# Movie Recommendation System

Aadish Sethiya, Harshit Agarwal, Premankur Chakraborty, Ramswaroop Rathi
*Computer Science Engineering*
*Indian Institute of Technology*

***Abstract*—This project develops a movie recommendation system that suggests films based on a person's viewing history and likes. It uses different recommendation techniques to match movies to an individual's preferences and habits.**

## I. Introduction

Today's vast range of entertainment options challenges companies aiming to provide personalized user experiences. This has prompted innovation in recommendation systems. This report examines a movie recommendation system integrating content-analysis and user-behavior modeling techniques. Starting with content filtering via cosine similarity measures, it includes collaborative filtering methods leveraging **Singular Value Decomposition (SVD)**, **K-Nearest Neighbors (KNN)** regression to model user preferences, and optimization strategies inspired by Netflix Prize approaches like the **Bellkor Pragmatic Chaos** algorithm. The project aims to not only refine collaborative filtering but also explore the transformative capabilities of **deep learning** models in the realm of movie recommendations.

This report documents the progression of recommendation approaches, moving from conventional methodologies to state-of-the-art neural network architectures. It presents three distinct neural network models, each iterative advancement upon the last. The path crosses through matrix factorization methods, the assimilation of dense neural network layers, ultimately arriving at a hybrid model synthesizing collaborative and content-based filtering. This effort signifies the dedication to advancement and the constant quest for peak user experience within the fast-changing landscape of movie recommendation engines.

## II. Content Based Filtering vs Collaborative Filtering

Content-based filtering and collaborative filtering are two prominent approaches employed in movie recommendation systems, each offering distinct advantages and addressing unique challenges. Content-based filtering relies on the intrinsic characteristics of items, such as movies, and users' historical preferences to make recommendations. This method involves analyzing features like genre, cast, and plot summaries to identify patterns and suggest items that align with a user's preferences. While content-based filtering excels in recommending items with similar attributes to those a user has liked before, it may face limitations in capturing diverse tastes and discovering new, unexpected preferences.

On the other hand, collaborative filtering focuses on user interactions and preferences within a community of users. It identifies similarities between users or items based on their historical behavior, leveraging the collective wisdom of the user base to make recommendations. Collaborative filtering is particularly effective in addressing the "cold start" problem, where new items or users lack sufficient historical data. However, it may struggle when dealing with sparse data or in scenarios where users' preferences are not well-aligned with the majority of the user base. Hybrid approaches, which combine elements of both content-based and collaborative filtering, are increasingly popular, aiming to overcome the limitations of each method and provide more accurate and diverse recommendations in movie recommendation systems.

## III. Models Used

To build an efficient recommendation system, a collection of different models were used. The results obtained from them were compared, using the RMSE loss as a metric. The various models used were:

- Content Based Cosine Similarity
- Singular Value Decomposition
- Collaborative Filtering using K-Nearest Neighbours
- BellKor's Pragmatic Chaos
- Neural Networks integrated Collaborative Filtering

A more detailed description of the above models can be found below.

## IV. Dataset Used

In order to train and test the models mentioned below, we have used the **MovieLens Latest-Small** dataset. The MovieLens Latest-Small dataset is a collection of 100,836 movie ratings and 3,683 tag applications applied to 9,742 movies by 610 users. This dataset is a subset of the larger MovieLens Latest dataset and is designed to be more manageable for smaller projects or initial experimentation with movie recommendation algorithms.

Key Features of the MovieLens Latest-Small Dataset:

- **Ratings**: The dataset contains 5-star ratings for each movie-user pair.
- **Tags**: Users have applied free-text tags to movies to describe their content or themes.
- **Users**: The dataset includes information about the users who provided the ratings and tags.
- **Date Range**: The ratings and tags span from March 29, 1996, to September 24, 2018.

## V. Cosine Similarity for Content Based Recommendations

### A. Introduction

The content-based recommendation system utilizes cosine similarity for matching movies based on their content. The cosine similarity is employed to measure the similarity between the textual features of movies, such as plot summaries. By representing movies as vectors in a high-dimensional space, cosine similarity provides a measure of the cosine of the angle between these vectors, determining their similarity.

