CSP 571 - PROJECT PROPOSAL

FALL 2022

Project Team:

1. Shashank Parameswaran (Team Leader)
2. Yasha Srinivas

1 Abstract

In today's world, almost everyone prefers to watch movies from the comfort of their homes. Streaming behemoths like Netflix and Amazon take advantage of this, and their services on OTT have a massive number of subscribers. They keep these subscribers by using a recommendation engine that gives them a list of movies they would prefer to watch based on their previous streaming patterns; even compare the ratings of other users who have watched and rated a particular movie/show. In this project, we have tried to understand how the recommendation engine predicts movies for different users.

2 Introduction

A recommendation engine provides suggestions to users based on browsing history/preferences set by users/ratings provided by users. Powerful recommendation engines are used every time someone selects a TV show using Netflix's "You May Also Like..." feature or purchases a product that Amazon suggests. Content-based, collaborative, or a combination of the two data-filtering methods are generally used by recommenders to provide results.

Content-based filtering: The "Similar items include..." recommendations employ this kind of filtering. Predictions about the actual characteristics of the goods and services being provided are made using content-based filtering.

Collaborative filtering: The "People who watched this show also watched..." types of recommenders employ this filtering technique. In order to predict what a person will like based on how their choices compare to those of other users. Collaborative filtering analyses behavioural data.

Hybrid filtering: This is the most effective form of recommendation engine as it overcomes the drawbacks of both content-based and collaborative filtering methods by collaborating them into one model.

In this project we will use a user-user based collaborative filtering technique to predict recommendations to users.

3 Dataset

3.1 Dataset used

MovieLens dataset from Kaggle will be used for this project. It has 5 files:

File	Size ($n \times p$)	Description
Ratings	20M*4	Contains User IDs and ratings for movies. Ratings are provided on 5-star scale
Movie	27K*3	Used as a lookup for identifying movies from movie ID and also has genre information
Tags	465K*4	These are user generated tags for movies
Genome Tag	1128*2	Provides tag descriptions for Tag IDs
Genome scores	11M*3	Provides a relevance score for the tag with the associated movie

Due to restrictions in the local RAM, only 4M data points were considered from the rating dataset.

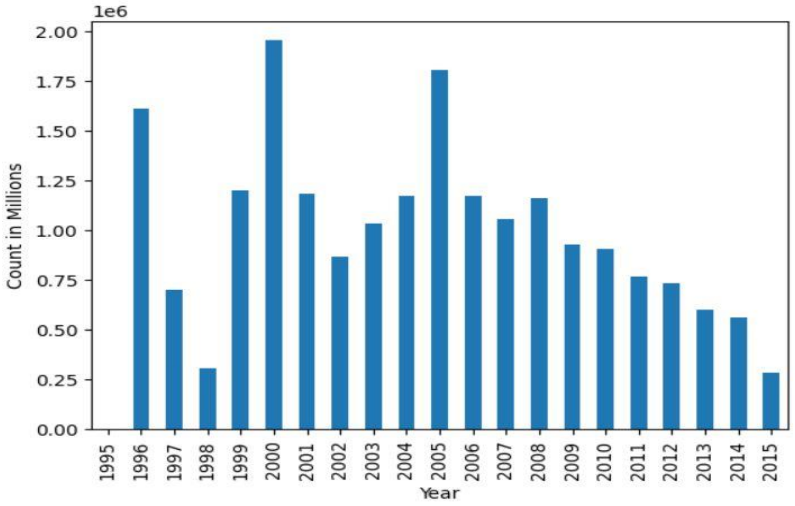


Figure 1: Movie count by Year

Only the Tags dataset has some null values in it. Other files do not have missing datapoints. Median value of rating is 3.5. The ratings range from the year 1995 to 2005.

