DSC 680 -PROJECT 1 - Movie Recommendation

Avinash Alapati

Recommendation Systems are a type of **information filtering systems** as they improve the quality of search results and provides items that are more relevant to the search item or are realted to the search history of the user.

They are used to predict the **rating** or **preference** that a user would give to an item. Almost every major tech company has applied them in some form or the other. Major companies like YouTube, Amazon, Netflix use recommendation systems in social and e-commerce sites use recommendation system for its users to suggest for an individual according to their requirement more precise and accurate. These online content and service providers have a huge amount of content so the problem which arises is which data is required for whom so the problem of providing apposite content frequently. This project represents the overview and approaches of techniques generated in a recommendation system.

The goal of this project is to provide a recommendation system for video content providers to predict whether someone will enjoy a movie based on how much they liked or disliked other movies.

There are basically three types of recommender systems:-

- **Demographic Filtering**- offers users with similar demographic background the similar movies that are popular and well-rated regardless of the genre or any other factors. Therefore, since it does not consider the individual taste of each person, it provides a simple result but easy to be implemented. The System recommends the same movies to users with similar demographic features. Since each user is different, this approach is too simple. The basic idea behind this system is that movies that are more popular and critically acclaimed will have a higher probability of being liked by the average audience.
- **Content Based Filtering**- consider the object's contents, this system uses item metadata, such as genre, director, description, actors, etc. for movies, to make these recommendations, it will give users the movie recommendation more closely to the individual's preference. They suggest similar items based on a particular item. The general idea behind these recommender systems is that if a person liked a particular item, he or she will also like an item that is similar to it.
- **Collaborative Filtering**- focuses on user's preference data and recommend movies based on it through matching with other users' historical movies that have a similar preference as well and does not require movies' metadata. This system matches persons with similar interests and provides recommendations based on this matching. Collaborative filters do not require item metadata like its content-based counterparts.

```python
## Import Libraries
import pandas as pd
import numpy as np
```

```
dfCredits=pd.read_csv('tmdb_5000_credits.csv')
dfMovies=pd.read_csv('tmdb_5000_movies.csv')

dfCredits.head()

   movie_id                                     title  \
0     19995                                    Avatar
1       285  Pirates of the Caribbean: At World's End
2    206647                                   Spectre
3     49026                     The Dark Knight Rises
4     49529                               John Carter

                                                cast  \
0  [{"cast_id": 242, "character": "Jake Sully", "...
1  [{"cast_id": 4, "character": "Captain Jack Spa...
2  [{"cast_id": 1, "character": "James Bond", "cr...
3  [{"cast_id": 2, "character": "Bruce Wayne / Ba...
4  [{"cast_id": 5, "character": "John Carter", "c...

                                                crew
0  [{"credit_id": "52fe48009251416c750aca23", "de...
1  [{"credit_id": "52fe4232c3a36847f800b579", "de...
2  [{"credit_id": "54805967c3a36829b5002c41", "de...
3  [{"credit_id": "52fe4781c3a36847f81398c3", "de...
