assignment 4

Nico

8/01/2021

Business Problem/Goal

Building a movie recommender system for users. It is a Item based collaborative filter. It will be based on user preferences and browsing history. The information about the user is taken as an input. The information is taken from the input that is in the form of browsing data. This information reflects the prior usage of the product as well as the assigned ratings. A recommendation system is a platform that provides its users with various contents based on their preferences and likings.

Dataset

In order to build our recommendation system, we have used the MovieLens Dataset. You can find the movies.csv and ratings.csv file that we have used in our Recommendation System Project here. This data consists of 105339 ratings applied over 10329 movies.

```
library(rmarkdown)
library(tidyr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(recommenderlab)
```

```
## Warning: package 'recommenderlab' was built under R version 3.6.3
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loading required package: arules

## Warning: package 'arules' was built under R version 3.6.3

##
## Attaching package: 'arules'

## The following object is masked from 'package:dplyr':
##
##     recode

## The following objects are masked from 'package:base':
##
##     abbreviate, write

## Loading required package: proxy

## Warning: package 'proxy' was built under R version 3.6.3

##
## Attaching package: 'proxy'

## The following object is masked from 'package:Matrix':
##
##     as.matrix

## The following objects are masked from 'package:stats':
##
##     as.dist, dist

## The following object is masked from 'package:base':
##
##     as.matrix

## Loading required package: registry

## Registered S3 methods overwritten by 'registry':
##   method               from
##   print.registry_field proxy
##   print.registry_entry proxy

library(ggplot2)                        #Author DataFlair
library(data.table)

##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last

library(reshape2)

##
## Attaching package: 'reshape2'

## The following objects are masked from 'package:data.table':
##
##     dcast, melt

## The following object is masked from 'package:tidyr':
##
##     smiths
```

Import file into Rstudio

We will now retrieve our data from movies.csv into movie_data dataframe and ratings.csv into rating_data. We will use the str() function to display information about the movie_data dataframe.

```
setwd("C://Users/NicoWong/Documents/sampledatasets/")
#Author DataFlair
movie_data <- read.csv("movies.csv",stringsAsFactors=FALSE)
rating_data <- read.csv("ratings.csv")
str(movie_data)

## 'data.frame':    10329 obs. of  3 variables:
##  $ movieId: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ title  : chr  "Toy Story (1995)" "Jumanji (1995)" "Grumpier Old Men
(1995)" "Waiting to Exhale (1995)" ...
##  $ genres : chr  "Adventure|Animation|Children|Comedy|Fantasy"
"Adventure|Children|Fantasy" "Comedy|Romance" "Comedy|Drama|Romance" ...
```

#Data Exploration Just to get a preview of our data we will use the head and summary command. We don't see any null values in our data so we will proceed to the next step.

```
summary(movie_data)

##     movieId          title              genres
##  Min.   :     1   Length:10329       Length:10329
##  1st Qu.:  3240   Class :character   Class :character
##  Median :  7088   Mode  :character   Mode  :character
##  Mean   : 31924
##  3rd Qu.: 59900
##  Max.   :149532

head(movie_data)
```

```
##   movieId                                title
## 1       1                    Toy Story (1995)
## 2       2                      Jumanji (1995)
## 3       3             Grumpier Old Men (1995)
## 4       4            Waiting to Exhale (1995)
## 5       5 Father of the Bride Part II (1995)
## 6       6                         Heat (1995)
##                                       genres
## 1 Adventure|Animation|Children|Comedy|Fantasy
## 2                  Adventure|Children|Fantasy
## 3                              Comedy|Romance
## 4                        Comedy|Drama|Romance
## 5                                      Comedy
## 6                       Action|Crime|Thriller
```

```
summary(rating_data)
```

```
##      userId         movieId           rating        timestamp
##  Min.   :  1.0   Min.   :     1   Min.   :0.500   Min.   :8.286e+08
##  1st Qu.:192.0   1st Qu.:  1073   1st Qu.:3.000   1st Qu.:9.711e+08
##  Median :383.0   Median :  2497   Median :3.500   Median :1.115e+09
##  Mean   :364.9   Mean   : 13381   Mean   :3.517   Mean   :1.130e+09
##  3rd Qu.:557.0   3rd Qu.:  5991   3rd Qu.:4.000   3rd Qu.:1.275e+09
##  Max.   :668.0   Max.   :149532   Max.   :5.000   Max.   :1.452e+09
```

```
head(rating_data)
```

```
##   userId movieId rating  timestamp
## 1      1      16    4.0 1217897793
## 2      1      24    1.5 1217895807
## 3      1      32    4.0 1217896246
## 4      1      47    4.0 1217896556
## 5      1      50    4.0 1217896523
## 6      1     110    4.0 1217896150
```