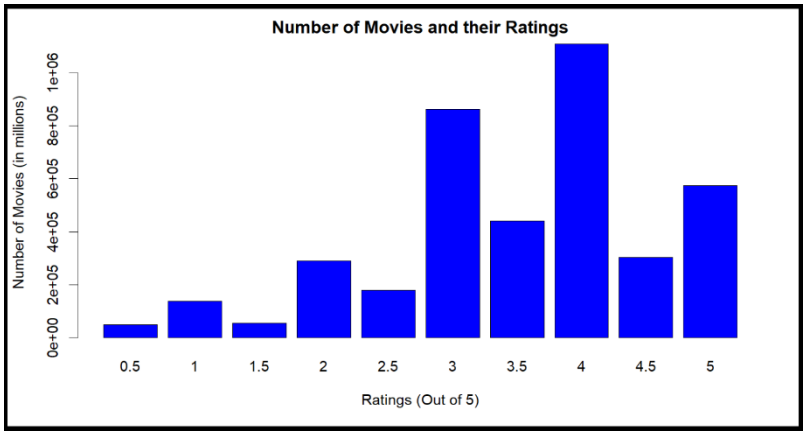
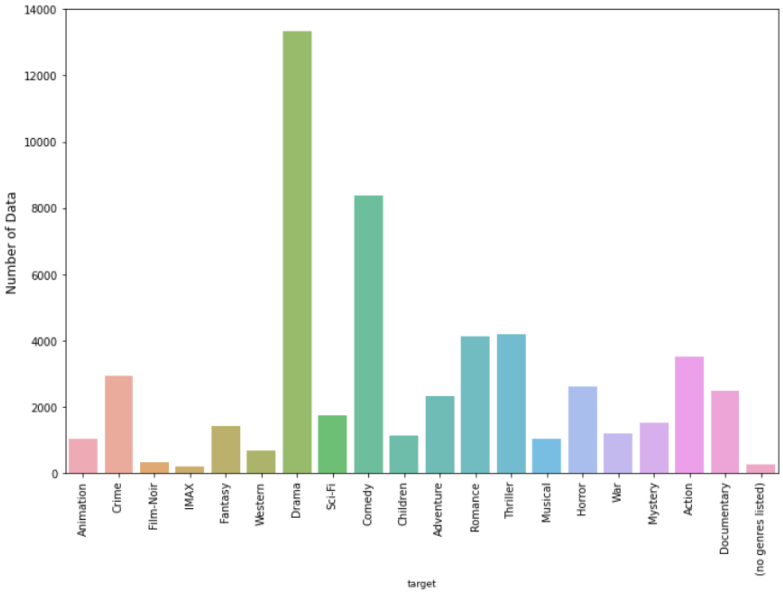


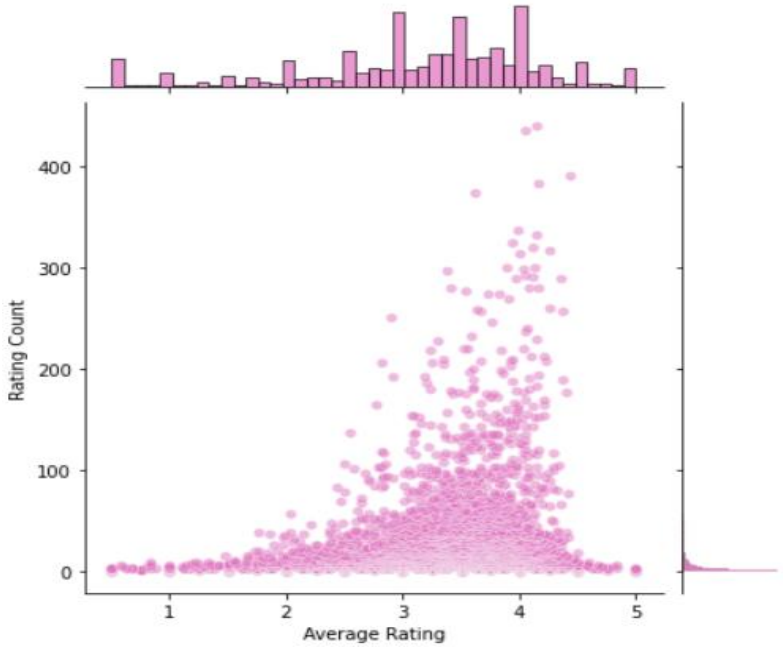
Figure 2: Movie count by Ratings

The plot above gives a visualization of the number of movies and their ratings. Maximum number of movies have a rating of 4, followed by 3.



