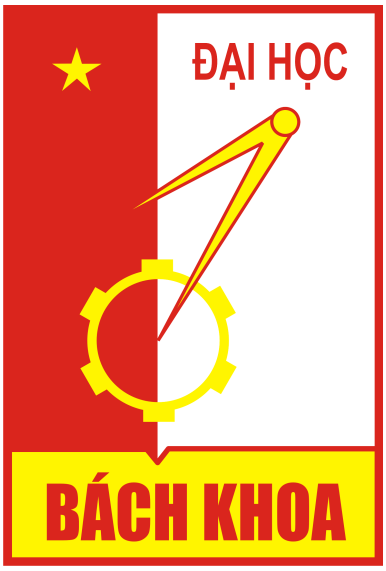
HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY
FACULTY OF COMPUTER SCIENCE

o0o



PROJECT REPORT:
MOVIE RECOMMENDATION SYSTEM

Course: Machine Learning - IT3190E
Class Code: 141178
Supervisor: Associate Prof Than Quang Khoat
Member of our group:

Trinh Hoang Giang	20214893	giang.th214893@sis.hust.edu.vn
Ho Ngoc Anh	20214877	anh.hn214877@sis.hust.edu.vn
Hoang Thanh Dat	20214899	dat.ht214889@sis.hust.edu.vn
Lang Van Quy	20214928	quy.lv214928@sis.hust.edu.vn
Vu Lam Anh	20214876	anh.vl214876@sis.hust.edu.vn

Ha Noi 2023

Abstract

Industry 4.0, recommender systems play an important role in numerous sectors. The use of recommender systems to enhance user experience is widespread in social media, e-commerce, and online platforms. Users are to receive personalized recommendations from the recommender system based on their preferences, behaviors, or previous interactions. The goal of this project is to explore two common methods for movie recommendation systems: collaborative filtering and content-based filtering. Additionally, we investigate several forms of collaborative filtering, such as matrix factorization-based Singular Value Decomposition and Gradient Descent. We also go through how these strategies' performance was judged using a variety of evaluation criteria that were selected and used with recommender systems in mind.

Contents

1 Introduction to Movie Recommendation System

1.1 Short description

A movie recommendation system, or a movie recommender system, is an ML-based approach to filtering or predicting the users' film preferences based on their past choices and behavior. It's an advanced filtration mechanism that predicts the possible movie choices of the concerned user and their preferences toward a domain-specific item (movie).

1.2 Input and output

There are two main elements in every recommendation system: **users** and **items**. In this case, the system generates movie predictions for its users, while items are the movies themselves. The movie recommendation system takes **input** is the data about the user's past choices and also the features of the movie, then it generates the **output** is the filtration or prediction for the user's film preferences.

When building this system, we have to deal with many data about the features of the movie (actor, rating, genre,....) and also the past choice of users. Therefore, the data type of input may be a string (actor name, genre,...) or real value(rating, year,...), or also a boolean(like/dislike,...).

2 Dataset

2.1 About the data

The primary dataset used in this project comes from MovieLens generated by GroupLens. It has two versions, the full version has a collection of 26 million reviews over 45,000 different movies by 270,000 users. Due to limited computing power, a smaller version, which contains 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users who joined MovieLens in 2000, is mainly used in this project instead.

The dataset, put in dataset, consists of the following files:

- ratings.dat: contains all ratings following format UserID::MovieID::Rating::Timestamp.
- users.dat: contains user information regarding User ID, Gender, Age, Occupation, and Zip-code.
- movies.dat: contains movie information regarding Movie ID, Title, and Genres.

2.2 Data pre-processing

We divided the dataset into test and training on the basis of UserID, using 90% of each user's reviews for training the model and the remaining 10% for validating the model's accuracy. Using the train-test-split technique from the Scikit-Learn package, we accomplish the aforementioned. The feature on which the dataset is split into training and testing is specified by the function's stratify property.

Due to unintentional duplicate entries and test entries, we discovered that several MovieIDs in this collection do not relate to movies. We decided to treat these erroneous movies as movies with no ratings in order to preserve consistency.

2.3 Data exploration

- **Distribution of movies genres**
  In the dataset, there are 18 genres total, but they have unequal representations. The top three genres with the most movies are Drama, Comedy, and Action.

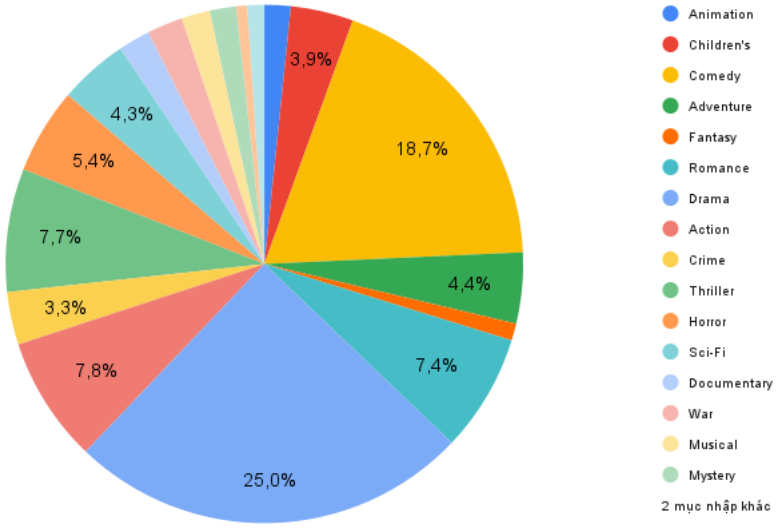


Figure 1: Distribution of movies genres

- **Distribution of ratings**
  Reviews are mostly towards the higher end, 3, 4 or 5. The average rating is 3.58.

