Movie Recommendation System

- Manasi Shah

1. Introduction

- A recommendation system provides suggestions to the users through a filtering process that is based on user preferences and content based.
- The information about the user is taken as an input. This information reflects the prior usage of the product as well as the assigned ratings.
- A recommendation system is a platform that provides its users with various contents based on their preferences and likings.
- I aim to build 3 types of recommendation system – Simple recommendation system, Content Based filtering and Collaborative filtering.

2. Data

- Source of the dataset: https://www.kaggle.com/rounakbanik/the-movies-dataset
- Size of dataset: 900 MB
- The dataset has 7 files and 45K+ instances.
- I have used 4 csv files (movies_metadata, credits and small_ratings, links)

File Name	Instances	Attributes
movies_metadata.csv	45467	24
credits.csv	45505	3
keywords.csv	46428	2
links.csv	45844	3
small_links.csv	9126	3
ratings.csv	26000000	4
ratings_small.csv	100005	4

The Attributes of movies_metadata.cvs:

1. adult: Indicates if the movie is X-Rated or Adult.
2. belongs_to_collection: A stringified dictionary that gives information on the movie series the particular film belongs to.
3. budget: The budget of the movie in dollars.
4. genres: A stringified list of dictionaries that list out all the genres associated with the movie.
5. homepage: The Official Homepage of the move.
6. id: The ID of the move.
7. imdb_id: The IMDB ID of the movie.
8. original_language: The language in which the movie was originally shot in.
9. original_title: The original title of the movie.
10. overview: A brief blurb of the movie.
11. popularity: The Popularity Score assigned by TMDB.

12. poster_path: The URL of the poster image.
13. production_companies: A stringified list of production companies involved with the making of the movie.
14. production_countries: A stringified list of countries where the movie was shot/produced in.
15. release_date: Theatrical Release Date of the movie.
16. revenue: The total revenue of the movie in dollars.
17. runtime: The runtime of the movie in minutes.
18. spoken_languages: A stringified list of spoken languages in the film.
19. status: The status of the movie (Released, To Be Released, Announced, etc.)
20. tagline: The tagline of the movie.
21. title: The Official Title of the movie.
22. video: Indicates if there is a video present of the movie with TMDB.
23. vote_average: The average rating of the movie.
24. vote_count: The number of votes by users, as counted by TMDB.

3. Problems to be Solved

- In today's world, every customer is faced with multiple choices. They might waste a lot of time browsing around on the internet and trawling through various sites hoping to strike gold. Without recommendation system they might waste time on various sites.
- Therefore, the recommendation systems are important as they help them make the right choices, without having to expend their cognitive resources.
- The proposed methodology of Movies Recommendation System deals with different stages of the project which consists of data collection, data preprocessing, model generation, prediction and outcomes.

4. KDD

4.1. Data Processing

1. Merging two datasets movies_metadata.csv and credits

```
# merging movie_metadata and credits dataset
df2["id"] = df1["id"].astype(str).astype(int)
df2= df2.merge(df1,on='id')
```

```
df2.columns
```

```
Index(['adult', 'belongs_to_collection', 'budget', 'genres', 'homepage', 'id',
       'imdb_id', 'original_language', 'original_title', 'overview',
       'popularity', 'poster_path', 'production_companies',
       'production_countries', 'release_date', 'revenue', 'runtime',
       'spoken_languages', 'status', 'tagline', 'title', 'video',
       'vote_average', 'vote_count', 'cast', 'crew'],
      dtype='object')
```