### B. Algorithm

The algorithm for content-based filtering using cosine similarity involves the following steps:

1) Text Preprocessing:
   Perform text preprocessing on the movie plot summaries, including lowercasing, removal of stop words, and stemming.
2) TF-IDF Vectorization:
   - Apply TF-IDF (Term Frequency-Inverse Document Frequency) vectorization to convert the processed text into numerical vectors.
   - TF-IDF vectorizes/scores a word by multiplying the word's Term Frequency (TF) with the Inverse Document Frequency (IDF).
   - **Term Frequency**: TF of a term or word is the number of times the term appears in a document compared to the total number of words in the document.

     $$TF = \frac{\text{Number of times term appear in document}}{\text{Total number of terms in document}}$$

   - **Inverse Document Frequency**: IDF of a term reflects the proportion of documents in the corpus that contain the term. Words unique to a small percentage of documents (e.g., technical jargon terms) receive higher importance values than words common across all documents (e.g., a, the, and).

     $$IDF = \log(\frac{\text{Number of documents in corpus}}{\text{Number of documents containing the term}})$$

   - The TF-IDF of a term is calculated by multiplying TF and IDF scores.

     $$\text{TF-IDF} = TF * IDF$$

   - Importance of a term is high when it occurs a lot in a given document and rarely in others.
3) Cosine Similarity Calculation:
   - Calculate the cosine similarity between the TF-IDF vectors of movies.
4) Recommendation:
   - Provide recommendations based on movies with the highest cosine similarity scores.

   Here is an example

```
get_recommendations('The Dark Knight Rises')
```

```
65                              The Dark Knight
299                              Batman Forever
428                              Batman Returns
1359                                    Batman
3854    Batman: The Dark Knight Returns, Part 2
119                               Batman Begins
2507                                  Slow Burn
9            Batman v Superman: Dawn of Justice
1181                                       JFK
210                               Batman & Robin
Name: title, dtype: object
```

Fig. 1: Visualization of Recommendations

### C. Implementation

The content-based recommendation system is implemented using the cosine similarity algorithm. The implementation involves using natural language processing (NLP) techniques and vectorization to quantify the similarity between movie plot summaries. The TF-IDF vectorization captures the importance of words in the plot summaries, and the cosine similarity calculation identifies movies with similar content.

## VI. Singular Value Decomposition

### A. Introduction

Singular Value Decomposition (SVD) is a matrix factorization technique used in collaborative filtering-based recommendation systems. It decomposes the user-item interaction matrix into three matrices representing users, latent factors, and items. The latent factors capture underlying patterns in user preferences, allowing the system to make predictions for missing entries in the original matrix.

### B. Algorithm and SVD Equation

The SVD algorithm involves the following steps:

1) User-Item Matrix Decomposition:
   - Decompose the user-item interaction matrix $M$ into three matrices: user matrix $U$, diagonal singular value matrix $\Sigma$, and item matrix $V^T$.
2) Matrix Approximation:
   - Reconstruct the original matrix by approximating it using the product of the three decomposed matrices.
   - The SVD equation is given by:

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

3) Prediction:
   - Use the reconstructed matrix $\hat{M} = U\Sigma V^T$ to make predictions for missing entries, representing user ratings for unrated items.

### C. Implementation and RMSE

The SVD-based collaborative filtering recommendation system is implemented by decomposing the user-item interaction matrix and reconstructing it to make predictions. The implementation utilizes matrix factorization techniques to capture latent factors contributing to user preferences and item characteristics.The RMSE (Root Mean Squared Error) for this model is 0.6645.

## VII. Collaborative Filtering using K-Nearest Neighbours

### A. Introduction

As described above, collaborative filtering uses user interactions and preferences within a community of users to predict ratings of users for movies they haven't rated themselves. To carry out this prediction, we can use the technique of K-Nearest Neighbours. This KNN model operates on the fact that users with similar preferences in the past are likely to have similar tastes in the future, for new movies.

### B. Algorithm

Before getting into the description of the algorithm, let's describe how we formulate our problem into one that can use KNN. Let's say that we want to predict the rating given to movie *m* by user *u*. In order to do this, we first look through our data-set and find the set of users which have rated movie *m*. Call this set *N(m)*. Now, we create a new dataset of only the users that belong to *N(m)* and the movies rated by them. In case there is a movie that a particular user has not yet rated, that datapoint is given the value 0.

Now, to find the users most similar to user *u* from *N(m)*, we can convert every user's movie ratings into a vector from a feature space where every dimension represents the rating given by that user for a particular movie. Next, we run the KNN algorithm to find the k most similar users to user *u*. To get the predicted rating for user *u* for movie *m*, we just take the average value of the rating for movie *m* given by each of the above computed k most similar users.

