Exploratory Data Analysis - CSE5007
Project Report

Movie Recommendation System Using Item Based Collaborative Filtering.

Done By:

Bijoya Mondal
(21MCB0005)

Prajwal R M
(21MCB0012)

Sridhar M
(21MCB0016)

TABLE OF CONTENTS

1. ABSTRACT

The availability of movies in various genre over the decade has drastically increased, overwhelming the users on any movie streaming platform to find movies they are interested in, a time consuming task, this manual work of the users to find their choice of interest in movies can be minimized with the use of recommendation systems, these systems use the users streamed history data-set to analyze the data and use different filtering methods to predict and suggest the user movies similar to their interest, the caviar of these recommendation systems is that they should be accurate and relevant with the help of concepts of exploratory data analysis (EDA) and prediction based on the user's history, else it is just a system recommending random movies, which is a redundant functionality for a user with a particular taste. This paper offers a solution to this problem by obtaining the users data based on their ratings given to a movie and preferences, and further analysis is performed based on this data with the use of different techniques like content-based filtering, collaborative based filtering etc., to understand the preferences of a user and recommend similar movies based on other's review and the user's choice.

Keywords:
Recommendation system, filtering, prediction, content-based filtering, collaborative based filtering.

2. INTRODUCTION

In the current age of the internet era the availability of different options is abundant, for a user streaming movies on any platform the amount of genre provided to view is uncanny and not all users would want to watch a specific type of genre, each of them have different preferences, to make sure each user is able to view a movie based on their liking and interest without the manual hustle task of scrapping through the Internet, recommendation systems are required. Recommendation systems are a sort of filtering tools, the key components required to perform analysis and recommend a preferred genre movie are the viewing history of the users and their the list of genres they have priorly liked, with these key components with the help of EDA(exploratory data analysis) and filtering algorithms the movies which the user might like can be predicted and suggested with good accuracy.

To perform these predictions different types of recommendation systems and algorithms can be used, the popular types of recommendation systems in use are collaborative and content based recommender systems. Collaborative recommender system is a filter that is a family of algorithms where there are multiple ways to find similar users or items and multiple ways to calculate rating based on ratings of similar users. The nearest neighbourhood algorithm is used a base to perform collaborative recommender system. The content based filtering process works with data that the user provides, either explicitly (rating) or implicitly (clicking on a link).Based on that data, a user profile is generated, which is then used to make suggestions to the user. The content based recommendation system uses two different algorithms to approach the recommendation problem one is the vector spacing method and the other is the classification model method.

The data processing steps to be performed for the recommendation system:

1. The information on the users and movies should be collected and organized.
2. Compare a user to all other users to find the group of users whose preferences and interests matches with that particular user.
3. A program or function should to be created to find the movies that the user has not seen but users similar to the user have watched.
4. The list of movies collected buy the program should be ranked in order of interest with respect to the user and then recommended.
5. The model should be evaluated based on the results and then tested again to improve the accuracy of the recommendation.

2.1. What is a Recommendation System?

Simply put a **Recommendation System** is a filtration program whose prime goal is to predict the “rating” or “preference” of a user towards a domain-specific item or item. In our case, this domain-specific item is a movie, therefore the main focus of our recommendation system is to filter and predict only those movies which a user would prefer given some data about the user him or herself.

