DataScience w/ Python

Project Report Document

Webflix Recommender System

TABLE OF CONTENTS

Introduction

How does youtube know what video you might want to watch next? How does the Google Play store pick an app just for you?or How does Amazon suggest you an item that is related to what you are looking for? Magic? No, in the above cases an ML-based recommendation model determines how similar videos and apps are to other things you like and then serves up a recommendation. This brings us to the question:

What are recommendation Systems?

A recommendation system is an artificial intelligence or AI algorithm,usually associated with machine learning,that uses big data to suggest or recommend additional products to consumers.These recommendations can be based on various criteria,including past purchases,search history,demographic information, and other factors. Recommender systems are highly useful as they help users discover products and services they might otherwise have not found on their own.

Recommender systems are trained to understand the preferences, previous decisions, and characteristics of people and products using data gathered about their interactions.These include impressions, Clicks, and purchases. Because of their capability to predict consumer interests and desires on a highly personalized level,recommender systems are a favorite with content and product providers. They can drive consumers to just about any product or service that interests them,from books to videos to health classes to clothing.

While there are a vast number of recommendation systems most of them fall under the category collaborative filtering, content filtering, and context filtering.

Applications of Recommender systems

Recommendation systems are one of the most successful and widespread application of machine learning technologies in business.You can find large scale recommender systems in retail,Video on demand,or music streaming.Best examples of current Recommendation systems are:

1. Offering news articles to online newspaper readers,based on a prediction of reader interests.
2. Recommending movies to user based on the previous watch
3. E-commerce & Retail :Personalized merchandising
4. Personalized Banking ,etc

Benefits of Recommendation Systems

Recommender systems play a key role in making personalized user experiences,user engagement and decision support tools in different industries such as healthcare, finance, entertainment etc.

Companies implement recommendation systems for the following:

1. Improving user retention by continuously catering to user preferences due to which businesses are more likely to retain them as loyal subscribers.
2. Increasing sales. Sales can be increased with recommendation system strategies as simple as adding matching product recommendations to a purchase confirmation.
3. Helping to form customer habits and trends
4. Speeding the pace of work.Analysts can save much when served tailored recommendations for resources and other materials necessary for further research.
5. Boosting cart value by using various means of filtering techniques.

How do recommenders work

How a recommender model makes recommendations will depend on the type of data you have.If you have only data about what interactions have occurred in the past,you will probably be interested in collaborative filtering.If you have data describing the user and items they have interacted with,you can model the likelihood of a new interaction given these properties at the current moment by adding content and context filtering.

Literature

In the 16 years from 1998 to 2013, more than 200 research articles were published in the field of research-paper recommender systems. The articles consisted primarily of peer reviewed conference papers (59%), journal articles (16%), pre-prints (5 %), and oilier documents such as presentations and web pages (15 %). The few existing literature surveys in this field cover only a fraction of these articles.

In this era,recommendation systems are a very popular technique and help to give a better experience for the user as well as the company .These systems are several types such as content based, collaborative, or hybrid, according to the system which the developers have made.

The table below shows a typical review of work done by various authors on Recommendation systems.

TABLE.1. A REVIEW OF WORK DONE BY VARIOUS AUTHORS

S. No.	Title of the Paper	Key Points	Conclusion	Reference
1.	A Case-Based Recommendation Approach for Market Basket Data.	CF; CB; AR (Association Rule); CBR (Case-Based Reasoning)	After compared the performance of developed RS conclude that CBR is the good method in case of transactions.	[1]
2.	Recommender Systems: An overview of different approaches to recommendations	Recommendation System; Information Retrieval System; CF; CBF; Hybrid Filtering	The three approaches of recommendation system and their advantages and disadvantages.	[2]
3.	Recommendation analysis on Item-based and User-based Collaboration Filtering	IBCF; UBCF; Recommender System	IBCF and UBCF with implementation metrics, and conclude that IBCF provide better results than UBCF.	[3]
4.	Recommender Systems Handbook. Springer.	CF; CB; Multi-criteria recommender; Robust CF Neighborhood-based	In the unique approaches, hybrid robust filtering methods are better.	[5]
5.	Towards privacy in a context-aware social network based recommendation system	Content aware; social networking; privacy	Focus on protecting data and request for data, at the point of data collection.	[7]
6.	A study of hybrid recommendation algorithm based on user.	Personalization; recommendation technology; collaborative filtering; hybrid algorithm	Hybrid algorithms are generates the results according to user's rating and history record.	[14]
7.	Recommender systems in e-commerce	Electronic Commerce, cross-sell, up-sell, mass customization	The ideas of new applications in the field of recommendation systems in e-commerce sites.	[18]

Techniques used

Content Based Filtering

This method is based on similarity of movies. If a user liked the movie "The Dark Knight", they would probably like "Joker".

Approach 1: Find the similarity between all pairs of movies using TF IDF concept, then use the most similar movies to a user's already-rated movies to generate a list of recommendations.

Approach 2: User Profile is created based on the data given by the user and suggestions are made on the basis of that. As the user provides more inputs or takes actions on the recommendations, the engine becomes more and more accurate.

The main advantage of Content Based Filtering is that the model will be able to recommend new and unpopular items as it does not interact with the preferences of other users. With this method the model is capable of recommending users with unique tastes since it strictly depends on the preferences given by the user. The method is highly interpretable as we can provide explanations of recommended movies for a user by listing the features that caused a movie to be recommended.

Couple of disadvantages associated with Content based filtering is that the model is not capable of exploiting quality judgments of other users as it doesn't take into account other user preferences. This method might not give accurate predictions as it is majorly dependent on hand-engineered features like genre etc.

Collaborative Filtering

This method is better as compared to content based filtering because it uses similarities between users and movies to provide recommendations. Here recommendation of movies to user A will be based on the preferences/interests of a similar user B. This approach produces latent features based on the patterns found in data unlike content based filtering which requires hand-engineered features like genre etc to match the user preferences with movies.

Some of the disadvantages of Collaborative Filtering could be that there needs to be enough other users already in the system to find a match. The model is not capable of recommending movies that have not been previously rated by any user. Popularity bias is one huge disadvantage of this method as the model tends to recommend popular items and cannot recommend items to someone with unique tastes.

Data & Findings

Webflix recommender system uses two datasets with sources from grouplens and Kaggle. Refer to the Appendix [1] and Appendix [2] for the source links respectively. Each dataset is segregated into two files. Dataset 1 contains 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 users whereas dataset 2 contains 26,000,000 ratings applied to 45,000 movies by 270,000 users. The datasets

have been cleaned by creating new columns, removing irrelevant columns, converting column values into lists, and filtering the rows grouped by criteria necessary to the analysis. The details of the columns and the description are given in the data dictionary. Refer to Appendix [3] and [4] for the data dictionary of dataset 1 and dataset 2 respectively. The two datasets have been combined with the standard variable imdbId.