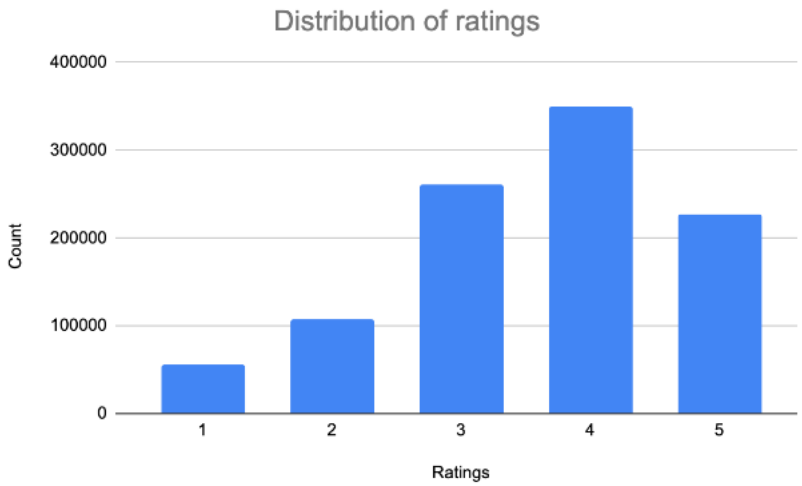


Figure 2: Distribution of ratings

- **Distribution of ratings per users**
  Most users review under 100 movies. On average, a user rates 165 movies, and a movie receives 253 reviews. This is caused by extreme outliers not displayed in the figure below (a user rates well over 3000 movies).

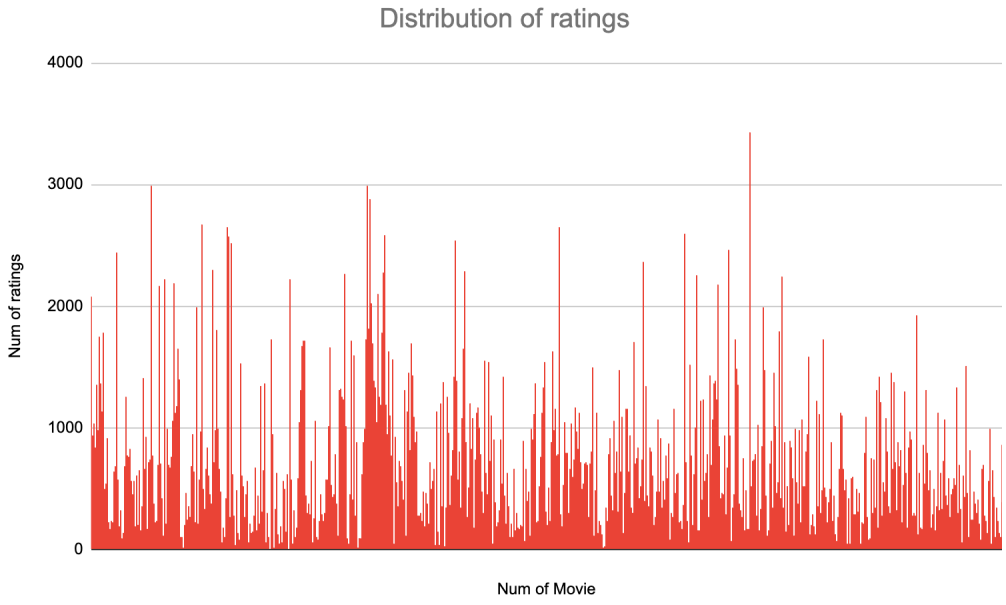


Figure 3: Distribution of ratings per users

3 Content-Based Filtering

3.1 Introduction to Content-Based Filtering

Content-based filtering is a type of recommender system that attempts to guess what a user may like based on that user's activity. Content-based filtering makes recommendations by using keywords and attributes assigned to objects in a database and matching them to a user profile. The user profile is created based on data derived from a user's actions, such as purchases, ratings (likes and dislikes), downloads, items searched for on a website and/or placed in a cart, and clicks on product links. Advantages of content-based recommender system are following:

- Because the recommendations are tailored to a person, the model does not require any information about other users. This makes scaling of a big number of people more simple.
- The model can recognize a user's individual preferences and make recommendations for niche things that only a few other users are interested in.
- New items may be suggested before being rated by a large number of users, as opposed to collective filtering.

3.2 Content-Based Filtering Implementation

3.2.1 Utility Matrix

In Recommender Systems, building the Utility Matrix is paramount. There are two main entities in Recommendation Systems, users and items. Each user will have a degree of preference for each item. This interest level, if known in advance, is assigned a value for each user-item pair. Assuming that interest is measured by the user rate value for the item, let's call this value rating. The collection of all ratings, including the unknown values that need to be predicted, forms a matrix called the Utility Matrix.

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	5	5	2	0	1	?	?
i_1	4	?	?	0	?	2	?
i_2	?	4	1	?	?	1	1
i_3	2	2	3	4	4	?	4
i_4	2	0	4	?	?	?	5
	↓	↓	↓	↓	↓	↓	↓
$\bar{u}_j$	3.25	2.75	2.5	1.33	2.5	1.5	3.33

Figure 4: Utility Matrix

3.2.2 Items Profiles

In content-based systems, an item profile is a record including important information of that item. In simple cases, this profile illustrates some characteristics, those are easily discovered. In movie recommender system, item profile may contains:

- The set of actors in the movie. Some viewers prefer movies with their favorite actors.
- The director. Some viewers have a preference for the work of certain directors.
- The year in which the movie was made. Some viewers prefer old movies; others watch only the latest releases.
- The genre or general type of movie. Some viewers like only comedies, others dramas or romances.

As computers can only process numeric data, the generation of these features is a very crucial step to make sure that high-quality recommendations can be produced. In our project, we utilize the binary value of genres for each movie, where 1 represents the presence of a genre in a movie and 0 represents the absence. Finally, we construct the feature vector with TF-IDF (Term Frequency-Inverse Document Frequency).

3.2.3 Loss Function

We construct a linear model following the Ridge Regression model. Assume that, we have N users, M items, and Utility matrix Y. Furthermore, R is rated-or-not matrix, namely r_{mn} equals 1 if item m is rated by user n, and 0 otherwise.

