Harvardx - Data Science Capstone - Movielens Project

Willian Martins Dalboni

2022-08-08

1) INTRODUCTION

1.1) Objective

This document creates a prediction system using root mean square error as a guideline to achieve the maximum score for the Recommendation System Capstone of the Harvardx Data Science course using the MovieLens 10M dataset. The final model would succeed if the RMSE is under "0.8649" and the presented code in this document surpasses this goal by a fair amount, as you see follows.

1.2) Terminology

Another critical thing to consider is how this document is presented. This document was written to make it easier to understand by colleagues with the same level of experience the writer has at this point. This decision was driven by the fact that the exercise explicitly informs us there will be a peer review, and the terminology follows this idea.

2) ANALYSIS

2.1) PREPARING THE DATA

2.1.1) Installing the packages

We will need tools to achieve our goal. They need to be installed and loaded to be called when we need them. We verify if we already have them and install them if they are not present.

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

##Others
if(!require(stringr)) install.packages("stringr", repos = "http://cran.us.r-project.org")
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(recosystem)) install.packages("recosystem")
if(!require(knitr)) install.packages('knitr', dependencies = TRUE)
```

```r
if(!require(RColorBrewer)) install.packages("RColorBrewer")
if(!require(ggrepel)) install.packages("ggrepel")
tinytex::install_tinytex()

#Libraries
library(tidyverse)
library(caret)
library(data.table)
library(stringr)
library(lubridate)
library(ggplot2)
library(dplyr)
library(recosystem)
library(RColorBrewer)
library(ggrepel)
```

2.1.1) Creating the datasets

This code creates the edx set and the validation set. The edx set will be divided into a train set and a test set, and both will be used to create and test the code. After we are sure it is working correctly, we can compare it with the validation set to guarantee we accomplished the task.

When this code was written, the "MovieLens 10M" dataset could be downloaded through this link. Even if it changed places, the logic behind what has been done does not change.

```r
# Downloads the Movielens
dl <- tempfile()
download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

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

# Creates the data splitting when it finds a ::
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)

# Creates the columns names
colnames(movies) <- c("movieId", "title", "genres")

# Creates the dataframe movies already definning the column type of data
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")

# Ensures we have the same seed (and dataset) as everyone else
set.seed(1, sample.kind="Rounding")

# 10% of the whole set becomes the edx set
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

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

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

# Clean space
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

2.2) EXPLORATION & INSIGHTS

Observing the datasets is essential to get insights into what we got and what we can create to predict better.

2.2.1) Heads

The "head" command shows us the six rows from the top of each set to find if they have the same names, column order, and data types.

```
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                          genres
## 1:               Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

```
head(validation)
```

```
##    userId movieId rating timestamp
## 1:      1     231      5 838983392
## 2:      1     480      5 838983653
## 3:      1     586      5 838984068
## 4:      2     151      3 868246450
## 5:      2     858      2 868245645
## 6:      2    1544      3 868245920
##                                          title
## 1:                          Dumb & Dumber (1994)
## 2:                          Jurassic Park (1993)
```

```
## 3:                                       Home Alone (1990)
## 4:                                          Rob Roy (1995)
## 5:                                    Godfather, The (1972)
## 6: Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                      genres
## 1:                                   Comedy
## 2:        Action|Adventure|Sci-Fi|Thriller
## 3:                           Children|Comedy
## 4:                 Action|Drama|Romance|War
## 5:                               Crime|Drama
## 6: Action|Adventure|Horror|Sci-Fi|Thriller
```

2.2.2) Glimpse

Glimpse helps us pay attention to the column's dimensions and variable type.

```
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, 2, ~
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 420, ~
## $ rating    <dbl> 5, 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, 83898~
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)", "S~
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|Sci~
```

```
glimpse(validation)
```

```
## Rows: 999,999
## Columns: 6
## $ userId    <int> 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, ~
## $ movieId   <dbl> 231, 480, 586, 151, 858, 1544, 590, 4995, 34, 432, 434, 85, ~
## $ rating    <dbl> 5.0, 5.0, 5.0, 3.0, 2.0, 3.0, 3.5, 4.5, 5.0, 3.0, 3.0, 3.0, ~
## $ timestamp <int> 838983392, 838983653, 838984068, 868246450, 868245645, 86824~
## $ title     <chr> "Dumb & Dumber (1994)", "Jurassic Park (1993)", "Home Alone ~
## $ genres    <chr> "Comedy", "Action|Adventure|Sci-Fi|Thriller", "Children|Come~
```

2.2.3) Summary

The summary shows the Quartiles, the Minimum, and Maximum values. It is crucial since it can show outliers not apparent using other approaches.