EDA for dataset1: Based on the variables present in the dataset, different graphs have been plotted to visualize the data. From the Figure in Appendix [5], we can observe that Drama, Comedy, and Action are the 3 top most popular movies among genres indicating the movie recommender system to focus more on these genres. The figure in Appendix [6] shows the top users who have rated most of the movies. Identifying and promoting top contributors can help the company develop more inputs to the database. Figures in Appendix [7] and [8] illustrate movies with high and low average ratings respectively and these findings can help the system to prioritize the movies and promote them according to the average rating value. It is very important to find the interest of the user for the specific genre and the figures in Appendix [9] and [10] describe the percentage of the genre with maximum and minimum ratings respectively. It can be seen that most percentages of Film-Noir and War have gained success with maximum rating whereas the Horror and Children movies seem to get less attention with the highest percentage of movies with less rating among the genres. Figure from Appendix [11] highlights the movie rating frequency in a specific range of 20 years starting from 1921 to 2020. We can observe the same pattern for all the time periods with a count for 3.5 to 4 rating being the highest and followed by 2.5 to 3 and 4.5 to 5 ratings. The same trend can also be found in Figure [12] of the Appendix with respect to the genres but the drama genre seems to have got more 4.5 to 5 ratings when compared to other genres.

EDA for dataset2: Dataset 2 has a rating value in terms of vote average on a scale of 10. Vote average and vote count for the movies can play a significant role in identifying the popularity of the movies and the figures from Appendix [13] and [14] show the movies with the highest vote average and vote count. Figure in Appendix [15] gives the movie vote average frequency for Genres and we can observe that the majority of the movies have a vote average from 6.1 to 8 followed by 4.1 to 6. Based on the title, a word cloud has been created and the figure in Appendix [16] shows the most frequent words and these can be used in the recommender system in giving the most familiar results.

Implementation

Content Based Implementation

From our EDA, we saw the best Romance, Action, Horror and other genre movies that can be recommended to users. But this is not taking into consideration the user's personal preferences and taste. In order to incorporate the same, we built a content recommendation filtering system.

Goal : To recommend movies that are the **most similar to a movie liked by a user**, by computing similarity between movies, using Movie Metadata. (Hence, Content Based Filtering)

Content-based recommenders use data exclusively on the items or in this case, the 'movies'. To do so, we require a basic understanding of the users' preferences so that we can suggest new movies with tags/keywords that are comparable to those stated (or inferred) by the user. We implement this using a **TF-IDF Vectorizer.**

Tf–idf or term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. TF-IDF will help capture the importance of movie titles/items by giving a higher weight to the less frequent items and vice-versa. It is for converting text into a meaningful numerical representation, which is then used to fit a machine learning algorithm for prediction.

$$w_{i,j} = tf_{i,j} \cdot log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j

$df_{i,j}$ = number of documents

N = total number of documents

Source : https://towardsdatascience.com/content-based-recommender-systems-28a1dbd858f5

We combine two datasets for our analysis, our original 'movielens' dataset and a new 'movie_metadata' dataset, which contains additional information like Movie descriptions and taglines. We combine a movie's title, overview and tagline to form a movie description which we then pass to our TF-IDF Vectorizer object.

Please see Appendix [17] for what the description passed for the movie 'Toy Story' looks like.

We then use sklearn's TfidfVectorizer. (Appendix [18]) and try to find similar tf-idf vectors (movies). We then define a proximity measure : cosine similarity which is equal to the cosine of the angle between the

two vectors being compared. The lower the angle between two vectors, the higher the cosine will be, hence yielding a higher similarity factor. It is expressed as follows:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

Source : *https://towardsdatascience.com/content-based-recommender-systems-28a1dbd858f5*

To compute the cosine similarities between all tf-idf vectors, we calculate the dot product of the TF-IDF vectorizer to give us a cosine similarity score. Since we have used the TF-IDF Vectorizer, calculating the Dot Product will directly give us the Cosine Similarity Score. Therefore, we will use sklearn's linear_kernel instead of cosine_similarities since it is much faster. (Appendix [19])

We now have a pairwise cosine similarity matrix for all the movies in our dataset, and then write a function to return the movies that are most similar to a given movie. (Appendix [20])

Content Based Filtering Model results

Refer to Appendix [21] for Model results.

We can see that the model returns similar movies in the Batman franchise, but it is not taking into consideration movie ratings at all. For example, Wayne's world and Batman and Robin are two such movies that have extremely low ratings and should not be recommended. Hence, we modify our system to include movies with good reviews. For this, we use IMDB's ***weighted rating*** **formula.** The formula for calculating the Top Rated 250 Titles gives a true Bayesian estimate:

weighted rating (WR) = (v ÷ (v+m)) × R + (m ÷ (v+m)) × C where:

R = average for the movie (mean) = (Rating)

v = number of votes for the movie = (votes)

m = minimum votes required to be listed in the Top 250

C = the mean vote across the whole report

Based on similarity scores, we take the top 25 movies and calculate the 60th percentile vote, which we use as 'm' in our equation. We then derive the weighted rating of every movie using the above formula. See Appendix[22] for the function used.

Refer to Appendix [23] for improved recommendations by taking rating into consideration for 'Batman Forever' and 'Jumanji'.

This model is great for new users, i.e. Users who are using a recommendation system for the first time wherein prior data on their taste and preferences does not exist.

Collaborative Filtering Model

The main idea behind this method is to use other users' preferences and taste to recommend new items to a user. It's based on the idea that people who agreed in their taste/evaluation of certain items are likely to agree again in the future. The usual procedure is to find similar users (or items) to recommend new items which were liked by those users, and which presumably will also be liked by the user being recommended.

Hence, Collaborative filtering systems apply such a similarity index-based technique, wherein a number of users are selected based on their similarity to the user under observation.Predictions for the active user is made by calculating a weighted average of the ratings of the selected users.

We use the Surprise library for our Collaborative Filtering analysis. We begin with initializing our Reader and Dataset objects, where we indicate the rating scale which in our case is between 0 and 5, and load our ratings data from our movielens dataset.

We then build and run the below algorithms :

1. **Baseline Model**

We begin with implementation of a baseline model, which is a KNNBaseline model without any hyperparameters. The process is like every other ML model where we instantiate the model, fit on the train set, then predict on the test set.

The algorithm predicts the baseline estimate for a given user and movie. (Appendix [24])

The RMSE for our baseline is 0.83 and the MAE is 0.64. These are the values we will look to improve by attempting different models and including hyperparameters in future models.

2. **SVD (Matrix Factorization-based algorithms)**

The SVD model is one of the most powerful recommendation systems. SVD is a type of matrix factorization that uses gradient descent to forecast user ratings while minimizing the difference between the expected and actual ratings from our original utility matrix. As a result, when forecasting these new

ratings, gradient descent minimizes RMSE. SVD is one of the best models when it comes to dimensionality reduction. An approach like this minimizes the number of features by shrinking the spatial dimension from N to K (where K<N). The elements are determined by the users' evaluations, and SVD creates a matrix with a row of users and columns of objects. Singular value decomposition decomposes a matrix into three other matrices and extracts the factors from the factorization of a high-level (user-item-rating) matrix.

Our first model will be an SVD Model using GridSearch. We will first apply GridSearch to identify the best parameters that reduce our RMSE, and then will re-instantiate/tune our model with these parameters so we can then fit on the train set and predict on the test set.

Note: When tweaking hyperparameters, we employ **GridSearch as an optimization tool**. We select the optimum combination of parameters for our data by defining the grid of parameters we wish to search through. The hypothesis is that there is a precise set of values for the various hyperparameters that will reduce our predictive model's inaccuracy. The purpose of GridSearch is to locate this precise set of parameters.

Hyperparameters tuned :

- **lr_all** – The learning rate for all parameters. Default is **0.005**.
- **n_factors** – The number of factors. Default is **100**.
- **n_epochs** – The number of iterations of the SVD procedure. Default is **20**.
- **reg_all** – The regularization term for all parameters. Default is **0.02**.