Linear model:
Assume that, we find a model for each user, illustrated by weight w_n and bias b_n so that the interest of nth user for the m^{th} item is calculated by a linear function:

$$y_{mn} = x_m w_n + b_n \tag{1}$$

where x_m is a row vector representing a feature vector, w_n is a column vector.
The formula of the loss function for n^{th} user is:

$$L_n = \frac{1}{2s_n} \sum_{m:r_{mn}=1} (x_m w_n + b_n - y_{mn})^2 + \frac{\lambda}{2s_n} \|W_n\|_2^2 \tag{2}$$

where s_n is the number of items rated by n^{th} user.
As loss function only depends on rated items. Therefore, we extract all variables used according to rated items. The formula of the loss function is rewritten as:

$$L_n = \frac{1}{2s_n} ||\hat{X}_n w_n + b_n e_n - \hat{y}_n||_2^2 + \frac{\lambda}{2s_n} ||w_n||_2^2 \tag{3}$$

4 Collaborative Filtering

A collaborative filtering (CF) system provides recommendations for items based on user and movie similarity metrics. The algorithm suggests products that comparable types of people have a preference for. The benefits of collaborative filtering are numerous:

- It is content-independent, relying solely on connections.
- Since users provide explicit ratings in CF, true quality assessment of things is performed.
- It offers serendipitous suggestions since it bases them on user's or movie's similarity rather than movie content's similarity.

4.1 Neighborhood-based Collaborative Filtering

In neighborhood-based approaches, a selection of users are chosen based on how closely they resemble the active user, and predictions for this person are generated using a weighted combination of their ratings. The algorithm outlined in the following stages may be used to generalize the majority of these techniques:

1. Give each user a weight based on how similar they are to the active user.
2. Choose the k users—often referred to as the neighborhood—who are most comparable to the current user.
3. Create a forecast using a weighted average of the ratings of the chosen neighbors.

Collaborative Filtering Implementation

- **Similarity metrics**

  1. In the first stage, determining the similarity between two users is the task that has to be completed first and foremost. The Utility matrix (which contains all user ratings) is the only piece of information we have, thus it is necessary to compare the two users' columns in this matrix to see how comparable they are.

     The weight $sim(u_i, u_j)$ is utilized as a measure of how comparable the user i and the active user j. The Pearson correlation coefficient between the evaluations of the two users is the most often used metric of similarity and is defined as follows:

     $$sim(u_i, u_j) = \frac{\sum_{m \in M}(r_{i,m} - \overline{r}_i)(r_{j,m} - \overline{r}_j)}{\sqrt{\sum_{m \in M}(r_{i,m} - \overline{r}_i)^2 (r_{j,m} - \overline{r}_j)^2}} \tag{4}$$

     Where M represents the group of movies that both users reviewed $r_{i,m}$ represents the rating that user i gave to movie m, and $\overline{r}_i$ represents the mean rating that user i provided.

  2. In the next step, predictions are generally calculated as the weighted mean of variances from the neighbor's mean, as in:

     $$p_{i,m} = \overline{r}_i + \frac{\sum_{u \in K}(r_{i,m} - \overline{r}_i) \times sim(u_i, u_j)}{\sum_{u \in K} |sum(u_i, u_j)|} \tag{5}$$

     where $sim(u_i, u_j)$ is the degree of similarity between users i and j, $p_{i,m}$ is the prediction for the active user i for item m, K is the group or neighborhood of users who are most similar, and so on.

- **User-user vs item-item neighborhood**
  Due to the computational difficulty of the search for comparable users, traditional neighborhood-based CF algorithms do not scale effectively when applied to millions of people and goods. Item-to-item Collaborative was suggested as an alternative. Filtering where they match a user's rated things to comparable items rather than matching similar persons. In fact, this strategy speeds up online systems and frequently yields better suggestions.

4.2 Matrix Factorization Collaborative Filtering

4.2.1 Introduction to MF Collaborative Filtering

Matrix factorization algorithms work by decomposing the user-item interaction matrix into the product of two lower-dimensionality rectangular matrices which are the item feature

matrix (MxK) and the user feature matrix (KxN). With M is the number of items, N is the number of users and K is the number of features.

- **Latent Feature**

  In the real life, the preferences of each users has the connections to the preferences of other users through features. Similarly, some movies have identical feature with other movies. But the problem is we just indentify the limited feature and alot of other connections we don't actually know. The matrix factorization help us to deal with it very well.

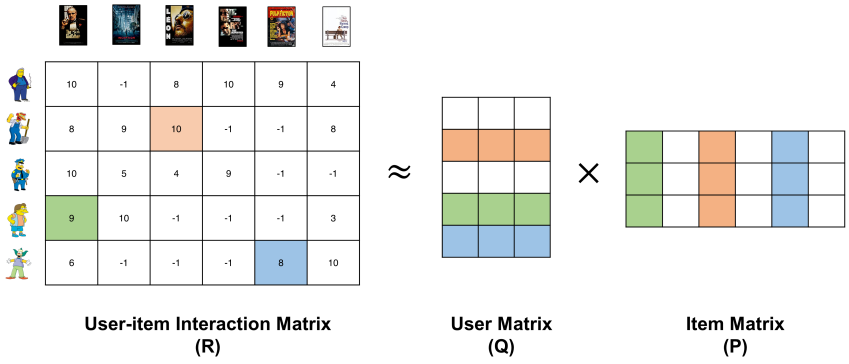


Figure 5: Matrix Factorization

The main idea behind the matrix factorization is representing users and items in lower dimensional latent space. That means we expressed the original matrix into 2 lower dimensional matrix is Users matrix which will tell us the connections between users and Items matrix which tell us the connections between movies through K latent features. We call it the latent features since we don't actually know what exactly K features is, we only know that those will help us to express the relationship between Users and between Movies. Therefore, matrix factorization will identify the connections between users and between movies based on known ratings and use it to figure out the unknown ratings, then make a recommendation to user. The high coefficient corresponds to one latent feature in both item and user matrix will also be high in the user-item matrix. That means the item has latent feature which the user likes, it will be suggested to the user.

