Title: G15 - Movie Recommendation System: Abhi Sathya, Xinying Lyu

Abstract
This project aims to develop an advanced movie recommendation system leveraging both collaborative and content-based filtering methods, enriched with machine learning techniques. By integrating these approaches, our models will not only consider user ratings but also the content features of the movies to provide personalized recommendations.

Introduction and Motivation

Tailored content has become the name of the game for every social media and content platform. From the addicting "For You" pages on TikTok and Instagram, and the ever-intriguing YouTube recommendation algorithm, to the perfectly curated personalized playlists and suggestions from Spotify and Netflix (Bennett and Lanning, 2007). The motivation for this project is to pull back the curtain on this technology and understand what makes it work.

Methodology

Dataset: We used MovieLens datasets 25M for collaborative filtering. Which includes 25 million ratings and one million tag applications applied to 62,000 movies by 162,000 users. Includes tag genome data with 15 million relevance scores across 1,129 tags, released by 12/2019. We used MovieLens-Small for Content-Based Filtering. Which includes 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users. The smaller dataset was used due to a computational bottleneck.
Visualization: Preprocessed the data, extracting useful columns, dropping any empty entries, and movies with very few ratings. Performed exploratory visualization to understand the overall data trends, user behavior, and rating distribution.
Content-Based Filtering – Data Preprocessing: The ratings in the **MovieLens dataset** ranging from 0.5 to 5.0 with a step size of 0.5 were normalized to fall in between (0, 1], with step size 0.1. This was done because models trained on unnormalized labels tended to predict outside the given rating range. Rare movies and rare users were then pruned from the dataset. Rare movies and rare users are defined by a threshold on the number of ratings, which was set to 15 for movies and 30 for users. Finally, given the tendency of users to rate movies higher, the number of entries for each rating was augmented to the maximum frequency. This approach prevented the model from skewing predictions high.
Content-Based Filtering – Cosine Similarity: The baseline for this approach was implemented by finding the Cosine similarity between feature vectors for a movie, and a user profile. Each movie is vectorized by computing the sum of the one-hot vectors representing its genres. A movie feature vector $m_i = [g_{i1}, g_{i2}, \ldots, g_{ij}];\ g_j \in G = \{genres\}$, and some $g_{ij} = 1$, if m_i has genre G_j, otherwise $g_{ij} = 0$. A user profile p_k is the weighted mean of the movie features vectors for movies rated by user k. For some user k and $I_k = \{movies\ rated\ by\ k\}$ profile is computed as follows:

$$p_k = \frac{\sum_{i \in I_k} w(r(i,k)) \times m_i}{[\max\left(\sum_{i \in I_k} g_{ij}, 1\right); for\ g_j \in G]}$$

Where w is a weighting function, $w(x) = x$ that accepts $r(i,k)$ the rating that user k has given to movie i. The denominator vector here, $[\max\left(\sum_{i \in I_k} g_{ij}, 1\right); for\ g_j \in G]$, normalizes the weighted sum for each genre by the number of ratings for that genre. The max function prevents divide-by-zero. Therefore, for each user k, we get a j-dimensional profile vector. The predicted rating for an unseen movie a for a user k is given as follows:

$$\hat{r}(a,k) = \text{Cos}\ (p_k, m_i)$$

Content-Based Filtering – Multi-Layer-Perceptron: For the MLP implementation, the input is passed through a model with 6 fully-connected layers and are then passed through the ReLU activation function, except at the last layer where the output of size 1 is passed through the linear/identity function, the result is the model's prediction. The optimizer for this model is Adam. In between two fully connected layers, layer normalization is performed to combat overfitting[1]. The feature input for this model is a $2j$-dimensional vector. Where j is the number of genres in the dataset. The input is $2j$-dimensional because it concatenates the profile vector p_k for some user k and the feature vector m_i for a movie i. This method was chosen over other the other feature vector inputs because it allowed the model to learn better. To provide a clearer idea the inputs to the model are as follows:

- $train_{labels} = [r(i,k)]$; where i is a movie user k has rated.
- $train_{features} = [p_{k1}, ..., p_{kj}, m_{i1}, ..., m_{ij}]$; where p_k, m_i are the profiles for user k and movie, i respectively.

