Movie recommendation system

November 3, 2023

Akash Jha (2022csb1065) ,
Vishal Garg (2022csb1143) ,
Nishant Yadav (2022csb1098)

Instructor:
Dr. Anil Shukla

Teaching Assistant:
Sravanthi Chede

Summary: Our project entails the implementation and analysis of Movie recommendation system. It incorporates functionalities to add new movies, receive user ratings, and provide recommendations based on two models: one using user ratings for 10 movies and another based on user-rated genres. Utilizing file handling, it stores movie names, ratings, and genre information, facilitating personalized recommendations. It has a function to add new movies to the system. Also the user who uses this his data is also stored and calculated for all movies. We have tried to keep the whole system's complexity to be O(Number of movies + Number of users).

---

1. Introduction

A movie recommendation system is an application or software designed to suggest movies to users based on their preferences, historical choices, or behavior. It utilizes various algorithms and data analysis techniques to provide personalized movie suggestions, aiming to enhance user experience and engagement in the vast world of film.

1.1. There are primarily two types of movie recommendation systems:

1.1.1 Content-Based Systems:

These systems recommend movies based on the features of the movies themselves, such as genre, cast, director, plot keywords, or even a description of the movie. They analyze the attributes of the movies and suggest similar items based on these characteristics.

1.1.2 Collaborative Filtering Systems:

These systems recommend movies based on user interactions and preferences. They identify patterns among users—suggesting movies that similar users have liked or recommending movies that are frequently liked by users who share your taste.

Our project includes both of these techniques to provide the user with best recommendations based on user's choice.

Users have a choice to get recommendations based on ratings given by him to few movies or to get recommendations based on genre rating given by user.

User can also add a new movie to the system.

Any user who uses this system his ratings for all movies is calculated and stored.

2. Mathematical Models

This section gives some mathematical concepts used in the project.

2.1. Normalization of ratings for comparison:

Ratings are normalised by dividing by their average ratings for comparison.

2.2. Distance Metrics or differences between user ratings and existing movie ratings:

Difference is calculated between the user ratings and the rating data and then sum of all the differences for all users is calculated and the minimum of these resembles most with the user and user is given that rating.

2.3. Weighted Summation for content-based recommendations based on genre preferences.:

Matrix multiplication of movies vs genre matrix and genre vs user matrix. the maximum value gives the most liked movie by the user.

2.4. Statistical Comparisons to determine similar users or movies:

Comparing a new user(new movie) data with existing users(movies) to get similar existing user(movie).

3. Complexities

function name	Time complexity	Space complexity
addMovie	O(number of movies+number of user)	O(number of movies+number of user)
ratingModel	O(number of movies+number of user)	O(number of movies+number of user)
genreModel	O(number of movies+number of user)	O(number of movies)
addUserInRatingModel	O(number of movies)	O(number of movies)
addUserInGenreModel	O(number of movies)	O(number of movies)

Table 1: Time and space complexity

4. Algorithms

4.1. AddMovie

It is a simple algorithm. We first increment movie i.e. number of movies. We then take name of the movie as input and append it in 'names.txt' which has names of all movies. Then number of movies is updated in 'number.txt'. Then we take imdb rating as input and append it in 'r.txt' which has imdb ratings of all movies. Take genres which are in this movie from user. Compute genre ratings and append in 'movgen.txt' having movies and genre data. Compare among all movie to get the most similar movie genre combination. Give this movie's ratings to the new movie for all users and append these in 'new.txt' the file having user vs movies ratings data.