4  [{"credit_id": "52fe479ac3a36847f813eaa3", "de...

dfMovies.head()

      budget                                             genres  \
0  237000000  [{"id": 28, "name": "Action"}, {"id": 12, "nam...
1  300000000  [{"id": 12, "name": "Adventure"}, {"id": 14, "...
2  245000000  [{"id": 28, "name": "Action"}, {"id": 12, "nam...
3  250000000  [{"id": 28, "name": "Action"}, {"id": 80, "nam...
4  260000000  [{"id": 28, "name": "Action"}, {"id": 12, "nam...

                                       homepage      id  \
0                   http://www.avatarmovie.com/   19995
1  http://disney.go.com/disneypictures/pirates/     285
2   http://www.sonypictures.com/movies/spectre/  206647
3            http://www.thedarkknightrises.com/   49026
4          http://movies.disney.com/john-carter   49529

                                            keywords original_language
\
0  [{"id": 1463, "name": "culture clash"}, {"id":...                en

1  [{"id": 270, "name": "ocean"}, {"id": 726, "na...                en

2  [{"id": 470, "name": "spy"}, {"id": 818, "name...                en

3  [{"id": 849, "name": "dc comics"}, {"id": 853,...                en
```

```
4  [{"id": 818, "name": "based on novel"}, {"id":...                en


                             original_title  \
0                                    Avatar
1  Pirates of the Caribbean: At World's End
2                                   Spectre
3                     The Dark Knight Rises
4                               John Carter

                                            overview  popularity  \
0  In the 22nd century, a paraplegic Marine is di...  150.437577
1  Captain Barbossa, long believed to be dead, ha...  139.082615
2  A cryptic message from Bond’s past sends him o...  107.376788
3  Following the death of District Attorney Harve...  112.312950
4  John Carter is a war-weary, former military ca...   43.926995

                                production_companies  \
0  [{"name": "Ingenious Film Partners", "id": 289...
1  [{"name": "Walt Disney Pictures", "id": 2}, {"...
2  [{"name": "Columbia Pictures", "id": 5}, {"nam...
3  [{"name": "Legendary Pictures", "id": 923}, {"...
4        [{"name": "Walt Disney Pictures", "id": 2}]

                                production_countries release_date
revenue  \
0  [{"iso_3166_1": "US", "name": "United States o...   2009-12-10
2787965087
1  [{"iso_3166_1": "US", "name": "United States o...   2007-05-19
961000000
2  [{"iso_3166_1": "GB", "name": "United Kingdom"...   2015-10-26
880674609
3  [{"iso_3166_1": "US", "name": "United States o...   2012-07-16
1084939099
4  [{"iso_3166_1": "US", "name": "United States o...   2012-03-07
284139100

   runtime                                   spoken_languages
status  \
0    162.0  [{"iso_639_1": "en", "name": "English"}, {"iso...
Released
1    169.0           [{"iso_639_1": "en", "name": "English"}]
Released
2    148.0  [{"iso_639_1": "fr", "name": "Fran\u00e7ais"},...
Released
3    165.0           [{"iso_639_1": "en", "name": "English"}]
Released
4    132.0           [{"iso_639_1": "en", "name": "English"}]
Released
```

```
                                          tagline  \
0                    Enter the World of Pandora.
1  At the end of the world, the adventure begins.
2                          A Plan No One Escapes
3                                The Legend Ends
4           Lost in our world, found in another.

                                      title  vote_average  vote_count

0                                    Avatar           7.2       11800

1  Pirates of the Caribbean: At World's End           6.9        4500

2                                   Spectre           6.3        4466

3                     The Dark Knight Rises           7.6        9106

4                               John Carter           6.1        2124
```

