Project 4: Movie Recommender System

Fall 2023

Contents

MovieLens Dataset

The dataset comprises approximately 1 million anonymous ratings for 3,706 movies, provided by 6,040 MovieLens users who joined the platform in 2000.

You can find some insights from our exploratory data analysis: [Rcode_W13_Movie_EDA.html] [Python_W13_Movie_RS.html]

You can download a copy of the 6040-by-3706 rating matrix in CSV format, complete with headers and row names, from Coursera/Canvas.

Submission Requirements

To complete this assignment, please provide the following:

1. An R Markdown or Python Jupyter Notebook saved in **HTML** format, or a link to such a file. This file should contain all the necessary code to replicate the reported results. There is no page limit for this part.

2. A web link to your movie recommendation application built by your team. You may share the source code link or submit the code as a zip file on Coursera/Canvas.

It's important to note that you cannot utilize any recommender packages from R or Python. However, you are free to use other packages as needed.

HTML File (4 points)

The HTML file should contain two key components:

System I: Recommendation Based on Genres

Imagine you know the user's favorite movie genre. How would you recommend movies to them?

Propose a recommendation scheme along with all the technical details necessary to implement it. For example, you can recommend the top five most popular movies in that genre, but you need to define what you mean by "most popular." Similarly, you can recommend the top five highly-rated movies in that genre, but you must

specify how you define "highly-rated." (Will the movie that receives only one 5-point review be considered highly rated?)

System II: Recommendation Based on IBCF

For this system, follow these steps. Let R denote the 6040-by-3706 rating matrix.

1. Normalize the rating matrix by centering each row. This means subtracting row means from each row of the rating matrix R. Row means should be computed based on non-NA entries. For instance, the mean of a vector like (2, 4, NA, NA) should be 3.

2. Compute the Cosine similarity among the 3,706 movies. For movies i and j, let $\mathcal{I}_{ij}$ denote the set of users who rated both movies i and j. We decide to ignore similarities computed based on less than three user ratings. Thus, define the similarity between movie i and movie j as follows, when the cardinality of $\mathcal{I}_{ij}$ is bigger than two,

$$S_{ij} = \frac{1}{2} + \frac{1}{2} \frac{\sum_{l \in \mathcal{I}_{ij}} R_{li} R_{li}}{\sqrt{\sum_{l \in \mathcal{I}_{ij}} R_{li}^2} \sqrt{\sum_{l \in \mathcal{I}_{ij}} R_{lj}^2}}$$

This transformation $(1 + \cos)/2$ ensures that similarity measures are between 0 and 1. NA values may occur when 1) the set $\mathcal{I}_{ij}$ has a cardinality less than or equal to two (i.e., this pair of movies have been rated by only zero, one, or two users) or 2) one of the denominators is zero.

3. Let S denote the 3706-by-3706 similarity matrix computed in previous step. For each row, sort the non-NA similarity measures and keep the top 30, setting the rest to NA. This new similarity matrix, still denoted as S, is no longer symmetric. Save this matrix online.

4. Create a function named `myIBCF`:

- **Input**: `newuser`, a 6040-by-1 vector (denoted as w) containing ratings for the 6,040 movies from a new user. Many entries in this vector will be zero. The order of the movies in this vector should match the rating matrix R. (Should we center w? If the primary focus is on ranking the ratings rather than their absolute values, centering may not be necessary.)

- **Inside the function**: Upon receiving this input, your function should download the similarity matrix and use it to compute predictions for movies that have not been rated by this new user yet. Use the following formula to compute the prediction for movie l:

$$\frac{1}{\sum_{i \in S(l)} S_{li}} \sum_{i \in S(l)} S_{li} w_i$$

where $S(l)$ denotes the set of movies in the 30-nearest neighborhood of movie l. Again NA values may occur.

- **Output**: Based on your predictions, recommend the **top 10** movies to this new user, using the column names of the rating matrix R. Explain what your code should do if fewer than 10 predictions are non-NA. Provide a method to suggest additional movies that have not been rated by this user.

Test your function

For your function `myIBCF`, print the top 10 recommendations for the following three users:

- User "u1181" from the rating matrix R
- User "u1351" from the rating matrix R
- A hypothetical user who rates movie "m1613" with 5 and movie "m1755" with 4.