Note that we also run some code to tune the hyper-parameter k and find the optimal value of k. Thus, the sequence of events carried out is:

- Obtaining a matrix of all ratings for all movies and users.
- Mean-center the data.
- For movie *m*, find the set *N(m)*.
- Split the total dataset into 2 parts - a sort of train and test split. This is done to find the optimal value of k.
- For each user in the 'test' set, find the k most similar users for all values of k from 1 to 50.
- For each value of k, calculate the predicted rating for the test users rating for movie *m* by averaging the ratings of the best k.
- Calculate the rmse loss of predictions and thus, find the optimal value of k (for which rmse loss is the least)
- Store the optimal value of k and use this for all further predictions.
- Note that to find the k most similar users, a variety of different distance metrics were used. The above last 4 steps were carried out separately for each distance metric.

### C. Distance Metrics

Here is a list of different distance metrics used to compute similarity between 2 users.

*1) **Cosine Similarity**:* The cosine similarity between vectors A and B is given by:

$$\text{Cosine Similarity}(A, B) = \frac{A.B}{\|A\|\|B\|}$$

Effectively, this gives us the value of the cosine of the angle between the vectors A and B. The smaller the angle, the larger the value of its cosine and hence, the greater the similarity.

*2) **Jaccard Similarity**:* The Jaccard similarity between vectors A and B is given by:

$$\text{Jaccard Similarity}(A, B) = \frac{|A \bigcap B|}{|A \bigcup B|}$$

In other words, it represents the size of the intersection of A and B divided by the size of the union of the two. For every dimension in the vector, if the component of vectors A and B have the same value along that dimension, we have an intersection. Thus, jaccard is just the ratio of components where the two vectors have the exact same value.

As expected, and as will be shown below, this is not a good metric because 2 users who like the same movie might rate them differently (4.5 and 5, for example) which Jaccard similarity can't interpret.

*3) **Manhattan Distance**:* The Manhattan distance between vectors $A$ and $B$ is computed as the sum of the absolute differences between their corresponding elements:

$$\text{Manhattan Distance}(A, B) = \sum_{i=1}^{n} |A_i - B_i|$$

This distance metric measures the absolute differences along each dimension, providing a measure of similarity based on the total absolute difference between corresponding elements of the vectors.

*4) **Euclidean Distance**:* The Euclidean distance between vectors $A$ and $B$ is given by:

$$\text{Euclidean Distance}(A, B) = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2}$$

This distance metric measures the straight-line distance between two points in $n$-dimensional space, representing the magnitude of the difference between corresponding elements of the vectors.

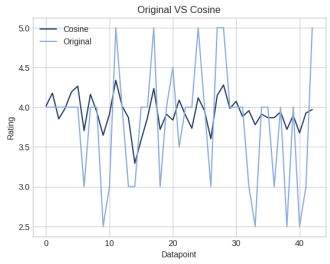
*5) Pearson Correlation Coefficient:* The Pearson correlation coefficient between vectors $A$ and $B$ is calculated as:

$$\rho_{A,B} = \frac{\sum_{i=1}^{n}(A_i - \bar{A})(B_i - \bar{B})}{\sqrt{\sum_{i=1}^{n}(A_i - \bar{A})^2}\sqrt{\sum_{i=1}^{n}(B_i - \bar{B})^2}}$$

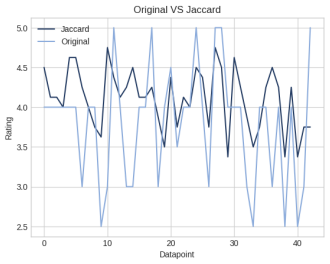
Where: $\bar{A}$ and $\bar{B}$ denote the mean of vectors $A$ and $B$ respectively.

This coefficient measures the linear correlation between $A$ and $B$, providing a value between -1 and 1. A value close to 1 indicates a strong positive linear relationship, while a value close to -1 indicates a strong negative linear relationship.

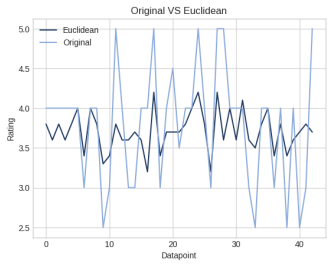
### D. Results



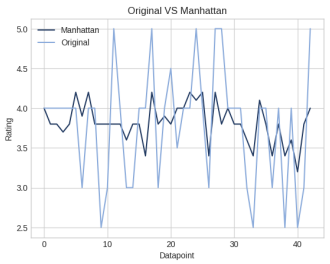

(a) Pred vs Org (cosine)




(b) Pred vs Org (jaccard)




(a) Pred vs Org (euclidean)




(b) Pred vs Org (manhattan)