- **Memory Reduction**

  Matrix factorization decomposes the user-item matrix into 2 lower dimensionality matrices will help the inference simpler since we only need to take the product of two vectors with length K which is much smaller than M, N. Also, storing 2 matrices: item and user requires a smaller memory than the user-item matrix. Therefore matrix factorization is

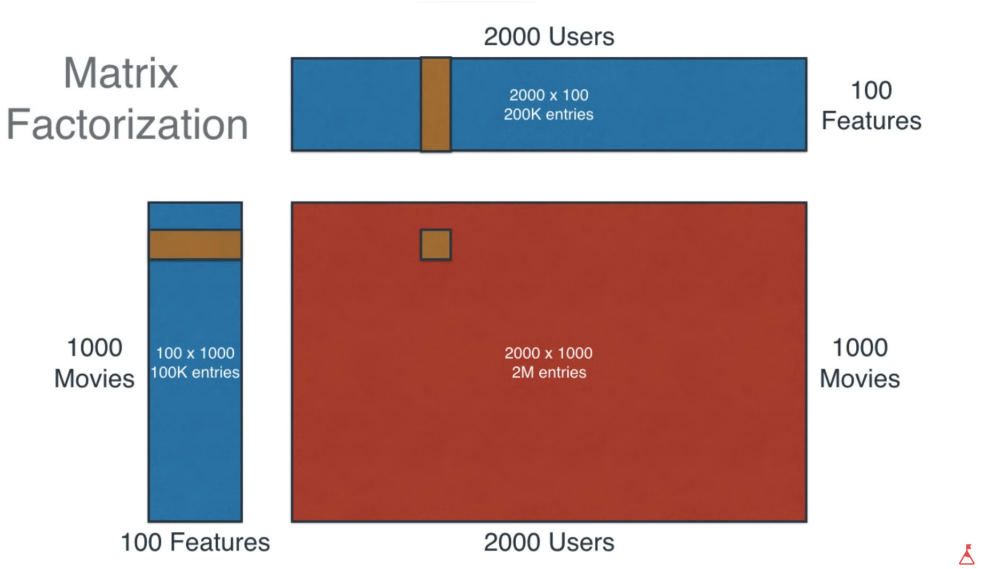


Figure 6: In this example, if we use original matrix to make recommendations, we need to store 2 millions entities. But by decomposing, we just need to store 2 matrices: One has 100K entities and one has 200K entities. So, in total, we just need to store 300K entities.

found to be the most accurate approach to large and high levels of sparsity datasets.

4.2.2 Matrix Factorization Implementation

There are many variations of the matrix factorization implementation. Because of the time limit of the project, our team decided to only chooses 2 ways to implement it:

1. Matrix factorization with Gradient Descent.
2. Matrix factorization with Truncated Singular Value Decomposition.

1. **Matrix factorization with Gradient Descent**

   In general, Gradient descent is an optimization algorithm which is commonly-used to train machine learning models. Training data helps these models learn over time, and the cost function within gradient descent specifically acts as a barometer, gauging its accuracy with each iteration of parameter updates. Until the function is close to or equal to zero, the model will continue to adjust its parameters to yield the smallest possible error.

   In particular in matrix factorization, we will initialize two random users and movies matricies. Then by each iteration, we will reduce the value of loss function by assigning new set of coefficients for user and movies matricies.

   - **Loss Function**

The loss function, we choose:

$$L(W, X) = \frac{1}{2s} \sum_{n=1}^{N} \sum_{m:r_{mn}=1} (y_{mn} - x_m w_n)^2 + \frac{\lambda}{2}(\|X\|_F^2 + \|W\|_F^2) \tag{6}$$

With X is the items matrix and W is user matrix. $r_{mn} = 1$ if movie m is rated by user n. Notation $\|\cdot\|_F^2$ is Frobineous norm is the square roots of the sum of the squares of all the entities in the matrix and s is the numbers of known ratings.

The simultaneous optimization of 2 matrices X and W is quite complicated.Therefore, the alternative method is in succession, we optimize one matrix while fixing another until converge.

- **Loss Function Optimization**
  When we fix matrix X:

$$L(W) = \frac{1}{2s} \sum_{n=1}^{N} \sum_{m:r_{mn}=1} (y_{mn} - x_m w_n)^2 + \frac{\lambda}{2}\|W\|_F^2 \tag{7}$$

When we fix matrix W:

$$L(X) = \frac{1}{2s} \sum_{n=1}^{N} \sum_{m:r_{mn}=1} (y_{mn} - x_m w_n)^2 + \frac{\lambda}{2}\|X\|_F^2 \tag{8}$$

We will optimize 2 above loss functions by using gradient descent

The updated formula for each column of W:

$$w_n = w_n - \alpha(-\frac{1}{s}\hat{X}^T(\hat{y}_n - \hat{X}_n w_n) + \lambda w_n) \tag{9}$$

Similarly, the updated formula for each column of X:

$$x_n = x_n - \alpha(-\frac{1}{s}(\hat{y}_n - x_n\hat{W}_n)\hat{W}_n + \lambda x_n) \tag{10}$$

2. **Matrix factorization with Truncated SVD**

- **Singular Value Decomposition (SVD)**
  The singular value decomposition of a matrix A is the factorization of A into the product of three matrices:

$$A = U\Sigma V^T$$

where the columns of U and V are orthonormal and the matrix Σ is diagonal with

positive real entries whose i^{th} diagonal entry equals the i^{th} singular value σ_i for i = 1, . . . , r and all other entries of Σ are zero.

- **Truncated SVD**

  Truncated SVD is the lower-rank approximation. That means we use low-rank matrix to approximate the original matrix. In the diagonal matrix Σ the singular values σ_i in the diagonal is non-negative and decreasing: $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq ... \geq \sigma_r \geq 0$, show how important the various columns of U and rows of V are. Therefore, U_1 is more important than U_2 and U_2 is more important than $U_3, ...$ Similar to V_1, $V_2, ...$