2. Dropping junk row

```
#dropping junk row
df2 = df2[df2.belongs_to_collection != '2.185485']
```

3. Extracting years and genres

```
df2['year'] = pd.to_datetime(df2['release_date'], errors='coerce').apply(lambda x: str(x).split('-')[0] if x != np.nan else np.nan)
df2['new_genres'] = df2['genres'].fillna('[]').apply(literal_eval).apply(lambda x: [i['name'] for i in x] if isinstance(x, list) else [])
```

```
0      1995   0           [Animation, Comedy, Family]
1      1995   1          [Adventure, Fantasy, Family]
2      1995   2                     [Romance, Comedy]
3      1995   3              [Comedy, Drama, Romance]
4      1995   4                              [Comedy]
5      1995   5      [Action, Crime, Drama, Thriller]
6      1995   6                     [Comedy, Romance]
7      1995   7    [Action, Adventure, Drama, Family]
8      1995   8         [Action, Adventure, Thriller]
9      1995   9         [Adventure, Action, Thriller]
```

4. Original Title

```
df2[df2['original_title'] != df2['title']][['title', 'original_title']].head()
```

	title	original_title
28	The City of Lost Children	La Cité des Enfants Perdus
29	Shanghai Triad	摇啊摇，摇到外婆桥
32	Wings of Courage	Guillaumet, les ailes du courage
57	The Postman	Il postino
58	The Confessional	Le confessionnal

5. Dropping irrelevant column

```
df2.drop(['adult', 'budget', 'homepage', 'poster_path', 'production_countries' ], axis=1)

df2.drop(['release_date', 'runtime', 'spoken_languages', 'status', 'video' ], axis=1)
```

4.2. Data Mining Methods and Processes

1. Simple Recommendation: This system used overall Vote Count and Vote Averages to build Top Movies Charts, in general and for a specific genre. The IMDB Weighted Rating System was used to calculate ratings on which the sorting was finally performed.

a) Popularity – It is most simple to implement system as it's impersonal.

```
# imdb rating
C= df2['vote_average'].mean()
m= df2['vote_count'].quantile(0.9)

def weighted_rating(x, m=m, C=C):
    v = x['vote_count']
    R = x['vote_average']
    # Calculation based on the IMDB formula
    return (v/(v+m) * R) + (m/(m+v) * C)
```

```
q_movies = df2.copy().loc[df2['vote_count'] >= m]

# Defining 'score' and calculating its value with weighted_rating()
q_movies['score'] = q_movies.apply(weighted_rating, axis=1)

#Sorting the movies based on score calculated above
q_movies = q_movies.sort_values('score', ascending=False)
```

```python
#Print the top 10 movies
q_movies[['title','year','vote_count','vote_average','popularity','score']].head(10)
```

	title	year	vote_count	vote_average	popularity	score
314	The Shawshank Redemption	1994	8358.0	8.5	51.6454	8.445873
837	The Godfather	1972	6024.0	8.5	41.1093	8.425444
10345	Dilwale Dulhania Le Jayenge	1995	661.0	9.1	34.457	8.421495
12525	The Dark Knight	2008	12269.0	8.3	123.167	8.265480
2854	Fight Club	1999	9678.0	8.3	63.8696	8.256388
292	Pulp Fiction	1994	8670.0	8.3	140.95	8.251410
522	Schindler's List	1993	4436.0	8.3	41.7251	8.206647
23743	Whiplash	2014	4376.0	8.3	64.3	8.205412
5501	Spirited Away	2001	3968.0	8.3	41.0489	8.196063
2219	Life Is Beautiful	1997	3643.0	8.3	39.395	8.187181

b) Popularity and Genre

```python
# function for genre based
def build_chart(genre):
    df = gen_md[gen_md['genre'] == genre]
    vote_counts = df[df['vote_count'].notnull()]['vote_count'].astype('int')
    vote_averages = df[df['vote_average'].notnull()]['vote_average'].astype('int')
    C = vote_averages.mean()
    m = vote_counts.quantile(0.9)

    q_movies = df[(df['vote_count'] >= m)][['title', 'year', 'vote_count', 'vote_average', 'popularity']]

    def weighted_rating(x, m=m, C=C):
      v = x['vote_count']
      R = x['vote_average']
      # Calculation based on the IMDB formula
      return (v/(v+m) * R) + (m/(m+v) * C)

    # Define a new feature 'score' and calculate its value with `weighted_rating()`
    q_movies['score'] = q_movies.apply(weighted_rating, axis=1)

    #Sort movies based on score calculated above
    q_movies = q_movies.sort_values('score', ascending=False)

    return q_movies
```

```python
build_chart('Romance').head(10)
```

