Movie Recommender System

By: Syed Multani, Alvin Lee, Shamit Singh, Andrew Dugal

**Abstract:** The goal of this project is to build a movie recommending application. The success of this project will be the creation of a working application that provides insightful and accurate movie recommendations. This application is meant to be utilized by consumers interested in finding movies that they might like, and movies that are most likely to fit their particular interests.

**Introduction and Discussion:**

Individual taste in movies varies, and is extremely personal. Time is very valuable and not many of us have a lot to spare. Imagine there was a way to maximize your time while being able to cater to your own viewing interests. That is the goal of this project. We are fortunate to live in a time where we have thousands of options at our fingertips when it comes to media. Netflix, Amazon Prime, Crave, Hulu, YouTube, to name a few, provide a nearly unlimited amount of movie options at the click of a button. All of these companies offer their own type of recommendations, but our goal of this project is to create an independent recommender based purely on data with no vested interest in exclusive content.

In this movie recommender model, the dataset is made up of two files movies.csv and ratings.csv and contains 100836 ratings over 9742 movies from 610 unique users. The first 80% are provided as our training dataset, and 20% as our test dataset. We built both item-based and user based collaborative filtering models to inspect the rating matrix. Upon comparing the two models and concluding that the user-based collaborative filtering model is most accurate we used that to build a recommender app in R Shiny.

**Dataset:**

The recommender will be trained using a dataset from: <https://grouplens.org/datasets/movielens/latest/>

These datasets are built from an active research platform and a result of member activity in the MovieLens movie recommendation system. As the datasets are maintained and updated (latest update was in 2018), the data is very high quality and does not require much cleaning. We have used two datasets from the above link.

Dataset 1: ratings.csv is a list of 100836 movie user ratings with three columns- user ID, movie ID, and rating.

Dataset 2: movies.csv is a list of 9742 movies with three columns- movie ID, title, and genre.

**Ethical ML Framework:**

The goal of our report and application is to recommend movies with a high likelihood of enjoyment based on user submitted data from MovieLens datasets. As stated previously, these datasets are built and maintained from an active research platform, therefore results are collected from willing participants and no personal identification data is collected from the user- ie. no names are used and privacy is maintained completely. There are no socio-economic implications for this recommender, and user identity is completely anonymous- we can only assume that there are a wide range of individuals who provided ratings to the dataset but have no way of knowing the demographic breakdown.

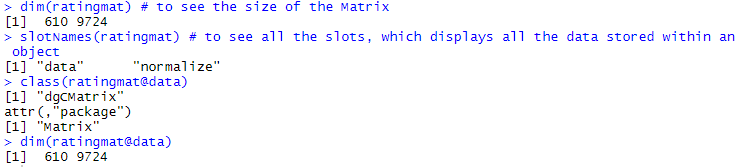
**Assumptions:**

We assume that the datasets are compiled using a wide distribution of opinion, background, and geographical location. As the datasets we have utilized have hundreds of unique users rating thousands of movies, we assume the dataset provides a diverse range of potential opinion representative of greater society and potential user base for our application.

