Movie Recommendation System

Bhanu Prakash Guntupalli, Anurag Lingala, Bhanu Prakash Reddy Gaddam

ABSTRACT

There are already too many video streaming sites, and a user may wish to watch movies that are comparable to ones they have already seen or enjoyed. They would also want to avoid films that are similar to ones they despised. It's difficult to recommend relevant movies to users: how can an algorithm tell if two films are similar? What is the standard deviation of similarity? It's also important from a business standpoint: you want the user to stay on the streaming platform for as long as possible. Using the collaborative filtering algorithm, we can create a recommendation system that proposes movies based on the user's viewing history and ratings.The aim of this project is to create a Movie Recommendation System based on a collaborative filtering (Matrix factorization) method that factors a matrix of user-movies to uncover hidden relations that can provide recommendations based on similar genres, crew, cast, and so on.

Index Terms - Movie Recommendation System, collaborative filtering,Matrix factorization

1. INTRODUCTION

A recommender system is a type of information filter that attempts to forecast a user's "rating" or "preference" for an item. Movies, music, news, products, and items in general all use recommendation algorithms. There are various methods that recommend systems that can generate a list of recommendations. Collaborative filtering and content-based filtering are two of the most prevalent approaches.

Recommender systems power the majority of the online goods we use today. YouTube, Netflix, Amazon, Pinterest, and a few other online services employ recommender systems to filter through millions of items and provide individualised recommendations to its users. Recommender systems have been thoroughly researched and demonstrated to give significant benefits to both online businesses and their customers.

2. RELATED WORK

Sometimes we rely on different recommendation systems to help us by recommending relevant books, music,etc. Most of the recommendation systems don’t perform well for a new user. So, we want to develop a system which works well for both new users and existing users. Initially, we are developing a model for new users based on number of views and ratings. Then, we developed a KNN model to recommend movies based on user id. The second model is based on SVD, one of the Matrix Factorization techniques for which we will use the Surprise library to develop the model for which we have to make a Dataset object in order to train recommender systems with

Surprise. A dataset that contains userId, movieId and rating is called a Surprise Dataset object. The third model is a neural network based recommender system where we trained a network with user and movie embeddings of size 50 each using tensorflow framework.

3. METHODS

3.1 Recommendation Based on Knowledge and KNN

The first approach is to suggest movies to a new user using knowledge from the dataset and to implement KNN model to recommend movies to a user.

For a new user, there is not much data available i.e., he didn't watch any movies and we don't know what kind of movies to recommend. So, this approach will help to generalise the suggestion i.e., recommending movies based on views and ratings.

3.1.1 Recommendation Based on Genre: The data has 19 different genres, and we are going to recommend movies based on genre. We will ask the user to select the genre he wants to watch and for the selected genre, we are considering the movies, which have more views and high average rating. Initially, we look for movies which have average rating equals to 5.0 and minimum views of 400. If there are not enough movies (we are recommending 10 movies at a time), then we decrease the rating by 0.5 and views by 50 till we get at least 10 movies for the selected genre.

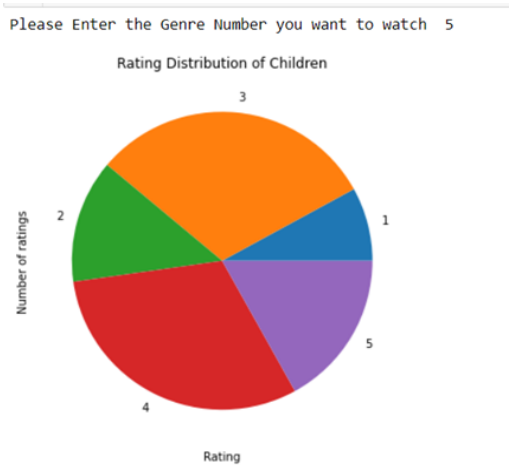

Fig.3.1 Rating Distribution of Genre- Children

The above pie chart shows the rating distribution for the 'Children' genre. We calculate the average rating for movies in this genre and we consider the movies which have more views and higher average rating.

