Movie Recommendatin System

Chirag Sarda
IIT Gandhinagar, India
20110047
bhavesh.jain@iitgn.ac.in

Bhavesh Jain
IIT Gandhinagar, India
20110038
bhavesh.jain@iitgn.ac.in

Hitesh jain
IIT Gandhinagar, India
20110077
hitesh.jain@iitgn.ac.in

Abstract

The movie recommendation system is a software application designed to suggest movies to users based on their preferences. The system uses machine learning algorithms to analyze a user's viewing history and behaviour and then recommends movies that are likely to be of interest to them. A movie recommendation system is required because it can help users discover new movies that they may not have otherwise found. With the vast amount of movies available today, it can be overwhelming for users to decide what to watch next. By using a movie recommendation system, users can receive personalized recommendations that are tailored to their individual preferences and viewing history. From a business perspective, by providing personalized recommendations, these business can increase user engagement and retention, which can ultimately lead to increased revenue and profitability.

Github link

2012 ACM Subject Classification CS328 Datascience

Keywords and phrases Movie Recommendation, IMDB, Content Based, Collaborative,Deep Neural Network

...23

Acknowledgements We want to thank Prof. Anirban Dasgupta, Computer Science and Engineering, IIT Gandhinagar for his help and support throughout the project

1 Introduction

Our project is a comprehensive movie recommendation system that utilizes four different types of recommendation algorithms to provide users with personalized movie recommendations. These four types of recommendation systems are:

1. Top-N recommendation
2. Content-based recommendation
3. Collaborative recommendation
4. Hybrid-based recommendation

The Top-N approach involves recommending a set of top movies to a new user who has not yet provided any information or preferences to the recommendation system. Top-N recommendation is a simple and effective approach that is often used as a baseline for evaluating recommendation systems.

The content-based recommendation system, on the other hand, uses the features of a movie, such as its plot, genre, and keywords, to make recommendations. This system is particularly useful for users who are looking for movies that are similar to ones they have already enjoyed.

The collaborative recommendation system uses the behavior and preferences of similar users to make recommendations. This system is particularly useful for users who are looking for recommendations based on the preferences of other users with similar tastes.

Finally, the hybrid recommendation system combines two or more of the above recommendation systems to provide more accurate and personalized recommendations.

By using all four types of recommendation systems, our project aims to provide the most comprehensive and accurate movie recommendations possible, tailored to each individual user's preferences and viewing history. This will enhance the overall movie watching experience for users, and provide valuable insights for businesses looking to improve user engagement and satisfaction.

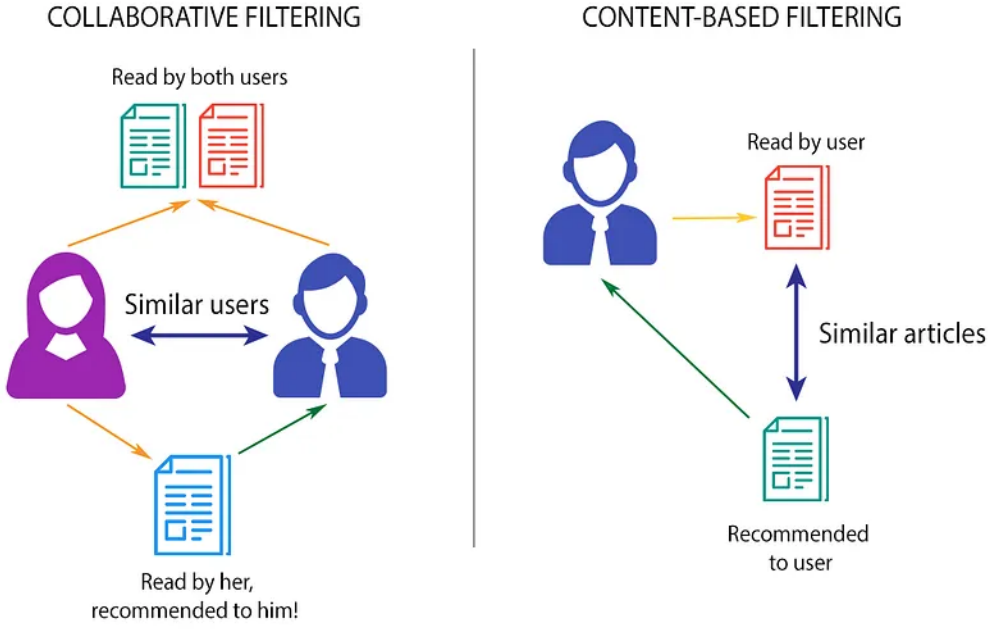

Figure 1 Collaborative and Content-Based Filtering Source

2 Methodology

2.1 Dataset

In this project of building a movie recommendation system, we have used Movie Lens 100K dataset , which comprises 95,490 ratings provided by 610 users for 9,082 different movies. The dataset includes two CSV files: "Ratings.csv," which contains information on the rating provided by each user for different movies, and "Movies data.csv," which provides comprehensive information on the 9,082 movies. The movie data file includes 30 columns with various details such as cast and crew information, genres, release dates, and revenue figures, as well as unique identifiers such as IMDB ID, TMDB ID, and movieId. This extensive

information allowed us to conduct a detailed analysis and build an effective recommendation system that accurately suggests movies based on user preferences.

In the process of cleaning and analyzing the data, we encountered some missing values in the movie data file, which we visualized using a pie chart. To develop an accurate and effective recommendation system, we chose to exclude columns with over 10% missing values while utilizing a significant amount of the available data.

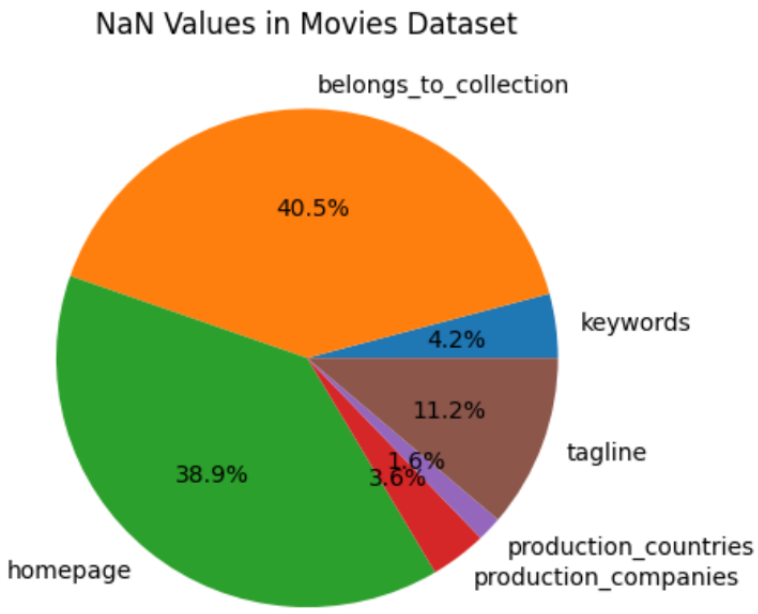

Figure 2 Pie chart showing the percentage of Nan values in each column

2.2 Data Visualization and Manipulation

2.2.1 Bar plot of rating frequency

A bar plot of shown in Figure 3 rating frequency revealed that different users have varying rating patterns and preferences. This insight is crucial to keep in mind during the development of our recommendation system. Additionally, we observed that most users rated the movies they watched with 4 stars, followed by 3 stars and 5 stars.