2.2. What are the different filtration strategies?

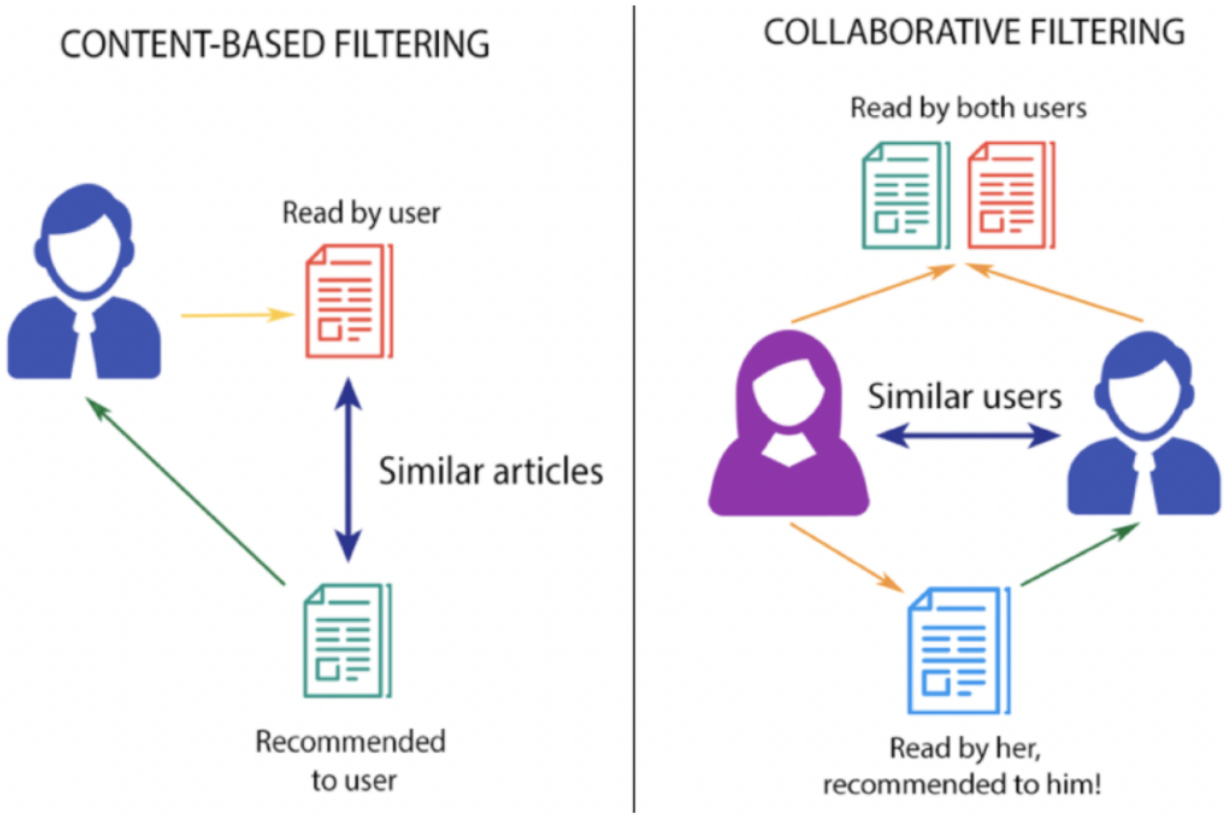

Figure 1 Types of Filtration Techniques

- ## Content-Based Filtering

This filtration strategy is based on the data provided about the items. The algorithm recommends products that are **similar** to the ones that a user has liked in the **past**. This similarity (generally cosine similarity) is computed from the data we have about the items as well as the user's past preferences. For example, if a user likes movies such as 'The Prestige' then we can recommend him the movies of 'Christian Bale' or movies with the genre 'Thriller' or maybe even movies directed by 'Christopher Nolan'. So what happens here the recommendation system checks the past preferences of the user and find the film "The Prestige", then tries to find similar movies to that using the information available in the database such as the lead actors, the director, genre of the film, production house, etc and based on this information find movies similar to "The Prestige".

Disadvantages:

- Different products do not get much **exposure** to the user.
- Businesses cannot be expanded as the user does not try different types of products.

- ## Collaborative Filtering

This filtration strategy is based on the combination of the user's behaviour and comparing and contrasting that with **other users'** behaviour in the database. The history of **all users** plays an important role in this algorithm. The main difference between content-based filtering and collaborative filtering that in the latter, the **interaction of all users with the items** influences the recommendation algorithm while for content-based filtering only the **concerned user's data** is taken into account.

There are multiple ways to implement collaborative filtering but the main concept to be grasped is that in collaborative filtering **multiple** user's data influences the outcome of the recommendation. and doesn't depend on **only one user's data** for modelling.

There are 2 types of collaborative filtering algorithms:

1. **User-based Collaborative filtering**

   The basic idea here is to find users that have **similar past preference patterns** as the user 'A' has had and then recommending him or her items liked by those similar users which 'A' has not encountered yet. This is achieved by making a matrix of items each user has rated/viewed/liked/clicked depending upon the task at hand, and then computing the similarity score between the users and finally recommending items that the concerned user isn't aware of but users similar to him/her are and liked it.

   For example, if the user 'A' likes 'Batman Begins', 'Justice League' and 'The Avengers' while the user 'B' likes 'Batman Begins', 'Justice League' and 'Thor' then they have similar interests because we know that these movies belong to the super-hero genre. So, there is a high probability that the user 'A' would like 'Thor' and the user 'B' would like The Avengers'.

   Disadvantages:

   - People are **fickle-minded** i.e. their taste change from time to time and as this algorithm is based on user similarity it may pick up initial similarity patterns between 2 users who after a while may have completely different preferences.
   - There are many **more users than items** therefore it becomes very difficult to maintain such large matrices and therefore needs to be recomputed very regularly.
   - This algorithm is very susceptible to **shilling attacks** where fake users profiles consisting of biased preference patterns are used to manipulate key decisions.

2. Item Based Collaborative Filtering

The concept in this case is to **find similar movies instead of similar users** and then recommending similar movies to that ‘A’ has had in his/her past preferences. This is executed by finding every pair of items that were rated/viewed/liked/clicked by the same user, then measuring the similarity of those rated/viewed/liked/clicked across all user who rated/viewed/liked/clicked both, and finally recommending them based on similarity scores.

Here, for example, we take 2 movies ‘A’ and ‘B’ and check their ratings by all users who have rated both the movies and based on the similarity of these ratings, and based on this rating similarity by users who have rated both we find similar movies. So if most common users have rated ‘A’ and ‘B’ both similarly and it is highly probable that ‘A’ and ‘B’ are similar, therefore if someone has watched and liked ‘A’ they should be recommended ‘B’ and vice versa.