(Refer to Appendix [25])

We then fit the SVD model with the parameters specified by Gridsearch. See results in Appendix [26).

Both our RMSE and MAE results are lower than in the baseline. This is possibly because we have filtered out users and movies with lower ratings so that the sparsity of the matrix decreases.

3. KNN Basic

Another common model to use is “KNN”. KNNBasic is a basic collaborative filtering algorithm. The algorithm looks at the nearest neighbors to determine which movie to predict.

We then applied GridSearch as our optimization tool to identify the best parameters that reduce our RMSE. Hyperparameters tuned :

- **min_k** (*int*) – The minimum number of neighbors to take into account for aggregation. Default is **1**.
- **sim_options** (*dict*) – A dictionary of options for the similarity measure. (example: cosine vs. pearson)

We then fit the KNNBasic model with the parameters specified by Gridsearch. See results in Appendix [27].

4. KNN Baseline

The next model we will explore is the KNN Baseline model. KNNBaseline is a basic collaborative filtering algorithm taking into account a baseline rating. The first model we ran was a KNN Baseline model with no hyperparameters. Here, we shall include hyperparameters to tune our model and try improving the results. The hyperparameters are the same as the previous KNN Model. We then applied GridSearch as our optimization tool to identify the best parameters that reduce our RMSE.

We then fit the KNNBasic model with the parameters specified by Gridsearch. See results in Appendix [28].

Cross-Validation

There's always a risk of overfitting when assessing a model's performance with different combinations of hyperparameters. We use the Cross Validation Error to gain a more accurate approximation of how well a set of hyperparameters performs. The data is separated into several pieces in Cross-Validation, say x sections, for instance. The model is then fit x times with 1/xth of the data missing each time. The performance is measured using the 1/xth data that was left out. For one combination of hyperparameter values, the average of the x errors constitutes the cross-validation error.

So we perform 3-fold cross validation on our 4 tuned models to verify the RMSE for test data. We can see the results below under 'CV'.

Model Comparison

Please see Appendix [29] for the Model comparison table.

Unfortunately, all of our models perform worse than the baseline, with the exception of the SVD model. SVD had an RMSE of 0.82, compared to the baseline of 0.83.

Additionally, SVD MAE score was 0.63 which is lower than the baseline result of 0.64. Lastly, when performing 3-fold cross validation, our SVD performs slightly better than the baseline with a result of 0.808, compared to the baseline result of 0.815.

As a result, we will disregard the KNN models for the remainder of our analysis and will move forward with our SVD tuned model, as it performed the best in terms of RMSE, MAE and CV.

Therefore, we can conclude that on average, our SVD model estimates ratings with an error of approximately 0.8. On a scale of 0-5, this 0.80 value is not too significant, as a rating of 3 compared to 3.8 is not a significant difference in context. Generally, with these models, we are trying to get a sense of what rating the user would rate a movie; and since these results are quite difficult to validate (we do not actually know if a user will enjoy a movie or not in reality), the estimation error of 0.80 is reasonably acceptable.

Generating New Ratings

We will then create a function that generates ratings for a brand new user, using predictions from our final SVD model. We will then show how our model can use these ratings in order to make predictions.

See Appendix [30] for Collaborative Filtering Model results.

- Incorporate other features like ‘Genres’, ‘Cast’, and ‘Crew’ in our collaborative model.

Recommendation

- The goal of the recommender system is to keep customers engaged with recommendation through which they retain existing customers and entice new customers with their AI powered recommendations.
- Webflix should make selective investments in movies/genres based on the customer ratings and popularity.
- As war and film noir has the highest rating, Webflix should concentrate on investing into the best films in other genres which brings the best ratings from the users.
- Improving customer engagement through personalized recommendations. Based on vote average frequency, comedy and drama are highly voted by the users.

Conclusion and Future work

Overall, our Collaborative filtering model does a decent job of estimating user's ratings with an error of 0.8, and our content based filtering model successfully recommends movies to new users.

Disadvantage :

Collaborative filtering Models do not address the 'cold start problem', i.e. If you do not have ratings from the users, how will you predict more ratings?

Solution :

Create a hybrid model that incorporates both content and collaborative models, where the Content based model is used for new users and the Collaborative filtering Model used when enough data on user's preferences has been gathered.

Future work :

- Incorporate other features like 'Genres' and 'Year' in our collaborative model.
- Incorporate other features like 'Genres', 'Cast', and 'Crew' in our collaborative model.

References

[1] *https://medium.com/@cmukesh8688/tf-idf-vectorizer-scikit-learn-dbc0244a911a*

[2] *https://towardsdatascience.com/content-based-recommender-systems-28a1dbd858f5*

[3] *https://www.irjet.net/archives/V7/i9/IRJET-V7I9633.pdf*

Appendix

[1] https://grouplens.org/datasets/movielens/1m/

[2] https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset?select=movies_metadata.csv

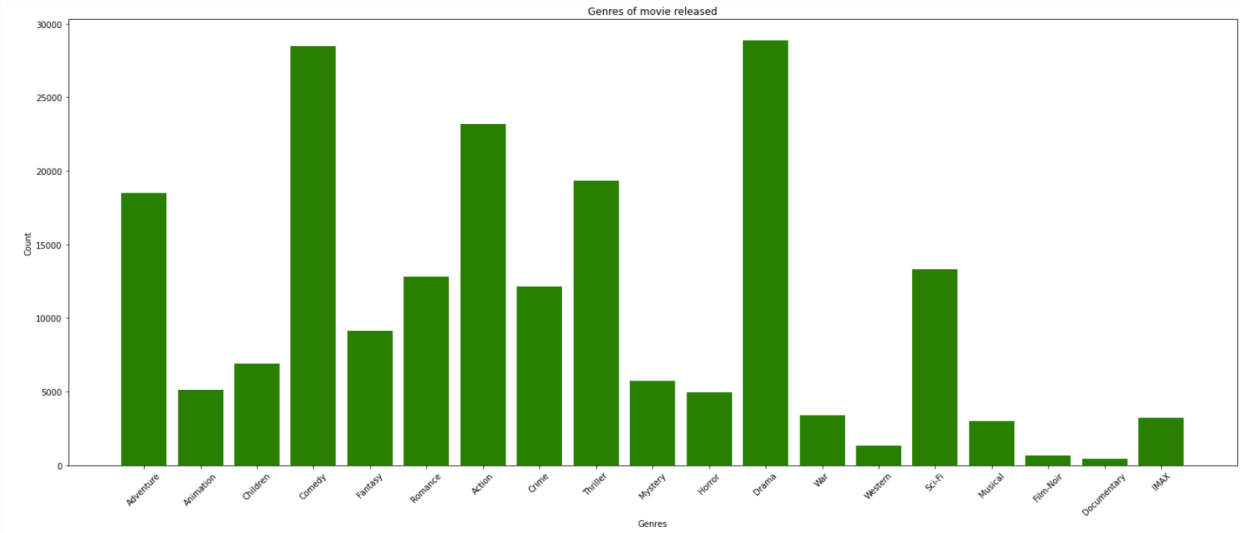
[3]

Column Name	**Description**
movieId	ID of the Movie
title	Title of the Movie
genres	Genre of the Movie
userId	ID of the User who rated the Movies
rating	Rating given by the User for a Movie
year	Year of the Movie being released