```
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
##
##
##
```

```
summary(validation)
```

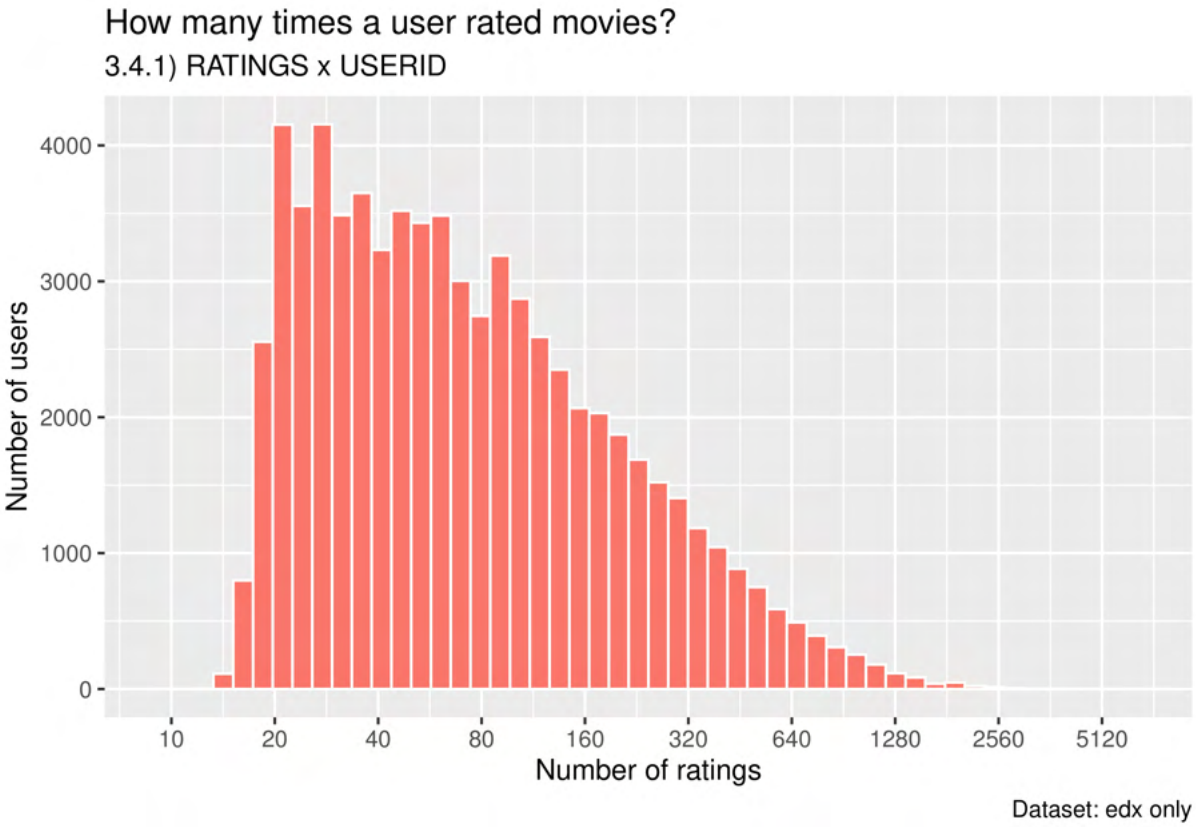
```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18096   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.467e+08
##  Median :35768   Median : 1827   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4108   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53621   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:999999      Length:999999
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

2.3) GRAPHS

2.3.1) Comparison: ratings x userid

```
#Creates a table containing how many times each userId repeats.
sorted_userId <- edx %>%
group_by(userId) %>%
count() %>%
arrange(n)

#Creates a histogram that divides the values between 50 bins.
ggplot (data = sorted_userId ) +
geom_histogram(mapping = aes(x=n, fill="gears", col=I("white")), bins=50) +
theme_gray() +
theme(legend.position="none") +
scale_x_log10(breaks=c(1,5,10,20,40,80,160,320,640,1280,2560,5120,10240,20480,40960)) +
labs(
    title = "How many times a user rated movies?",
    subtitle = "3.4.1) RATINGS x USERID",
    caption = "Dataset: edx only",
    x = "Number of ratings",
    y = "Number of users")
```