3.1.2 Recommendation Based on Popularity: We are recommending the movies to a user, which has more views. This is one of the direct ways to recommend movies to a new user. Most of the people would like to watch the popular movies, so we considered this approach.

3.1.3 Recommendation Based on Rating: We are recommending movies to the user based on rating, i.e., the movies which have the highest average rating. This is also a direct way to recommend movies because people give high ratings to movies which are nice to watch.

3.1.4 Recommendation Based on Popularity and Average Rating: Recommending movies simply based on popularity or rating may not helps all the time, so we combined both the results and formed a new dataset which helps us

to recommend better movies i.e., popular movies which has good average rating. This is the best way to recommend a movie to a new user.

KNN Model: We are using K-NN to recommend relevant movies to a user based on cosine similarity. We are considering 15 nearest users and recommending 10 movies. To implement KNN we need to form a new data matrix i.e., rows should represent users and columns be movie titles. As the data have a lot of missing ratings because a user didn't watch all the movies, the matrix formed is a sparse matrix with a lot of missing values in it. We need to fill all the missing values with 0s. Then we need to transform this matrix to fit the KNN model. So, we transformed the matrix to SciPy sparse matrix for most efficient calculations.

We'll train the KNN model to locate users that are very similar to the user we've given as input, and then we'll propose the best movies that would be of interest to the input user. The next step would be taking the user id as input from the user. Then pick the movies which have more views and a higher average rating. And the most important step, eliminating the movies which the user has already seen and the movies which are not seen by the nearest users. Set weights for each nearest user's ratings depending on their distance from the input user. We'd be able to provide more accurate suggestions if we could define these weights.

3.2 Recommendation system using SVD: The second approach is using the SVD(Singular Value Decomposition) model which is a latent factor model, which means it can extract features and correlation from the user-item matrix. We can use SVD to basically discover relationships between items. Since , we are using Model-based collaborative filtering which estimates the parameters of statistical models to predict how individual users would rate an unrated item which can be a classification task that considers items over users as features and ratings as prediction labels.

The main idea of Collaborative filtering is to use interactions between groups of users and items(movies) to predict how users would rate items they have not rated before. For that, it identifies various patterns in interactions between a group of users and a group of items to infer the interests of individuals based on past interactions between a user and a set of items (movies) that are matched against past item-user interactions within a larger group of people.

While some dependencies are easy to identify, some exist where some movies receive high ratings from the same users where the dependencies between items are similar, we can identify this from the patterns that can exist in the user/item matrix in dependencies between users and items. For example, assume two users, Sam and Joy, have rated movies. Sam enjoyed Spiderman, Star Wars, and Godzilla. Joy enjoyed the same movies as Sam, except Godzilla, which he has not yet rated. Now, based on the similarity between Joy and Sam, we would assume that Joy would also enjoy Godzilla.

For example, assume that a different user Alan gave a three-star

rating to five different movies. Now, if another user, Bob, rated the same movies as Alan but always gave five stars, which is an instance of latent dependency. There is some form of dependence between the two users, though it is not as significant as mentioned in the previous example, considering latent dependencies will improve predictions.

We mentioned above that prediction of individual ratings for unseen movies as a classification task, we can use various algorithms however, user/item matrices can become very large, making searching for patterns computationally expensive. Also, users will typically rate only a minute fraction of the items in the matrix, so algorithms need to deal with an ample number of missing values (sparse matrix).

Matrix Factorization is one of the most widely used techniques for dimensionality reduction.Matrix Factorization compresses the initial sparse user/item matrix and factors it into two separate matrices that present items(movies) and users as unknown feature vectors. Now that the matrix is densely populated it is easier to handle, but it also enables the model to uncover the latent dependencies(which increases predictions) present among items and users, which increases model accuracy.

SingularValueDecomposition(SVD), is one of the Matrix Factorization techniques that has no limitation for the shape or the properties of the matrix to be decomposed. SVD is an approach to produce features that factors a matrix A (m * n) matrix into the three matrices U, Σ, and V.