The plot above visualizes the movies and their genres. It is noticeable that maximum number of movies are classified as "Drama".

Figure 3: Movie count by Genre



The plot was obtained by merging the data from the ratings and the movies and was grouped by the movie ID. The visualization gives us the average rating of the movies as watched by all users.

Figure 4: Histogram of ratings

3 Data Cleaning

The rating dataset contains 4M data points of user ratings. Each row represents a rating from a particular user for one movieId. Hence, each row is unique across user and movieId. The rating dataset is pivoted to produce a matrix of user and ratings. This matrix is a sparse matrix as not all users have watched all movies and it will contain a lot of null values. In order to reduce the sparsity of the matrix and improve the accuracy of the model, movies that were not watched by at least 1% of the total users were removed. This reduced the total number of movies from 19K to 3K. Then, all the null values were replaced with 0.

4 Methodology

This project will use 2 collaborative filtering techniques for the recommendation system. The first one is a model-based approach that uses matrix factorization. It basically creates a matrix where the features will represent the movie title and each row represents the user. A simple example is as follows, person A has watched movie M1 and M2 and gives a high rating for both movies. Person B has watched movie M2 and gives a high rating for M2. Now, our model suggests movie M1 for Person B. Matrix factorization techniques work better in general because it can identify hidden relationships between different users and movies. Here is the mathematical formulation of the matrix factorization of R (user-movie matrix):

$$R \sim P.Q^T = \hat{R}$$

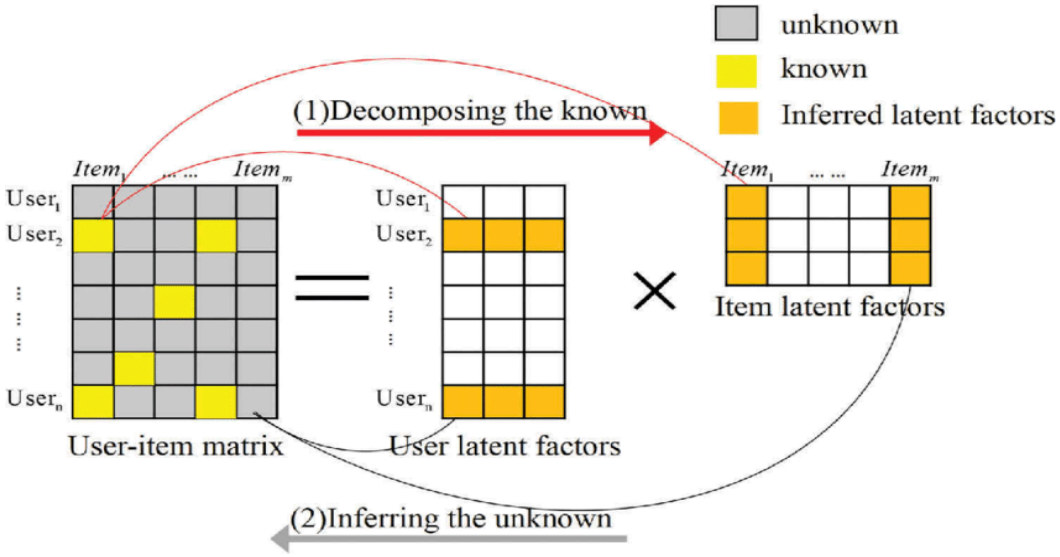


Figure 5: Matrix factorization illustration

If we assume that k hidden features affect the relationship in the matrix with U users and M movies, then P will be a $U \times k$ matrix which will contain the strength of associations between users and the features. Q will be a $M \times k$ matrix which will contain the strength of associations between movies and features. In order to estimate P & Q, we use a gradient descent algorithm to iteratively change the values of a randomly initialized P & Q until $P.Q^T$ is close enough to R. The error (E) that needs to be minimized so that we achieve this can be written as:

$$P, Q = argmin_{P,Q}(R - P.Q^T) + \frac{\beta}{2}\left(||P||^2 + ||Q||^2\right)$$

The second term is a penalty term that is used to avoid overfitting similar to the ridge regression equation.