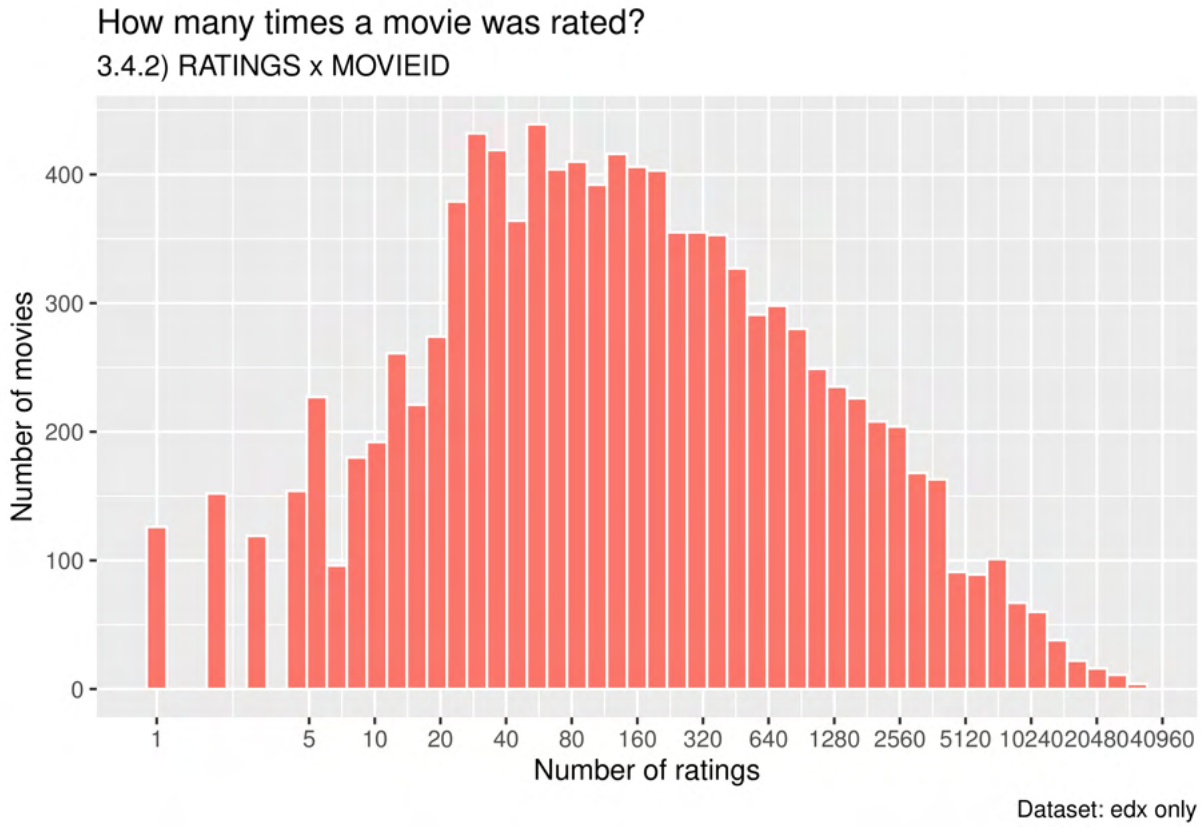


There is a histogram approach to verify whether users rated many movies or not. The graph shows that most users rate a low number of movies, but the data starts at ten ratings per user.

2.3.2) Comparison: ratings x movieid

```
#Creates a table containing how many times each movieId repeats.
sorted_movieId <- edx %>%
group_by(movieId) %>%
count() %>%
arrange(n)

#Creates a histogram that divides the values between 50 bins.
ggplot (data = sorted_movieId ) +
geom_histogram(mapping = aes(x=n, fill="gears", col=I("white")), bins=50) +
theme_gray() +
theme(legend.position="none") +
scale_x_log10(breaks=c(1,5,10,20,40,80,160,320,640,1280,2560,5120,10240,20480,40960)) +
labs(
    title = "How many times a movie was rated?",
    subtitle = "3.4.2) RATINGS x MOVIEID",
    caption = "Dataset: edx only",
    x = "Number of ratings",
    y = "Number of movies")
```



Some movies have almost no ratings, and some have many. It is vital to reduce the outliers' weight.