	title	year	vote_count	vote_average	popularity	score
10345	Dilwale Dulhania Le Jayenge	1995	661.0	9.1	34.457	8.350944
351	Forrest Gump	1994	8147.0	8.2	48.3072	8.143504
40345	Your Name.	2016	1030.0	8.5	34.461252	8.065803
40975	La La Land	2016	4745.0	7.9	19.681686	7.814641
22240	Her	2013	4215.0	7.9	13.8295	7.804319
7237	Eternal Sunshine of the Spotless Mind	2004	3758.0	7.9	12.9063	7.793182
1141	Cinema Paradiso	1988	834.0	8.2	14.177	7.731170
4860	Amélie	2001	3403.0	7.8	12.8794	7.687268
25054	The Theory of Everything	2014	3403.0	7.8	11.853	7.687268
882	Vertigo	2058	1162.0	8.0	18.2082	7.672054

2. Collaborative Rating: I have used the powerful Surprise Library to build a collaborative filter based on single value decomposition. The RMSE obtained was less than 1 and the engine gave estimated ratings for a given user and movie.

```
from surprise import Reader, Dataset, SVD
from surprise.model_selection import cross_validate
reader = Reader()
ratings = pd.read_csv('ratings_small.csv')
ratings.head()
```

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205

```
data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)
```

```
svd = SVD()
cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8939  0.8989  0.8977  0.8952  0.8988  0.8969  0.0020
MAE (testset)     0.6866  0.6936  0.6931  0.6915  0.6898  0.6909  0.0025
Fit time          6.39    4.89    5.35    5.27    5.05    5.39    0.53
Test time         0.44    0.16    0.20    0.16    0.15    0.22    0.11
{'test_rmse': array([0.89391113, 0.89891495, 0.89765865, 0.89524431, 0.89881055]),
 'test_mae': array([0.68661831, 0.69356335, 0.69310485, 0.69151351, 0.68977632]),
 'fit_time': (6.3937668800354,
  4.894758224487305,
  5.34983491897583,
  5.2669336795806885,
  5.05420708656311),
 'test_time': (0.4426910877227783,
  0.15575289726257324,
  0.20410704612731934,
  0.16177892684936523,
  0.15295910835266113)}
```

```
trainset = data.build_full_trainset()
svd.fit(trainset)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7f4ec07ed080>
```

```
ratings[ratings['userId'] == 5]
```

	userId	movieId	rating	timestamp
351	5	3	4.0	1163374957
352	5	39	4.0	1163374952
353	5	104	4.0	1163374639
354	5	141	4.0	1163374242
355	5	150	4.0	1163374404
...	...	...	...	...
446	5	35836	4.0	1163374275
447	5	40819	4.5	1163374283
448	5	41566	4.0	1163374144
449	5	41569	4.0	1163374167
450	5	48385	4.5	1163374357

100 rows × 4 columns

```
svd.predict(5, 300, 3)
```

```
Prediction(uid=5, iid=300, r_ui=3, est=4.147823509772245, details={'was_impossible': False})
```

3. Content-Based System: Built two content-based engines; one that took movie overview and taglines as input and the other which took metadata such as cast, crew, genre and keywords to come up with predictions.