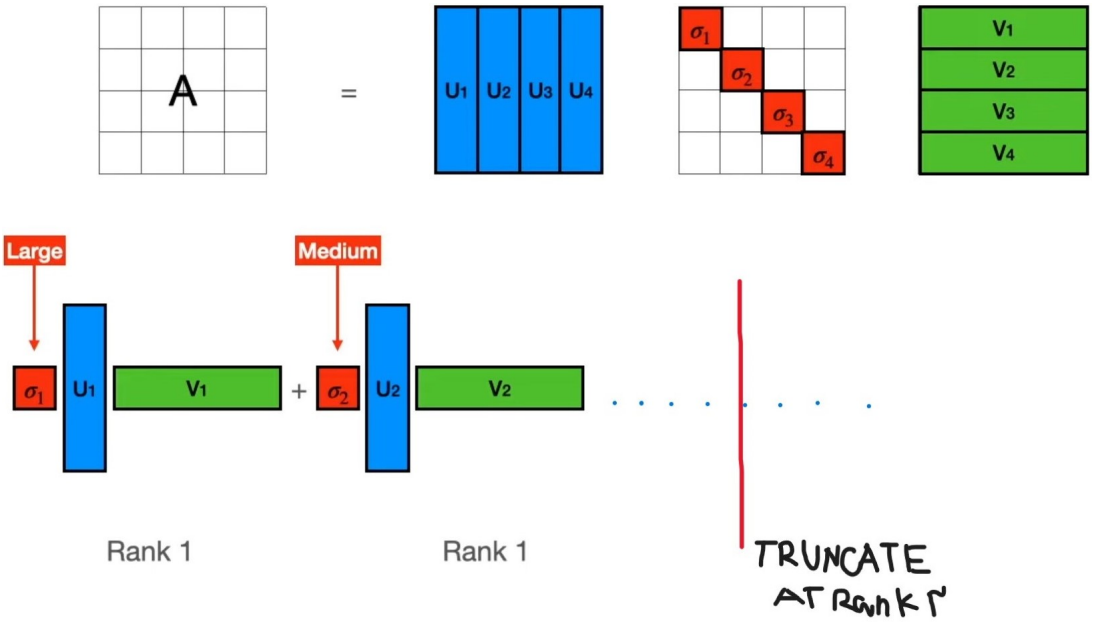


Figure 7: Truncated SVD

The word "important" here express which amount of values those columns or rows contribute to the original matrix. To be clearly, we can see the picture above. We can express the original matrix as the sum of many rank-1 matrices and the more to the right side, value of sigma is decreasing so it contribute value less to the original matrix.And maybe some last matrices, value of $\sigma \approx 0$. Now, truncated is applied, truncated is the action that we only keep the most important $r(r < rank(A))$ matrices or r matrices are left to right and all matrices after r we discard.So, from that, by using lower-rank matrix, we almost still have important information about the original matrix.

- **Matrix factorization with Truncated SVD**

  From the original ratings matrix, we use the truncated SVD to generate the low-rank r matrix which still have almost information, then use it make the recommendation.

5 Experiment and result

5.1 Evaluation Metrics

We employ the RMSE and MAE assessment metrics to assess the correctness of our models. To assess the model's suggestions, we also evaluated other accuracy measures including Precision, Recall, and F1, and ranking accuracy metrics including NDCG, and MAP. To determine the suggested and pertinent items utilized in the computation of the Precision and Recall metrics, we set a threshold equal to the average ratings of each user. This threshold can manage the bias issue that arises when several users score their favorite films as 5, but not-so-loved films receive ratings of 3, and vice versa. Recommended items are those items with predicted rating greater than or equal to the threshold while relevant items are those with actual rating greater than or equal to the threshold. The recommedations are sorted by the ratings. In order to acquire the average result, we ran our models 30 times.

5.2 Content-based model

In this project, we model Content based Recommendation as a Ridge Regression problem. We predict users' ratings based on items' feature vectors. We modified the hyperparameter lambda and derive the Graph below:

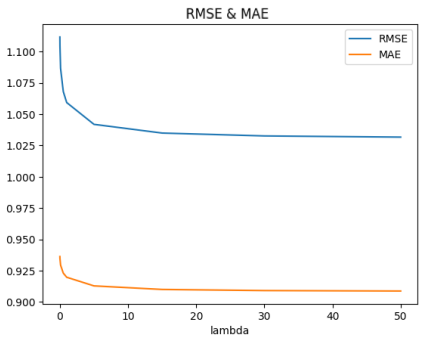


Figure 8: RMSE and MAE depends on λ

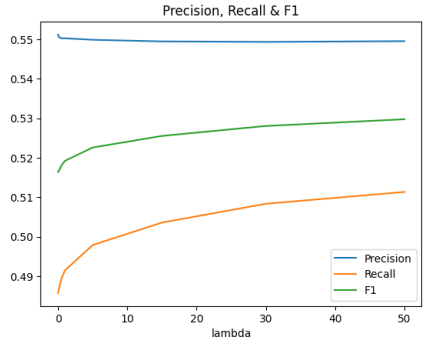


Figure 9: Precision, Recall, and F1 depends on λ

We choose $\lambda = 10$ as the best option based on the findings in the graph. We examine our model for $\lambda = 10$ to determine how many movies we should suggest to the user in order to get the optimal result.

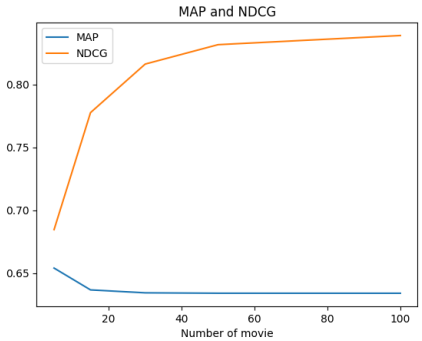


Figure 10: MAP and NDCG depends on number of movie

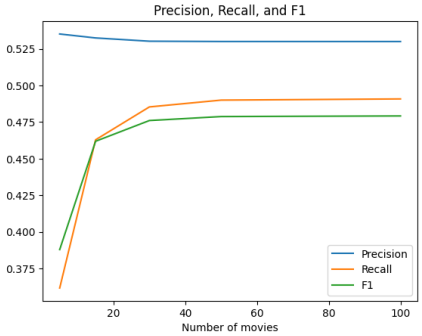


Figure 11: Precision, Recall, and F1 depends on number of movie

In light of the outcome shown in this graph, we determined that 30 movies was the optimum solution.

5.3 Collaborative Filtering model

We construct Collaborative Filtering model based on Neighborhood-based model. In this model, the hyperparameter is the K most similar neighbors used to predict unknown ratings. Hence, we adjust the number of neighbors to get the ideal value for this hyperparameter. Furthermore, we examine our model to determine the how many movies we should recommend for user to achieve the highest accuracy.