The second approach used is Cosine Similarity which compares the similarity between two non-zero vectors (irrespective of their size). In simple words, this is the cosine of angle between the two vectors.

Mathematically,

$$Cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \, \|\vec{b}\|} = \frac{\sum_1^n a_i b_i}{\sqrt{\sum_1^n a_i^2} \sqrt{\sum_1^n b_i^2}}$$

where, $\vec{a} \cdot \vec{b} = \sum_1^n a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$ is the dot product of the two vectors.

Cosine Similarity Formula

The result obtained from it, is a value that ranges between 0 and 1. As the angle gets smaller, the value of cosine similarity gets closer to 1, this implies that the two vectors are highly similar.

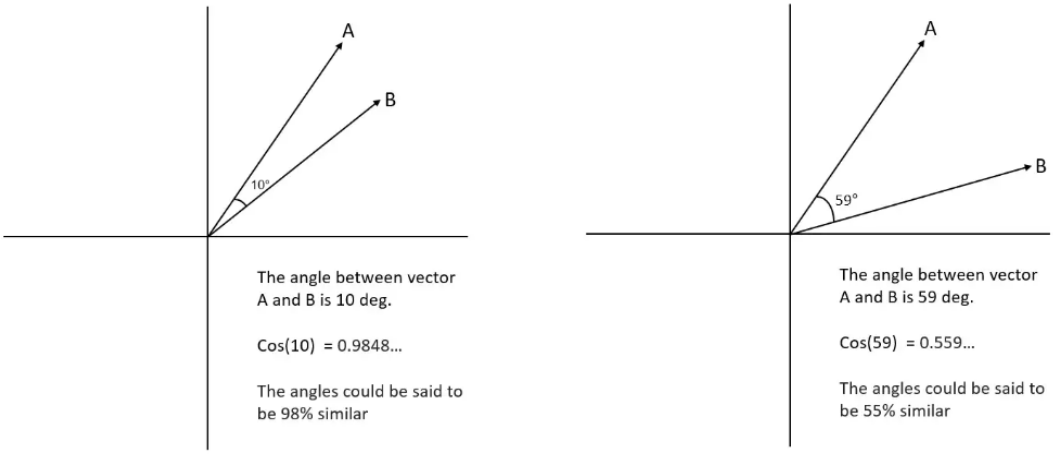


Figure 6: Left: Two vectors with 98% similarity based on cosine angle between the vectors. Right: Two vectors with 55% similarity based on cosine angle between the vectors

There are two major advantages of using Cosine Similarity. Firstly, even if two vectors have a very large Euclidian Distance between them, they could still have a smaller angle between them; hence smaller the angle, greater the similarity. Lastly, on a multidimensional space, the cosine similarity captures the angle between them and not the magnitude.

4.2 Metrics for evaluation

For matrix factorization, Mean Absolute Error (MAE) and Mean Cubic Error (MCE) were used to compare the train and test sets. Mean Absolute Error would give an idea of the absolute error on our predictions and Mean Cubic Error would help us penalize predictions that are far off from the true value. Comparing both metrics would give an idea on the performance of the model.

5 Experiment and Analysis

5.1 Matrix Factorization

The dataset was split into train and test matrices. Splitting into train and test is a challenge here as unlike the conventional way of splitting into train and test, we need to ensure that each row of train and test contains all the users. First, train matrix was assigned as a copy of the user-movie matrix. The test matrix is initialized as 0s with the size of the user-movie matrix. Then

for each user, 80% of the ratings in the train matrix were assigned the value of 0 and the corresponding rating was assigned in the test matrix. The model produces the predicted ratings by taking the train matrix as the input and the model is evaluated for all the test ratings/observations. All values in the predicted matrix with values less than 0 and greater than 5 are capped to the respective values.

The model was trained based on several hyper-parameters – features (k), learning rate (α), regularization penalty (β) and epochs. The model was trained for 50 epochs and the best stopping point was determined by visualizing the loss graphs.