A) Overview and Tagline

```
#Import TfIdfVectorizer from scikit-learn
from sklearn.feature_extraction.text import TfidfVectorizer

#Define a TF-IDF Vectorizer Object. Remove all english stop words such as 'the', 'a'
tfidf = TfidfVectorizer(stop_words='english')

#Construct the required TF-IDF matrix by fitting and transforming the data
tfidf_matrix = tfidf.fit_transform(df2['description'])
```

```
#Output the shape of tfidf_matrix
tfidf_matrix.shape
```

```
(45555, 77746)
```

```
# Import linear_kernel
from sklearn.metrics.pairwise import linear_kernel

# Compute the cosine similarity matrix (similarity measure)
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

```python
# Function for description system
def get_recommendations(title, cosine_sim=cosine_sim):
    # Getting index of the movie that matches title
    idx = indices[title]

    # Pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sorting based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get score to Top 10 movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return df2['title'].iloc[movie_indices]
```

```python
get_recommendations('The Godfather')
```

```
1187                    The Godfather: Part II
44123    The Godfather Trilogy: 1972-1990
23197                              Blood Ties
1922                   The Godfather: Part III
32069                        Honor Thy Father
11339                        Household Saints
33560                  The Most Beautiful Wife
34814                       Start Liquidation
38126                 A Mother Should Be Loved
10860                                Election
Name: title, dtype: object
```

B) Cast, Crew and Genre:

```python
# Get the director's name from the crew feature.
# If director is not listed, return NaN
def get_director(x):
    for i in x:
        if i['job'] == 'Director':
            return i['name']
    return np.nan
```

```python
# Returns the list top 3 elements or entire list; whichever is more.
def get_list(x):
    if isinstance(x, list):
        names = [i['name'] for i in x]
        if len(names) > 3:
            names = names[:3]
        return names
    #Return empty list in case of missing/malformed data
    return []
```

```python
# All metadata that we want to feed to our vectorizer(namely actors, director and keywords).
def create_soup(x):
    return ' '.join(x['cast']) + ' ' + x['director'] + ' ' + ' '.join(x['genres'])
df2['soup'] = df2.apply(create_soup, axis=1)
```

```python
# Import CountVectorizer and create the count matrix
from sklearn.feature_extraction.text import CountVectorizer

count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(df2['soup'])
```

```python
# Compute the Cosine Similarity matrix based on the count_matrix
from sklearn.metrics.pairwise import cosine_similarity

cosine_sim2 = cosine_similarity(count_matrix, count_matrix)
```

```python
def get_recommendations(title, cosine_sim2=cosine_sim2):
    # Getting index of the movie that matches title
    idx = indices[title]

    # Pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim2[idx]))

    # Sorting by similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return df2['title'].iloc[movie_indices]
```

```python
get_recommendations('The Dark Knight Rises', cosine_sim2)
```

```
10158          Batman Begins
12525        The Dark Knight
28411            The Outsider
31570                Baseline
44043    The State Counsellor
516         Romeo Is Bleeding
9263                   Shiner
11399            The Prestige
20496               Rainy Dog
25400                    Tell
Name: title, dtype: object
```

C)New Improved (Popularity)

```python
def improved_recommendations(title):
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim2[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]

    movies = df2.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()
    m = vote_counts.quantile(0.60)

    def weighted_rating(x, m=m, C=C):
      v = x['vote_count']
      R = x['vote_average']
      # Calculation based on the IMDB formula
      return (v/(v+m) * R) + (m/(m+v) * C)

    qualified = movies[(movies['vote_count'] >= m) ]

    qualified['score'] = qualified.apply(weighted_rating, axis=1)
    qualified = qualified.sort_values('score', ascending=False).head(10)
    return qualified
```

```python
improved_recommendations('The Dark Knight Rises')
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:23: Se
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pan
```

	title	vote_count	vote_average	year	score
12525	The Dark Knight	12269.0	8.3	2008	8.295247
11399	The Prestige	4510.0	8.0	2006	7.988335
10158	Batman Begins	7511.0	7.5	2005	7.494220
35615	Mr. Six	30.0	7.4	2015	6.542716
33100	Agneepath	46.0	6.3	2012	5.971765
28768	Kidnapping Mr. Heineken	193.0	5.8	2015	5.743743
516	Romeo Is Bleeding	36.0	5.7	1993	5.516044
30685	Run	22.0	5.4	2013	5.290049
25400	Tell	21.0	5.4	2014	5.287273
31335	Hyena	32.0	5.3	2014	5.248538

4. Hybrid Recommendation System – Ideas from content and collaborative filtering are brought together to build an engine that gave movie suggestions to a particular user based on the estimated ratings that it had internally calculated for that user.

- Join the dataset with links csv file which contains userId information.