Algorithm 1 addMovie(number of movies,number of user)

1: Increment 'movie'
2: Open file "names.txt" for appending
3: Print "Enter the movie name: "
4: Read and write the movie name to the file
5: Close the file "names.txt"
6: Open file "number.txt" for writing
7: Write 'movie' and 'user' to the file
8: Close the file "number.txt"
9: Print "Enter IMDb rating:"
10: Read 'r' as float
11: Open file "r.txt" for appending
12: Write 'r' to the file
13: Close the file "r.txt"
14: Define genreNames and genre as arrays
15: Define 'num' as 0
16: **for** i from 0 to 20 **do**
17: Read genre ratings, adjust 'num' accordingly
18: **end for**
19: Open file "movgen.txt" for appending
20: Compute genre ratings, write to the file
21: Close the file "movgen.txt"
22: Compute and find the movie with the minimum genre differences
23: Open file "new.txt" for reading and writing
24: Read ratings of the minimum difference movie
25: Write new movie ratings to the file
26: Close the file "new.txt"
27: Print "Movie added successfully."

4.1.1 Analysis of addMovie :

For line 22 we need 21∗(number of movies) comparisons. For line 24,25 we need O(number of users) steps.
So, overall time complexity is O(number of movies+number of user).
Space complexity is also O(number of movies+number of user) as we need some arrays of space number of movies or number of users.

4.2. RatingModel

First we take 10 movies ratings from user. Then user is added by addUserInRatingModel. Then we normalise the ratings by dividing by imdb ratings of respective movies. Then in matrix we store the 10 movies ratings from all users from 'movuse.txt'. Write names of all movies in movieList[]. Compare ratings for these 10 movies for new user with all 1000 users. Take the most similar user's ratings and give these ratings to te new user. Then top rated movies from this are recommended.

Algorithm 2 ratingModel(number of movies,number of user)

Print: "Please rate these movies on a scale of 0-10 (write rating and press enter)"
$movies \leftarrow 10$
$givenRating[10] \leftarrow \{8.1, 8.4, 8.2, 8.1, 8, 8.1, 8.2, 8.2, 8.2, 7.8\}$
$rate[10] \leftarrow \{\}$
Print: "pk :"
Input: $rate[0]$
Print: "dangal :"
Input: $rate[1]$
... (continue similarly for remaining inputs)
$addUserInRatingModel(movie, user, movies, rate)$
$user \leftarrow 1000$
for i from 0 to 9 **do**
    $rate[i] \leftarrow rate[i]/givenRating[i]$
end for
$matrix[movies][user]$
for i from 0 to movies **do**
    for j from 0 to user **do**
        Read $matrix[i][j]$ from "movuse.txt"
    end for
end for
$movieList[movie]$
Write movie names in movieList from "names.txt"
$compare[movies][user]$
$sum[user]$
for i from 0 to user **do**
    $sum[i] \leftarrow 0$
    for j from 0 to movies **do**
        $compare[j][i] \leftarrow mod(rate[j] - matrix[j][i])$
        $sum[i] \leftarrow sum[i] + compare[j][i]$
    end for
end for
$idx \leftarrow i : minimum(sum[i])$
$highest[movie]$
for i from 0 to movie **do**
    Read $highest[i]$ from "new.txt"
end for
max
$numberOfRecommendations$
$top[numberOfRecommendations]$
Print: "How many movies do you want to be recommended (choose between 1 and movie) :"
Input: $numberOfRecommendations$
for j from 0 to $numberOfRecommendations$ **do**
    $max \leftarrow 0$
    for i from 0 to movie **do**
        $top[j] \leftarrow i : max(highest[i])$
    end for
    $highest[top[j]] \leftarrow 0$
end for
Print: "top numberOfRecommendations recommended movies are :"
Print top[i] one by one in next line.
Print: "NOTE : This list is in decreasing order of recommendations"

4.2.1 Analysis of ratingModel :

Line 18,25-30 takes O(10∗(number of user) steps. Line 32 takes O(number of user). Line 34-36 takes O(number of movie).
So, overall time complexity is O(number of movies+number of user)
matrix takes O(number of user) space and movieList takes O(number of movie) space.
So, overall space complexity is O(number of movies+number of user)

4.3. GenreModel

First we take genre ratiings from the user and store the in genre[]. User is added using addUserInGenreModel. For this user we make content based rating for every movie. We then select the highest of these ratings and recommend those movies.

