Project: Movie Recommendation with Movielens

Murat Ulcay

1. INTRODUCTION

One of the most basic ways to make recommendations is to go with the knowledge of the crowd and recommend what is already the most popular. The aim of this project is to create a personalized movie recommendation system using the 10 M version of the MovieLens dataset. I am going to apply what I have learned on the MovieLens data to build movie recommendations based on what movies users consume. Thus, the aim is to predict movie ratings given the other available features of the dataset.

The dataset includes six columns:

> **userId:** Unique identification number for each user,
> **movieId:** Unique identification number for each movie,
> **rating:** Users' evaluation of the movies which are made on a 5-star scale, with half-star increments,
> **timestamp:** Users' evaluation times of the movies which represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.
> **title:** Title of the movie being rated,
> **genre:** Genre of the movie being rated.

Below is the code to load the necessary dataset into R environment:

```
# installing the required packages (tidyverse, caret, data.table):
if(!require(tidyverse)) install.packages("tidyverse",
                                         repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse

## -- Attaching packages -----------------------------------------------------------------------------

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0

## -- Conflicts -------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
if(!require(caret)) install.packages("caret",
                                     repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
## The following objects are masked from 'package:Metrics':
##
##     precision, recall
```

```
if(!require(data.table)) install.packages("data.table",
                                          repos = "http://cran.us.r-project.org")
```

```
## Loading required package: data.table
```

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

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

```
## The following object is masked from 'package:purrr':
##
##     transpose
```

```
## The following objects are masked from 'package:reshape2':
##
##     dcast, melt
```

```
# loading the required packages (tidyverse, caret, data.table):
library(tidyverse)
library(caret)
library(data.table)

# Downloading and arranging "MovieLens 10M dataset" from
# "http://files.grouplens.org/datasets/movielens/ml-10m.zip"):
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t",
                             readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))


movielens <- left_join(ratings, movies, by = "movieId")
```

2. DATA EXPLORATION AND PREPROCESSING

In this part we will concentrate on data exploration and preprocessing for optimal model building.

Let's get more familiar with `movielens` dataset:

```
# Have a glimpse at the dataset
movielens %>% glimpse()
```

```
## Rows: 10,000,054
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ movieId   <dbl> 122, 185, 231, 292, 316, 329, 355, 356, 362, 364, 370, 37...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ...
## $ timestamp <int> 838985046, 838983525, 838983392, 838983421, 838983392, 83...
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Dumb & Dumber (19...
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Comedy", "Act...
```

```r
# The number of distinct users and movies:
paste("There are ", n_distinct(movielens$userId),
      " distinct users in the `movielens` dataset.")
```

```
## [1] "There are  69878  distinct users in the `movielens` dataset."
```

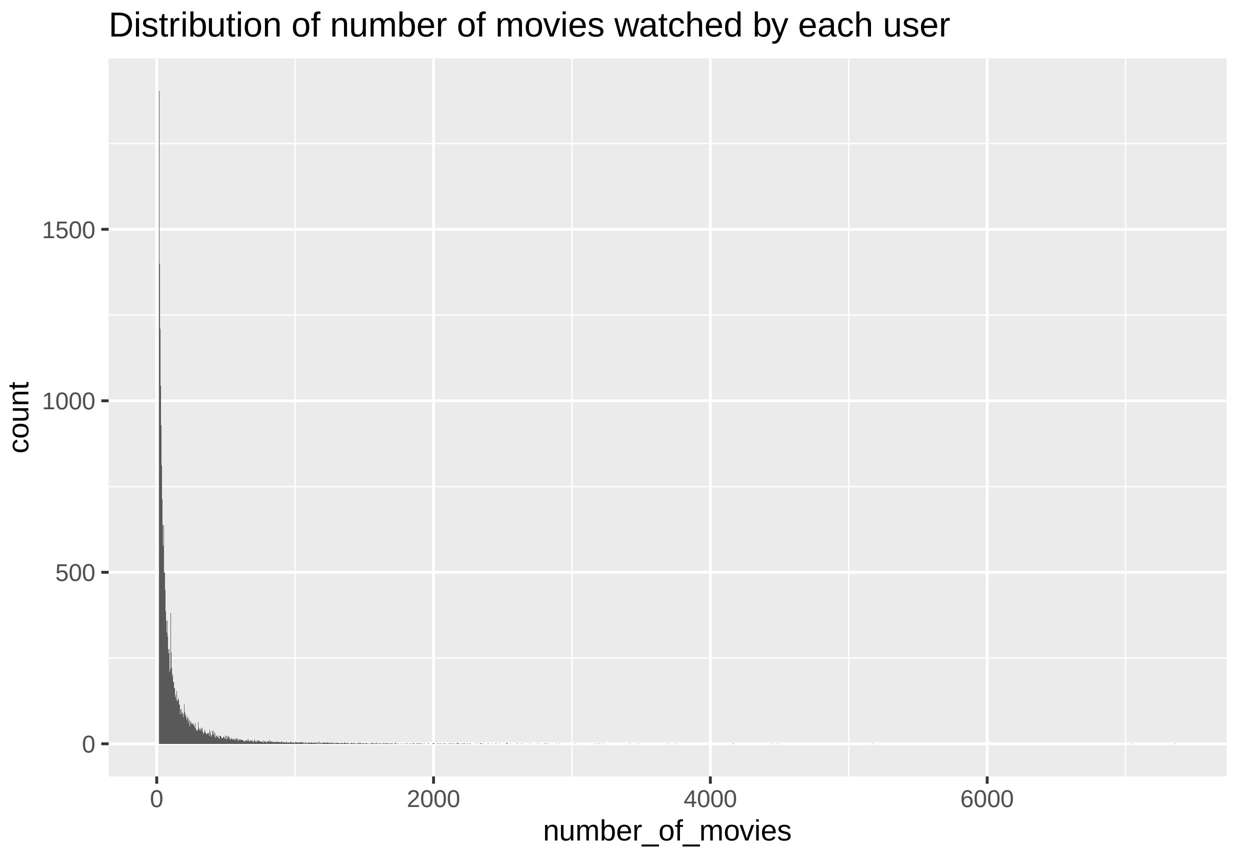
```r
paste("There are ", n_distinct(movielens$movieId),
      " distinct movies in the `movielens` dataset")
```

```
## [1] "There are  10677  distinct movies in the `movielens` dataset"
```

```r
# Displaying distribution of the number of movies watched by users:
movielens_group_user <- movielens %>%
  group_by(userId) %>%
  summarize(number_of_movies = n_distinct(movieId))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
movielens_group_user %>%
  ggplot(aes(x = number_of_movies)) +
  geom_bar() +
  ggtitle("Distribution of number of movies watched by each user")
```