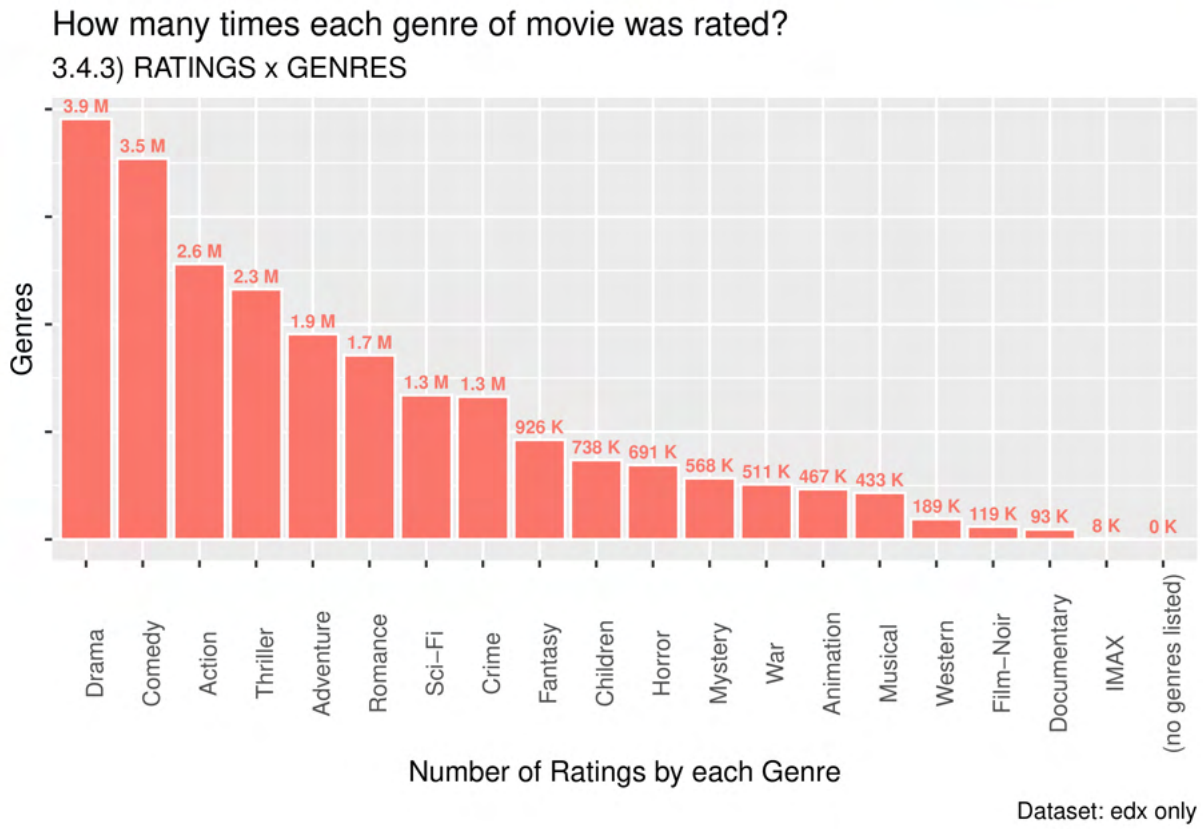
2.3.3) Comparison: ratings x genres

```
#Create a table that breaks the nested genres separated by | and counts
#how many times each genre repeats.
sorted_genres <- edx %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  count() %>%
  arrange(n)
```

```
#Prevents scientific notation.
options(scipen=999)

#Create labels using M or K to shorten the numbers,
#then creates sorted columns for each genre.
sorted_genres %>%
mutate(genres = factor(genres, levels = genres)) %>%
mutate(labels = ifelse(n>999999,paste(round(n/1000000,1),"M"),
                       paste(round(n/1000,0),"K"))) %>%
ggplot (mapping = aes(y=n, x=reorder(genres, -n), fill="gears", col=I("white"))) +
geom_bar(stat="identity") +
theme_gray() +
```

```
labs(
    title = "How many times each genre of movie was rated?",
    subtitle = "3.4.3) RATINGS x GENRES",
    caption = "Dataset: edx only",
    x = "Number of Ratings by each Genre",
    y = "Genres") +
theme(legend.position="none",
    axis.text.x = element_text(angle=90), axis.text.y = element_blank()) +
geom_text(aes(label = labels, fontface = "bold", colour="#F8766D"), vjust=-0.5, size=2.5)
```



There is a high discrepancy between movie genres. Regularization is needed.

2.4) CLEANING & MANIPULATING THE DATA

There is a need to create new columns, and it is more productive to do them now using the edx dataset before splitting the test and train dataset.

2.4.1) Release Year (rey)

It is noticeable that the title and the year of release are together, and it may be helpful to use them separately. The zeitgeist may impact the ratings, and it is preferred to use two extracts to assure the process was outputting what was idealized. This assures the finish, making the regex more readable and the years become detached from the title.

```
# Extracts the release year from the title
release_year <- edx$title %>%
str_extract(.,".{6}$") %>% #Extract the last 6 values
str_extract(.,"(?<=\\().+?(?=\\))") %>% #Remove parenthesis
as.integer() #Change it to a integer value

# Add the result as a column
edx <- as.data.frame(cbind(edx, release_year)) #Add the result as a new column
```

2.4.2) Rating Year (ray)

If the Zeitgeist is a reasonable thought on handling the problem, it is natural to have not only the release_year but the rating_year too.

```
rating_year <- as.integer( #Assures the year is an integer
  year( #Extracts the year
    as.Date( #Extracts the date
      as.POSIXct( #R default time system.
        edx$timestamp,
          origin="1972-01-01")))) #The highest timestamp on the dataset. Unix Epoch
edx <- as.data.frame(cbind(edx, rating_year)) #Add the result as a new column
```

