MovieLens Recommendation System

Alejandro Oscar BOFFA

april 3 2020

Contents

1. Introduction, overview, executive summary

1.1 Introduction

Recommendation systems are Machine Learning Algorithms that use ratings given by users to make specific recommendations. Companies that commercialize many products to many customers and permit these customers to rate their products, like Amazon, collect massive datasets that can be used to predict what rating that a particular user will give a specific item based on historical user behavior. Items for which a high rating is predicted for a given user are then recommended to that user.

Netflix uses a recommendation system to predict how many stars a user will give a specific movie. One star suggests it is not a good movie and five stars suggests it is an excellent movie.

This project is based on some of the approaches taken by the winners of the Netflix challenges. On October 2006, Netflix offered a challenge to the data science community: **"Improve our recommendation algorithm by 10% and win a million dollars"**

In September 2009, the winners were announced. Here we will use some of the data analysis strategies used by the winning team. You can read a good summary of how the winning algorithm was put together here: http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/ and a more detailed explanation here: http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf.

1.2 Overview

The purpose for this project is create a Movie Recommendation System using MovieLens dataset.

The Netflix data is not publicly available, but the GroupLens research lab generated their own database with over 27million ratings applied to 58,000 movies by 280,000 users(full version, found it here:https://grouplens.org/datasets/movielens/latest/).

We will use the 10Million version of the MovieLens dataset to make the computation a little easier.(found it here: https://grouplens.org/datasets/movielens/10m/ or here http://files.grouplens.org/datasets/movielens/ml-10m.zip)

The content of datasets is divided in: userId, movieId, rating, title, genres, timestamp. We´ll explore them later.

This Movielens dataset was provided by Harvard´s Data science staff, so we will download the MovieLens data and run code they provided to generate our datasets, splitting it: **edx**(90% of the data) and **validation**(10% of the data)

Next we have to clean and transform our data to permit us a clear and efficient process and apply our machine learning algorithms to obtain that one with the smallest RMSE value (smallest error in our prediction). The Netflix challenge used the typical error loss: they decided on a winner based on the residual mean squared

error ($RMSE$) on a test set.The value to achieve is $lessthan 0.86490$. We can interpret the RMSE similarly to a standard deviation: it is the typical error we make when predicting a movie rating. If this number is larger than 1, it means our typical error is larger than ONE STAR, which is not good. The Residual Mean Square Error (RMSE) is our LOSS function:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}((y.hat_{u,i} - y_{u,i})^2}$$

We define $y_{u,i}$ as the rating for movie i by user u and denote our prediction with $y.hat_{u,i}$, with N being the number of user/movie combinations and the sum occurring over all these combinations.

Here we write the function that computes the RMSE for vectors of ratings and their corresponding predictors:

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

In this report we´ll do data exploration, visualization, pre-processing, evaluate machine learning algorithms and RMSE analysis to create the best predictive model.

1.3 Executive Summary

After a initial data cleaning, pre-processing and exploration, **the recommendation system builted on the train dataset are evaluated on the test dataset (edx splitted) and the best model is chosen based on the Root Mean Squared Error that must be lower than 0.86490 (RMSE < 0.86490), using original edx and validation datasets**
We´ll use different models, from the simplest possible recommendation system, predicting the same rating for all movies regardless of user through Regularized movie + user effects. The regularization method allows us to add a tuning parameter *lambda* to penalize movies with large estimates from a small sample size. **On the Regularized Movie + User Effects Model we already get RMSE = 0.8648177**, that is less than our priority goal RMSE = 0.86490, so we have met our goal here.

2. Methods and Analysis

(NOTE: This project is made in R version 3.5.3, on my laptop: Intel I5+8gb ram+ssd 500gb, every time I ran it took about 3 hours, because separate_row() function on large dataset. In the last hours before finishing it, I discovered the existence of Microsoft R Open 3.5.3, and after installing it, I was surprised to see that I executed the total same project in 20 minutes, 10 X faster, being fully compatible. I recommend it).

Our first step is create edx and validation datasets:

We install R packages if they don´t exist, download and read movielens file, process it, create datasets an remove temporal files .....

2.1 Data exploration and visualization

The edx dataset is a subset of the MovieLens 10M data table made of 6 variables (columns) and a total of approx 9million observations (rows) and the validation dataset represents approximately 10%= approx 1million observations and contains the same 6 variables.

