Movielens Capstone Project

Awala Fortune

6/1/2020

I. Introduction Recommender Systems has gained much popularity in data science community because of the Netflix prize contest. It has emerged as an important factor for e-commerce and as machine learning technology it has a wide range of application in most industries today.

The movielens project is aimed at creating recommendation system using the edx data set in the training of the algorithms and movie ratings prediction.The Root Mean Square Error are used to evaluate the closeness of the predictions to the true values in the validation set. The 10M version of the MovieLens dataset were generated by the GroupLens research lab.

II. Method/Analysis The main methods of this paper targets to develop a recommender system based on users ratings and to evaluate the system using simple baseline techniques such as Linear regression model with regularized movie and user effects using Lambda for validation. And further built models based on popularity, similarity between items and similarity between users hence, optimizing Root Mean Square Error (RMSE)between the predicted and actual ratings as shown in the formula: RMSE = function(m, o){ sqrt(mean((m - o)^2)) } where, m: is for model (fitted) values and o: is for observed (true) values.

Dataset

```r
load("~/Awala Capstone Project/Awala_Fortune_Project.rda")
```

Loading library and dataset

```r
library(tidyverse)
```

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

```
## -- Attaching packages ------------------------------------------ tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.1     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   0.8.5
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

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

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

```
## Warning: package 'readr' was built under R version 3.6.3
## Warning: package ’purrr’ was built under R version 3.6.3

## Warning: package ’dplyr’ was built under R version 3.6.3

## Warning: package ’stringr’ was built under R version 3.6.3

## Warning: package ’forcats’ was built under R version 3.6.3

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

```
library(caret)
```

```
## Warning: package ’caret’ was built under R version 3.6.3

## Loading required package: lattice

## Warning: package ’lattice’ was built under R version 3.6.3

##
## Attaching package: ’caret’

## The following object is masked from ’package:purrr’:
##
##     lift
```

```
library(data.table)
```

```
## Warning: package ’data.table’ was built under R version 3.6.3

##
## Attaching package: ’data.table’

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

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

```
library(kableExtra)
```

```
## Warning: package ’kableExtra’ was built under R version 3.6.3

##
## Attaching package: ’kableExtra’

## The following object is masked from ’package:dplyr’:
##
##     group_rows
```

```r
library(lubridate)
```

```
## Warning: package ’lubridate’ was built under R version 3.6.3
```

```
##
## Attaching package: ’lubridate’
```

```
## The following objects are masked from ’package:data.table’:
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
```

```
## The following objects are masked from ’package:dplyr’:
##
##     intersect, setdiff, union
```

```
## The following objects are masked from ’package:base’:
##
##     date, intersect, setdiff, union
```

```r
library(Matrix.utils)
```

```
## Warning: package ’Matrix.utils’ was built under R version 3.6.3
```

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

```
## Warning: package ’Matrix’ was built under R version 3.6.3
```

```
##
## Attaching package: ’Matrix’
```

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

```r
library(DT)
```

```
## Warning: package ’DT’ was built under R version 3.6.3
```

```r
library(wordcloud)
```

```
## Warning: package ’wordcloud’ was built under R version 3.6.3
```

```
## Loading required package: RColorBrewer
```

```r
library(RColorBrewer)
library(ggthemes)
```

```
## Warning: package ’ggthemes’ was built under R version 3.6.3
```

```r
library(irlba)
```

```
## Warning: package ’irlba’ was built under R version 3.6.3
```

```r
library(recommenderlab)
```

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

```
## 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

##
## Attaching package: 'recommenderlab'

## The following objects are masked from 'package:caret':
##
##     MAE, RMSE
```

```r
library(recosystem)
```

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

```r
library(h2o)
```

```
## Warning: package 'h2o' was built under R version 3.6.3

##
## ----------------------------------------------------------------------
##
## Your next step is to start H2O:
##     > h2o.init()
##
## For H2O package documentation, ask for help:
##     > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit http://docs.h2o.ai
##
## ----------------------------------------------------------------------

##
## Attaching package: 'h2o'

## The following object is masked from 'package:arules':
##
##     %in%

## The following objects are masked from 'package:lubridate':
##
##     day, hour, month, week, year

## The following objects are masked from 'package:data.table':
##
##     hour, month, week, year

## The following objects are masked from 'package:stats':
##
##     cor, sd, var
## The following objects are masked from 'package:base':
##
##     %*%, %in%, &&, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc
```

```r
library(googledrive)
```

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

```r
library(stringr)
```

```r
url <- "https://drive.google.com/drive/folders/1IZcBBX0OmL9wu9AdzMBFUG8GoPbGQ38D?usp=sharing"
```

```r
dl <- tempfile()
download.file(url, dl)
```

```r
validation <- readRDS("C:/Users/HP/Downloads/validation.rds")
edx <- readRDS("C:/Users/HP/Downloads/edx.rds")
```

III. Results/Discussion After generating codes provided in the project overview, it is observed that the edX dataset is made of 6 features for a total of about 9,000,055 observations.The validation set which represents 10% of the 10M Movielens dataset contains the same features , but with a total of 999,999 occurences. we made sure that userId and movieId in edx set are also in validation set. Each row represents a rating given by one user to one movie. The column "rating" is the outcome we want to predict, y. Taking into account both dataset, edx dataset is used to predict while validation dataset is used for validation below.

A. Data Exploration

```r
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1                 Comedy|Romance
## 2          Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5        Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7        Children|Comedy|Fantasy
```

```r
class(edx)
```

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

```r
glimpse(edx)
```

```
## Rows: 9,000,055
## 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, 292, 316, 329, 355, 356, 362, 364, 370, 377, 42...
## $ 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, 838983421, 838983392, 838983392, 83...
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)",...
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|...
```

```r
dim(edx)
```

```
## [1] 9000055       6
```

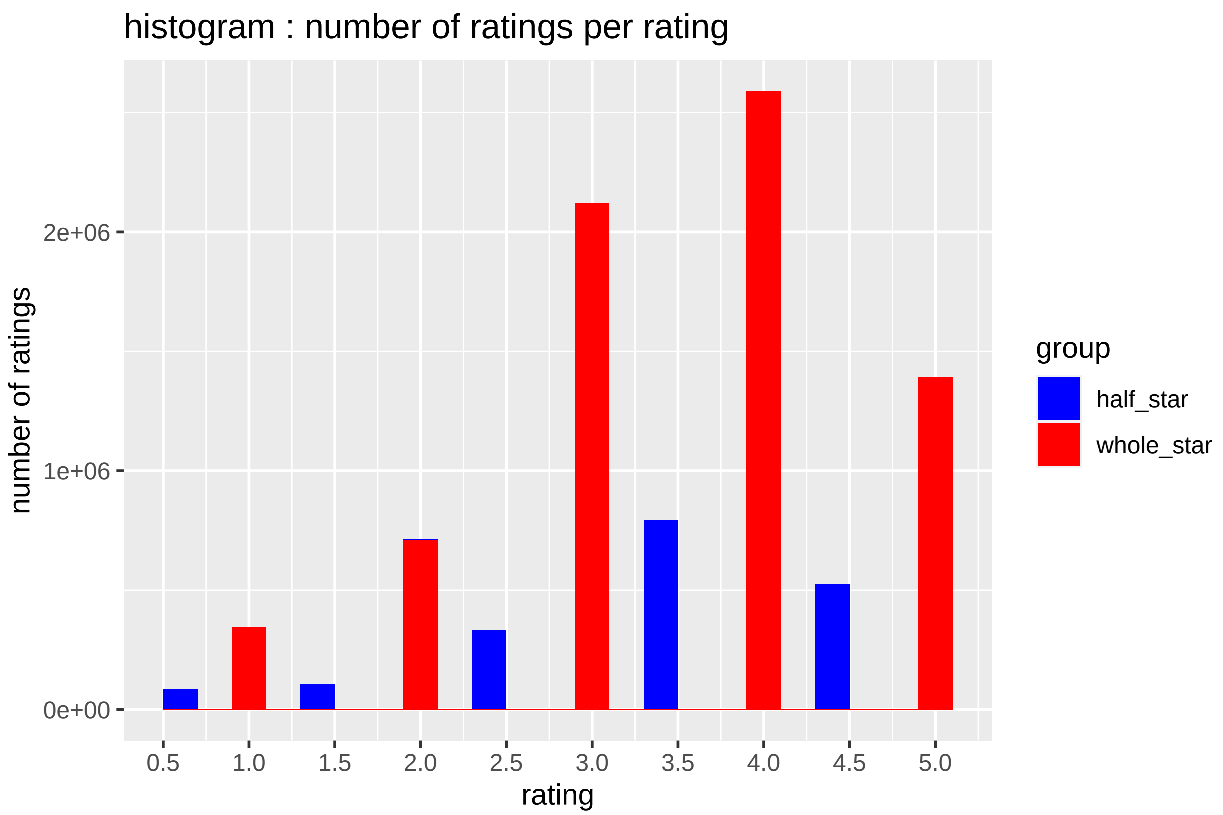
```r
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```r
#create a dataframe "explore_edx_ratings" which contains half star and whole star ratings  from the edx
group <-  ifelse((edx$rating == 1 |edx$rating == 2 | edx$rating == 3 |
                  edx$rating == 4 | edx$rating == 5) ,
                   "whole_star",
                   "half_star")
```