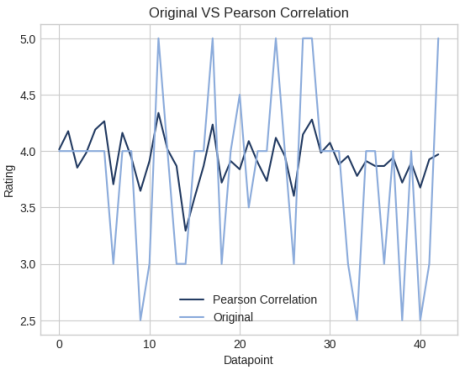
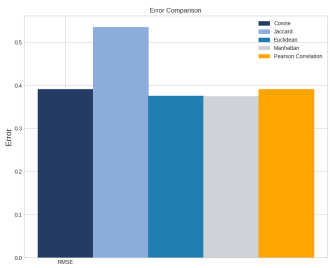
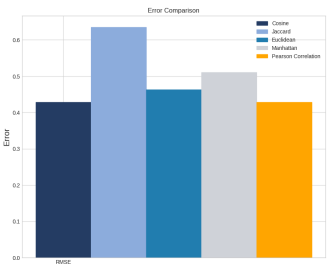



Fig. 4: Pred vs Org (pearson)

The above images show the plots of the ratings obtained for the testing data-set for each of the above distance metrics when compared with the actual true values of the ratings for the same data-set.




(a) Loss at k=10




(b) Loss across best k

Here, we see the rmse loss for all of the above distance metrics for, first - k=10 and second - the best value of k for each distance metric separately.

We see that when k=10, the best metric is cosine similarity or pearson coefficient. Note that when the data is mean centered (as in our case), these two evaluate to the exact same thing. Thus, they show the exact same results. The worst metric is Jaccard similarity, which was expected to perform poorly, as described above.

When we tune k and take the optimal value of k for each distance metric, we obtain smaller values of k for euclidean and manhattan distance metrics. Further, these also slightly outperform the cosine similarity model. This can in part be explained by the small data-set used to test the same. When using a larger data-set, cosine similarity can better generalize the trends in the data while in this particular data-set, manhattan and euclidean distance metrics captured small averages in the differentials of individual dimensions.

## VIII. BellKor's Pragmatic Chaos

### A. The Netflix Prize

The Netflix Prize was a competition launched by Netflix in 2006 to try and find the best collaborative filtering model for their own recommendation system. The task provided a data-set of their own actual users along with ratings by them. It consisted of 100,480,507 ratings that 480,189 unique users gave to 17,770 different movies. The goal was to efficiently predict ratings for films that users hadn't yet watched by using historical rating data. To win the grand prize of $1,000,000, the submitted model needed to beat Netflix's model by at least 10% in terms of RMSE loss on the test set. Netflix's own model, called CineMatch, achieved an RMSE loss of 0.9525 and so, winning teams needed to achieve 0.8572 or better.

After 3 years of the competition and numerous submissions, the contest concluded in 2009 when a team named "BellKor's Pragmatic Chaos" achieved the required benchmark and won the prize.

### B. Algorithm

The basic algorithm to find the predicted rating for a given user, movie pair is very complicated in this algorithm. There are a large number of parameters that need to be calculated to find the final prediction and the final prediction is a combination of these parameters along with a number of tunable hyperparameters.

Note that the algorithm belongs to the scientists Robert Bell, Chris Volinsky, and Yehuda Koren, the members of the winning team. This description and implementation of the algorithm was created after referring to a paper written by Koren, linked here.

Let's first start with a description of the parameters. We can describe our predictive rating as:

$$\hat{r}_{ui} = b_{ui} + q_i^T \left( p_u(t_{ui}) + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j \right)$$

Here, the first term, $b_{ui}$, is called the baseline prediction because it involves terms that are independently based on u or i but not on their interaction. The second term involved this interaction.

*1) Baseline Predictor:* Here is a more detailed view of how the baseline prediction is calculated:

$$b_{ui} = \mu + b_u + \alpha_u.dev_u(t_{ui}) + b_{u,t_{ui}} + (b_i + b_{i,Bin(t_{ui})}).c_u(t_{ui})$$

Here, $\mu$ refers to the average rating of all movies by all users part of the data-set.

The parameters $b_u$ and $b_i$ indicate the observed deviations of user u and item i, respectively, from the average. For example, suppose that we want a baseline estimate for the rating of the movie Titanic by user Joe. Now, say that the average rating over all movies, μ, is 3.7 stars. Furthermore, Titanic is better than an average movie, so it tends to be rated 0.5 stars above the average. On the other hand, Joe is a critical user, who tends to rate 0.3 stars lower than the average. Thus, the baseline estimate for Titanic's rating by Joe would be 3.9 stars by calculating 3.7 - 0.3 + 0.5.