Advantages over User-based Collaborative Filtering

- Unlike people’s taste, **movies don’t change**.
- There are usually a lot **fewer items than people,** therefore easier to maintain and compute the matrices.
- Shilling attacks are much harder because **items cannot be faked**.

3. LITERATURE SURVEY

- **Movie Recommendation System**

  Authors: S. Rajarajeswari, Sharat Naik, Shagun Srikant, M.K. Sai Prakash and Prarthana Uday.

  This paper the recommendation system is implemented using both collaborative as well as content-based recommenders available and tried to extend the knowledge to obtain a more efficient hybrid model.

- **Content-Based Movie Recommendation System Using Genre Correlation**

  Authors: SRS Reddy, Sravani Nalluri, Subramanyam Kunisetti,S. Ashok and B. Venkatesh:

  In this paper, the recommendation system has-been built on the type of genres that the user might prefer to watch. The approach adopted to do so is content-based filtering using genre correlation. The dataset used for the system is Movie Lens dataset.

- **Movie Recommendation System Using Sentiment Analysis From Microblogging Data**

  Authors: Sudhanshu Kumar , Kanjar De, and Partha Pratim Roy:

  This paper proposes a hybrid recommendation system for the movies that leverage the best of concepts used from collaborative filtering and content based filtering along with sentiment analysis of tweets from micro-blogging sites. The purpose to use movie tweets is to understand the current trends, public sentiment, and user response of the movie.

- **Movie Recommendation System: Hybrid Information Filtering System**

  Authors: Kartik Narendra Jain, Vikrant Kumar, Praveen Ku-mar and Tanupriya Choudhury:

  The system proposed in this paper conforms a different approach where it seeks the similarity of users among others clustered around the various genres and utilize his/her preference of movies based on their content in terms of genres as the deciding factor of the recommendation of the movies to them. The system is based on the belief theta user rates movies in a similar fashion to other users that harbour the same state as the current user and is also affected by the other activities (in terms of rating) he performs with other movies. It follows the hypothesis that a user can be accurately recommended media on the basis others interests(collaborative filtering) and the movies themselves (content-based filtering).

- **Movie Recommender System Based on Collaborative Filtering Using Apache Spark:**

  Authors: Mohammed Fadhel Aljunid and D. H. Manjaiah:

  The paper proposes a method to use collaborative filtering with alternating least squares(ALS) algorithm for a movie recommendation system.. The ALS algorithm is one of the models of matrix factorization related CF which is considered as the values in the item list of user matrix. As there is a need to perform analysis on the ALS algorithm by selecting different parameters which can eventually help in building efficient movie recommender engine A movie recommender system based on ALS using Apache spark is suggested, the research focuses on the selection of parameters of ALS algorithms that can affect the performance of a building robust RS. From the results, a conclusion is drawn according to the selection of parameters of ALS algorithms which can affect the performance of building of a movie recommender engine. The model evaluation is done using different metrics such as execution time, root mean squared error (RMSE) of rating prediction, and rank in which the best model was trained.

- **A Movie Recommender System Using Modified Cuckoo Search:**

  Authors: Suraj Pal Singh and Shano Solank:

  The paper proposes how a movie recommender system can be programmed by using data clustering and nature-inspired algorithm. K-means algorithm is used for clustering , handling large amount of data but the running time is low. But it falls into local optima due to its randomly generated initial centroids. This algorithm can achieve global optimum solution if it is integrated with nature-inspired algorithm. This paper integrates k-means with nature-inspired algorithms(bat, firefly, cuckoo, modified cuckoo search) on data set. Outcomes are compared on the basis of objective function, less the objective function better the results.

- **Movie Recommendation System Using Genome Tagsand Content-Based Filtering**:

  The paper propose a hybrid model which utilizes genomic tags of movie coupled with the content based filtering to recommend similar movies. The algorithm uses principal component analysis (PCA) and Pearson correlation techniques to reduce the tags which are redundant and show low proportion of variance, hence reducing the computation complexity. The initial results conducted prove that the genomic tags give the better result in terms of finding similar type of movies, and give more accurate and personalized recommendation as compared to existing models.

- **Personalized Real-Time Movie Recommendation System:**

  Authors: Jiang Zhang, Yufeng Wang , Zhiyuan Yuan, and QunJin

  Practical Prototype and Evaluation Scalability and practical usage feedback and verification based on real time implementation, collaborative filtering prevailing technique for implementing recommendation systems(traditional CF is suffers from a time complexity problem) , high efficient algorithm is proposed, it exploits users profile attributes to partition them into several clusters. For each cluster, a virtual opinion leader is conceived to represent the whole cluster, such that the dimension of the original user item matrix can be significantly reduced, then a Weighted Slope One-VU method is designed and applied to the virtual opinion leader-item matrix to obtain the recommendation results. Compared to traditional clustering based CF recommendation schemes, our method can significantly reduce the time complexity, while achieving comparable recommendation performance.