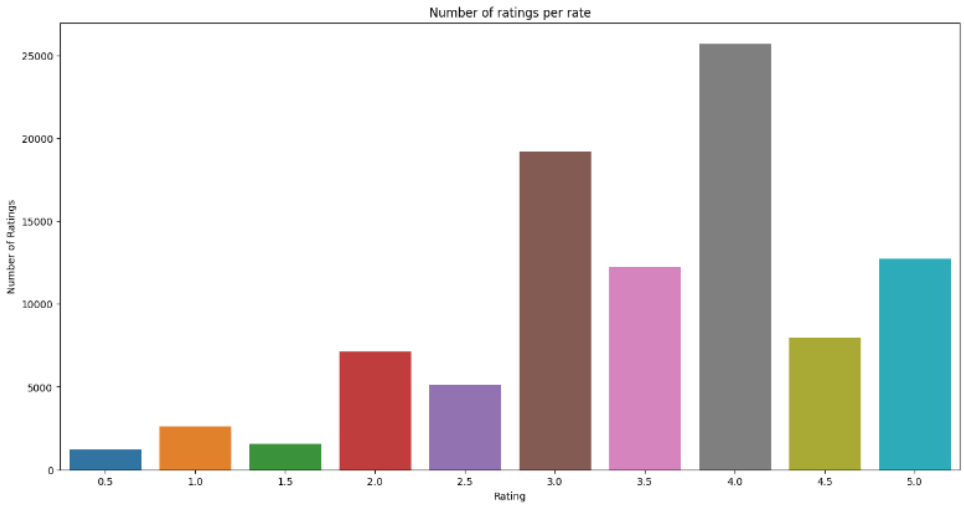

Figure 3 Frequency of Ratings

2.2.2 Bar plot of genre-based movie counts

Figure 4 reveales that movies belonging to the drama genre were the most commonly present in the dataset, followed by comedy, thriller, romance and action genres. It also highlighted that the movies in the dataset belonged to one or multiple genres out of the 20 unique genres present.

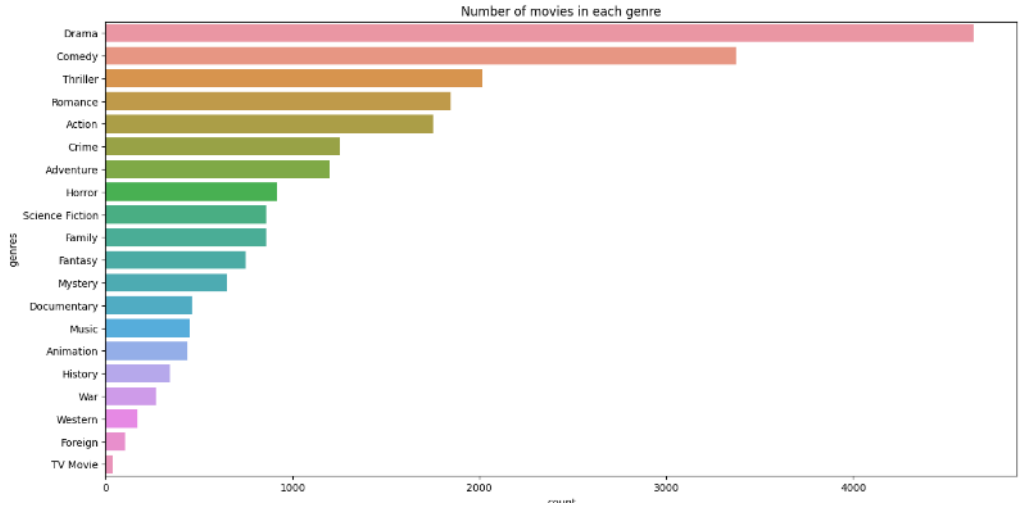

Figure 4 bar plot of genre based movie counts

2.2.3 Production of Movies in Differnt countries

The pie chart in figure 5 analysis of the top movie production countries revealed that the majority of movies in the Movie Lens 10K dataset were produced in the USA and other foreign countries. However, we cannot utilize this column information while developing our recommendation system as there is no uniform distribution of movies across all countries.

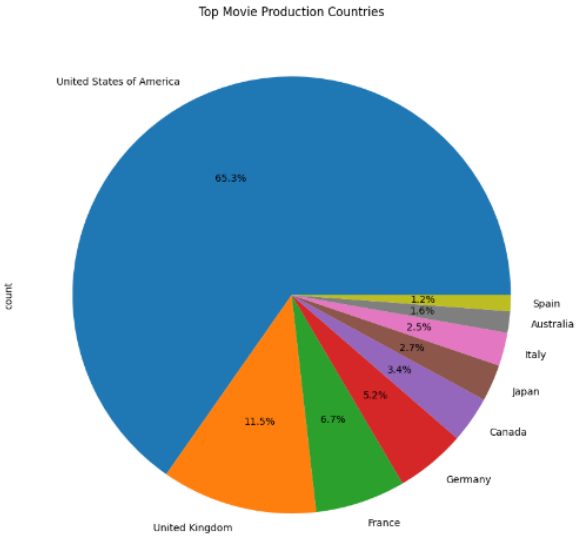

Figure 5 Pie chart showing top movie production countries

2.2.4 Distribution of ratings overs movies

The distribution of the number of ratings per movie as shown in fingure 6 follows the long-tail property, a common occurrence in real-world scenarios. This property implies that a small number of popular movies receive a large number of ratings, while the majority of movies receive only a few ratings. To address this issue, one solution that we have used in our recommendation system is matrix factorization techniques, which involve training a model to learn user-item interactions by capturing user information in user latent factors and item information in item latent factors. This approach can significantly reduce dimensionality and sparsity, making the recommendation system more scalable and accurate.