**Data exploration:**

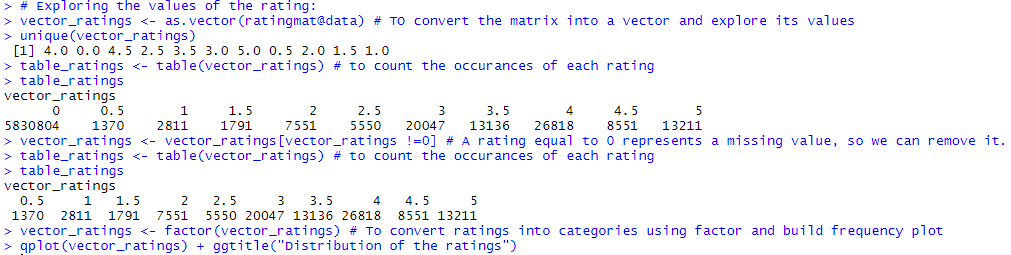
Exploring the nature of the data-

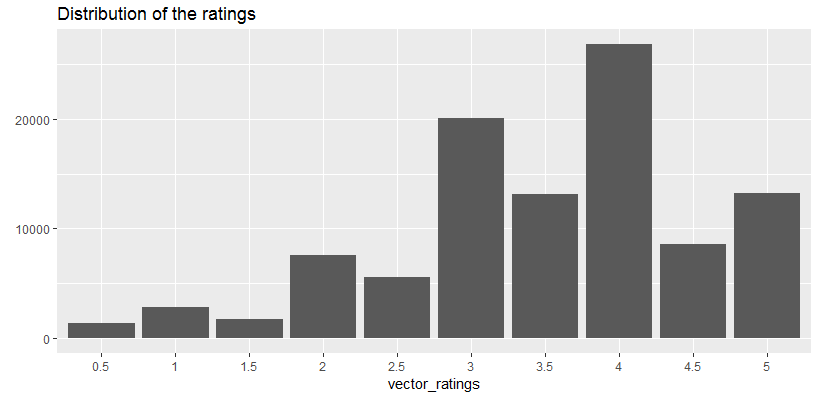
With following code, we can check the size of our realRatingMatrix. The components of the objects are contained in ratingmat slots and we can check all the slots using slotNames to display all the data stored within an object:



Exploring the values of the rating-

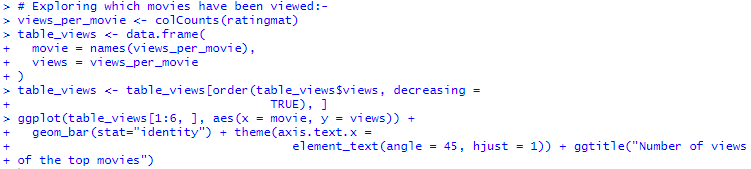
With the help of the following codes we can explore the values of the rating. Rating zero represents a missing value so it has been removed. We have also built frequency plot of the rating and found most of the ratings are above 2 and the most common is 4.

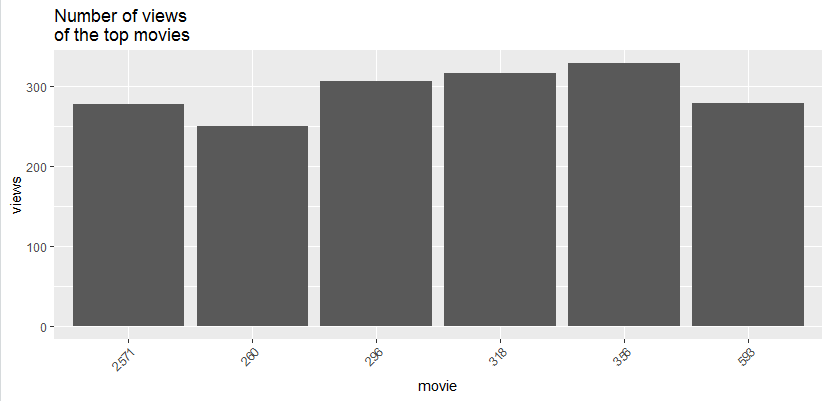




Exploring which movies have been viewed-

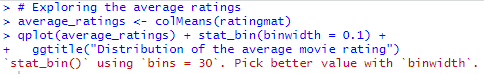
In order to see the most viewed movies following code have been used. From the chart you can notice that movie code 356 (Forrest Gump, 1994) was the most watched movie.