Two versions of this model were trained, one to capture user likes and another to capture user dislikes. This is because the initial model MLP_{likes} was unable to capture user dislike preferences effectively. Henceforth, these models shall be referred to as MLP_{likes} and $MLP_{dislikes}$, and their predictions $\hat{r}_{likes}$, $\hat{r}_{dislikes}$.
The profile p_k for a user k is calculated in the same way as for cosine similarity, however, the weighting function w is different for each of the MLP models. For this model w is as follows:

$$w_{like}(x) = (x - 0.3)^3$$
$$w_{disike}(x) = \big((1.1 - x) - 0.3\big)^3$$

This weighting function performed better than a simple linear approach and allowed lower ratings to contribute equally (negatively) to a user profile.

Neural collaborative filtering: First, we utilized generalized matrix factorization (GMF) to capture linear relationships, which creates user-item matrix and fills in empty values using SVD. The interaction y is denoted as the inner product of p_u and q_i, where **p**u denotes the latent vector of user, and q_i denotes the vector of items, which is movies in this case. With this inner product, we can have the cosine similarity measured. $\hat{y}_{ui} = p_u^T q_i$

Then we used multi-layer perceptrons (MLP) to capture non-linear relationships, based on user-preference similarity, which use ReLU as the activation function. To combine the linear layer with the non linear layer together, and to ensure the performance of the model by not restricting them with same size of embeddings, we concatenate the last hidden layer using this formula:

$$\phi^{GMF} = \mathbf{p}_u^G \odot \mathbf{q}_i^G,$$
$$\phi^{MLP} = a_L(\mathbf{W}_L^T(a_{L-1}(...a_2(\mathbf{W}_2^T \begin{bmatrix} \mathbf{p}_u^M \\ \mathbf{q}_i^M \end{bmatrix} + \mathbf{b}_2)...)) + \mathbf{b}_L),$$
$$\hat{y}_{ui} = \sigma(\mathbf{h}^T \begin{bmatrix} \phi^{GMF} \\ \phi^{MLP} \end{bmatrix}),$$

Where **W** denotes the weight matrix, **b** denotes the bias vector and **a** denotes the activation function, **G** and **M** denotes the GMF layer and MLP layer respectively. This layer is called NeuMF, which used sigmoid activation function.
The loss function we used is represented in this formula:

$$L = -\sum_{(u,i)\in\mathcal{Y}} \log \hat{y}_{ui} - \sum_{(u,j)\in\mathcal{Y}^-} \log(1 - \hat{y}_{uj})$$
$$= -\sum_{(u,i)\in\mathcal{Y}\cup\mathcal{Y}^-} y_{ui} \log \hat{y}_{ui} + (1 - y_{ui}) \log(1 - \hat{y}_{ui}).$$

Where Y denotes the set of observed interactions in **Y**, and Y^- denotes the set of negative instances. We utilized Pandas library to convert MovieLens datasets into dataframe and left with only ['userId', 'movieId', 'rating', 'timestamp'] columns for user-item interaction model training and testing. This dataframe is separated using the