Algorithm 3 genreModel(number of movies,number of user)

1: $genreNames$[200] ← "Action Adventure Animation Biography Comedy Crime Documentary Drama Family Fantasy History Horror Music Musical Mystery Romance Sci-Fi Short Sport Thriller War "
2: $genre$[21]
3: Print: "Please rate the given genres on a scale of 0-10"
4: $j \leftarrow 0$
5: **for** i from 0 to 20 **do**
6: Print: genre name until " " comes
7: Print: ":"
8: Input: $genre[i]$
9: **end for**
10: $addUserInGenreModel(movie, user, genre)$
11: $contentBased[movie]$
12: $movieList[movie]$
13: Write movie names in movieList from "names.txt"
14: $fptr \leftarrow$ open file "movgen.txt" for reading
15: **for** i from 0 to movie **do**
16: $contentBased[i] \leftarrow 0$
17: $points$
18: **for** j from 0 to 20 **do**
19: Read $points$ from $fptr$
20: $contentBased[i] \leftarrow contentBased[i] + genre[j] \times points$
21: **end for**
22: **end for**
23: $numberOfRecommendations$
24: Print: "How many movies do you want to be recommended (choose between 1 and movie) :"
25: Input: $numberOfRecommendations$
26: $top[numberOfRecommendations]$
27: max
28: **for** j from 0 to $numberOfRecommendations$ **do**
29: $max \leftarrow 0$
30: **for** i from 0 to movie **do**
31: **if** $max < contentBased[i]$ **then**
32: $max \leftarrow contentBased[i]$
33: $top[j] \leftarrow i$
34: **end if**
35: **end for**
36: $contentBased[top[j]] \leftarrow 0$
37: **end for**
38: Print: "Top $numberOfRecommendations$ recommended movies are :"
39: Print top[i] one by one in next line.
40: Print: "NOTE : This list is in decreasing order of recommendations"

4.3.1 Analysis of genreModel :

Line 15-22 takes O(21∗(number of movies)) steps. Line 28-37 takes O(c∗(number of user)) (assuming number of recommendations to be constant).
So, overall time complexity is O(number of movies+number of user)
contentBased, movieList takes O(number of movie) space.
So, overall space complexity is O(number of movies).

4.4. addUserInRatingModel

First we take genre rating of 10 movies and calculate content based rating for all 21 genres and store them in genre[]. Normalise these from 0 to 10. We calculate content based rating for all movies from these genre[] and store them in sum[]. Then we normalise the sum from 0 to 10 and these are new user's ratings which are stored in 'newUser.txt'. Update the number of users.

Algorithm 4 addUserInRatingModel(number of movies,number of user,number of movies user rates,rate array)

```
1: genre[21]
2: for j from 0 to 20 do
3:     genre[j] ← 0
4: end for
5: bin
6: fptr ← open file "genre10.txt" for reading
7: for i from 0 to movies do
8:     for j from 0 to 20 do
9:         Read bin from fptr
10:        if bin == 1 then
11:            genre[j] ← genre[j] + rate[i]
12:        end if
13:    end for
14: end for
15: Close fptr
16: maximum ← max(genre[])
17: for j from 0 to 20 do
18:    genre[j] ← (genre[j] × 10)/maximum
19: end for
20: sum[movie]
21: temp
22: fptr ← open file "movgen.txt" for reading
23: for i from 0 to movie do
24:    sum[i] ← 0
25:    for j from 0 to 20 do
26:        Read temp from fptr
27:        temp ← genre[j] × temp
28:        sum[i] ← sum[i] + temp
29:    end for
30: end for
31: Close fptr
32: max ← 0
33: for i from 0 to movie do
34:    if max < sum[i] then
35:        max ← sum[i]
36:    end if
37: end for
38: for i from 0 to movie do
39:    sum[i] ← (sum[i] × 10)/max
40: end for
41: for i from 0 to movie do
42:    Write sum[i] to "newUser.txt"
43: end for
44: user ← user + 1
45: fptr ← open file "number.txt" for writing
46: Write movie and user to fptr
```

4.4.1 Analysis of addUserInRatingModel :

Lines 7-14 takes O(21∗10)=O(1) steps. Lines 23-30 takes O(21∗(number of movies)) steps. Lines 33-37,38-40,41-43 takes O(number of movies) steps.