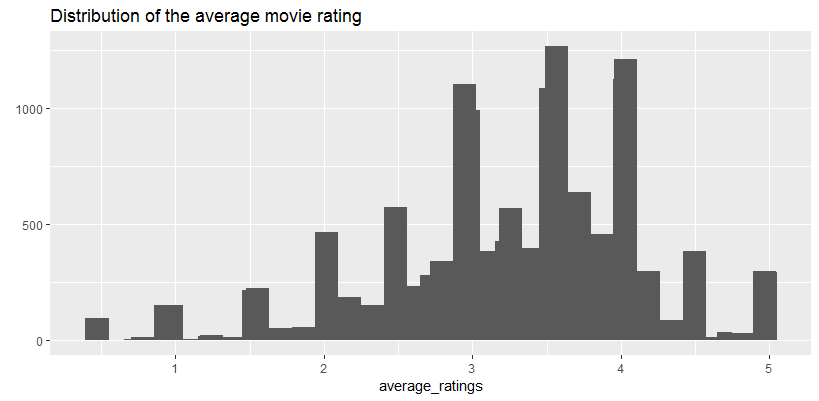




Exploring the average ratings-

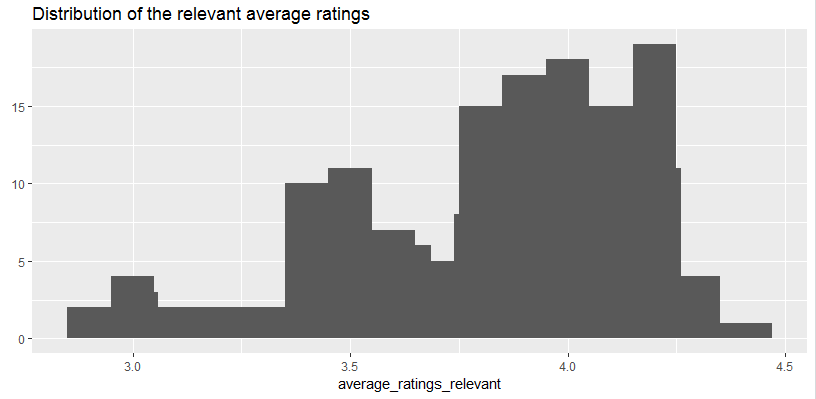
In order to identify the top-rated movies, we have computed the average rating of each of them. Notice for the purpose of ignoring the zero values we have used colMeans. The image below shows the distribution of the average movie rating. The highest value is between 3 and 4, and only few movies are with either 1 or 5 rating.





The possible reason for fewer movies getting 1 and 5 rating may be the small number of people rated them and this can be a good reason to not take them into account. We can remove the movies whose number of the views is below a defined threshold, for example 100. After removing the extreme values all the rankings are between 3 and 4.5.

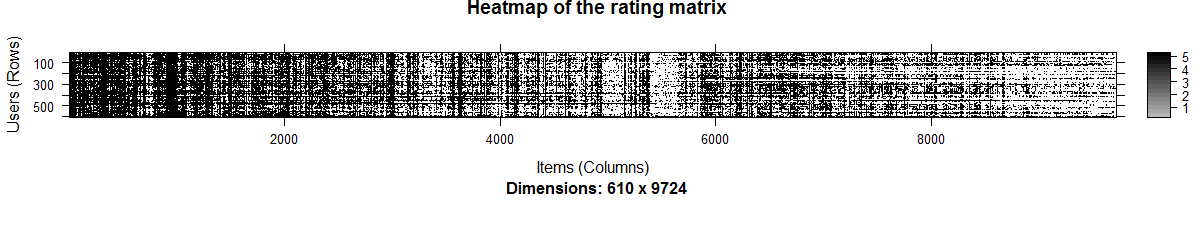




Visualize the matrix-

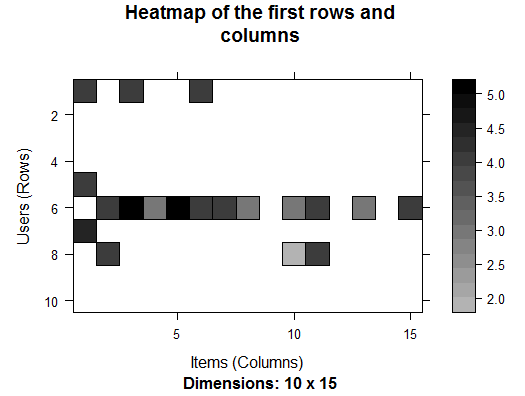
In order to visualize the matrix, we have built Heatmap whose colours represent the ratings. Each row of the matrix corresponds to a user, each column to a movie and each cell to its rating. As it can be noticed a white area in the top-right region.