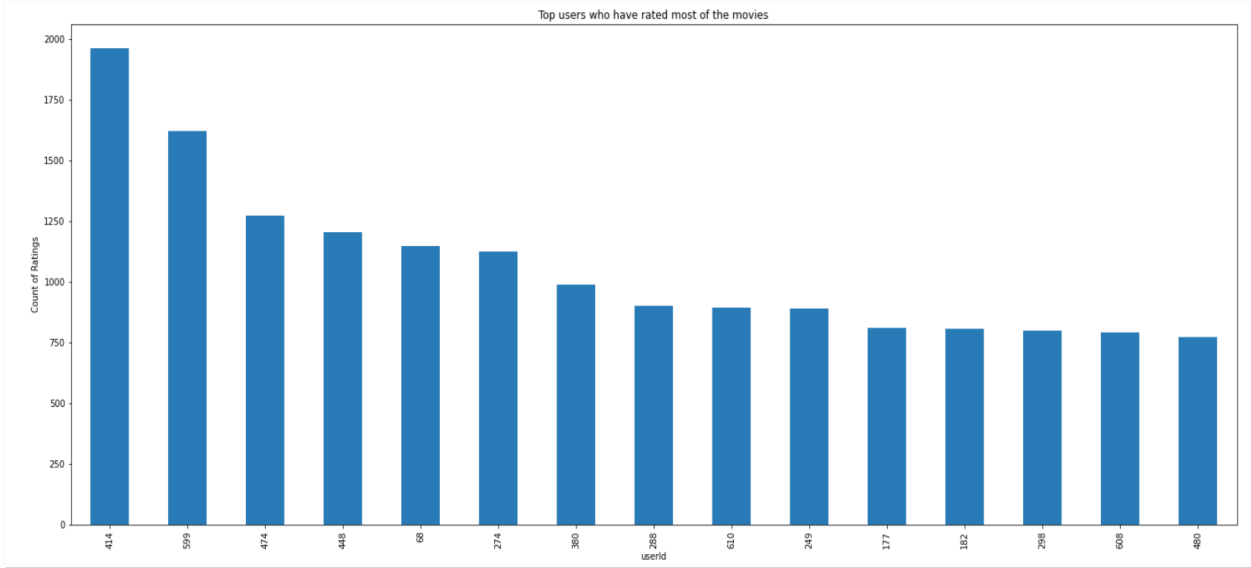
[4]

Column Name	Description
movieId	ID of the Movie
imdbId	ID of Movie in IMDB
title	Title of the Movie
genres	Genre of the Movie
overview	Overview of the Movie
tagline	Tagline of the Movie
vote_count	Count of Votes given to Movie by Users
vote_average	Vote average of Movie given by Users

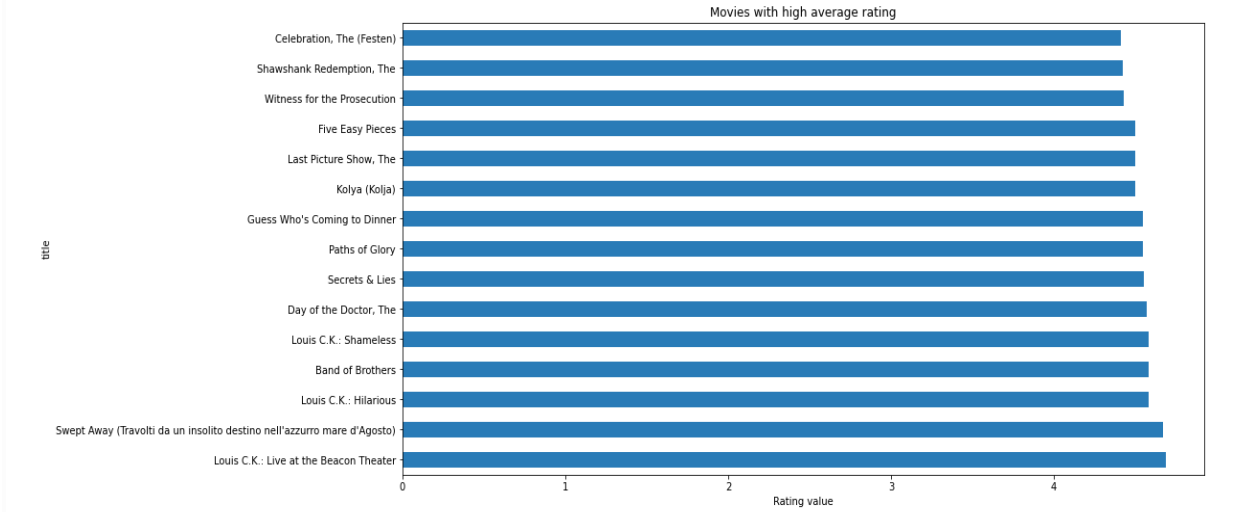
[5]



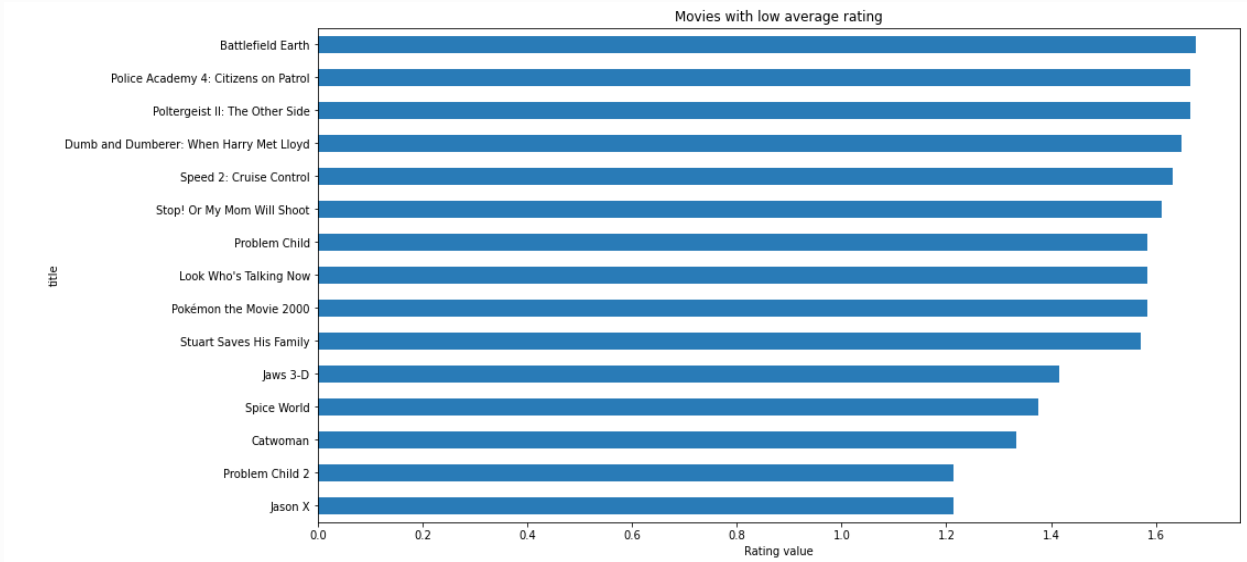
[6]


Top users who have rated most of the movies
Count of Ratings
userId
2000
1750
1500
1250
1000
750
500
250
0
414
599
474
448
68
274
380
288
610
249
177
182
298
608
480