```r
explore_edx_ratings <- data.frame(edx$rating, group)
```

```r
#Histogram
ggplot(explore_edx_ratings, aes(x= edx.rating, fill = group)) +
  geom_histogram( binwidth = 0.2) +
  scale_x_continuous(breaks=seq(0, 5, by= 0.5)) +
  scale_fill_manual(values = c("half_star"="blue", "whole_star"="red")) +
  labs(x="rating", y="number of ratings", caption = "source data: edx set") +
  ggtitle("histogram : number of ratings per rating")
```



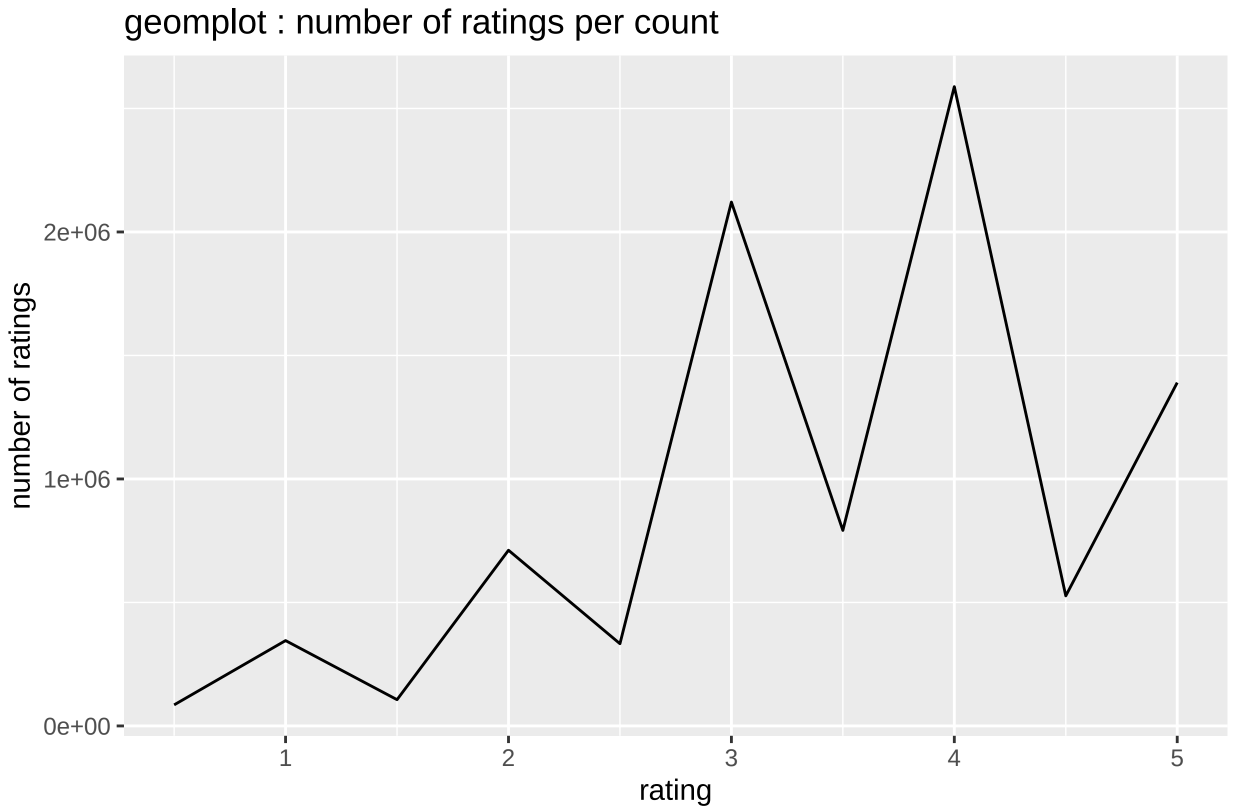
source data: edx set

Exploring ratings of the edx dataset, it is observed that the average user's activity reveals that no user gives 0 as rating, the top 5 ratings from most to least are : 4, 3, 5, 3.5 and 2. The histogram shows that the half star ratings are less common than whole star ratings.

```
#summary of five rating count
edx %>% group_by(rating) %>% summarize(count = n()) %>% top_n(5, count) %>%
    arrange(desc(count))
```

```
## # A tibble: 5 x 2
##   rating   count
##    <dbl>   <int>
## 1    4   2588430
## 2    3   2121240
## 3    5   1390114
## 4    3.5  791624
## 5    2    711422
```

```
#geom_line showing rating
edx %>%
    group_by(rating) %>%
    summarize(count = n()) %>%
    ggplot(aes(x = rating, y = count)) +
    geom_line()+
  labs(x="rating", y="number of ratings", caption = "source data: edx set") +
  ggtitle("geomplot : number of ratings per count")
```



source data: edx set

The result above shows that the rating was rated highest at point 4 and 5.

Exploration of the features contained in “genres” and “title” of our edx dataset.

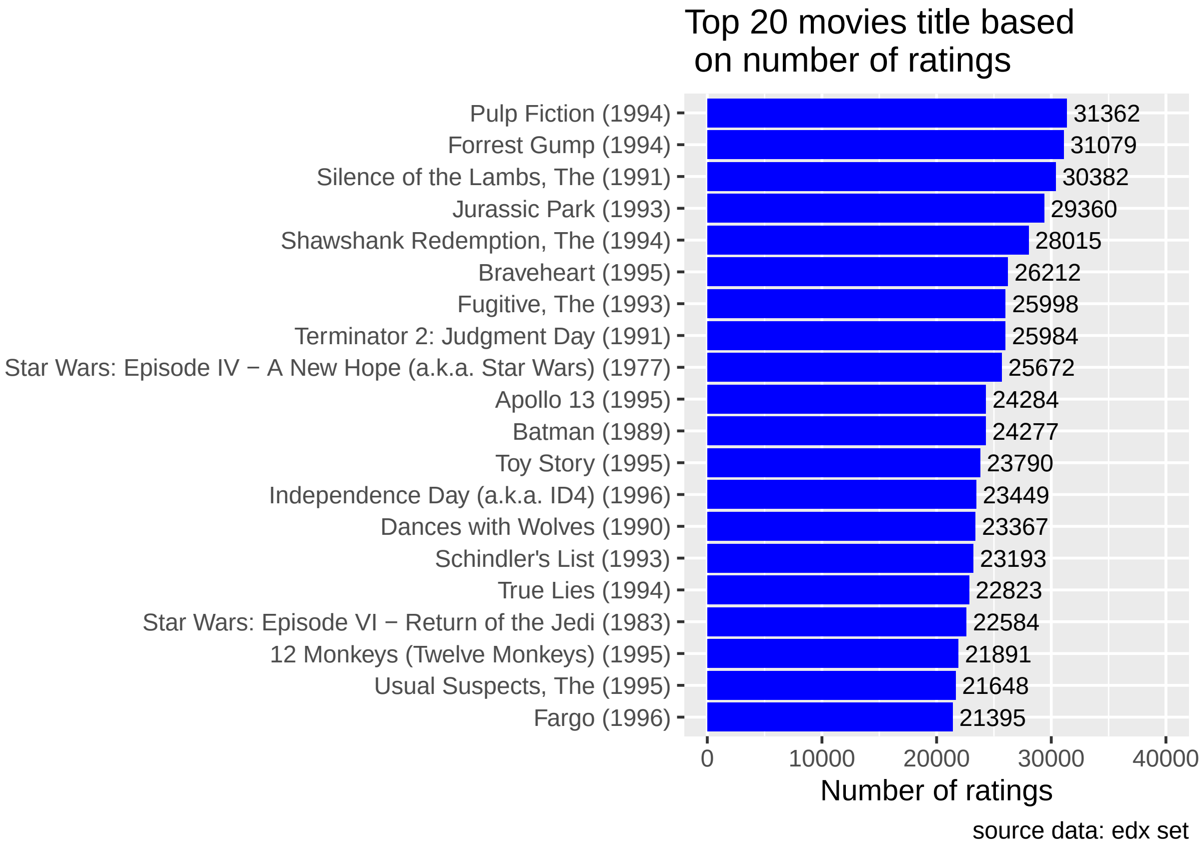
```
#top five title and genres
 edx %>%
  group_by(title, genres) %>%
  summarize(count=n()) %>%
  top_n(5,count) %>%
  arrange(desc(count))
```

```
## # A tibble: 10,677 x 3
## # Groups:   title [10,676]
##    title                                             genres                    count
##    <chr>                                             <chr>                     <int>
##  1 Pulp Fiction (1994)                               Comedy|Crime|Drama        31362
##  2 Forrest Gump (1994)                               Comedy|Drama|Romance|War  31079
##  3 Silence of the Lambs, The (1991)                  Crime|Horror|Thriller     30382
##  4 Jurassic Park (1993)                              Action|Adventure|Sci-Fi~  29360
##  5 Shawshank Redemption, The (1994)                  Drama                     28015
##  6 Braveheart (1995)                                 Action|Drama|War          26212
##  7 Fugitive, The (1993)                              Thriller                  25998
##  8 Terminator 2: Judgment Day (1991)                 Action|Sci-Fi             25984
##  9 Star Wars: Episode IV - A New Hope (a.k.a. St~    Action|Adventure|Sci-Fi   25672
## 10 Apollo 13 (1995)                                  Adventure|Drama           24284
## # ... with 10,667 more rows
```

```
# the data frame top_title contains the top 20 movies which count the major number of ratings
top_title <- edx %>%
  group_by(title) %>%
  summarize(count=n()) %>%
  top_n(20,count) %>%
  arrange(desc(count))

# with the head function i output the top 5

kable(head(edx %>%
     group_by(title,genres) %>%
     summarize(count=n()) %>%
     top_n(20,count) %>%
     arrange(desc(count)) ,
     5)) %>%
  kable_styling(bootstrap_options = "bordered", full_width = F , position ="center") %>%
  column_spec(1,bold = T ) %>%
  column_spec(2,bold =T) %>%
  column_spec(3,bold=T)
```

title	genres	count
Pulp Fiction (1994)	**Comedy|Crime|Drama**	**31362**
Forrest Gump (1994)	**Comedy|Drama|Romance|War**	**31079**
Silence of the Lambs, The (1991)	**Crime|Horror|Thriller**	**30382**
Jurassic Park (1993)	**Action|Adventure|Sci-Fi|Thriller**	**29360**
Shawshank Redemption, The (1994)	**Drama**	**28015**