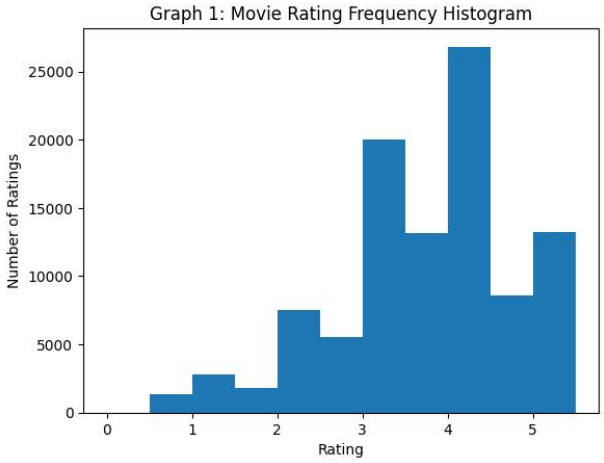
---

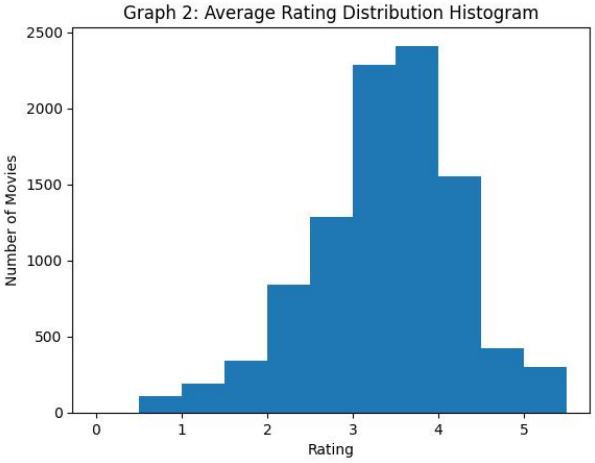
Li, M. *et al.* (2022) 'MLP4Rec: A pure MLP Architecture for Sequential Recommendations', *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence* [Preprint]. doi:10.24963/ijcai.2022/297.

random_split method into training data, evaluation data, and test data, according to the Train-Test-Val ratio [0.8, 0.1, 0.1] respectively.

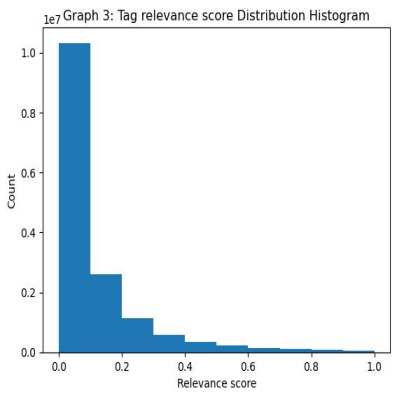
Empirical Results:

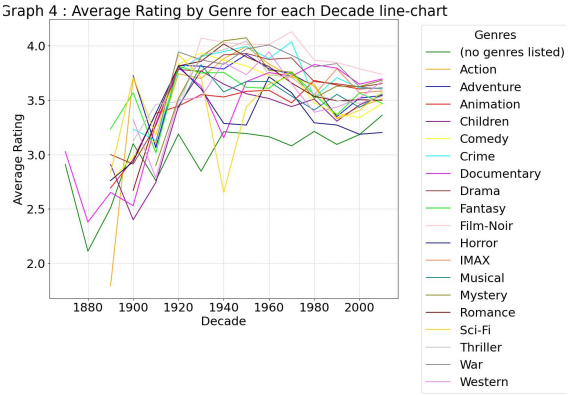
Visualization



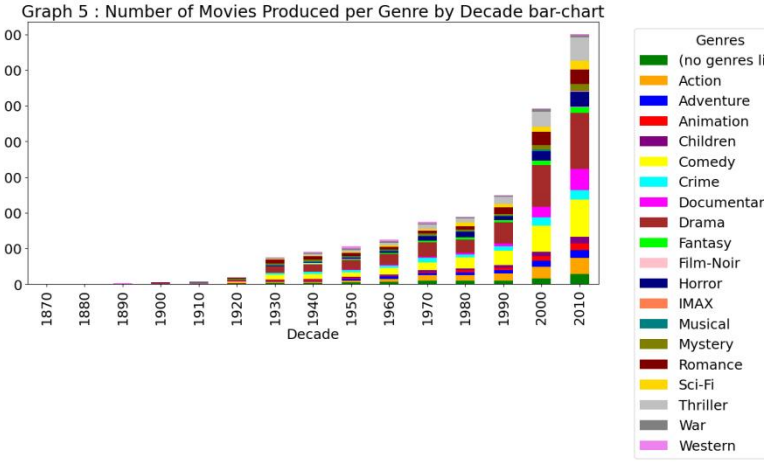


We started with some visualization to understand the broad trends in our dataset. The above histograms give us an idea of how users tend to rate movies. The ratings seem slightly skewed to the higher end of the range.





By computing the mean median of the relevance score in the dataset, we found that the median lies on the interval between 0 to 0.2, so we plot another histogram graph 3 which visualizes movie-tag relevance distribution data. According to this diagram, since most of the relevance scores of movies and tags are significantly low, indicating these tags may not be particularly descriptive or applicable to those corresponding movies, we should focus more on the movie and tag pairs with high relevance.



We've also grouped the movies into year groups and calculated the mean rating for each genre. We made this line plot graph 4 to visualize the trend of average rating by genre and decade. As the dataset collects movies from different years, we've done a visualization of number of movies produced per genre

by decade in graph 5. We could see ascending increasing rate over the whole time-period, the trend of movies produced for each genre over decades, and a clear understanding of the most and least commercially successful film genre in each time-period.