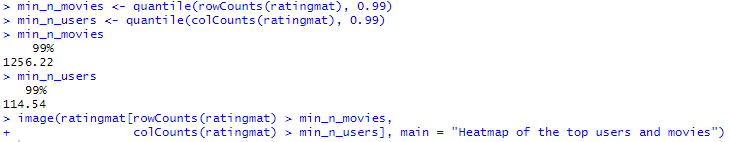


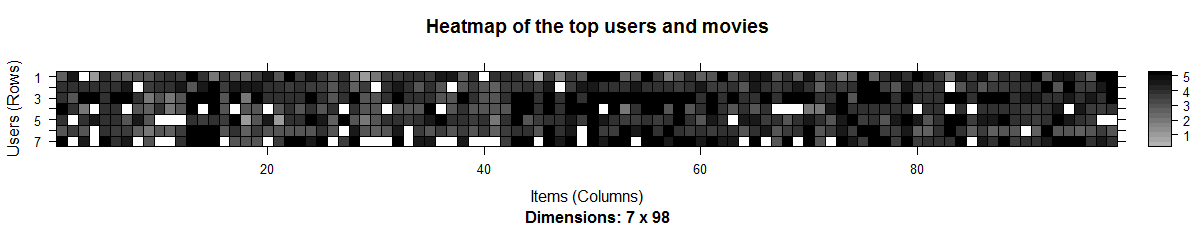
Because there are too many users and items, this chart is difficult to read. We have built another chart zooming in on the first rows and columns. As it can be noticed only some users watched more movies than the others because this chart is based on some random users and item





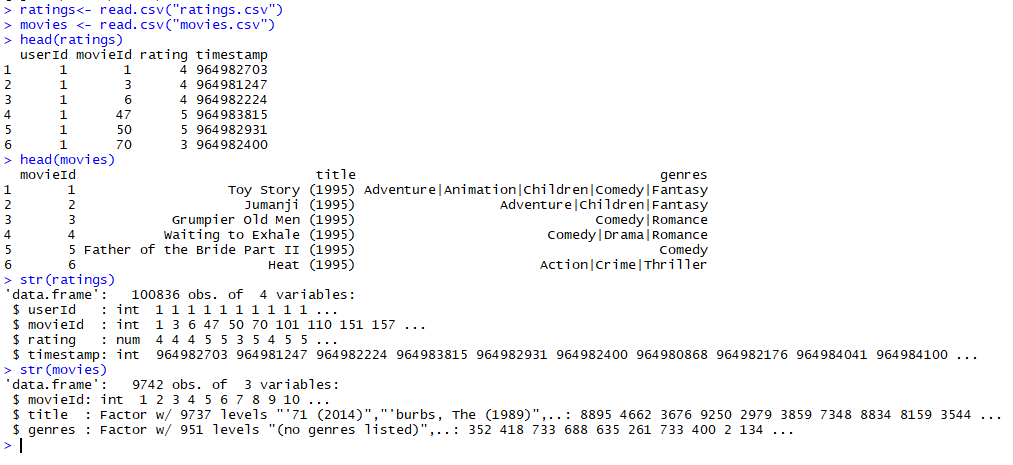
We can select the most relevant users and items or in other words visualize only the users who have seen many movies and the movies that have been seen by many users. For example, we can visualize the top percentile of users and movies. By taking account of the users having watched more movies. It is not surprising most of them have seen all the top movies. We can notice some columns that are darker than the others. These columns represent the highest-rated movies. On the other hand, darker rows represent users giving higher ratings. Because of this, we might need to normalize the data.