```
#bar chart of top_title
top_title %>%
  ggplot(aes(x=reorder(title, count), y=count)) +
  geom_bar(stat='identity', fill="blue") + coord_flip(y=c(0, 40000)) +
  labs(x="", y="Number of ratings") +
  geom_text(aes(label= count), hjust=-0.1, size=3) +
  labs(title="Top 20 movies title based \n on number of ratings" , caption = "source data: edx set")
```


Top 20 movies title based on number of ratings

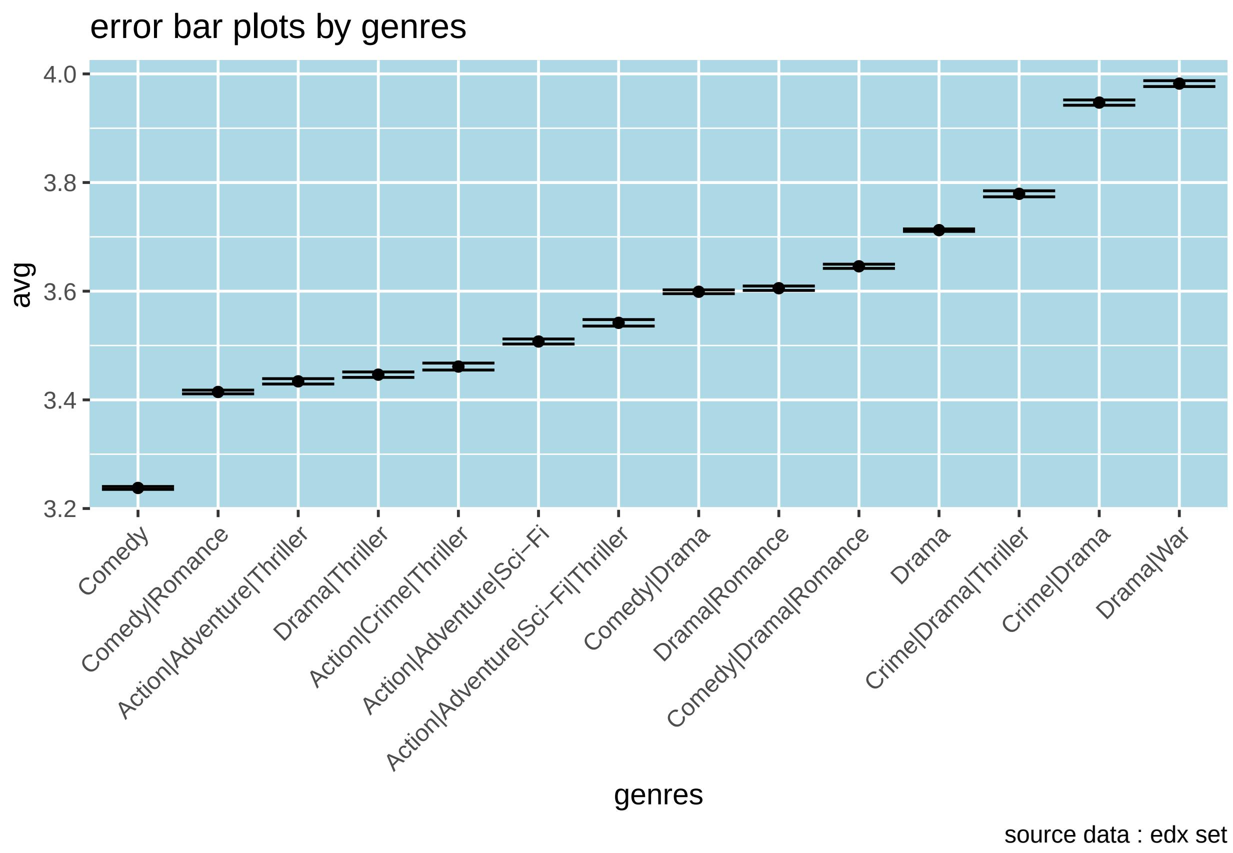
source data: edx set

From the “title” attributes it is observed that it is in line with the previous analysis. The movies which have the highest number of ratings are in the top genres categories : movies like Pulp fiction (1994), Forrest Gump(1994) or Jurrasic Park(1993) which are in the top 5 of movieâ€™s ratings number , are part of the Drama, Comedy or Action genres.

```
edx %>%
group_by(movieId) %>%
summarize(count = n()) %>%
filter(count == 1) %>%
left_join(edx, by = "movieId") %>%
group_by(title) %>%
summarize(rating = rating, n_rating = count) %>%
slice(1:20) %>%
knitr::kable()
```

title	rating	n_rating
1, 2, 3, Sun (Un, deuz, trois, soleil) (1993)	2.0	1
100 Feet (2008)	2.0	1
4 (2005)	2.5	1
Accused (Anklaget) (2005)	0.5	1
Ace of Hearts (2008)	2.0	1
Ace of Hearts, The (1921)	3.5	1
Adios, Sabata (Indio Black, sai che ti dico: Sei un gran figlio di...) (1971)	1.5	1
Africa addio (1966)	3.0	1
Aleksandra (2007)	3.0	1
Bad Blood (Mauvais sang) (1986)	4.5	1
Battle of Russia, The (Why We Fight, 5) (1943)	3.5	1
Bellissima (1951)	4.0	1
Big Fella (1937)	3.0	1
Black Tights (1-2-3-4 ou Les Collants noirs) (1960)	3.0	1
Blind Shaft (Mang jing) (2003)	2.5	1
Blue Light, The (Das Blaue Licht) (1932)	5.0	1
Borderline (1950)	3.0	1
Brothers of the Head (2005)	2.5	1
Chapayev (1934)	1.5	1
Cold Sweat (De la part des copains) (1970)	2.5	1

```
#An error bar plots for genres with more than 100000 ratings
edx %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 100000) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = "error bar plots by genres" , caption = "source data : edx set") +
  theme(
    panel.background = element_rect(fill = "lightblue",
                                    colour = "lightblue",
                                    size = 0.5, linetype = "solid"),
    panel.grid.major = element_line(size = 0.5, linetype = 'solid',
                                    colour = "white"),
    panel.grid.minor = element_line(size = 0.25, linetype = 'solid',
                                    colour = "white")
  )
```

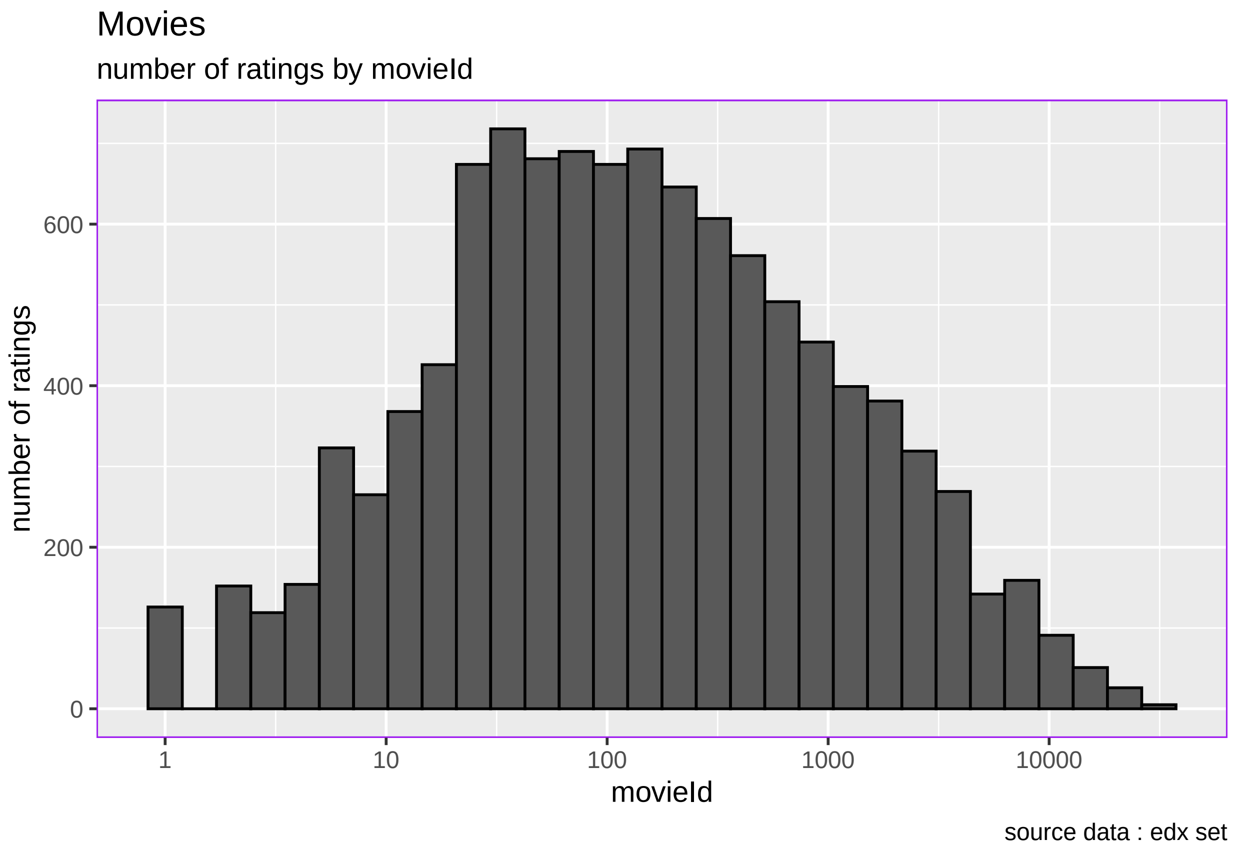


