Movie Recommendation System Using Cosine Similarity, KNN, Matrix Factorization

Submitter

Mithila Arman
23266024
Department of Computer Science And Engineering
Brac University
mithila.arman@g.bracu.ac.bd

Submitted To

Annajiat Alim Rasel
Department of Computer Science And Engineering
Brac University
annajiat@bracu.ac.bd

Dhaka, Bangladesh
December 2023

Contents

ABSTRACT i

List of Figures ..... **iii**

List of Tables ..... Error! Bookmark not defined.

1 Introduction **1**

2 Literature Reviews **1**

3 Data Collection & Processing **2**

4 Methodology **3**

4.1 Feature Extraction 3

4.1.1 Converting Text Data into Feature Vectors 4

4.2 Cosign Similarity 5

4.3 KNN 7

4.3.1 Importing the Dataset: 7

4.3.2 Creating a sparse matrix for movies and users(Data Preprocessing part): 7

4.3.3 Build and Train the model: 8

4.4 Matrix Factorization 10

4.4.1 Creating a sparse matrix: 11

5 Experiments and Results **14**

5.1 Cosine Similarity results: 14

5.2 KNN results: 14

5.3 Matrix Factorization results: 15

6 Future Work and Conclusion **16**

List of Figures

2.1 Literature review 2

3.1 Checking for null values 3

4.1 Combined features(a) 4
4.2 Combined features(b) 5
4.3 Tfidf 5
4.4 Cosine Similarity 6

Chapter 1

Introduction

People prefer entertainment for refreshment to get rid of the work stress of daily life, which helps to recover mentally and physically. Nowadays, there are many ways to have this entertainment. Watching movies & TV series at home or anywhere is among these. There are several categories of movies & TV series. The wish list varies people to people. But lack of available time makes it hard to find a certain movie category or TV series in a short period of time in our life. So, for this I focused on making an attempt at a system that resolves these issues.

In our project, I applied 2 types of filtering for movie recommendation:

- Content based filter.
  - Cosine Similarity
- Collaborative Filtering
  - Knn
  - Matrix Factorization

Chapter 2

Literature Reviews

Here is some information about the literature papers related to our project:

Previous Works	Authors	Methodology
Movie Recommendation System using Cosine Similarity and KNN	Ramni Harbir Singh, Sargam Maurya, Tanisha Tripathi, Tushar Narula, Gaurav Srivastav	Cosine Similarity and KNN
Movie Recommendation Systems using Cosine Similarity	Bamshad Mobasher, Sarabjot Anand	Cosine Similarity
Matrix Factorization for Movie Recommendation	Lili Zhao, Zhongqi Lu, Sinno Jialin Pan, Qiang Yang	Matrix Factorization

Figure 2.1: Literature review

Chapter 3

Data Collection & Processing

I collected the dataset from kaggle. in the dataset 4803 number of rows & 25 columns. The rows are the index of every movie name and userids. The columns are. index, budget, genres, homepage, id, userid, keywords, originallanguage, originaltitle, overview, popular- ity, productioncompanies, productioncountries, releasedate, revenue, runtime, spokenlan- guages, status, tagline, title, voteaverage, votecount, cast, crew, director.

For preprocessing I applied forwardfill for the null values.

```
movies_data_drop.isnull().sum()

index           0
genres          0
user_id         0
keywords        0
popularity      0
tagline         0
title           0
vote_average    0
cast            0
director        0
dtype: int64
```

Figure 3.1: Checking for null values.

For collaborative filtering I used sparse matrix of the required columns of the dataset to fit the models.

Chapter 4

Methodology

4.1 Feature Extraction

Feature extraction is a process of dimensionality reduction by which an initial set of raw data is reduced to more manageable groups for processing. A characteristic of these large data sets is a large number of variables that require a lot of computing resources to process. Fea- ture extraction is the name for methods that select and /or combine variables into features, effectively reducing the amount of data that must be processed, while still accurately and completely describing the original data set. The process of feature extraction is useful when you need to reduce the number of resources

needed for processing without losing important or relevant information. Feature extraction can also reduce the amount of redundant data for a given analysis. Also, the reduction of the data and the machine's efforts in building variable combinations (features) facilitate the speed of learning and generalization steps in the machine learning process.

```
[ ] features = ['genres','keywords','tagline','cast','director']
    print(features)

    for feature in features:
      movies_data_drop[feature] = movies_data_drop[feature].fillna('')

    combined_features = movies_data_drop['genres']+' '+movies_data_drop['keywords']+' '+movies_data_drop['tagline']+' '+movies_data_drop['cast']+' '+movies_data_drop['director']
    print(combined_features)
```

Figure 4.1: Combined features(a)