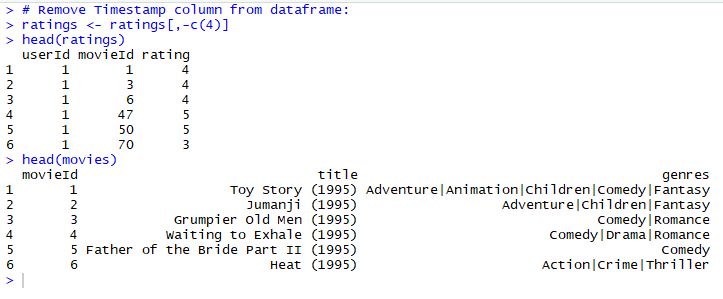


**Data Preparation:**

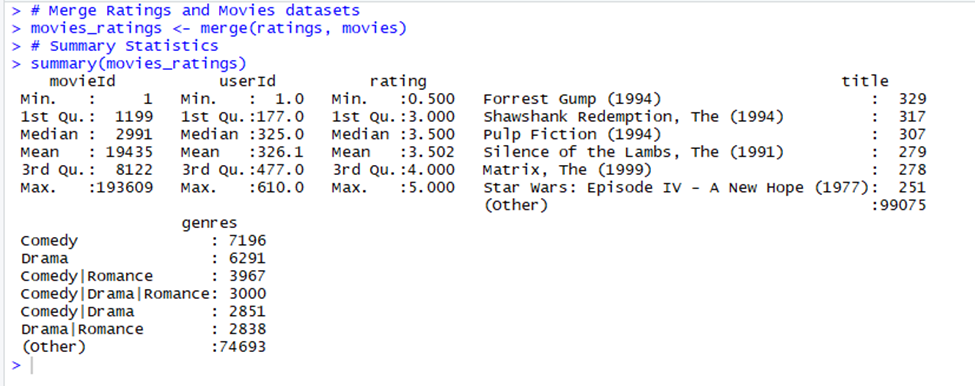
Load data and explore structure-



Timestamp column is not required for our model. Remove timestamp column and confirm changes-

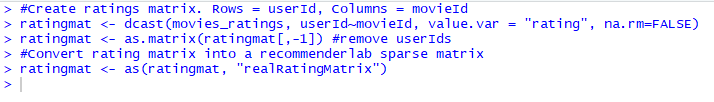


Merge the two datasets and view the summary statistics-

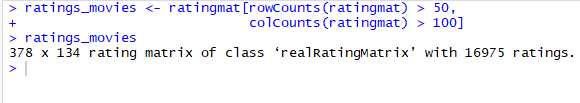


**Build Recommender:**

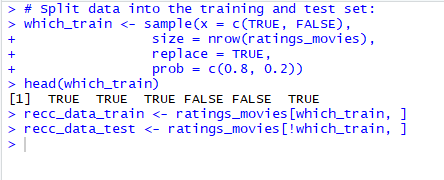
Create ratings matrix and convert ratings matrix into recommender lab sparse matrix-



