Comparing Collaborative Filtering and Hybrid based Approaches for Movie Recommendation

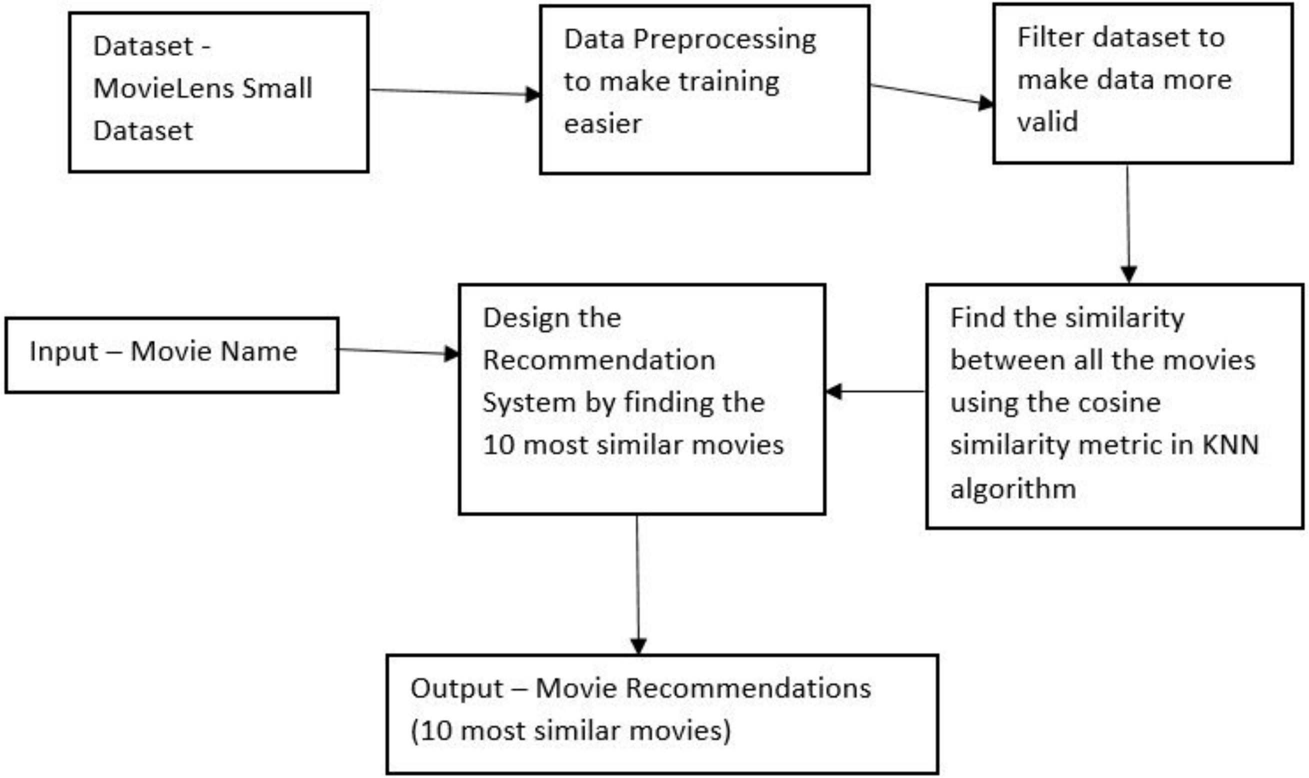
TEAM MEMBERS

1) ASHWIN MENON - 2019103510
2) SESHATHILAK G - 2019103579
3) VIGNESH KUMAR K - 2019103594

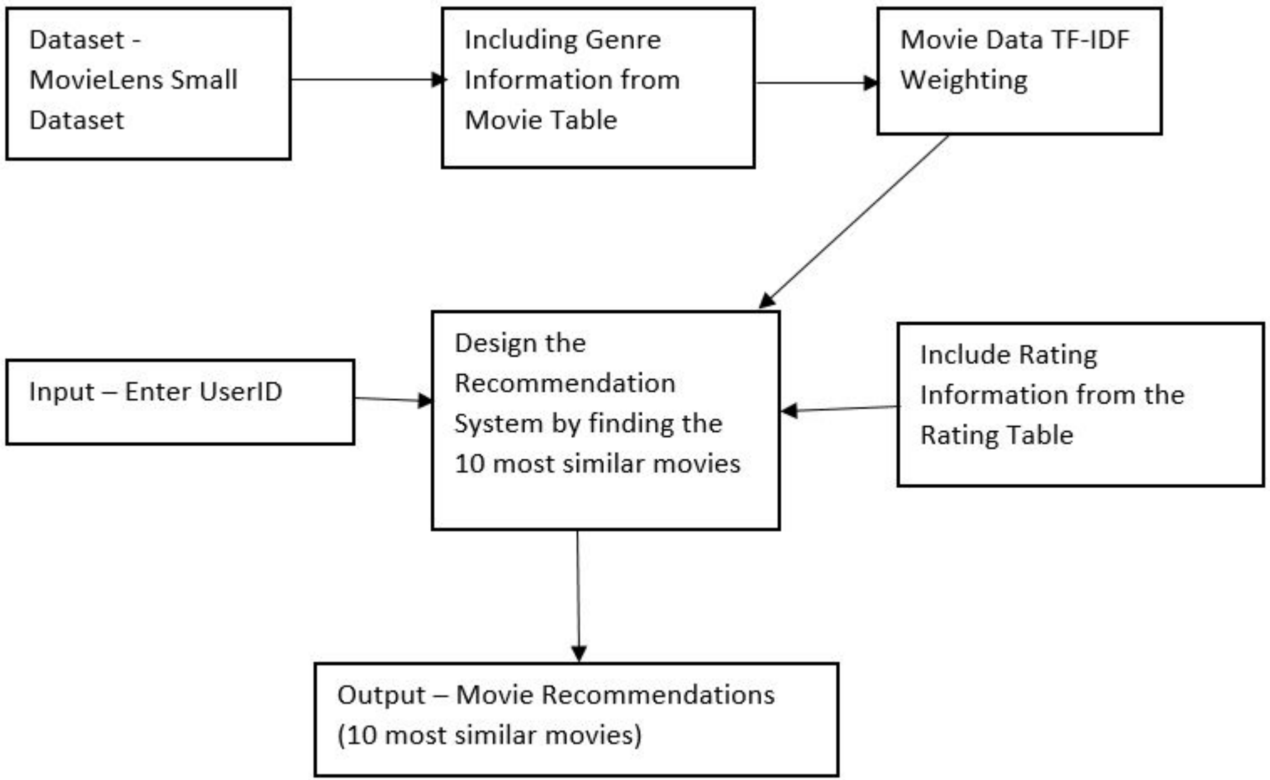
INTRODUCTION

- The main aim of the mini project is to create a movie recommender system using collaborative filtering and hybrid based approaches.
- Movies are something which we all see to relax and entertain ourselves and finding a similar movie to the one you like will be very much useful and helpful
- The objective of automating the movie recommender is to make the selection of which movie to see next quick and less cumbersome for the user.

BLOCK DIAGRAM FOR COLLABORATIVE FILTERING



BLOCK DIAGRAM FOR HYBRID BASED APPROACH



LIST OF MODULES FOR COLLABORATIVE FILTERING

1. Importing Data
2. Data Pre Processing
3. Removing Noise from the data
4. Removing sparsity
5. Designing movie recommendation system model
6. Recommend Similar Movies

1) IMPORTING DATA

- First, we import libraries which we'll be using in our movie recommendation system.
- Next we import the 2 datasets we require in our system
- The 2 datasets are rating dataset and movies dataset from the Movielens small dataset (https://www.kaggle.com/shubhammehta21/movie-lens-small-latest-dataset)

```python
import pandas as pd
import numpy as np
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt
import seaborn as sns

# Dataset Source : https://www.kaggle.com/shubhammehta21/movie-lens-small-latest-dataset
movies = pd.read_csv("movies.csv")
ratings = pd.read_csv("ratings.csv")
```

1)IMPORTING DATA (contd)

Movie dataset has

- movieId – once the recommendation is done, we get a list of all similar movieId and get the title for each movie from this dataset.
- genres – which is not required for this filtering approach.

```
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

1)IMPORTING DATA (contd)

Ratings Dataset has

- userId – unique for each user.
- movieId – using this feature, we take the title of the movie from the movies dataset.
- rating – Ratings given by each user to all the movies using this we are going to predict the top 10 similar movies..