The credits dataset contains the following features:-

- movie_id - A unique identifier for each movie.
- cast - The name of lead and supporting actors.
- crew - The name of Director, Editor, Composer, Writer etc.

The Movie dataset has the following features:-

- budget - The budget in which the movie was made.
- genre - The genre of the movie, Action, Comedy ,Thriller etc.
- homepage - A link to the homepage of the movie.
- id - This is infact the movie_id as in the first dataset.
- keywords - The keywords or tags related to the movie.
- original_language - The language in which the movie was made.
- original_title - The title of the movie before translation or adaptation.
- overview - A brief description of the movie.
- popularity - A numeric quantity specifying the movie popularity.
- production_companies - The production house of the movie.
- production_countries - The country in which it was produced.
- release_date - The date on which it was released.
- revenue - The worldwide revenue generated by the movie.
- runtime - The running time of the movie in minutes.
- status - "Released" or "Rumored".
- tagline - Movie's tagline.
- title - Title of the movie.
- vote_average - average ratings the movie recieved.
- vote_count - the count of votes recieved.

```
## Joining both Datasets
dfCredits.columns = ['id','title','cast','crew']
dfMovies= dfMovies.merge(dfCredits,on='id')

dfMovies.head()
```

```
      budget                                             genres  \
0  237000000  [{"id": 28, "name": "Action"}, {"id": 12, "nam...
1  300000000  [{"id": 12, "name": "Adventure"}, {"id": 14, "...
2  245000000  [{"id": 28, "name": "Action"}, {"id": 12, "nam...
3  250000000  [{"id": 28, "name": "Action"}, {"id": 80, "nam...
4  260000000  [{"id": 28, "name": "Action"}, {"id": 12, "nam...

                                       homepage      id  \
0                   http://www.avatarmovie.com/   19995
1  http://disney.go.com/disneypictures/pirates/     285
2   http://www.sonypictures.com/movies/spectre/  206647
3            http://www.thedarkknightrises.com/   49026
4          http://movies.disney.com/john-carter   49529

                                            keywords original_language
\
0  [{"id": 1463, "name": "culture clash"}, {"id":...                en

1  [{"id": 270, "name": "ocean"}, {"id": 726, "na...                en

2  [{"id": 470, "name": "spy"}, {"id": 818, "name...                en

3  [{"id": 849, "name": "dc comics"}, {"id": 853,...                en

4  [{"id": 818, "name": "based on novel"}, {"id":...                en


                            original_title  \
0                                   Avatar
1  Pirates of the Caribbean: At World's End
2                                  Spectre
3                    The Dark Knight Rises
4                              John Carter

                                            overview  popularity  \
0  In the 22nd century, a paraplegic Marine is di...  150.437577
1  Captain Barbossa, long believed to be dead, ha...  139.082615
2  A cryptic message from Bond’s past sends him o...  107.376788
3  Following the death of District Attorney Harve...  112.312950
4  John Carter is a war-weary, former military ca...   43.926995

                                production_companies  ... runtime  \
0  [{"name": "Ingenious Film Partners", "id": 289...  ...   162.0
1  [{"name": "Walt Disney Pictures", "id": 2}, {"...  ...   169.0
2  [{"name": "Columbia Pictures", "id": 5}, {"nam...  ...   148.0
```

```
3  [{"name": "Legendary Pictures", "id": 923}, {"...  ...   165.0
4        [{"name": "Walt Disney Pictures", "id": 2}]  ...   132.0

                                    spoken_languages    status  \
0  [{"iso_639_1": "en", "name": "English"}, {"iso...  Released
1           [{"iso_639_1": "en", "name": "English"}]  Released
2  [{"iso_639_1": "fr", "name": "Fran\u00e7ais"},...  Released
3           [{"iso_639_1": "en", "name": "English"}]  Released
4           [{"iso_639_1": "en", "name": "English"}]  Released

                                         tagline  \
0                    Enter the World of Pandora.
1  At the end of the world, the adventure begins.
2                          A Plan No One Escapes
3                                The Legend Ends
4             Lost in our world, found in another.

                                    title_x vote_average vote_count  \
0                                    Avatar          7.2      11800
1  Pirates of the Caribbean: At World's End          6.9       4500
2                                   Spectre          6.3       4466
3                     The Dark Knight Rises          7.6       9106
4                               John Carter          6.1       2124

                                    title_y  \
0                                    Avatar
1  Pirates of the Caribbean: At World's End
2                                   Spectre
3                     The Dark Knight Rises
4                               John Carter

                                                cast  \
0  [{"cast_id": 242, "character": "Jake Sully", "...
1  [{"cast_id": 4, "character": "Captain Jack Spa...
2  [{"cast_id": 1, "character": "James Bond", "cr...
3  [{"cast_id": 2, "character": "Bruce Wayne / Ba...
4  [{"cast_id": 5, "character": "John Carter", "c...

                                                crew
0  [{"credit_id": "52fe48009251416c750aca23", "de...
1  [{"credit_id": "52fe4232c3a36847f800b579", "de...
2  [{"credit_id": "54805967c3a36829b5002c41", "de...
3  [{"credit_id": "52fe4781c3a36847f81398c3", "de...
4  [{"credit_id": "52fe479ac3a36847f813eaa3", "de...

[5 rows x 23 columns]
```

Demographic Filtering -

Before getting started with this -

- we need a metric to score or rate movie
- Calculate the score for every movie
- Sort the scores and recommend the best rated movie to the users.

We can use the average ratings of the movie as the score but using this won't be fair enough since a movie with 9 average rating and only 10 votes cannot be considered better than the movie with 8 as as average rating but 60 votes. So, we will be using IMDB's weighted rating (wr) which is given as :-

$$\text{Weighted Rating (WR)} = \left(\frac{v}{v+m} . R\right) + \left(\frac{m}{v+m} . C\right)$$

where,

- v is the number of votes for the movie;
- m is the minimum votes required to be listed in the chart;
- R is the average rating of the movie; And
- C is the mean vote across the whole report

We already have v(**vote_count**) and R (**vote_average**) and C can be calculated as

```
C= dfMovies['vote_average'].mean()
C

6.092171559442016
```

The mean rating for all the movies is approx 6 on a scale of 10.The next step is to determine an appropriate value for m, the minimum votes required to be listed in the chart. We will use 90th percentile as our cutoff. In other words, for a movie to feature in the charts, it must have more votes than at least 95% of the movies in the list.

```
m= dfMovies['vote_count'].quantile(0.95)
m

3040.8999999999996
```