#Data preprocessing From the above table, we observe that the userId column, as well as the movieId column, consist of integers. Furthermore, we need to convert the genres present in the movie_data dataframe into a more usable format by the users. In order to do so, we will first create a one-hot encoding to create a matrix that comprises of corresponding genres for each of the films.

```
movie_genre <- as.data.frame(movie_data$genres, stringsAsFactors=FALSE)
library(data.table)
movie_genre2 <- as.data.frame(tstrsplit(movie_genre[,1], '[|]',
                                        type.convert=TRUE),
                              stringsAsFactors=FALSE) #DataFlair
colnames(movie_genre2) <- c(1:10)

list_genre <- c("Action", "Adventure", "Animation", "Children",
                "Comedy", "Crime","Documentary", "Drama", "Fantasy",
                "Film-Noir", "Horror", "Musical", "Mystery","Romance",
```

```
                "Sci-Fi", "Thriller", "War", "Western")
genre_mat1 <- matrix(0,10330,18)
genre_mat1[1,] <- list_genre
colnames(genre_mat1) <- list_genre

for (index in 1:nrow(movie_genre2)) {
  for (col in 1:ncol(movie_genre2)) {
    gen_col = which(genre_mat1[1,] == movie_genre2[index,col]) #Author
DataFlair
    genre_mat1[index+1,gen_col] <- 1
}
}
genre_mat2 <- as.data.frame(genre_mat1[-1,], stringsAsFactors=FALSE) #remove
first row, which was the genre list
for (col in 1:ncol(genre_mat2)) {
  genre_mat2[,col] <- as.integer(genre_mat2[,col]) #convert from characters
to integers
}
str(genre_mat2)
```

```
## 'data.frame':    10329 obs. of  18 variables:
##  $ Action     : int  0 0 0 0 0 1 0 0 1 1 ...
##  $ Adventure  : int  1 1 0 0 0 0 0 1 0 1 ...
##  $ Animation  : int  1 0 0 0 0 0 0 0 0 0 ...
##  $ Children   : int  1 1 0 0 0 0 0 1 0 0 ...
##  $ Comedy     : int  1 0 1 1 1 0 1 0 0 0 ...
##  $ Crime      : int  0 0 0 0 0 1 0 0 0 0 ...
##  $ Documentary: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Drama      : int  0 0 0 1 0 0 0 0 0 0 ...
##  $ Fantasy    : int  1 1 0 0 0 0 0 0 0 0 ...
##  $ Film-Noir  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Horror     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Musical    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Mystery    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Romance    : int  0 0 1 1 0 0 1 0 0 0 ...
##  $ Sci-Fi     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Thriller   : int  0 0 0 0 0 1 0 0 0 1 ...
##  $ War        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Western    : int  0 0 0 0 0 0 0 0 0 0 ...
```

we will create a 'search matrix' that will allow us to perform an easy search of the films by specifying the genre present in our list.

```
SearchMatrix <- cbind(movie_data[,1:2], genre_mat2[])
head(SearchMatrix)    #DataFlair
```

```
##   movieId                               title Action Adventure Animation
## 1       1                    Toy Story (1995)      0         1         1
## 2       2                      Jumanji (1995)      0         1         0
## 3       3             Grumpier Old Men (1995)      0         0         0
## 4       4            Waiting to Exhale (1995)      0         0         0
```

```
## 5       5 Father of the Bride Part II (1995)      0         0         0
## 6       6                        Heat (1995)      1         0         0
##   Children Comedy Crime Documentary Drama Fantasy Film-Noir Horror Musical
## 1        1      1     0           0     0       1         0      0       0
## 2        1      0     0           0     0       1         0      0       0
## 3        0      1     0           0     0       0         0      0       0
## 4        0      1     0           0     1       0         0      0       0
## 5        0      1     0           0     0       0         0      0       0
## 6        0      0     1           0     0       0         0      0       0
##   Mystery Romance Sci-Fi Thriller War Western
## 1       0       0      0        0   0       0
## 2       0       0      0        0   0       0
## 3       0       1      0        0   0       0
## 4       0       1      0        0   0       0
## 5       0       0      0        0   0       0
## 6       0       0      0        1   0       0
```

From the observation above, we can see there are movies that fall under several genres. In order for the algorithm to make sense of the ratings, we have to convert our matrix to a sparse matrix.

```
ratingMatrix <- dcast(rating_data, userId~movieId, value.var = "rating",
na.rm=FALSE)
ratingMatrix <- as.matrix(ratingMatrix[,-1]) #remove userIds
#Convert rating matrix into a recommenderlab sparse matrix
ratingMatrix <- as(ratingMatrix, "realRatingMatrix")
ratingMatrix

## 668 x 10325 rating matrix of class 'realRatingMatrix' with 105339 ratings.
```

The code below will show us the various parameters for building recommendation system for movies.

```
recommendation_model <- recommenderRegistry$get_entries(dataType =
"realRatingMatrix")
names(recommendation_model)

##  [1] "HYBRID_realRatingMatrix"       "ALS_realRatingMatrix"
##  [3] "ALS_implicit_realRatingMatrix" "IBCF_realRatingMatrix"
##  [5] "LIBMF_realRatingMatrix"        "POPULAR_realRatingMatrix"
##  [7] "RANDOM_realRatingMatrix"       "RERECOMMEND_realRatingMatrix"
##  [9] "SVD_realRatingMatrix"          "SVDF_realRatingMatrix"
## [11] "UBCF_realRatingMatrix"

lapply(recommendation_model, "[[", "description")

## $HYBRID_realRatingMatrix
## [1] "Hybrid recommender that aggegates several recommendation strategies
using weighted averages."
##
## $ALS_realRatingMatrix
```

```
## [1] "Recommender for explicit ratings based on latent factors, calculated
by alternating least squares algorithm."
##
## $ALS_implicit_realRatingMatrix
## [1] "Recommender for implicit data based on latent factors, calculated by
alternating least squares algorithm."
##
## $IBCF_realRatingMatrix
## [1] "Recommender based on item-based collaborative filtering."
##
## $LIBMF_realRatingMatrix
## [1] "Matrix factorization with LIBMF via package recosystem
(https://cran.r-
project.org/web/packages/recosystem/vignettes/introduction.html)."
##
## $POPULAR_realRatingMatrix
## [1] "Recommender based on item popularity."
##
## $RANDOM_realRatingMatrix
## [1] "Produce random recommendations (real ratings)."
##
## $RERECOMMEND_realRatingMatrix
## [1] "Re-recommends highly rated items (real ratings)."
##
## $SVD_realRatingMatrix
## [1] "Recommender based on SVD approximation with column-mean imputation."
##
## $SVDF_realRatingMatrix
## [1] "Recommender based on Funk SVD with gradient descend
(https://sifter.org/~simon/journal/20061211.html)."
##
## $UBCF_realRatingMatrix
## [1] "Recommender based on user-based collaborative filtering."
```