```
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

2)DATA PRE PROCESSING

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	604	605	606	607	608	609	610
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	5.0
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	NaN
3	4.0	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.0	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	3.0	NaN	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	NaN

5 rows × 610 columns

2)DATA PRE PROCESSING(contd)

We can see that some users have not rated movies hence the Nan value present in the dataset. We can replace this Nan value with 0 to make the computing more easier

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

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

5 rows × 610 columns

3) NOISE REMOVAL FROM DATA

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.

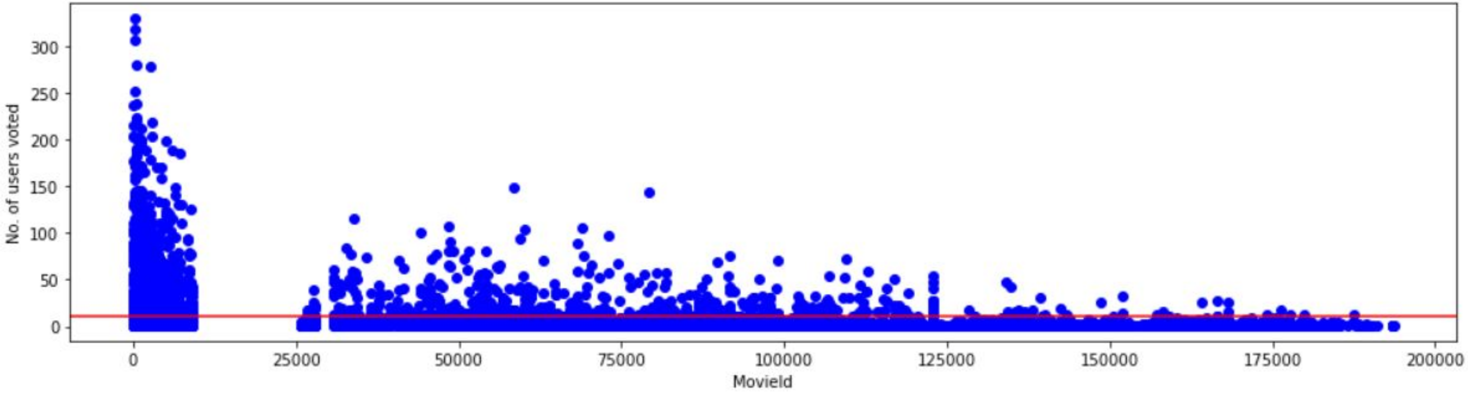
Aggregating the number of users who voted and the number of movies that were voted.

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

3) NOISE REMOVAL FROM DATA(contd)

- Visualizing the number of users who voted and then making the necessary changes to remove the non credible data from the dataset (Minimum threshold : 10)

