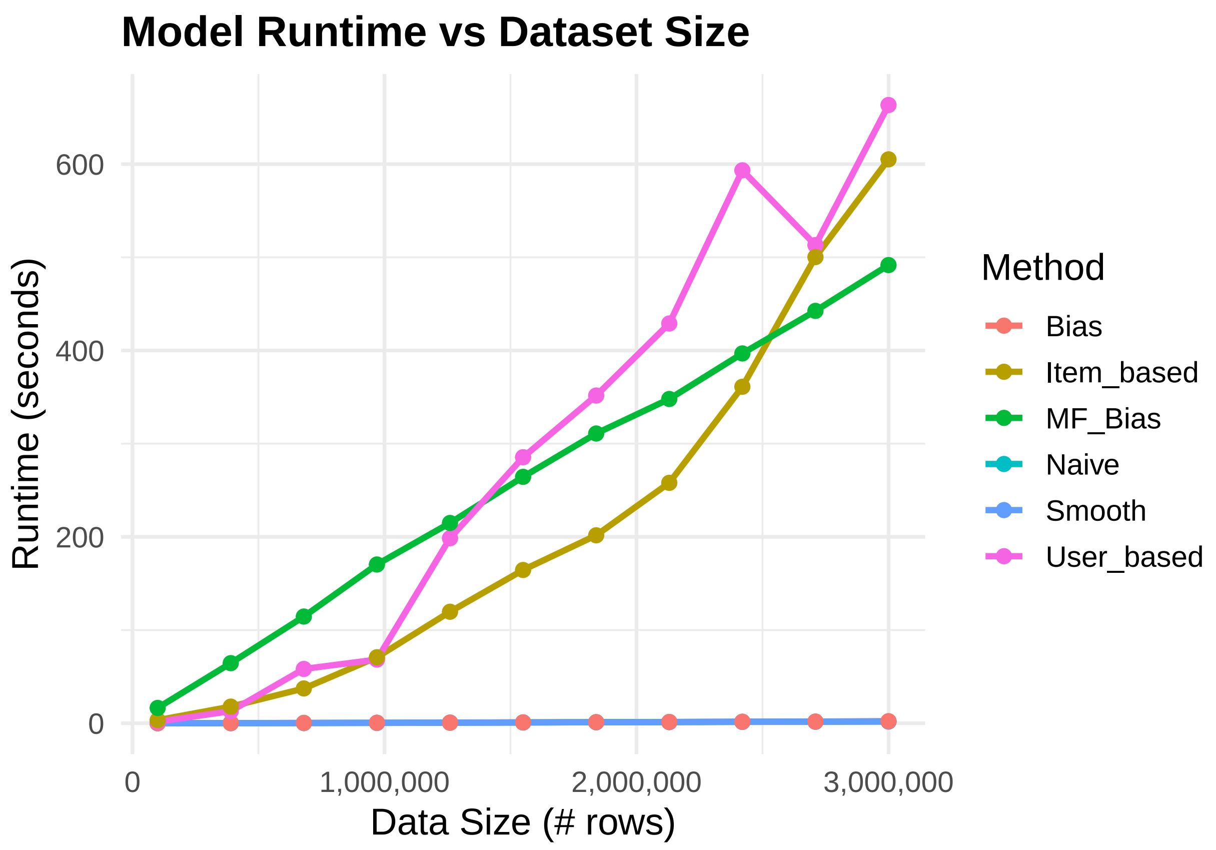


When we test the models on larger subsets of the dataset, we see that accuracy generally improves as more data becomes available. The naive and smoothing models improve only slightly, while the bias and neighborhood-based methods show bigger gains early on and flats out later. User-based and item-based models eventually run into memory limits, which is why the largest size shows NA. Matrix Factorization continues to improve steadily and remains the best-performing model at every data size, showing that it scales well and benefits the most from additional data.

5.2 Runtime

Data_Size	Naive	Smooth	Bias	User_based	Item_based	MF_Bias
100000	0.0271	0.0241	0.0343	1.6206	3.3163	16.4436
390000	0.1141	0.1149	0.1457	12.9707	17.8395	64.4759
680000	0.2473	0.2418	0.3130	58.2569	37.3038	114.4774
970000	0.4185	0.3976	0.4840	68.3280	70.8036	170.3497
1260000	0.5865	0.6059	0.6647	198.5395	119.6258	214.8889

Data_Size	Naive	Smooth	Bias	User_based	Item_based	MF_Bias
1550000	0.8269	0.8492	0.9058	285.4511	164.4532	264.4086
1840000	1.0940	1.0259	1.1256	351.6410	201.6709	310.8954
2130000	1.2901	1.2345	1.3295	428.9599	257.9948	347.9719
2420000	1.4966	1.4768	1.5914	593.3156	361.0794	396.8294
2710000	1.6495	1.6498	1.7919	513.1156	500.2761	442.5002
3000000	1.8613	1.8925	2.2211	663.4323	605.1232	491.6662
33000000	35.2422	32.4903	42.4668	NA	NA	5764.8565



On the small dataset, the naive, smoothing, and bias models all run extremely fast, finishing in well under a second. These models rely on simple aggregations such as computing movie means or user/movie biass, so they scale efficiently and require very little computation. In contrast, the user-based and item-based models take much longer even at small size because they must calculate similarities between users or between movies. Matrix Factorization with Bias is the most expensive method on the small dataset because it trains multiple latent factors through many gradient descent iterations.