```
#unique userId and movieId
edx %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

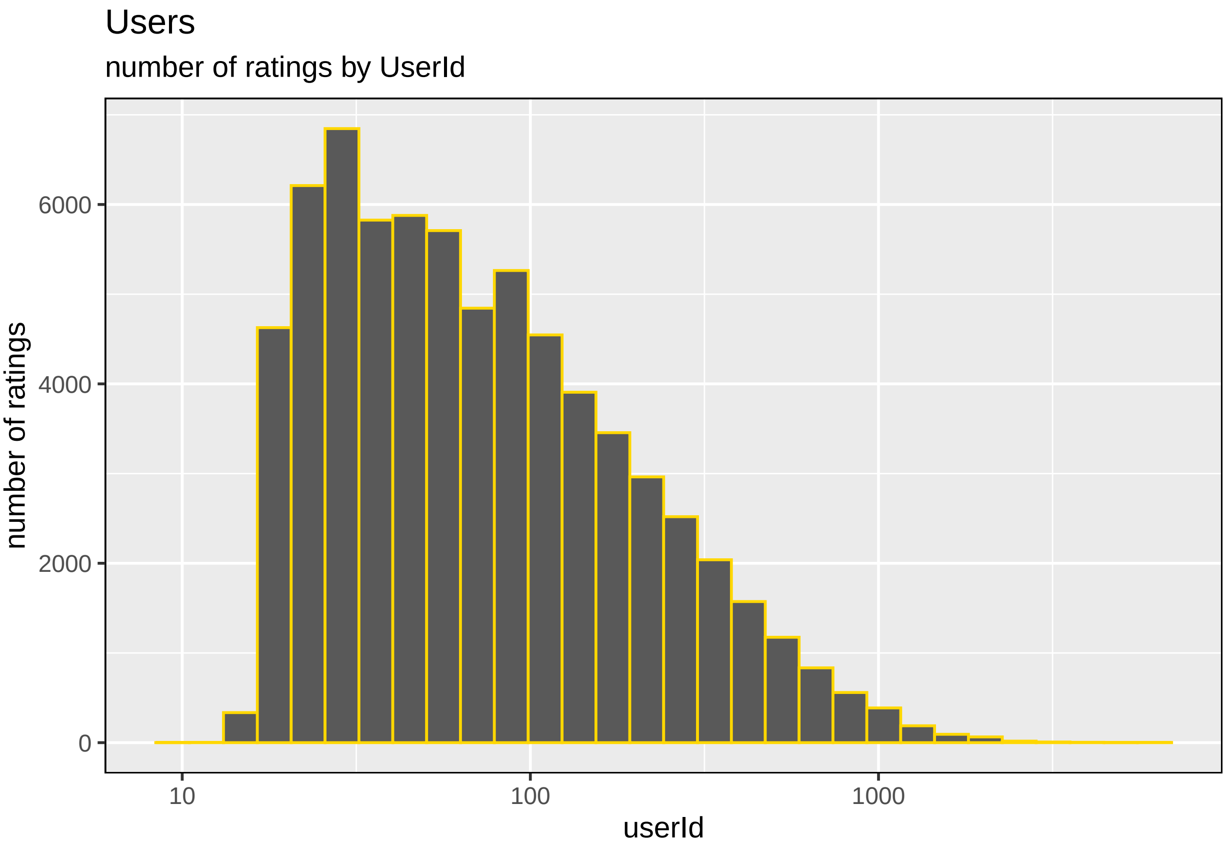
```
##   n_users n_movies
## 1   69878    10677
```

It is assumed that each row represents a rating given by one user to one movie, the number of unique values for the userId is 69878 and for the movieId 10677 : Both usersId and movieId which are presented as integer should be presumably treated as factors for analysis purposes. This invariably means that there are less movies provided for ratings than users that rated them . If we think in terms of a large matrix, with user on the rows and movies on the columns, the challenge would be the sparsity of the matrix. This large matrix will contain many empty cells. This would further present a problem of dimensionality.These issues would be treated in further analysis.

```
# histogram of number of ratings by movieId
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram( bins=30, color = "black") +
  scale_x_log10() +
  ggtitle("Movies") +
  labs(subtitle  ="number of ratings by movieId",
       x="movieId" ,
       y="number of ratings",
       caption ="source data : edx set") +
  theme(panel.border = element_rect(colour="purple", fill=NA))
```



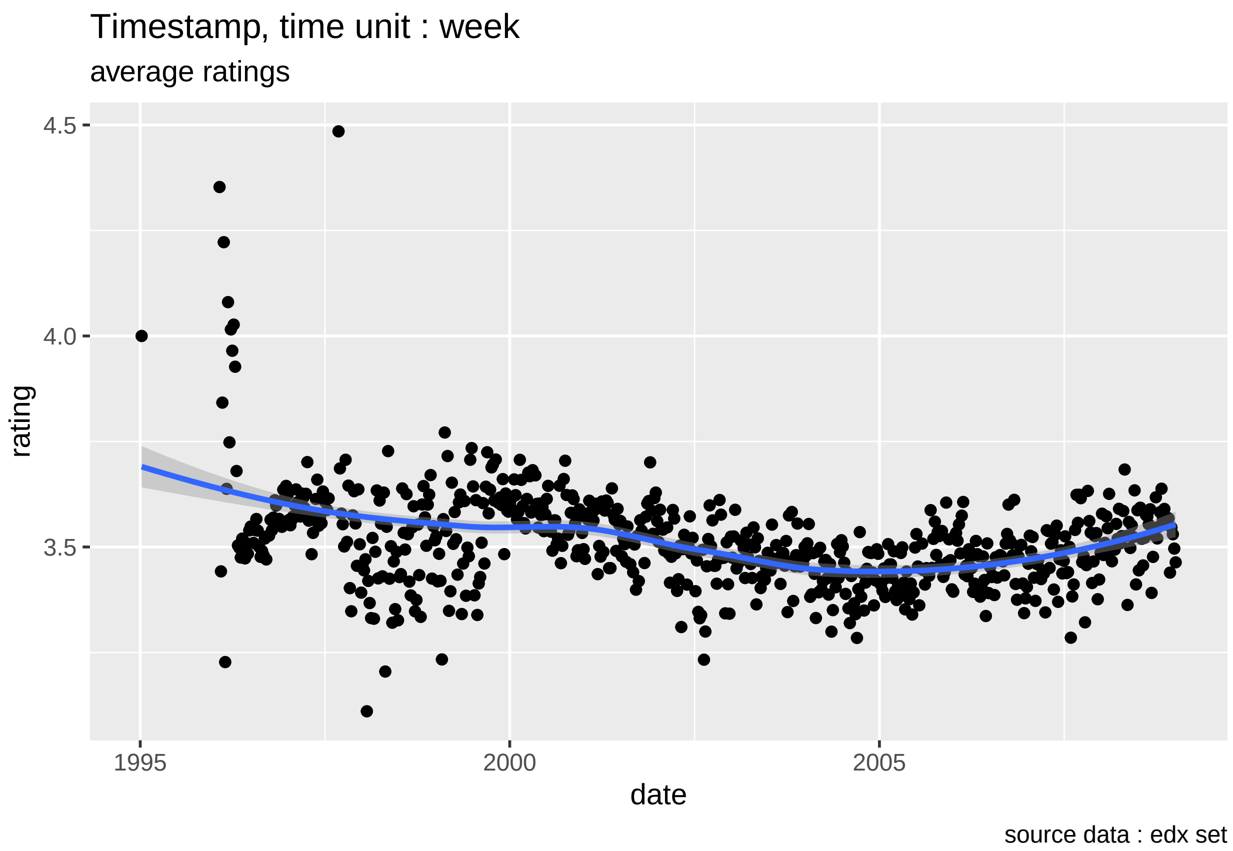
```
# histogram of number of ratings by userId
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram( bins=30, color = "gold") +
  scale_x_log10() +
  ggtitle("Users") +
  labs(subtitle ="number of ratings by UserId",
       x="userId" ,
       y="number of ratings") +
  theme(panel.border = element_rect(colour="black", fill=NA))
```



Visual exploration of the number of ratings by movieId and by userId shows the following relationships : some movies get rated more than others, and some users are more active than others at rating movies. This explains the presence of movies and users effect.