5.3.1 Item-item model

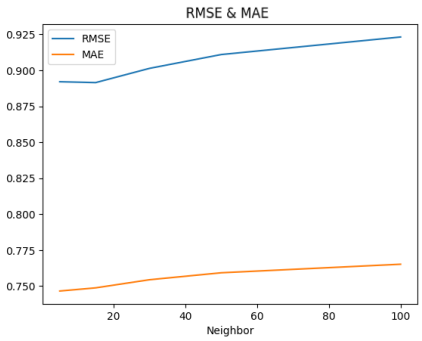


Figure 12: RMSE and MAE depends on number of neighbor

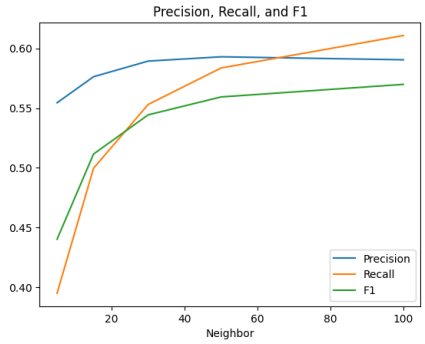


Figure 13: Precision, Recall, and F1 depends on number of neighbor

Based on the results in the graph, we decide that $neighbor = 20$ is the best option. In order to obtain the best outcome, we analyze our model for $neighbor = 20$ to decide how many movies we should recommend to the user.

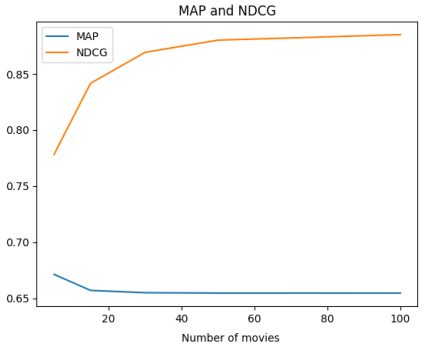


Figure 14: MAP and NDCG depends on number of movie

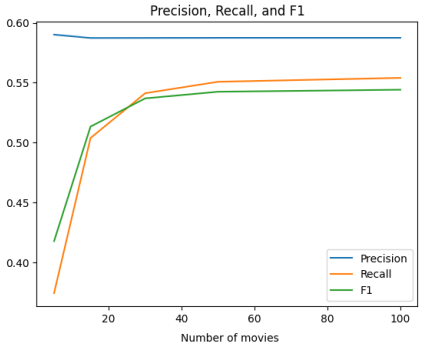


Figure 15: Precision, Recall, and F1 depends on number of movie

The result of this graph led us to the conclusion that at 30 films, ranking metrics remain stable. Hence, 30 is the best option for number of movie recommended.

5.3.2 User-user model

We determine that $neighbor = 50$ is the best choice based on the results shown in the graph. In order to select how many movies to suggest to the user, we assess our model for $neighbor = 50$ in order to get the best

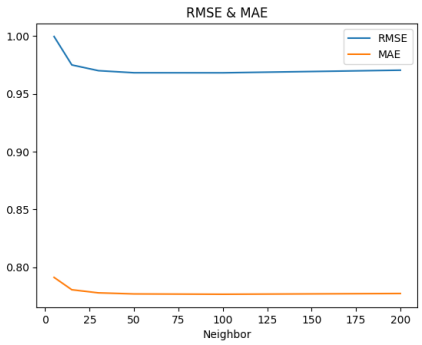


Figure 16: RMSE and MAE depends on number of neighbor

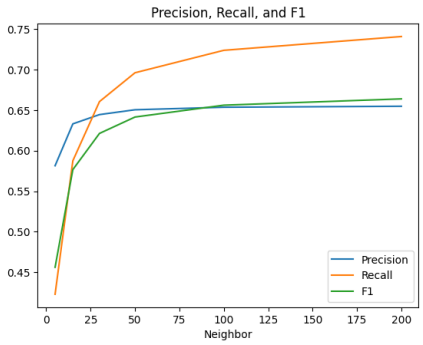


Figure 17: Precision, Recall, and F1 depends on number of neighbor

results.

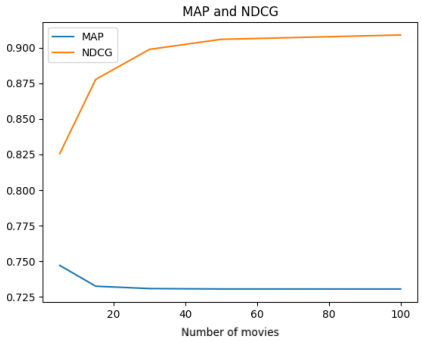


Figure 18: MAP and NDCG depends on number of movie

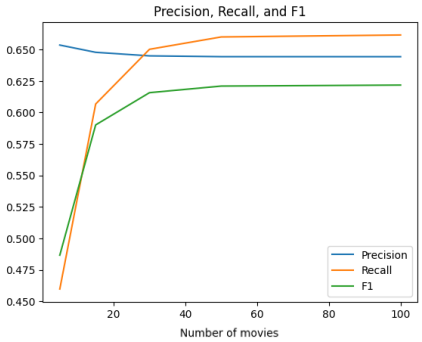


Figure 19: Precision, Recall, and F1 depends on number of movie

We deduced from the graph's outcome that ranking measures are stable after 30 films. Therefore, the optimal choice for the number of recommended movies is 30.

5.4 Matrix Factorization model

5.4.1 MF using Gradient Descent

We fix the size of latent features K = 10, the weight for regularization component $\lambda = 0.1$. Then we tune the number of iterations in Gradient Descent. We concluded from the graph's outcome that RMSE and MAE remain stable after 30 iterations. We evaluate our model with 30 iterations in order to achieve the greatest results when determining how many movies to recommend to the user.

Based on the results provided in this graph, we discovered that 30 movies is the ideal selection.