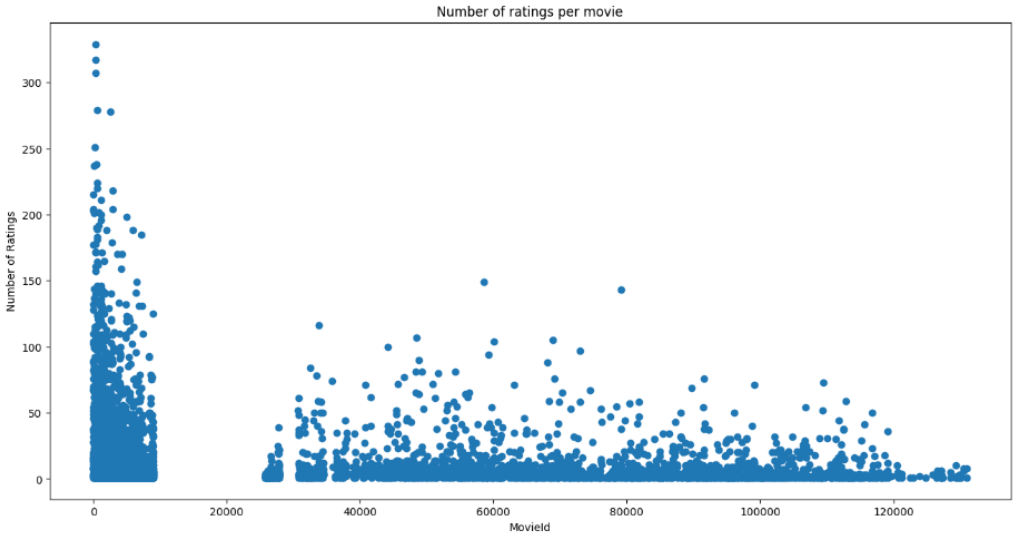

Figure 6 Plot of number of ratings per movie

3 Different Models

3.0.1 Top-N Recommendation

Directly using the average ratings of movies to generate Top-N recommendations may not work because it does not take into account the number of votes each movie has received. Movies with a small number of ratings may have a higher average rating by chance or because only a small number of people with a particular taste have rated the movie. In contrast, movies with a larger number of ratings are likely to have a more stable and accurate average rating, and are therefore more likely to be of higher quality or popularity.

$$\text{IMDb average} = \frac{v \times R + m \times C}{v + m}$$

Source

where:

- v = average for the movie (mean rating)
- R = number of votes for the movie
- m = minimum number of votes to be listed in the top movies. (we have taken the 75th percentile of vote counts)
- C = mean vote across the whole report

To address this issue, the IMDB weighted rating formula can be used for generating Top-N recommendations. The formula takes into account both the average rating and the number of votes a movie has received, with a higher weight given to movies with a larger number of votes. This ensures that movies with a larger number of votes are more likely to be ranked higher than those with a smaller number of votes, and that the overall rankings are more accurate and reliable.

The IMDB weighted rating formula can be seen as a specific implementation of Bayesian averaging that is tailored for ranking movies based on their ratings and vote counts.

In addition, the formula also includes a parameter m, which is the minimum number of votes a movie must have to be included in the Top-N recommendations. This helps to filter out movies with very few ratings or those that are not popular enough to be of interest to most users.

Top 10 movie recommendation based on the weighted average are:

```
movies_data_modified.head(10)
```

	movieId	title	adult	genres	vote_average	vote_count	year	popularity	weighted_average
284	318	The Shawshank Redemption	False	Drama|Crime	8.5	8358.0	1994	51.645403	8.410370
692	858	The Godfather	False	Drama|Crime	8.5	6024.0	1972	41.109264	8.377631
6896	58559	The Dark Knight	False	Drama|Action|Crime|Thriller	8.3	12269.0	2008	123.167259	8.243904
2369	2959	Fight Club	False	Drama	8.3	9678.0	1999	63.869599	8.229433
266	296	Pulp Fiction	False	Thriller|Crime	8.3	8670.0	1994	140.950236	8.221561
472	527	Schindler's List	False	Drama|History|War	8.3	4436.0	1993	41.725123	8.152396
8585	112552	Whiplash	False	Drama	8.3	4376.0	2014	64.299990	8.150528
4222	5618	Spirited Away	False	Fantasy|Adventure|Animation|Family	8.3	3968.0	2001	41.048867	8.136456
1835	2324	Life Is Beautiful	False	Comedy|Drama	8.3	3643.0	1997	39.394970	8.123197
321	356	Forrest Gump	False	Comedy|Drama|Romance	8.2	8147.0	1994	48.307194	8.121038

Figure 7 Top 10 movies when IMDB average formula applied

In addition to using the IMDB weighted rating formula, we also incorporated the popularity parameter of each movie into our recommendation system. We gave equal weight to both the weighted average and popularity to calculate a final score for each movie. This approach ensures that popular movies with a high number of votes and ratings are given appropriate consideration in our recommendations, while also ensuring that the quality of the movies is taken into account through the weighted rating formula. Based on this final score, we recommend the top N movies to the users. Since the weighted average is between the 0 to 10 so we are normalizing the popularity between the 0 to 10.
Top 10 movie recommendation after incorporating the popularity factor.

```
movies_data_modified.head(10)
```

	movieId	title	adult	genres	vote_average	vote_count	year	popularity	weighted_average	score
8889	135887	Minions	False	Family|Animation|Adventure|Comedy	6.4	4729.0	2015	4.843810	6.397285	5.620547
8664	115617	Big Hero 6	False	Adventure|Family|Animation|Action|Comedy	7.8	6289.0	2014	1.892001	7.720976	4.806489
266	296	Pulp Fiction	False	Thriller|Crime	8.3	8670.0	1994	1.247033	8.221561	4.734297
6896	58559	The Dark Knight	False	Drama|Action|Crime|Thriller	8.3	12269.0	2008	1.089701	8.243904	4.666803
8586	112556	Gone Girl	False	Mystery|Thriller|Drama	7.9	6023.0	2014	1.369576	7.811961	4.590768
8752	122904	Deadpool	False	Action|Adventure|Comedy	7.4	11444.0	2016	1.662064	7.367858	4.514961
284	318	The Shawshank Redemption	False	Drama|Crime	8.5	8358.0	1994	0.456924	8.410370	4.433647
7389	72998	Avatar	False	Action|Adventure|Fantasy|Science Fiction	7.2	12114.0	2009	1.637383	7.175446	4.406414
2369	2959	Fight Club	False	Drama	8.3	9678.0	1999	0.565075	8.229433	4.397254
692	858	The Godfather	False	Drama|Crime	8.5	6024.0	1972	0.363707	8.377631	4.370669

Figure 8 Top 10 movie recommendation after incorporating the popularity factor

3.0.2 Content Based Recommendation

Content-based recommendation is a type of recommendation system that recommends items to users based on their preferences and interests. In this approach, the system uses information about the items (such as their genres, actors, directors, and plot summaries) to create a profile of each movie item. Then, the system creates a profile for each user based on their previous interactions with the items. The system then recommends items to the user that match their preferences, based on the similarity between the user profile and the item profiles.