From the above distribution graphic, we understand that the distribution of number of movies watched per user is very dispersed:

```r
# Calculating the descriptive statistics of the `number_of_movies`
# watched by each user:
summary(movielens_group_user$number_of_movies)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    20.0    35.0    69.0   143.1   156.0  7359.0
```

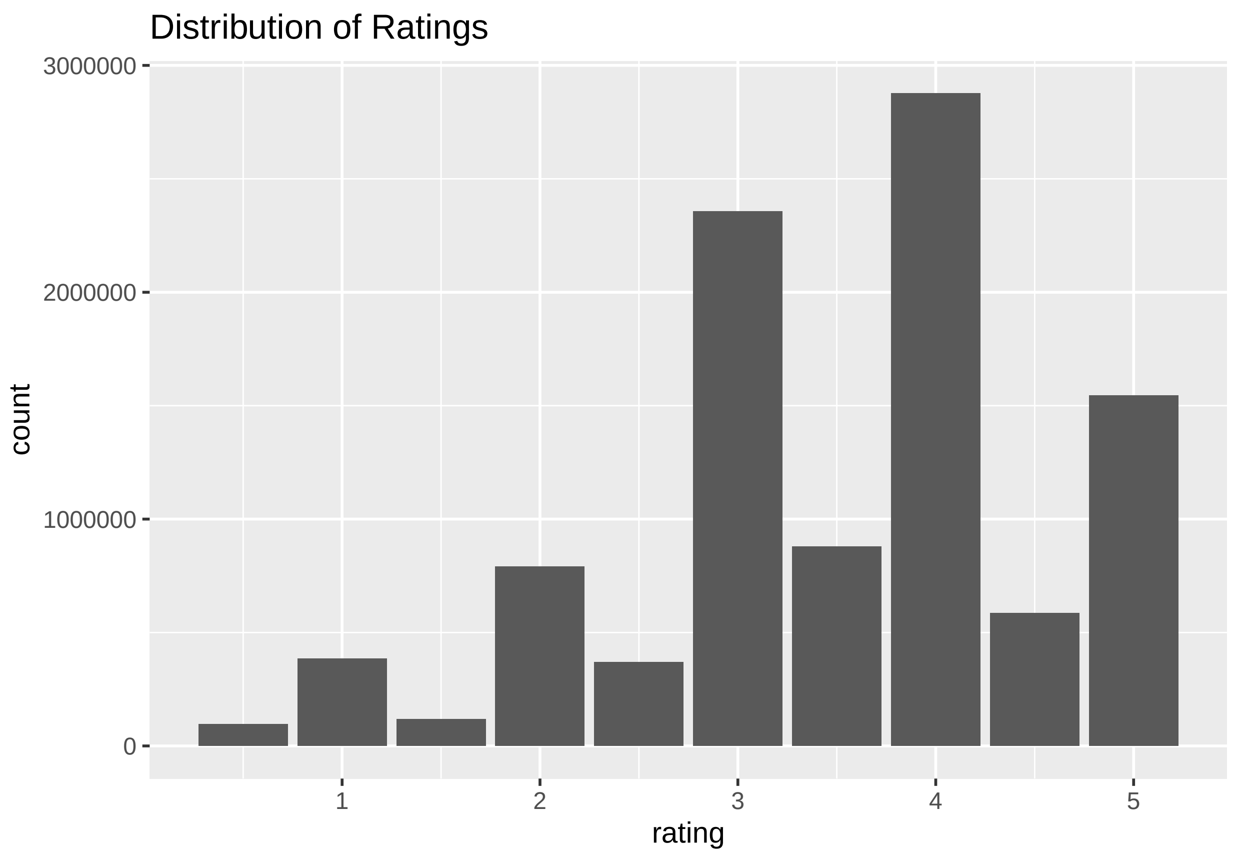
```r
# Displaying distribution of the number of users by each movie:
movielens_group_movie <- movielens %>%
  group_by(title) %>%
  summarize(number_of_users = n_distinct(userId))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
# Printing the titles of the top five most frequently seen movies:
head(movielens_group_movie[order(movielens_group_movie$number_of_users,
                                 decreasing = TRUE),])
```

```
## # A tibble: 6 x 2
##   title                            number_of_users
##   <chr>                                      <int>
## 1 Pulp Fiction (1994)                        34864
## 2 Forrest Gump (1994)                        34457
## 3 Silence of the Lambs, The (1991)           33668
## 4 Jurassic Park (1993)                       32631
## 5 Shawshank Redemption, The (1994)           31126
## 6 Braveheart (1995)                          29154
```

```r
# Displaying a histogram of the values in the `rating` column
# of the `movielens` dataset:
movielens %>%
  ggplot(aes(x = rating)) +
  geom_bar() +
  ggtitle("Distribution of Ratings")
```



The distribution of ratings is somewhat skewed to the left.

```
# Calculating the descriptive statistics of the `ratings` column in
# `movielens` dataset:
summary(movielens$rating)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.000   4.000   3.512   4.000   5.000
```

```
# Printing the titles of the top five highest rated movies:
movielens %>%
  group_by(title) %>%
  summarize(average_rating = mean(rating)) %>%
  arrange(desc(average_rating)) %>%
  head()
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 6 x 2
##   title                                      average_rating
##   <chr>                                               <dbl>
## 1 Blue Light, The (Das Blaue Licht) (1932)             5
## 2 Fighting Elegy (Kenka erejii) (1966)                 5
## 3 Satan's Tango (Sátántangó) (1994)                    5
## 4 Shadows of Forgotten Ancestors (1964)                5
## 5 Sun Alley (Sonnenallee) (1999)                       5
## 6 Constantine's Sword (2007)                           4.75
```

Despite this being a real-world dataset, one might be surprised that the highest rated movies are not movies that was most frequently seen by the users. This is because very infrequently viewed movies are skewing the results.