- **Multilingual Opinion Mining Movie Recommendation System Using RNN**

  Authors: Tarana Singh, Anand Nayyar and Arun Solanki

  The paper talks about how it is using twitter's big data for implementing a movie recommendation system, and how they have used sentimental analysis to the data to improve the performance of the framework, a movie recommendation system is developed using real time multilingual tweets. The tweets are classified as positive, negative, and neutral tweets. Prepossessing of the tweets is done to remove unwanted words, URLs, emoticons, etc. Based on the classification the movie is suggested to the user.

- **Survey on Collaborative Filtering, Content-based Filtering and Hybrid Recommendation System**

  Authors: Poonam B. Thorat , R. M. Goudar , Sunita Barve

  This paper provides an overview of recommender systems that include collaborative filtering, content-based filtering and hybrid approach of recommender system.

- **Movie Recommendation System using Cosine Similarity and KNN**

  Authors: Ramni Harbir Singh, Sargam Maurya, Tanisha Tri-pathi, Tushar Narula, Gaurav Srivastav

  Content-based filtering and Cosine Similarity is used to predict and recommend a movie to the user. Mathematically, Cosine Similarity shows the cosine of the angle of two vectors projected in a multidimensional space .The angle theta between the two movies will determine the similarity between the two movies. If the value of the theta is near 1then it is most similar and if it's near to 0 then it is least similar. The movie will be recommended if it is close to 1 otherwise there would be no similarity between them. Then by using the KNN functionality, we have found the nearest neighbour which will be recommended to the user.

- **Hybrid Model for Movie Recommendation System Using Fireflies and Fuzzy C-Means**

  Authors: M. Sandeep Kumar and Prabhu J.

  Firefly algorithm and Fuzzy c-means is used to design a recommender system. The firefly algorithm performs efficiently in providing quality results in the optimization process, as it compares with other meta-heuristic approaches. The Fuzzy c-means is familiar clustering method in which it is employed in Movie lens dataset.

- **Deep Learning Based Recommender System: A Survey and New Perspectives**

  Authors: Shuai Zhang and Lina Yao, University of New SouthWales Aixin Suun and Yi Tay, Nanyang Technological University

  In this paper, several Deep Learning methods have been proposed to design recommender systems.
  They are:
  •The Multilayer Perceptron
  •An Autoencoder
  •The Convolutional Neural Network
  •The Recurrent Neural Network
  •The Restricted Boltzmann Machine
  •Neural Auto aggressive Distribution Estimation
  •The Adversarial Network

- **A hybrid model collaborative movie recommendation system using K-means clustering with ant colony optimisation**

  Authors: M. Sandeep Kumar and J. Prabhu Department of Department of Software System and Engineering, School of Information Technology and Engineering, Vellore Institute of Technology

  In this paper, a hybrid model collaborative movie recommendation system that performs with a combination of K-means clustering with ant colony optimisation technique (ACO-KM)is proposed that has employed in movie dataset. The evaluation process of movie recommendation system that offers improved result from ACO-KM collaborative movie recommender system based on precision, recall, mean square error(MSE), and accuracy. By comparison of speed (in seconds)of various approaches in Movie lens dataset, their approach gives best result 42.24 compared with existing

one 53.22.The outcome of this experiment from Movie lens dataset that offers scalability and efficiency in a recommendation by de-creasing cold start issues.

- **A Hybrid Movie Recommendation System Using Graph-Based Approach**

  Authors: Göksu Tüysüzoğlu and Zerrin Işık, Graduate Schooled Natural and Applied Sciences, Dokuz Eylul University, Izmir, Turkey Department of Computer Engineering, Dokuz Eylul University, Izmir, Turkey

  The proposed system, a content-based movie recommendation is automatically made using a graph based approach according to past film preferences of users between 1995 and2016; using demographic information of users, the recommendation list is updated. A combination of outputs of these two techniques reveals more precise recommendations concerning movies. The Movie Lens dataset was used to explore the proposed hybrid system.

4. DATASET

We are going to use the Movie Lens Data Set (Small). This dataset was put together by the Group lens research group at the University of Minnesota. It contains 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users.

DATASET SAMPLE :

```
import pandas as pd
import numpy as np
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.model_selection import cross_val_score
from sklearn.metrics.pairwise import cosine_similarity, cosine_distances
movies = pd.read_csv("/Users/sridhar/Downloads/ml-latest-small/movies.csv")
ratings = pd.read_csv("/Users/sridhar/Downloads/ml-latest-small/ratings.csv")
```

```
movies.head()
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

Figure 2 movies dataset

```
movies.shape
```

```
(9742, 3)
```

Figure 3 Size of movie dataset

```
ratings.head()
```

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

Figure 4 Ratings Dataset

```
ratings.shape
```

```
(100836, 4)
```

Figure 5 Size of Ratings Dataset

5. Walkthrough of building a recommender system

In this implementation, when the user searches for a movie we will recommend the top 10 similar movies using our movie recommendation system. We will be using **an item-based collaborative filtering** algorithm for our purpose.

5.1. Data Cleaning

Here, in the ratings dataset we can see that userId 1 has **watched** movieId 1 & 3 and rated both of them 4.0 but has **not rated** movieId 2 at all. This interpretation is harder to extract from this dataframe. Therefore, to make things easier to understand and work with, we are going to make a new dataframe where each column would represent each unique userId and each row represents each unique movieId.