Filter data-



Split data into training and test data-

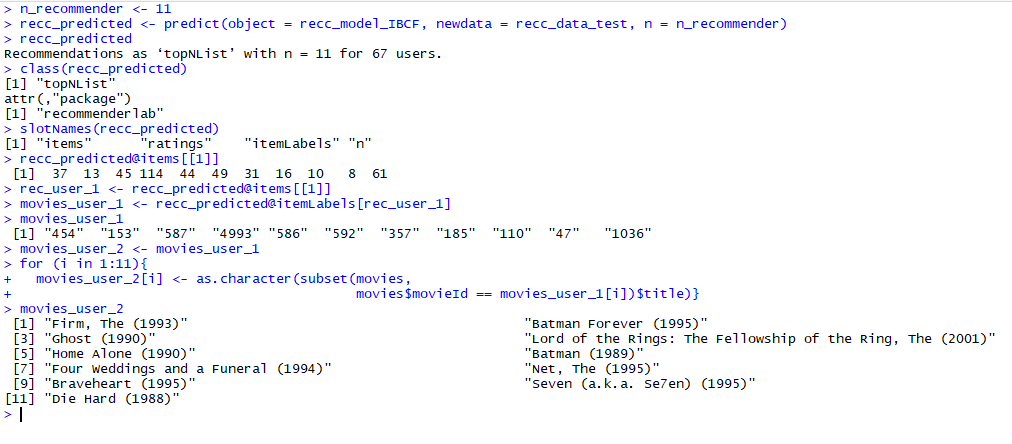


**Item-based Collaborative Filtering model-**

Default parameters for IBCF model, building IBCF model based on parameters-

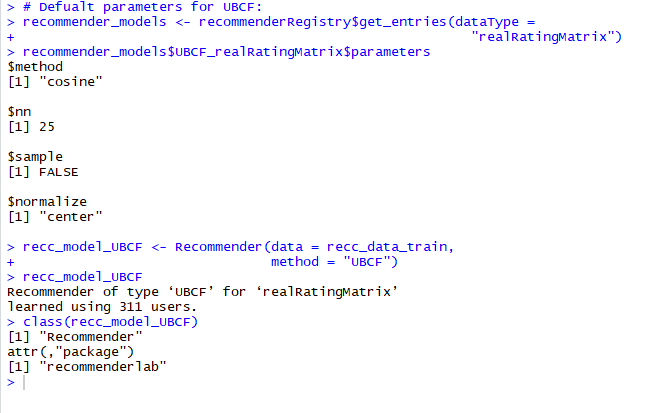


Explore recommender model and apply recommender model on test set-

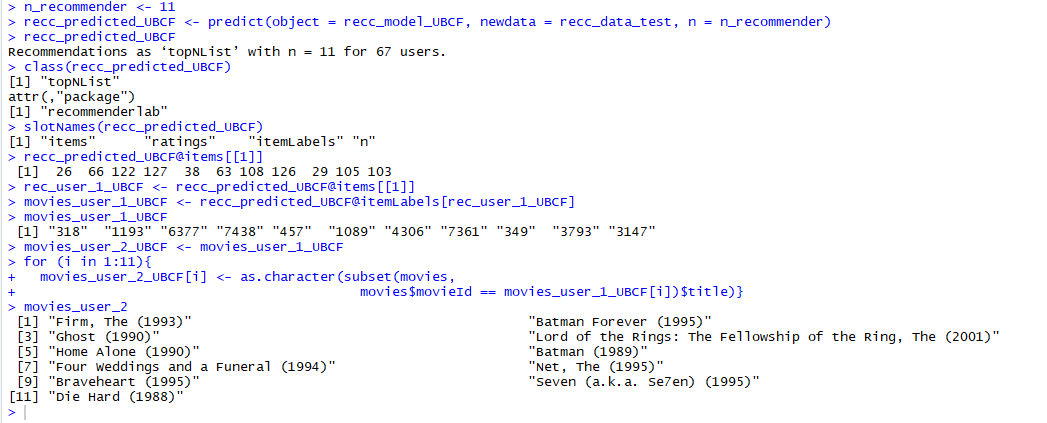


**User-based Collaborative Filtering model-**

Default parameters for UBCF model, building UBCF model based on parameters-

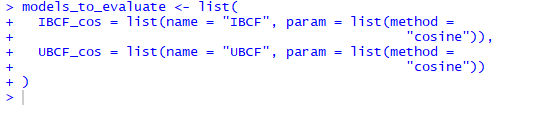


Explore recommender model and apply recommender model on test set-

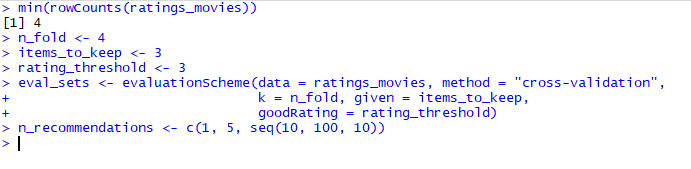


**Identify most suitable model - compare and evaluate:**

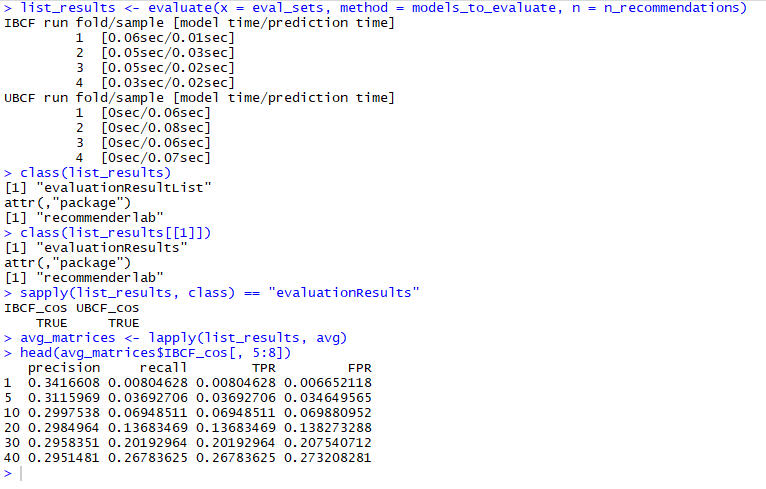