We will be moving forward with the Item Based Collaborative Filtering for this model

```
recommendation_model$IBCF_realRatingMatrix$parameters
```

```
## $k
## [1] 30
##
## $method
## [1] "Cosine"
##
## $normalize
## [1] "center"
##
## $normalize_sim_matrix
## [1] FALSE
##
```

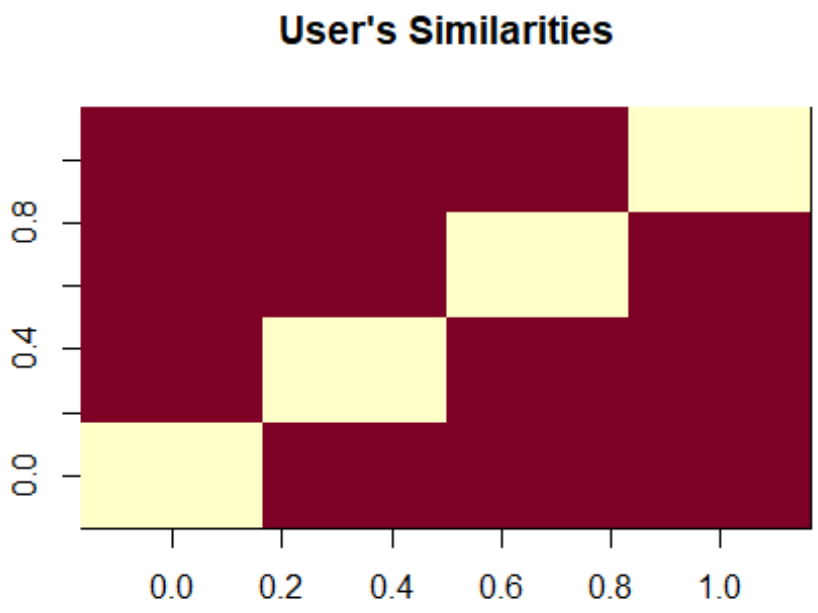
```
## $alpha
## [1] 0.5
##
## $na_as_zero
## [1] FALSE
```

#Exploring Similar Data Collaborative Filtering involves suggesting movies to the users that are based on collecting preferences from many other users. For example, if a user A likes to watch action films and so does user B, then the movies that the user B will watch in the future will be recommended to A and vice-versa. Therefore, recommending movies is dependent on creating a relationship of similarity between the two users. With the help of recommenderlab, we can compute similarities using various operators like cosine, pearson as well as jaccard.

```
similarity_mat <- similarity(ratingMatrix[1:4, ],
                             method = "cosine",
                             which = "users")
as.matrix(similarity_mat)
```

```
##           1         2         3         4
## 1 0.0000000 0.9760860 0.9641723 0.9914398
## 2 0.9760860 0.0000000 0.9925732 0.9374253
## 3 0.9641723 0.9925732 0.0000000 0.9888968
## 4 0.9914398 0.9374253 0.9888968 0.0000000
```

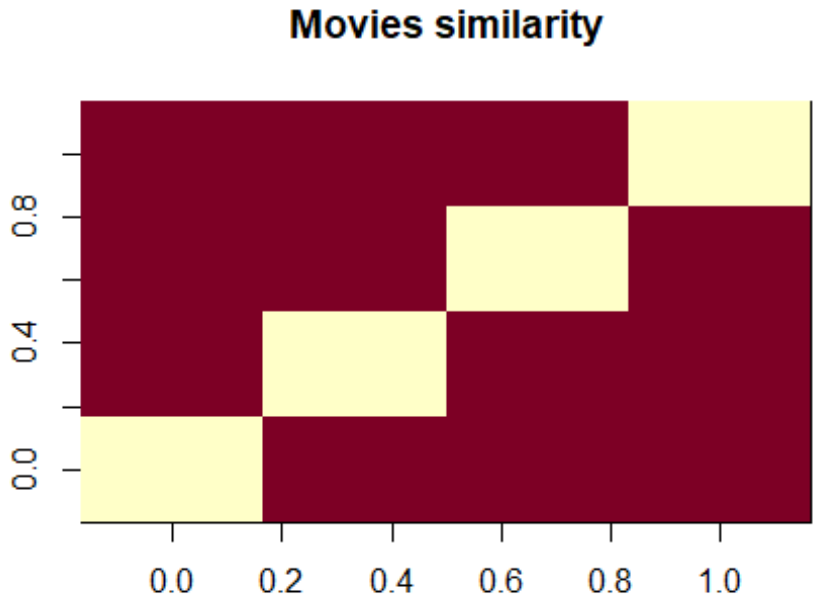
```
image(as.matrix(similarity_mat), main = "User's Similarities")
```



In the above matrix, each row and column represents a user. We have taken four users and each cell in this matrix represents the similarity that is shared between the two users.

