Content Based Movie Recommendation Systems

Jeya Santhosh Kumar D

19BCE0762

Submitted to
Prof. Geraldine Bessie Amali, SCOPE

School of Computer Science and Engineering

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Contents

S.No	Topic	Page No
1.	Introduction	4
2.	Hardware Requirements	4
3.	System Requirements	4
4.	Existing Systems	5
4.1	Drawbacks of Existing Systems	5
5.	Proposed System	5
5.1	Design	5
5.2	Module Wise Description	6
5.3	Implementation	6
6.	Results and Discussions	9
7.	Conclusion	10
8.	References	10
9.	Sample Code	11

Abstract

A movie recommendation is important in our social life due to its strength in providing enhanced entertainment. Such a system can suggest a set of movies to users based on their interest, or the popularities of the movies. Although, a set of movie recommendation systems have been proposed, most of these either cannot recommend a movie to the existing users efficiently or to a new user by any means. In this paper we propose a movie recommendation system that has the ability to recommend movies to a new user as well as the others. It mines movie databases to collect all the important information, such as, popularity and attractiveness, required for recommendation.

1.Introduction

A recommendation system is a type of information filtering system which challenges to assume the priorities of a user, and make recommendations on the basis of user's priorities. Huge range of applications of recommendation systems are provided to the user. The popularity of recommendations systems have gradually increased and are recently implemented in almost all online platforms that people use. The content of such system differs from films, podcasts, books and videos, to colleagues and stories on social media, to commodities on e-commerce websites, to people on commercial and dating websites. Often, these systems are able to retrieve and filter data about a user's preferences, and can use this intel to advance their suggestions in the upcoming period. For an instance, Twitter can analyze your collaboration with several stories on your wall so as to comprehend what types of stories please you. Many a times, these systems can be improvised on the basis of activities of a large number of people. For example, if Flipkart notices that a large number of users who buy the modern laptop also buy a laptop bag. They can commend the laptop bag to a new customer who has just added a laptop to his cart. Due to the advances in recommender systems, users continuously expect good results. They have a low edge for services that are not able to make suitable recommendations. If a music streaming application is not able to foresee and play song that the user prefers, then the user will just stop using it. This has led to a high importance by technical corporations on refining their recommendation structures. However, the problem is more complicated than it appears. Every user has different likes and dislikes. In addition, even the taste of a single customer can differ depending on a large number of aspects, such as mood, season, or type of activity the user is performing. For an instance, the type of music one would prefer to listen during exercising varies critically from the type of music he would listen to while preparing dinner. They must discover new areas to determine more about the customer, whilst still determining almost all of what is already known about of the customer. Two critically important methods are widely used for recommender systems. One is content-based filtering, where we attempt to shape the users preferences using data retrieved, and suggest items based on the movie which he likes.

2.Hardware Requirements

Processor: Intel i5 4th Generation

VRAM:NVIDIA GTX 1050 and Above

RAM:4 GB

3.Software Requirements

Jupyter Notebooks

Python 3.0

Python IDE

All Required Packages required for the Code

4.Existing System

Movie recommendation systems which are existing have poor efficiency due to which movies are suggested in view of aspects for example - movie rated & evaluated by the User. They have almost same viewing tastes, by means of data mining

