Comparative Analysis of Movie Recommendation Systems

Abhavya Chandra 16110001
Kshitij Gajapure 16110055
Shubham Deshpande 16110050

Abstract

In this project, we compare different approaches of the recommendation system. We have done a comparative study on an ensemble of MovieLens dataset and TMDB dataset. Based on all our observations we conclude that CUR decomposition performs best among other recommendation systems.

1 Introduction

In the last decade, as internet service becomes more and more accessible and affordable we are seeing a transition of users from TV, cinema halls to OTT platforms, from music players to subscription-based music applications, from offline shopping to online shopping, from live matches to live streaming, etc. This digital transformation growth is further fuelled by the Covid-19 pandemic and work from home, physical distancing restrictions.

As a result of such market growth, a number of online service providers are getting popular and distributing large amounts of content.

Fortunately, this online consumption has also generated large amounts of user data such as user ratings, search, view, buy history, user interests, etc. By taking the right insights from this data we can recommend personalized content/products to users. Hence the profitability of such online enterprises now heavily depends on the effectiveness of their recommendation systems.

2 Dataset

In our project, we are exploring the Content-based and Collaborative filtering-based recommendation systems, we need the datasets that have information about the individual movies as well as the information about the users and their ratings for movies. Both of these requirements were individually satisfied by the TMDB which contains the information about the movieId, title, voteAverage, voteCount, popularity, keywords, cast, crew, budget, genre, overview, revenue, tagline, runtime, etc. and the MovieLens Dataset which contains the information about the userId, movieId, tag, title, genre, and rating. But for simplicity, we wanted to keep a common dataset for our different approaches, hence we decided to use The Movies Dataset, which is an ensemble of the TMDB and the MovieLens Dataset. The Movies Dataset contains 26 million ratings from 270,000 users for 45,000 movies.

3 Approaches

Broadly speaking, recommendation systems are of two types- Content-based and Collaborative filtering. In this project, we have implemented different approaches and tried to compare them based on different metrics like MSE, Recall, Precision, and F score. Below are different approaches that we have implemented.

3.1 Simple Model

3.1.1 Experimental setup

We are using the data provided in the movies_metadata.csv. It had 24 different columns containing information about the id, genres, popularity, overview, vote_count, vote_average, belongs_to_collection, etc. but since we only need information about the id, title, popularity, vote_average, and vote_count, we've dropped the remaining columns.
The dataset contains ratings corresponding to 45460 movies.

3.1.2 Simple Weighted

Simple weighted is a recommendation system based on the weighted ratings formula that was used by IMDB in the mid-2000s. This is a very simple recommendation method that does not take into account the users as well as any relations between the movies. In this method, new movies are recommended based on weighted ratings assigned to the movies already rated by users. Each movie is assigned a new weighted rating. The weighting is done such that movies that have more number of votes have higher weights. The weighting also takes care that if a movie has been voted by more number of voters then the ratings are more trusted than movies that have a high average rating but being rated by fewer people. The weighted average is calculated by using the following formula:

$$W = \frac{R * v + C * m}{v + m}$$

where,
W = weighted rating
R = average rating of the movie
v = number of votes for the movie
m = minimum votes required to be listed in the top 75%
C = mean vote for the whole database

3.1.3 Weighted average + popularity

In the weighted average method, it is possible that there are some movies that are popular but are not rated by many users. This would lead to assigning poor weighted ratings to those movies even though they are popular. This can be handled by taking the popularity of the movie into account. Which is precisely what we are doing in this method. Here we are 50% weightage to popularity and 50% to the weighted average. But even after taking the popularity into account, we still are just ranking movies without considering their relations with other movies and user's rating history. In order to build a recommendation system that takes the user's history, information about movies, as well as the interactions between movies and users into account, the Content-based and Collaborative filtering-based recommendation systems were developed, which we have mentioned in the following sections.

3.2 Content based

This recommendation system is based on the properties of items. Using these properties we find similarities between different items and assign some score to the similarity. Using these similarity scores we then give recommendations to users based on their previous interests and ratings.

3.2.1 Experimental setup

For characteristics of movies we are using 'Overview' data from movies_metadata.csv. It has information about 45466 movies. For user ratings, we are using user-movie ratings data provided in ratings_small.csv. It has 100005 ratings for 9066 movies by 671 users. We split the dataset into train and test sets in an 8:2 ratio. Selection of a (userId, movieId) tuple for a set is done at random.