Now, we delineate the similarity that is shared between the films –

```
movie_similarity <- similarity(ratingMatrix[, 1:4], method =
                                "cosine", which = "items")
as.matrix(movie_similarity)

##           1         2         3         4
## 1 0.0000000 0.9669732 0.9559341 0.9101276
## 2 0.9669732 0.0000000 0.9658757 0.9412416
## 3 0.9559341 0.9658757 0.0000000 0.9864877
## 4 0.9101276 0.9412416 0.9864877 0.0000000

image(as.matrix(movie_similarity), main = "Movies similarity")
```



For this section, they use only the ratings table to find similarities between movies and also for Users. I can understand this table we can find similarities in users by what ratings they give each movie/by views. But similarities in movies should come from the movies table where we judge them according to genre. Based on their code, they don't give a good explanation of how the similarities are being judged. Are they find similarities is movies by how good the ratings are? If so, I think that is flawed.

We now extract the most unique ratings

```
rating_values <- as.vector(ratingMatrix@data)
unique(rating_values) # extracting unique ratings
```

```
##  [1] 0.0 5.0 4.0 3.0 4.5 1.5 2.0 3.5 1.0 2.5 0.5
```

Now, we will create a table of ratings that will display the most unique ratings.

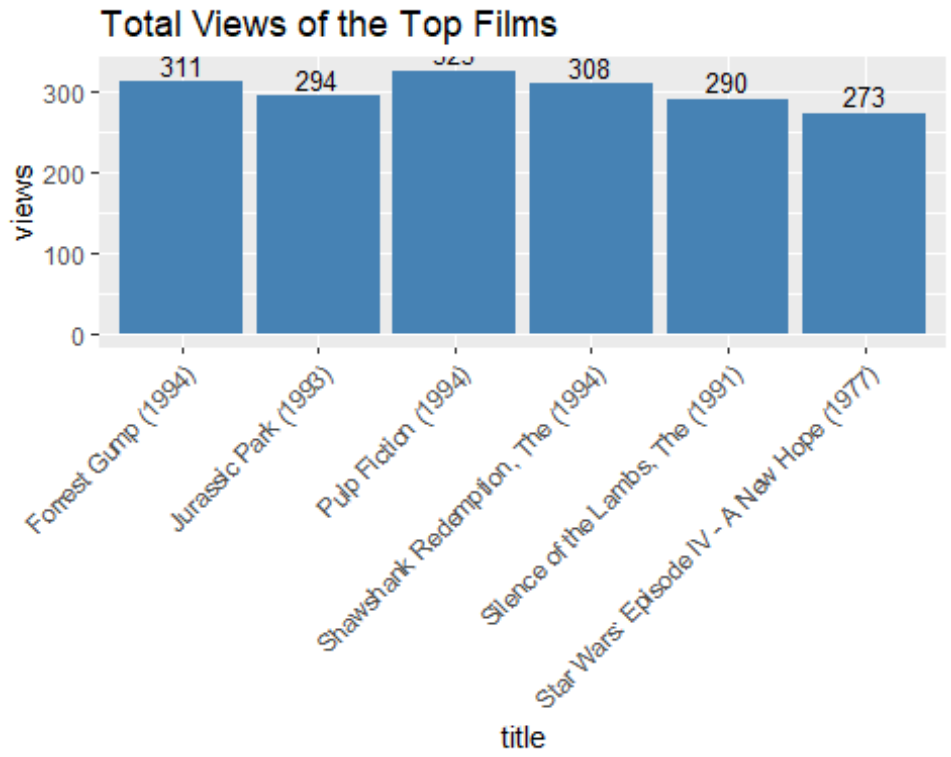
```
Table_of_Ratings <- table(rating_values) # creating a count of movie ratings
Table_of_Ratings

## rating_values
##       0     0.5       1     1.5       2     2.5       3     3.5       4
4.5
## 6791761    1198    3258    1567    7943    5484   21729   12237   28880
8187
##       5
##   14856
```

#Most Viewed Movies Visualization In this section of the machine learning project, we will explore the most viewed movies in our dataset. We will first count the number of views in a film and then organize them in a table that would group them in descending order.

```
library(ggplot2)
movie_views <- colCounts(ratingMatrix) # count views for each movie
table_views <- data.frame(movie = names(movie_views),
                          views = movie_views) # create dataframe of views
table_views <- table_views[order(table_views$views,
                                 decreasing = TRUE), ] # sort by number of
views
table_views$title <- NA
for (index in 1:10325){
  table_views[index,3] <- as.character(subset(movie_data,
                                      movie_data$movieId ==
table_views[index,1])$title)
}
table_views[1:6,]

##     movie views                                        title
## 296   296   325                          Pulp Fiction (1994)
## 356   356   311                          Forrest Gump (1994)
## 318   318   308           Shawshank Redemption, The (1994)
## 480   480   294                        Jurassic Park (1993)
## 593   593   290           Silence of the Lambs, The (1991)
## 260   260   273 Star Wars: Episode IV - A New Hope (1977)
```