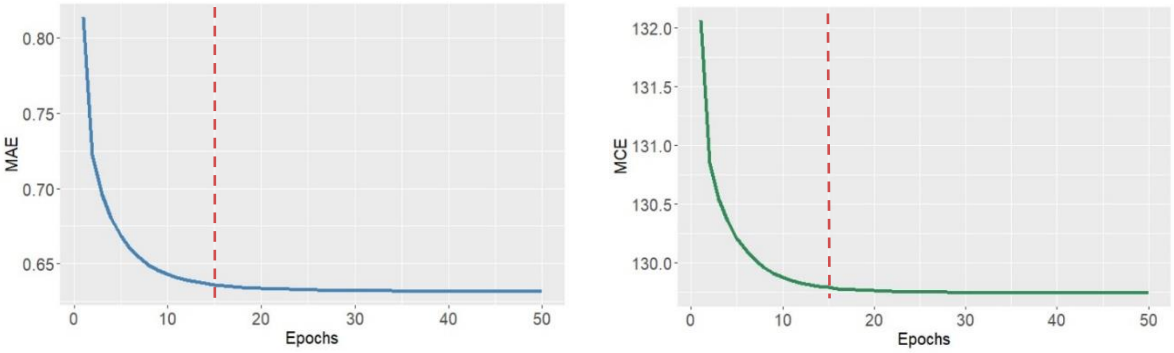


Figure 7: Left: Mean Absolute Error vs Epochs. Right: Mean Cubic Error vs Epochs

The ideal number of epochs that the model needed to be trained is 15 as after that the MAE and MCE values starts to plateau and there is not much improvement in the model post that.

The results provided by collaborative filtering (method 1) and cosine similarity (method 2) are similar. Let us consider an example to clearly explain this. Here, the user has watched the following movies, and based on this, we make the following observations about both methods.

Sample 1

User ID= 58 has rated the following movies high:

Table 1: Movies watched by User ID 58

MovieID	Rating	Title	Genre
1247	5	Graduate, The (1967)	Comedy|Drama|Romance
1227	5	Once Upon a Time in America (1984)	Crime|Drama
1228	5	Raging Bull (1980)	Drama
1086	5	Dial M for Murder (1954)	Crime|Mystery|Thriller
1244	5	Manhattan (1979)	Comedy|Drama|Romance

Using method 1, the model recommends the following top 5 movies:

Table 2: Movies recommended by Matrix Factorization

MovieID	Title	Genre
1131	Jean de Florette (1986)	Drama|Mystery
1178	Paths of Glory (1957)	Drama|War
3307	City Lights (1931)	Comedy|Drama|Romance
5008	Witness for the Prosecution (1957)	Drama|Mystery|Thriller
912	Casablanca (1942)	Drama|Romance

Using method 2, the model recommends the following top 5 movies:

Table 3: Movies recommended by Cosine Similarity

MovieID	Title	Genre
105	Bridges of Madison County, The (1995)	Drama|Romance
1103	Rebel Without a Cause (1955)	Drama
1204	Lawrence of Arabia (1962)	Adventure|Drama|War
912	Casablanca (1942)	Drama|Romance
1248	Touch of Evil (1958)	Crime|Film-Noir|Thriller

Both the models suggest Casablanca (1942) as a top recommendation. Also, the models predict movies which fall in the Crime/Mystery/Thriller genre.

Sample 2

Consider another instance. Here, the user has watched the following movies, and based on this we make the following observations about both methods.

User ID = 281 has rated the following movies high:

Table 4: Movies watched by User ID 58

MovieID	Rating	Title	Genre
1095	4	Glengarry Glen Ross (1992)	Drama
1198	4	Raiders of the Lost Ark (1981)	Action|Adventure

318	4	Shawshank Redemption, The (1994)	Crime|Drama
898	4	Philadelphia Story, The (1940)	Comedy|Drama|Romance
912	4	Casablanca (1942)	Drama|Romance

Using matrix factorization, the model recommends the following top 5 movies:

Table 5: Movies recommended by Matrix Factorization

MovieID	Title	Genre
111	Taxi Driver (1976)	Crime|Drama|Thriller
1178	Paths of Glory (1957)	Drama|War
1207	To Kill a Mockingbird (1962)	Drama
1213	Goodfellas (1990)	Crime|Drama
593	Silence of the Lambs, The (1991)	Crime|Horror|Thriller