So, overall time complexity is O(number of movies).
sum takes O(number of movie) space.
So, overall space complexity is O(number of movies).

4.5. addUserInGenreModel

We have genre[] as input. We calculate content based rating for all movies from these genre[] and store them in sum[]. Then we normalise the sum from 0 to 10 and these are new user's ratings which are stored in 'newUser.txt'. Update the number of users.

Algorithm 5 addUserInGenreModel(number of movies,number of user,genre array)

1: $sum[movie]$
2: $temp$
3: $fptr \leftarrow$ open file "movgen.txt" for reading
4: **for** i from 0 to movie **do**
5: $sum[i] \leftarrow 0$
6: **for** j from 0 to 20 **do**
7: Read $temp$ from $fptr$
8: $temp \leftarrow genre[j] \times temp$
9: $sum[i] \leftarrow sum[i] + temp$
10: **end for**
11: **end for**
12: Close $fptr$
13: $max \leftarrow 0$
14: **for** i from 0 to movie **do**
15: **if** $max < sum[i]$ **then**
16: $max \leftarrow sum[i]$
17: **end if**
18: **end for**
19: **for** i from 0 to movie **do**
20: $sum[i] \leftarrow (sum[i] \times 10)/max$
21: **end for**
22: $fptr \leftarrow$ open file "newUser.txt" for appending
23: **for** i from 0 to movie **do**
24: Write $sum[i]$ to $fptr$
25: **end for**
26: Write newline to $fptr$
27: Close $fptr$
28: $user \leftarrow user + 1$
29: $fptr \leftarrow$ open file "number.txt" for writing
30: Write $movie$ and $user$ to $fptr$
31: Close $fptr$

4.5.1 Analysis of addUserInGenreModel :

Lines 4-11 takes O(21∗(number of movies)) steps. Lines 14-18,19-21,23-25 takes O(number of movies) steps.
So, overall time complexity is O(number of movies).
sum takes O(number of movie) space.
So, overall space complexity is O(number of movies).

5. Conclusions

We have implemented a movie recommenddation system to provide user with the best recommendations. We have also added functions to add a movie and to add users. We have kept the whole system's complexity to be

O(n).

5.1. Possible future developments of this project would be

1. Adding new user's data to 'new.txt' so that it can also be used for comparisons.
2. Make a combination of ratingModel and genreModel to make more better recommendations.
3. Make a search function where user can search for the movies and rate them.

6. Bibliography and citations

bollywood_meta.csv file is taken from [1].
Some useful information and knowledge is taken from [2].

Acknowledgements

We wish to thank our instructor, Dr Anil Shukla and our Teaching Assistant Sravanthi Chede for their guidance and valuable inputs provided during the project.

References

[1] pncnmnp. 2010-2019, 2019. URL `https://github.com/pncnmnp/TIMDB`.

[2] Wikipedia. Recommender system, 2019. URL `https://en.wikipedia.org/wiki/Recommender_system`.

A. Appendix A

A.1. Other file used

A.1.1 bollywood_meta.csv

Contains all the movies data, their imdb id ,imdb ratings, genre and all information about movies.

A.1.2 new.txt

Contains normalised ratings of all movies and users. In this rows are movies and columns are users.

A.1.3 genre10.txt

Contains genre data of the 10 movies which are used in ratingModel.

A.1.4 movgen.txt

Contains normalised genre data of all the movies.

A.1.5 movuse.txt

Contains normalised user ratings data for the 10 movies used in ratingModel.

A.1.6 names.txt

Contains names of all the movies.

A.1.7 newUser.txt

To store the ratings of new users for all movies.

A.1.8 number.txt

To store the number of movies and number of users.

A.1.9 r.txt

Contains the imdb ratings for all the movies.

A.2. Note

We have assumed that the time complexity for reading n bits from a file or to write n bits to a file is O(n).