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

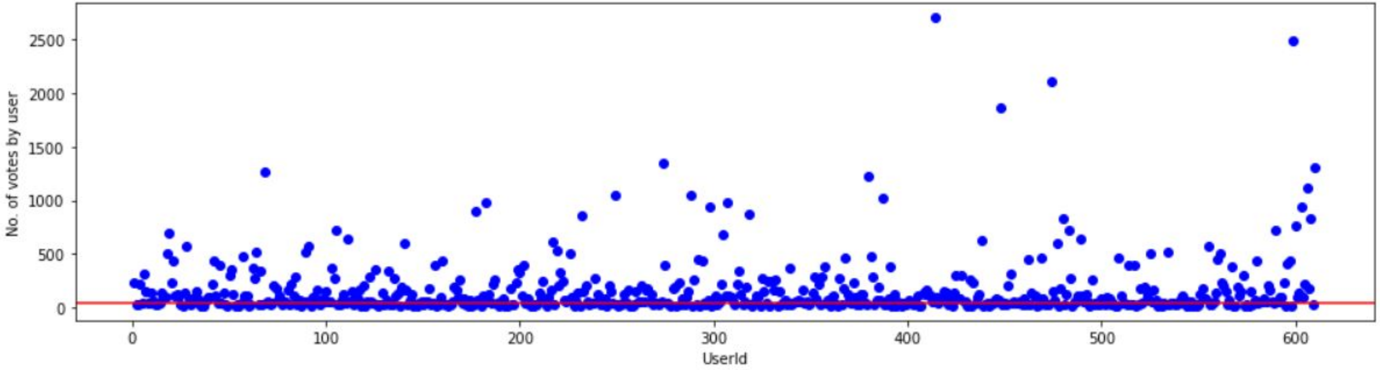



```
final_dataset = final_dataset.loc[no_user_voted[no_user_voted > 10].index,:]
```

3) NOISE REMOVAL FROM DATA(contd)

- Visualizing the number of movie which were voted by users and then making the necessary changes to remove the non credible data from the dataset (Minimum threshold : 50)

```
f,ax = plt.subplots(1,1,figsize=(16,4))
plt.scatter(no_movies_voted.index,no_movies_voted,color='blue')
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]
```

4)SPARSITY REMOVAL

Our system may run out of computational resources when the large dataset is feed to the model. To reduce the sparsity we use the csr_matrix function from the scipy library

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

5) DESIGNING THE 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

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

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

5) DESIGNING THE MODEL (contd)

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"
```

6) RECOMMEND SIMILAR MOVIES

Once the user enters the movie name the 10 most similar movies present in the dataset are displayed

```
get_movie_recommendation('Memento')
```

	Title	Distance
1	American Beauty (1999)	0.389346
2	American History X (1998)	0.388615
3	Pulp Fiction (1994)	0.386235
4	Lord of the Rings: The Return of the King, The...	0.371622
5	Kill Bill: Vol. 1 (2003)	0.350167
6	Lord of the Rings: The Two Towers, The (2002)	0.348358
7	Eternal Sunshine of the Spotless Mind (2004)	0.346196
8	Matrix, The (1999)	0.326215
9	Lord of the Rings: The Fellowship of the Ring,...	0.316777
10	Fight Club (1999)	0.272380

6) RECOMMEND SIMILAR MOVIES(contd)

Here are few more examples

```
get_movie_recommendation('Titanic')
```

	Title	Distance
1	Good Will Hunting (1997)	0.460759
2	Truman Show, The (1998)	0.460628
3	Catch Me If You Can (2002)	0.460281
4	Sixth Sense, The (1999)	0.452878
5	Saving Private Ryan (1998)	0.437196
6	Shrek (2001)	0.433120
7	Finding Nemo (2003)	0.432400
8	Star Wars: Episode I - The Phantom Menace (1999)	0.427623
9	Forrest Gump (1994)	0.427187
10	Men in Black (a.k.a. MIB) (1997)	0.420254

```
get_movie_recommendation('Avatar')
```

	Title	Distance
1	Zombieland (2009)	0.398180
2	Inception (2010)	0.393521
3	I Am Legend (2007)	0.389856
4	Hangover, The (2009)	0.364190
5	Dark Knight, The (2008)	0.358937
6	Kung Fu Panda (2008)	0.358604
7	Iron Man (2008)	0.310893
8	District 9 (2009)	0.309947
9	WALL·E (2008)	0.306969
10	Up (2009)	0.289607

LIST OF MODULES FOR HYBRID BASED APPROACH

1. Importing Data
2. Include Genre Information
3. Movie Data TF-IDF Weighting
4. Include Rating Information
5. Designing Movie Recommendation System Model
6. Recommend Similar Movies

1) IMPORTING DATA

- First, we import libraries which we'll be using in our movie recommendation system.
- Next we import the 2 datasets we require in our system
- The 2 datasets are rating dataset and movies dataset from the Movielens small dataset (https://www.kaggle.com/shubhammehta21/movie-lens-small-latest-dataset)

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

# Dataset Source : https://www.kaggle.com/shubhammehta21/movie-lens-small-latest-dataset
movies = pd.read_csv("movies.csv")
ratings = pd.read_csv("ratings.csv")
movies = movies.replace({np.nan: None})

movie_initial = movies
```