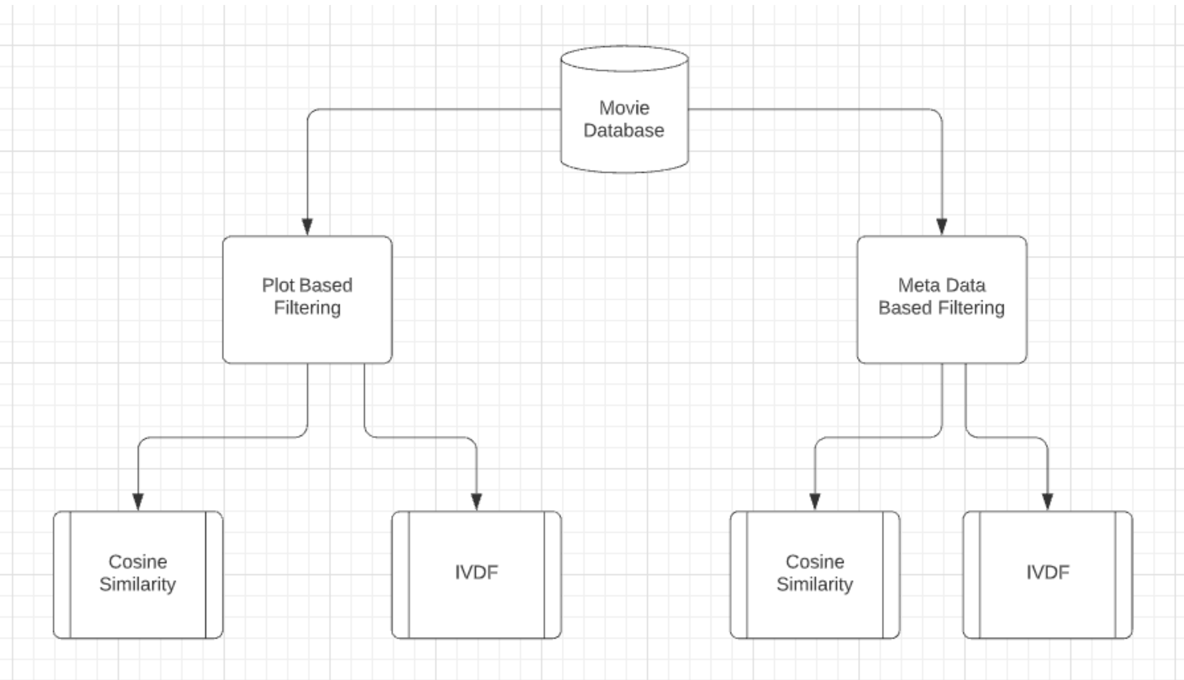
4.1 Drawbacks of Existing Systems

Collaborative filtering systems are based on the action of available data from similar users. If you are building a brand new recommendation system, you would have no user data to start with Ratings provided by User is not always accurate. Some People rate movies with a political mindset which will definitely affect the movie watching experience of other Users.

5.Proposed Model

In this Project ,I have proposed 2 ways of content filtering. One Filtering uses the Meta Data and the Other one Uses the Overall Plot of the Movie. I have Implemented 2 Algorithms which can handle the Filtering. One is Cosine Filtering and other is Tf-IDVF

5.1 Design



5.2 Module Wise Description

Cosine Similarity

Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. It is often used to measure document similarity in text analysis. **Cosine similarity** is a measure of similarity that can be used to compare documents or, say, give a ranking of documents with respect to a given vector of query words. Let x and y be two vectors for comparison.

Term Frequency Inverse Document Frequency

It is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. tf–idf is one of the most popular term-weighting schemes today. A survey conducted in 2015 showed that 83% of text-based recommender systems in digital libraries use tf–idf.

Plot Based Filtering

The Plot of the Movie is Filtered.

Meta Data Based Filtering

The Feature of the Movie Database is Combined and it is Filtered.

5.3 Implementation

Importing Libraries

```
In [1]: import pandas as pd
        import numpy as np
        import nltk
```

```
In [2]: movies = pd.read_csv("movie_dataset.csv")
        from nltk.corpus import stopwords
        stop = stopwords.words('english')
```

Data Cleaning and Feature Engineering

```
In [4]: #Feature Engineering
        movies_cleaned = movies.drop(columns=['homepage','production_countries'])
        features = ['keywords', 'cast', 'genres', 'director']
        for feature in features:
            movies_cleaned[feature] = movies_cleaned[feature].fillna('')
        def combined_features(row):
            return row['keywords']+" "+row['cast']+" "+row['genres']+" "+row['director']
        movies_cleaned["combined_features"] = movies_cleaned.apply(combined_features, axis =1)
        pat = r'\b(?:{})\b'.format('|'.join(stop))
        movies_cleaned['overview_without_stopwords'] = movies_cleaned['overview'].str.replace(pat, '')
        movies_cleaned['overview_without_stopwords'] = movies_cleaned['overview_without_stopwords'].str.replace(r'\s+', ' ')
```