We can have two different non-personalized recommendation methods, using the two different rankings above. However, as we can notice, they both have their own weaknesses.

Finding the most frequently watched movies will show us what has been watched, but not how people explicitly feel about it. However, finding the average ratings has the opposite problem where we have users' explicit feedback, but individual preferences are skewing the data.

We can combine both methods to find the average rating for movies that have been reviewed more than a specified number of times:

```
# Printing the first top five highest rated movies watched
# by at least 100 users:
movielens %>%
  group_by(title) %>%
  summarize(average_rating = mean(rating),
            number_of_users = n_distinct(userId)) %>%
  filter(number_of_users>100) %>%
  arrange(desc(average_rating)) %>%
  head()
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 6 x 3
##   title                                          average_rating number_of_users
##   <chr>                                                   <dbl>           <int>
## 1 Shawshank Redemption, The (1994)                         4.46           31126
## 2 Godfather, The (1972)                                    4.42           19814
## 3 Usual Suspects, The (1995)                               4.37           24037
## 4 Schindler's List (1993)                                  4.36           25777
## 5 Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)            4.32            3255
## 6 Casablanca (1942)                                        4.32           12507
```

While suggesting the highest-ranked items will generally return items that most people do not object to, it lacks any understanding of user tastes, or what items are liked by the same people.

Since the aim of this research is to create a model to make movie recommendations,in order to develop a more solid model and correcting the downward skewness of the number of ratings per users, I included only the users with number of ratings five times as large as the median of the number of movies per user of the whole `movielens` dataset.

```
# Finding the median number of movies watched by each user:
median_movies_user <- median(movielens_group_user$number_of_movies)
```

```
# Creating a mask for the five times of the median number of
# movies per user:
mask_user <- movielens_group_user[movielens_group_user$number_of_movies>5*median_movies_user,]
```

```
# Applying the `mask_user`to the actual `movielens` dataset:
movielens_ <- movielens[movielens$userId %in% mask_user$userId,]
# Printing the number of users in the final dataset:
nrow(movielens_)
```

```
## [1] 4354532
```

3. METHODS AND ANALYSIS

Our first step is to create the `edx` and the `validation` datasets from the `movielens` dataset. We will develop our algorithms using the `edx` dataset. For a final test of our final algorithms, we will predict movie ratings in the `validation` set (the final hold-out test set) as if they were unknown.

RMSE will be used to evaluate how close our predictions are to the true values in the validation set (the final hold-out test set).

```
# Validation set will be 10% of MovieLens data:
set.seed(1)
test_index <- createDataPartition(y = movielens_$rating, times = 1,
                                  p = 0.1, list = FALSE)
edx <- movielens_[-test_index,]
temp <- movielens_[test_index,]

# Making sure that userId and movieId in validation set are also in edx set:
validation <- temp %>%
      semi_join(edx, by = "movieId") %>%
      semi_join(edx, by = "userId")

# Adding rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
edx <- rbind(edx, removed)
```

As the first step we format `edx` data. We begin with a dataset containing users and their ratings as individual rows with the following columns: `userId`, `title`, `rating`

```
# Converting `edx` dataset to a `dataframe`:
edx <- as.data.frame(edx)

# Selecting `userId`, `title` and `rating`:
edx_ <- edx[c("userId","title","rating")]
```

We need to transform the `edx` dataframe into a *user rating matrix* where each row represents a user, and each column represents the movies on the platform. This will us to easily compare users and their preferences (Warning: Running the code requires more than one hour):

```
# Loading `reshape2` library for `dcast()` function:
library(reshape2)
# Creating user-rating matrix:
user_rating_matrix <- reshape2::dcast(data = edx_,
                                      formula = userId ~ title,
                                      fun.aggregate=mean, value.var = "rating")

# Checking the class of `user_rating_matrix`:
class(user_rating_matrix)
```

```
## [1] "data.frame"
```

```
# Checking the dimentions of `user_rating_matrix`:
dim(user_rating_matrix)
```

```
## [1]  6697 10651
```

We have a matrix consisting of 69.878 rows (which is equal to the unique number of users) and 8.317 columns (which is equal to the movies watched at least once by one of the users).

To clear the data we delete the column named `NA` from the dataset.

```r
# Printing the first 5 rows and first 4 columns of `user_rating_matrix`:
user_rating_matrix[1:5,1:4]
```

```
##   userId ...All the Marbles (a.k.a. The California Dolls) (1981)
## 1      8                                                      NaN
## 2     18                                                      NaN
## 3     34                                                      NaN
## 4     36                                                      NaN
## 5     51                                                      NaN
##   ...And God Created Woman (Et Dieu... créa la femme) (1956)
## 1                                                        NaN
## 2                                                        NaN
## 3                                                        NaN
## 4                                                        NaN
## 5                                                        NaN
##   ...And God Spoke (1993)
## 1                     NaN
## 2                     NaN
## 3                     NaN
## 4                     NaN
## 5                     NaN
```

```r
# Setting the userId as index (row names) of the user_rating_matrix dataframe:
rownames(user_rating_matrix) <- user_rating_matrix$userId

# Eliminating the `userId` column of the `user_rating_matrix`:
user_rating_matrix <- user_rating_matrix[,-1]

# Printing the first 5 rows and first 4 columns of `user_rating_matrix`:
user_rating_matrix[1:5,1:4]
```

```
##    ...All the Marbles (a.k.a. The California Dolls) (1981)
## 8                                                       NaN
## 18                                                      NaN
## 34                                                      NaN
## 36                                                      NaN
## 51                                                      NaN
##    ...And God Created Woman (Et Dieu... créa la femme) (1956)
## 8                                                          NaN
## 18                                                         NaN
## 34                                                         NaN
## 36                                                         NaN
## 51                                                         NaN
##    ...And God Spoke (1993) ...And Justice for All (1979)
## 8                      NaN                           NaN
## 18                     NaN                           NaN
## 34                     NaN                           NaN
## 36                     NaN                           NaN
## 51                     NaN                           NaN
```