In this approach, we create a user-movie matrix that contains the ratings given by the user to each movie. The matrix is of size (number of users x number of movies), where the un-rated movies are marked as zero. To implement content-based recommendation, we first randomly initialize a vector for each user. We then represent each movie as a vector using different content-based features, such as genre, director, cast, keywords, and plot summary. Each feature represents a different aspect of the movie, and when combined, they form a unique vector that represents the movie. The user vectors and the movie vectors are multiplied, and the resulting scores are used to predict the ratings for each user-movie pair.

3.0.2.1 Objective Function

Problem formulation $r(i,j) = 1$ if user j has rated movie i (0 otherwise)
$y^{(i,j)}$ = rating by user j on movie i (if defined)
$\theta^{(j)}$ = parameter vector for user j
$x^{(i)}$ = feature vector for movie i For user j, movie i, predicted rating: $\left(\theta^{(j)}\right)^T \left(x^{(i)}\right)$
$m^{(j)}$ = no. of movies rated by user j

Optimization objective: To learn $\theta^{(j)}$ (parameter for user j):

$$\min_{\theta^{(j)}} \frac{1}{2} \sum_{i:r(i,j)=1} \left(\left(\theta^{(j)} \right)^T x^{(i)} - y^{(i,j)} \right)^2 .$$

To learn $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(n_u)}$:

$$\min_{\theta^{(1)},\ldots,\theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left(\left(\theta^{(j)} \right)^T x^{(i)} - y^{(i,j)} \right)^2$$

References

Our goal is to minimize the above objective function to improve the accuracy of the recommendations. We use the Adam optimizer, which is an extended version of a stochastic gradient descent optimization algorithm with momentum, to optimize the parameters of our model.

To ensure that the predicted ratings fall between 0 and 5, we use the sigmoid function, which maps any real number to a value between 0 and 1. We then multiply the output of the sigmoid function by 5 to obtain the predicted rating for each user-movie pair.

To determine which content-based features to use for representing each movie as a vector, we conducted experiments to compare the performance of different features. Specifically, we plotted the training and validation loss at each step of the training process for different combinations of features.

3.0.2.2 One hot encoded genre

To ensure that our content-based recommendation system is unbiased and inclusive of all genres, we have one-hot encoded the genre feature for our dataset of 9082 unique movies. This technique allows us to convert categorical data, such as movie genres, into a numerical format that can be easily used in machine learning algorithms.(See Figure 9)

3.0.2.3 Director + Cast + Production company

To enhance our content-based recommendation system, we explored the use of three features - Director, Cast, and Production Company - and employed *TfidfVectorizer* to vectorize them. This technique converts raw documents into numerical matrices by assigning significance to each term. We also removed common English words and set minimum/maximum document frequency parameters to optimize the process.(See Figure 10)