In order to compare and evaluate different models, we can define a list with them. We can build the following evaluation models by filtering IBCF and UBCF using cosine as the distance function-



In order to evaluate the models properly, we need to test them, varying the number of items. We used K-fold to validate models which measures performance more accurately as it tests the recommendation on each user-



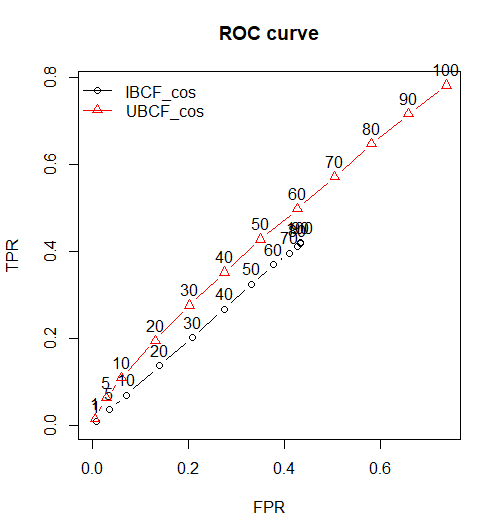
Run and evaluate models (note- now the input method is a list of models)-



Compare models-

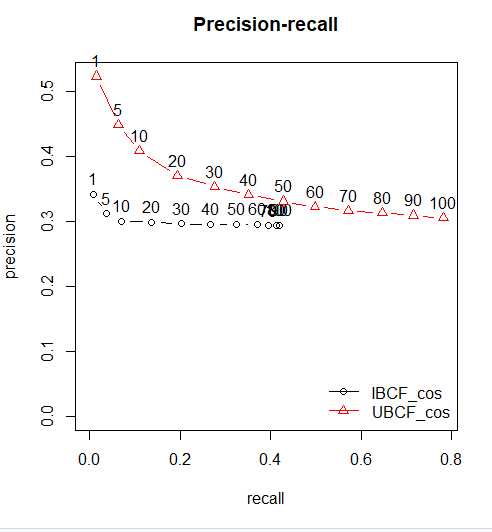
We can compare the models by building a chart displaying their ROC curves. The annotate argument specifies which curves will contain the labels. A good performance index is the area under the curve (AUC), that is, the area under the ROC curve. The highest curve determine the best performing technique.