Plot Based Filtering With Tf IVDF

In [6]:
```
#TFIDF Plot Based Filtering
from sklearn.feature_extraction.text import TfidfVectorizer
tfv = TfidfVectorizer(min_df=3,  max_features=None,
            strip_accents='unicode', analyzer='word',token_pattern=r'\w{1,}',
            ngram_range=(1, 3),
            stop_words = 'english')
```

In [7]:
```
tfv_matrix = tfv.fit_transform(movies_cleaned['overview_without_stopwords'].values.astype('U'))
```

In [8]:
```
from sklearn.metrics.pairwise import sigmoid_kernel
```

In [9]:
```
sig = sigmoid_kernel(tfv_matrix, tfv_matrix)
```

In [10]:
```
indices = pd.Series(movies_cleaned.index, index=movies_cleaned['original_title']).drop_duplicates()
```

In [11]:
```
def give_recomendationsivdfpf(title, sig=sig):
    idx = indices[title]
    sig_scores = list(enumerate(sig[idx]))
    sig_scores = sorted(sig_scores, key=lambda x: x[1], reverse=True)
    sig_scores = sig_scores[1:11]
    movie_indices = [i[0] for i in sig_scores]
    return movies_cleaned['original_title'].iloc[movie_indices]
```

Plot Based Filtering With Cosine Similarity

In [13]:
```
#Cosine Similarity with Plot Based Filtering
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

In [14]:
```
cv = CountVectorizer()
count_matrix = cv.fit_transform(movies_cleaned["overview_without_stopwords"].values.astype('U'))
print("Count Matrix:", count_matrix.toarray())
```

```
Count Matrix: [[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]]
```

In [15]:
```
cosine_sim = cosine_similarity(count_matrix)
print(cosine_sim)
```

```
[[1.         0.         0.         ... 0.         0.         0.        ]
 [0.         1.         0.04260143 ... 0.03311331 0.         0.        ]
 [0.         0.04260143 1.         ... 0.02680281 0.         0.        ]
 ...
 [0.         0.03311331 0.02680281 ... 1.         0.04003204 0.02041241]
 [0.         0.         0.         ... 0.04003204 1.         0.05883484]
 [0.         0.         0.         ... 0.02041241 0.05883484 1.        ]]
```

In [16]:
```
def getrecomcosp(title):
    movie_index = movies_cleaned[movies_cleaned.title == title]["index"].values[0]
    similar_movies = list(enumerate(cosine_sim[movie_index]))
    sorted_similar_movies = sorted(similar_movies, key=lambda x:x[1], reverse=True)
    i=0
    for movie in sorted_similar_movies:
        print(movies_cleaned[movies_cleaned.index == movie[0]]["title"].values[0])
        i=i+1
        if i>15:
            break
```

Meta Data Based Filtering With Tf IVDF

In [19]:
```
#Meta Data Based Filtering With TfIDF
tfv2 = TfidfVectorizer(min_df=3,  max_features=None,
            strip_accents='unicode', analyzer='word',token_pattern=r'\w{1,}',
            ngram_range=(1, 3),
            stop_words = 'english')
```

In [21]:
```
tfv_matrix2 = tfv2.fit_transform(movies_cleaned['combined_features'].values.astype('U'))
```

In [22]:
```
sig2 = sigmoid_kernel(tfv_matrix2, tfv_matrix2)
```

In [23]:
```
indices2 = pd.Series(movies_cleaned.index, index=movies_cleaned['original_title']).drop_duplicates()
```

In [26]:
```
def give_recomendationsivdfmf(title, sig=sig):
    idx = indices2[title]
    sig_scores = list(enumerate(sig[idx]))
    sig_scores = sorted(sig_scores, key=lambda x: x[1], reverse=True)
    sig_scores = sig_scores[1:11]
    movie_indices = [i[0] for i in sig_scores]
    return movies_cleaned['original_title'].iloc[movie_indices]
```

Meta Data Based Filtering With Cosine Similarity

In [28]:
```
#Metadata based Filtering With Cosine Similarity
cv2 = CountVectorizer()
count_matrix2 = cv2.fit_transform(movies_cleaned["combined_features"])
print("Count Matrix:", count_matrix2.toarray())
```