```
In [208]: def convert_int(x):
              try:
                  return int(x)
              except:
                  return np.nan
```

```
In [209]: id_map = pd.read_csv('links.csv')[['movieId', 'tmdbId']]
          id_map['tmdbId'] = id_map['tmdbId'].apply(convert_int)
          id_map.columns = ['movieId', 'id']
          id_map = id_map.merge(df2[['title', 'id']], on='id').set_index('title')
          #id_map = id_map.set_index('tmdbId')
```

```
In [210]: indices_map = id_map.set_index('id')
```

- Building the system

```
In [60]: def hybrid(userId, title):
             idx = indices[title]
             tmdbId = id_map.loc[title]['id']
             #print(idx)
             movie_id = id_map.loc[title]['movieId']

             sim_scores = list(enumerate(cosine_sim[int(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]

             movies = smd.iloc[movie_indices][['title', 'vote_count', 'vote_average', 'year', 'id']]
             movies['est'] = movies['id'].apply(lambda x: svd.predict(userId, indices_map.loc[x]['movieId']).est)
             movies = movies.sort_values('est', ascending=False)
             return movies.head(10)
```

- Result – UserId = 1 if watched “Avatar” than what to recommend him to watch next.

```
In [62]: hybrid(1, 'Avatar')
```

Out[62]:

	title	vote_count	vote_average	year	id	est
2466	The Matrix	9079.0	7.9	1999	603	3.087891
3546	Pandora and the Flying Dutchman	19.0	6.5	2051	38688	2.966712
2286	A Simple Plan	191.0	6.9	1998	10223	2.894891
30091	Success At Any Price	0.0	0.0	2034	105869	2.677548
35765	La Rabbia Di Pasolini	0.0	0.0	2008	15994	2.677548
38359	Veeram	11.0	5.8	2014	188540	2.677548
28745	Saints and Soldiers: The Void	22.0	5.2	2014	139334	2.677548
6740	Mobsters	34.0	5.7	1991	21219	2.677548
16135	Bloodbrothers	4.0	6.1	1978	114096	2.677548
33050	Beyond Darkness	3.0	6.0	1990	288154	2.677548

- Evaluation-

```python
svd = SVD()
cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.1846  0.1838  0.3749  0.1266  0.0116  0.1763  0.1176
MAE (testset)     0.1049  0.1111  0.2009  0.0825  0.0090  0.1017  0.0615
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.00    0.00    0.00    0.00    0.00    0.00    0.00

{'test_rmse': array([0.18461726, 0.18376988, 0.3748556 , 0.12657431, 0.01155334]),
 'test_mae': array([0.10487664, 0.11112223, 0.20089471, 0.08252767, 0.00904531]),
 'fit_time': (0.0017540454864501953,
  0.0020689964294433594,
  0.002664804458618164,
  0.0017161369323730469,
  0.0028481483459472656),
 'test_time': (8.606910705566406e-05,
  9.226799011230469e-05,
  8.034706115722656e-05,
  8.130073547363281e-05,
  0.00011706352233886719)}
```

Few visualizations:

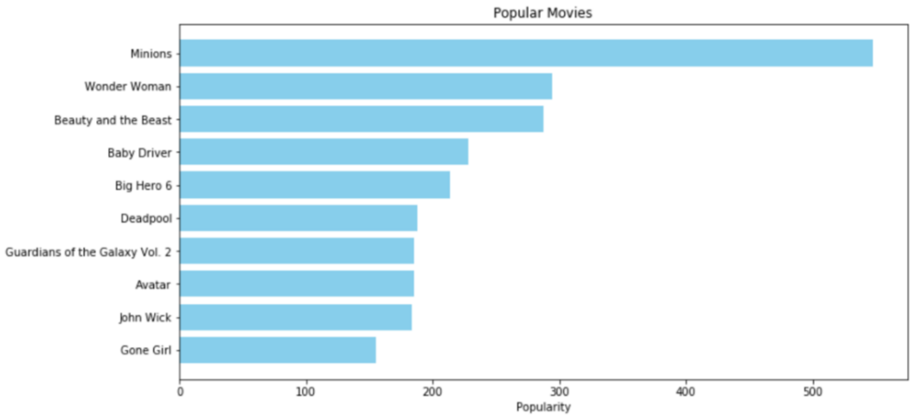
Word Cloud- This show how important is to use keywords in our system