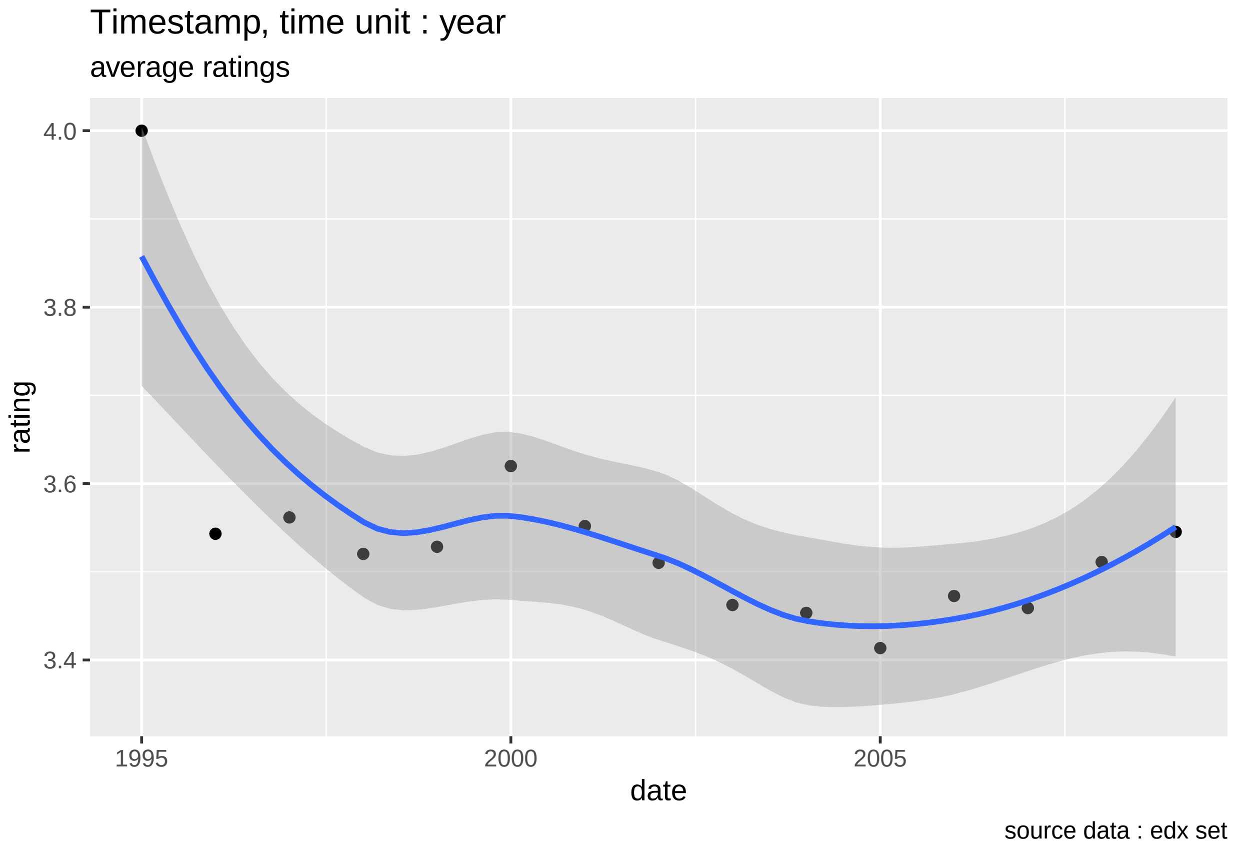
```
#ggplot showing timestamp per week
edx %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Timestamp, time unit : week")+
  labs(subtitle = "average ratings",
       caption = "source data : edx set")
```

```
## ‘geom_smooth()‘ using method = ’loess’ and formula ’y ~ x’
```



```
#ggplot showing timestamp per year
edx %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "year")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Timestamp, time unit : year")+
  labs(subtitle = "average ratings",
       caption = "source data : edx set")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Analyzing the trend of the average ratings versus date, shows that time has a weak effect on the average ratings as presented in the scatter plot.

B.Data Preprocessing

There is need for the processing of real-life data example includes cleansing, filtering, transformation in order to be used for the machine learning algorithm. This section mainly focuses on data preprocessing techniques that are of particular importance when designing a Recommender system. These techniques include similarity measures (such as Euclidean distance, Cosine distance,etc) , sampling methods , and dimensionality reduction (such as PCA or SVD). It has earlier been highlighted in the Data exploration step the problem of sparsity when considering a large matrix with users on the rows and movies on the columns, hence the need to build an effective matrix.

C. Data transformation

Trying to build our matrix, we somehow encountered that with the huge amount of data we have, the dcast , acast functions of the reshape2 and data.table packages are very time consuming and donâ€™t allocate vectors of size more than 2.8G. Then, we decided to go further with the Matrix packages : Matrix and Matrix.utils which contain the sparseMatrix function. The latter is less time consuming and deal more efficiently with the memory problem.

```
#Edx dataset transformation:usersId and movieId should be treat as factors for some analysis purposes.
edx.copy <- edx
edx.copy$userId <- as.factor(edx.copy$userId)
edx.copy$movieId <- as.factor(edx.copy$movieId)
```

```
#SparseMatrix function is used in order to get an output Of sparse matrix of class dgcMatrix.
# To use this function, the userId & movieId are converted to numeric vectors.
```

```
edx.copy$userId <- as.numeric(edx.copy$userId)
edx.copy$movieId <- as.numeric(edx.copy$movieId)

sparse_ratings <- sparseMatrix(i = edx.copy$userId,
                               j = edx.copy$movieId ,
                               x = edx.copy$rating,
                               dims = c(length(unique(edx.copy$userId)),
                                        length(unique(edx.copy$movieId))),
                               dimnames = list(paste("u", 1:length(unique(edx.copy$userId)), sep = ""),
                                               paste("m", 1:length(unique(edx.copy$movieId)), sep = "")))
```

```
#Remove the copy created

rm(edx.copy)
```

```
#The first 10 users
sparse_ratings[1:10,1:10]
```

```
## 10 x 10 sparse Matrix of class "dgCMatrix"

##    [[ suppressing 10 column names ’m1’, ’m2’, ’m3’ ... ]]

##
## u1  . .   . . . . . . . .
## u2  . .   . . . . . . . .
## u3  . .   . . . . . . . .
## u4  . .   . . . . . . . .
## u5  1 .   . . . . 3 . . .
## u6  . .   . . . . . . . .
## u7  . .   . . . . . . . .
## u8  . 2.5 . . 3 4 . . . .
## u9  . .   . . . . . . . .
## u10 . .   . . . . 3 . . .
```

```
##Convert rating matrix into a recommenderlab sparse matrix
rate_Mat <- new("realRatingMatrix", data = sparse_ratings)
rate_Mat
```

```
## 69878 x 10677 rating matrix of class ’realRatingMatrix’ with 9000055 ratings.
```

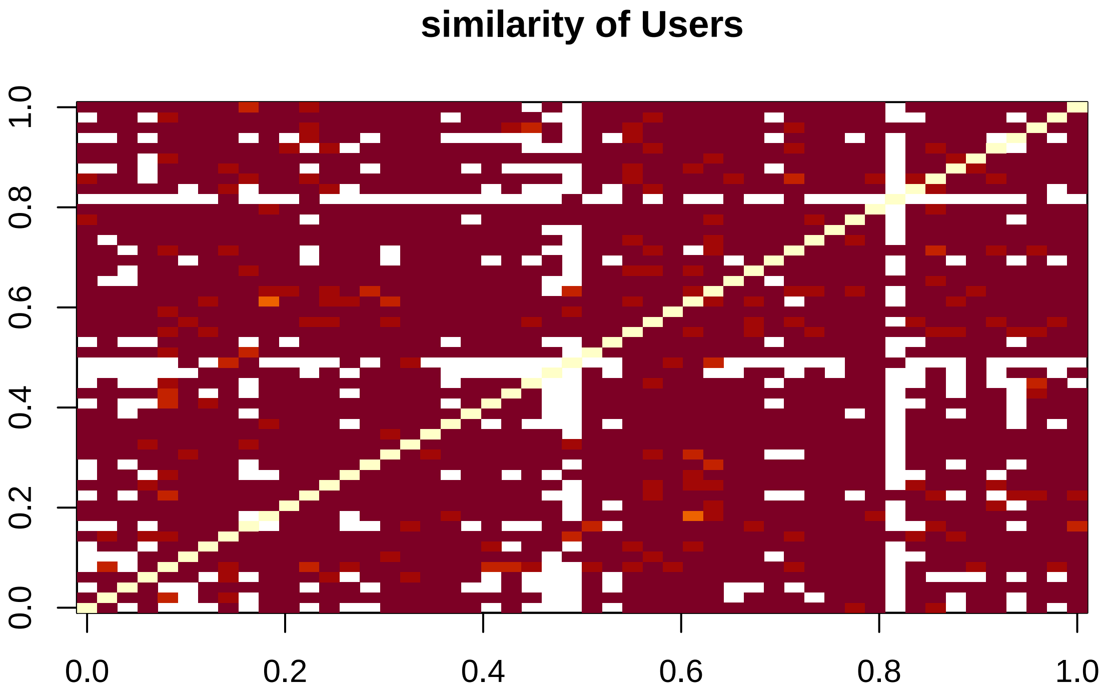
D. Similarity measures

Data mining techniques are used for the modelling of different recommender systems approaches(collaborative filtering, content based, hybrid methods) are highly dependent on defining an appropriate similarity or distance measure. To measure the similarity between users or between items, the following measures are used Minkowski Distance, Mahalanobis distance, Pearson correlation and Cosine similarity. The cosine similarity which is a measure of similarity between two non-zero vectors of an inner product space which measures the cosine of the angle between them. The main advantages of using this distance measure, as reported by Saranya et al (2016) includes: 1. Solves the problem of sparsity, scalability and cold start and it is more robust to noise. 2. It improves prediction accuracy and consistency 3. The Cosine similarity can still be calculated even though the matrix has many missing elements. 4. As the dimensional space becomes large, it still works well with low Computational complexity , especially for sparse vectors.

As a result of the large nature of the data, similarity on the first 50 users are calculated for visualization.

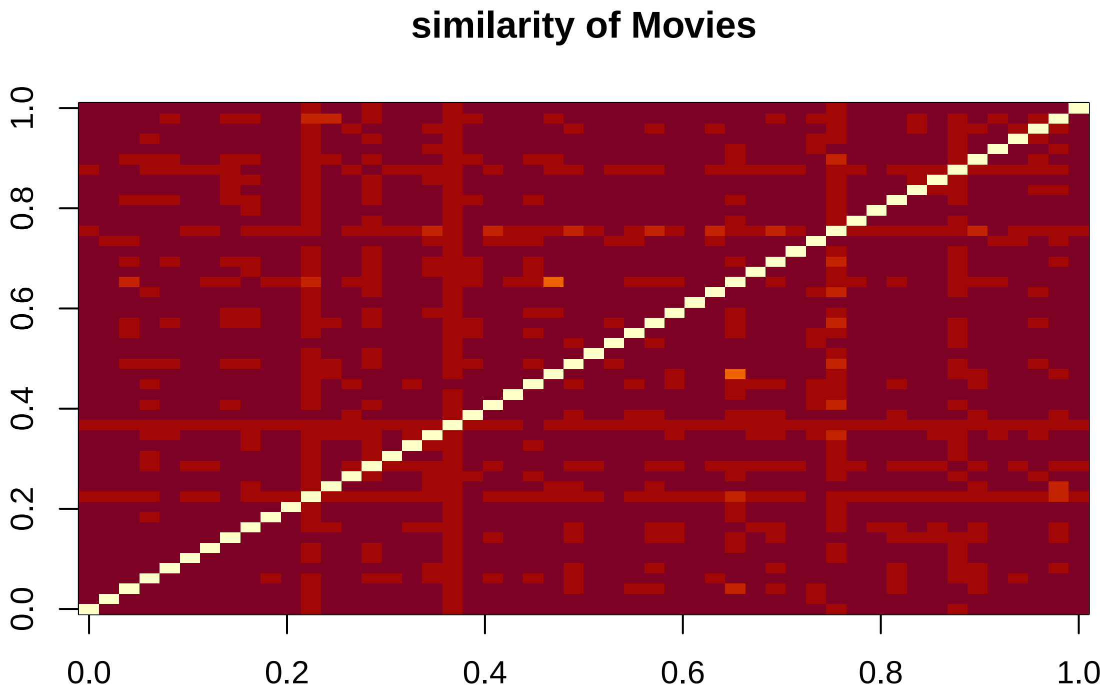
```
#calculate the user similarity using the cosine similarity
similarity_users <- similarity(rate_Mat[1:50,],
                               method = "cosine",
                               which = "users")
```