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

And, we will not be implementing Collaborative Filtering from scratch in case of SVD as mentioned in the project progress report. Instead, we will use the Surprise (open-source) python library that uses algorithms like Singular Value Decomposition (SVD) to minimise the RMSE (Root Mean Square Error) and give accurate recommendations.

We need to make a Dataset object in order to train recommender systems with Surprise. A dataset that contains the following fields in the order mentioned below is called a Surprise Dataset object,

1. userId
2. movieId and,
3. Rating

3.3 - Recommendation System using Neural Network Embeddings:

The third approach we follow is neural network embeddings.

In general, neural networks achieve higher accuracy with more data and greater number of layers and are also flexible in terms of design. This model is aimed to explore the use of neural networks for Recommendation tasks.

One-Hot Encoding is a generic approach for vectorizing categorical characteristics of any kind. Creating and updating the vectorization is straightforward and quick; simply add a new item in the vector for each new category. However, the ease and rapidity of computation is overtaken by the “curse of dimensionality” when we

make a new dimension for every category. Embedding on the other hand requires more data with a good amount of distribution into occurrences of particular examples and greater training time complexity. The outcome is a dense vector with a predetermined number of dimensions.

Embeddings are widely used in natural language processing (NLP) to insert words with dense representation. The embedding vectors of words with comparable meanings are similar. The same method may be used with recommender systems. Users and items form the basis of the most basic recommender system: Which items should we suggest to a user? User IDs and item IDs are both available. We require two embedding tables one for each, users and objects. The embedding vectors of similar movies/users in several aspects are similar.

While the embeddings themselves aren't really fascinating - they're just vectors — they can be useful for three cases:Locating in the embedding space, the closest neighbours; input for a model to be trained; visualisation in lower dimensions.This model focuses on the first case.

Calculating the dot-product between the user embedding and the item embedding we get a final score, the likelihood that a user interacts with an item which is similar to Matrix factorization.We applied the sigmoid activation function as a last step to transform the output to desired range.

4. RESULTS

4.1.1 Recommendation Based on Genre:

```
Below movies have rating more than  3.0  with atleast  150  viewers.

                                movie title    rating  \
1                     Wizard of Oz, The (1939)  4.077236
2                                 Babe (1995)  3.995434
3                           Toy Story (1995)  3.878319
4              E.T. the Extra-Terrestrial (1982)  3.833333
5                             Aladdin (1992)  3.812785
7                    Beauty and the Beast (1991)  3.792079
8                          Lion King, The (1994)  3.781818
9                             Fantasia (1940)  3.770115
10                          Mary Poppins (1964)  3.724719
11        Snow White and the Seven Dwarfs (1937)  3.709302
15  Willy Wonka and the Chocolate Factory (1971)  3.631902
16                          Fly Away Home (1996)  3.600000
```

Fig: 4.1.1 Recommendations by Genre

Above are the recommended movies based on Genre. For the children genre, we considered the movies which had a minimum rating of 3.0 and 150 views.

4.1.2 Recommendation Based on Popularity:

	movie title	Total Views
0	Star Wars (1977)	583
1	Contact (1997)	509
2	Fargo (1996)	508
3	Return of the Jedi (1983)	507
4	Liar Liar (1997)	485
5	English Patient, The (1996)	481
6	Scream (1996)	478
7	Toy Story (1995)	452
8	Air Force One (1997)	431
9	Independence Day (ID4) (1996)	429

Fig 4.1.2. Popular Movies

Above are the most popular movies from the dataset.

4.1.3 Recommendation Based on Rating:

	movie title	Average Rating
0	They Made Me a Criminal (1939)	5.0
1	Marlene Dietrich: Shadow and Light (1996)	5.0
2	Saint of Fort Washington, The (1993)	5.0
3	Someone Else's America (1995)	5.0
4	Star Kid (1997)	5.0
5	Great Day in Harlem, A (1994)	5.0
6	Aiqing wansui (1994)	5.0
7	Santa with Muscles (1996)	5.0
8	Prefontaine (1997)	5.0
9	Entertaining Angels: The Dorothy Day Story (1996)	5.0

Fig 4.1.3. Highly Rated Movies

Above movies are top 10 highly rated movies by users.