From the output showing only the first 5 rows and first 4 columns of `user_rating_matrix`, it can be noticed it has lots of missing data. This is to be expected since the majority of users rate only a small number of items, users rarely see all movies, and most movies are not seen by everyone, resulting in gaps in the user rating matrix. Merely filling in gaps with zeros without adjusting the data otherwise can cause issues by skewing the reviews more negative and should not be done. Wow we deal with users who do not have ratings for an item can greatly influence the validity models.

In order to fill in missing data with information that should not bias the data, we'll get the average score each user has given across all their ratings, and then use this average to center the users' scores around zero. Finally, we'll be able to fill in the empty values with zeros, which is now a neutral score, minimizing the impact on their overall profile, but still allowing the comparison of users.

```
# Finding the average of the ratings given by each user in
# user_rating_matrix:
average_ratings <- rowMeans(user_rating_matrix, na.rm=TRUE)

# Creating a function for subtracting `average_ratings`
# vector from another vector:
subtracting <- function(x) {
    dif <- x - average_ratings
    return(dif)
}

# Subtracting the row averages from each column in user_rating_matrix,
# to center each users ratings around 0:
user_rating_matrix_centered <- apply(user_rating_matrix, 2, subtracting )

# Printing the first 5 rows and first 4 columns of `user_rating_matrix_centered`:
user_rating_matrix_centered[1:5,1:4]
```

```
##    ...All the Marbles (a.k.a. The California Dolls) (1981)
## 8                                                      NaN
## 18                                                     NaN
## 34                                                     NaN
## 36                                                     NaN
## 51                                                     NaN
##    ...And God Created Woman (Et Dieu... créa la femme) (1956)
## 8                                                         NaN
## 18                                                        NaN
## 34                                                        NaN
## 36                                                        NaN
## 51                                                        NaN
##    ...And God Spoke (1993) ...And Justice for All (1979)
## 8                      NaN                           NaN
## 18                     NaN                           NaN
## 34                     NaN                           NaN
## 36                     NaN                           NaN
## 51                     NaN                           NaN
```

```
#Filling the empty values in the newly created `user_rating_matrix_centered` with zeros:
user_rating_matrix_normed <- replace_na(user_rating_matrix_centered,0)

# Printing the first 5 rows and first 4 columns of `user_rating_matrix_normed`:
user_rating_matrix_normed[1:5,1:4]
```

```
##    ...All the Marbles (a.k.a. The California Dolls) (1981)
## 8                                                        0
## 18                                                       0
## 34                                                    0
## 36                                                    0
## 51                                                    0
##    ...And God Created Woman (Et Dieu... créa la femme) (1956)
## 8                                                        0
## 18                                                       0
## 34                                                       0
## 36                                                       0
## 51                                                       0
##    ...And God Spoke (1993) ...And Justice for All (1979)
## 8                        0                             0
## 18                       0                             0
## 34                       0                             0
## 36                       0                             0
## 51                       0                             0
```

It can now be compared between rows without adding an unnecessary bias to the data when values are missing.

Spartsity Methods like KNN (K-nearest neighbors) works great for dense datasets in which every item has been reviewed by multiple people, which ensures that the K nearest neighbors are genuinely similar to the user of concern. However, the sparsity of the dataset is actually a common concern in real-world rating data as the number of users and items are generally quite high and the number of reviews are quite low. The percentage of a dataset that is empty is called the dataset's **sparsity**. In other words, the number of empty cells over the number of cells with data.

```r
# Counting the number of empty cells in `user_rating_matrix`:
nas <- sum(is.na(user_rating_matrix))

# Counting the total number of cells in `user_rating_matrix`:
size <- dim(user_rating_matrix)[1] * dim(user_rating_matrix)[2]

# Calculating the sparsity of the `user_rating_matrix` by
# dividing `nas` by `size` and printing the result.
sparsity <- nas/size
print(sparsity)
```

```
## [1] 0.9450517
```

As can be seen, the `user_rating_matrix`dataframe which we use in our analysis is over 98.5% empty. This means that less than 1.5% of the dataframe includes any data. This suggests that it would be limited in its value for making predictions using KNN.