3.2.2 Baseline

We first calculated the average rating given by a user to all rated movies. This average rating will

then be assigned to the new movie for prediction.

3.2.3 Model

In this, we have calculated two profiles 1. Item profiles and 2. User profiles

3.2.3.1 Item profiles

We construct this profile for each item in the dataset. This profile is built using some of the specific characteristics of the item. For the movie recommendation system, these characteristics can be a set of actors, directors, genre, location, year, movie industry, etc. In our model, we have used "Overview" data to extract these characteristics. In this, we have used TF.IDF scores to find the similarity between different items. Where TF-IDF is given by
TF: Term Frequency

$$\text{TF(t)} = \frac{Frequency\ occurrence\ of\ term\ t\ \ in\ document}{Total\ number\ of\ terms\ in\ document}$$

IDF: Inverse Document Frequency

$$\text{IDF(t)} = \log_{10}\left(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ containing\ term\ t}\right)$$

$$\text{TF-IDF score : } w_{ti} = TF_{t,j} \times IDF_t$$

We have calculated this TF-IDF vector for every movie in the movies_metadata table.
After this, we have calculated the similarity between these vectors using 'Cosine Similarity'. Higher the cosine similarity between movies, the more similar they are. This score ranges between 0 and 1 while 1 being the cosine similarity score of the movie with itself.

3.2.3.2 User profiles

For this, we have used movies rated by users in the past. Then we have normalized these ratings. After this, we have used item profiles of the movies rated by the user in the past. Using these item profiles and normalized rating we have predicted the ratings for that user for all movies. We have recommended top movies based on this predicted rating to that user.

3.2.4 Disadvantages of content recommendation systems:

1. Finding appropriate characteristics of an item is difficult and subjective.
2. It also causes overspecialization as it neer recommend item outside user's profile
3. Difficult to predict items for new users as past ratings are not available.

3.3 Collaborative filtering

Collaborative filtering based recommendation systems are based on the idea that similar people will share similar kinds of taste. If user A and user B have similar preferences and movie Q is watched by user A but not by B. Then user B would also enjoy watching movie Q. Collaborative filtering approaches capture users' taste which was missing in content based approaches.

Broadly collaborative filtering approaches can be divided into three types: User-User based filtering, Item-Item based filtering, and User-Item based filtering.

3.3.1 Experiment setup

We are using user-movie ratings data provided in ratings_small.csv. It has 100005 ratings for 9066 movies by 671 users. We split the dataset into train and test sets in an 8:2 ratio. Selection of a (userId, movieId) tuple for a set is done at random.
Then we create the user-movie rating matrix using all the tuples ((userId, MovieId, ratings)) in train and test sets. For all the tuples in the test set, we assign a rating of 0. We finally obtained the user-movie rating matrix of size 671x9066.

Our aim is to predict the ratings for test set tuples and compare the performance of different

approaches.

3.3.2 Baseline

The rating for a new movie is the average of the ratings for a movie by all users who rated that movie. The baseline model fails to give predictions for movies that have not already been watched/ rated by any of the users.

3.3.3 User-User based

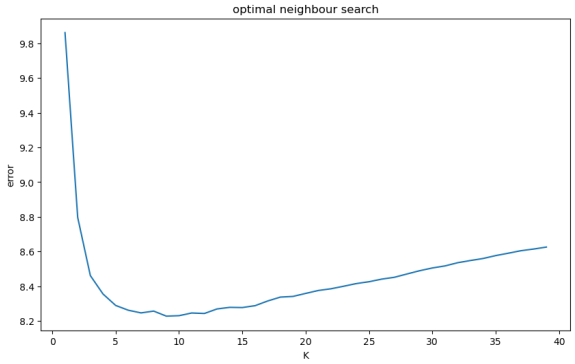
In this approach k similar users are picked to give movie recommendations to the desired user. Similarity is computed between users in the user-movie rating matrix. We have used Pearson correlation as a similarity measure here. Estimate for a movie rating for user x is given by,

$$r_{xi} = \frac{\sum_{y \in N} s_{xy} r_{yi}}{\sum_{y \in N} s_{xy}}$$

$$s_{xy} = correlation(x, y)$$

where,
s_{xy}: similarity of user x and y
r_x: user x's rating vector
N: set of k nearest neighbours of user x.

We can find the optimal k, i.e number of nearest neighbours by exhaustive search method. **K=9** came out to be optimal k values for us.