```
image(as.matrix(similarity_users), main = "similarity of Users")
```



```
#Using the same approach, compute similarity between  movies.

similarity_movies <- similarity(rate_Mat[,1:50],
                               method = "cosine",
                               which = "items")
```

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



In the given matrices above, each row and column corresponds to a user, and each cell corresponds to the similarity between two users . The more red the cell is, the more similar two users are. Note that the diagonal is red, since itâ€™s comparing each user with itself. From the observation of the two similarity matrices, the following inferences are drawn; there are more similar ratings between certain users and movies than others, this can therefore be an evidence that the existence of a group of users pattern or a group of movies pattern exist.

E. Dimension Reduction

The inference from the analysis of the previous similarity matrices leads to the thought that the possibility of users and movies with similar rating patterns. However Sparsity and the curse of dimensionality remain a recurent problem, and we have to deal with many NA too. Dimensionality reduction techniques such as “pca” and “svd” can help overcome these problem by transforming the original high-dimensional space into a lower-dimensionality. To face the RAM memory problem, we are going to use the Irlba package, which it is a fast and memory-efficient way to compute a partial SVD. The augmented implicitly restarted Lanczos bidiagonalization algorithm (IRLBA) finds a few approximate largest (or, optionally, smallest) singular values and corresponding singular vectors of a sparse or dense matrix using a method of Baglama and Reichel, which is further substantiated by Irizarry, 2018, on matrix factorization.

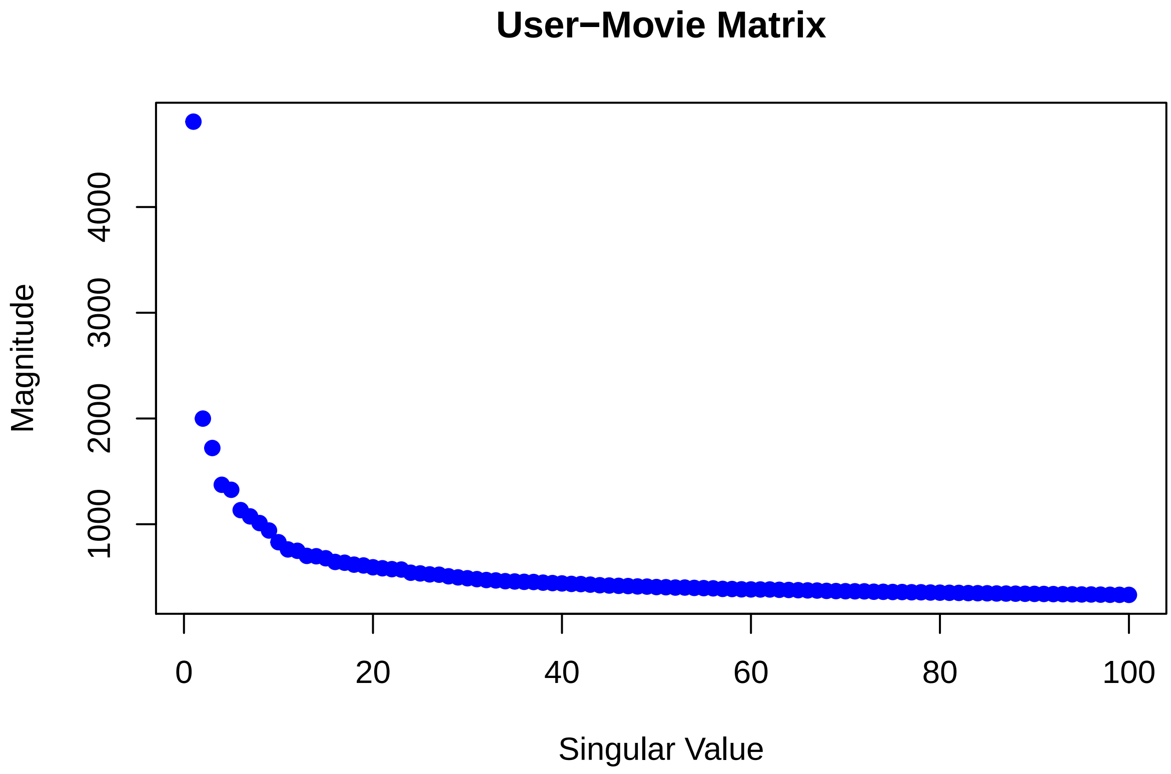
```
#implicitly restarted Lanczos bidiagonalization algorithm (IRLBA)
set.seed(1, sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
Y <- irlba(sparse_ratings,tol=1e-4,verbose=TRUE,nv = 100, maxit = 1000)
```

```
## Working dimension size 107

## Initializing starting vector v with samples from standard normal distribution.
## Use `set.seed` first for reproducibility.

## irlba: using fast C implementation
```

```
# plot singular values
plot(Y$d, pch=20, col = "blue", cex = 1.5, xlab='Singular Value', ylab='Magnitude',
     main = "User-Movie Matrix")
```



```
# calculate sum of squares of all singular values
all_sing_val <- sum(Y$d^2)
```

```
# variability described by first 6, 12, and 20 singular values
first_six <- sum(Y$d[1:6]^2)
print(first_six/all_sing_val)
```

```
## [1] 0.6187623
```

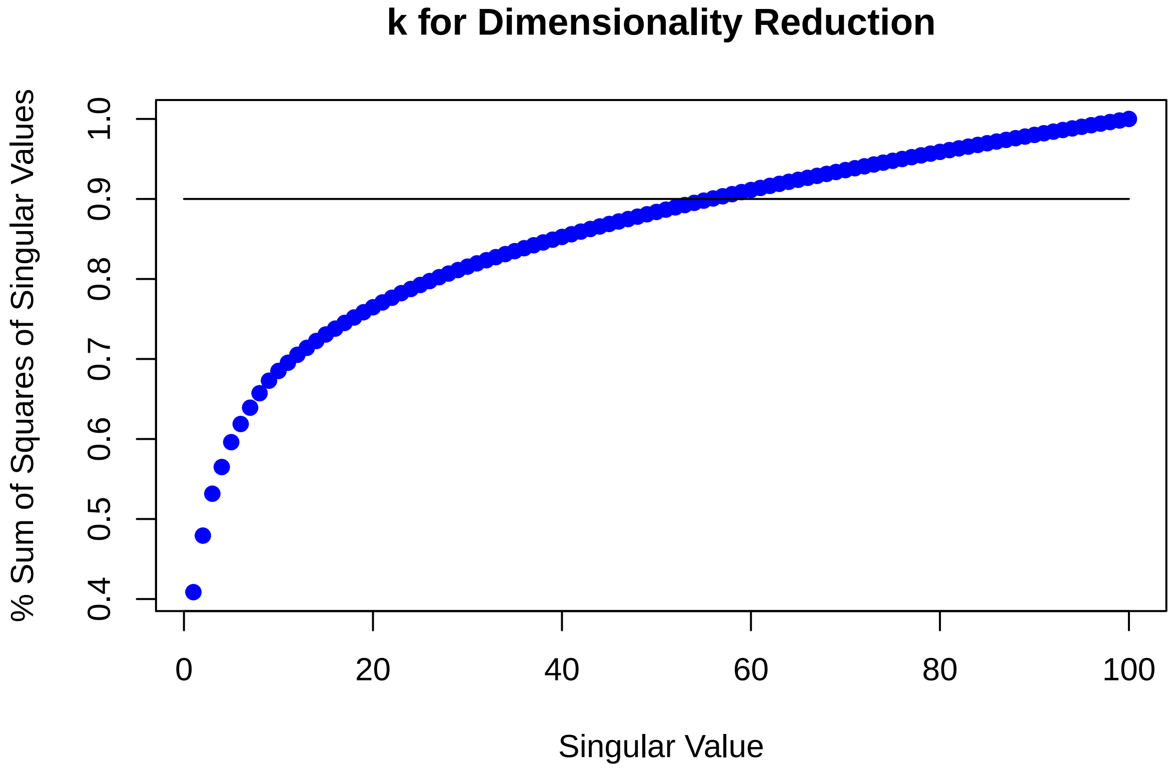
```
first_twl <- sum(Y$d[1:12]^2)
print(first_twl/all_sing_val)
```

```
## [1] 0.7052297
```

```r
first_twt <- sum(Y$d[1:20]^2)
print(first_twt/all_sing_val)
```

```
## [1] 0.7646435
```

```r
perc_vec <- NULL
for (i in 1:length(Y$d)) {
  perc_vec[i] <- sum(Y$d[1:i]^2) / all_sing_val
}

plot(perc_vec, pch=20, col = "blue", cex = 1.5, xlab='Singular Value', ylab='% Sum of Squares of Singula
lines(x = c(0,100), y = c(.90, .90))
```



First six singular values explain more than half of the variability of the imputed ratings matrix, with the first dozen explaining nearly 70% and the first twenty explaining more than 75%. However,the goal is to identify the first k singular values whose squares sum to at least 90% of the total of the sums of the squares of all of the singular values. A plot of a running sum of squares for the singular values shows that the 90% hurdle is achieved using somewhere between 50 and 60 singular values.