Now, we can filter out the movies that qualify for the chart

```
filtered_movies = dfMovies.copy().loc[dfMovies['vote_count'] >= m]
filtered_movies.shape

(241, 23)
```

We see that there are 481 movies which qualify to be in this list. Now, we need to calculate our metric for each qualified movie. To do this, we will define a function, **weighted_rating()** and define a new feature **score**, of which we'll calculate the value by applying this function to our DataFrame of qualified movies:

```python
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()`
filtered_movies['score'] = filtered_movies.apply(weighted_rating,
axis=1)
```

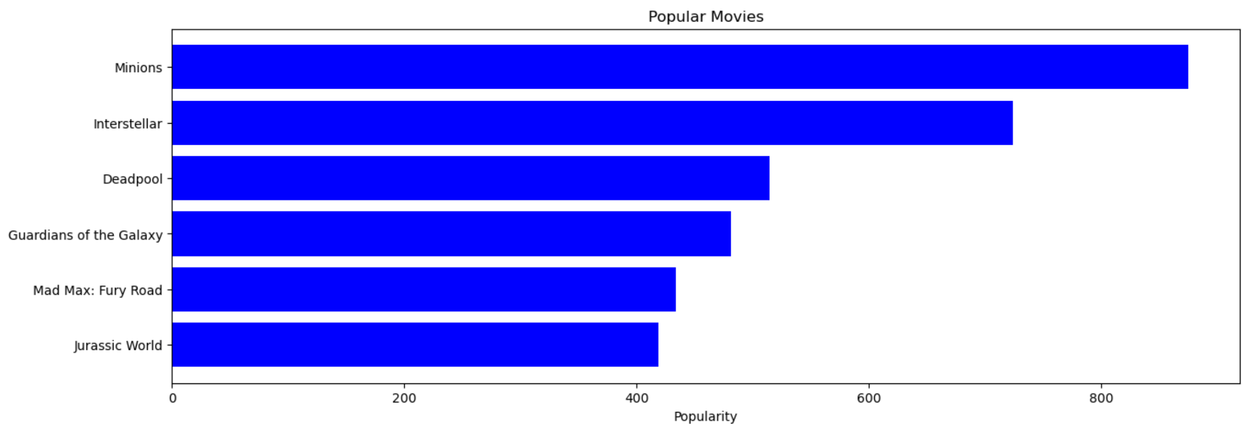
let's sort the DataFrame based on the score feature and output the title, vote count, vote average and weighted rating or score of the top 10 movies.

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

#Print the top 15 movies
filtered_movies[['original_title', 'vote_count', 'vote_average',
'score']].head(10)
```

```
                                         original_title  vote_count  \
1881                           The Shawshank Redemption        8205
65                                      The Dark Knight       12002
662                                          Fight Club        9413
96                                            Inception       13752
3232                                       Pulp Fiction        8428
95                                         Interstellar       10867
809                                        Forrest Gump        7927
3337                                      The Godfather        5893
329       The Lord of the Rings: The Return of the King        8064
262   The Lord of the Rings: The Fellowship of the Ring        8705

      vote_average     score
1881           8.5  7.848921
65             8.2  7.773906
662            8.3  7.760909
96             8.1  7.736417
3232           8.3  7.714609
95             8.1  7.660997
809            8.2  7.615595
3337           8.4  7.614467
329            8.1  7.550188
262            8.0  7.506082
```

Under the **Trending Now** tab of these systems we find movies that are very popular and they can just be obtained by sorting the dataset by the popularity column.

```
pop= dfMovies.sort_values('popularity', ascending=False)
import matplotlib.pyplot as plt
plt.figure(figsize=(15,5))

plt.barh(pop['original_title'].head(6),pop['popularity'].head(6),
align='center',
        color= 'blue')
plt.gca().invert_yaxis()
plt.xlabel("Popularity")
plt.title("Popular Movies")

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



The above demographic recommender provide a general chart of recommended movies to all the users. They are not sensitive to the interests and tastes of a particular user. In the following section we will do Content Basesd Filtering.