1)IMPORTING DATA (contd)

Movie dataset has

- movieId – once the recommendation is done, we get a list of all similar movieId and get the title for each movie from this dataset.
- genres – which is not required for this filtering approach.

```
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

1)IMPORTING DATA (contd)

Ratings Dataset has

- userId – unique for each user.
- movieId – using this feature, we take the title of the movie from the movies dataset.
- rating – Ratings given by each user to all the movies using this we are going to predict the top 10 similar movies..

```
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

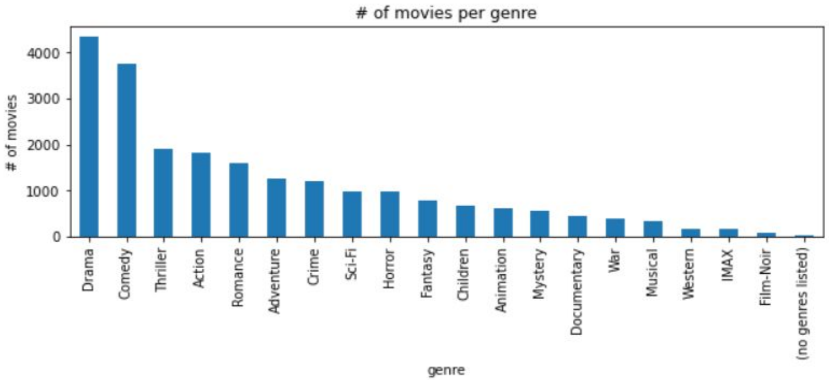
2)INCLUDE GENRE INFORMATION

All the genres are present in the genres field inside movies dataset so we first find all the genres present in the dataset and visualize the count for each genre

```
all_genres = [s.split("|") for s in movies[movies.genres.notnull()].genres]
genres = [item for l in all_genres for item in l ]
unique_genres = set(genres)
print (f"total of {len(unique_genres)} unique genres from {len(genres)} occurances.")

pd.Series(genres).value_counts().plot(kind='bar', figsize=(10, 3))
plt.title("# of movies per genre")
plt.ylabel("# of movies")
plt.xlabel("genre")
plt.show()
```

```
total of 20 unique genres from 22084 occurances.
```



2)INCLUDE GENRE INFORMATION (contd.)

Now we update the movies dataframe by adding all the genres as column headings and then iterate over all the movies and find the genres for each movie and mark only that field in the dataframe as 1

```
genres = [item.strip() for l in all_genres for item in l ]
unique_genres = set(genres)
for genre in unique_genres:
    movies[genre] = 0
```

```
for i in range(len(movies)):
    if type(movies['genres'].iloc[i]) != None.__class__:
        Genres = movies.iloc[i].genres.split('|')
        for g in Genres:
            movies[g].iloc[i] = 1

movies.head()
```

2)INCLUDE GENRE INFORMATION (contd.)