Below we visualize a bar plot for the total number of views of the top films.

```
ggplot(table_views[1:6, ], aes(x = title, y = views)) +
  geom_bar(stat="identity", fill = 'steelblue') +
  geom_text(aes(label=views), vjust=-0.3, size=3.5) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +

  ggtitle("Total Views of the Top Films")
```

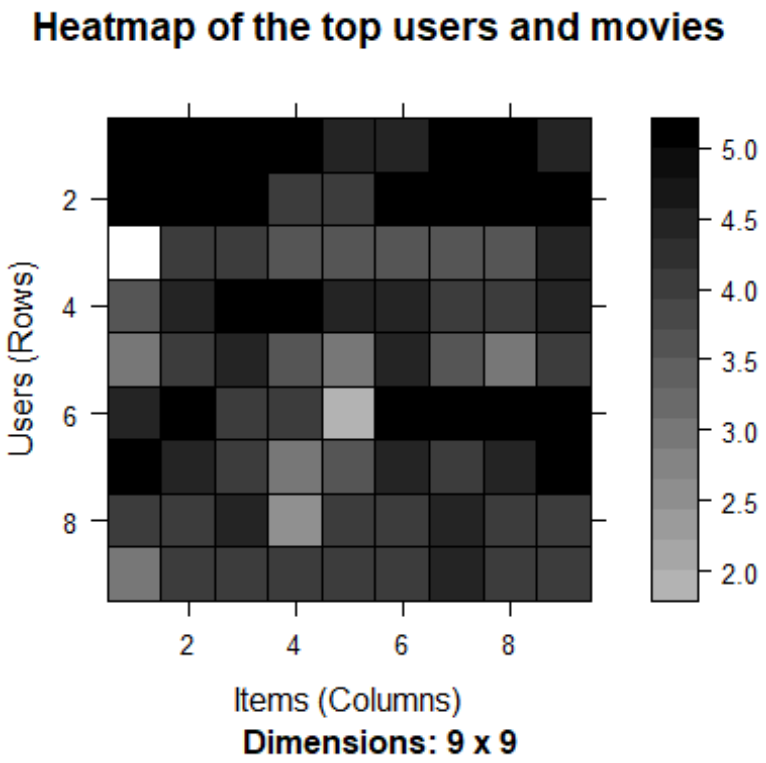


#Data preparation For finding useful data in our dataset, we have set the threshold for the minimum number of users who have rated a film as 50. This is also same for minimum number of views per film. This way, we have filtered a list of watched films from least-watched ones.

```
movie_ratings <- ratingMatrix[rowCounts(ratingMatrix) > 50,
                              colCounts(ratingMatrix) > 50]
movie_ratings

## 420 x 447 rating matrix of class 'realRatingMatrix' with 38341 ratings.
```

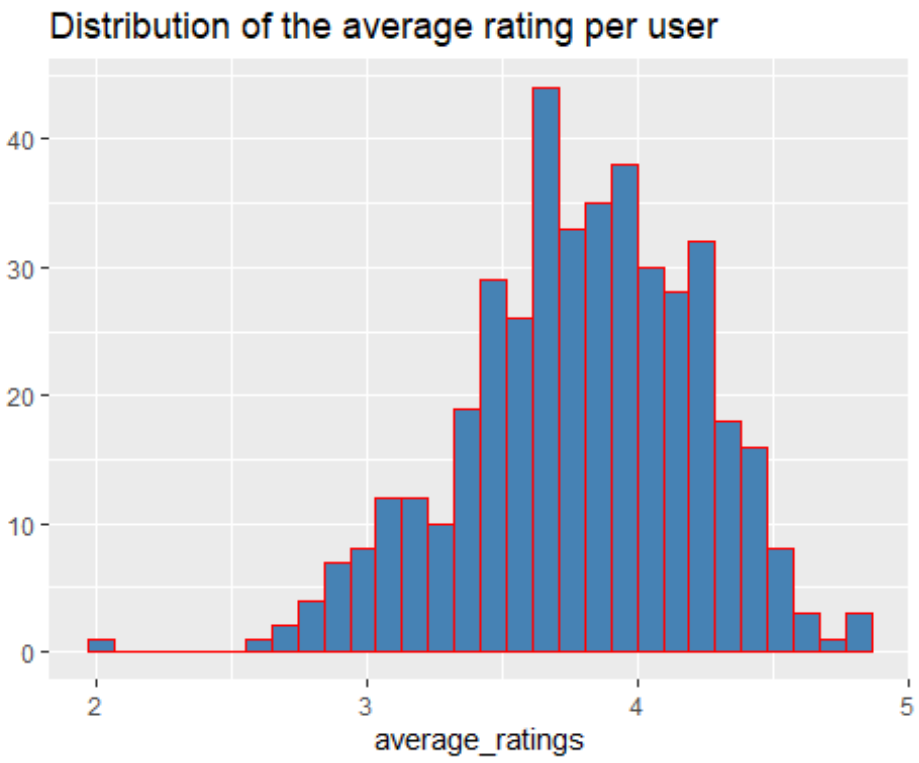
From the above output of 'movie_ratings', we observe that there are 420 users and 447 films as opposed to the previous 668 users and 10325 films. We can now delineate our matrix of relevant users as follows –

```
minimum_movies<- quantile(rowCounts(movie_ratings), 0.98)
minimum_users <- quantile(colCounts(movie_ratings), 0.98)
image(movie_ratings[rowCounts(movie_ratings) > minimum_movies,
                    colCounts(movie_ratings) > minimum_users],
main = "Heatmap of the top users and movies")
```