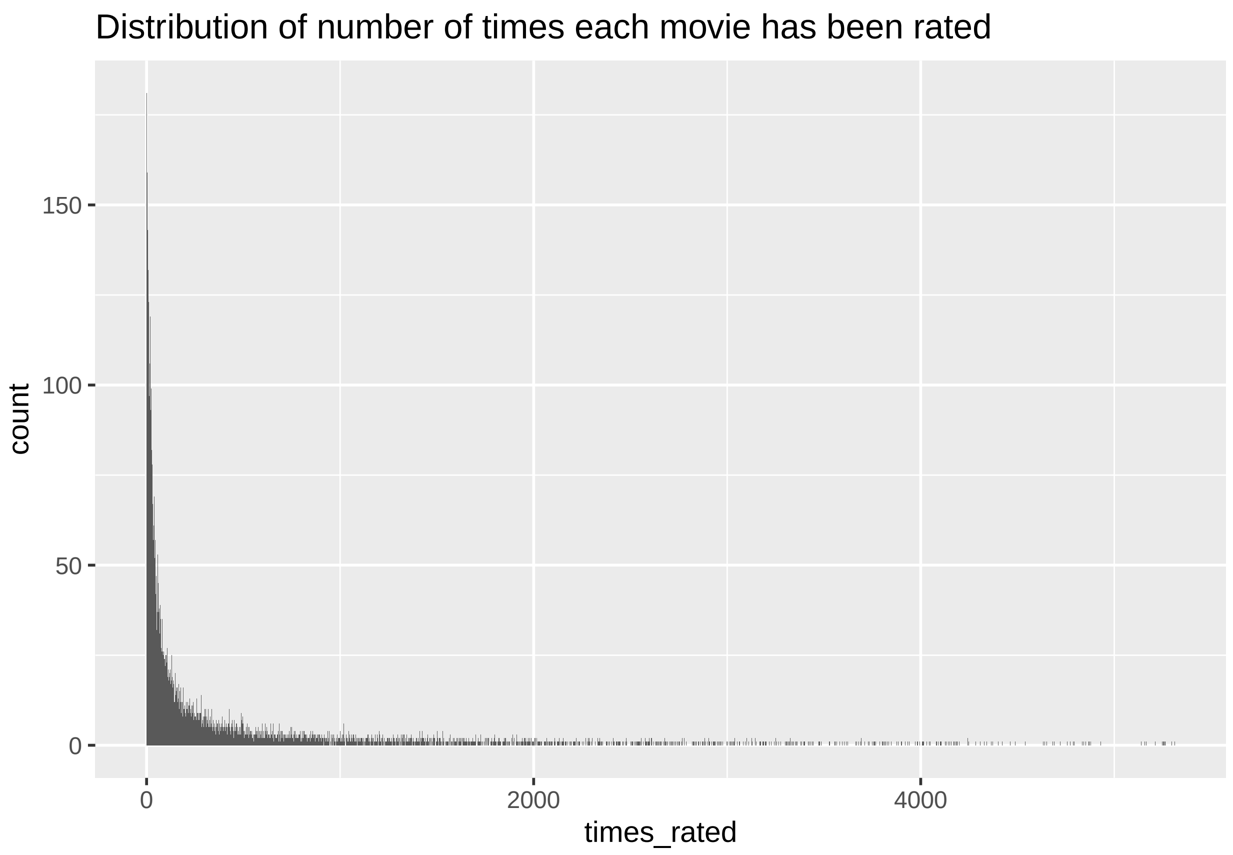
We can also count how often each movie in the `user_rating_matrix` dataframe has been given a rating, and then see the distribution of the number of times each movie has been rated.

```r
# Creating a counter function which counts non-empty cells in its argument:
count_nonna <- function(x) {
    length(x) - sum(is.na(x))
}

# Counting the number of non-empty cells in each column of `user_rating_matrix`:
nas_col <- apply(user_rating_matrix, 2, count_nonna)

# Converting the `nas_col` vector into a dataframe and printing the first 6 rows:
nas_col_df <- as.data.frame(nas_col)
```

```r
colnames(nas_col_df) <- "times_rated"
head(nas_col_df)
```

```
##                                                           times_rated
## ...All the Marbles (a.k.a. The California Dolls) (1981)            13
## ...And God Created Woman (Et Dieu... créa la femme) (1956)         47
## ...And God Spoke (1993)                                            19
## ...And Justice for All (1979)                                     327
## 'burbs, The (1989)                                                885
## 'night Mother (1986)                                              116
```

```r
# Creating a distribution graph of the number of times each movie has rated:
nas_col_df %>%
  ggplot(aes(x=times_rated)) +
  geom_bar() +
  ggtitle("Distribution of number of times each movie has been rated")
```



```r
# Calculating the statistics of the `times_rated` column:
summary(nas_col_df$times_rated)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     1.0    23.0    85.0   368.0   344.8  5312.0
```

```r
# Calculating proportional (to the number of users) statistics of
# the `times_rated` column:
summary(nas_col_df$times_rated)/nrow(user_rating_matrix)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.0001493 0.0034344 0.0126923 0.0549483 0.0514783 0.7931910
```

It can be inferred from the statistics above that %75 (3rd Quarter) of the movies are rated by at most 13% of the total users. This also shows how sparse our dataset is.

MATRIX FACTORIZATION

Matrix factorization is applied for our research. Just as matrices can be multiplied together, they can be broken into their factors.

A huge benefit of this, when performed in conjunction with recommendation systems, is that factors can be found as long as there is at least one value in every row and column. Or, in other words, every user has given at least one rating, and every item has been rated at least once, which is the case for our `user_rating_matrix` dataset.

This is valuable, because these factors can be multiplied together to create a fully filled in matrix. Thus, calculating what values should be in the gaps of the sparse matrix based off of the incomplete matrix's factors is possible.

Singular Value Decomposition The `user_rating_matrix_normed` dataset is prepared above by centering `user_rating_matrix` dataset and then filling in the remaining empty values with zeros. Using this dataset, matrix factors can be found. We need to break the `user_ratings_normed` dataset into 3 factors: U, sigma, and Vt:

> **U:** is a matrix with a row for each user,
> **Vt:** has a column for each movie,
> **sigma:** is an array of weights that is needed to convert to a diagonal matrix

Thus, predictions for the `user_ratings_normed` dataset can be found by the following:

> U x sigma x Vt

```
# Computing the Singular Value Decomposition (SVD) matrix of the
# `user_ratings_normed` dataset:
SVD_ <- svd(user_rating_matrix_normed)

# Checking the elements of SVD_:
names(SVD_)
```

```
## [1] "d" "u" "v"
```