We now remove the title and genres column as they are no longer needed and then set the movieId as index and sort the data frame based on movieId

```
movies = movies.drop(columns=['title', 'genres']).set_index('movieId')

movies.sort_index(axis=0, inplace=True)
movies.head()
```

	Western	Children	Action	Animation	(no genres listed)	Thriller	Horror	Documentary	IMAX	Crime	Musical	Mystery	Romance	War	Sci-Fi	Fantasy	Film-Noir	C
movieId																		
1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	
2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	
3	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

3) MOVIE DATA TF-IDF WEIGHTING

We first calculate the document frequency (df) the number of terms(in this case genres) within all movies and then calculate Inverse document frequency (IDF) which measures the importance of each term (in this case genre) within all movies

```
df = movies.sum()
idf = (len(movies)/df).apply(np.log)

print("\nDocument Frequency : \n",df,"\n\nInverse Document Frequency : \n",idf,"\n")
```

3) MOVIE DATA TF-IDF WEIGHTING(contd)

We now print the document and inverse document frequency calculated

```
Document Frequency :
 Western               167
Children               664
Action                1828
Animation              611
(no genres listed)      34
Thriller              1894
Horror                 978
Documentary            440
IMAX                   158
Crime                 1199
Musical                334
Mystery                573
Romance               1596
War                    382
Sci-Fi                 980
Fantasy                779
Film-Noir               87
Comedy                3756
Adventure             1263
Drama                 4361
dtype: int64
```

```
Inverse Document Frequency :
 Western               4.066208
Children               2.685920
Action                 1.673224
Animation              2.769105
(no genres listed)     5.657841
Thriller               1.637755
Horror                 2.298692
Documentary            3.097427
IMAX                   4.121607
Crime                  2.094959
Musical                3.373061
Mystery                2.833316
Romance                1.808946
War                    3.238781
Sci-Fi                 2.296649
Fantasy                2.526191
Film-Noir              4.718294
Comedy                 0.953092
Adventure              2.042957
Drama                  0.803745
dtype: float64
```

3) MOVIE DATA TF-IDF WEIGHTING(contd)

We now calculate the tf-idf value and print it

```
TFIDF = movies.mul(idf.values)
TFIDF
```

movieId	Western	Children	Action	Animation	(no genres listed)	Thriller	Horror	Documentary	IMAX	Crime	Musical	Mystery	Romance	War	Sci-Fi	Fantasy	Film-Noir
1	0.0	2.68592	0.000000	2.769105	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	2.526191	0.0
2	0.0	2.68592	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	2.526191	0.0
3	0.0	0.00000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.808946	0.0	0.0	0.000000	0.0
4	0.0	0.00000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.808946	0.0	0.0	0.000000	0.0
5	0.0	0.00000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
193581	0.0	0.00000	1.673224	2.769105	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	2.526191	0.0
193583	0.0	0.00000	0.000000	2.769105	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	2.526191	0.0
193585	0.0	0.00000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0
193587	0.0	0.00000	1.673224	2.769105	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0
193609	0.0	0.00000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0

9742 rows × 20 columns

4)INCLUDE RATING INFORMATION

We are now going to make a new dataframe where each column would represent each unique userId and each row represents each unique movieId so that it is easier to visualize. We make all the missing values as 0

```
user_x_movie = pd.pivot_table(ratings, values='rating', index=['movieId'], columns = ['userId'])
user_x_movie.sort_index(axis=0, inplace=True)

user_x_movie.fillna(0,inplace=True)
user_x_movie
```

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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
193581	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
193583	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
193585	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
193587	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
193609	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

9724 rows × 610 columns

4)INCLUDE RATING INFORMATION(contd)

We are now going to calculate the user's preference of each genre in any movie using the user ratings for each movie from the ratings dataset

```python
from tqdm import tqdm

userIDs = user_x_movie.columns
user_profile = pd.DataFrame(columns = movies.columns)

for i in tqdm(range(len(user_x_movie.columns))):
    working_df = movies.mul(user_x_movie.iloc[:,i], axis=0)
    user_profile.loc[userIDs[i]] = working_df.mean(axis=0)
```

```
100%|██████████████████████████████████████████████████| 610/610 [00:02<00:00, 204.72it/s]
```

```python
user_profile
```

