Table of Contents

Title

Custom Made Movie Recommendation System

Abstract

The rapid expansion of data gathering has led to a new era of statistics. Data is being used to create more effective systems and this is where Recommendation Systems come into play. Recommendation Systems are a type of data sorting systems as they enhance the quality of search results and offers items that are more important to the search item or are linked to the search description of the consumer.

Introduction

As internet penetration in India is increasing, the number of users that can access internet is also increasing. With arrival of smartphone and reduction in cost of mobile data through out India have allowed more people to access net than ever before. As people now browse internet and generate huge amount of data now then ever. Big companies like Facebook, Google, YouTube are visited very frequently they try to collect as much data is possible to improve their local recommendation features. When ever someone searchers any thing or watches a video or search about any movies this data is saved by the companies for future use.

With rise in connectivity came rise in people using OTT. Over the top Content or OTT are streaming media services which over services to customer directly to home using internet connection. OTT companies by-pass all know traditional content distribution channel and bring content directly to customer. Some of these well know OTT companies are Disney+ Hotstar, Netflix, Amazon Prime etc. Now main function of these platform is to allow access to movie and recommend another movie in which they might be interested in. Now these companies use Machine Learning to improve their recommendation system and offer different types of recommendation based on Popularity.

Here we will try to create a movie recommendation system based on Demography and as well as show you Content based and Collaborative based filtering method which can be used to recommend movies to other.

Literature Review

There are basically three types of recommender systems: -

Popularity based: -

Perhaps, this is the simplest kind of recommendation engine that you will come across. The trending list you see in YouTube or Netflix is based on this algorithm. It keeps a track of view counts for each movie/video and then lists movies based on views in descending order.

Content based: -

This type of recommendation systems takes in a movie that a user currently likes as input. Then it analyses the contents (storyline, genre, cast, director etc.) of the movie to find out other movies which have similar content. Then it ranks similar movies according to their similarity scores and recommends the most relevant movies to the user.

Collaborative Filtering: -

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.

They are used to calculate the ranking or inclination that a user would give to an point. Almost every main tech corporation has applied them in some form or the other: Amazon uses it to recommend goods to customers, YouTube uses it to choose which video to play later on autopay, and Facebook uses it to advise pages to like and persons to follow. Moreover, companies like Netflix and Spotify hang very much on the usefulness of their recommendation mechanisms for their industry and achievement.

Machine Learning

Regression Analysis

Regression analysis is a form of predictive modelling technique which investigates the relationship between a dependent (target) and independent variable (s) (predictor). This technique is used for forecasting, time series modelling and finding the causal effect relationship between the variables

Regression analysis is an important tool for modelling and analysing data. Here, we fit a curve / line to the data points, in such a manner that the differences between the distances of data points from the curve or line is minimized. regression analysis estimates the relationship between two or more variables

There are multiple benefits of using regression analysis. They are as follows:

- It indicates the significant relationships between dependent variable and independent variable.
- It indicates the strength of impact of multiple independent variables on a dependent variable.

Dataset

The first 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 second 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.

<u>Demographic Recommendation -</u>

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 8.9 average rating and only 3 votes cannot be considered better than the movie with 7.8 as as average rating but 40 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

Credits, Genres and Keywords Based Recommender

It goes without saying that the quality of our recommender would be increased with the usage of better metadata. That is exactly what we are going to do in this section. 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. Right now, our data is present in the form of "stringified" lists , we need to convert it into a safe and usable structure

Implementation & Code

Step 1) Importing and loading data sets

```
import pandas as pd

import numpy as np

df1=pd.read_csv('tmdb_5000_credits.csv')

df2=pd.read_csv('tmdb_5000_movies.csv')
```

Step2) Merging column and viewing Top 5 movies

```
df1.columns = ['id','tittle','cast','crew']

df2= df2.merge(df1,on='id')
```

df2.head(5)

Out[4]:

	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	production_com
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	en	Avatar	In the 22nd century, a paraplegic Marine is di...	150.437577	[{"name": "Ing Film Partner
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	139.082615	[{"name": "Walt Pictures", "id":
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	en	Spectre	A cryptic message from Bond's past sends him o...	107.376788	[{"name": "Cc Pictures", ' {
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	http://www.thedarkknightrises.com/	49026	[{"id": 849, "name": "dc comics"}, {"id": 853,...	en	The Dark Knight Rises	Following the death of District Attorney Harve...	112.312950	[{"name": "Leg Pictures", "id": 92
4	260000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://movies.disney.com/john-carter	49529	[{"id": 818, "name": "based on novel"}, {"id":...	en	John Carter	John Carter is a war-weary, former military ca...	43.926995	[{"name": "Walt Pictures", '

5 rows × 23 columns

Step3) Calculating mean votes across whole file(c)

```
C= df2['vote_average'].mean()
C
```

Out[5]: 6.092171559442011

Step4) Now we will determine an apt value for m, the minimum votes required to be listed in the chart. We will use 90th percentile as our cut-off.

```
m= df2['vote_count'].quantile(0.9)
m
```

Out[6]: 1838.4000000000015

Step5) Now we will file the movies in this step

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

```
Out[7]: (481, 23)
```

We can see 481 movies have been selected.

Step6) Now we will calculate a weighted rating for all those movies

```
def weighted_rating(x, m=m, C=C):
v = x['vote_count']
R = x['vote_average']
 return (v/(v+m) * R) + (m/(m+v) * C)
```

Step7) Define a new score and assign weighted value to it.