The elements of SVD__ are `d` (required to be diagonalized) , `u` and `v`, corresponding to our Singular Value Decomposition matrices of `sigma`, `U` and `Vt` respectively.

```
# Diagonalizing the vector `d` in SVD_ elements, and finding
# sigma component of matrix factorization:
sigma <- diag(SVD_$d)

# Extracting `u` element of `SVD_` list as U:
U <- SVD_$u

# Extracting `v` element of `SVD_` list as Vt:
Vt <- t(SVD_$v)

# Multiplication of U, sigma and Vt (prediction of
# `user_ratings_normed` dataset):
U_sigma_Vt <- U %*% sigma %*% Vt


# Creating a function for adding `average_ratings`
```

```r
# (original dataframe's row averages) vector to another vector:
adding <- function(x) {
    add <- x + average_ratings
    return(add)
}

# Adding the row averages to each column in U_sigma_Vt,
# to uncenter each users ratings (prediction results):
user_rating_matrix_uncentered <- apply(U_sigma_Vt, 2, adding)

# Printing the first 20 rows and first 20 columns of `user_rating_matrix_uncentered`:
user_rating_matrix_uncentered[1:20,1:20]
```

```
##         [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]     [,8]
## 8   3.389503 3.389503 3.389503 3.389503 4.000000 3.389503 3.389503 3.389503
## 18  3.481030 3.481030 3.481030 3.481030 2.500000 3.481030 3.481030 3.481030
## 34  2.729965 2.729965 2.729965 2.729965 2.729965 2.729965 2.729965 2.729965
## 36  3.359629 3.359629 3.359629 3.359629 3.359629 3.359629 3.359629 3.359629
## 51  4.443953 4.443953 4.443953 4.443953 4.443953 4.443953 4.443953 4.443953
## 56  4.397172 4.397172 4.397172 4.397172 4.397172 4.397172 4.397172 4.397172
## 65  3.192678 3.192678 3.192678 3.192678 3.000000 3.192678 3.192678 3.192678
## 78  4.132701 4.132701 4.132701 4.132701 4.132701 4.132701 4.132701 4.132701
## 96  3.525164 3.525164 3.525164 3.525164 3.525164 3.525164 3.525164 3.525164
## 112 3.292225 3.292225 3.292225 3.292225 3.292225 3.292225 3.292225 3.292225
## 122 4.005236 4.005236 4.005236 4.005236 4.005236 4.005236 4.005236 4.005236
## 125 3.626923 3.626923 3.626923 3.626923 3.626923 3.626923 3.626923 3.626923
## 126 3.190278 3.190278 3.190278 3.190278 3.190278 3.190278 3.190278 3.190278
## 139 3.440129 3.440129 3.440129 3.440129 3.440129 3.440129 3.440129 3.440129
## 143 3.400442 3.400442 3.400442 3.400442 5.000000 3.400442 3.400442 3.400442
## 144 3.669243 3.669243 3.669243 3.669243 3.669243 3.669243 3.669243 3.669243
## 182 3.706224 3.706224 3.706224 3.706224 3.000000 3.706224 3.706224 3.000000
## 196 3.625000 3.625000 3.625000 3.625000 3.625000 3.625000 3.625000 3.625000
## 198 4.058912 4.058912 4.058912 3.500000 4.058912 4.058912 4.058912 4.058912
## 212 3.352011 3.352011 3.352011 3.352011 3.352011 3.352011 3.352011 3.352011
##         [,9]    [,10]    [,11]    [,12]    [,13]    [,14]    [,15]    [,16]
## 8   3.389503 3.389503 3.500000 3.389503 3.389503 3.389503 3.389503 3.389503
## 18  3.481030 3.481030 3.481030 3.481030 3.481030 3.481030 3.481030 3.481030
## 34  2.729965 2.729965 2.729965 2.729965 2.729965 2.729965 2.729965 2.729965
## 36  3.359629 3.359629 3.359629 3.359629 3.359629 3.359629 3.359629 3.359629
## 51  4.443953 4.443953 4.443953 4.443953 4.443953 4.443953 4.443953 4.443953
## 56  4.397172 4.397172 4.397172 4.397172 4.397172 4.397172 4.397172 4.397172
## 65  3.192678 3.192678 3.192678 3.192678 3.192678 3.192678 3.192678 3.192678
## 78  4.132701 4.132701 4.132701 4.132701 4.132701 4.132701 4.132701 4.132701
## 96  3.525164 3.525164 3.525164 3.525164 3.525164 3.525164 3.525164 3.525164
## 112 3.292225 3.292225 3.292225 3.292225 3.292225 3.292225 3.292225 3.292225
## 122 4.005236 4.005236 4.005236 4.005236 4.005236 4.005236 4.005236 4.005236
## 125 3.626923 3.626923 2.500000 3.626923 3.626923 3.626923 3.626923 3.626923
## 126 3.190278 3.190278 3.190278 3.190278 3.190278 3.190278 3.190278 3.190278
## 139 3.440129 3.440129 3.440129 3.440129 3.440129 3.440129 3.440129 3.440129
## 143 3.400442 3.400442 3.400442 3.400442 3.400442 3.400442 3.400442 3.400442
## 144 3.669243 3.669243 3.669243 3.669243 3.669243 3.669243 3.669243 3.669243
## 182 3.706224 3.706224 3.706224 3.706224 3.706224 3.706224 3.706224 3.706224
## 196 3.625000 3.625000 3.625000 3.625000 3.625000 3.625000 3.625000 3.625000
## 198 4.058912 4.058912 4.058912 4.058912 4.058912 4.058912 4.058912 4.058912
## 212 3.352011 3.352011 3.352011 3.352011 3.352011 3.352011 3.352011 3.352011
##        [,17]    [,18]    [,19]    [,20]
## 8   3.500000 3.389503 3.389503 3.389503
## 18  4.000000 3.481030 3.481030 3.481030
## 34  2.729965 2.729965 2.729965 2.729965
## 36  4.000000 3.359629 3.359629 3.359629
## 51  4.443953 4.443953 4.443953 4.443953
## 56  4.397172 4.397172 4.397172 4.397172
## 65  3.192678 3.192678 3.192678 3.192678
## 78  4.132701 4.132701 4.132701 4.132701
## 96  3.525164 3.525164 3.525164 3.525164
## 112 3.292225 3.292225 3.292225 3.292225
## 122 4.005236 4.005236 4.005236 4.005236
## 125 3.626923 3.626923 3.626923 3.626923
## 126 3.190278 3.190278 3.190278 3.190278
## 139 3.440129 3.440129 3.440129 3.440129
## 143 4.000000 3.400442 3.400442 3.400442
## 144 3.669243 3.669243 3.669243 3.669243
## 182 3.000000 3.706224 3.706224 3.706224
## 196 3.625000 3.625000 3.625000 3.625000
## 198 4.000000 4.058912 4.058912 4.058912
## 212 3.352011 3.352011 3.352011 3.352011
```

```
dim(sigma)
```

```
## [1] 6697 6697
```

```
dim(U)
```

```
## [1] 6697 6697
```

```
dim(Vt)
```

```
## [1]  6697 10650
```

```
# Creating a dataframe of the prediction results:
user_rating_matrix_pred <- as.data.frame(user_rating_matrix_uncentered)

# Adding row and column names to `user_rating_matrix_pred` dataset:
rownames(user_rating_matrix_pred) <- rownames(user_rating_matrix)
colnames(user_rating_matrix_pred) <- colnames(user_rating_matrix)
```