2.5) CREATING TRAIN & TEST SET

2.5.1) Partitioning

The edx dataset will be split into train_set and test_set to reduce the chance of over-fitting when finding the best final model.

```
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating,
                                  times = 1,
                                  p = 0.2,
                                  list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
test_set <- test_set %>%
     semi_join(train_set, by = "movieId") %>%
     semi_join(train_set, by = "userId")
```

2.5.2) Summary & Glimpse (Train & Test)

The summary ensures the datasets are as intended.

```
summary(train_set)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   : 789652009
##  1st Qu.:18127   1st Qu.:  648   1st Qu.:3.000   1st Qu.: 946764625
##  Median :35750   Median : 1834   Median :4.000   Median :1035423682
```

```
##  Mean   :35873   Mean   : 4122   Mean   :3.512   Mean   :1032591287
##  3rd Qu.:53611   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1126658211
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1231131736
##     title              genres          release_year   rating_year
##  Length:7200043     Length:7200043     Min.   :1915   Min.   :1997
##  Class :character   Class :character   1st Qu.:1987   1st Qu.:2001
##  Mode  :character   Mode  :character   Median :1994   Median :2004
##                                        Mean   :1990   Mean   :2004
##                                        3rd Qu.:1998   3rd Qu.:2007
##                                        Max.   :2008   Max.   :2011
```

```
glimpse(train_set)
```

```
## Rows: 7,200,043
## Columns: 8
## $ userId       <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, ~
## $ movieId      <dbl> 185, 316, 329, 355, 364, 377, 420, 539, 588, 589, 616, 11~
## $ rating       <dbl> 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.~
## $ timestamp    <int> 838983525, 838983392, 838983392, 838984474, 838983707, 83~
## $ title        <chr> "Net, The (1995)", "Stargate (1994)", "Star Trek: Generat~
## $ genres       <chr> "Action|Crime|Thriller", "Action|Adventure|Sci-Fi", "Acti~
## $ release_year <int> 1995, 1994, 1994, 1994, 1994, 1994, 1994, 1993, 1992, 199~
## $ rating_year  <int> 1998, 1998, 1998, 1998, 1998, 1998, 1998, 1998, 1998, 199~
```

```
summary(test_set)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   : 789652009
##  1st Qu.:18109   1st Qu.:  648   1st Qu.:3.000   1st Qu.: 946788339
##  Median :35700   Median : 1834   Median :4.000   Median :1035735841
##  Mean   :35856   Mean   : 4121   Mean   :3.512   Mean   :1032710716
##  3rd Qu.:53569   3rd Qu.: 3627   3rd Qu.:4.000   3rd Qu.:1126895357
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1231131137
##     title              genres          release_year   rating_year
##  Length:1799966     Length:1799966     Min.   :1915   Min.   :1997
##  Class :character   Class :character   1st Qu.:1987   1st Qu.:2002
##  Mode  :character   Mode  :character   Median :1994   Median :2004
##                                        Mean   :1990   Mean   :2004
##                                        3rd Qu.:1998   3rd Qu.:2007
##                                        Max.   :2008   Max.   :2011
```

```
glimpse(test_set)
```

```
## Rows: 1,799,966
## Columns: 8
## $ userId       <int> 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, ~
## $ movieId      <dbl> 122, 292, 356, 362, 370, 466, 520, 594, 260, 376, 539, 59~
## $ rating       <dbl> 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 3.0, 3.0, 5.~
## $ timestamp    <int> 838985046, 838983421, 838983653, 838984885, 838984596, 83~
## $ title        <chr> "Boomerang (1992)", "Outbreak (1995)", "Forrest Gump (199~
## $ genres       <chr> "Comedy|Romance", "Action|Drama|Sci-Fi|Thriller", "Comedy~
## $ release_year <int> 1992, 1995, 1994, 1994, 1994, 1993, 1993, 1937, 1977, 199~
## $ rating_year  <int> 1998, 1998, 1998, 1998, 1998, 1998, 1998, 1998, 1999, 199~
```

2.6) METHODOLOGY

2.6.1) RMSE Function

Even thou the ratings are factors (from 0.5 to 5, incrementing by 0.5 each step), the exercise requires an RMSE approach. There will be a function created to calculate when it becomes needed.

```
##Creating a function to RMSE
RMSE <- function(reality, prediction){
     sqrt(mean((reality - prediction)^2))}
```

2.6.2) Mean

Every time a new model is run, it is needed to know if there are improvements or not. The most basic model will always predict the average and will be used to create more accurate and complex models.

```
plain_mean <- mean(edx$rating)
plain_mean
```

```
## [1] 3.512465
```

2.6.3) Naive RMSE

Naive RMSE determines how much a model fails when we guess the average value for every situation. If any model fails more than the Naive RMSE, something is wrong, and there is a need to start over. If the current model accomplishes a result very near to it, the model is not worth running due to the computational processing it uses. If it performs better than Naive RMSE, it is a step in the right direction.

```
naive_rmse <- RMSE(validation$rating, plain_mean)
naive_rmse
```

```
## [1] 1.061202
```