Using cosine similarity, the model recommends the following top 5 movies:

Table 6: Movies recommended by Cosine Similarity

MovieID	Title	Genre
44555	Lives of Others, The (2006)	Drama|Romance|Thriller
1186	Sex, Lies, and Videotape (1989)	Drama
1203	12 Angry Men (1957)	Drama
593	Silence of the Lambs, The (1991)	Crime|Horror|Thriller
1962	Driving Miss Daisy (1989)	Drama

Both the models suggest Silence of the Lambs, The (1991) as a top recommendation. A lot of movies in the user's predicted genre are dramas and the User has rated Drama movies high.

6 Conclusion

This project uses 2 recommender systems to recommend movies. The models can predict any number of movies to be recommended. Both models seem to offer similar recommendations to the user. But there are a few trade-offs. Matrix factorization takes more memory and time to run. It also requires hyperparameter optimization. However, once the predicted matrix is generated, inference is fast. Specifying the number of features that would be relevant to the problem gives more control. The Cosine similarity method takes a user as input and generates

recommendations. It does not take much memory but when it needs to be done for a large number of users, it is slower (based on the methodology used).

The Matrix factorization model has an MAE of 0.6 which suggests that predicted ratings are not very far off. Comparing the MCE graph for different models (tuned with hyper-parameters) helps us ensure that predicted values are not outliers. The recommendations provided by the model outputs make practical sense and model output can be tuned to provide any number of recommendations.

7 Future Work

There are many improvements that can be done to improve our user-based recommendations. First, only the ratings dataset was used for building the model. The genre or tag dataset can also be used in a similar fashion to help with this. Predictions from genre datasets (rather than movies) can assist in drilling down on the types of movies to consider. Then Matrix factorization or Cosine similarity can help in providing more targeted recommendations. If we are interested in the top 5 recommendations, it need not come from the same model. As suggested above, it could be a mix of both predictions from movies, and predictions from genre & movies, or even a mix of item-item based recommendations. The tags dataset can be used in tandem with the genome dataset that contains relevance score for the tags and help in produce similar recommendations.

Not all users might have watched movies that are available on an OTT platform. So, for first-time users, recommendations can come in the form of most watched movies, or genre/Actor filtered based on a survey during signup.

Other techniques like Content based filtering can also be used to compare, benchmark and help with additional recommendations. Models such as hybrid recommenders help in incorporating a combination of collaborative and content-based filtering and have proven to provide more accurate recommendations. They can help with sparse matrices or the cold start problem.

8 References

[1] 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

[2] Vilakone, P., Park, DS., Xinchang, K. et al. An Efficient movie recommendation algorithm based on improved k-clique. Hum. Cent. Comput. Inf. Sci. 8, 38 (2018).

[3] Chen Li and Cheng Yang, "The research based on the Matrix Factorization recommendation algorithms," 2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), 2016, pp. 691-698, doi: 10.1109/IMCEC.2016.7867298.

[4] Alake, R. (2021, December 15). Understanding Cosine Similarity and Its Application. Medium. https://towardsdatascience.com/understanding-cosine-similarity-and-its-application-fd42f585296a

[5] GeeksforGeeks. (2020, October 6). Cosine Similarity. https://www.geeksforgeeks.org/cosine-similarity/

[6] (2021, August 10). How to Calculate Cosine Similarity in R. R-bloggers. https://www.r-bloggers.com/2021/08/how-to-calculate-cosine-similarity-in-r/

[7] Prabhakaran, S. (2022, April 20). Cosine Similarity – Understanding the math and how it works (with python codes). Machine Learning Plus. https://www.machinelearningplus.com/nlp/cosine-similarity/

[8] Javed, M. (2021, December 16). Using Cosine Similarity to Build a Movie Recommendation System. Medium. https://towardsdatascience.com/using-cosine-similarity-to-build-a-movie-recommendation-system-ae7f20842599

[9] Stieber, B. (n.d.). Recommending Songs Using Cosine Similarity in R. Brad Stieber. https://bgstieber.github.io/post/recommending-songs-using-cosine-similarity-in-r/