Recommendation Systems

The goal of this project is to build a recommender system for movies. Therefore, the primary metrics for evaluation are RMSE for error in ratings, and Precision@k and Recall@k for recommendations. The formulae for these metrics are:

- $RMSE(x, y) = \sqrt{\frac{1}{n}\sum_{i=0}^{n}(x_i - y_i)^2}$
- Precision@k $= \frac{\#\ recommendations\ relevant\ at\ k}{\#\ of\ recommendations\ at\ k}$
- Recall@k $= \frac{\#\ recommendations\ relevant\ at\ k}{\#\ of\ relevant\ recommendations}$

Content-Based Filtering: The objective of this method is to demonstrate how a model can learn a user's content (genre) preferences and make targeted recommendations. For each of the models, we set a like threshold ≥ 0.7, and dislike threshold ≤ 0.4. In the MovieLens dataset, this translates to a user liking a movie if they rate it ≥ 3.5 and disliking a movie if they rate it ≤ 2.0. These thresholds were chosen as a reasonable bound to represent "like" and "disliked" movies. Additionally, the final recommendation metric values are the recommendation metrics averaged across their values for all users.

Cosine Similarity: This model is implemented as a baseline and is quite a popular system for recommendations and document retrieval tasks.[2] The dataset is divided into Train-Test data using a 70:30 ratio. User profile vectors are generated using the training data. The model's predicted ratings for test data are then computed as the cosine similarity between user profile and movie features. The RMSE for this model is 0.3907. At k = 15, the Precision for top-k likes and dislikes = 62.5% and 9.79% respectively. Recall for top-k likes and dislikes = 0.8%, 55.2%. From the evaluation metrics, it is evident that if the model makes a recommendation for top-k likes it is reasonably likely to be correct. However, it fails to identify many true likes, leading to low recall. In contrast, the model has poor precision for predicting likes, therefore, a prediction for a top-k dislike is unlikely to be correct. It does capture a higher number of true dislikes, leading to higher dislike recall. What we understand from this is that the model is too conservative in predicting user likes.

Multi-Layer Perceptron: Following the split of data into Train-Test-Val subsets. User profiles are created using the training data. The profiles along with the movie feature vectors are fed into the MLP with ratings as labels. The training loss for the "like" model = 0.1486, the validation loss = 0.1694. User profiles were then generated from the test data and were concatenated with feature vectors for movies in the test data. The "like" model was then made to predict ratings for the test data. The test loss = 0.1660. This is already significantly better than the Cosine Similarity baseline. Similarly for the dislike model, train loss = 0.1642, validation loss = 0.1726, and the test loss = 0.1708. Coming to the recommendations, since we have two models, we use the like model to predict likes and the dislike model to predict dislikes, with a catch.

- Let, $likes = \{\hat{r}_{likes}(i,k) \geq 0.7\}$ and $dislikes = \{\hat{r}_{dislikes}(i,k) \leq 0.4\}$.
- Then, $top_{likes} = likes - (likes \cap dislikes)$ and $top_{dislikes} = dislikes - (likes \cap dislikes)$.

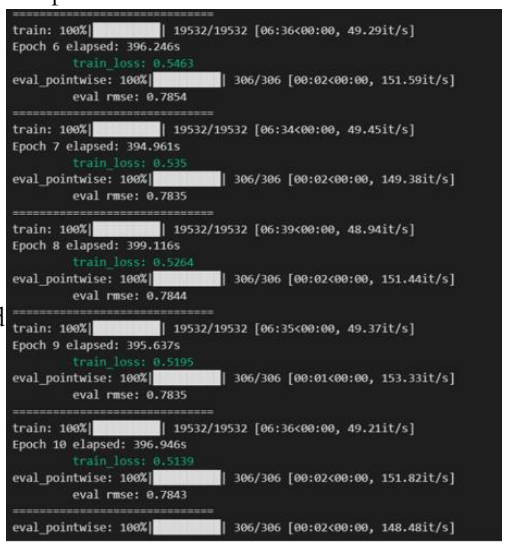
We perform this set operation because for some users the models are sensitive to predicting a few ratings on the extremes regardless of the feature vectors. This set operation irons out those outliers and prevents an entry from showing up in both top likes and top dislikes. We then compute Precision@k and Recall@k with top_{likes} and $top_{dislikes}$ as our models' predictions. We then get, with k = 15, like Precision@k = 84.7%, Recall@k = 83.57%, and dislike Precision@k = 66.1%, Recall@k = 46.16%. The MLP models, working in tandem, perform much better than the baseline. Their precision for both likes and dislikes is significantly