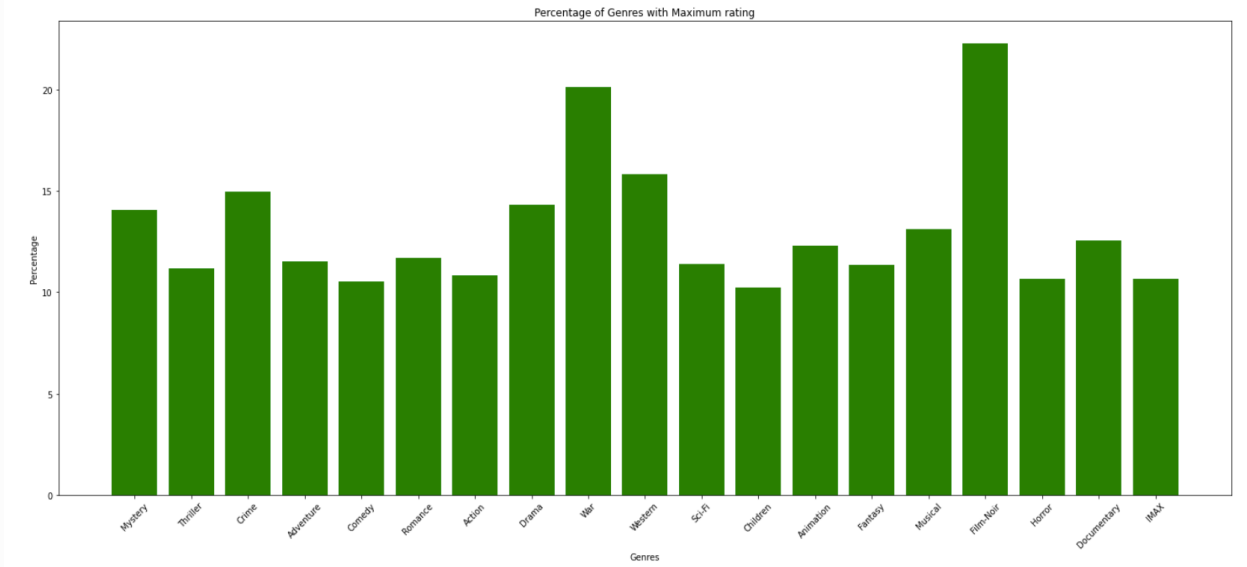
[7]


Movies with high average rating
title
Rating value
Celebration, The (Festen)
Shawshank Redemption, The
Witness for the Prosecution
Five Easy Pieces
Last Picture Show, The
Kolya (Kolja)
Guess Who's Coming to Dinner
Paths of Glory
Secrets & Lies
Day of the Doctor, The
Louis C.K.: Shameless
Band of Brothers
Louis C.K.: Hilarious
Swept Away (Travolti da un insolito destino nell'azzurro mare d'Agosto)
Louis C.K.: Live at the Beacon Theater
0
1
2
3
4

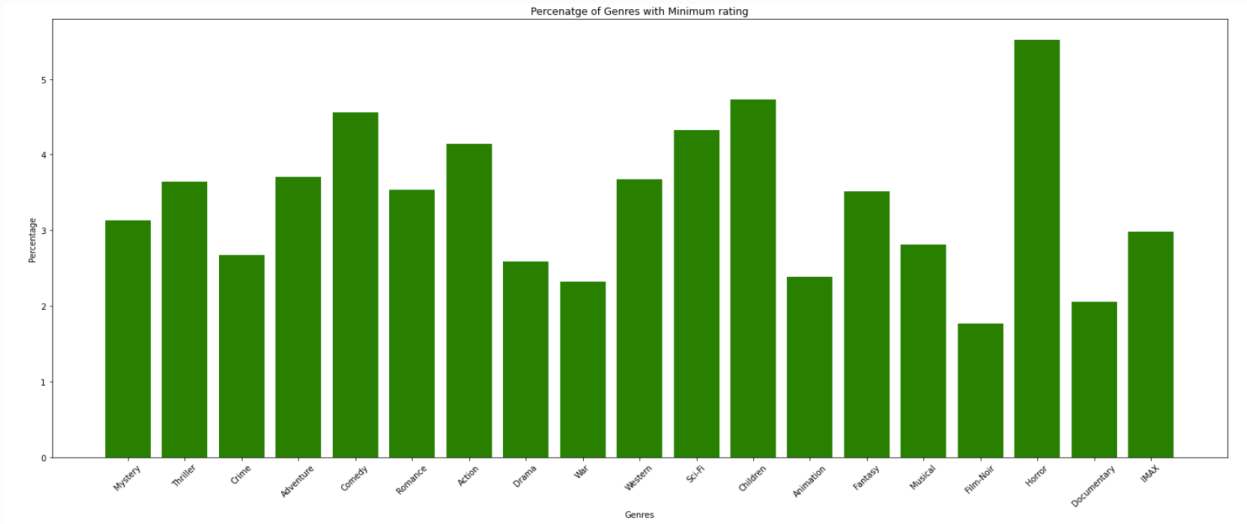
[8]


Movies with low average rating
Battlefield Earth
Police Academy 4: Citizens on Patrol
Poltergeist II: The Other Side
Dumb and Dumberer: When Harry Met Lloyd
Speed 2: Cruise Control
Stop! Or My Mom Will Shoot
Problem Child
Look Who's Talking Now
Pokémon the Movie 2000
Stuart Saves His Family
Jaws 3-D
Spice World
Catwoman
Problem Child 2
Jason X
title
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
Rating value

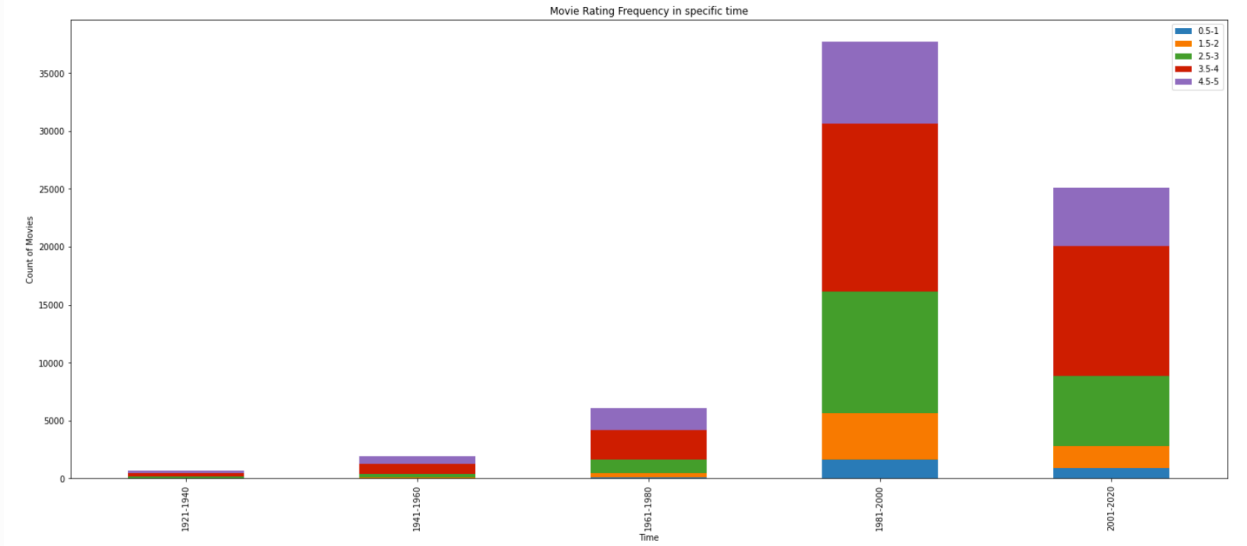
[9]