Content Based Filtering

In this recommender system the content of the movie (overview, cast, crew, keyword, tagline etc) is used to find its similarity with other movies. Then the movies that are most likely to be similar are recommended.

image-2.png

We will compute pairwise similarity scores for all movies based on their plot descriptions and recommend movies based on that similarity score.

```
dfMovies['overview'].head()

0    In the 22nd century, a paraplegic Marine is di...
1    Captain Barbossa, long believed to be dead, ha...
2    A cryptic message from Bond's past sends him o...
3    Following the death of District Attorney Harve...
```

```
4    John Carter is a war-weary, former military ca...
Name: overview, dtype: object
```

We will compute Term Frequency-Inverse Document Frequency (TF-IDF) vectors for each overview. Term Frequency : **(term instances/total instances)**. Inverse Document Frequency : **log(number of documents/documents with term)** The overall importance of each word to the documents in which they appear is equal to **TF * IDF**

This will give us a matrix where each column represents a word in the overview vocabulary and each row represents a movie. This is done to reduce the importance of words that occur frequently in plot overviews and therefore, their significance in computing the final similarity score.

We will use the library scikit-learn which gives us a built-in TfIdfVectorizer class that produces the TF-IDF matrix.

```
#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')

#Replace NaN with an empty string
dfMovies['overview'] = dfMovies['overview'].fillna('')

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

#Output the shape of tfidf_matrix
tfidf_matrix.shape

/opt/anaconda3/lib/python3.8/site-packages/scipy/__init__.py:146:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this
version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and
<{np_maxversion}"

(4803, 20978)
```

We see that over 20,000 different words were used to describe the 4800 movies in our dataset.

We will now compute a similarity score. There are several candidates for this; such as the euclidean, the Pearson and the cosine similarity scores. Different scores work well in different scenarios and it is often a good idea to experiment with different metrics.

We will be using the cosine similarity to calculate a numeric quantity that denotes the similarity between two movies. We use the cosine similarity score since it is independent of magnitude and is relatively easy and fast to calculate. Mathematically, it is defined as follows:

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

```
# Import linear_kernel
# we will use sklearn's **linear_kernel()** instead of
cosine_similarities() since it is faster.
from sklearn.metrics.pairwise import linear_kernel

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

We are going to define a function that takes in a movie title as an input and outputs a list of the 10 most similar movies. We need a reverse mapping of movie titles and DataFrame indices. We need a mechanism to identify the index of a movie in our metadata DataFrame, given its title.

```
#Construct a reverse map of indices and movie titles
indices = pd.Series(dfMovies.index,
index=dfMovies['original_title']).drop_duplicates()
```

These are the following steps we'll follow to define our recommender system :-

- Get the index of the movie given its title.
- Get the list of cosine similarity scores for that particular movie with all movies. Convert it into a list of tuples where the first element is its position and the second is the similarity score.
- Sort the mentioned list of tuples based on the similarity scores; that is, the second element.
- Get the top 10 elements of this list. Ignore the first element as it refers to self (the movie most similar to a particular movie is the movie itself).
- Return the titles corresponding to the indices of the top elements.

```
# Function that takes in movie title as input and outputs most similar
movies
def get_recommendations(title, cosine_sim=cosine_sim):
    # Get the index of the movie that matches the title
    idx = indices[title]

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

    # Sort the movies based on the 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 dfMovies['original_title'].iloc[movie_indices]
```

```
get_recommendations('The Dark Knight Rises')
```

```
65                              The Dark Knight
299                              Batman Forever
428                              Batman Returns
1359                                     Batman
3854    Batman: The Dark Knight Returns, Part 2
119                               Batman Begins
2507                                  Slow Burn
9            Batman v Superman: Dawn of Justice
1181                                        JFK
210                              Batman & Robin
Name: original_title, dtype: object
```

```
get_recommendations('The Avengers')
```

```
7               Avengers: Age of Ultron
3144                            Plastic
1715                            Timecop
4124                 This Thing of Ours
3311              Thank You for Smoking
3033                      The Corruptor
588     Wall Street: Money Never Sleeps
2136         Team America: World Police
1468                       The Fountain
1286                        Snowpiercer
Name: original_title, dtype: object
```

As we see, the quality of recommendations is not that great. "The Dark Knight Rises" returns all Batman movies while it is more likely that the people who liked that movie are more inclined to enjoy other Christopher Nolan movies. This is something that cannot be captured by the present system.

Improving Recommender System

The quality of our recommender would be increased with the usage of better metadata. For this we are going to build a recommender based on the following metadata: the 3 top actors, the director, related genres and the movie plot keywords.

From the cast, crew and keywords features, we need to extract the three most important actors, the director and the keywords associated with that movie.