2.6.4) Linear model & Regularization

A linear model was created where each predictor groups data then the error between the whole dataset mean and the actual rating is stored. Like most real datasets, there are outliers in each group, and it is needed to diminish their impact on the final model. Considering that the average is the sum of values divided by the count of entries, a way to regulate is to enhance by how much the sum is divided. This way, the now regulated average of a group with thousand entries will not be much impacted, but in a group with a low number of entries, the value will diminish and impact RMSE by very little. The idea is that larger groups are more reliable than small groups while we calculate central tendencies(mean, median, mode). Multiple loops are run trying incremental values, testing RMSE each time a loop is finished. This is done to find the correct arbitrary divisor value(lambda) for each group. An excessively high or low lambda value would worsen the RMSE, not improve it, so this is a long but necessary step.

```
rmse<-data.frame()
for(j in 1:5){ #The code is going to run 5 times with...
  var_group_by <- ifelse(j==1, "userId",
                  ifelse(j==2, "movieId",
```

```
                  ifelse(j==3, "genres",
                  ifelse(j==4, "release_year",
                               "rating_year"))))
for(i in 0:30){ #...31 times for each j loop.
  var_lambda <-i

temp_set <- train_set %>% #It creates a temp_set...
     group_by_at(var_group_by) %>% #...grouped by the name of the group...
     summarize(stray = sum(rating - plain_mean)/(n()+var_lambda))
#..and calculates with regulation.

#After this we join the values by the group...
stray_set <- test_set %>%
             left_join(temp_set, by=var_group_by) %>%
             select(-rating,-title,-genres,-release_year,-rating_year)

#...then we create a set with the prediction using the mean and the error as reference.
pred_set <- stray_set %>%
                  mutate(y = ifelse((plain_mean + stray)>5, 5,
                               ifelse((plain_mean + stray)<0.5,0.5,
                               plain_mean + stray))) %>%
                               select(-stray)

#This create a organized dataset of result so we can compare.
alocate <- nrow(rmse)+1
rmse[alocate,1] <- var_group_by
rmse[alocate,2] <- var_lambda
rmse[alocate,3] <- RMSE(test_set$rating, pred_set$y)
}
}
names(rmse) <- c("predictor","lambda","result")

#Now that we have every combination the code below shows us the best lambda
#in each group so we can make our decision.
best_reg <- rmse %>%
     group_by(predictor) %>%
     slice(which.min(result)) %>%
     arrange(result)

lambda_u <- best_reg  %>%
                      filter(predictor=="userId") %>%
                      .$lambda

lambda_m <- best_reg  %>%
                      filter(predictor=="movieId") %>%
                      .$lambda

lambda_g <- best_reg  %>%
                      filter(predictor=="genres") %>%
                      .$lambda

lambda_rey <- best_reg  %>%
                      filter(predictor=="release_year") %>%
```

```
                      .$lambda

lambda_ray <- best_reg  %>%
                      filter(predictor=="rating_year") %>%
                      .$lambda

best_reg
```

```
## # A tibble: 5 x 3
## # Groups:   predictor [5]
##   predictor    lambda result
##   <chr>         <int>  <dbl>
## 1 movieId           2  0.944
## 2 userId            5  0.978
## 3 genres            1  1.02
## 4 release_year     30  1.05
## 5 rating_year      30  1.06
```