```python
title_corpus = ' '.join(df2['title'])
overview_corpus = ' '.join(df2['overview'])

title_wordcloud = WordCloud(stopwords=STOPWORDS, background_color='white', height=2000, width=4000).generate(title_c
plt.figure(figsize=(16,8))
plt.imshow(title_wordcloud)
plt.axis('off')
plt.show()
```

Popularity Based Bar Chart- This shows how popularity affect the system

```
#popular movies visualization
pop= df2.sort_values('popularity', ascending=False)
import matplotlib.pyplot as plt
plt.figure(figsize=(12,6))

plt.barh(pop['title'].head(10),pop['popularity'].head(10), align='center',
        color='skyblue')
plt.gca().invert_yaxis()
plt.xlabel("Popularity")
plt.title("Popular Movies")
```

```
Text(0.5, 1.0, 'Popular Movies')
```

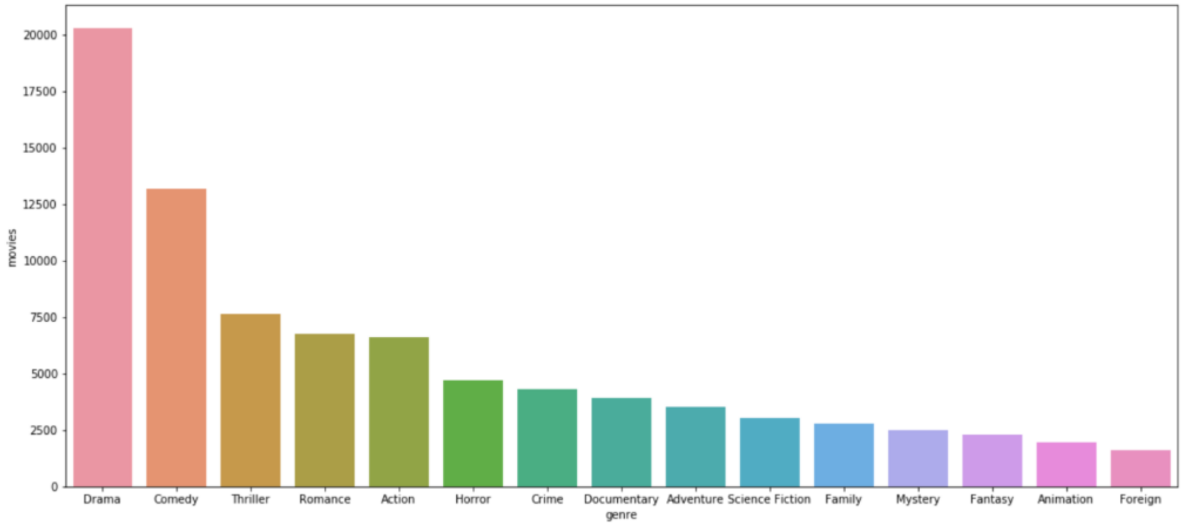


Genre-

```
pop_gen = pd.DataFrame(gen_md['genre'].value_counts()).reset_index()
pop_gen.columns = ['genre', 'movies']
pop_gen.head(10)
```

	genre	movies
0	Drama	20318
1	Comedy	13198
2	Thriller	7643
3	Romance	6749
4	Action	6608
5	Horror	4677
6	Crime	4318
7	Documentary	3937
8	Adventure	3506
9	Science Fiction	3055