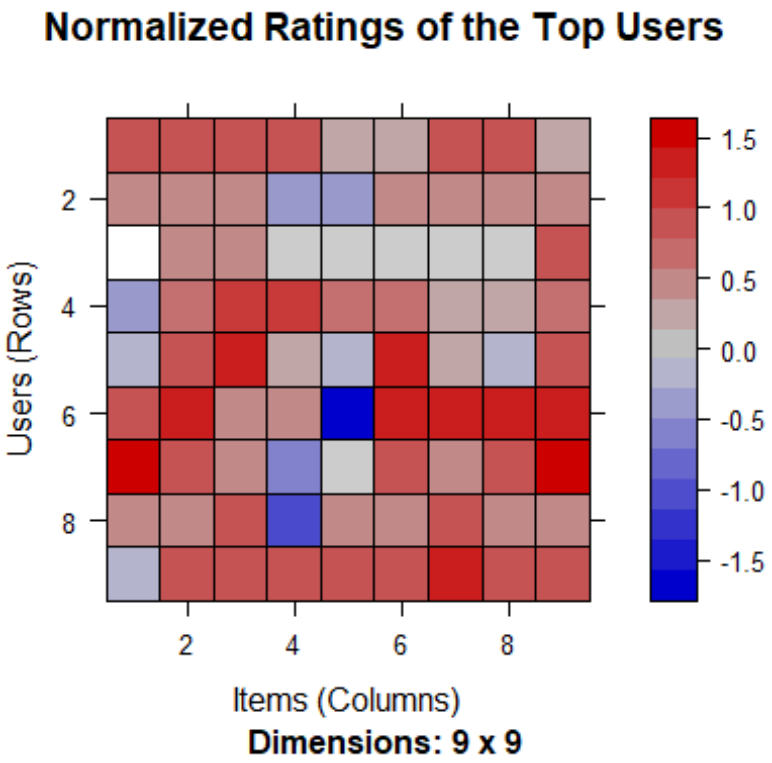
Now, we will visualize the distribution of the average ratings per user.

```
average_ratings <- rowMeans(movie_ratings)
qplot(average_ratings, fill=I("steelblue"), col=I("red")) +
  ggtitle("Distribution of the average rating per user")

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



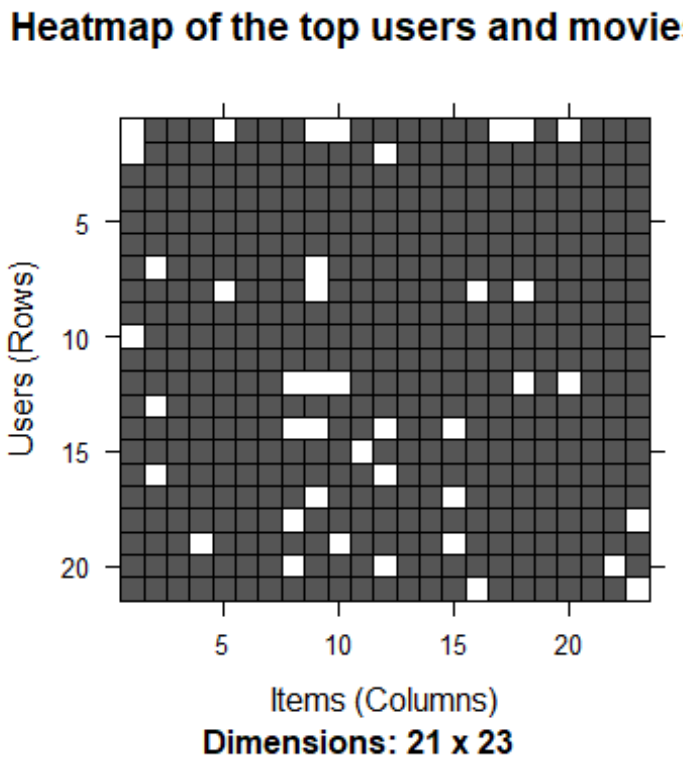
#Data Normalization In the case of some users, there can be high ratings or low ratings provided to all of the watched films. This will act as a bias while implementing our model. In order to remove this, we normalize our data. Normalization is a data preparation procedure to standardize the numerical values in a column to a common scale value. This is done in such a way that there is no distortion in the range of values. Normalization transforms the average value of our ratings column to 0. We then plot a heatmap that delineates our normalized ratings.

```
normalized_ratings <- normalize(movie_ratings)
sum(rowMeans(normalized_ratings) > 0.00001)

## [1] 0

image(normalized_ratings[rowCounts(normalized_ratings) > minimum_movies,
                         colCounts(normalized_ratings) > minimum_users],
main = "Normalized Ratings of the Top Users")
```



#Performing Data Binarization In the final step of our data preparation in this data science project, we will binarize our data. Binarizing the data means that we have two discrete values 1 and 0, which will allow our recommendation systems to work more efficiently. We will define a matrix that will consist of 1 if the rating is above 3 and otherwise it will be 0.