The above are constants for each user and each movie but, we must also account for some form of time dependency. In order to do this, we need time dependent parameters. Here, $t_{ui}$ refers to the time at which the given rating was carried out.

Much of the temporal variability in the data is included within the baseline predictors, through two major temporal effects. The first addresses the fact that an item's popularity may change over time. For example, movies can go in and out of popularity as triggered by external events such as the appearance of an actor in a new movie. This is manifested in our models by treating the item bias $b_i$ as a function of time. The second major temporal effect allows users to change their baseline ratings over time. For example, a user who tended to rate an average movie "4 stars", may now rate such a movie "3 stars". This may reflect several factors including a natural drift in a user's rating scale. Thus, we have terms of the form $dev_u(t_{ui})$ and $b_{i,Bin(t_{ui})}$. These form the temporal dependency terms for users and movies. Note that here, $\alpha_u$ is a hyper-parameter that we can tune.

Another effect within the scope of baseline predictors is related to the changing scale of user ratings. While $b_i(t)$ is a user-independent measure for the merit of item i at time t, users tend to respond to such a measure differently. For example, different users employ different rating scales, and a single user can change his rating scale over time. Accordingly, the raw value of the movie bias is not completely user-independent. To address this, we add a time-dependent scaling feature to the baseline predictors, denoted by $c_u(t)$.

This completes the description of our baseline predictor.

*2) Movie - User Interactions:* This refers to the second term in our equation for the predicted rating. We have $q_i$, which act as weights to the user factors $p_u$. Together, these account for user biases and their weights for the corresponding movies. Additionally, we add a sort of bias term to approach the true value of the ratings by simply taking the set N(u), which is the set of all movies rated by user u and taking the average value of the rating for these movies. Note that the normalization for the average rating here is actually the square root of the cardinality of the set N(u), instead of just the cardinality. This was experimentally found to actually give better results.

*3) Training:* Training the above model and obtaining values for each of the parameters involved a pretty standard run of stochastic gradient descent. A train-test split was determined (80-20) and data was first trained to find the optimal values for the parameters outlined above.

Further, testing was carried out. Note that a fair bit of hyper-parameter tuning was also required to achieve the best possible model.

| | $b_u$ | $b_{ut}$ | $\alpha_u$ | $b_i$ | $b_{i,\mathrm{Bin}(t)}$ | $c_u$ | $c_{ut}$ |
|---|---|---|---|---|---|---|---|
| lrate $\times 10^3$ | 3 | 25e-1 | 1e-2 | 2 | 5e-2 | 8 | 2 |
| reg $\times 10^2$ | 3 | 5e-1 | 5000 | 3 | 10 | 1 | 5e-1 |

Fig. 6: Tuned Hyper-parameter values

The above can be referred to as the optimal values for the learning rates and regularisation constants for the corresponding parameters.

### C. Results

The computed RMSE loss on the testing data came out to be 1.045. This is compared to Netlfix's own rmse loss of about 0.95.

We can explain the slight decrease in performance from the fact that our training and testing data used 100,000 data points while the actual data-set used for the competition was the more complete version of the data-set containing over 100,000,000 such ratings. Thus, our training proved to be much less efficient.

Further, there are a few small optimizations over this basic model that involved some non-determinism in the predictions. These last mile optimizations were not completely implemented. Thus, our rmse loss, while close to the actual value, is not the same as the one obtained by the publishers of the paper.

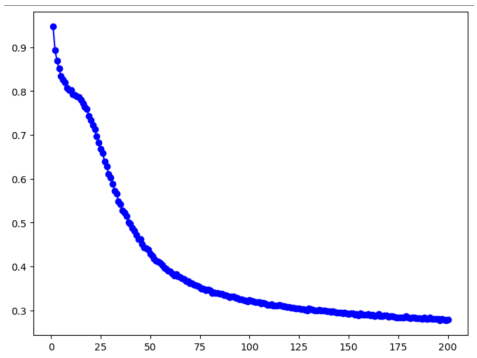



Fig. 7: Training loss per epoch

## IX. Neural Networks integrated Collaborative Filtering

### A. Model 1 - Matrix Factorization

*1) Model Description:* Model 1 employs a simple matrix factorization technique for collaborative filtering. It factors the user-product interaction matrix into user and product feature matrices, enabling the prediction of user preferences based on past interactions. Here product refers to movie.