```
final_dataset = ratings.pivot(index='movieId',columns='userId',values='rating')
final_dataset.head()
```

```
userId   1    2    3    4    5    6    7    8    9    10   ...  601  602  603  \
movieId                                                     ...
1        4.0  NaN  NaN  NaN  4.0  NaN  4.5  NaN  NaN  NaN  ...  4.0  NaN  4.0
2        NaN  NaN  NaN  NaN  NaN  4.0  NaN  4.0  NaN  NaN  ...  NaN  4.0  NaN
3        4.0  NaN  NaN  NaN  NaN  5.0  NaN  NaN  NaN  NaN  ...  NaN  NaN  NaN
4        NaN  NaN  NaN  NaN  NaN  3.0  NaN  NaN  NaN  NaN  ...  NaN  NaN  NaN
5        NaN  NaN  NaN  NaN  NaN  5.0  NaN  NaN  NaN  NaN  ...  NaN  NaN  NaN

userId   604  605  606  607  608  609  610
movieId
1        3.0  4.0  2.5  4.0  2.5  3.0  5.0
2        5.0  3.5  NaN  NaN  2.0  NaN  NaN
3        NaN  NaN  NaN  NaN  2.0  NaN  NaN
4        NaN  NaN  NaN  NaN  NaN  NaN  NaN
5        3.0  NaN  NaN  NaN  NaN  NaN  NaN

[5 rows x 610 columns]
```

Figure 6 Final Dataset after using pivot function

Now, it's much easier to interpret that userId 1 has rated movieId 1& 3 4.0 but has not rated movieId 3,4,5 at all (therefore they are represented as NaN) and therefore their rating data is missing.

Impute NaN with 0 to make things understandable for the algorithm and also making the data more eye-soothing.

```
final_dataset.fillna(0,inplace=True)
final_dataset.head()
```

```
userId   1    2    3    4    5    6    7    8    9    10   ...  601  602  603  \
movieId                                                     ...
1        4.0  0.0  0.0  0.0  4.0  0.0  4.5  0.0  0.0  0.0  ...  4.0  0.0  4.0
2        0.0  0.0  0.0  0.0  0.0  4.0  0.0  4.0  0.0  0.0  ...  0.0  4.0  0.0
3        4.0  0.0  0.0  0.0  0.0  5.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
4        0.0  0.0  0.0  0.0  0.0  3.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
5        0.0  0.0  0.0  0.0  0.0  5.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0

userId   604  605  606  607  608  609  610
movieId
1        3.0  4.0  2.5  4.0  2.5  3.0  5.0
2        5.0  3.5  0.0  0.0  2.0  0.0  0.0
3        0.0  0.0  0.0  0.0  2.0  0.0  0.0
4        0.0  0.0  0.0  0.0  0.0  0.0  0.0
5        3.0  0.0  0.0  0.0  0.0  0.0  0.0

[5 rows x 610 columns]
```

Figure 7 Final Dataset after Zero Imputation

5.2. Removing Noising from the data

In the real-world, ratings are very sparse and data points are mostly collected from very popular movies and highly engaged users. We wouldn't want movies that were rated by a small number of users because it's not credible enough. Similarly, users who have rated only a handful of movies should also not be taken into account.

So with all that taken into account and some trial and error experimentations, we will reduce the noise by adding some filters for the final dataset.

- To qualify a movie, a minimum of 10 users should have voted a movie.
- To qualify a user, a minimum of 50 movies should have voted by the user.

5.3. Visualizing Filters Applied

Aggregating the number of users who voted and the number of movies that were voted.

```
no_user_voted = ratings.groupby('movieId')['rating'].agg('count')
no_movies_voted = ratings.groupby('userId')['rating'].agg('count')
print(no_user_voted)
print(no_movies_voted)
```

```
movieId
1        215
2        110
3         52
4          7
5         49
        ...
193581     1
193583     1
193585     1
193587     1
193609     1
Name: rating, Length: 9724, dtype: int64
userId
1      232
2       29
3       39
4      216
5       44
      ...
606    1115
607     187
608     831
609      37
610    1302
Name: rating, Length: 610, dtype: int64
```

Figure 8 Aggregating MovieId and UserID based in the ratings

Let's visualize the number of users who voted with our threshold of 10.