Percentage of Genres with Maximum rating
Percentage
0
5
10
15
20
Mystery
Thriller
Crime
Adventure
Comedy
Romance
Action
Drama
War
Western
Sci-Fi
Children
Animation
Fantasy
Musical
Film-Noir
Horror
Documentary
IMAX
Genres

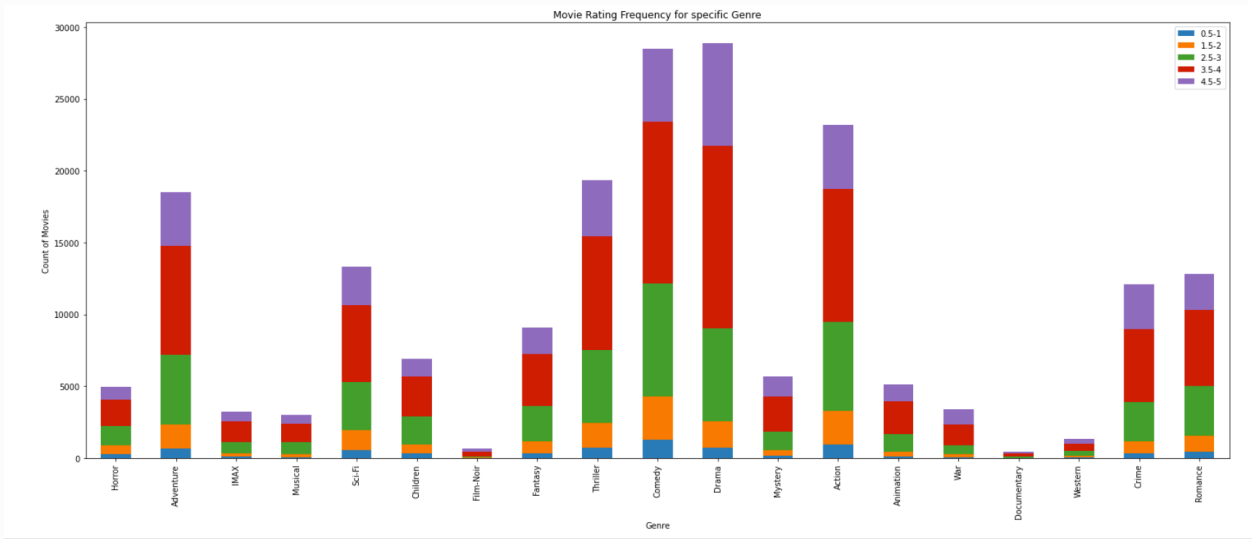
[10]


Percenatge of Genres with Minimum rating
Percentage
Genres
Mystery
Thriller
Crime
Adventure
Comedy
Romance
Action
Drama
War
Western
Sci-Fi
Children
Animation
Fantasy
Musical
Film-Noir
Horror
Documentary
IMAX

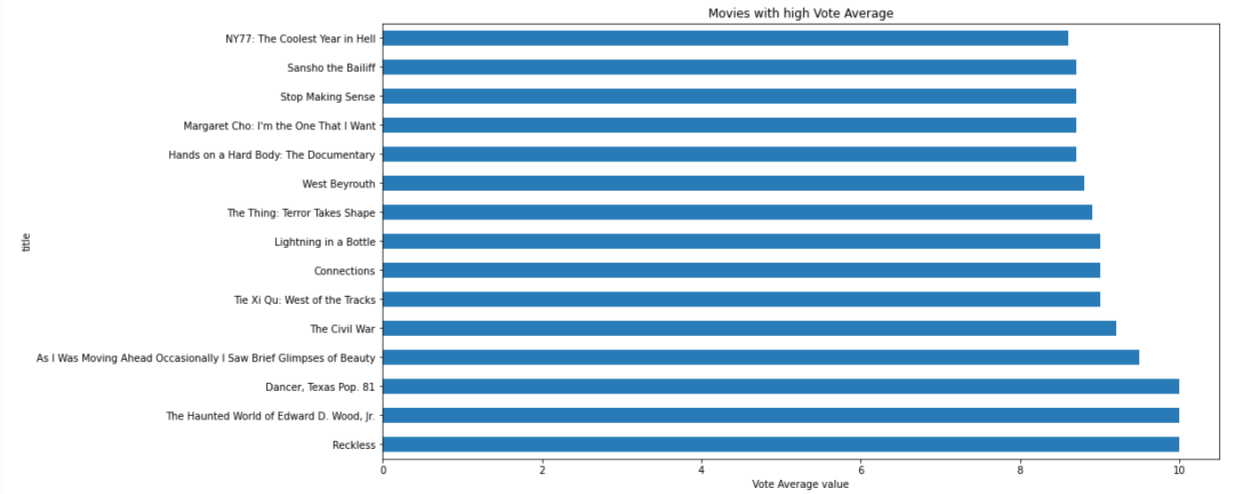
[11]


Movie Rating Frequency in specific time
Count of Movies
Time
0.5-1
1.5-2
2.5-3
3.5-4
4.5-5
1921-1940
1941-1960
1961-1980
1981-2000
2001-2020

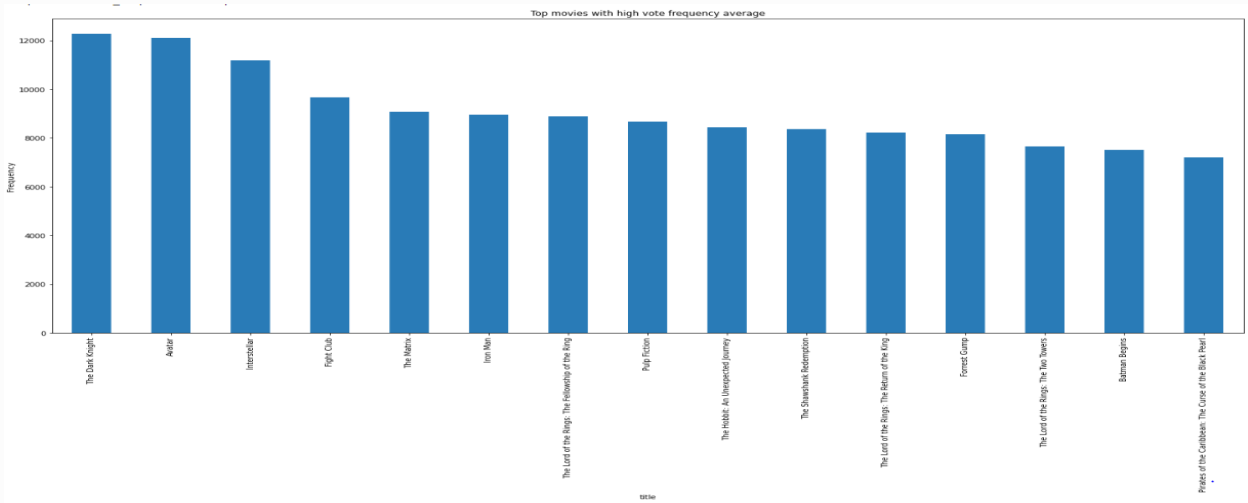
[12]