	Western	Children	Action	Animation	(no genres listed)	Thriller	Horror	Documentary	IMAX	Crime	Musical	Mystery	Romance	War	Sci-Fi
1	0.003085	0.019642	0.040004	0.013986	0.0	0.023447	0.006067	0.000000	0.000000	0.020156	0.010592	0.007713	0.011518	0.010181	0.017380
2	0.000360	0.000000	0.004473	0.000000	0.0	0.003805	0.000309	0.001337	0.001543	0.003908	0.000000	0.000823	0.000463	0.000463	0.001594
3	0.000000	0.000257	0.005142	0.000206	0.0	0.002982	0.003856	0.000000	0.000000	0.000103	0.000051	0.000514	0.000257	0.000257	0.006479
4	0.003908	0.003908	0.008536	0.002468	0.0	0.013883	0.001748	0.000823	0.000309	0.010592	0.006582	0.008227	0.020156	0.002571	0.003497
5	0.000617	0.003805	0.002879	0.002674	0.0	0.003291	0.000309	0.000000	0.001131	0.004731	0.002262	0.000411	0.003497	0.001028	0.000514
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
606	0.005965	0.017380	0.049362	0.016043	0.0	0.072141	0.017894	0.001954	0.005039	0.049979	0.016865	0.035479	0.136569	0.025350	0.028898
607	0.000823	0.006684	0.027561	0.002057	0.0	0.025812	0.014809	0.000000	0.000514	0.010592	0.001851	0.008124	0.010490	0.002571	0.012032
608	0.002982	0.022265	0.094868	0.017637	0.0	0.094200	0.033114	0.001851	0.004936	0.054247	0.009358	0.025195	0.031469	0.006993	0.056613
609	0.000411	0.000617	0.003497	0.000309	0.0	0.004731	0.000720	0.000617	0.000309	0.002160	0.000000	0.000000	0.001645	0.001440	0.001543
610	0.012701	0.021030	0.191434	0.026481	0.0	0.187423	0.109266	0.002160	0.030594	0.106695	0.005656	0.046483	0.045660	0.018254	0.094457

5)DESIGNING MOVIE RECOMMENDATION SYSTEM MODEL

We now calculate the sum product of the importance weights and users' preferences towards different genres

```python
df_predict = pd.DataFrame()

for i in tqdm(range(len(user_x_movie.columns))):
    working_df = TFIDF.mul(user_profile.iloc[i], axis=1)
    df_predict[user_x_movie.columns[i]] = working_df.sum(axis=1)
```

```
100%|████████████████████████████████████████| 610/610 [00:02<00:00, 263.53it/s]
```

```python
df_predict
```

	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	6
movieId																
1	0.257124	0.005371	0.011952	0.093561	0.035717	0.167487	0.097899	0.025238	0.018216	0.094271	...	0.118148	0.048139	0.230575	0.054178	0.2828
2	0.183600	0.002626	0.010501	0.050951	0.023216	0.109304	0.069180	0.016267	0.011686	0.052467	...	0.082283	0.032922	0.122745	0.038130	0.2326
3	0.055630	0.003582	0.001347	0.072237	0.011422	0.089015	0.029982	0.016663	0.008925	0.073655	...	0.021549	0.028006	0.243859	0.017213	0.0649
4	0.081088	0.009037	0.002339	0.106787	0.019274	0.130839	0.044736	0.022614	0.014877	0.092418	...	0.039486	0.046521	0.405286	0.028454	0.0801
5	0.034795	0.002744	0.000882	0.035775	0.005097	0.041950	0.015192	0.007547	0.005391	0.025288	...	0.008527	0.012938	0.105267	0.009213	0.0284
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
193581	0.192937	0.010230	0.013562	0.075077	0.024853	0.121831	0.082911	0.019231	0.013430	0.072661	...	0.085997	0.047781	0.234842	0.046659	0.2015
193583	0.126001	0.002744	0.004959	0.060795	0.020035	0.082083	0.047034	0.012348	0.009128	0.057002	...	0.058207	0.021971	0.151215	0.026699	0.1632
193585	0.025458	0.005455	0.000992	0.034550	0.007852	0.041824	0.014754	0.005951	0.005951	0.018763	...	0.017936	0.018515	0.161427	0.011241	0.0152
193587	0.105665	0.007485	0.009173	0.021116	0.012222	0.055980	0.049404	0.008307	0.005441	0.032175	...	0.055127	0.028089	0.086190	0.026795	0.0600
193609	0.034795	0.002744	0.000882	0.035775	0.005097	0.041950	0.015192	0.007547	0.005391	0.025288	...	0.008527	0.012938	0.105267	0.009213	0.0284