4.2. COSIGN SIMILARITY

```
['genres', 'keywords', 'tagline', 'cast', 'director']
0       Action Adventure Fantasy Science Fiction cultu...
1       Adventure Fantasy Action ocean drug abuse exot...
2       Action Adventure Crime spy based on novel secr...
3       Action Crime Drama Thriller dc comics crime fi...
4       Action Adventure Science Fiction based on nove...
                              ...
4798    Action Crime Thriller united states\u2013mexic...
4799    Comedy Romance united states\u2013mexico barri...
4800    Comedy Drama Romance TV Movie date love at fir...
4801    Comedy Drama Romance TV Movie date love at fir...
4802    Documentary obsession camcorder crush dream gi...
Length: 4803, dtype: object
```

Figure 4.2: Combined features(b)

4.1.1 Converting Text Data into Feature Vectors

turning text into vectors that can be then fed to machine learning models in a classical way.

```
[ ] vectorizer = TfidfVectorizer()

    feature_vectors = vectorizer.fit_transform(combined_features)

    print(feature_vectors)
```

Figure 4.3: Tfidf

4.2 Cosign Similarity

Cosine similarity is a metric used to measure how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors pro- jected in a multi-dimensional space. The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance, chances are they may still be oriented closer together. The smaller the angle, the higher the cosine similarity. A commonly used approach to match similar documents is based on counting the maximum number of common words between the documents. But this approach has an inherent flaw. That is, as the size of the document increases, the number of common words tends to increase even if the documents talk about different topics. The cosine similarity helps overcome this fundamental flaw in the 'count-the-common-words'. Cosine similarity is a metric used to determine how similar the documents are irrespective of their size. Mathematically, it mea- sures the cosine of the angle between two vectors projected in a multi-dimensional space. In this context, the two vectors I am talking about are arrays containing the word counts of two documents. When plotted on a multi-dimensional space, where each dimension corresponds to a word in the document, the cosine similarity captures the orientation (the angle) of the documents and not the magnitude. If you want the magnitude, compute the

Euclidean distance instead. The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance because of the size they could still have a smaller angle between them. The smaller the angle, the higher the similarity.

```
similarity = cosine_similarity(feature_vectors)
print(similarity)
print(similarity.shape)

# Getting the movie name list from the user

movie_name = input(' Enter your favourite movie name : ')

list_of_all_titles = movies_data_drop['title'].tolist()

find_close_match = difflib.get_close_matches(movie_name, list_of_all_titles)

close_match = find_close_match[0]

index_of_the_movie = movies_data_drop[movies_data.title == close_match]['index'].values[0]

similarity_score = list(enumerate(similarity[index_of_the_movie]))

len(similarity_score)
sorted_similar_movies = sorted(similarity_score, key = lambda x:x[1], reverse = True)

print('Movies suggested for you : \n')

i = 1

for movie in sorted_similar_movies:
  index = movie[0]
  title_from_index = movies_data_drop[movies_data_drop.index==index]['title'].values[0]
  if (i<=30):
    print(i, '.',title_from_index)
    i+=1
```

Figure 4.4: Cosine Similarity

4.3 KNN

4.3.1 Importing the Dataset:

```
[ ] movies = pd.read_csv('/content/drive/MyDrive/Pattern_Dataset/movies.csv')

    movies.shape

    (4803, 25)

                                                    + Code   + Text

[ ] df_movies = movies[['id','original_title']]
    df_movies.head()
    df_ratings = movies[['user_id','id','vote_average']]
    df_ratings.head()
```

In this section, I have imported the dataset into movies, where I can see there are 4803 rows and 25 columns. I don't need all of the columns. That's why I have selected id (basically movie id), original title into dfmovies. In dfratings, I have selected userid, id and voteaverage.

4.3.2 Creating a sparse matrix for movies and users(Data Preprocessing part):

user_id	1	2	3	4	5	6	7	8	9	10	...	1490	1491	1492	1493	1494	1495	1496	1497	1499	1500
id																					
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14	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	7.9	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
454	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
455	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
462	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
464	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
468	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

200 rows × 1430 columns

Here I have created a sparse matrix which is basically a pivot table on dfratings. In the index I have selected id and columns as userid. Meaning, the movie ids will be row wise and userid will be on column wise. Now, I will fill the pivot table with voteaverage values. There will be many nan values in the table that's why I replace those values by 0 by calling the fillna function. Then, by calling the

csrmatrix function, it will automatically create a sparse matrix for the pivot table which looks like this.