[2] Gomes, N.D. (2023) *The cosine similarity and its use in recommendation systems*, *Medium*. Available at: https://naomy-gomes.medium.com/the-cosine-similarity-and-its-use-in-recommendation-systems-cb2ebd811ce1 (Accessed: 06 December 2023).

higher, and the cautious tendency of the baseline has been overcome. The "like" performance, in particular, is quite good with the model having both high precision and high recall showing it has effectively learned user content preferences.

MLP-Train-Test-Val Splits: The data is split into three groups, according to the Train-Test-Val ratio: 70:15:15. Although bootstrapping and k-fold were discussed in lecture, implementing bootstrapping hampered the performance of this model. The model was plagued with overfitting issues for each bin, and reducing complexity and adding dropout layers did not seem to help. The model would learn reasonably, with the validation loss reducing to 0.1734, before increasing to 0.2014, while the train loss continued to decrease. Stopping the training at the minima for the loss and reducing the learning rate and reducing the number of training epochs only yielded marginal gains, if any. Therefore, a 70:15:15 Train-Test-Val split was used.

MLP-Hyperparameters: The layer sizes for the model are [128, 64, 32, 16, 8, 1]. The model was trained for 250 epochs with a batch size of 128 at an initial learning rate of 0.01. The learning rate was set to decay exponentially at a rate of 0.98 for every 10000 steps. The weight function was changed from that used in cosine similarity to yield optimum results. Additionally, movie feature vector smoothing, and normalization was performed. Smoothing was achieved by adding a small constant to the vector, and the feature vectors were normalized using their L2-norm. Dropout layers were tested as part of the architecture of the MLP but had little to no effect on the loss.

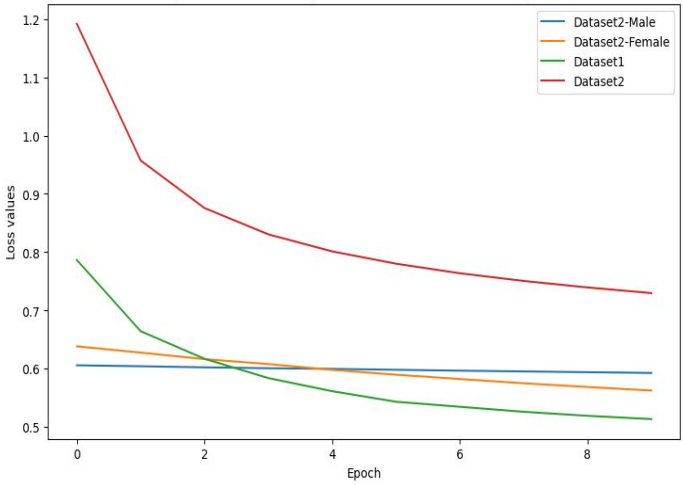


Neural Collaborative Filtering: Through utilizing Tensorflow and torchvision and after hyper parameter tuning, we trained the model on the 25M MovieLens dataset(dataset1) with learning rate = 1e-3, number of epochs = 10, batch size = 2048, and hidden unit = [128,64,32] respectively, we got the cross-entropy loss value for each epoch as follows:

The loss starts at the value of 0.7867, and as the training proceed it has a continuous and steady decrease, with a minimum loss value to be 0.5139 after training finished.

Then, as to see if there could be further improvements of the loss, we performed custom binning for gender. And as the 25M MovieLens dataset(dataset1) doesn't contain any information with the user gender, we trained the model based on another dataset - the 1M MovieLens dataset(dataset2), which captures user gender for each rating. This time the model is trained separately for male and female users. I've adjusted the learning rate to be 1e-4 this time. For the smaller dataset(dataset2), the loss value start with 1.1917, and end with 0.7299; if we only consider the male users, we have the loss value to be 0.6061 at the start and 0.593 after training; if we only consider the female users, we have the loss value to be 0.6386 at the start and 0.5629 after training;. The results of loss for each epoch during the training process are shown in graph 6.