*2) Implementation:*

- **User-Product Interaction Matrix** - The user-product interaction matrix $\text{usr}(u, p)$ represents the interaction strength between user $u$ and product $p$. If $\text{usr}(u, p) > 0$, it indicates a user's preference for the product.
- **Matrix Factorization** - The factorized matrix $\text{usr}_{\text{ft}}$ is obtained by multiplying the user-product interaction matrix usr with the product features matrix prd:

$$\text{usr}_{\text{ft}}(u, f) = \sum_{p} \text{usr}(u, p) \times \text{prd}(p, f)$$

- **Normalization** - The factorized matrix is normalized to obtain weights, representing the relative importance of each user's preferences:

$$\text{weights}(u, f) = \frac{\text{usr}_{\text{ft}}(u, f)}{\sum_{f'} \text{usr}_{\text{ft}}(u, f')}$$

- **Predicted Ratings** - The predicted ratings pred are calculated as the dot product of weights and the transpose of the product features matrix prd:

$$\text{pred}(u, p) = \sum_{f} \text{weights}(u, f) \times \text{prd}(p, f)$$

### B. Model 2 - Matrix Factorization with Embeddings

*1) Model Description:* In Model 2, a collaborative filtering approach is adopted with matrix factorization using embeddings. This model leverages neural network architecture to capture user and product embeddings, enhancing the representation of latent features. The network is trained to predict user ratings based on these embeddings, providing a personalized recommendation system.

*2) Implementation (Network Architecture):* The model architecture consists of two primary components: user and product embeddings. Each user and product is represented by embeddings of size 50, extracted through embedding layers. These embeddings are reshaped and combined through a dot product layer, producing a prediction for the user-product interaction. The network is compiled with the Adam optimizer and mean squared error loss. The architecture is visualized in Figure 8.

### C. Model 3 - Hybrid Model (Content + Collaborative)

*1) Model Description:* Model 3 combines both content-based and collaborative filtering approaches to enhance recommendation accuracy. The model incorporates matrix factorization for collaborative filtering and a neural network for content-based features. By merging these techniques, the model captures both user-item interactions and item features, resulting in a more robust recommendation system.

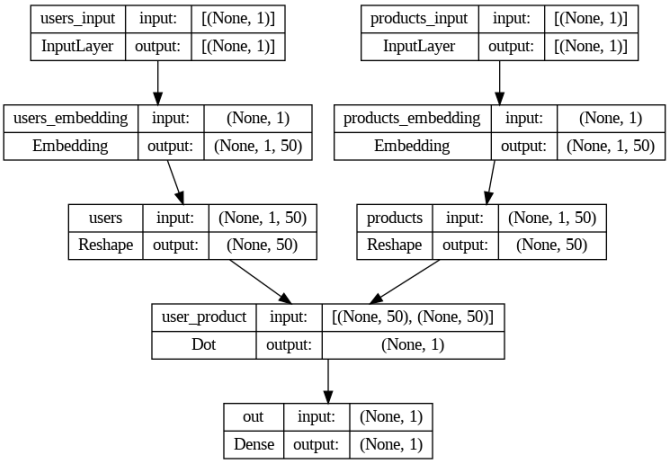



Fig. 8: Model 2 Architecture

*2) Implementation (Network Architecture):* The model architecture consists of two main components: matrix factorization (A) and neural network (B). The matrix factorization component involves embedding layers for users and products, producing user-product interactions. The neural network component incorporates user and product embeddings, concatenates them, and processes them through a dense layer. The final output is a combination of both components, predicting user ratings. The architecture is visualized in Figure 9.

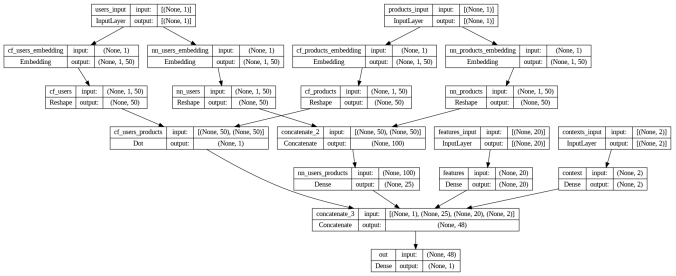


Fig. 9: Model 3 Architecture

### D. Model 4 - Context-Aware Hybrid Model

*1) Model Description:* Model 4 introduces contextual information to enhance recommendation accuracy. The model combines collaborative filtering, content-based, and knowledge-based approaches. Two new features, "Daytime" and "Weekday," are introduced to capture temporal contexts and improve personalized recommendations. For instance, the "Daytime" feature categorizes ratings as daytime (6 AM to 8 PM), considering the user's preference based on the time of day. The "Weekday" feature signals if a rating was given during weekdays (Monday to Friday), providing insights into user behavior related to weekdays.