It is noticeable that release_year(rey) and rating_year(ray) will probably not be helpful because they got the most prominent possible lambda and the worse RMSE by themselves. This points to the fact that they lack explicative power and should not be used in our final model.

```
reg_rmse <- data.frame()
#Creates the prediction based on each predictor and its best lambda.

#Using only userId
by_userId <- train_set %>%
     group_by(userId) %>%
     summarize(stray_u = sum(rating - plain_mean)/(n()+lambda_u))
#Using only movieId
by_movieId <- train_set %>%
     group_by(movieId) %>%
     summarize(stray_m = sum(rating - plain_mean)/(n()+lambda_m))
#Using only genres
by_genres <- train_set %>%
     group_by(genres) %>%
     summarize(stray_g = sum(rating - plain_mean)/(n()+lambda_g))
#Using only rey
by_release_year <- train_set %>%
     group_by(release_year) %>%
     summarize(stray_rey = sum(rating - plain_mean)/(n()+lambda_rey))
#Using only ray
by_rating_year <- train_set %>%
     group_by(rating_year) %>%
     summarize(stray_ray = sum(rating - plain_mean)/(n()+lambda_ray))

#Joins the sets and gets only by how much we failed.
stray_set <- test_set %>%
                      left_join(by_userId, by="userId") %>%
                      left_join(by_movieId, by="movieId") %>%
                      left_join(by_genres, by="genres") %>%
                      left_join(by_release_year, by="release_year") %>%
                      left_join(by_rating_year, by="rating_year") %>%
```

```
                    select(-rating,-title,-genres,-release_year,-rating_year)

#Creates a prediction to each mixed possibility of predictor using the
#information above but it never let it goes under 0.5 or above 5.
pred_set <- stray_set %>%

            mutate(y_u_m = ifelse(
            (plain_mean + stray_u + stray_m)>5,5,
            ifelse(
            (plain_mean + stray_u + stray_m)<0.5,0.5,
            plain_mean + stray_u + stray_m))) %>%

            mutate(y_u_m_g = ifelse(
            (plain_mean + stray_u + stray_m + stray_g)>5,5,
            ifelse(
            (plain_mean + stray_u + stray_m + stray_g)<0.5,0.5,
            plain_mean + stray_u + stray_m + stray_g))) %>%

            mutate(y_u_m_rey = ifelse(
            (plain_mean + stray_u + stray_m + stray_rey)>5,5,
            ifelse(
            (plain_mean + stray_u + stray_m + stray_rey)<0.5,0.5,
            plain_mean + stray_u + stray_m + stray_rey))) %>%

            mutate(y_u_m_ray = ifelse(
            (plain_mean + stray_u + stray_m + stray_ray)>5,5,
            ifelse(
            (plain_mean + stray_u + stray_m + stray_ray)<0.5,0.5,
            plain_mean + stray_u + stray_m + stray_ray))) %>%

            mutate(y_u_m_rey_ray = ifelse(
            (plain_mean + stray_u + stray_m + stray_rey + stray_ray)>5,5,
            ifelse(
            (plain_mean + stray_u + stray_m + stray_rey + stray_ray)<0.5,0.5,
            plain_mean + stray_u + stray_m + stray_rey + stray_ray))) %>%

            mutate(y_u_m_g_rey = ifelse(
            (plain_mean + stray_u + stray_m + stray_g + stray_rey)>5,5,
            ifelse(
            (plain_mean + stray_u + stray_m + stray_g + stray_rey)<0.5,0.5,
            plain_mean + stray_u + stray_m + stray_g + stray_rey))) %>%

            mutate(y_u_m_g_ray = ifelse(
            (plain_mean + stray_u + stray_m + stray_g + stray_ray)>5,5,
            ifelse((plain_mean + stray_u + stray_m + stray_g + stray_ray)<0.5,0.5,
            plain_mean + stray_u + stray_m + stray_g + stray_ray))) %>%

            mutate(y_u_m_g_rey_ray = ifelse(
            (plain_mean + stray_u + stray_m + stray_g + stray_rey + stray_ray)>5,5,
            ifelse(
            (plain_mean + stray_u + stray_m + stray_g + stray_rey + stray_ray)<0.5,0.5,
            plain_mean + stray_u + stray_m + stray_g + stray_rey + stray_ray))) %>%
```

```
            mutate(average = plain_mean) %>%
            select(-stray_u, -stray_m, -stray_g, -stray_rey, -stray_ray, -timestamp)

#Calculates the RMSE with each mixed probability and creates an organized table
#where we can observe the results.
for(i in 3:ncol(pred_set)){
alocate <- i-2
reg_rmse[alocate,1] <- names(pred_set)[i]
reg_rmse[alocate,2] <- RMSE(test_set$rating, pred_set[,i])
}
colnames(reg_rmse) <- c("method","rmse")

reg_rmse %>%
        arrange(rmse)
```

```
##             method      rmse
## 1            y_u_m 0.8835378
## 2        y_u_m_ray 0.8859393
## 3        y_u_m_rey 0.8984555
## 4    y_u_m_rey_ray 0.9006558
## 5          y_u_m_g 0.9387109
## 6      y_u_m_g_ray 0.9406950
## 7      y_u_m_g_rey 0.9546951
## 8  y_u_m_g_rey_ray 0.9564228
## 9          average 1.0599043
```

Even though the best model(y_u_m) using the linear model is far better than Naive RMSE, it is not enough to achieve the proposed exercise goal. The answer lay in Matrix Factorization.

2.6.5) Matrix Factorization - Recosystem

The MovieLens Grading Rubric stated:

> *Note that to receive* ***full marks*** *on this project (. . .) your work on this project* ***needs to build on code that is already provided***

The FAQ stated:

> *Q:* ***Can I use*** *the recommenderlab,* ***recosystem****, etc. packages for my MovieLens project?*
>
> *A: If you understand how they work,* ***yes!***

Recosystem is a package that utilizes parallel matrix factorization to solve this problem. It has built-in variables not only to find the best tunning parameters (dim, costp_l2, costq_l2, costp_l1, costq_l1, lrate) but to create the iterations and cross-validation by itself (niter, nfold) and even choose how many threads of parallel computing are going to be used (nthread) to accomplish the goal.