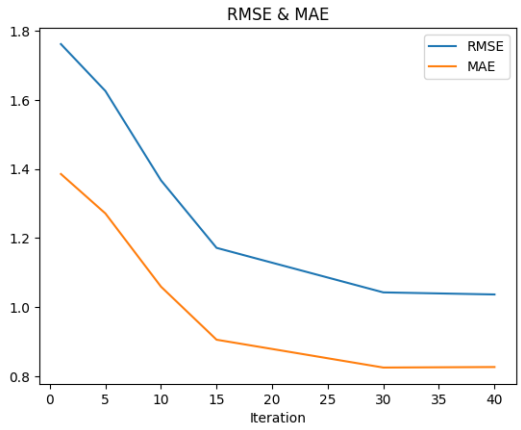


Figure 20: RMSE and MAE depends on number of iteration

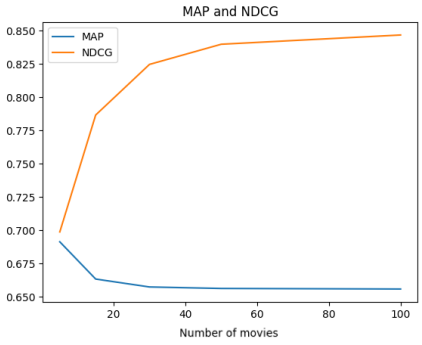


Figure 21: MAP and NDCG depends on number of movie

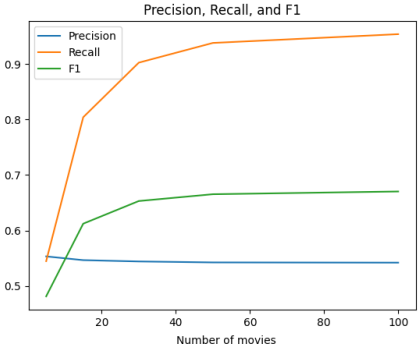


Figure 22: Precision, Recall, and F1 depends on number of movie

5.4.2 MF using Singular Value Decomposition

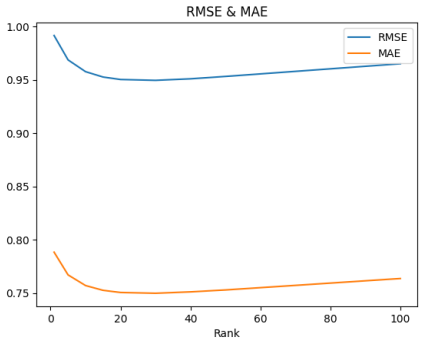


Figure 23: RMSE and MAE depends on matrix rank

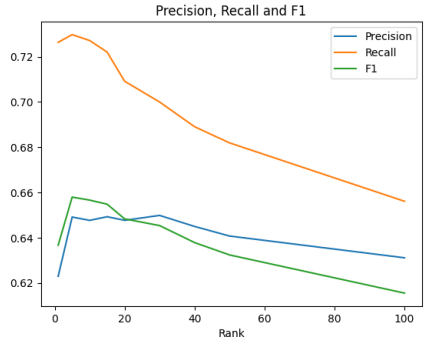


Figure 24: Precision, Recall, and F1 depends on matrix rank

Based on the findings in the graph, we determine that $rank = 10$ is the most suitable choice. To get the optimal results, we examine our model with $rank = 10$ to determine the number of films we need to recommend to the user.

As a consequence of this graph, we concluded that at 30 movies, ranking metrics remain steady. As a

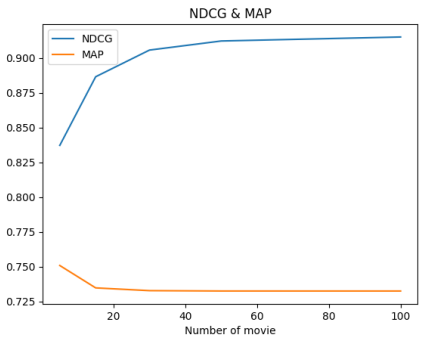


Figure 25: MAP and NDCG depends on number of movie

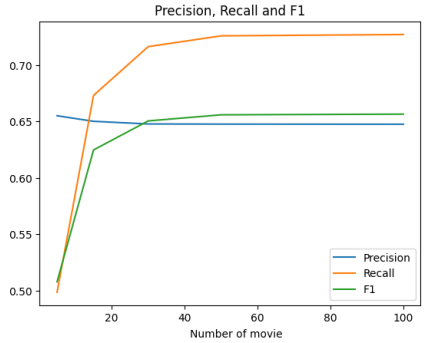


Figure 26: Precision, Recall, and F1 depends on number of movie

result, 30 is the optimum option for the number of movies advised.

5.5 Summary

After experiments, we discover that the most suitable number of recommendations is 30.

In conclusion, the results of all models tuned with optimum hyperparameters are shown in the Table below:

K=30	RMSE	MAE	Precison	Recall	F1	MAP	NDCG
Content-based	1.03	0.91	0.63	0.59	0.58	0.73	0.77
User-User	0.97	0.78	0.65	0.66	0.62	0.73	0.90
Item-Item	0.91	0.75	0.64	0.60	0.58	0.70	0.87
MFGD	1.03	0.81	0.54	0.91	0.66	0.65	0.82
MFSVD	0.95	0.75	0.65	0.73	0.65	0.73	0.92

Based on the table, we can infer that the Item-item model delivers the most accurate results, but the efficiency of suggestions is low. Meanwhile, Matrix Factorization using SVD offers lower accuracy but greater efficiency. In general, we conclude that Matrix Factorization using SVD approach provides the most efficient recommendations. Moreover, we can see that the order of recommendations is quite good.

6 Conclusion and Future work

The Content-based recommenders lacks any personalisation for the users. Collaborative-filtering recommenders is more complicated and perform better. User-user CF has some limitations when the number of users is large. In those cases, Item-item is often used and gives better results. Matrix Factorization Collaborative Filtering using SVD recommenders give the best performance. Throughout this project, we were able to have a better insight on various kinds of recommender systems and underlying algorithms, especially SVD.

In future, we can work on others approaches like Nonnegative Matrix Factorization, Fast Incremental Matrix Factorization, Neural Collaborative Filtering, etc to obtain better performance. Our approach can be further extended to other domains to recommend other product.

7 Reference

1. F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872
2. Kalervo Järvelin, Jaana Kekäläinen: Cumulated gain-based evaluation of IR techniques. ACM Transactions on Information Systems 20(4), 422–446 (2002)
3. Yining Wang, Liwei Wang, Yuanzhi Li, Di He, Wei Chen, Tie-Yan Liu. 2013. A Theoretical Analysis of Normalized Discounted Cumulative Gain (NDCG) Ranking Measures. In Proceedings of the 26th Annual Conference on Learning Theory (COLT 2013).
4. SPSS Tutorials: Pearson Correlation https://libguides.library.kent.edu/SPSS/PearsonCorr
5. Mean Average Precision at K (MAP@K) clearly explained: https://towardsdatascience.com/mean-average-precision-at-k-map-k-clearly-explained-538d8e032d2
6. Recommendation-systems https://machinelearningcoban.com