After all this process we have the complete user rating matrix with the predictions filled in each data point of the dataset:

```
# Printing the first 20 rows and first 5 columns of `user_rating_matrix_pred` dataset:
user_rating_matrix_pred[1:20,1:5]
```

```
##     ...All the Marbles (a.k.a. The California Dolls) (1981)
## 8                                                  3.389503
## 18                                                 3.481030
## 34                                                 2.729965
## 36                                                 3.359629
## 51                                                 4.443953
## 56                                                 4.397172
## 65                                                 3.192678
## 78                                                 4.132701
## 96                                                 3.525164
## 112                                             3.292225
## 122                                             4.005236
## 125                                             3.626923
## 126                                             3.190278
## 139                                             3.440129
## 143                                             3.400442
## 144                                             3.669243
## 182                                             3.706224
## 196                                             3.625000
## 198                                             4.058912
## 212                                             3.352011
##     ...And God Created Woman (Et Dieu... créa la femme) (1956)
## 8                                                 3.389503
## 18                                                3.481030
## 34                                                2.729965
## 36                                                3.359629
## 51                                                4.443953
## 56                                                4.397172
## 65                                                3.192678
## 78                                                4.132701
## 96                                                3.525164
## 112                                               3.292225
## 122                                               4.005236
## 125                                               3.626923
## 126                                               3.190278
## 139                                               3.440129
## 143                                               3.400442
## 144                                               3.669243
## 182                                               3.706224
## 196                                               3.625000
## 198                                               4.058912
## 212                                               3.352011
##     ...And God Spoke (1993) ...And Justice for All (1979) 'burbs, The (1989)
## 8                  3.389503                      3.389503          4.000000
## 18                 3.481030                      3.481030          2.500000
## 34                 2.729965                      2.729965          2.729965
## 36                 3.359629                      3.359629          3.359629
## 51                 4.443953                      4.443953          4.443953
## 56                 4.397172                      4.397172          4.397172
## 65                 3.192678                      3.192678          3.000000
## 78                 4.132701                      4.132701          4.132701
## 96                 3.525164                      3.525164          3.525164
## 112                3.292225                      3.292225          3.292225
## 122                4.005236                      4.005236          4.005236
## 125                3.626923                      3.626923          3.626923
## 126                3.190278                      3.190278          3.190278
## 139                3.440129                      3.440129          3.440129
## 143                3.400442                      3.400442          5.000000
## 144                3.669243                      3.669243          3.669243
## 182                3.706224                      3.706224          3.000000
## 196                3.625000                      3.625000          3.625000
## 198                4.058912                      3.500000          4.058912
## 212                3.352011                      3.352011          3.352011
```

```r
# Printing the first 20 rows and first 5 columns of `user_rating_matrix`:
user_rating_matrix[1:20,1:5]
```

```
##     ...All the Marbles (a.k.a. The California Dolls) (1981)
## 8                                                       NaN
## 18                                                      NaN
## 34                                                      NaN
## 36                                                      NaN
## 51                                                      NaN
## 56                                                      NaN
## 65                                                      NaN
## 78                                                      NaN
## 96                                                      NaN
## 112                                                     NaN
## 122                                                     NaN
## 125                                                     NaN
## 126                                                     NaN
## 139                                                     NaN
## 143                                                     NaN
## 144                                                     NaN
## 182                                                     NaN
## 196                                                     NaN
## 198                                                     NaN
## 212                                                     NaN
##     ...And God Created Woman (Et Dieu... créa la femme) (1956)
## 8                                                          NaN
## 18                                                         NaN
## 34                                                         NaN
## 36                                                         NaN
## 51                                                         NaN
## 56                                                         NaN
## 65                                                         NaN
## 78                                                         NaN
## 96                                                         NaN
## 112                                                        NaN
## 122                                                        NaN
## 125                                                        NaN
## 126                                                        NaN
## 139                                                        NaN
## 143                                                        NaN
## 144                                                        NaN
## 182                                                        NaN
## 196                                                        NaN
## 198                                                        NaN
## 212                                                        NaN
##     ...And God Spoke (1993) ...And Justice for All (1979) 'burbs, The (1989)
## 8                       NaN                           NaN               4.0
## 18                      NaN                           NaN               2.5
## 34                      NaN                           NaN               NaN
## 36                      NaN                           NaN               NaN
## 51                      NaN                           NaN               NaN
## 56                      NaN                           NaN               NaN
## 65                      NaN                           NaN               3.0
## 78                      NaN                           NaN               NaN
## 96                        NaN                           NaN                 NaN
## 112                       NaN                           NaN                 NaN
## 122                       NaN                           NaN                 NaN
## 125                       NaN                           NaN                 NaN
## 126                       NaN                           NaN                 NaN
## 139                       NaN                           NaN                 NaN
## 143                       NaN                           NaN                 5.0
## 144                       NaN                           NaN                 NaN
## 182                       NaN                           NaN                 3.0
## 196                       NaN                           NaN                 NaN
## 198                       NaN                           3.5                 NaN
## 212                       NaN                           NaN                 NaN
```