```r
#value of K
k = length(perc_vec[perc_vec <= .90])
k
```

```
## [1] 55
```

```
#get the decomposition of Y ; matrices U, D, and V
U_k <- Y$u[, 1:k]
dim(U_k)
```

```
## [1] 69878    55
```

```
D_k <- Diagonal(x = Y$d[1:k])
dim(D_k)
```

```
## [1] 55 55
```

```
V_k <- t(Y$v)[1:k, ]
dim(V_k)
```

```
## [1]    55 10677
```

It is noticed that k=55 will retain 90% of variability. Therefore, the total number of numeric values required to house these component matrices is (69878*55)+(55*55)+(55*10677)=4,433,550 (69878*55)+(55*55)+(55*10677)=4,433,550. This represents an approximately 50.7% decrease in required storage relative to the original 9,000,055 entries. Reducing the dimensionality, the RAM memory problem persists. Thats why we needed to go further with another reduction technique. We selected the relevant data using the whole rating matrix.

F. Relevant Data

It is important reiterate that some users saw more movies than others. So, instead of displaying some random users and movies, instead an intentional approach to select the most relevant users and movies is put in place, this therefore aids in the visualization of only the users who have seen many movies and the movies that have been seen by many users.To identify and select the most relevant users and movies, the following steps are adopted: 1. Determine the minimum number of movies per user. 2. Determine the minimum number of users per movie. 3. Select the users and movies matching these criteria.

```
#1. Determine the minimum number of movies per user.
min_no_movies <- quantile(rowCounts(rate_Mat), 0.9)
print(min_no_movies)
```

```
## 90%
## 301
```

```
#2. Determine the minimum number of users per movie.
min_no_users <- quantile(colCounts(rate_Mat), 0.9)
print(min_no_users)
```

```
##    90%
## 2150.2
```

```
#3. Select the users and movies matching these criteria.
rate_movies <- rate_Mat[rowCounts(rate_Mat) > min_no_movies,
                             colCounts(rate_Mat) > min_no_users]
rate_movies
```

```
## 6978 x 1068 rating matrix of class 'realRatingMatrix' with 2313148 ratings.
```

From the analysis so far, the rating matrix obtained consist of 6978 distinct users in rows x 1068 distinct movies in columns, with 2,313,148 ratings . The data preprocessing phase is usually not definitive because it requires a lot of attention and subsequently, various explorations on the variables. It must be aimed at obtaining better predictive results and in this sense, the further phases of model evaluations can help us to understand which particular preprocessing approaches are actually indispensable or useful for a specific model purpose.

```
#define the RMSE function
RMSE <- function(true_ratings, predicted_ratings){
    sqrt(mean((true_ratings - predicted_ratings)^2))
  }
```

Movie effect

```
#calculate the average of all ratings of the edx dataset
mu <- mean(edx$rating)
```

```
#calculate b_i on the training dataset
movie_m <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

```
# predicted ratings
predicted_ratings_bi <- mu + validation %>%
  left_join(movie_m, by='movieId') %>%
  .$b_i
```

Movie and user effect

```
#calculate b_u using the training set
user_m <- edx %>%
  left_join(movie_m, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

```
#predicted ratings
predicted_ratings_bu <- validation %>%
  left_join(movie_m, by='movieId') %>%
  left_join(user_m, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
```

Movie, user and time effect

```
#create a copy of validation set , valid, and create the date feature which is the timestamp converted
valid <- validation
valid <- valid %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))
```

```
#calculate time effects ( b_t) using the training set
temp_m <- edx %>%
  left_join(movie_m, by='movieId') %>%
  left_join(user_m, by='userId') %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(b_t = mean(rating - mu - b_i - b_u))
```

```
#predicted ratings
predicted_ratings_bt <- valid %>%
  left_join(movie_m, by='movieId') %>%
  left_join(user_m, by='userId') %>%
  left_join(temp_m, by='date') %>%
  mutate(pred = mu + b_i + b_u + b_t) %>%
  .$pred
```

The root mean square error (RMSE) models for movies, users and time effects

```
#calculate the RMSE for movies
rmse_model_1 <- RMSE(validation$rating,predicted_ratings_bi)
rmse_model_1
```

```
## [1] 0.9439087
```

```
#calculate the RMSE for users
rmse_model_2 <- RMSE(validation$rating,predicted_ratings_bu)
rmse_model_2
```

```
## [1] 0.8653488
```

```
#calculate the RMSE for time effects
rmse_model_3 <- RMSE(valid$rating,predicted_ratings_bt)
rmse_model_3
```

```
## [1] 0.8652511
```

From the movie and user effects combined, our RMSE decreased by almost 10% with respect to the only movie effect. The improvement on the time effect is not significant,(about a decrease of 0.011%). The regularization would be performed using only the movie and user effects.

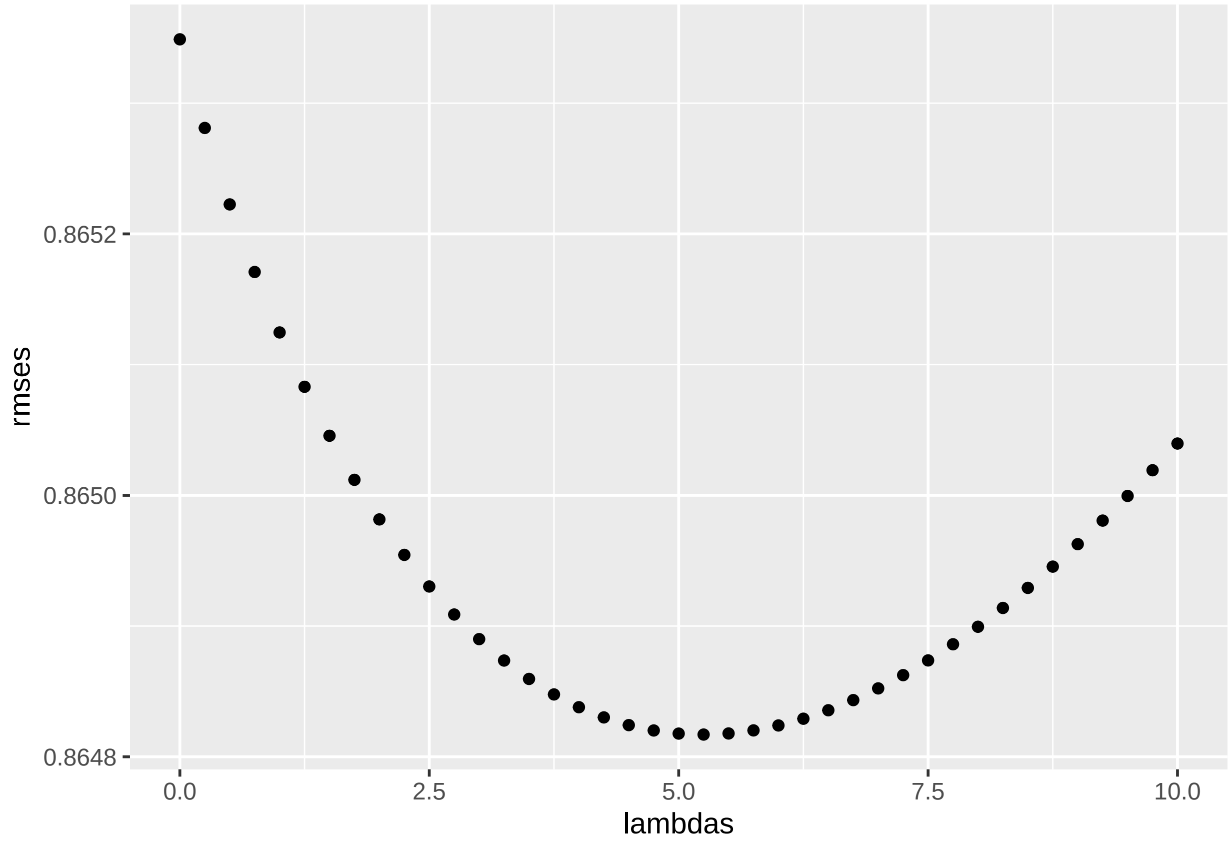
```
#remove valid before regularization
rm(valid)
```

G. Regularization

```
## remembering that lambda is a tuning parameter. We can use cross-validation to choose it
lambdas <- seq(0, 10, 0.25)
```

```
rmses <- sapply(lambdas, function(l){

    mu_reg <- mean(edx$rating)

    b_i_reg <- edx %>%
      group_by(movieId) %>%
      summarize(b_i_reg = sum(rating - mu_reg)/(n()+l))

    b_u_reg <- edx %>%
      left_join(b_i_reg, by="movieId") %>%
      group_by(userId) %>%
      summarize(b_u_reg = sum(rating - b_i_reg - mu_reg)/(n()+l))

    predicted_ratings_b_i_u <-
      validation %>%
      left_join(b_i_reg, by = "movieId") %>%
      left_join(b_u_reg, by = "userId") %>%
      mutate(pred = mu_reg + b_i_reg + b_u_reg) %>%
      .$pred

    return(RMSE(validation$rating,predicted_ratings_b_i_u))
  })
```

```
qplot(lambdas, rmses)
```



The optimal lambda for the full model

```r
#For the full model, the optimal  Î»  is given as
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 5.25
```

