To build a Movie Recommendations with Movielens Dataset

Why Do We Need Recommender Systems?

We now live in what some call the “era of abundance”. For any given product, there are sometimes thousands of options to choose from and we don’t have sufficient time to look through all. Think of the examples : streaming videos, movies, social networking, online shopping; the list goes on.

Everyone loves movies irrespective of age, gender, race, colour, or geographical location. We all in a way are connected to each other via this amazing medium. Yet what most interesting is the fact that how unique our choices and combinations are in terms of movie preferences. Some people like genre-specific movies be it a thriller, romance, or sci-fi, while others focus on lead actors and directors. When we take all that into account, it’s astoundingly difficult to generalize a movie and say that everyone would like it. But with all that said, it is still seen that similar movies are liked by a specific part of the society. So we needed a system which would recommend things based on our likings.

Recommender systems help to personalize a platform and help the user find something they like without searching.

DIFFERENT TYPES OF RECOMMENDATION ENGINES:

There are two types of filtration strategy i.e. Content based filtering and Collaborative filtering.

Collaborative Filtering is the most common technique used when it comes to building intelligent recommender systems

that can learn to give better recommendations as more information about users is collected.

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:

User-based Collaborative filtering:

The basic idea here is to find users that have similar past preference patterns. It is not used much as people are fickle-minded and 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.

Item-based Collaborative Filtering:

The concept in this case is to find similar movies instead of similar users and then recommending similar movies. 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.

Item based collaborative filtering is used because unlike people's taste, movies don't change and there are usually a

lot fewer items than people, therefore easier to maintain and compute the matrices.

The algorithm used here is KNN as it can compete with the most accurate models because it makes highly accurate predictions. The quality of the predictions depends on the distance measure. When KNN makes inference about a movie, KNN will calculate the “distance” between the target movie and every other movie in its database, then it ranks its distances and returns the top K nearest neighbour movies as the most similar movie recommendations.

Using the Cosine function & K-Nearest Neighbor algorithm, we can determine how similar or different two sets of items are and use it to determine the classification. The Cosine function is used to calculate the Similarity or the Distance of the observations in high dimensional space. Cosine similarity is the dot product of the two vectors divided by the product of the two vectors' lengths (or magnitudes). A cosine value of 0 means that the two vectors are at 90 degrees to each other (orthogonal) and have no match. The closer the cosine value to 1, the smaller the angle and the greater the match between vectors.

DATASET: To experiment with recommendation algorithms, the data must contains a set of items and a set of users who have reacted to some of the items.The reaction can be explicit (rating on a scale of 1 to 5, likes or dislikes) or implicit (viewing an item, adding it to a wish list, the time spent on an article). The proposed approach is implemented using the MovieLens dataset, which is the most generalized data in movie recommendation systems.

MovieLens dataset:

MOVIES:

movie Id	title	genres

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

...

RATINGS:

userId	movieId	rating	timestamp
1	1	4	9.65E+08
1	3	4	9.65E+08
1	6	4	9.65E+08
1	47	5	9.65E+08

...

PYTHON CODE FOR MOVIE RECOMMENDATION SYSTEM:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

movies = pd.read_csv("movies.csv")
movies.head(10)

ratings = pd.read_csv("ratings.csv")
ratings.head(10)

final_dataset =
ratings.pivot(index='movieId',columns='userId',values='rating'
)
final_dataset.head(10)
```

```
final_dataset.fillna(0,inplace=True)
final_dataset.head(10)

no_user_voted =
ratings.groupby('movieId')['rating'].agg('count')
no_user_voted[:5]

no_movies_voted =
ratings.groupby('userId')['rating'].agg('count')
no_movies_voted[:5]

f,ax = plt.subplots(1,1,figsize=(16,4))
plt.scatter(no_user_voted.index,no_user_voted,color='mediu
mseagreen')
plt.axhline(y=10,color='r')
plt.xlabel('MovieId')
plt.ylabel('No. of users voted')
plt.show()

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

f,ax = plt.subplots(1,1,figsize=(16,4))
plt.scatter(no_movies_voted.index,no_movies_voted,color='m
ediumseagreen')
plt.axhline(y=50,color='r')
plt.xlabel('UserId')
plt.ylabel('No. of votes by user')
plt.show()

final_dataset=final_dataset.loc[:,no_movies_voted[no_movies
_voted > 50].index]
final_dataset.head()

from scipy.sparse import csr_matrix
csr_data = csr_matrix(final_dataset.values)
```