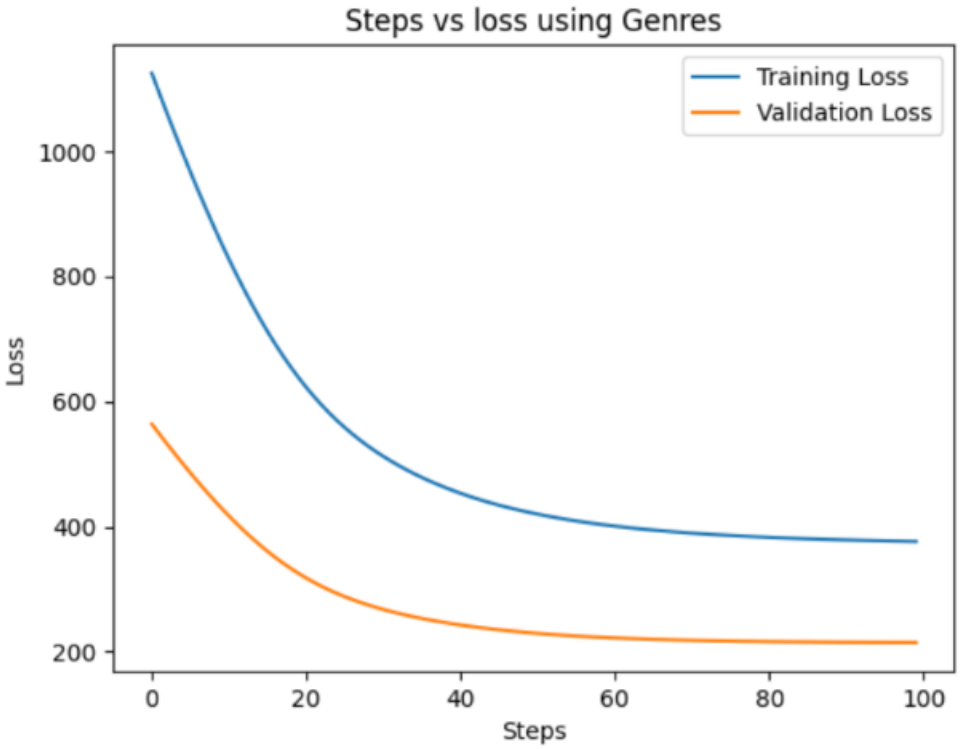

Figure 9 Variation of Loss with step when genre is one hot encoded

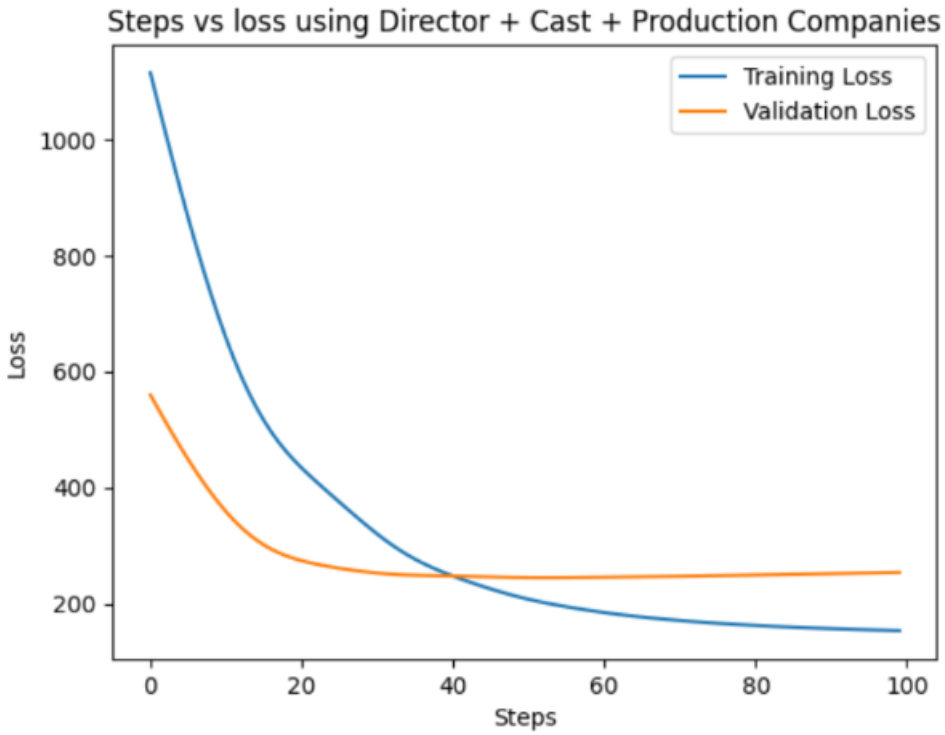

Figure 10 Variation of Loss with step when Director, Cast and Production Company is one hot encoded

3.0.2.4 Director + Cast + Production Companies + Keywords +Genres

We also explored the use of five features - Director, Cast, Production Companies, Keywords, and Genres - and utilized *TfidfVectorizer* to vectorize them. (See Figure 11)

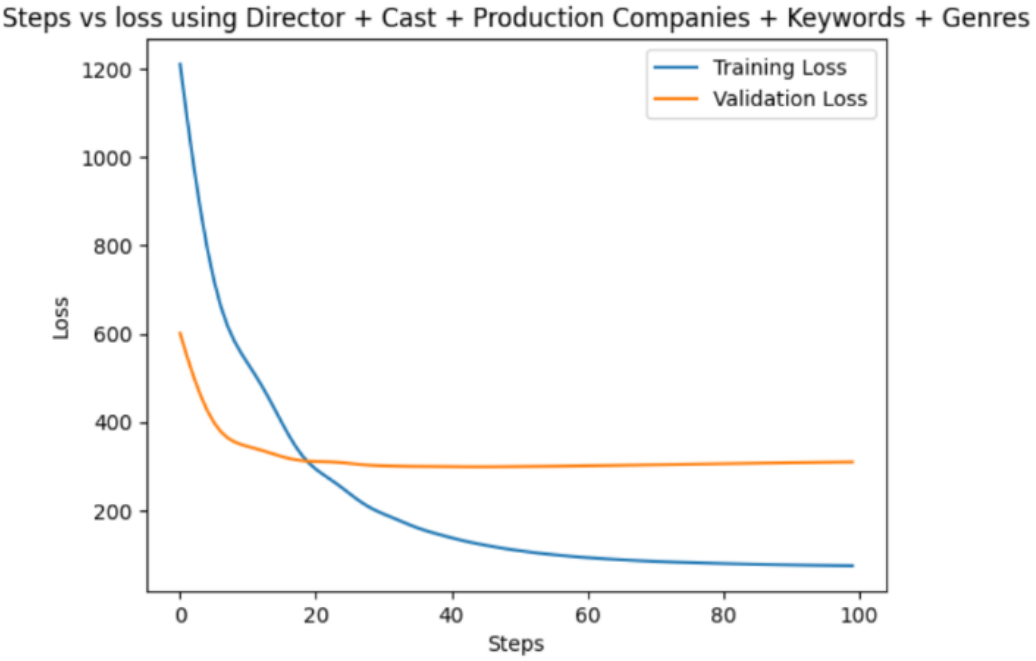

Figure 11 Variation of Loss with step when Director, Cast, Production Company and genre is one hot encoded

To provide recommendations to users on the top 10 movies, we sorted the ratings vector for a particular user, removed the ratings of the movies that the user has already seen, and returned the top 10 ratings predicted by the model. Below is an example output of the top 10 movie recommendations for User 1. (See Figure 12)

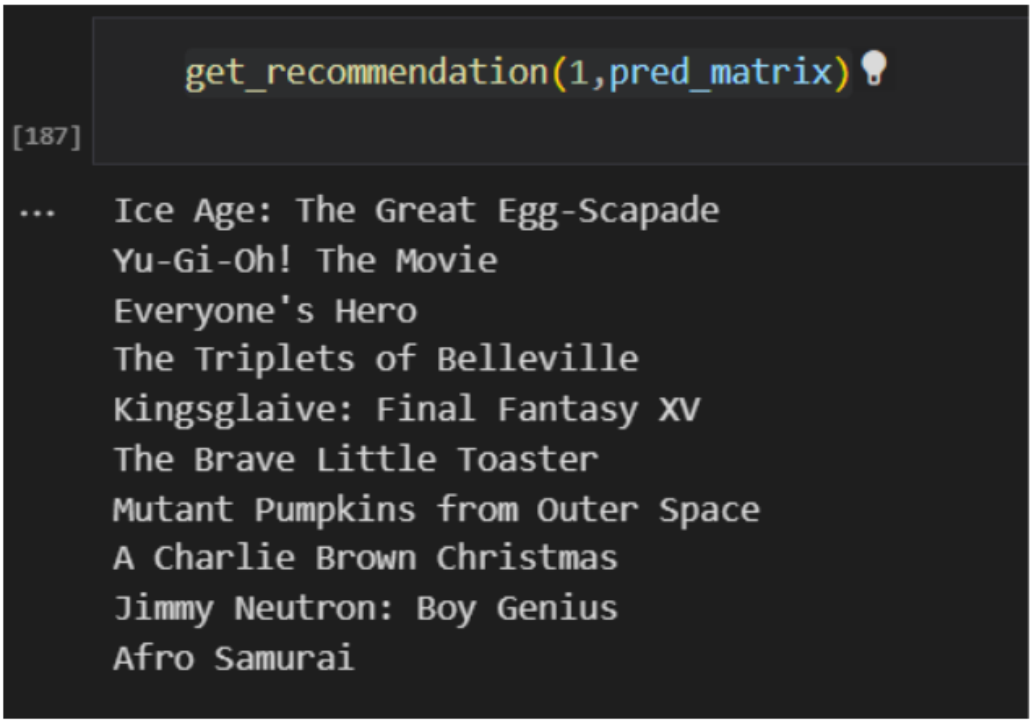

Figure 12 Content Based recommendation

3.0.2.5 Results

The final plot showing validation loss for different content at each step is shown in figure 13:

Table 1 shows the RMSE scores for the content-based movie recommendation using different types of content: Genres had the lowest RMSE score, followed by using three columns and then using five columns.