In order to measure our model accuracy, RMSE (Root Mean Square Error) is used. The actual ratings and the predicted ratings will be compared to find this measure of accuracy:

```
# Creating the mask to find filter the actually rated items by users:
mask<- !is.na(user_rating_matrix)

# loading the `Metrics` library for `rmse()` function:
library(Metrics)

# Finding the RMSE by comparing the `user_rating_matrix_pred`
# (giving the rating predictions) and
# `user_rating_matrix` (actual ratings) datasets:
RMSE_model <- rmse(user_rating_matrix[mask],user_rating_matrix_pred[mask])
paste("RMSE of the model is:",RMSE_model)
```

```
## [1] "RMSE of the model is: 0.000000000000000597857435590898"
```

MODEL VALIDATION AND RESULTS

In order to validate the RMSE result we found above, we conduct SVD algorithm also with the `validation` dataset and with the same process and found the final RMSE.

```
# Converting `validation` dataset to a `dataframe`:
validation <- as.data.frame(validation)

# Selecting `userId`, `title` and `rating`:
validation_ <- validation[c("userId","title","rating")]

# Calculating the dimensions of the `validation_` dataset:
dim(validation_)
```

```
## [1] 435436      3
```

```
# Creating user-rating matrix_val:
user_rating_matrix_val <- reshape2::dcast(data = validation,
                                          formula = userId ~ title,
                                          fun.aggregate=mean, value.var = "rating")

# Checking the dimentions of `user_rating_matrix_val`:
dim(user_rating_matrix_val)
```

```
## [1] 6697 9603
```

```
# Setting the userId as index (row names) of the user_rating_matrix_val dataframe:
rownames(user_rating_matrix_val) <- user_rating_matrix_val$userId
```

```r
# Eliminating the `userId` column of the `user_rating_matrix_val`:
user_rating_matrix_val <- user_rating_matrix_val[,-1]

# Finding the average of the ratings given by each user in user_rating_matrix_val:
average_ratings_val <- rowMeans(user_rating_matrix_val, na.rm=TRUE)

# Creating a function for subtracting `average_ratings_val` vector from another vector:
subtracting_val <- function(x) {
    dif <- x - average_ratings_val
    return(dif)
}

# Subtracting the row averages from each column in user_rating_matrix_val,
# to center each users ratings around 0:
user_rating_matrix_centered_val <- apply(user_rating_matrix_val, 2,
                                         subtracting_val )

library(tidyr)
# Filling the empty values in the newly created `user_rating_matrix_centered_val` with zeros:
user_rating_matrix_normed_val <- replace_na(user_rating_matrix_centered_val,0)

# Computing the Singular Value Decomposition (SVD) matrix of the
# `user_ratings_normed_val` dataset:
SVD_val <- svd(user_rating_matrix_normed_val)

# Diagonalizing the vector `d` in SVD_val elements, and finding sigma component
# of matrix factorization:
sigma_val <- diag(SVD_val$d)

# Extracting `u` element of `SVD_val` list as U_val:
U_val <- SVD_val$u

# Extracting `v` element of `SVD_` list as Vt_val:
Vt_val <- SVD_val$v

# Multiplication of U_val, sigma_val and Vt_val (prediction of `user_ratings_normed` dataset):
U_sigma_Vt_val <- U_val %*% sigma_val %*% t(Vt_val)

# Creating a function for adding `average_ratings_val` (original dataframe's
# row averages) vector to another vector:
adding_val <- function(x) {
    add <- x + average_ratings_val
    return(add)
}

# Adding the row averages to each column in U_sigma_Vt_val, to uncenter each
# users ratings (prediction results):
user_rating_matrix_uncentered_val <- apply(U_sigma_Vt_val, 2, adding_val)

# Creating a dataframe of the prediction results:
user_rating_matrix_pred_val <- as.data.frame(user_rating_matrix_uncentered_val)

# Adding row and column names to `user_rating_matrix_pred_val` dataset:
```

```r
rownames(user_rating_matrix_pred_val) <- rownames(user_rating_matrix_val)
colnames(user_rating_matrix_pred_val) <- colnames(user_rating_matrix_val)

# Creating the mask to find filter the actually rated items by users:
mask_val<- !is.na(user_rating_matrix_val)

# Calculating the RMSE of the validation model:
RMSE_validation <- rmse(user_rating_matrix_val[mask_val],user_rating_matrix_pred_val[mask_val])
paste("RMSE of the validation model is:",RMSE_validation)
```

```
## [1] "RMSE of the validation model is: 0.00000000000000422584098489888"
```

The validation model also confirms that the RSME of the model received by the SVD algorithm is almost 0.

Now that we have the recalculated *user rating matrix* with all of its gaps filled in, the next step is to use it to generate predictions and recommendations.

Using `user_rating_matrix_pred_val` that we generated, with all rows and columns filled, we can find the movies that every user is most likely to enjoy.

Let's try this for user 96:

```r
# Selecting User 96 from `user_rating_matrix_pred-val` dataframe:
user_96_ratings <- user_rating_matrix_pred_val["96",]
# Sort the ratings of User 96 from high to low and selecting the first
# 5 movies, and creating a mask:
mask_96 <- order(user_96_ratings,decreasing = T)[1:5]
# Printing the names of these 5 movies:
names(user_rating_matrix_pred_val)[mask_96]
```

```
## [1] "Heat (1995)"
## [2] "Green Mile, The (1999)"
## [3] "My Man Godfrey (1957)"
## [4] "Englishman Who Went Up a Hill But Came Down a Mountain, The (1995)"
## [5] "Fish Called Wanda, A (1988)"
```

CONCLUSION, LIMITATIONS AND FURTHER RESEARCH

In this article, Singular Value Decomposition (SVD) algorithm is used in order to create a personalized recommendation engine. The model's RSME is close to 0.

Unfortunately, there is no information about the users in the `MovieLens` dataset other than how they rate the movies they watched. So, personalization is only through using the similarities base off of the movie ratings of different users. For further research, a dataset containing the demographic features of the users can be collected to reach at the more robust models.