```
# Parse the stringified features into their corresponding python
objects
from ast import literal_eval

features = ['cast', 'crew', 'keywords', 'genres']
for feature in features:
    dfMovies[feature] = dfMovies[feature].apply(literal_eval)
```

Next, we'll write functions that will help us to extract the required information from each feature.

```
# 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

# 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]
        #Check if more than 3 elements exist. If yes, return only
first three. If no, return entire list.
        if len(names) > 3:
            names = names[:3]
        return names

    #Return empty list in case of missing/malformed data
    return []

# Define new director, cast, genres and keywords features that are in
a suitable form.
dfMovies['director'] = dfMovies['crew'].apply(get_director)

features = ['cast', 'keywords', 'genres']
for feature in features:
    dfMovies[feature] = dfMovies[feature].apply(get_list)

# Print the new features of the first 3 films
dfMovies[['original_title', 'cast', 'director', 'keywords',
'genres']].head(3)
```

```
                              original_title  \
0                                     Avatar
1  Pirates of the Caribbean: At World's End
```

```
2                                   Spectre

                                              cast         director  \
0  [Sam Worthington, Zoe Saldana, Sigourney Weaver]   James Cameron   
1     [Johnny Depp, Orlando Bloom, Keira Knightley]  Gore Verbinski   
2      [Daniel Craig, Christoph Waltz, Léa Seydoux]      Sam Mendes   

                               keywords                        genres  
0   [culture clash, future, space war]  [Action, Adventure, Fantasy]  
1   [ocean, drug abuse, exotic island]  [Adventure, Fantasy, Action]  
2  [spy, based on novel, secret agent]    [Action, Adventure, Crime]  
```

The next step would be to convert the names and keyword instances into lowercase and strip all the spaces between them.

```
# Function to convert all strings to lower case and strip names of
spaces
def clean_data(x):
    if isinstance(x, list):
        return [str.lower(i.replace(" ", "")) for i in x]
    else:
        #Check if director exists. If not, return empty string
        if isinstance(x, str):
            return str.lower(x.replace(" ", ""))
        else:
            return ''

# Apply clean_data function to your features.
features = ['cast', 'keywords', 'director', 'genres']

for feature in features:
    dfMovies[feature] = dfMovies[feature].apply(clean_data)
```

Now we will create a string that contains all the metadata that we want to feed to our vectorizer (namely actors, director and keywords).

```
def create_metadata(x):
    return ' '.join(x['keywords']) + ' ' + ' '.join(x['cast']) + ' ' +
x['director'] + ' ' + ' '.join(x['genres'])
dfMovies['soup'] = dfMovies.apply(create_metadata, axis=1)
```

The next steps are the same as what we did with our plot description based recommender. One important difference is that we use the **CountVectorizer()** instead of TF-IDF. This is because we do not want to down-weight the presence of an actor/director if he or she has acted or directed in relatively more movies.

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

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

# 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)

# Reset index of our main DataFrame and construct reverse mapping as
before
dfMovies = dfMovies.reset_index()
indices = pd.Series(dfMovies.index, index=dfMovies['original_title'])
```

We can now reuse our **get_recommendations()** function by passing in the new **cosine_sim2** matrix as your second argument.

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

65                 The Dark Knight
119                  Batman Begins
4638    Amidst the Devil's Wings
1196                  The Prestige
3073             Romeo Is Bleeding
3326                Black November
1503                        Takers
1986                        Faster
303                       Catwoman
747                 Gangster Squad
Name: original_title, dtype: object

get_recommendations('The Godfather', cosine_sim2)