```r
rmse_model_4 <- min(rmses)
rmse_model_4
```

```
## [1] 0.864817
```

Summarry of the rmse on validation set for Linear regression models

```r
#summarize all the rmse on validation set for Linear regression models
rmse_results <- data.frame(methods=c("movie effect","movie + user effects","movie + user + time effects

kable(rmse_results) %>%
  kable_styling(bootstrap_options = "striped" , full_width = F , position = "center") %>%
  kable_styling(bootstrap_options = "bordered", full_width = F , position ="center") %>%
  column_spec(1,bold = T ) %>%
  column_spec(2,bold =T ,color = "white" , background ="#D7261E")
```

methods	rmse
movie effect	**0.9439087**
movie + user effects	**0.8653488**
movie + user + time effects	**0.8652511**
Regularized Movie + User Effect Model	**0.8648170**

The regularization gets down the RMSE's value to 0.8648170.

POPULAR , UBCF and IBCF algorithms of the recommenderlab package

```r
#POPULAR algorithms of the recommenderlab package
model_pop <- Recommender(rate_movies, method = "POPULAR",
                      param=list(normalize = "center"))
```

```r
#prediction example on the first 10 users
pred_pop <- predict(model_pop, rate_movies[1:10], type="ratings")
as(pred_pop, "matrix")[,1:10]
```

```
##            m1       m2       m3       m5       m6       m7       m9      m10
## u8   3.855564       NA 2.923282       NA       NA 3.092309 2.574400 3.314650
## u17        NA       NA 2.969625       NA 3.802000       NA 2.620742       NA
## u28  3.271469       NA       NA       NA 3.171562 2.508213       NA 2.730555
## u30        NA       NA       NA 2.792031       NA 3.109045 2.591136       NA
## u43  4.664153 3.756804 3.731871 3.583884 4.564246       NA 3.382989 4.123238
## u48        NA       NA       NA 3.448791 4.429153 3.765804 3.247895       NA
## u57        NA       NA 2.652900 2.504913 3.485275 2.821926 2.304018 3.044268
## u70  4.421321 3.513973 3.489039 3.341052 4.321415 3.658066 3.140157 3.880407
## u88        NA 3.038295 3.013362 2.865375 3.845737 3.182388 2.664480 3.404729
## u103       NA 2.777942 2.753008 2.605021       NA 2.922034 2.404126 3.144375
##           m11      m14
## u8   3.459774 3.416521
## u17  3.506116 3.462863
## u28  2.875678 2.832426
## u30        NA 3.433257
## u43        NA 4.225109
## u48  4.133269 4.090016
## u57        NA 3.146139
## u70  4.025531 3.982278
## u88  3.549853 3.506600
## u103 3.289499 3.246247
```

Rmse for popularity based recommender engine

```r
set.seed(1, sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
#Calculation of rmse for popular method
eval <- evaluationScheme(rate_movies, method="split", train=0.7, given=-5)
```

```r
#ratings of 30% of users are excluded for testing
model_pop <- Recommender(getData(eval, "train"), "POPULAR")

prediction_pop <- predict(model_pop, getData(eval, "known"), type="ratings")

rmse_pop <- calcPredictionAccuracy(prediction_pop, getData(eval, "unknown"))[1]
rmse_pop
```

```
##      RMSE
## 0.8482917
```

User based cosine factorization recommender engine

```r
#Estimating rmse for UBCF using Cosine similarity and selected n as 50 based on cross-validation
set.seed(1, sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
model <- Recommender(getData(eval, "train"), method = "UBCF",
                     param=list(normalize = "center", method="Cosine", nn=50))

prediction <- predict(model, getData(eval, "known"), type="ratings")

rmse_ubcf <- calcPredictionAccuracy(prediction, getData(eval, "unknown"))[1]
rmse_ubcf
```

```
##      RMSE
## 0.8589153
```

Item based Cosine factorization recommender engine

```
#Estimating rmse for IBCF using Cosine similarity and selected n as 350 based on cross-validation
set.seed(1, sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
model <- Recommender(getData(eval, "train"), method = "IBCF",
                     param=list(normalize = "center", method="Cosine", k=350))

prediction <- predict(model, getData(eval, "known"), type="ratings")

rmse_ibcf <- calcPredictionAccuracy(prediction, getData(eval, "unknown"))[1]
rmse_ibcf
```

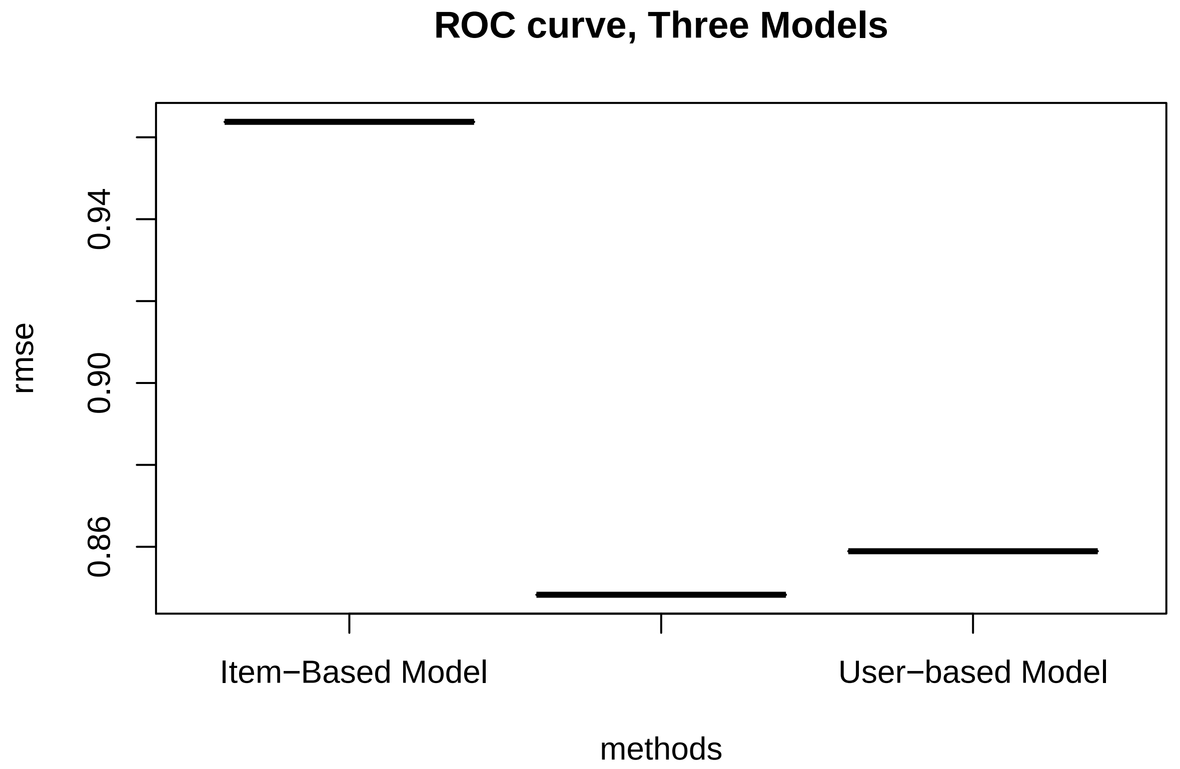
```
##     RMSE
## 0.963769
```

Rmse from popularity, user and item based recommender engine

```
rmse_crossvalidation <- data.frame(methods=c("Popularity-Based model","User-based Model"," Item-Based Mo

kable(rmse_crossvalidation) %>%
  kable_styling(bootstrap_options = "striped" , full_width = F , position = "center") %>%
  kable_styling(bootstrap_options = "bordered", full_width = F , position ="center") %>%
  column_spec(1,bold = T ) %>%
  column_spec(2,bold =T ,color = "white" , background ="#D7261E")
```

methods	rmse
Popularity-Based model	**0.8482917**
User-based Model	**0.8589153**
Item-Based Model	**0.9637690**

```
plot(rmse_crossvalidation, annotate = 1, legend = "topleft")
title("ROC curve, Three Models")
```



Root mean square error (RMSE) the standard deviation of the difference between the real and predicted ratings, the higher the error, the worse the model performs.From the result, of the cross validation model, the item based model is the worst and should not be reccommended, the popularity based model worked better with an rmse of 0.8482917. The best model recommended from this research from the validation set is the regularized Movie + User effect Model with the least root mean square of 0.8648170.

IV. Conclusion This MovieLens project has examined the potential best recommender system algorithm to predict movie ratings for the 10M version of the Movielens data. Using the provided training set (edx) and validation set, we successively trained different linear regression models and some recommender engines. The model evaluation performance through the RMSE (root mean squared error) showed that the Linear regression model with regularized effects on users and movie is an appropriate recommender systems to predict ratings on the validation set.

V. Recommendation The research strongly reccommend the following for further investigations using other models, such as essemble,and matrix factorization to get the latent root mean square error which was not possible due to hardware issues.

```
print("Operating System:")
```

```
## [1] "Operating System:"
```

```
version
```

```
##                  _
## platform         x86_64-w64-mingw32
## arch             x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          3
## minor          6.0
## year           2019
## month          04
## day            26
## svn rev        76424
## language       R
## version.string R version 3.6.0 (2019-04-26)
## nickname       Planting of a Tree
```