LINK: https://cran.r-project.org/web/packages/recosystem/recosystem.pdf

```
# The https://github.com/yixuan/recosystem states:
# The data file for training set needs to be a
# sparse matrix with each line containing 03 numbers
# with the names user_index, item_index and rating.
# data_memory(): Specifies a data set from R objects

train_data <-  with(train_set,
                    data_memory(user_index = userId,
                                item_index = movieId,
                                rating = rating))
test_data  <-  with(test_set,
                    data_memory(user_index = userId,
                                item_index = movieId,
                                rating = rating))

# Create the model object.
r <-  recosystem::Reco()

# Select the minimum and maximum tuning parameters
opts <- r$tune(train_data, opts = list(dim = c(10, 20, 30), #default
                                       lrate = c(0.05, 0.1, 0.2), #default
                                       nthread = 3, niter = 10)) #default

# Train the algorithm using the optimal value ($min) of the tunning parameters above.
r$train(train_data,
        opts = c(opts$min,
                 nthread = 3,
                 niter = 10))
```

```
## iter      tr_rmse          obj
##    0       0.9915   1.0020e+07
##    1       0.8788   8.0845e+06
##    2       0.8477   7.5078e+06
##    3       0.8264   7.1709e+06
##    4       0.8086   6.9158e+06
##    5       0.7947   6.7255e+06
##    6       0.7836   6.5835e+06
##    7       0.7743   6.4702e+06
##    8       0.7664   6.3764e+06
##    9       0.7593   6.3013e+06
```

```
pred_reco <- r$predict(test_data, out_memory())
RMSE(test_set$rating, pred_reco)
```

```
## [1] 0.7989018
```

As can be seen, the recosystem achieved the goal when compared to the test_set even with the default parameters.

Even though it is a good indicator of the right direction, success with the test_set does not guarantee the same results using validation and the edx set. If the parameter creates some over-fitting, they will need to adjust.

```
edx_data <-  with(edx,
                  data_memory(user_index = userId,
                              item_index = movieId,
                              rating = rating))
val_data  <-  with(validation,
                  data_memory(user_index = userId,
                              item_index = movieId,
                              rating = rating))

# Create the model object.
r <-  recosystem::Reco()

# Select the minimum and maximum tuning parameters
opts <- r$tune(edx_data, opts = list(dim = c(10, 20, 30), #default
                                     lrate = c(0.05, 0.1, 0.2), #default
                                     nthread = 3, niter = 10)) #default

# Train the algorithm using the optimal value ($min) of the tunning parameters above.
r$train(edx_data,
        opts = c(opts$min,
                 nthread = 3,
                 niter = 10))
```

```
## iter      tr_rmse          obj
##    0       0.9726   1.2007e+07
##    1       0.8719   9.8800e+06
##    2       0.8381   9.1534e+06
##    3       0.8167   8.7525e+06
##    4       0.8008   8.4667e+06
##    5       0.7886   8.2666e+06
##    6       0.7788   8.1144e+06
##    7       0.7708   7.9904e+06
##    8       0.7642   7.8995e+06
##    9       0.7585   7.8204e+06
```

```
pred_reco <- r$predict(val_data, out_memory())
RMSE(validation$rating, pred_reco)
```

```
## [1] 0.7918954
```

As can be seen, the recosystem achieved the goal compared to the validation.

3) RESULTS

```
line <- nrow(reg_rmse)+1
reg_rmse[line,1] <- "Matrix Factorization"
reg_rmse[line,2] <- RMSE(validation$rating, pred_reco)

reg_rmse %>%
          arrange(rmse)
```

```
##                  method      rmse
## 1  Matrix Factorization 0.7918954
## 2                 y_u_m 0.8835378
## 3             y_u_m_ray 0.8859393
## 4             y_u_m_rey 0.8984555
## 5         y_u_m_rey_ray 0.9006558
## 6               y_u_m_g 0.9387109
## 7           y_u_m_g_ray 0.9406950
## 8           y_u_m_g_rey 0.9546951
## 9       y_u_m_g_rey_ray 0.9564228
## 10              average 1.0599043
```

The matrix factorization clearly accomplished success (`<0.8649`) by a fair margin and beats every single method based on linear models. Recosystem is a powerful tool that can solve this problem even with default parameters.

4) CONCLUSION

The presented solution is the result of many tries that are not fully described here because most crashed during execution or had such long-running times that it became impossible to proceed even when using any of the specialized cloud-based services that were affordable by the writer. Probably there are more elegant ways of accomplishing what was done here with more knowledge, experience, and computational power at disposal. Changing the Recosystem parameter would probably enhance the result even more if more time was at its disposal and could be considered for future iterations.