Content Used	RMSE Score
One Hot encoded Genre	1.5351
Director + Cast + Production Companies	1.5652
Director + Cast + Production Companies + Keywords + Genres	1.6008

Table 1 RMSE scores for different content-based recommendation models

3.0.3 Collaborative Based Recommendation

In collaborative filtering recommendation of a particular user are calculated using the data of all the users. It is also based on the idea that similar people are likely to have similar interests. The collaborative model can be thought of as multiple users are collaborating(by giving data) to create a recommendation system. With addition of each user, the recommendation system is improved as it helps us to find more similarity and generalize better. The more the users, the better the recommendation system. It broadly uses two approaches, memory based and model based. Memory based approach learns all the data given by all the users and then makes recommendations for a particular user. Model based approach uses the data

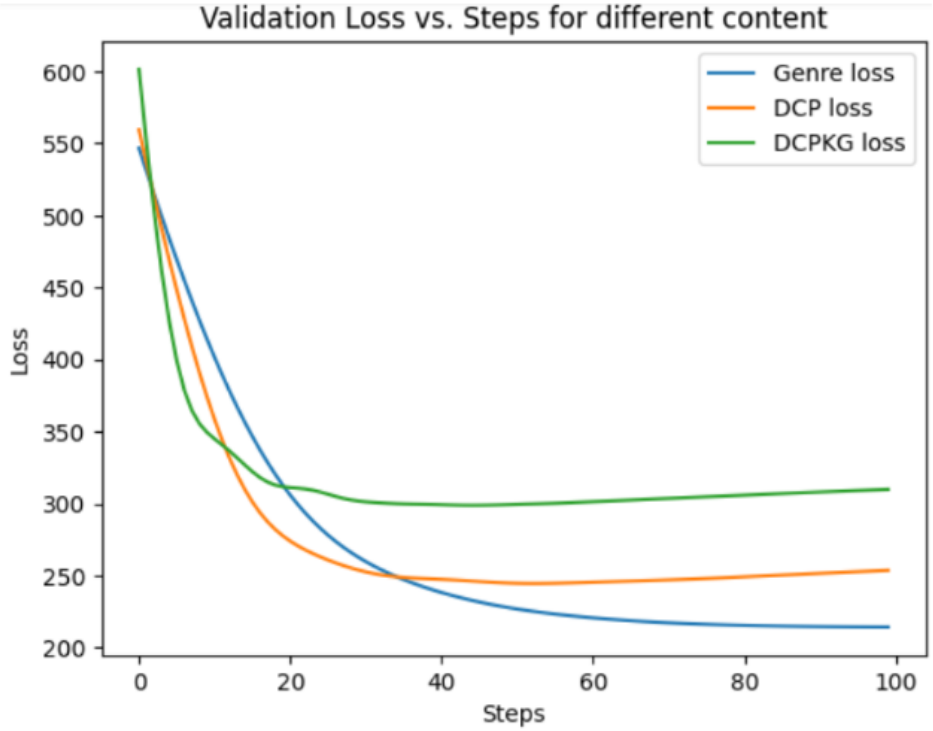

Figure 13 Comparing all the content based recommendations

to train a model and then uses the model to make recommendation. We have used both the approaches in our project. For memory based approach we have used KNN and for model based approach we have used matrix factorisation. Due to limits on computational power, we have used a small dataset. In general we believe that a larger dataset would have produced better results as it would have helped us to generalize better. We also compared between the similarity coefficients used in KNN.

3.0.3.1 Memory Based Approach

In a memory based approach we learn the complete data and then make recommendations for a particular user. KNN is a lazy algorithm. It has no "training time". For each user, we find the K closest users and then use aggregate functions to calculate the rating. In our program we first look for 10 closest neighbors to predict the ratings for a particular user. If all the 10 users have not watched the movie we take the next 10 users. Still if the movie is not watched by any of the person we use the mean rating of the movie over the whole dataset. For the neighbors which have seen the movie, we take the weighted average over similarity to predict the rating. The similarity becomes an important factor for the prediction. We have used two similarity coefficients, cosine similarity and pearson correlation coefficient. If we are getting a good similarity score between users of different interests then the recommendations won't be good.

3.0.3.2 Model Based Approach

In model based approaches we use the data to train a model and then use the model to make recommendations. We have used matrix factorisation for our project. In this approach we factorize the original user-rating matrix into two matrices. Let's say the number of users is N and the number of movies is M. We factorize the original matrix of NM into two matrices of $N \times K$ and $K \times M$. Here K can be thought of as the number of features. The entry in the first matrix for each user depicts the weight or the importance of each feature for the user. Similarly each column specifies the value/inheritance of K features for a particular movie. We multiply the two vectors to predict the rating for a particular user and movie. We use gradient descent to minimize the error between the predicted and the actual rating. As the original matrix is a sparse matrix, we compute loss only over the given entries.

3.0.3.3 KNN

With cosine similarity

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} \tag{1}$$

We first applied cosine similarity to get the nearest neighbors. Now we selected a particular user and predicted ratings for random 30 movies which were already rated by the user. We also took a weighted average to predict the rating.(See Figure 14)

movieid	predicted	actual	difference
1198	4.707585	4.0	7.075849e-01
1210	4.639760	4.0	6.397596e-01
1127	3.484727	4.0	5.152730e-01
1036	4.133043	3.0	1.133043e+00
2539	3.000000	4.0	1.000000e+00
1197	4.671018	4.0	6.710181e-01
1200	4.295675	4.0	2.956754e-01
344	3.000000	3.0	4.440892e-16
1101	3.052675	2.0	1.052675e+00
1089	4.511062	3.0	1.511062e+00

Figure 14 Difference Between Predicted and Actual Ratings

The absolute sum of differences between the predicted and the actual ratings for 30 movies came out to be = **13.377737404488002** or **0.4459245801496** per movie.