5)DESIGNING MOVIE RECOMMENDATION SYSTEM MODEL(contd)

Based on the previous sum of products we suggest the users the top 10 candidates as recommendations. We first get the predicted rating of all films for the user, then we combine film rating and film detail. We then recommend the films only which has not been seen by the user

```
def recommender(user_no):
    user_predicted_rating = df_predict[df_predict.columns[user_no - 1]]
    user_rating_film = pd.merge(user_predicted_rating, movie_initial, left_on='movieId', right_on='movieId')
    already_watched = ratings[ratings['userId'].isin([user_no])]['movieId']
    all_rec = user_rating_film[~user_rating_film.index.isin(already_watched)]
    return all_rec.sort_values(by=[user_no], ascending=False).iloc[0:10][['movieId', 'title']]
```

6) RECOMMEND SIMILAR MOVIES

Once the user enters the userID the 10 most similar movies present in the dataset are displayed

```
recommender(600)
```

	movieId	title
6626	56152	Enchanted (2007)
1390	1907	Mulan (1998)
7530	84637	Gnomeo & Juliet (2011)
3194	4306	Shrek (2001)
7805	92348	Puss in Boots (Nagagutsu o haita neko) (1969)
7170	71999	Aelita: The Queen of Mars (Aelita) (1924)
4631	6902	Interstate 60 (2002)
9169	148775	Wizards of Waverly Place: The Movie (2009)
2250	2987	Who Framed Roger Rabbit? (1988)
5819	32031	Robots (2005)

6) RECOMMEND SIMILAR MOVIES(contd)

Here are few more examples

```
recommender(400)
```

	movieId	title
7372	79132	Inception (2010)
7441	81132	Rubber (2010)
5556	26701	Patlabor: The Movie (Kidô keisatsu patorebâ: T...
167	198	Strange Days (1995)
1978	2625	Black Mask (Hak hap) (1996)
7170	71999	Aelita: The Queen of Mars (Aelita) (1924)
2248	2985	RoboCop (1987)
454	519	RoboCop 3 (1993)
400	459	Getaway, The (1994)
6358	49530	Blood Diamond (2006)

```
recommender(267)
```

	movieId	title
7170	71999	Aelita: The Queen of Mars (Aelita) (1924)
5980	36509	Cave, The (2005)
9394	164226	Maximum Ride (2016)
6145	43932	Pulse (2006)
5161	8361	Day After Tomorrow, The (2004)
9707	187031	Jurassic World: Fallen Kingdom (2018)
7767	91500	The Hunger Games (2012)
6330	48774	Children of Men (2006)
8590	117529	Jurassic World (2015)
5665	27618	Sound of Thunder, A (2005)

PERFORMANCE METRICS

1. Accuracy
2. F1 Score
3. Recall
4. Precision

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

$$F1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

F1 Score

$$Precision = \frac{TruePositives}{TruePositives + FalseNegatives}$$

Recall

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

Precision

COMPARISON

- When you are using collaborative filtering the system may recommend movies which other users have rated or watched similarly while you may not have any interest.
- For Example, a user would be suggested some genre which they may not like, because other audience who watched the same movies which the user had, watched that particular genre.
- So that is why we combine content based filtering and collaborative filtering so that even the user’s interests are acknowledged leading to a much better recommendation for a user

REFERENCE

N. Ifada, T. F. Rahman and M. K. Sophan, "Comparing Collaborative Filtering and Hybrid based Approaches for Movie Recommendation," 2020 6th Information Technology International Seminar (ITIS), 2020, pp. 219-223, doi: 10.1109/ITIS50118.2020.9321014. (https://ieeexplore.ieee.org/document/9321014)

THANK YOU