4.3.3 Build and Train the model:

Now, I perform the KNN model by calling the NearestNeighbors function and I fit the model with our sparse matrix. I take a function called recommender which takes the

```
model_knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20)
model_knn.fit(mat_movies_users)
```

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

```
## Function for recommendation system (List of the movies recommended)

def recommender(movie_name, data, model, n_recommendations):
  model.fit(data)
  idx = process.extractOne(movie_name, df_movies['original_title'])[2]
  print('Movie Selected: ',df_movies['original_title'][idx], 'Index: ',idx)
  print('Searching for recommendations.....')
  distances, indices = model.kneighbors(data[idx], n_neighbors = n_recommendations)
  for i in indices:
    print(df_movies['original_title'][i].where(i!=idx))

movie_name = input('Enter your favourite movie name: ')
recommender(movie_name, mat_movies_users, model_knn, 20)
```

parameters movie name, data as the sparse matrix, the nearest neighbor model and the no of recommendations. First of all, I fit the model with our sparse matrix here. Then I get the index number of the movie from our movies dataframe. Then find the nearest distance of the given movie with all other movies from the data (sparse matrix) by using the ratings characteristics as the feature. The indices store the index numbers of the recommended movies. From there I print the recommended movies by matching the indices with the index of those movies.

```
movie_name = input('Enter your favourite movie name: ')
recommender(movie_name, mat_movies_users, model_knn, 20)
```

```
Enter your favourite movie name: Avatar
Movie Selected:  Avatar Index:  0
Searching for recommendations.....
638                              You've Got Mail
1897                              Half Past Dead
4625                             Midnight Cabaret
1298                                      Chi bi
0                                            NaN
2554                             The Wedding Date
3208           Star Wars: Clone Wars (Volume 1)
3201                    Veronika Decides to Die
3195                          Princess Kaiulani
3197                         Flammen & Citronen
3196                                 Opal Dream
3194                              All or Nothing
3204                                The I Inside
3199    Red Riding: In the Year of Our Lord 1974
3205                             Beneath Hill 60
3206                                     Polisse
3200                             La Fille du RER
3202                            Crocodile Dundee
3203      Ultramarines: A Warhammer 40,000 Movie
```

Here I input a movie name, and the recommender function will return us the recommended movies based on their nearest ratings.

4.4 Matrix Factorization

Creating two dataframes for moviesdf & user or ratingsdf. After that merging these dataframes in one table (combinemovieratings):

```
movies_df= movies_data[['id','original_title']]
movies_df.head()
ratings_df = movies_data[['user_id','id','vote_average']]
ratings_df.head()
```

```
combine_movie_rating = pd.merge(ratings_df, movies_df, on='id')
combine_movie_rating.head(10)
```

	user_id	id	vote_average	original_title
0	79	19995	7.2	Avatar
1	31	285	6.9	Pirates of the Caribbean: At World's End
2	164	206647	6.3	Spectre
3	348	49026	7.6	The Dark Knight Rises
4	360	49529	6.1	John Carter
5	81	559	5.9	Spider-Man 3
6	218	38757	7.4	Tangled
7	232	99861	7.3	Avengers: Age of Ultron

Now, get the total rating count for every index of the movies:

```
combine_movie_rating = combine_movie_rating.dropna(axis = 0, subset = ['original_title'])

movie_ratingCount = (combine_movie_rating.
     groupby(by = ['original_title'])['vote_average'].
     count().
     reset_index().
     rename(columns = {'vote_average': 'totalRatingCount'})
     [['original_title', 'totalRatingCount']]
    )
movie_ratingCount.head(10)
```

	original_title	totalRatingCount
0	#Horror	1
1	$upercapitalist	1
2	(500) Days of Summer	1
3	...E tu vivrai nel terrore! L'aldilà	1

Rating list (ratingwithtotalRatngCount) with Total rating count with merging the table-combinemovieratings:

```
rating_with_totalRatingCount = combine_movie_rating.merge(movie_ratingCount, left_on = 'original_title', right_on = 'original_title', how = 'left')
rating_with_totalRatingCount.head(100)
```

	user_id	id	vote_average	original_title	totalRatingCount
0	79	19995	7.2	Avatar	1
1	31	285	6.9	Pirates of the Caribbean: At World's End	1
2	164	206647	6.3	Spectre	1
3	348	49026	7.6	The Dark Knight Rises	1
4	360	49529	6.1	John Carter	1

4.4.1 Creating a sparse matrix:

Here have created a sparse matrix which is basically a pivot table. In the index I have selected userid and columns as originaltitle. Meaning, the movie titles will be column wise and userid will be on row wise. Now, I will fill the pivot table with voteaverage values, means it can be seen which user has given what rating in every single movie. There will be many nan values in the table that's why I replace those values by 0 by calling the fillna function.

```
from sklearn.decomposition import TruncatedSVD

SVD = TruncatedSVD(n_components=499)
matrix = SVD.fit_transform(X)
matrix.shape
```

```
(4801, 499)
```