As the dataset grows, the differences between the algorithms become more noticeable. The naive, smoothing, and bias models continue to run quickly and increase only slightly in runtime as data size grows. This shows that simple statistical models are very efficient and scale very well.

Since there are more data means more users or movies to compare against, User-based and item-based methods increase much faster in runtime, and almost look like a exponential growth. Their runtime becomes large enough that at 33 million rows, these models cannot complete due to memory limits.

The MF+Bias model also increases steadily in runtime because it must repeatedly update large matrices as the dataset gets bigger. Although slower than the simple models, it remains manageable and completes up to 33 million rating. At the largest size, MF+Bias still finishes, but with very long training time.

6 Conclusions

6.1 Lesson Learned

This project explored several recommender system algorithms. From simple average-based models to neighborhood methods and finally Matrix Factorization with Bias. Evaluation on both their accuracy and scalability using small and large versions of the MovieLens dataset. The results provide a clear takeaway: model complexity strongly affects both predictive performance and computational cost, and it is constant trade off between the two.

Simple model such as naive, smoothing and bias baseline are extremely fast and scale effortlessly, but they offer only limited accuracy because they cannot capture user-level or item-level structure.

Neighborhood-based methods (user-based and item-based) provide better accuracy on small and medium datasets, but their computational requirements grow exponentially with dataset size, making them impractical at a large scale.

Matrix Factorization with Bias achieves the best accuracy across all dataset sizes, improving steadily as more data is added. Although it is slower than the simple models, MF remains computationally feasible up to millions of rows and provides a strong balance between accuracy and scalability.

The main takeaway from this study is that the choice of recommendation algorithm must balance predictive performance, computational costs, and dataset size. For small datasets or real-time computation, simple bias-based models may be sufficient. For large and growing datasets, Matrix Factorization with Bias offers the best long-term value, with higher accuracy and effectively using additional data. Overall, the project highlights how different modeling approaches behave in practice and shows us the importance of scalability when designing real-world recommender systems.

6.2 Limitations and Future Work

A key limitation of our project is the handling of the tags data. The tags provided in the MovieLens dataset are written in free-form natural language, meaning users can type anything they want to describe a movie. Because these tags are unstructured, inconsistent, and linguistically complex, our current models are unable to process or incorporate them effectively. We do not perform text normalization, semantic analysis, or embedding-based interpretation of tag meanings, which prevents us from using this potentially rich source of information. Although tags could significantly enhance recommendations by capturing themes, emotions, or subtle characteristics not reflected in numeric ratings, our project does not include natural language processing techniques needed to utilize them.

Another limitation arises from computational constraints when evaluating the scalability of our algorithms. Although the full "large" dataset contains over 30 million ratings, our hardware resources limit us to processing only the first 3 million rows, approximately 1/11 of the entire dataset. As a result, our scalability assessment does not fully reflect how the algorithms would behave when trained on the complete dataset. Certain models, especially matrix factorization and bias-based methods, may exhibit different time complexities or memory requirements at larger scales. Therefore, while our experiments provide insight into performance trends, they cannot definitively represent true large-scale behavior due to the reduced dataset size imposed by computational limitations.

Implementing the hybrid approach that combines user- and item-based models will remain as future work. As demonstrated by Su and Khoshgoftaar (2009), such models require a hybrid design to achieve more realistic and unbiasd results.

7 Citation

- Google Developers. (n.d.). Content-based filtering | Machine Learning | Google for Developers. Retrieved from https://developers.google.com/machine-learning/recommendation/content-based/basics
- IBM. (n.d.). What is content-based filtering? IBM Think. Retrieved from https://www.ibm.com/think/topics/content-based-filtering
- Y. Koren, R. Bell, and C. Volinsky, "Matrix Factorization Techniques for Recommender Systems," IEEE Computer, vol. 42, no. 8, pp. 30–37, Aug. 2009. https://datajobs.com/data-science-repo/Recommender-Systems-%5BNetflix%5D.pdf
- Su, X., & Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. Advances in Artificial Intelligence, 2009(1), 421425. https://doi.org/10.1155/2009/421425
- GroupLens. (2018). MovieLens latest datasets. GroupLens. https://grouplens.org/datasets/movielens/latest/