8 Work distribution

1. Proposal: Lâm Anh
2. Data exploration and preprocessing: Quý
3. Do research and implement Content-based filtering: Quý
4. Do research and implement User-User based filtering: Dat
5. Do research and implement Item-Item based filtering: Giang
6. Do research and implement Matrix Factorization: Lâm Anh
7. Do research on evaluation metrics: Ánh
8. Implement evaluation metrics: Dat, Lâm Anh, Giang
9. Report:
   (a) Introduction: Lâm Anh
   (b) Dataset: Dat
   (c) Content-based filtering: Quý
   (d) Neighborhood filtering: Giang, Dat
   (e) Matrix Factorization: Lâm Anh
   (f) Experiment and Result: Dat, Lâm Anh, Giang,
   (g) Appendix: Ánh
10. Slide: Ánh

11. Presentation

    (a) Introduction to Movie Recommendation System: Ánh

    (b) Dataset: Ánh

    (c) Content-Based Filtering: Quý

    (d) Neighborhood-based Collaborative Filtering: Dat

    (e) Matrix Factorization Collaborative Filtering: Lâm Anh

    (f) Experiment and result: Giang

9 Appendix

9.1 RMSE

Root Mean Square Error (RMSE): RMSE is a common metric in prediction and recommendation problems, used to measure the average error between the actual value and the predicted value. In this instance, the RMSE is determined by comparing the actual rating to the rating predicted by the model. The following are the steps to compute RMSE in the movie recommendation problem:

$$RMSE = \sqrt{\frac{\sum_{n=1}^{N}(x_i - \hat{x}_i)^2}{N}} \tag{11}$$

where N is the number of data observed, x_i is the actual value, $\hat{x_i}$ is the predicted value.

9.2 MAE

Mean Absolute Error (MAE): MAE is a common metric in prediction and prediction problems, used to measure the average difference between the actual value and the predicted value. In this case, the MAE is calculated based on the difference between the actual rating and the rating predicted by the model. The formula for calculating the MAE in the problem of movie recommendation is as follows:

$$MAE = \frac{\sum_{n=1}^{N}|x_i - \hat{x}_i|}{N} \tag{12}$$

where N is the number of data observed, x_i is the actual value, $\hat{x_i}$ is the predicted value.

9.3 Precision,Recall and F1

Precision helps evaluate the movie recommendation model's ability to recommend movies that are quality and tailored to the user's personal preferences. Precision focuses on ensuring that the movies recommended to the user are precise and relevant.

The formula for calculating Precision is:

$$Precision = \frac{TP}{TP + FP} \tag{13}$$

Recall: Recall is a crucial measurement for assessing how well a model can identify all the appropriate movies for the user. Recall concentrates on finding as many appropriate movies as possible for users, avoiding missing

important movies. To calculate the Recall, we need to know the following values: True Positive (TP): Number of movies suggested by the model and actually relevant to the user. False Negative (FN): Number of movies that match the user but are not suggested by the model.

The formula for calculating Recall is:

$$Recall = \frac{TP}{TP + FN} \tag{14}$$

Where:

1. True Positive (TP): Number of movies suggested by the model and actually relevant to the user.
2. False Positive (FP): Number of movies suggested by the model but not suitable for the user.
3. False Negative (FN): Number of movies that match the user but are not suggested by the model.

F1-score: F1-score combines Precision and Recall into a single statistic to assess how well model accuracy and coverage are balanced. The F1-score aids in evaluating the movie recommendation model's overall effectiveness, ensuring that the model achieves high accuracy and coverage.

The formula for calculating F1-score is:

$$F1_score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{15}$$

However, the use of F1-score should note that it is a composite metric and only gives an overview of performance. We need to consider the need to combine Precision and Recall individually to get a more detailed view of the model's performance.

9.4 Normalized Discounted Cumulative Gain (NDCG)

The NDCG evaluates the quality and priority of the movie recommendations given by the model. NDCG measures the priority of suggested movies by considering the distribution and position of movies in the suggested list relative to the actual rating of the user.

The NDCG calculation formula will include the following steps: Calculate Discounted Cumulative Gain (DCG): The premise of DCG is that highly relevant documents appearing lower in a search result list should be penalized as the graded relevance value is reduced logarithmically proportional to the position of the result.

The traditional formula of DCG accumulated at a particular rank position r is defined as:

$$DCG_r = \sum_{i=1}^{r} \frac{rel_i}{log_2(i+1)} \tag{16}$$

where rel_i is the actual rating of the movie at position i in the recommended list.

To stronger emphasis on retrieving relevant documents, we utilize an alternative formulation of DCG:

$$DCG_r = \sum_{i=1}^{r} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{17}$$

Calculate Ideal Discounted Cumulative Gain (IDCG): IDCG is the ideal DCG value calculated from a list of actual ratings sorted in descending order of rating. The IDCG is the maximum DCG value that can be obtained for an evaluation list.

Calculate Normalized Discounted Cumulative Gain (NDCG): To make DCGs directly comparable between users, we need to normalize them. We divide the raw DCG by this ideal DCG to get NDCG, a number between 0 and 1.

$$NDCG = \frac{DCG}{IDCG} \tag{18}$$

9.5 Mean Average Precision (MAP)

Precision@K (P@K): Normal Precision metric does not consider the rank of the recommendations. However, Precision@K calculate the precision for the top K recommended movies:

$$P@K = \frac{\#\text{Relevant items in top K recommendations}}{\#\text{Items in top K recommendations}} \tag{19}$$

Average Precision@K (AP@K): The Average Precision@K is the sum of Precision@K where the item at the k_{th} rank is relevant divided by the total number of relevant items (r) in the top K recommendations

$$AP@K = \frac{1}{r}\sum_{k=1}^{K} P@K \cdot rel(k) \tag{20}$$

$$rel(i) = \begin{cases} 1, & \text{if items at } k_{th} \text{ rank is relevant} \\ 0, & \text{otherwise} \end{cases} \tag{21}$$

Average Precision@K is higher if the most relevant recommendations are on the first ranks. Hence, AP@K shows the goodness of the top recommendations.

Mean Average Precision@K (MAP@K): MAP@K calculate the mean value of AP@K for all users.

$$MAP@K = \frac{1}{M}\sum_{i=1}^{M} AP@K_i \tag{22}$$

where M is the number of users

MAP@K not only provides insights if your recommendations are relevant but also considers the rank of your correct predictions.