Now we have increased the dataset from 671 to approximately 30000 users. We hoped for a huge increase in performance. However the gain was limited. (See Figure 15)

movieid	predicted	actual	difference
296	4.198935	5.0	0.801065
380	3.397699	5.0	1.602301
349	3.896147	5.0	1.103853
595	3.796766	5.0	1.203234
344	2.795462	4.0	1.204538
588	3.796508	5.0	1.203492
590	3.689217	3.0	0.689217
153	3.699144	3.0	0.699144
457	4.115804	4.0	0.115804
592	3.696079	4.0	0.303921

Figure 15 Difference Between Predicted and Actual Ratings

The absolute sum of differences between the predicted and the actual ratings for 30 movies came out to be = **9.43769428872524** or **0.314589809624175** per movie.

Although the value improved, it was still significant. This shows that cosine similarity is giving high similarity scores even for not so similar users. One of the major reasons for this is that it does not consider the magnitude of the rating. If User 1 has given a rating of 5 to movie M and User 2 has given 1, their similarity score should be decreased, however cosine would only increase it.

With Pearson's Coefficient

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{2}$$

Now we calculated the nearest neighbors using Pearson's Coefficient. Using this the behavior of other users during rating is also accounted for. A person may be lenient in giving ratings, while the other might be reluctant to give high scores, therefore this is taken care of by subtracting the mean of each user's rating in the coefficient. Moreover, if both the users have given ratings on opposite sides of the scale(say one has given 1 while the other has given 5) the similarity score would reduce. Therefore, we expected much better results with Pearson's inequality even with a small dataset. (See Figure 16)

1611	5.000000	5.0	0.000000
1423	4.000000	4.0	0.000000
2907	2.000000	2.0	0.000000
50	4.857143	5.0	0.142857
1719	5.000000	5.0	0.000000
2539	4.000000	4.0	0.000000
1459	3.000000	3.0	0.000000
1036	3.000000	3.0	0.000000
1240	3.500000	4.0	0.500000
152	4.000000	4.0	0.000000

Figure 16 Difference Between Predicted and Actual Ratings

The absolute sum of differences between the predicted and the actual ratings for 30 movies came out to be = **1.1428571428571432** or **0.038095238095238** per movie which is significantly lower.

3.0.3.4 Matrix Factorisation

In matrix factorisation we divided the dataset into test and train. The train set consisted of 80% of the sample. For the remaining 20% entries, we predicted the rating using the trained factor matrices and computed the loss. We calculated the loss as the Frobenius norm of the test and train user-rating matrices. We tried for multiple values of K to see the decrease in loss. Due to limited data, for large values of K we were not able to generalize the features well so we compared for only $K = 2$ and 4. In general we believe that the loss would decrease by increasing the number of users as the features would be learnt better.

$$J\left(x^{(1)},\ldots,x^{(n_m)},\theta^{(1)},\ldots,\theta^{(n_u)}\right) = \frac{1}{2}\sum_{(i,j):r(i,j)=1}\left(\left(\theta^{(j)}\right)^T x^{(i)} - y^{(i,j)}\right)^2.$$

$$\min_{\substack{x^{(1)},\ldots,x^{(n_m)}\\ \theta^{(1)},\ldots,\theta^{(n_u)}}} J\left(x^{(1)},\ldots,x^{(n_m)},\theta^{(1)},\ldots,\theta^{(n_u)}\right)$$

This is the loss function that is used. We use Adam Optimizer to train the model.

From the figure 17 we see that for a higher value of K the test loss is lesser initially, but then the model begins to overfit and the validation loss is increased. In general due to less computation power, we trained on limited data. If we trained on more data, the features would be learnt better.

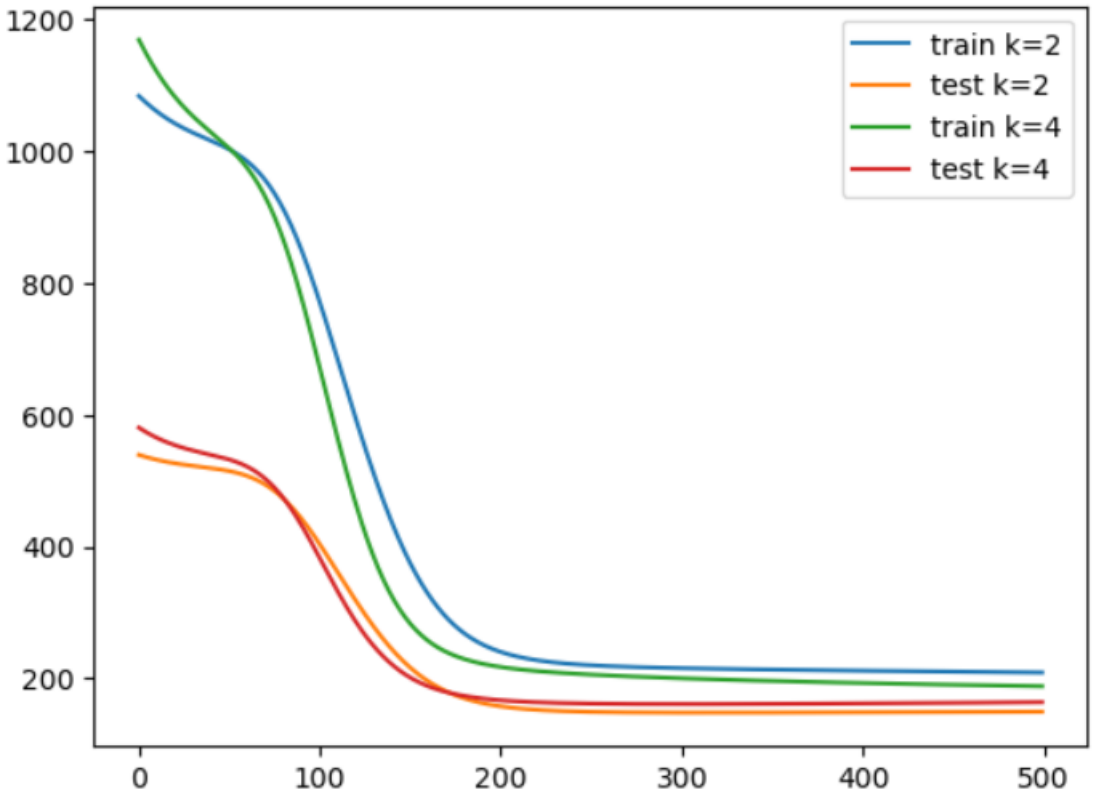

Figure 17 Total test and validation losses for k=2 and k=4

3.0.4 Recommendation using Deep Neural Network

3.0.4.1 Model Architecture

The model architecture consists of two main parts: user input and movie input, which are concatenated together before the final output. The user input consists of four features: userId, gender, age, and occupationId. The userId is first fed through an embedding layer with a size of 16. The gender, age, and occupationId are all categorical features, so they are one-hot encoded and fed through three separate dense layers with activation function 'relu'. The output of each dense layer is then concatenated with the userId embedding output using the Keras Concatenate layer.