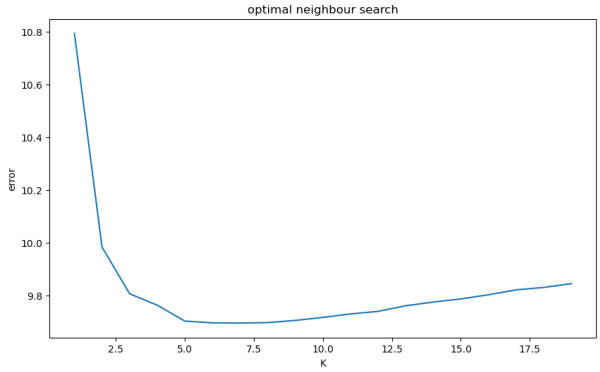
3.3.4 Item-Item based

Here recommendation is given based on similarity between items for a user. Similarity is computed between items in the user-movie rating matrix. Each item is a vector based on the rating received by all the users. We have used Pearson correlation as a similarity measure here. Estimate for a movie rating for user x is given by,

$$r_{xi} = \frac{\sum_{y \in N(i;x)} s_{ij} r_{xj}}{\sum_{y \in N} s_{ij}}$$

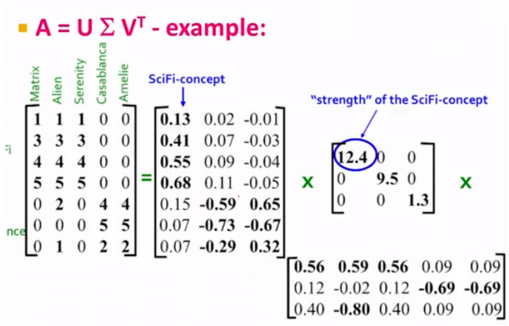
where,
s_{ij}: similarity of movie i and movie j
r_{xj}: rating of user x on item j
N: set of k nearest neighbours of movie i.

We can find the optimal k, i.e number of nearest neighbours by exhaustive search method. **K=6** came out to be optimal k values for us.

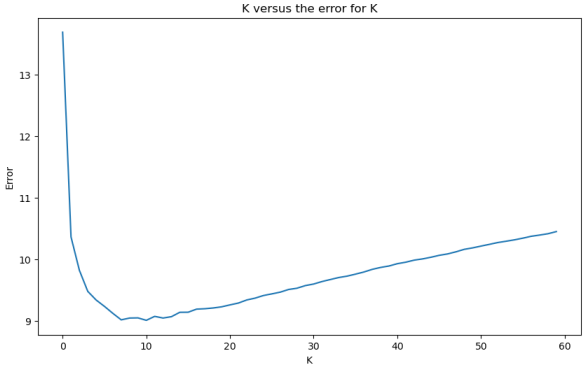


3.3.5 User-Item based (SVD)

With low-rank matrix factorization, we can find the latent features in the data from observed features. SVD is a low-rank matrix factorization technique. Through SVD the user-movie rating matrix is projected into another dimension and then the top k relevant features out of all the latent features are picked to form the decomposed new user-movie rating matrix. For eg. The latent feature could be the genre of a movie.



We used exhaustive search for optimal selection of k (number of latent features). **K=8** came out to be the optimal value of k for us.



3.3.6 User-Item (CUR)

In this section, instead of using SVD to decompose the matrix, we will be using CUR

decomposition. Given a *m* x *n* matrix M and r, C is a *m* x *r* matrix formed from choosing r columns from M and R is a *r* x *n* matrix formed from choosing r rows from M. For selecting rows of R, we choose a row *i* of M with probability p_i, which is equal to the sum of squares of all elements of row *i* divided by the square of the Frobenius norm of M.

$$p_i = \frac{\sum_j m_{ij}^2}{\sum_{i,j} m_{ij}^2}$$

where,
p_i = probability of selecting row *i*

After selecting the row, the selected row *i* on M is scaled by dividing it by $\sqrt{rp_i}$.