```
Count Matrix: [[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]]
```

In [29]:
```
cosine_sim2 = cosine_similarity(count_matrix2)
```

In [31]:
```
def getrecomcosmf(title):
    movie_index = movies_cleaned[movies_cleaned.title == title]["index"].values[0]
    similar_movies = list(enumerate(cosine_sim2[movie_index]))
    sorted_similar_movies = sorted(similar_movies, key=lambda x:x[1], reverse=True)
    i=0
    for movie in sorted_similar_movies:
        print(movies_cleaned[movies_cleaned.index == movie[0]]["title"].values[0])
        i=i+1
        if i>15:
            break
```

6.Results and Discussion

Plot Based Filtering With Tf IVDF

```
give_recomendationsivdfpf('The Dark Knight')
```

```
3                            The Dark Knight Rises
299                                 Batman Forever
428                                 Batman Returns
3854      Batman: The Dark Knight Returns, Part 2
1181                                           JFK
1359                                        Batman
2507                                     Slow Burn
879                            Law Abiding Citizen
119                                  Batman Begins
205            Sherlock Holmes: A Game of Shadows
Name: original_title, dtype: object
```

Plot Based Filtering With Cosine Similarity

```
getrecomcosp('The Dark Knight')
```

```
The Dark Knight
The Dark Knight Rises
Batman
Batman: The Dark Knight Returns, Part 2
Batman Returns
Batman Forever
Batman Begins
Blackhat
Despicable Me 2
JFK
The Railway Man
Puss in Boots
The Man from U.N.C.L.E.
Inglourious Basterds
The Transporter Refueled
Django Unchained
```

Meta Data Based Tf-IVDF Filtering

```
give_recomendationsivdfmf('The Dark Knight')
```

```
3                            The Dark Knight Rises
299                                 Batman Forever
428                                 Batman Returns
3854      Batman: The Dark Knight Returns, Part 2
1181                                           JFK
1359                                        Batman
2507                                     Slow Burn
879                            Law Abiding Citizen
119                                  Batman Begins
205            Sherlock Holmes: A Game of Shadows
Name: original_title, dtype: object
```

Meta Data Based Cosine Similarity Filtering