Later on, I transpose the movieuserratingpivot for the equation of Matrix Factorization:

```
X = movie_user_rating_pivot.values.T
X.shape
```

```
(4801, 500)
```

Now making the correlation coefficient matrix of the matrix that gone under Singular Value Decomposition (SVD).

```
import warnings
warnings.filterwarnings("ignore",category =RuntimeWarning)
corr = np.corrcoef(matrix)
corr.shape
```

```
(4801, 4801)
```

Now taking the movie name as input from the user, firstly I get the index of the input movie by movietitlelist.index and store the index in inputmovie. Later sending the value of inputmovie in corr[] to get the correlation coefficients with other movies' indexes and store it in corrinputmovie. Finally, the values of corrinputmovie is equal or greater than 0.8 of the input value I print those list of movies' titles.

```
movie_title = movie_user_rating_pivot.columns
movie_title_list = list(movie_title)
try:
  movie_name = input(' Enter movie name : ')
  input_movie = movie_title_list.index(movie_name)
except ValueError:
  print('Oops!This movie do not exist in the dataset.')
  print('Enter movie matching with the dataset.')
  movie_name = input(' Enter movie name : ')
  input_movie = movie_title_list.index(movie_name)
print('Your recommended movie list :')
corr_input_movie  = corr[input_movie]
list(movie_title[(corr_input_movie >= 0.8)])
```

```
...  Enter movie name :
```

Chapter 5

Experiments and Results

5.1 Cosine Similarity results:

The recommendation results based on content-based filtering.

```
 Enter your favourite movie name : The Avengers
Movies suggested for you :

1 . The Avengers
2 . Avengers: Age of Ultron
3 . Captain America: The Winter Soldier
4 . Captain America: Civil War
5 . Iron Man 2
6 . Thor: The Dark World
7 . X-Men
8 . The Incredible Hulk
9 . X-Men: Apocalypse
10 . Ant-Man
11 . Thor
12 . X2
13 . X-Men: The Last Stand
14 . Deadpool
15 . X-Men: Days of Future Past
16 . Captain America: The First Avenger
17 . The Amazing Spider-Man 2
18 . Iron Man
19 . The Image Revolution
20 . This Is Martin Bonner
```

5.2. KNN RESULTS:

5.2 KNN results:

The recommendation results are based on collaborative filtering.

```
Enter your favourite movie name: The Avengers
Movie Selected:  The Avengers Index:  16
Searching for recommendations......
2601                    Barney's Great Adventure
3879                                        Adam
907                                    Mr. Deeds
16                                           NaN
2745                               This Christmas
3207                                       Awake
3201                        Veronika Decides to Die
3205                               Beneath Hill 60
3204                                  The I Inside
3200                               La Fille du RER
3203      Ultramarines: A Warhammer 40,000 Movie
3209                                    Skin Trade
3196                                    Opal Dream
3198                                  Undiscovered
3195                             Princess Kaiulani
3202                              Crocodile Dundee
3197                            Flammen & Citronen
3199    Red Riding: In the Year of Our Lord 1974
3206                                        Polisse
3208            Star Wars: Clone Wars (Volume 1)
Name: original_title, dtype: object
```

5.3. MATRIX FACTORIZATION RESULTS:

5.3 Matrix Factorization results:

The recommendation results based on content based filtering.

```
 Enter movie name : The Avengers
Your recommended movie list :
['16 Blocks',
 'Aloft',
 'City Hall',
 "My Sister's Keeper",
 'Rubber',
 'The Avengers',
 'The Pallbearer',
 'Ultramarines: A Warhammer 40,000 Movie',
 'Vampires']
```

Here the content-based recommendation: Cosine Similarity is applied based on 5 special combined features: 'genres','keywords','tagline','cast','director'. The recommended movies are provided, depends on the similarity score compared to input movie's features.

For KNN, If I take a movie as input, then what KNN does is, it first finds similar users who have liked the same movies based on ratings. Then it recommeds us those movies which the input user hasn't watched yet, but the similar users have watched close to similar ratings of the input user.

Lastly, Matrix Factorization recommend the movie with help of Simple Value Decomposition with coefficient correlation value of input movie from the users 'movie list based on their total ratings.

Chapter 6

Future Work and Conclusion

So far, I have only worked with features that are either user based, which includes user ratings and user's personal preference or content based which are information retrieval sys- tems. Like in the content-based recommender system I are giving the movie name and based on the similarity checking like genre, actor, cast, director features it's giving us the out- put. On the other hand, I are using Collaborative filtering that doesn't need anything other than user's historical preference (like ratings) in a set of movies that they have watched. But I haven't really tried the approach of

combining these 2 systems which could be a hybrid system. Using a hybrid system could be more efficient than the individual approach. So, in future, I are looking forward to using the hybrid system for the recommender system which can be more efficient.