```
binary_minimum_movies <- quantile(rowCounts(movie_ratings), 0.95)
binary_minimum_users <- quantile(colCounts(movie_ratings), 0.95)
#movies_watched <- binarize(movie_ratings, minRating = 1)

good_rated_films <- binarize(movie_ratings, minRating = 3)
image(good_rated_films[rowCounts(movie_ratings) > binary_minimum_movies,
colCounts(movie_ratings) > binary_minimum_users],
main = "Heatmap of the top users and movies")
```



I guess this is where it gets confusing because the normalization and binarization don't seem to connect. It seems like you use either method not both. And what's even more confusing is the part where the training data does not even use either one of them. So I don't fully understand why we did them if we're not using it in our training data.

#Collaborative Filtering System The algorithm first builds a similar-items table of the customers who have purchased them into a combination of similar items. This is then fed into the recommendation system.

The similarity between single products and related products can be determined with the following algorithm –

For each Item i1 present in the product catalog, purchased by customer C. And, for each item i2 also purchased by the customer C. Create record that the customer purchased items i1 and i2. Calculate the similarity between i1 and i2. We will build this filtering system by splitting the dataset into 80% training set and 20% test set.

```
sampled_data<- sample(x = c(TRUE, FALSE),
                      size = nrow(movie_ratings),
                      replace = TRUE,
                      prob = c(0.8, 0.2))
training_data <- movie_ratings[sampled_data, ]
testing_data <- movie_ratings[!sampled_data, ]
```

We will now explore the various parameters of our Item Based Collaborative Filter. These parameters are default in nature. In the first step, k denotes the number of items for

computing their similarities. Here, k is equal to 30. Therefore, the algorithm will now identify the k most similar items and store their number. We use the cosine method which is the default one but you can also use pearson method.

```
recommendation_system <- recommenderRegistry$get_entries(dataType
="realRatingMatrix")
recommendation_system$IBCF_realRatingMatrix$parameters

## $k
## [1] 30
##
## $method
## [1] "Cosine"
##
## $normalize
## [1] "center"
##
## $normalize_sim_matrix
## [1] FALSE
##
## $alpha
## [1] 0.5
##
## $na_as_zero
## [1] FALSE
```

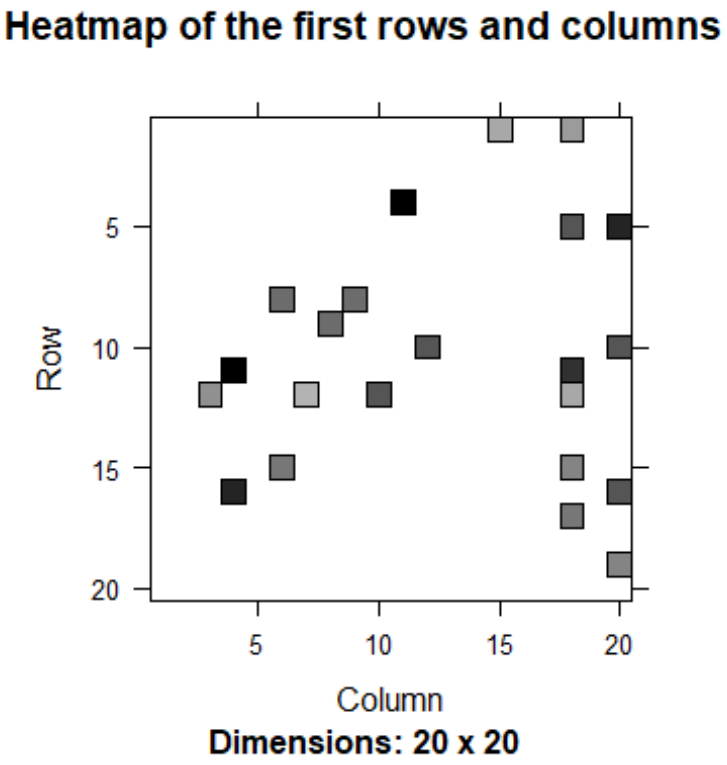
Train our model

```
recommen_model <- Recommender(data = training_data,
                              method = "IBCF",
                              parameter = list(k = 30))
recommen_model

## Recommender of type 'IBCF' for 'realRatingMatrix'
## learned using 338 users.

class(recommen_model)