```
getrecomcosmf('The Dark Knight')
```

```
The Dark Knight
The Dark Knight Rises
Batman Begins
Amidst the Devil's Wings
The Prestige
Kick-Ass
Kick-Ass 2
Batman Returns
Batman
The Killer Inside Me
Batman & Robin
Harry Brown
In Too Deep
Defendor
Point Blank
Harsh Times
```

7.Conclusion:

We get a total of 4 Different Recommendations involving different features and with different Algorithms. It is based upon the User which Algorithms he prefers out of these 4.I Prefer the Meta Data Based Cosine Similarity Filtering. This project can be further enhanced by combining all the results of these 4 and finding an Optimal 10 Movies from it.

References:

1.Ghuli, P., Ghosh, A., Shettar, R.: A collaborative filtering recommendation engine in a distributed environment. In: 2014 International Conference on Contemporary Computing and Informatics (IC3I). IEEE (2014)

2.Zhao, L., et al.: Matrix factorization + for movie recommendation. In: IJCAI (2016)

Bhatt, B.: A review paper on machine learning based recommendation system. Int. J. Eng. Dev. Res. (2014)

3.Debnath, S., Ganguly, N., Mitra, P.: Feature weighting in content based recommendation system using social network analysis. In: Proceedings of the 17th International Conference on World Wide Web. ACM (2008)

4.SRS Reddy, Sravani Nalluri, Content Based Movie Recommendation using Genre Correlation

Code:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import sigmoid_kernel
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import nltk
movies = pd.read_csv("movie_dataset.csv")
from nltk.corpus import stopwords
stop = stopwords.words('english')
movies_cleaned = movies.drop(columns=['homepage','production_countries'])
features = ['keywords', 'cast', 'genres', 'director']
for feature in features:
    movies_cleaned[feature] = movies_cleaned[feature].fillna('')
def combined_features(row):
    return row['keywords']+" "+row['cast']+" "+row['genres']+" "+row['director']
movies_cleaned["combined_features"] = movies_cleaned.apply(combined_features, axis
=1)
pat = r'\b(?:{})\b'.format('|'.join(stop))
movies_cleaned['overview_without_stopwords'] =
movies_cleaned['overview'].str.replace(pat, '')
movies_cleaned['overview_without_stopwords'] =
movies_cleaned['overview_without_stopwords'].str.replace(r'\s+', ' ')
#TFIDF Plot Based Filtering
tfv = TfidfVectorizer(min_df=3,  max_features=None,
            strip_accents='unicode', analyzer='word',token_pattern=r'\w{1,}',
            ngram_range=(1, 3),
            stop_words = 'english')
tfv_matrix =
tfv.fit_transform(movies_cleaned['overview_without_stopwords'].values.astype('U'))
sig = sigmoid_kernel(tfv_matrix, tfv_matrix)
indices = pd.Series(movies_cleaned.index,
index=movies_cleaned['original_title']).drop_duplicates()
def give_recomendationsivdfpf(title, sig=sig):
```

```
    idx = indices[title]
    sig_scores = list(enumerate(sig[idx]))
    sig_scores = sorted(sig_scores, key=lambda x: x[1], reverse=True)
    sig_scores = sig_scores[1:11]
    movie_indices = [i[0] for i in sig_scores]
    return movies_cleaned['original_title'].iloc[movie_indices]
#Cosine Similarity With Plot Based Filtering
cv = CountVectorizer()
count_matrix =
cv.fit_transform(movies_cleaned["overview_without_stopwords"].values.astype('U'))
cosine_sim = cosine_similarity(count_matrix)
def getrecomcosp(title):
    movie_index = movies_cleaned[movies_cleaned.title == title]["index"].values[0]
    similar_movies = list(enumerate(cosine_sim[movie_index]))
    sorted_similar_movies = sorted(similar_movies, key=lambda x:x[1], reverse=True)
    i=0
    for movie in sorted_similar_movies:
        print(movies_cleaned[movies_cleaned.index == movie[0]]["title"].values[0])
        i=i+1
        if i>15:
            break
#Meta Data Based Filtering With TfIDF
tfv2 = TfidfVectorizer(min_df=3,  max_features=None,
            strip_accents='unicode', analyzer='word',token_pattern=r'\w{1,}',
            ngram_range=(1, 3),
            stop_words = 'english')
tfv_matrix2 = tfv2.fit_transform(movies_cleaned['combined_features'].values.astype('U'))
sig2 = sigmoid_kernel(tfv_matrix2, tfv_matrix2)
indices2 = pd.Series(movies_cleaned.index,
index=movies_cleaned['original_title']).drop_duplicates()
def give_recomendationsivdfmf(title, sig=sig):
    idx = indices2[title]
    sig_scores = list(enumerate(sig[idx]))
```

```
    sig_scores = sorted(sig_scores, key=lambda x: x[1], reverse=True)
    sig_scores = sig_scores[1:11]
    movie_indices = [i[0] for i in sig_scores]
    return movies_cleaned['original_title'].iloc[movie_indices]
#Metadata based Filtering With Cosine Similarity
cv2 = CountVectorizer()
count_matrix2 = cv2.fit_transform(movies_cleaned["combined_features"])
cosine_sim2 = cosine_similarity(count_matrix2)
def getrecomcosmf(title):
    movie_index = movies_cleaned[movies_cleaned.title == title]["index"].values[0]
    similar_movies = list(enumerate(cosine_sim2[movie_index]))
    sorted_similar_movies = sorted(similar_movies, key=lambda x:x[1], reverse=True)
    i=0
    for movie in sorted_similar_movies:
        print(movies_cleaned[movies_cleaned.index == movie[0]]["title"].values[0])
        i=i+1
        if i>15:
            break
print('Welcome TO Movie Recommendations Systems')
c='0'
while(c=='0'):
    movieulike=input("Enter The Movie You Like:")
    ch=input(" Enter 1 for Plot Based TfIVDF Filtering \n Enter 2 for Plot Based Cosine
Similarity Filtering \n Enter 3 for Meta Data Based TfIVDF Filtering \n Enter 4 for Meta
Data based Cosine Similarity Filtering")
    if(ch=='1'):
        give_recomendationsivdfpf(movieulike)
    if(ch=='2'):
        getrecomcosp(movieulike)
    if(ch=='3'):
        give_recomendationsivdfmf(movieulike)
    if(ch=='4'):
        getrecomcosmf(movieulike)
```

```
c=input("Press 0 to Continue")
```