Movie Rating Frequency for specific Genre
0.5-1
1.5-2
2.5-3
3.5-4
4.5-5
Count of Movies
Genre
Horror
Adventure
IMAX
Musical
Sci-Fi
Children
Film-Noir
Fantasy
Thriller
Comedy
Drama
Mystery
Action
Animation
War
Documentary
Western
Crime
Romance

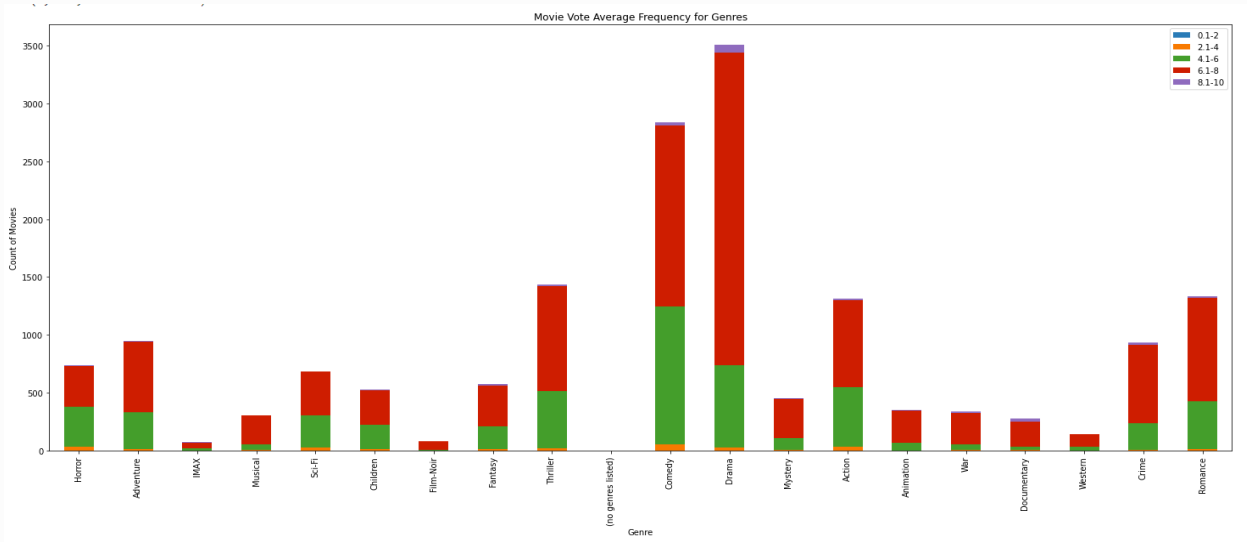
[13]


Movies with high Vote Average
title
NY77: The Coolest Year in Hell
Sansho the Bailiff
Stop Making Sense
Margaret Cho: I'm the One That I Want
Hands on a Hard Body: The Documentary
West Beyrouth
The Thing: Terror Takes Shape
Lightning in a Bottle
Connections
Tie Xi Qu: West of the Tracks
The Civil War
As I Was Moving Ahead Occasionally I Saw Brief Glimpses of Beauty
Dancer, Texas Pop. 81
The Haunted World of Edward D. Wood, Jr.
Reckless
Vote Average value

[14]


Top movies with high vote frequency average
Frequency
0
2000
4000
6000
8000
10000
12000
The Dark Knight
Avatar
Interstellar
Fight Club
The Matrix
Iron Man
The Lord of the Rings: The Fellowship of the Ring
Pulp Fiction
The Hobbit: An Unexpected Journey
The Shawshank Redemption
The Lord of the Rings: The Return of the King
Forrest Gump
The Lord of the Rings: The Two Towers
Batman Begins
Pirates of the Caribbean: The Curse of the Black Pearl
title

[15]


Movie Vote Average Frequency for Genres
Count of Movies
0
500
1000
1500
2000
2500
3000
3500
0.1-2
2.1-4
4.1-6
6.1-8
8.1-10
Horror
Adventure
IMAX
Musical
Sci-Fi
Children
Film-Noir
Fantasy
Thriller
(no genres listed)
Comedy
Drama
Mystery
Action
Animation
War
Documentary
Western
Crime
Romance
Genre

[16]

[17] Description passed to TdifVectorizer

```
.24] dfc['description'][0]
```

```
'Led by Woody, Andy's toys live happily in his room until Andy's birthday brings Buzz Lightyear onto the scene. Afraid of losing his pl
ace in Andy's heart, Woody plots against Buzz. But when circumstances separate Buzz and Woody from their owner, the duo eventually lear
ns to put aside their differences.Toy Story'
```

[18] We use sklearn's TfidfVectorizer as follows : (Refer to Appendix [18])

```
[125] from sklearn.feature_extraction.text import TfidfVectorizer
      tf = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words='english')
      tfidf_matrix = tf.fit_transform(dfc['description'])
```

[19] Sklearn's linear kernel to calculate cosine similarity

```
from sklearn.metrics.pairwise import linear_kernel
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
cosine_sim[0]
```

[20] Function that fetches movie recommendations

```
# Function that get movie recommendations based on the cosine similarity score of movie descriptions
def generate_recommendations(title):
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:31]
    movie_indices = [i[0] for i in sim_scores]
    return titles.iloc[movie_indices]
```

[21] Content Based Filtering Model results

```
generate_recommendations('Jumanji').head(20)
```

```
5218                        Big Game
6272                      Stay Alive
3899      The Giant Spider Invasion
3050             Dungeons & Dragons
5227      Night of the Living Dead
1379                     He Got Game
7303                     Geri's Game
1580                       Peter Pan
5701                         Nirvana
6205                   Grandma's Boy
3845                      Panic Room
3652                       Spy Game
2195           For Love of the Game
2003                        eXistenZ
4329                Poolhall Junkies
6161     Zathura: A Space Adventure
7367     Alan Partridge: Alpha Papa
1557                     BASEketball
4535        Spy Kids 3-D: Game Over
1029    Amityville: It's About Time
Name: title, dtype: object
```

```
[290] generate_recommendations('Batman Forever')
```

```
2469                Batman: Mask of the Phantasm
6798                             The Dark Knight
6014                               Batman Begins
1197                             Batman & Robin
517                                       Batman
1080                             Batman Returns
7368             Batman: Mystery of the Batwoman
5716    The Batman Superman Movie: World's Finest
5727          Batman Beyond: Return of the Joker
660                          Eyes Without a Face
6083                                    Cry_Wolf
2492                           Wayne's World 2
144                                     Hackers
1998                          Open Your Eyes
2579                                      JFK
6027                            At the Circus
6847                         The Incredible Hulk
2491                               Wayne's World
6150                                Just Friends
4529                             Loose Cannons
558                               DragonHeart
436                    The Man without a Face
4764                                  WarGames
4303                         They Call Me Bruce?
797                 Robin Hood: Prince of Thieves
1588                             The Shaggy D.A.
```