*2) Context:* The "Daytime" and "Weekday" features play a crucial role in understanding user preferences. For example, users might prefer watching horror movies during the night, impacting the ratings given during that time. Similarly, users

may have different preferences for movies on weekdays compared to weekends, affecting their ratings. These contextual features provide valuable information about the user's viewing habits, allowing the model to make more accurate and personalized recommendations.

*3) Implementation (Network Architecture):* The model architecture consists of collaborative filtering (A), neural network (B), content-based (C), and knowledge-based (D) components. The collaborative filtering and neural network components are similar to the ones used in Model 3. The content-based component processes product features, and the knowledge-based component incorporates contextual information. The final output is a combination of all components, predicting user ratings. The architecture is visualized in Figure 10.

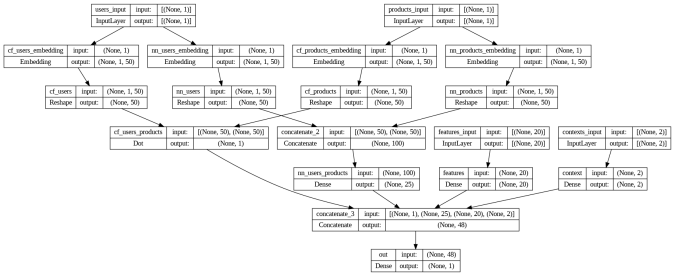



Fig. 10: Model 4 Architecture

## E. Evaluation

Several evaluation metrics used across the models include:

- **Mean Reciprocal Rank (MRR)**: The MRR is computed based on the predicted rankings of products for a user:

$$\text{MRR} = \frac{1}{\text{rank}(p_1)+1} + \frac{1}{\text{rank}(p_2)+1} + \ldots + \frac{1}{\text{rank}(p_k)+1}$$

  where $rank(p_i)$ is the position of the correct movie $p_i$ in the predicted rankings.

- **Root Mean Squared Error (RMSE)**: The RMSE measures the accuracy of predicted ratings compared to actual ratings:

$$\text{RMSE} = \sqrt{\frac{\sum_{u,p}(\text{user}(u,p) - \text{pred}(u,p))^2}{\text{number of non-null entries}}}$$

  where $\text{user}(u,p)$ represents the actual rating given by user $u$ to product $p$, and $\text{pred}(u,p)$ is the predicted rating.

- **Mean Absolute Percentage Error (MAPE)**: MAPE calculates the average percentage difference between predicted and actual ratings:

$$\text{MAPE} = \frac{1}{n}\sum_{u,p}\left|\frac{\text{user}(u,p) - \text{pred}(u,p)}{\text{user}(u,p)}\right|$$

  where $n$ is the number of non-null entries

Models 2,3 and 4 are compiled using the Adam optimizer and mean squared error loss. Training involves fitting the model to the training data, and the training history plots illustrate the learning process. Evaluation metrics such as Root Mean Squared Error (RMSE) and Mean Absolute Percentage Error (MAPE) are reported on the test set. Individual user cases are examined to assess the accuracy of top recommendations and provide insights into the model's performance.

The training plots for Models 2, 3, and 4 are depicted below, illustrating the Mean Squared Error (MSE) and Mean Absolute Percentage Error (MAPE) on both the training and validation sets. The graphs provide insights into the performance of the models as they iteratively learn from the training data.

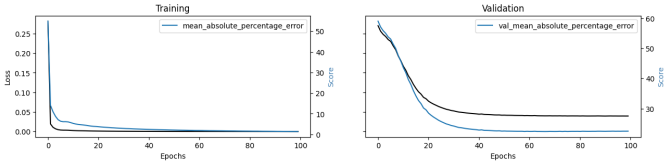



Fig. 11: Model 2 Plot

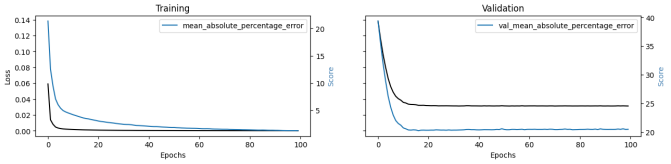



Fig. 12: Model 3 Plot

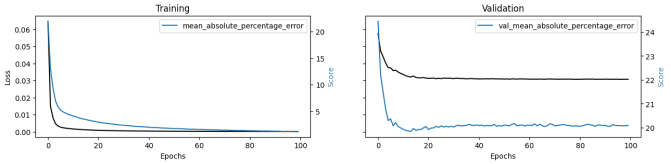



Fig. 13: Model 4 Plot

The observed trend reveals a progressive reduction in training and validation losses when transitioning from Model 2 to Model 3 and finally to Model 4. This decline in losses indicates an improvement in the models' ability to generalize and make accurate predictions. The enhancement in performance can be attributed to the increased complexity and contextual understanding introduced in Models 3 and 4, incorporating additional features and contextual information, leading to more refined and effective recommendations.