Precision-recall chart-



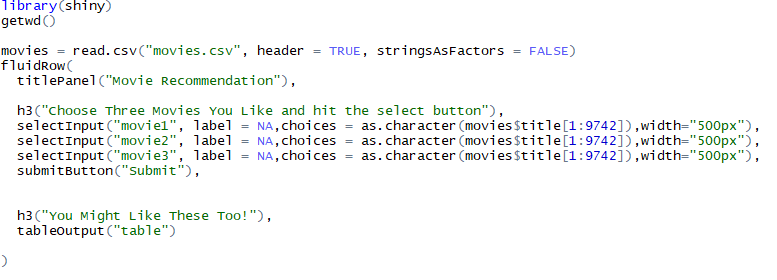


As you can see from the above visualizations, the plots indicate that User-based collaborative filtering offers more accurate results than Item-based collaborative filtering. This is why our Shiny app is based on our UBCF model.

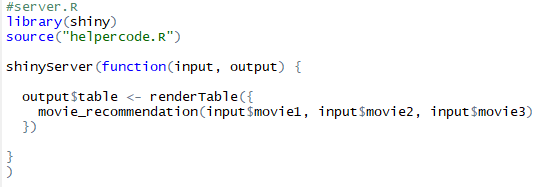
**Prepare Shiny App for Deployment:**

App can be found at <https://alvin20142.shinyapps.io/Lab1/>

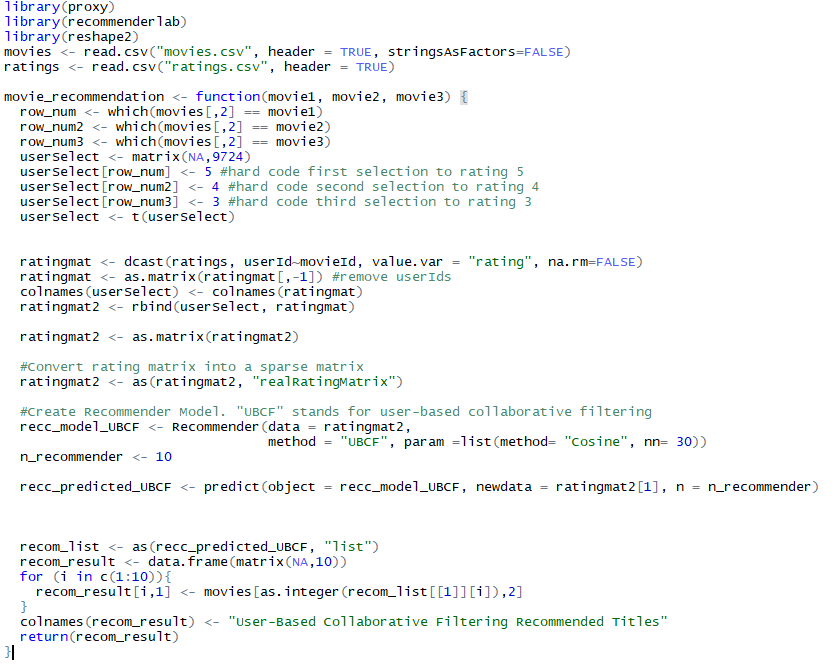
Build and prepare ui.R (user interface)-



Build and prepare server.R file-



Write helpercode.R file for app server-



**App Deployment Discussion:**

Our model provides a basic recommending tool for movies, and allows fans to either use the dropdown menu to choose movie options, or type in their own movies of interest to see what is recommended. Further improvement in accuracy and complexity (more options such as looking for genre or by actor) would be required to make it an ideal recommender, however it does serve as proof of concept.

Some further exploration of the data will be required to refine the recommender as sometimes, one or two out of the 10 recommendations returns a value of “N/A”. This could potentially show us that the model isn’t accurately associating relationships of movies and ratings in order to make a recommendation.

**Resources:**

<https://rpubs.com/jeknov/movieRec>

<https://github.com/jeknov/movieRec>

<https://muffynomster.wordpress.com/2015/06/07/building-a-movie-recommendation-engine-with-r/>

<https://muffynomster.wordpress.com/2015/06/16/building-an-online-recommender-system/>