Graph 6: Train loss comparison between datasets and genders



NCF Evaluation: In the training loss analysis of our NCF model, we found that the losses for male and female users decreased in a similar trend across epochs and were notably lower compared to the whole dataset, indicating more efficient learning from gender-specific data. In contrast to dataset2, dataset1 had demonstrated the model's capability to scale and

adapt to larger, more complex datasets. It also indicates that the accuracy of the NCF movie recommendation model tends to be higher as the training size increase, where the model captures more nuanced similarity information though training.

Conclusion and Discussion

We developed a movie recommendation system using the MovieLens 25M and MovieLens-Small datasets for collaborative and content-based filtering, respectively. The methodology included preprocessing data, exploring trends through visualization, and implementing recommendation models. For content-based filtering, cosine similarity and a multi-layer perceptron (MLP) were used, while for collaborative filtering, a combination of generalized matrix factorization and MLP was employed. Our visualization has highlighted intriguing trends in user movie ratings and the relevance of movie tags. We've also identified patterns concerning the average movie ratings by genre over the decades and the evolution of movie genres over time. The system's performance was evaluated using metrics like RMSE, Precision@k, and Recall@k, with separate models trained for user likes and dislikes, and also explorations of gender-specific training to enhance the model's accuracy.

A potential weakness of the Content-Based Filtering method in a real-world application is the cold start problem. Particularly, in our implementation, user rating data is required to create user profiles. One of the key places Collaborative Filtering and Content-Based Filtering work well together is during this phase. Initially, when a user has just joined a content platform, content could be served to them with a preference towards collaborative filtering recommendations.

Potential Extensions

Future work could explore hybrid models that blend collaborative and content-based methods to enhance recommendation accuracy and address limitations such as the cold start problem. Additionally, integrating more complex machine learning techniques like deep learning could offer more nuanced user profiles and item categorizations, improving the personalization of recommendations.
The use of additional metadata (e.g., director, cast, and user demographics) could further refine recommendations, making them more context-aware and relevant. Lastly, considering temporal dynamics in user preferences and movie trends could add another layer of sophistication to the recommendation logic.

Related Work

The paper "The Netflix Prize" by Bennett and Lanning, 2007 employs a variant of Pearson's correlation to determine a list of "similar" movies that are predictive of enjoyment for a particular movie. Additionally, as users provide ratings, an online, real-time portion of the system computes a multivariate regression based on these correlations to generate unique personalized predictions for each predictable movie. If no personalized prediction is available, the average rating based on all ratings for the film is used. The performance of Cinematch is measured using the root mean squared error (RMSE) of the system's prediction against the actual rating provided by a subscriber.

MLP4Rec: A pure MLP Architecture for Sequential Recommendations, (Li, M. et al. (2022) describes a new sequential recommendation system called MLP4Rec, based on MLP (Multi-layer Perceptron) architectures. This system addresses limitations of self-attention models by being inherently sensitive to the order of item sequences, without needing positional embeddings. It uses a novel tri-directional fusion scheme for coherent capture of sequential, cross-channel, and cross-feature correlations in user-item interactions. MLP4Rec achieves this through a combination of sequence, channel, and feature mixers in its architecture, offering linear computational complexity and fewer model parameters compared to self-attention methods.

References

1. Li, M. et al. (2022) 'MLP4Rec: A pure MLP Architecture for Sequential Recommendations', Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence [Preprint]. doi:10.24963/ijcai.2022/297.
2. Gomes, N.D. (2023) The cosine similarity and its use in recommendation systems, Medium. Available at: https://naomy-gomes.medium.com/the-cosine-similarity-and-its-use-in-recommendation-systems-cb2ebd811ce1 (Accessed: 06 December 2023).
3. 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
4. Bennett, J., & Lanning, S. (2007). The Netflix Prize.
5. Covington, P., Adams, J., & Sargin, E. (2016, September). Deep neural networks for youtube recommendations. In Proceedings of the 10th ACM conference on recommender systems (pp. 191-198).
6. Cheng, H. T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., ... & Shah, H. (2016, September). Wide & deep learning for recommender systems. In Proceedings of the 1st workshop on deep learning for recommender systems (pp. 7-10).
7. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T. S. (2017, April). Neural collaborative filtering. In Proceedings of the 26th international conference on world wide web (pp. 173-182).
8. https://www.kaggle.com/code/ecembayindir/hybrid-recommender-project
9.