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

Step8) Sort movies based on above score just calculated

```
q_movies = q_movies.sort_values('score', ascending=False)
```

Step9) Print top 10 movies

```
q_movies[['title', 'vote_count', 'vote_average', 'score']].head(10)
```

	title	vote_count	vote_average	score
1881	The Shawshank Redemption	8205	8.5	8.059258
662	Fight Club	9413	8.3	7.939256
65	The Dark Knight	12002	8.2	7.920020
3232	Pulp Fiction	8428	8.3	7.904645
96	Inception	13752	8.1	7.863239
3337	The Godfather	5893	8.4	7.851236
95	Interstellar	10867	8.1	7.809479
809	Forrest Gump	7927	8.2	7.803188
329	The Lord of the Rings: The Return of the King	8064	8.1	7.727243
1990	The Empire Strikes Back	5879	8.2	7.697884

Step10) Printing top 10 movies based on popularity

```
q_movies = q_movies.sort_values('popularity', ascending=False)
q_movies[['title','popularity']].head(10)
```

Out[50]:

	title	popularity
546	Minions	875.581305
95	Interstellar	724.247784
788	Deadpool	514.569956
94	Guardians of the Galaxy	481.098624
127	Mad Max: Fury Road	434.278564
28	Jurassic World	418.708552
199	Pirates of the Caribbean: The Curse of the Bla...	271.972889
82	Dawn of the Planet of the Apes	243.791743
200	The Hunger Games: Mockingjay - Part 1	206.227151
88	Big Hero 6	203.734590

Step11) Finding overview of top 5 movies

```
df2['overview'].head(5)
```

```
Out[12]: 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
```

Step12) Finding total number of words used to describe movies

```
#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

df2['overview'] = df2['overview'].fillna('')
```

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

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

Step13) Finding cosine similarity

```
# Import linear_kernel
from sklearn.metrics.pairwise import linear_kernel
# Compute the cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)


#Construct a reverse map of indices and movie titles
indices = pd.Series(df2.index, index=df2['title']).drop_duplicates()
```

Step14) Use the cosine similarity to sort movies

```
# 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 df2['title'].iloc[movie_indices]
```

Step15)Enter a movie name to get recommendation

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

```
Out[17]: 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: title, dtype: object
```

Improved Content based recommendation system

Step16) Take multiple keywords as for recommendation

```
# Parse the stringified features into their corresponding python objects
from ast import literal_eval
features = ['cast', 'crew', 'keywords', 'genres']
for feature in features:
df2[feature] = df2[feature].apply(literal_eval)
# 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 []

Step16) Clean data names so no double instance

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:

df2[feature] = df2[feature].apply(clean_data)

Step17) Create a new string of metadata to be used for filtering

def create_soup(x):

return ' '.join(x['keywords']) + ' ' + ' '.join(x['cast']) + ' ' + x['director'] + ' ' + ' '.join(x['genres'])

df2['soup'] = df2.apply(create_soup, axis=1)

Step18) Crate a count matrix

```
# 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'])
```

Step19) Initialize new cosine silimarity based on count matrix

```
# 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)
```

Step20) Get recommendations

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

```
Out[29]: 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: title, dtype: object
```

Result

From the above implementation we can see that our Demographic recommender and Popularity based recommender are very different.

Demographic based recommendation system: -

-

	title	vote_count	vote_average	score
1881	The Shawshank Redemption	8205	8.5	8.059258
662	Fight Club	9413	8.3	7.939256
65	The Dark Knight	12002	8.2	7.920020
3232	Pulp Fiction	8428	8.3	7.904645
96	Inception	13752	8.1	7.863239
3337	The Godfather	5893	8.4	7.851236
95	Interstellar	10867	8.1	7.809479
809	Forrest Gump	7927	8.2	7.803188
329	The Lord of the Rings: The Return of the King	8064	8.1	7.727243
1990	The Empire Strikes Back	5879	8.2	7.697884

Demographic based given above shows us that how likely a certain group of people watched a movie and liked it and gave it a high rating as its weighted score can be seen above

Popularity based recommendation system: -

Out[50]:

	title	popularity
546	Minions	875.581305
95	Interstellar	724.247784
788	Deadpool	514.569956
94	Guardians of the Galaxy	481.098624
127	Mad Max: Fury Road	434.278564
28	Jurassic World	418.708552
199	Pirates of the Caribbean: The Curse of the Bla...	271.972889
82	Dawn of the Planet of the Apes	243.791743
200	The Hunger Games: Mockingjay - Part 1	206.227151
88	Big Hero 6	203.734590

On the other hand, the popularity-based recommender shows us how likely a certain group of view watches a movie and what is the most view among them.

Normal Content based recommendation system: -

```
Out[17]: 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: title, dtype: object
```

Here we can see that using a single parameter for finding similar movies yield this result and it is not very accurate.

Improved Content based recommendation system: -

```
Out[29]: 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: title, dtype: object
```

Here we can see the recommendation system uses four factors while suggesting similar movies and they are cast, keyword , director , genre.

Conclusion

In this project we have seen how many different type of recommendation system can be made and we have created a different recommendation system called Demographic filtering recommendation system that used weighted rating to sort movies. This is beneficial as popularity-based recommendation system uses most viewed data to show recommendation.

On other hand the improved Content based recommendation system used four parameters instead of one or two thus increasing its effectiveness and offer a more wide and different approach rather than showing similar movies based on one parameters.

References

1. https://www.kaggle.com/rounakbanik/movie-recommender-systems
2. https://towardsdatascience.com/how-to-build-a-simple-recommender-system-in-python-375093c3fb7d
3. https://scikit-learn.org/stable/tutorial/basic/tutorial.html
4. https://datascience.stackexchange.com/questions/12101/machine-learning-technique-to-calculate-weighted-average-weights