```
f,ax = plt.subplots(1,1,figsize=(20,16))
#ratings['rating'].plot(kind='hist')
plt.scatter(no_user_voted.index,no_user_voted,color='cyan')
plt.axhline(y=10,color='r')
plt.xlabel('MovieId')
plt.ylabel('No. of users voted')
plt.show()
```

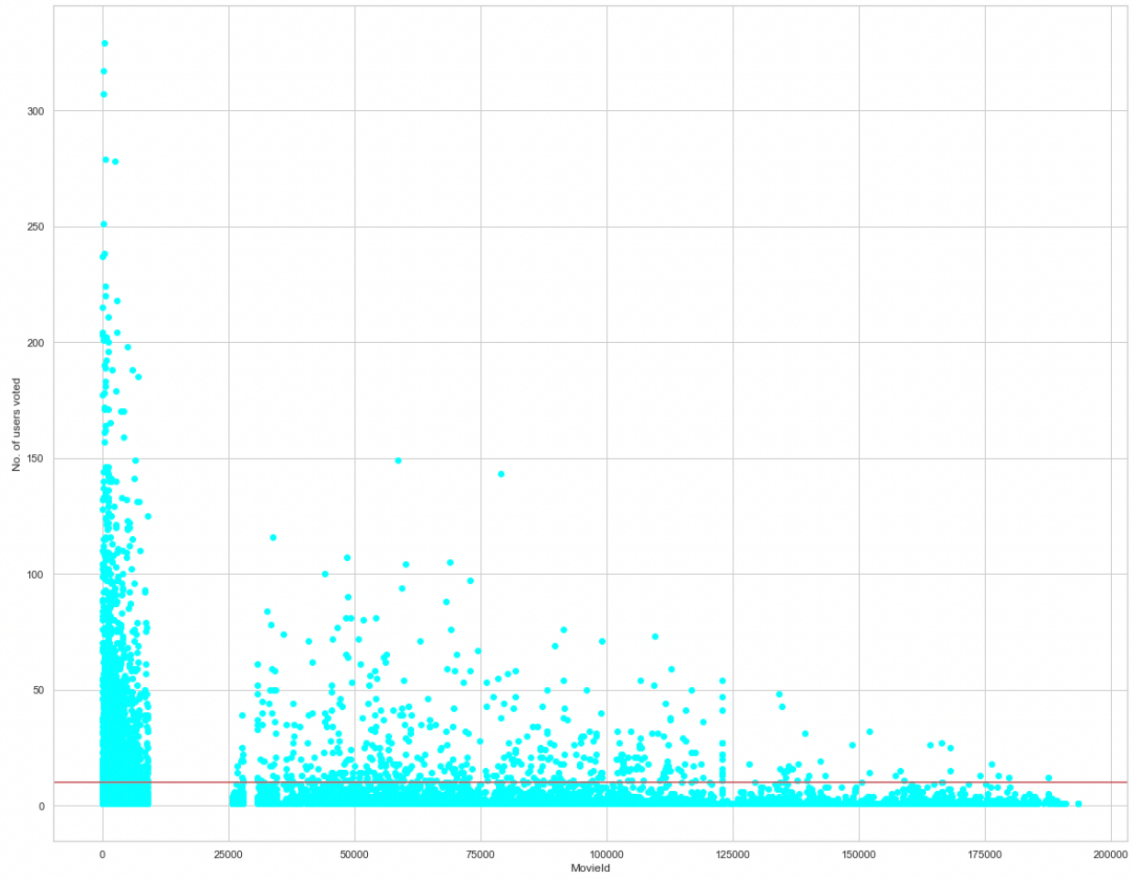

Figure 9 Visualizing no.of Movies voted with a threshold of 10

Making the necessary modifications as per the threshold set.

```
final_dataset = final_dataset.loc[no_user_voted[no_user_voted > 10].index,:]
```

Let's visualize the number of votes by each user with our threshold of 50.