```
final_dataset.reset_index(inplace=True)
final_dataset.head()

from sklearn.neighbors import NearestNeighbors
knn = NearestNeighbors(metric='cosine', algorithm='brute',
n_neighbors=20, n_jobs=-1)
knn.fit(csr_data)

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().toli
st())),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'].valu
es[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"

movie=input('Enter a movie')
```

```
print("Recommendation for movie :  {}\n".format(movie))
get_movie_recommendation(movie)
```

Results and Discussion:

The amount of movie has increased to become more congested; therefore, to find a movie what users are looking for through the existing technologies are very hard. For this reason, the users want a system that can suggest the movie requirement to them and the best technology about these is the recommendation system.
Recommendation engines basically are data filtering tools that make use of algorithms and data to recommend the most relevant items to a particular user. Or in simple terms, they are nothing but an automated form of a “shop counter guy”. You ask him for a product. Not only he shows that product, but also the related ones which you could buy. They are well trained in cross-selling and upselling.

Recommender systems provide a better experience for the users by giving them a broader explain exposure to many different products that might be interested in and it also encouraging user towards continual usage.
In this movie recommendation model, movies are recommended according to their previous taste or liking.

References:

- https://realpython.com/build-recommendation-engine-collaborative-filtering/
- https://www.analyticsvidhya.com/blog/2020/11/create-your-own-movie-movie-recommendation-system/
- https://towardsdatascience.com/how-to-build-a-movie-recommendation-system-67e321339109
- https://www.kaggle.com/gauravduttakiit/movie-recommendation-system-collaborative-filter
- http://infolab.stanford.edu/~ullman/mmds/ch9.pdf

Getting the data up and running

First, we need to import libraries which we'll be using in our movie recommendation system. Also, we'll import the dataset by adding the path of the CSV files.

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:

```
movies = pd.read_csv("movies.csv")
movies.head(10)
```

Out[2]:

	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
5	6	Heat (1995)	Action|Crime|Thriller
6	7	Sabrina (1995)	Comedy|Romance
7	8	Tom and Huck (1995)	Adventure|Children
8	9	Sudden Death (1995)	Action
9	10	GoldenEye (1995)	Action|Adventure|Thriller

In [3]:

```
ratings = pd.read_csv("ratings.csv")
ratings.head(10)
```

Out[3]:

	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
5	1	70	3.0	964982400
6	1	101	5.0	964980868
7	1	110	4.0	964982176
8	1	151	5.0	964984041
9	1	157	5.0	964984100

Making a new dataframe where each column would represent each unique userId and each row represents each unique movieId.

In [4]:

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

Out[4]:

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609
movieId																				
1	4.0	NaN	NaN	NaN	4.0	NaN	4.5	NaN	NaN	NaN	...	4.0	NaN	4.0	3.0	4.0	2.5	4.0	2.5	3.0
2	NaN	NaN	NaN	NaN	NaN	4.0	NaN	4.0	NaN	NaN	...	NaN	4.0	NaN	5.0	3.5	NaN	NaN	2.0	NaN
3	4.0	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.0	NaN
4	NaN	NaN	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
193581	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193583	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193585	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193587	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193609	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

9724 rows × 610 columns

Imputing NaN with 0 to make things understandable for the algorithm

In [5]:

```
final_dataset.fillna(0,inplace=True)
final_dataset.head(10)
```

Out[5]:

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
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	3.0	4.0	2.5	4.0	2.5	3.0	5.0
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	5.0	3.5	0.0	0.0	2.0	0.0	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	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	3.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	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	3.0	0.0	0.0	0.0	0.0	0.0	0.0
6	4.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	...	0.0	3.0	4.0	3.0	0.0	0.0	0.0	0.0	0.0	5.0
7	0.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	2.5	0.0	0.0	0.0	0.0
8	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	3.0	0.0	2.0	0.0	0.0	...	0.0	3.0	0.0	0.0	0.0	0.0	0.0	4.0	4.0	0.0

10 rows × 610 columns

Removing Noise from the data by adding some filters for the final dataset.

In [6]:

```
no_user_voted = ratings.groupby('movieId')['rating'].agg('count')
no_user_voted[:5]
```

Out[6]:

```
movieId
1    215
2    110
3     52
4      7
5     49
Name: rating, dtype: int64
```

In [7]:

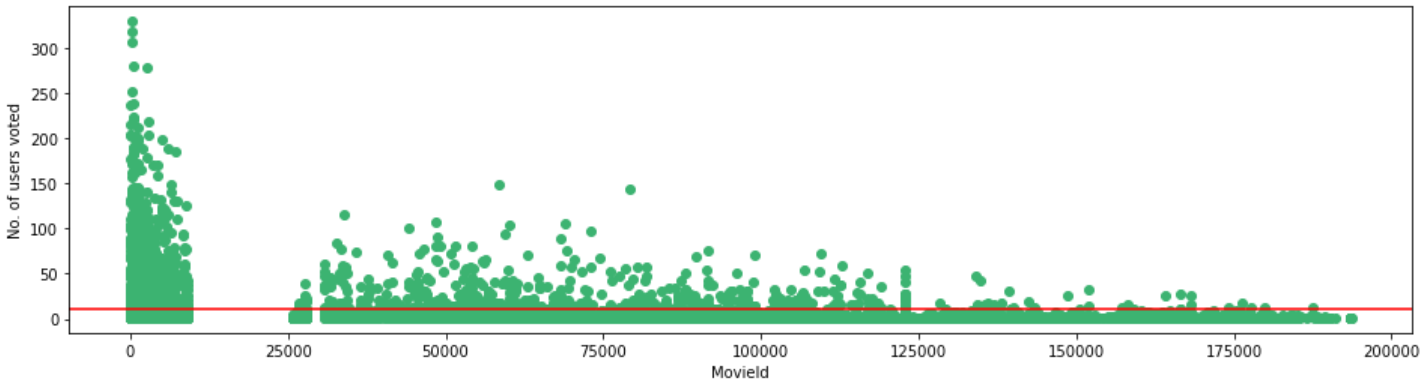
```
no_movies_voted = ratings.groupby('userId')['rating'].agg('count')
no_movies_voted[:5]
```

Out[7]:

```
userId
1    232
2     29
3     39
4    216
5     44
Name: rating, dtype: int64
```

In [8]:

```
f,ax = plt.subplots(1,1,figsize=(16,4))
# ratings['rating'].plot(kind='hist')
plt.scatter(no_user_voted.index,no_user_voted,color='mediumseagreen')
plt.axhline(y=10,color='r')
plt.xlabel('MovieId')
plt.ylabel('No. of users voted')
plt.show()
```



In [9]:

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

Out[9]:

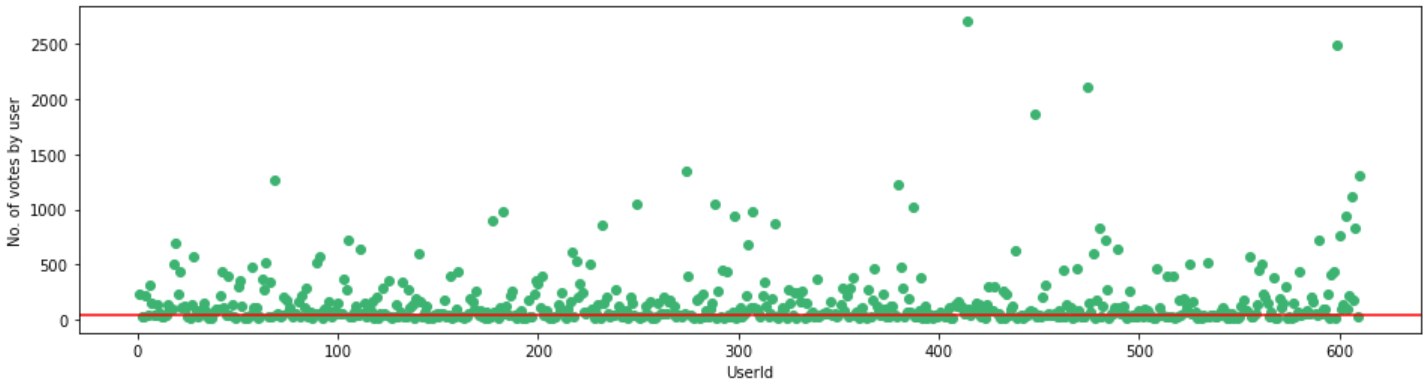
userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
movieId																					
1	4.0	0.0	0.0	0.0	4.0	0.0	4.5	0.0	0.0	0.0	...	4.0	0.0	4.0	3.0	4.0	2.5	4.0	2.5	3.0	5.0

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
movieId																					
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	0.0	0.0	0.0	0.0	2.0	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	3.0	0.0	0.0	0.0	0.0	0.0	0.0
6	4.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	...	0.0	3.0	4.0	3.0	0.0	0.0	0.0	0.0	0.0	5.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
174055	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
176371	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
177765	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	4.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
179819	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
187593	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

2121 rows × 610 columns

In [10]:

```
f,ax = plt.subplots(1,1,figsize=(16,4))
plt.scatter(no_movies_voted.index,no_movies_voted,color='mediumseagreen')
plt.axhline(y=50,color='r')
plt.xlabel('UserId')
plt.ylabel('No. of votes by user')
plt.show()
```



In [11]:

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

Out[11]:

userId	1	4	6	7	10	11	15	16	17	18	...	600	601	602	603	604	605	606	607	608	610
movieId																					
1	4.0	0.0	0.0	4.5	0.0	0.0	2.5	0.0	4.5	3.5	...	2.5	4.0	0.0	4.0	3.0	4.0	2.5	4.0	2.5	5.0
2	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	...	4.0	0.0	4.0	0.0	5.0	3.5	0.0	0.0	2.0	0.0
3	4.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0
5	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	2.5	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0
6	4.0	0.0	4.0	0.0	0.0	5.0	0.0	0.0	0.0	4.0	...	0.0	0.0	3.0	4.0	3.0	0.0	0.0	0.0	0.0	5.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
174055	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
176371	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
177765	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	4.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
179819	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
187593	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

2121 rows × 378 columns

Removing sparsity

In [12]:

```
from scipy.sparse import csr_matrix
csr_data = csr_matrix(final_dataset.values)
final_dataset.reset_index(inplace=True)
final_dataset.head()
```

Out[12]:

userId	movieId	1	4	6	7	10	11	15	16	17	...	600	601	602	603	604	605	606	607	608	610
0	1	4.0	0.0	0.0	4.5	0.0	0.0	2.5	0.0	4.5	...	2.5	4.0	0.0	4.0	3.0	4.0	2.5	4.0	2.5	5.0
1	2	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	...	4.0	0.0	4.0	0.0	5.0	3.5	0.0	0.0	2.0	0.0
2	3	4.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0
3	5	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	...	2.5	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0
4	6	4.0	0.0	4.0	0.0	0.0	5.0	0.0	0.0	0.0	...	0.0	0.0	3.0	4.0	3.0	0.0	0.0	0.0	0.0	5.0

5 rows × 379 columns

Making the movie recommendation system model

In [13]:

```
from sklearn.neighbors import NearestNeighbors
knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20, n_jobs=-1)
knn.fit(csr_data)
```

Out[13]:

```
NearestNeighbors(algorithm='brute', metric='cosine', n_jobs=-1, n_neighbors=20)
```

Making the recommendation function

In [14]:

```
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],'Distanc
e':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"
```

Finally, Let's Recommend some movies!

In [15]:

```
movie=input('Enter a movie')
print("Recommendation for movie :  {}\n".format(movie))
get_movie_recommendation(movie)
```

```
Enter a movieJumanji
Recommendation for movie :  Jumanji
```

Out[15]:

	Title	Distance
1	Casper (1995)	0.474253
2	Stargate (1994)	0.469654
3	Nightmare Before Christmas, The (1993)	0.462612
4	Home Alone (1990)	0.443432
5	Beauty and the Beast (1991)	0.435007
6	Aladdin (1992)	0.425428
7	Jurassic Park (1993)	0.420563
8	Mrs. Doubtfire (1993)	0.416164
9	Mask, The (1994)	0.413743
10	Lion King, The (1994)	0.377013

In [ ]:

```

```