867       The Godfather: Part III
2731       The Godfather: Part II
4638    Amidst the Devil's Wings
2649             The Son of No One
1525                Apocalypse Now
1018               The Cotton Club
1170       The Talented Mr. Ripley
1209                The Rainmaker
1394                Donnie Brasco
1850                     Scarface
Name: original_title, dtype: object
```

After Applying the cosine similarity, we see that our recommender has been successful in capturing more information due to more metadata and has given us better recommendations. We can also increase the weight of the director , by adding the feature multiple times in the metadata.

Collaborative Filtering

The content based engine is only capable of suggesting movies which are close to a certain movie. It is not capable of capturing tastes and providing recommendations across genres.

Also, the engine that we built doesn't capture the personal tastes and biases of a user. Anyone querying our engine for recommendations based on a movie will receive the same recommendations for that movie, regardless of who she/he is.

Therefore, in this section, we will use Collaborative Filtering to make recommendations to Movie Watchers. It is basically of two types:-

- **User based filtering**- User-Based Collaborative Filtering is a technique used to predict the items that a user might like on the basis of ratings given to that item by the other users who have similar taste with that of the target user. Many websites use collaborative filtering for building their recommendation system. Imagine that we want to recommend a movie to our friend Stanley. We could assume that similar people will have similar taste. Suppose that me and Stanley have seen the same movies, and we rated them all almost identically. But Stanley hasn't seen 'The Godfather: Part II' and I did. If I love that movie, it sounds logical to think that he will too. With that, we have created an artificial rating based on our similarity. Well, UB-CF uses that logic and recommends items by finding similar users to the active user (to whom we are trying to recommend a movie). A specific application of this is the user-based Nearest Neighbor algorithm. This algorithm needs two tasks:

- Find the K-nearest neighbors (KNN) to the user a, using a similarity function w to measure the distance between each pair of users:

- Predict the rating that user a will give to all items the k neighbors have consumed but a has not. We Look for the item j with the best predicted rating. In other words, we are creating a User-Item Matrix, predicting the ratings on items the active user has not see, based on the other similar users. This technique is memory-based.

Although computing user-based CF is very simple, it suffers from several problems. One main issue is that users' preference can change over time. It indicates that precomputing the matrix based on their neighboring users may lead to bad performance. To tackle this problem, we can apply item-based CF.

- **Item Based Collaborative Filtering** - Instead of measuring the similarity between users, the item-based CF recommends items based on their similarity with the items that the target user rated. Likewise, the similarity can be computed with Pearson Correlation or Cosine Similarity. The major difference is that, with item-based collaborative filtering, we fill in the blank vertically, as opposed to the horizontal manner that user-based CF does.

image-2.png

It successfully avoids the problem posed by dynamic user preference as item-based CF is more static.

```
## We'll be using the Surprise library to implement SVD
from surprise import Reader, Dataset, SVD
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
```

Note that in this dataset movies are rated on a scale of 5 unlike the earlier one.

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

from surprise.model_selection import cross_validate
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)    0.8909  0.9022  0.8943  0.8907  0.9060  0.8968
0.0062
MAE (testset)     0.6853  0.6938  0.6895  0.6863  0.6990  0.6908
0.0051
Fit time          0.83    0.82    0.86    0.90    1.12    0.91    0.11

Test time         0.11    0.11    0.12    0.12    0.16    0.13    0.02

{'test_rmse': array([0.89087093, 0.90223474, 0.89428672, 0.8906609 ,
0.90604408]),
 'test_mae': array([0.68527761, 0.69383931, 0.68953129, 0.68631711,
0.69895368]),
 'fit_time': (0.8303709030151367,
  0.8244428634643555,
  0.8606112003326416,
  0.8993079662322998,
  1.1234490871429443),
 'test_time': (0.11418271064758301,
  0.11322021484375,
  0.12273001670837402,
  0.12232780456542969,
  0.16020417213439941)}
```

We get a mean Root Mean Sqaure Error of 0.89 approx which is more than good enough for our case. Let us now train on our dataset and arrive at predictions.

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

```
<surprise.prediction_algorithms.matrix_factorization.SVD at
0x7fd9be39d7c0>
```

Let us pick user with user Id 2 and check the ratings she/he has given.

```
ratings[ratings['userId'] == 2]
```

```
    userId  movieId  rating  timestamp
20       2       10     4.0  835355493
21       2       17     5.0  835355681
22       2       39     5.0  835355604
23       2       47     4.0  835355552
24       2       50     4.0  835355586
..     ...      ...     ...        ...
91       2      592     5.0  835355395
92       2      593     3.0  835355511
93       2      616     3.0  835355932
94       2      661     4.0  835356141
95       2      720     4.0  835355978

[76 rows x 4 columns]
```

```
svd.predict(2, 258, 3)
```

```
Prediction(uid=2, iid=258, r_ui=3, est=2.9696520238513715,
details={'was_impossible': False})
```

For movie with ID 258, we get an estimated prediction of **2.99**. This recommender system works on the basis of an assigned movie ID and tries to predict ratings based on how the other users have predicted the movie, it doesn't care what the movie is

Conclusion

We create recommenders using demographic , content- based and collaborative filtering. While demographic filtering is very elemantary 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.