4.1.4 Recommendation Based on Popularity and Average Rating:

	movie title	Average Rating	Total Views
23	Star Wars (1977)	4.358491	583
32	Silence of the Lambs, The (1991)	4.289744	390
34	Godfather, The (1972)	4.283293	413
40	Raiders of the Lost Ark (1981)	4.252381	420
45	Titanic (1997)	4.245714	350
49	Empire Strikes Back, The (1980)	4.204360	367
61	Princess Bride, The (1987)	4.172840	324
64	Fargo (1996)	4.155512	508
98	Monty Python and the Holy Grail (1974)	4.066456	316
101	Pulp Fiction (1994)	4.060914	394
114	Fugitive, The (1993)	4.044643	336
129	Return of the Jedi (1983)	4.007890	507

Fig 4.1.4. Popular movies with high rating

Above movies are popular movies with high ratings.

KNN Model: The movies recommended for a user with user id equals to 45.

```
Movies Recommended Based on Nearest Users:

['Silence of the Lambs, The (1991)',
 'Raiders of the Lost Ark (1981)',
 'Back to the Future (1985)',
 'Contact (1997)',
 'Liar Liar (1997)',
 'Pulp Fiction (1994)',
 'Titanic (1997)',
 'Terminator 2: Judgment Day (1991)',
 'Princess Bride, The (1987)',
 'Fish Called Wanda, A (1988)']
```

Fig 4.1.5 Movies recommended by KNN model

We were able to eliminate the movies which were seen by the user and the movies which were not seen by the nearest users. The KNN model is easy to implement and there is no need to tune several parameters. But the algorithm gets slower as the number of examples increases. As the data has more zeros, the data sparsity is the real issue for KNN models and the distance in the model starts to fall apart.

4.2 Recommendation System using SVD:

Matrix-factorization-based recommender systems often factor a matrix of ratings into a product of matrices indicating latent factors for the objects (movies) and the users. The rating matrix, R, has missing values in several areas, as seen in the diagram below. When predicting existing ratings using matrix components, the matrix factorization technique employs a procedure known as gradient descent to reduce inaccuracy. As a result, a recommendation system is built using an algorithm like SVD, which fills in the rating matrix by predicting the ratings that each user would assign to each item in the dataset.

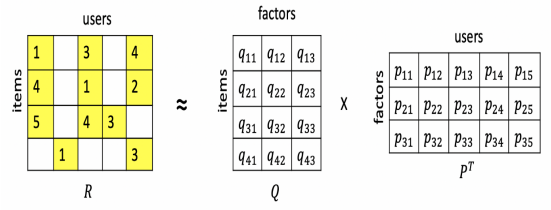

Fig. 4.2.1. Ratings matrix factored into product of item and user matrices

From the equation below input matrix A is factored into three matrices using SVD.

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

$$A = R, \qquad Q = U, \qquad P^T = \Sigma V^T$$

In the case of our movie recommendation system, the SVD algorithm will represent the rating matrix as a product of matrices representing the items(movies) factors and user factors respectively.

```
[ ]  svd = SVD() #SVD Model Training
     model_inst = svd.fit(train_set)
```

Fig. 4.2.2. Image showing SVD Model Training.

In the code above, the first line creates an untrained model which uses Probabilistic Matrix Factorization(PMF) for dimensionality reduction. The model will be fitted to the training data in the second line.

```
[ ] # 10-fold cross validation to validate the
    #performance of our movie recommendation system
    cross_val_10Fold = cross_validate(model_inst, user_ratings,
                                      measures=['RMSE', 'MAE'],
                                      cv=10, verbose=True)
```

Fig.4.2.3. Image showing cross-validation.