For selecting columns of C, we choose a column *j* of M with probability q_j, where q_j is the sum of squares of all elements of column *j* divided by the square of the Frobenius norm of M.

$$q_j = \frac{\sum_i m_{ij}^2}{\sum_{i,j} m_{ij}^2}$$

where,
q_j = probability of selecting column *j*

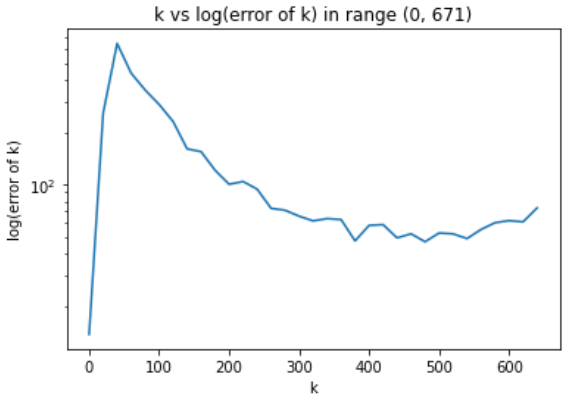
After selecting the column, the selected column *j* of M is scaled by dividing it by $\sqrt{rq_j}$.

As we are randomly selecting the rows and columns of M, it is possible that we may select the same row or column multiple times. To get rid of the duplicate rows or columns, we count the number of duplicates for each row or column, combine those rows and columns into single rows or columns, and then we multiply all of the values of that particular row or column by $\sqrt{k}$, where k is the number of duplicates of that particular row or column.

For calculating the *r* x *r* matrix U, we need to first calculate the matrix W of the same size. The matrix W of size *r* x *r* is formed by taking the intersection of the chosen columns of C and chosen rows of R. For calculating the matrix W, we first compute SVD of W, let that be $X\Sigma Y^T$. Then we take the Moore-Penrose pseudoinverse of Σ, which is denoted by Σ^+. Then we compute $U = Y (\Sigma^+)^2 X^T$

In our implementation of CUR decomposition, in order to avoid taking sub-optimal values because of the randomness of the selection of C and R, we've added an extra loop while calculating the CUR decomposition. At the end of the loop, we only take the decomposition with minimum MSE.

For finding the optimal k, we've calculated the MSE at various values of k, and it was found that **k = 480** is the optimal value of k for our input matrix.



4 Evaluation

Most common evaluation metrics to evaluate Recommendation systems are MSE and Recall @K, Precision @K, F score and time for prediction. We evaluate different recommendation system approaches on the same.

4.1 Mean Square Error

$$MSE = \frac{y_{hat} - y}{length\ of\ y_{hat}}$$

where,

y_{hat}: estimated rating

y: actual rating

Approach	MSE	No prediction for (out of 20k)
Simple Weighted	8.37	11138
Weighted average + popularity	9.95	11138
Content Based	13.24	15260
Collaborative filtering (baseline)	12.17	4209
Collaborative filtering (User-User)	8.23	-
Collaborative filtering (Item-Item)	9.69	-
Collaborative filtering (User Item - SVD)	9.04	-
Collaborative filtering (User Item - CUR)	46.79	-

4.2 Precision @k and Recall @k

In binary classification Precision and Recall helps in identifying the quality of predictions, it takes into account the values from the confusion matrix into account to evaluate a model. We can see the quality of different recommendation systems using Precision and Recall.

In the recommendation system setting, we first convert the problem into a classification problem, We set a relevance threshold, say 3. All movies above 3 will be marked as relevant and all the movies with ratings below 3 will be marked as irrelevant. Similarly, if a movie has a predicted rating greater than 3 we call that recommendation to be relevant and vica versa.

Precision helps us identify what proportion of recommended movies are relevant. While Recall is the proportion of relevant movies that are present in the recommended movies. Mathematically we can write it as -

$$Precision = \frac{\#\ of\ Recommended\ movies\ that\ Relevant}{\#\ of\ Recommended\ movies}$$

$$Recall = \frac{\#\ of\ Recommended\ movies\ that\ Relevant}{\#\ of\ Relevant\ movies}$$

For our experiments, we have chosen the relevance threshold to be 2.

In the recommendation system, we mostly make top n predictions for a user. And therefore we compute precision and recall for top n predictions/recommendations. Precision and Recall on top-k recommendation is called Precision @k and Recall @k. For our experiment, we have taken k to be 18.

If a model has high Precision then most of the recommended movies will be ground truth relevant movies. But it might not contain all the ground truth relevant movies. Also, there is a possibility that the model predicts ground truth relevant movies as irrelevant. This will lead to a low Recall.

If the model has high Recall then the recommended movies set covers almost all the ground truth relevant movies. But it might predict a lot of ground truth irrelevant movies as relevant. This will lead to low Precision.