[22] Function to fetch weighted rating recommendations

```
def improved_recommendations(title):
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:26]
    movie_indices = [i[0] for i in sim_scores]
    global n
    global C
    movies = dfc.iloc[movie_indices][['title', 'vote_count', 'vote_average', 'year']]
    vote_counts = movies[movies['vote_count'].notnull()]['vote_count'].astype('int')
    vote_averages = movies[movies['vote_average'].notnull()]['vote_average'].astype('int')
    C = vote_averages.mean()
    n = vote_counts.quantile(0.60)
    qualified = movies[(movies['vote_count'] >= n) & (movies['vote_count'].notnull()) & (movies['vote_average'].notnull())]
    qualified['vote_count'] = qualified['vote_count'].astype('int')
    qualified['vote_average'] = qualified['vote_average'].astype('int')
    qualified['weightedrating'] = qualified.apply(weighted_rating, axis=1)
    qualified = qualified.sort_values('weightedrating', ascending=False).head(10)
    return qualified
```

[23] Improved recommendations output

	title	vote_count	vote_average	year	weightedrating
6798	The Dark Knight	12269	8	2008	7.919250
6014	Batman Begins	7511	7	2005	6.935694
1515	Back to the Future Part II	3926	7	1989	6.883796
517	Batman	2145	7	1989	6.806027
6562	Fracture	908	7	2007	6.637551
4764	WarGames	517	7	1983	6.500387
4439	Bruce Almighty	3121	6	2003	6.000000
1080	Batman Returns	1706	6	1992	6.000000
6847	The Incredible Hulk	3086	6	2008	6.000000
2491	Wayne's World	738	6	1992	6.000000
558	DragonHeart	553	6	1996	6.000000
797	Robin Hood: Prince of Thieves	937	6	1991	6.000000
6189	Fun with Dick and Jane	639	5	2005	5.446849
1197	Batman & Robin	1447	4	1997	4.525876

	title	vote_count	vote_average	year	weightedrating
6445	Casino Royale	3930	7	2006	6.900262
1580	Peter Pan	1380	7	1953	6.748169
5495	Love Me If You Dare	531	7	2003	6.488445
7303	Geri's Game	309	7	1997	6.299548
6171	The Chronicles of Narnia: The Lion, the Witch ...	2709	6	2005	5.957047
3845	Panic Room	1303	6	2002	5.919188
6161	Zathura: A Space Adventure	808	6	2005	5.882829
2819	The Running Man	713	6	1987	5.871755
3652	Spy Game	592	6	2001	5.854205
2003	eXistenZ	487	6	1999	5.834559
2442	Any Given Sunday	430	6	1999	5.821501
3195	Manhunter	340	6	1986	5.796090
6075	The Brothers Grimm	847	5	2005	5.143336
4535	Spy Kids 3-D: Game Over	525	4	2003	4.557377

[24]

```
accuracy.rmse(baselinepreds)
accuracy.mae(baselinepreds)
```

```
RMSE: 0.8394
MAE:  0.6400
0.6400094733419025
```

[25]

```
[150] #Print best score and best parameters from the GridSearch
      print(gridsvd.best_score)
      print(gridsvd.best_params)
```

```
{'rmse': 0.823981994341997, 'mae': 0.6304605498938514}
{'rmse': {'n_factors': 80, 'reg_all': 0.06, 'n_epochs': 30, 'lr_all': 0.01}, 'mae': {'n_factors': 80, 'reg_all': 0.06, 'n_epochs': 30, 'lr_all': 0.01}}
```

[26]

```
#Print RMSE and MAE results
accuracy.rmse(svdpreds)
accuracy.mae(svdpreds)
```

```
RMSE: 0.8278
MAE:  0.6305
0.6305203063892995
```

[27]

```
[159] #Display the best scores and parameters from GridSearch
      print(gsknnbasic.best_score)
      print(gsknnbasic.best_params)
```

```
{'rmse': 0.9151500003405437, 'mae': 0.7058652021103266}
{'rmse': {'name': 'cosine', 'user_based': True, 'min_support': True, 'min_k': 2}, 'mae': {'name': 'cosine', 'user_based': True, 'min_support': True, 'min_k': 2}}
```

```
[162] #Print RMSE and MAE results
      accuracy.rmse(knnbpreds)
      accuracy.mae(knnbpreds)

      RMSE: 0.9331
      MAE:  0.7182
      0.7181573300402186
```

[28]

```
[308] #Display the best score and the best parameters
      print(gsknnbaseline.best_score)
      print(gsknnbaseline.best_params)

      {'rmse': 0.8441191532485849, 'mae': 0.6461472666530459}
      {'rmse': {'name': 'cosine', 'user_based': True, 'min_support': True, 'min_k': 2}, 'mae': {'name': 'cosine', 'user_based': True, 'min_support': True, 'min_k': 2}}
```

```
[311] #Print the RMSE and MAE scores
      accuracy.rmse(knnbaselinepreds)
      accuracy.mae(knnbaselinepreds)

      RMSE: 0.8434
      MAE:  0.6432
      0.6432064638150256
```

[29]

model	RMSE	MAE	CV
baseline	0.839416	0.640009	0.844745
svd	0.827832	0.630520	0.835998
knnbasic	0.933129	0.718157	0.937588
knnbaseline	0.843430	0.643206	0.848062

[30]

Ratings for user 610 :

```
[178] generate(610,10)
```

	userId	movieId	estimatedrating	title	genres	year
0	610	858	4.787480	Godfather, The	[Crime, Drama]	1972
1	610	720	4.756982	Wallace & Gromit: The Best of Aardman Animation	[Adventure, Animation, Comedy]	1996
2	610	541	4.731696	Blade Runner	[Action, Sci-Fi, Thriller]	1982
3	610	4226	4.717029	Memento	[Mystery, Thriller]	2000
4	610	296	4.704554	Pulp Fiction	[Comedy, Crime, Drama, Thriller]	1994
5	610	1178	4.700238	Paths of Glory	[Drama, War]	1957
6	610	1221	4.689095	Godfather: Part II, The	[Crime, Drama]	1974
7	610	57669	4.682130	In Bruges	[Comedy, Crime, Drama, Thriller]	2008
8	610	3451	4.682016	Guess Who's Coming to Dinner	[Drama]	1967
9	610	1213	4.659128	Goodfellas	[Crime, Drama]	1990

Ratings for user 10 :

```
[317] generate(10,20)
```

	userId	movieId	estimatedrating	title	genres	year
0	10	104879	4.710111	Prisoners	[Drama, Mystery, Thriller]	2013
1	10	79091	4.455791	Despicable Me	[Animation, Children, Comedy, Crime]	2010
2	10	49286	4.435494	Holiday, The	[Comedy, Romance]	2006
3	10	2324	4.422262	Life Is Beautiful (La Vita è bella)	[Comedy, Drama, Romance, War]	1997
4	10	33794	4.364698	Batman Begins	[Action, Crime, IMAX]	2005
5	10	7458	4.361341	Troy	[Action, Adventure, Drama, War]	2004
6	10	96079	4.324363	Skyfall	[Action, Adventure, Thriller, IMAX]	2012
7	10	2064	4.286749	Roger & Me	[Documentary]	1989
8	10	527	4.285837	Schindler's List	[Drama, War]	1993
9	10	2150	4.281553	Gods Must Be Crazy, The	[Adventure, Comedy]	1980
10	10	951	4.261411	His Girl Friday	[Comedy, Romance]	1940
11	10	1217	4.260595	Ran	[Drama, War]	1985
12	10	81845	4.257655	King's Speech, The	[Drama]	2010

- End of Document -