```
plt.figure(figsize=(18,8))
sns.barplot(x='genre', y='movies', data=pop_gen.head(15))
plt.show()

#Drama is the most commonly occurring genre with almost half the movies identifying itself as a drama film
```



5. Evaluations and Results

5.1. Evaluation Methods

Evaluating this system on the basis of

1. Root Square Mean Error (RSME) and
2. Mean Absolute Error (MAE)

A) Collaborative Filtering – 5 Fold

```
data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)
```

```
svd = SVD()
cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8939  0.8989  0.8977  0.8952  0.8988  0.8969  0.0020
MAE (testset)     0.6866  0.6936  0.6931  0.6915  0.6898  0.6909  0.0025
Fit time          6.39    4.89    5.35    5.27    5.05    5.39    0.53
Test time         0.44    0.16    0.20    0.16    0.15    0.22    0.11
{'test_rmse': array([0.89391113, 0.89891495, 0.89765865, 0.89524431, 0.89881055]),
 'test_mae': array([0.68661831, 0.69356335, 0.69310485, 0.69151351, 0.68977632]),
 'fit_time': (6.3937668800354,
  4.894758224487305,
  5.34983491897583,
  5.2669336795806885,
  5.05420708656311),
 'test_time': (0.4426910877227783,
  0.15575289726257324,
  0.20410704612731934,
  0.16177892684936523,
  0.15295910835266113)}
```

B) Hybrid System – 5 Fold – Evaluating the system on specific user that's why I found better system. USERID=1 and MOVIE= "Avatar"

```
svd = SVD()
cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.1846  0.1838  0.3749  0.1266  0.0116  0.1763  0.1176
MAE (testset)     0.1049  0.1111  0.2009  0.0825  0.0090  0.1017  0.0615
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.00    0.00    0.00    0.00    0.00    0.00    0.00

{'test_rmse': array([0.18461726, 0.18376988, 0.3748556 , 0.12657431, 0.01155334]),
 'test_mae': array([0.10487664, 0.11112223, 0.20089471, 0.08252767, 0.00904531]),
 'fit_time': (0.0017540454864501953,
  0.0020689964294433594,
  0.002664804458618164,
  0.0017161369323730469,
  0.0028481483459472656),
 'test_time': (8.606910705566406e-05,
  9.226799011230469e-05,
  8.034706115722656e-05,
  8.130073547363281e-05,
  0.00011706352233886719)}
```

5.2. Results and Findings

I have built few recommendation systems above and the Hybrid System gave us best recommendation and much more accurate results than the collaborative filtering and other as we can observe that RMSE and MAE values in hybrid system is lower than collaborative filtering and lower the values better the results.

6. Conclusions and Future Work

6.1. Conclusions

- With this system, user will get new movie suggestions based on user queries by recommending the Top 10 movies which would save lot of their time and resources.
- Companies can make use of the system and can benefit from it by generating more revenue
- This is a project that can be extended way beyond the scope of this problem, and it can be applied in a wide range of contexts in addition to the movie industry.

6.2. Limitations

I created recommenders using demographic, content- based and collaborative filtering. While demographic filtering is very elementary and cannot be used practically, Hybrid Systems can take advantage of content-based and collaborative filtering as the two approaches are proved to be almost complimentary. This model was very baseline and only provides a fundamental framework to start with.

6.3. Potential Improvements or Future Work

Finally, a few things were not considered when building the engine and they should deserve some attention:

- The language of the film was not checked. This could be important to get sure that the films recommended are in the same language than the one chosen by user
- The replacement of the keywords by more frequent synonyms. In some cases, it was shown that the synonyms selected had a different meaning that the original word.

- Another Improvement could be to create a list of connections between actors to see which are the actors that use to play in similar movies. Hence, rather than only looking at the actors who are in the film selected by the user, we could enlarge this list by a few more people. Something similar could be done also with the directors
- Extend the detections of sequels to films that don't share similar titles like James Bond series