The App (3.5 points)

Build an application for both System I and System II. Here are the requirements:

System I

- Allow users to input their favorite movie genre.
- Provide 10 movie recommendations based on the user's selected genre.
- Save your top movie recommendations for each genre (e.g., in a table) to avoid recomputing them each time.
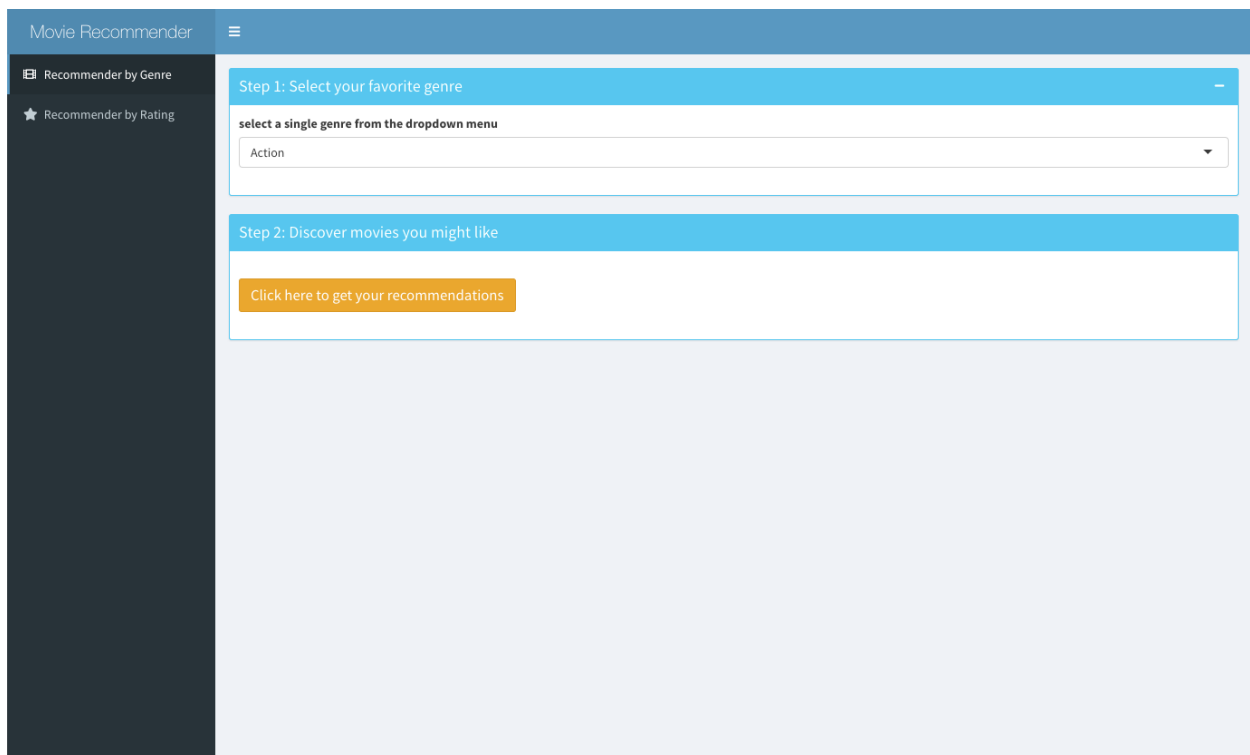
System II

- Present users with a set of sample movies and ask them to rate them.
- Use the ratings provided by the user as input for your `myIBCF` function.
- Display 10 movie recommendations for the user based on their ratings.

We developed an imitation app inspired by a [book recommendation system]. While it appears to gather user data, the 10 recommendations it provides are, in fact, fixed and unchanging, consistently featuring the same initial 10 movies.

https://fengliang.shinyapps.io/MovieRecommend/

Front Page

Additionally, create a front page that enables users to choose between System I and System II for movie recommendations.

Resources

You are welcome to use any existing code, provided you cite the source. For example, you can check how two packages implement IBCF:

- R code for package recommenderlab [https://github.com/mhahsler/recommenderlab/tree/master/R]

- Python code for package Surprise [https://github.com/NicolasHug/Surprise]

- The Github repository for the Book Recommender System mentioned above: [https://github.com/pspachtholz/BookRecommender].

For the App, you can use Shiny if using R. Python users can consider using frameworks like [Shiny], [Dash], or [Flask].