4.3 F score

It takes into account both Recall and Precision. Using it we can make conclusions about how robust and precise our model is. F score is the harmonic progression of recall and precision. It takes into account both how many of the movies are correctly predicted as relevant and how many relevant movies are missed while prediction.

$$\text{F score} = 2 \times \frac{Precision\ x\ Recall}{Precision + Recall}$$

F score achieves its highest value when recall and precision both have the same values.

	SVD	**CUR**	**user-user**	**user-item**
time	20.5 s	8.48 s	36.8 s	50.2 s
precision	1.0	0.95	0.98	0.88
Precision @ k = 18	1.0	0.94	1.0	1.0
recall	0.12	0.25	0.09	0.2
Recall @ k = 18	0.51	1.0	0.3	0.89
f_score	0.22	0.40	0.16	0.36
f_score @ k = 18	0.67	0.97	0.46	0.94

(relevance threshold=2)

Predictions for userId = 17 :

SVD	**CUR**	**Content-Based**
Point Break	The Million Dollar Hotel	Local Color

The Million Dollar Hotel	Eight Miles High	Dead Man
Cool Hand Luke	Deep Blue Sea	Voyage to the Bottom of the Sea
Judgment Night	The Rapture	Baise-moi
Lonely Hearts	The Host	The Robbery of the Third Reich

5 Conclusion

Based on MSE we can see that SVD is the best approach. The variances of values estimated after CUR decomposition is high, this justifies the high values of MSE achieved. Simple Weighted, Weighted average + popularity have comparable MSE but we should also consider the number of test tuples for which we have no predicted rating. The approach used in these algorithms makes them prone to missing some test tuples. Along with that they also don't take the user's history, information about movies, as well as the interactions between movies and users into account, which makes the model less general and more biased.

Content based does take information about movies into account but it also lacks interaction between users.

We can see that SVD performs better if we take Precision as a metric of evaluation while CUR has a comparable Precision, performs better if we take Recall and F score into account. Better F score means that the model is both robots as well as precise.

While we can see that user-user and user-item based approaches are precise but they have low Recall which guides that they fail to recommend all the ground truth relevant movies.

For an input matrix M of size m x n, the time complexity for SVD is O(mn^2) or O(m^2n), whichever is the least. The time complexity for CUR decomposition is O(mn). This was also reflected in our results, where the time taken for SVD was 20.5 s and the time taken for CUR was 8.48 s. Hence we conclude that CUR is the best approach among all the other approaches. We choose CUR decomposition as an algorithm for our recommendation system.

For content based recommendations, we can recommend to users with unique tastes and we can recommend new and unpopular items. Also, in this, we don't need data of other users. Further, we can argue that a better recommendation system can be built by ensembling content-based and collaborative filter approaches. As it would involve combining item content as well.

Acknowledgements

This project was done as part of Introduction to Data Science course at IIT Gandhinagar in spring semester. We would like to thank our course instructor Prof. Anirban Dasgupta for constant help and guidance.

References

[1] Leskovec, J., Rajaraman, A., & Ullman, J. D. (2020). Mining of massive datasets. Cambridge, United Kingdom: Cambridge University Press.
[2] Leskovec, J., Rajaraman, A., & Ullman, J. (2016, April 12). Mining Massive Datasets - Stanford University. YouTube. https://www.youtube.com/watch?v=xoA5v9AO7S0&list=PLLssT5z_DsK9JDLcT8T62VtzwyW9LNepV.
[3] Cates, J. (Director). (2019, March 14). How to design and build a recommendation System pipeline in Python (Jill Cates) [Video file]. Retrieved March 28, 2021, from https://www.youtube.com/watch?v=v_mONWiFv0k
[4] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4: 19:1–19:19. https://doi.org/10.1145/2827872
[5] TMDB 5000 Movie Dataset: https://www.kaggle.com/tmdb/tmdb-movie-metadata
[6] Malaeb, M. (2020, September 02). Recall and precision at K for recommender systems. Retrieved May

11, 2021, from
https://medium.com/@m_n_malaeb/recall-and-precision-at-k-for-recommender-systems-618483226c54
[7] Wikimedia Foundation. (2021, May 9). IMDb. Wikipedia. https://en.wikipedia.org/wiki/IMDb.
[8] Banik, R. (2017, November 10). The Movies Dataset. Kaggle.
https://www.kaggle.com/rounakbanik/the-movies-dataset.
[9] Reikeras, H. (n.d.). How to Build a Content-Based Recommender System For Your Product. The OfferZen Community Blog.
https://www.offerzen.com/blog/how-to-build-a-content-based-recommender-system-for-your-product.