The dataset was split into training and testing sets with an 80-20 ratio. Subsequently, the Root Mean Squared Error (RMSE) loss was computed for various models on the test dataset.

1) **Model 1** - Simple Matrix Factorization : **0.22361**
2) **Model 2** - Matrix Factorization + Embeddings : **0.21525**
3) **Model 3** - Hybrid Model : **0.20959**
4) **Model 4** - Context-Aware Hybrid Model : **0.20476**

The progression from Model 1 to Model 4 reveals a consistent reduction in RMSE loss, reaching its minimum in Model 4.

We also conducted an individual user analysis by leveraging the trained Models 1, 2, 3, and 4 to recommend the top 5 movies for a particular user. Subsequently, we compared these recommendations against the top 5 movies from the corresponding user's test dataset. The evaluation metrics, including Mean Reciprocal Rank (MRR), accuracy, and Root Mean Squared Error (RMSE), were calculated based on this comparative analysis. This process provides insights into the models' effectiveness in generating personalized and accurate recommendations for individual users.

The results for User 1 are depicted in the following figures.

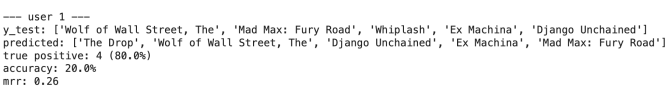



Fig. 14: Model 1 - Recommendations

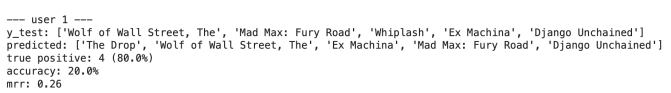



Fig. 15: Model 2 - Recommendations


Fig. 16: Model 3 - Recommendations

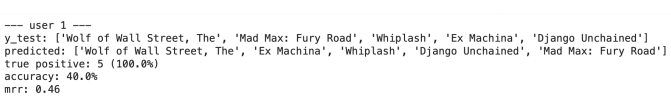



Fig. 17: Model 4 - Recommendations

Observing the transition from Model 1 to Model 4, notable trends emerge:

1) **MRR Trend** : The MRR improves across models, with Models 3 and 4 outperforming, indicating enhanced precision in ranking relevant movies due to the incorporation of contextual information and additional features.
2) **Accuracy Trend** : The trend in accuracy reveals an upward trajectory, notably in Model 4, indicating enhanced precision in predicting the exact movies present in the user's test dataset. It's crucial to note that Model 3, despite exhibiting zero accuracy, showcases a higher Mean Reciprocal Rank (MRR) compared to the previous two models. MRR, being a more informative metric, suggests that Model 3 provides more accurate predictions in terms of the ranking of recommended movies, even if the exact order is not replicated.
3) **True Positives Trend** : True positives increase consistently, with Model 4 achieving the highest count, showcasing the effectiveness of incorporating contextual features in providing personalized recommendations and leading to more successful movie recommendations from the user's test data-set.

## X. Summary

We have thus been able to use a number of different models for the same task with varying performances.

First, we start with the basic model of content based recommendation used cosine similarity. We obtain good results, but we note that these are inherently incomplete recommendations as they do not use any of the user interactions to make predictions. So, we switch to a collaborative filtering based approach where we try the technique of Singular Value Decomposition.

This too, was a standard model that did not involve anything much innovative and did not test as well. Hence, our team decided to try the technique of KNN.

Using KNN with multiple distance metrics allowed us to build a newer model that gave better quality predictions, as can be seen by the rmse loss. However, this too, was just another standard ML technique and our group wanted to try some innovative techniques. This lead to the next two algorithms.

First, we discovered The Netflix Prize and BellKor's Pragmatic Chaos. We read the paper and found it interesting and hence, decided to try our hand at implementing the algorithm in code.

Once this was done, we still did not achieve amazing test results - partly due to the fact that even this model used simple, conventional ML techniques without any deep learning and also due to the fact that it is an outdated algorithm.

This lead us to an attempt at using neural networks to try and find recommendations. We thus implemented a series of more modern, NN-enabled models that turned out to provide much better rmse losses.

This concludes our overview of recommendation systems.

## XI. References

1) Cosine Similarity and SVD
2) Collaborative Filtering using KNN
3) Bellkor's Pragmatic Chaos - 1
4) Bellkor's Pragmatic Chaos - 2
5) Collaborative Filtering using Neural Networks