## [1] "Recommender"
## attr(,"package")
## [1] "recommenderlab"
```

Using the getModel() function, we will retrieve the recommen_model. We will then find the class and dimensions of our similarity matrix that is contained within model_info. Finally, we will generate a heatmap, that will contain the top 20 items and visualize the similarity shared between them.

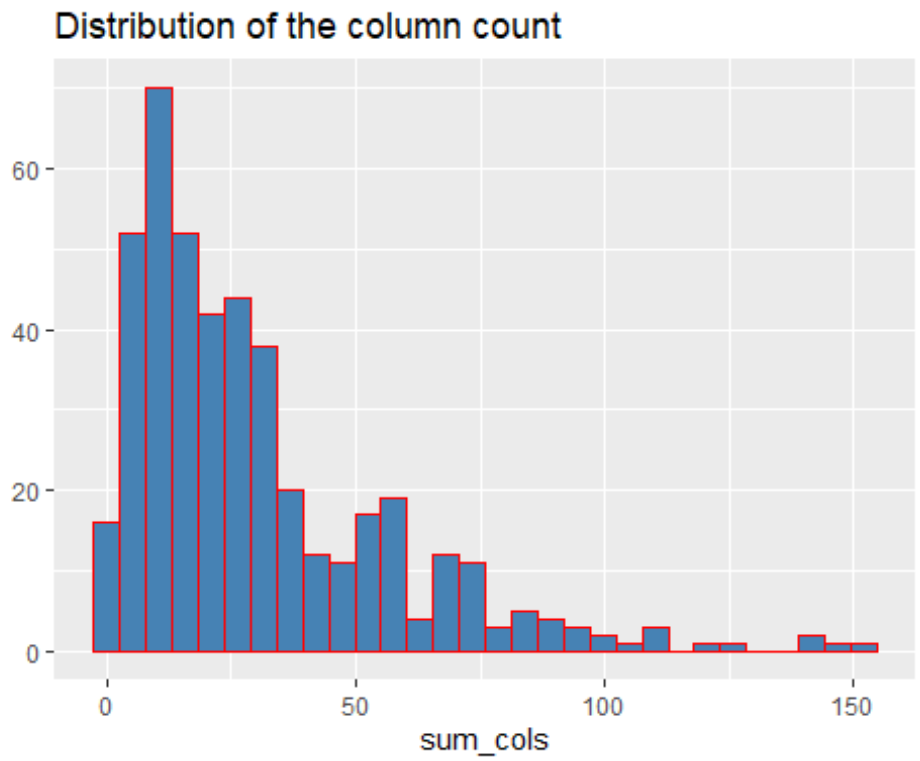
```
model_info <- getModel(recommen_model)
class(model_info$sim)
```

```
## [1] "dgCMatrix"
## attr(,"package")
## [1] "Matrix"

dim(model_info$sim)

## [1] 447 447

top_items <- 20
image(model_info$sim[1:top_items, 1:top_items],
   main = "Heatmap of the first rows and columns")
```



In the next step of ML project, we will carry out the sum of rows and columns with the similarity of the objects above 0. We will visualize the sum of columns through a distribution as follows.

```
sum_rows <- rowSums(model_info$sim > 0)
table(sum_rows)

## sum_rows
##  30
## 447

sum_cols <- colSums(model_info$sim > 0)
qplot(sum_cols, fill=I("steelblue"), col=I("red"))+ ggtitle("Distribution of
the column count")

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The visualization above is very confusing because I have no clue what the distribution means. How the model should work - First, identify if we're trying to make recommendations in terms of user preference or movie preference (or both). I assume the model works by taking both factors into account. So once the model is trained, the test data that is being fed to the model contains data on the movies that a user has viewed and what he/she rated it as. So this is taken into account to identify what kind of movies the user is interested in. Then the model will crosscheck this based on other similar users and movies of similar genres. Since the movie_data was not used in this model, I believe genre was not included here. So it is purely based on ratings.

We will create a top_recommendations variable which will be initialized to 10, specifying the number of films to each user. We will then use the predict() function that will identify similar items and will rank them appropriately. Here, each rating is used as a weight. Each weight is multiplied with related similarities. Finally, everything is added in the end.

```
top_recommendations <- 10 # the number of items to recommend to each user
predicted_recommendations <- predict(object = recommen_model,
                             newdata = testing_data,
                             n = top_recommendations)
predicted_recommendations

## Recommendations as 'topNList' with n = 10 for 82 users.

user1 <- predicted_recommendations@items[[1]] # recommendation for the first
user
movies_user1 <- predicted_recommendations@itemLabels[user1]
movies_user2 <- movies_user1
```

```
for (index in 1:10){
  movies_user2[index] <- as.character(subset(movie_data,
                                             movie_data$movieId ==
movies_user1[index])$title)
}
movies_user2
```

```
##  [1] "Star Trek III: The Search for Spock (1984)"
##  [2] "Dark City (1998)"
##  [3] "Mr. Holland's Opus (1995)"
##  [4] "Godfather: Part III, The (1990)"
##  [5] "South Park: Bigger, Longer and Uncut (1999)"
##  [6] "Little Mermaid, The (1989)"
##  [7] "Tomorrow Never Dies (1997)"
##  [8] "Payback (1999)"
##  [9] "Birdcage, The (1996)"
## [10] "Cast Away (2000)"
```

```
recommendation_matrix <- sapply(predicted_recommendations@items,
                      function(x){ as.integer(colnames(movie_ratings)[x]) })
# matrix with the recommendations for each user
#dim(recc_matrix)
recommendation_matrix[,1:4]
```

```
##       [,1] [,2]  [,3] [,4]
##  [1,] 1375    1 49272    1
##  [2,] 1748    3   440   47
##  [3,]   62    5    17   62
##  [4,] 2023    6   112   70
##  [5,] 2700    7   288  111
##  [6,] 2081   11   442  368
##  [7,] 1722   16   543  485
##  [8,] 2490   17  1094 1285
##  [9,]  141   19  1097 2005
## [10,] 4022   21  1127 2424
```

#Conclusion and Discussion As a precaution, we should validate and check the source of our data to ensure that it is usable. Cleaning the data helps but we should ensure that the data collected is authentic so we can make the appropriate recommendations to users. Next, while this method only focuses on the IBCF algorithm, there are ways to estimate multiple algorithms in one go, so we should not assume IBCF is the best algorithm. We can estimate the different models and evaluate them easily and select the best performing one.

As for the results, since this a recommendation model, we don't have any actuals to compare our results to. We will require users to rate what they think of our recommendations in order to evaluate how well our model performed.