The concatenated output is then passed through two dense layers with 'relu' activation, batch normalization, and dropout layers to prevent overfitting.

The movie input consists of five features: movieId, genre, director, cast, and production companies. The movieId is first fed through an embedding layer with a size of 16. The genre, director, cast, and production companies are all 20 length vector (cast, director and production companies are vectorised in terms of the genre), so they are fed through four separate dense layers with 'relu' activation. The output of each dense layer is then concatenated with the movieId embedding output using the Keras Concatenate layer.

The concatenated output is then passed through two dense layers with 'relu' activation, batch normalization, and dropout layers to prevent overfitting.

The final output is the concatenated output from the user and movie inputs. The concatenated output is then passed through a dense layer with 'softmax' activation to

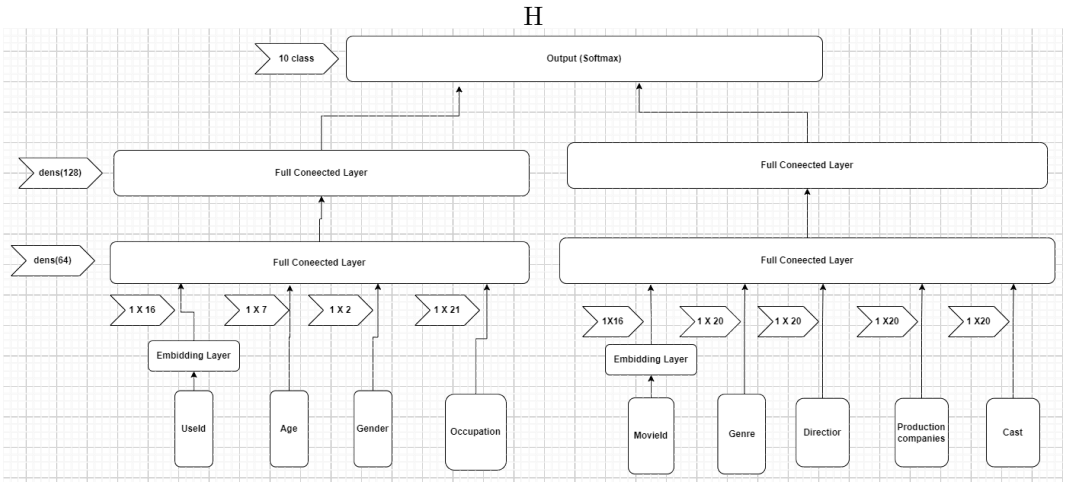

Figure 18 Neural Network Architecture

produce the rating score for the corresponding user-movie pair.

The model is trained using the Keras fit method, with mean absolute error (MAE) as the loss function and Adam optimizer with a learning rate of 0.01. The training is run for 50 epochs with a batch size of 1000. We have take the MAE not accuracy because if a movie has a 4.5 rating and we predict rating of 4, our prediction is good, while accuracy will not cosider the close ness of our solution.

The model is then tested using the Keras evaluate method with the test set. The MAE on the test set is reported.

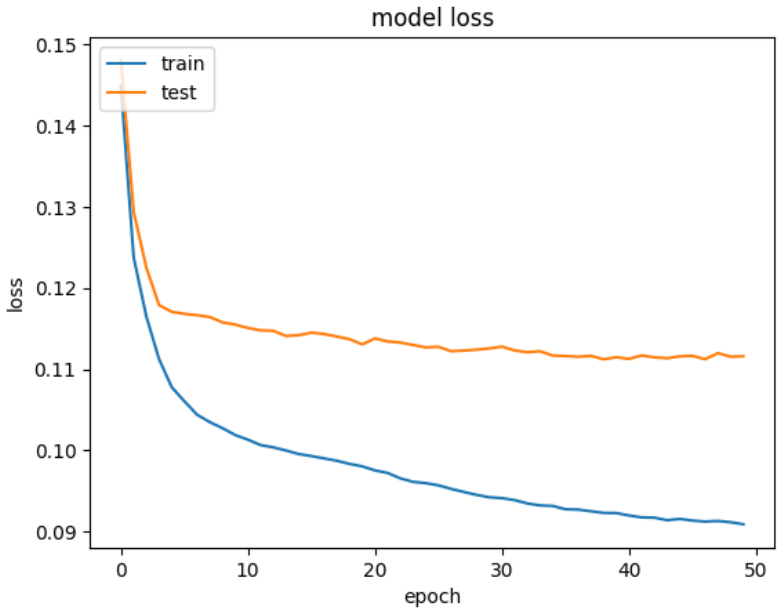

Figure 19 Loss with epochs

4 Result and Conclusion

1. Based on the graphs generated in figure 9, 9, and 9, it can be concluded that using one-hot encoded genres as a feature to vectorize each movie is the most effective approach for this dataset. For the other four features (Director, Cast, and Production Company, and Keywords), the training loss initially goes much lower than that of one-hot encoded genres. However, while checking the validation loss, we can see that it becomes constant after a small number of steps. This implies that we cannot use a larger number of features as the dataset is small and will become overfitted. In other words, the model will perform well on the training data but poorly on new, unseen data. Therefore, using one-hot encoded genres is the most effective approach to vectorize each movie for this dataset.
2. In collaborative recommendation systems, we tried two methods. First we solved using KNN. In KNN we used cosine similarity and Pearson's coefficient and compared the result. We saw that in this case, because of the nature of the problem, Pearson Coefficient gave a much better result. Matrix Factorisation was also used to predict the movie ratings. We compared the results between different parameters for matrix factorization.
3. Matrix factorization assumes a linear relationship between users and items, which limits its ability to capture non-linear patterns and complex relationships between users and items. DNN-based models, on the other hand, can capture non-linear patterns and learn complex relationships between users and items, which can improve the accuracy of recommendations
4. Matrix factorization can be computationally expensive and slow for large-scale datasets. DNN-based models can handle large-scale datasets more efficiently, making them more scalable.

.

5 References

1. https://github.com/Lei-Xu/Movie-Recommendation-System
2. A Deep Neural Network Approach for Movie Recommendation
3. https://www.youtube.com/watch?v=giIXNoiqO$_U list = PL-6SiIrhTAi6x4Oq28s7yy94ubLzVXabj$
4. https://github.com/rposhala/Recommender-System-on-MovieLens-dataset
5. https://nipunbatra.github.io/ml2023/notebooks/posts/movie-recommendation-knn-mf.html