Now, running the code where we cross-validated an SVD model using ten-fold cross-validation, wil produce the following output, as shown below

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10
RMSE (testset)	0.8655	0.8606	0.8601	0.8720	0.8685	0.8731	0.8685	0.8783	0.8735	0.8673
MAE (testset)	0.6645	0.6621	0.6622	0.6689	0.6675	0.6680	0.6711	0.6731	0.6722	0.6637
Fit time	5.77	5.74	5.78	5.73	5.74	5.76	5.75	5.73	6.18	6.62
Test time	0.10	0.09	0.07	0.17	0.08	0.07	0.07	0.07	0.18	0.09

Fig.4.2.4. Image showing the RMSE and MAE results.

From the above result, it is clear that mean deviation of our predictions from the actual rating is a little below 0.7 and there is little below 0.8 root mean square deviation of our predictions from the actual rating for all the k(10)-folds which implies that there are no significant differences between the performance in the different folds. As we know that MAE says little about possible outliers in the predictions, but since we are dealing with ordinal predictions (1-5), the influence of outliers is naturally restricted.

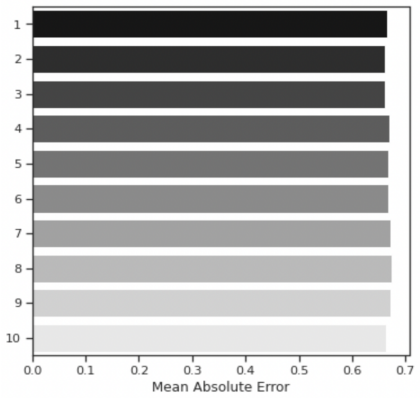

Fig.4.2.4. Plot showing No.of folds vs MAE.

4.3 - **Recommendation System using**

Neural Network Embeddings

The movies recommended for a user with user id equals to 120.

```
10 Movies recommended to user 120 are:
============================================================
Dead Presidents (1995) -------- Action|Crime|Drama
Goofy Movie, A (1995) -------- Animation|Children|Comedy|Romance
Kika (1993) -------- Comedy|Drama
Bhaji on the Beach (1993) -------- Comedy|Drama
Little Big League (1994) -------- Comedy|Drama
Spanking the Monkey (1994) -------- Comedy|Drama
Princess Caraboo (1994) -------- Drama
Celluloid Closet, The (1995) -------- Documentary
Chinese Box (1997) -------- Drama|Romance
Toys (1992) -------- Comedy|Fantasy
Son of the Sheik, The (1926) -------- Adventure|Comedy|Romance
My Life (1993) -------- Drama
Carnosaur (1993) -------- Horror|Sci-Fi
============================================================
```

Fig.4.3.1. Image showing movies recommended by Neural Network Embeddings

The loss over 10 epochs can be seen as shown in Fig.4.3.2. Also the train and test set losses are plotted against the number of epochs as shown in Fig.4.3.3 for a better understanding of the functioning of the model.

```
Epoch 1/10
2836/2836 - 19s - loss: 0.6318 - val_loss: 0.6174 - 19s/epoch - 7ms/step
Epoch 2/10
2836/2836 - 13s - loss: 0.6098 - val_loss: 0.6132 - 13s/epoch - 5ms/step
Epoch 3/10
2836/2836 - 13s - loss: 0.6059 - val_loss: 0.6103 - 13s/epoch - 4ms/step
Epoch 4/10
2836/2836 - 13s - loss: 0.6041 - val_loss: 0.6096 - 13s/epoch - 5ms/step
Epoch 5/10
2836/2836 - 14s - loss: 0.6024 - val_loss: 0.6099 - 14s/epoch - 5ms/step
Epoch 6/10
2836/2836 - 13s - loss: 0.6018 - val_loss: 0.6101 - 13s/epoch - 5ms/step
Epoch 7/10
2836/2836 - 13s - loss: 0.6013 - val_loss: 0.6095 - 13s/epoch - 5ms/step
Epoch 8/10
2836/2836 - 13s - loss: 0.6002 - val_loss: 0.6096 - 13s/epoch - 5ms/step
Epoch 9/10
2836/2836 - 13s - loss: 0.6000 - val_loss: 0.6085 - 13s/epoch - 5ms/step
Epoch 10/10
2836/2836 - 13s - loss: 0.6001 - val_loss: 0.6095 - 13s/epoch - 5ms/step
```