```
f,ax = plt.subplots(1,1,figsize=(20,16))
plt.scatter(no_movies_voted.index,no_movies_voted,color='cyan')
plt.axhline(y=50,color='r')
plt.xlabel('UserId')
plt.ylabel('No. of votes by user')
plt.show()
```

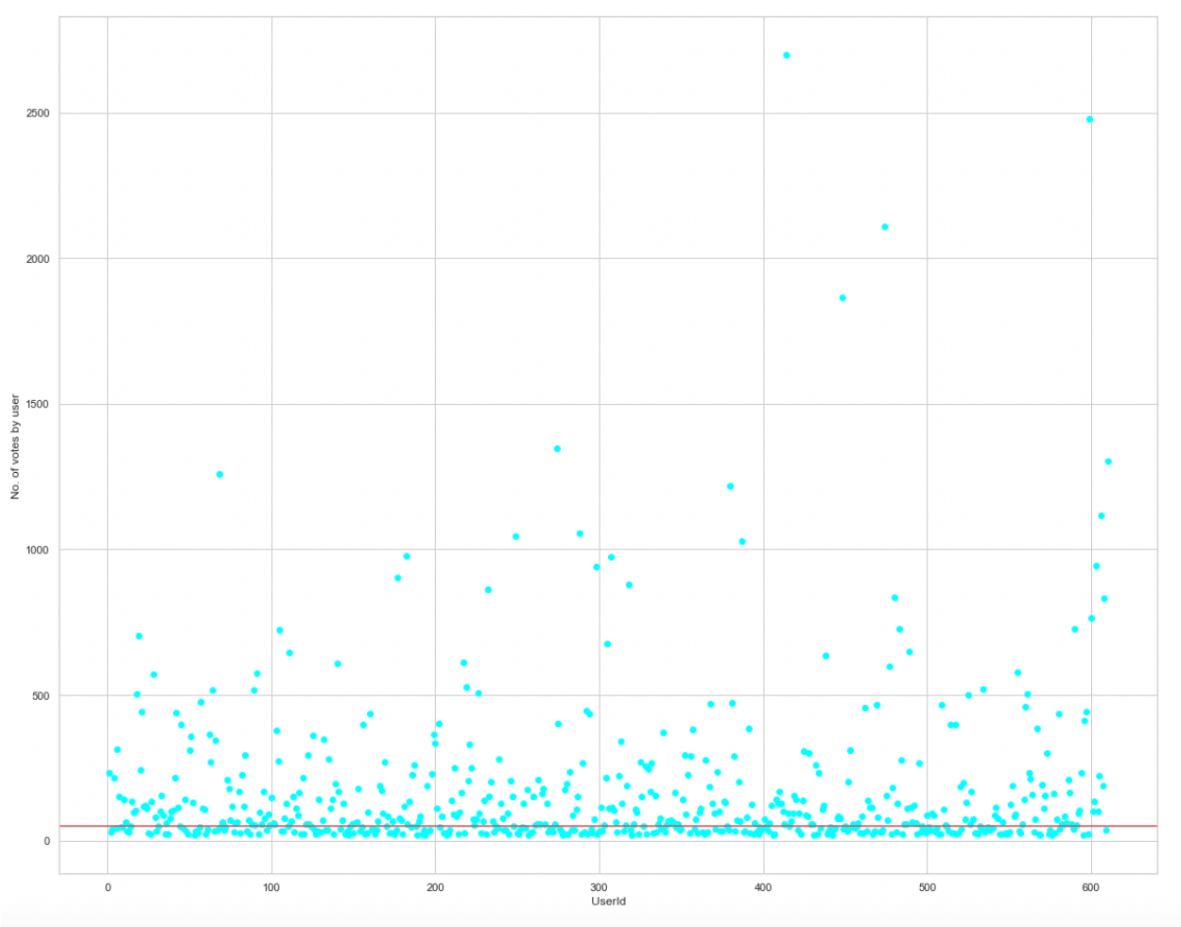

Figure 10 Visualizing no.of Users by their votes with a threshold of 50

Making the necessary modifications as per the threshold set.

```
final_dataset=final_dataset.loc[:,no_movies_voted[no_movies_voted > 50].index]
final_dataset
```

5.4. Removing Sparsity

Our final_dataset has dimensions of **2121 * 378** where most of the values are sparse. We are using only a small dataset but for the original large dataset of movie lens which has more than **100000** features, our system may run out of computational resources when that is feed to the model. To reduce the sparsity we use the csr_matrix function from the scipy library.

Applying the csr_matrix method to the dataset :

```
csr_data = csr_matrix(final_dataset.values)
final_dataset.reset_index(inplace=True)
final_dataset.head()
```

```
userId  index  movieId    1    4    6    7   10   11   15   16  ...  600  601  \
0           0        1  4.0  0.0  0.0  4.5  0.0  0.0  2.5  0.0  ...  2.5  4.0
1           1        2  0.0  0.0  4.0  0.0  0.0  0.0  0.0  0.0  ...  4.0  0.0
2           2        3  4.0  0.0  5.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0
3           3        5  0.0  0.0  5.0  0.0  0.0  0.0  0.0  0.0  ...  2.5  0.0
4           4        6  4.0  0.0  4.0  0.0  0.0  5.0  0.0  0.0  ...  0.0  0.0

userId  602  603  604  605  606  607  608  610
0       0.0  4.0  3.0  4.0  2.5  4.0  2.5  5.0
1       4.0  0.0  5.0  3.5  0.0  0.0  2.0  0.0
2       0.0  0.0  0.0  0.0  0.0  0.0  2.0  0.0
3       0.0  0.0  3.0  0.0  0.0  0.0  0.0  0.0
4       3.0  4.0  3.0  0.0  0.0  0.0  0.0  5.0

[5 rows x 380 columns]
```

Figure 11 Applying csr_matrix method to the Final Dataset

5.5. Making the movie recommendation system model

We will be using the KNN algorithm to compute similarity with cosine distance metric which is very fast and more preferable than pearson coefficient.

- **Cosine Similarity:**

  Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. It is often used to measure document similarity in text analysis.

```
cos_sim_1_2 = cosine_similarity([arr[0, :], arr[5, :]])
print('Cosine Similarity is \n',cos_sim_1_2 )
```

```
Cosine Similarity is
 [[1.         0.24997777]
 [0.24997777 1.        ]]
```

Figure 12 Cosine Similarity Example

- **K – Nearest Neighbour:**

  The k-nearest neighbours algorithm is a non-parametric classification method first developed by Evelyn Fix and Joseph Hodges in 1951, and later expanded by Thomas Cover. It is used for classification and regression. In both cases, the input consists of the k closest training examples in a data set.

```
knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=10,
n_jobs=-1)
model = knn.fit(csr_data)
```

- **Making The Recommendation Function:**

  The working principle is very simple. We first check if the movie name input is in the database and if it is we use our recommendation system to find similar movies and sort them based on their similarity distance and output only the top 10 movies with their distances from the input movie.

```
def get_movie_recommendation(movie_name):
    n_movies_to_reccomend = 10
    movie_list = movies[movies['title'].str.contains(movie_name)]
    if len(movie_list):
        movie_idx= movie_list.iloc[0]['movieId']
        movie_idx = final_dataset[final_dataset['movieId'] ==
movie_idx].index[0]
        distances , indices =
knn.kneighbors(csr_data[movie_idx],n_neighbors=n_movies_to_reccomend+1)
        rec_movie_indices =
sorted(list(zip(indices.squeeze().tolist(),distances.squeeze().tolist())),key=
lambda x: x[1])[:0:-1]
        recommend_frame = []
        for val in rec_movie_indices:
            movie_idx = final_dataset.iloc[val[0]]['movieId']
            idx = movies[movies['movieId'] == movie_idx].index

recommend_frame.append({'Title':movies.iloc[idx]['title'].values[0],'Distance'
:val[1]})
        df =
pd.DataFrame(recommend_frame,index=range(1,n_movies_to_reccomend+1))
        return df
    else:
        return "No movies found. Please check your input"
```

- **Working of Recommendation System:**

```
get_movie_recommendation('Iron Man')
```

	Title	Distance
1	X-Men: First Class (2011)	1.111104e-07
2	Guardians of the Galaxy (2014)	1.106718e-07
3	District 9 (2009)	1.085074e-07
4	Sherlock Holmes (2009)	1.077268e-07
5	Kung Fu Panda (2008)	1.066761e-07
6	Watchmen (2009)	1.061401e-07
7	Star Trek (2009)	1.047793e-07
8	Iron Man 2 (2010)	9.618056e-08
9	Avatar (2009)	9.313901e-08
10	Avengers, The (2012)	8.755834e-08

Figure 13 Output 1

```
get_movie_recommendation('jsnckjsdcsc')
```

```
'No movies found. Please check your input'
```

Figure 14 Output 2

6. RESULTS AND DISCUSSION

- **Measuring the Accuracy of the model:**

  To measure the accuracy of the model we will use Root Mean Squared Error.

  Root mean squared error (RMSE) is the square root of the mean of the square of all of the error. ... RMSE is **a good measure of accuracy**.

  Converting the CSR_DATA to array format to calculate the RMSE.

```
arr = csr_matrix(csr_data, dtype=np.int8).toarray()
print(arr)
```

```
[[4 0 0 ... 4 2 5]
 [0 0 4 ... 0 2 0]
 [4 0 5 ... 0 2 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

Figure 15 Converting the csr_data to numpy array

```
print('Root Mean Squared Error: ')
np.sqrt(np.mean(arr**2))
```

```
Root Mean Squared Error:

1.0774792624494953
```

Figure 16 RMSE Value

So here we get the RMSE value as. 1.077 which implies that it has a good fitting.

One way to gain a better understanding of whether a certain RMSE value is “good” is to normalize it using the following formula:

Normalized RMSE = RMSE / (max value – min value)

This produces a value between 0 and 1, where values closer to 0 represent better fitting models.

```
norm=np.amax(arr)-np.amin(arr)
Nomalized_RMSE = RMSE/norm
print('Normalised Root Mean Squared Error: ')
Nomalized_RMSE
```

```
Normalised Root Mean Squared Error:

0.21549585248989905
```

Figure 17 Normalized RMSE Value

So, here we can see that Normalized RMSE value that we get is 0.21, therefore the model gives us a better accuracy.

Therefore, we can conclude with the fact that the recommendation system built using **Sparse Matrix** gives a better accuracy on the whole with the given dataset.

7. FUTURE WORK

This movie recommender could be further improvised using “Matrix Factorization”.

In a real world setting, the vast majority of movies receive very few or even no ratings at all by users. We are looking at an extremely sparse matrix with more than 99% of entries are missing values. With such a sparse matrix, what ML algorithms can be trained and reliable to make inference?

So, data scarcity problem has to be addressed to build a better model and Matrix Factorization is the solution.