Fig.4.3.2. Loss over 10 epochs

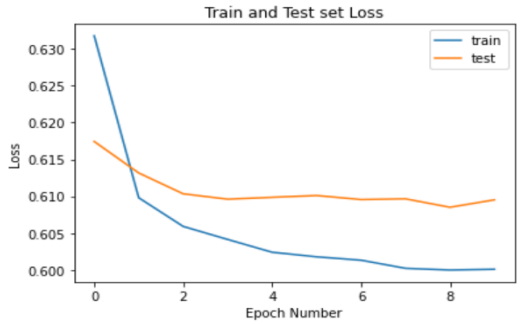

Fig.4.3.3. Plot showing Train and Test Set Loss vs Epoch Number

5. CONCLUSION

Movie recommendation system is a vast concept to explore and never a single model or method would suffice for predicting accurately user preferred recommendations there might be some error that we can encounter or inaccurate predictions that might happen and can be suggested to the user which user didn't intend to get. Here, we trained each of the three models separately using the same dataset.

Though there are many movie recommendation systems out there, these systems have the limitation of not recommending the movie efficiently to the existing users and there might be a chance of some unwanted recommendations for the new user that is new in the system who don't have rating history in which case it is hard to give recommendations to that user.

For such cases where the user is new and doesn't have any rating history, we implemented a KNN model where users will be suggested some good(preferred) movies based on Genre, Popularity, Rating or combination of popularity and averaged rating (best method) based on the input given by the user.

And the system evaluation is done using various metrics such as RMSE(Root Mean Squared Error) and MAE(Mean Absolute Error). Evaluation is done by cross-validation on the given dataset where we divide the dataset into 10 different folds in method-2(SVD) and perform evaluation on each fold from which we get RMSE and MAE on each of the folds.

The lowest value of the RMSE is considered the best case for prediction in building movie recommendation systems. Therefore, the model where we get the low

RMSE score is recommended to be used.

For the SVD Model, the RMSE value is around 0.8 and for the neural network system the loss is around 0.6, so from these two values it is clear that the neural network system performs better when compared to the SVD Model.

In future, we want to consider a larger dataset with more features like cast, crew, budget and earnings. So, we can train a model to recommend movies based on those attributes, like favourite hero or heroine, director and movies which performed well at box office. We also want to add a new feature, so that a user can also search movies based on keywords.

Finally, as said movie recommendation system is really a vast concept and there is a lot that we didn't inspect but each one of us in the team have learned and implemented some things that we didn't have hands-on learning before, which made us proficient eventually through the project in each model that we individually worked on like KNN,SVD and neural networks.

6. REFERENCES

[1] What is the k-nearest neighbors algorithm? | IBM

[2] EDA for Machine Learning | Exploratory Data Analysis in Python

[3] k-nearest neighbours algorithm - Wikipedia

[4] sklearn.neighbors.KNeighborsClassifier — scikit-learn 1.0.2 documentation

[5] https://www.freecodecamp.org/news/singular-value-decomposition-vs-matrix-factorization-in-recommender-systems-b1e99bc73599/

[6]https://dl.acm.org/doi/pdf/10.1145/371920.372071

[7] http://surpriselib.com/

[8] Recommender Systems: By Charu C. Aggar.

[9] https://dl.acm.org/doi/pdf/10.1145/3038912.3052569?casa_token=FOTobrfZfi8AAAAA:mRGIaTjOkeTqBniUnDZBf4VlohwMEnYuoNgOkH7L1ePv-wYOWOlZqVKf4DL048W3FXwHTwlpAFU

[10] https://dl.acm.org/doi/pdf/10.1145/371920.372071?casa_token=krwHjtmcZCwAAAAA:swCpfzE4DZdluqfqZPfsfm9Ra1Qk20qosgM1bYcaSOEe5qIJCObziBQxV5bVyt5ugas2rD7IrUc