How are the first rows of edx dataset? Each row represents a rating given by one user to one movie...

```
## # A tibble: 9,000,055 x 6
##    userId movieId rating timestamp title              genres
##     <int>   <dbl>  <dbl>     <int> <chr>              <chr>
##  1      1     122      5 838985046 Boomerang (1992)   Comedy|Romance
##  2      1     185      5 838983525 Net, The (1995)    Action|Crime|Thrill~
##  3      1     292      5 838983421 Outbreak (1995)    Action|Drama|Sci-Fi~
```

```
##  4      1     316      5 838983392 Stargate (1994)     Action|Adventure|Sc~
##  5      1     329      5 838983392 Star Trek: Generat~ Action|Adventure|Dr~
##  6      1     355      5 838984474 Flintstones, The (~ Children|Comedy|Fan~
##  7      1     356      5 838983653 Forrest Gump (1994) Comedy|Drama|Romanc~
##  8      1     362      5 838984885 Jungle Book, The (~ Adventure|Children|~
##  9      1     364      5 838983707 Lion King, The (19~ Adventure|Animation~
## 10      1     370      5 838984596 Naked Gun 33 1/3: ~ Action|Comedy
## # ... with 9,000,045 more rows
```

We can see the number of unique users that provided ratings and how many unique movies were rated in the edx training dataset:

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

We see the edx dataset contains approximately 9 Millions of rows with ~70.000 different users and ~11.000 movies with rating score between 0.5 and 5.
There is no missing values (0 or NA):

```
##   lower_score upper_score missing_values
## 1         0.5           5              0
```

Each observation or row of edx dataset contains the following variables or columns:

1. **userId**: the unique identifier for the user.
2. **movieId**: the unique identifier for the movie.
3. **rating**: a numeric rating between 0 and 5 for the movie in 0.5 increments.
4. **timestamp**: unix timestamp when the user rating was provided in seconds since Unix Epoch on January 1st, 1970 at UTC.
5. **title**: movie title with year of release at the end between "()".
6. **genres**: genres associated with the movie. Unique movie can have several genres separated by "|".

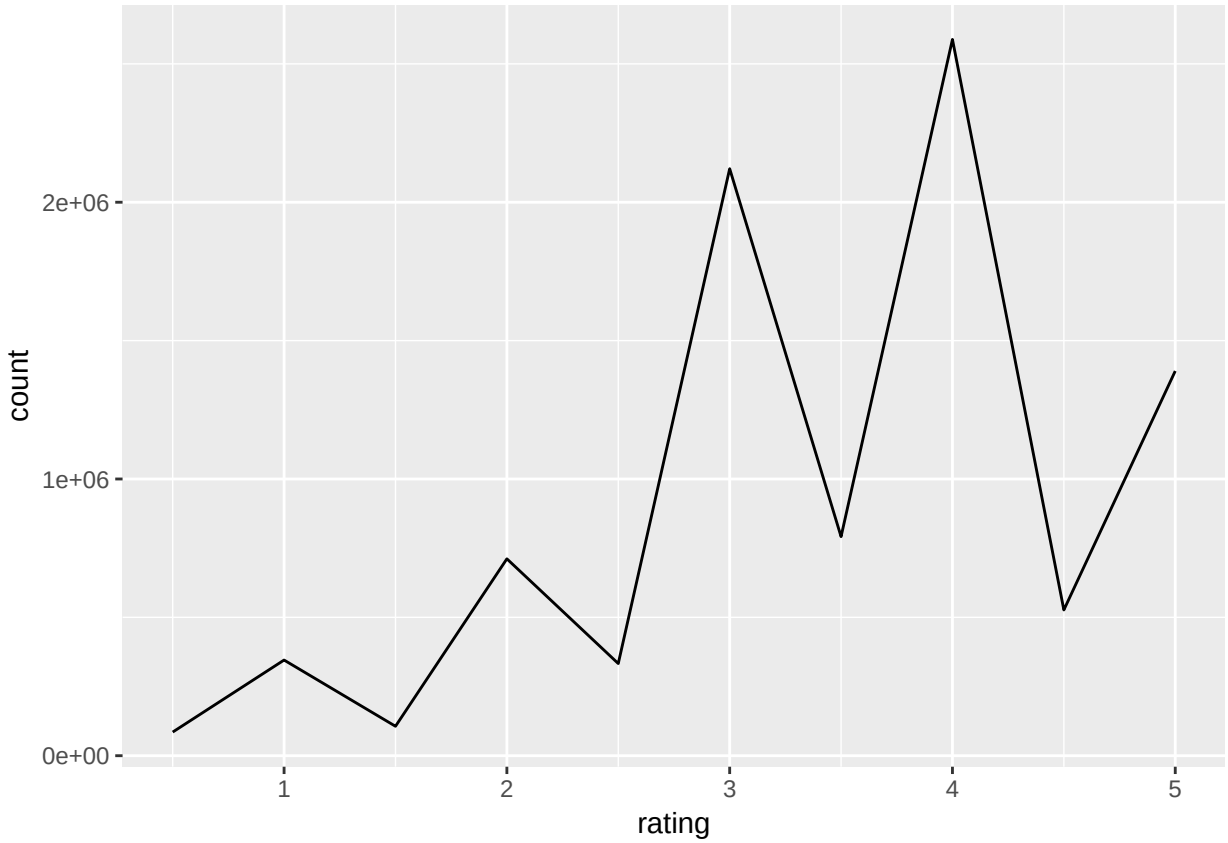
Now we can see the top 10 most rated movies, organized by the quantity of rating:

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                     count
##      <dbl> <chr>                                                     <int>
##  1     296 Pulp Fiction (1994)                                       31362
##  2     356 Forrest Gump (1994)                                       31079
##  3     593 Silence of the Lambs, The (1991)                          30382
##  4     480 Jurassic Park (1993)                                      29360
##  5     318 Shawshank Redemption, The (1994)                          28015
##  6     110 Braveheart (1995)                                         26212
##  7     457 Fugitive, The (1993)                                      25998
##  8     589 Terminator 2: Judgment Day (1991)                         25984
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (19~ 25672
## 10     150 Apollo 13 (1995)                                          24284
## # ... with 10,667 more rows
```

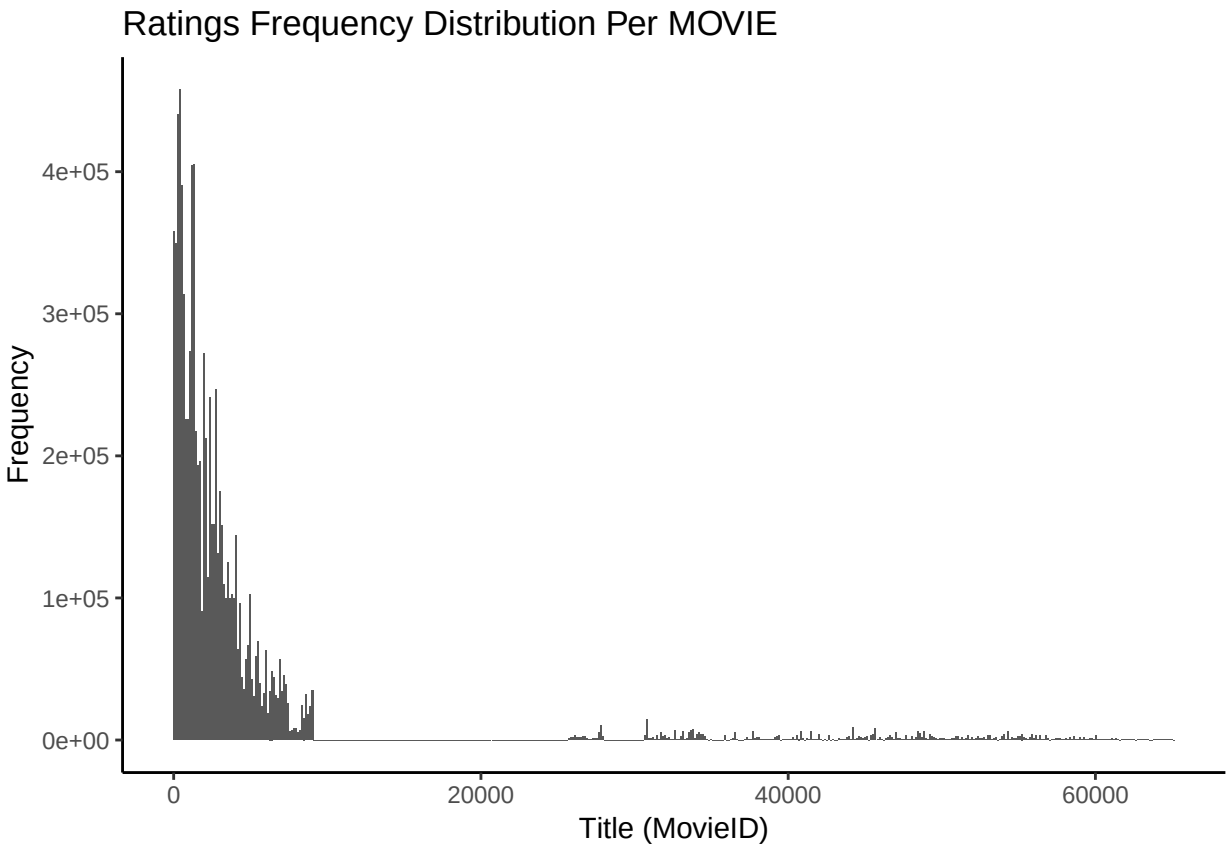
And the most given ratings and their distribution plot:

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

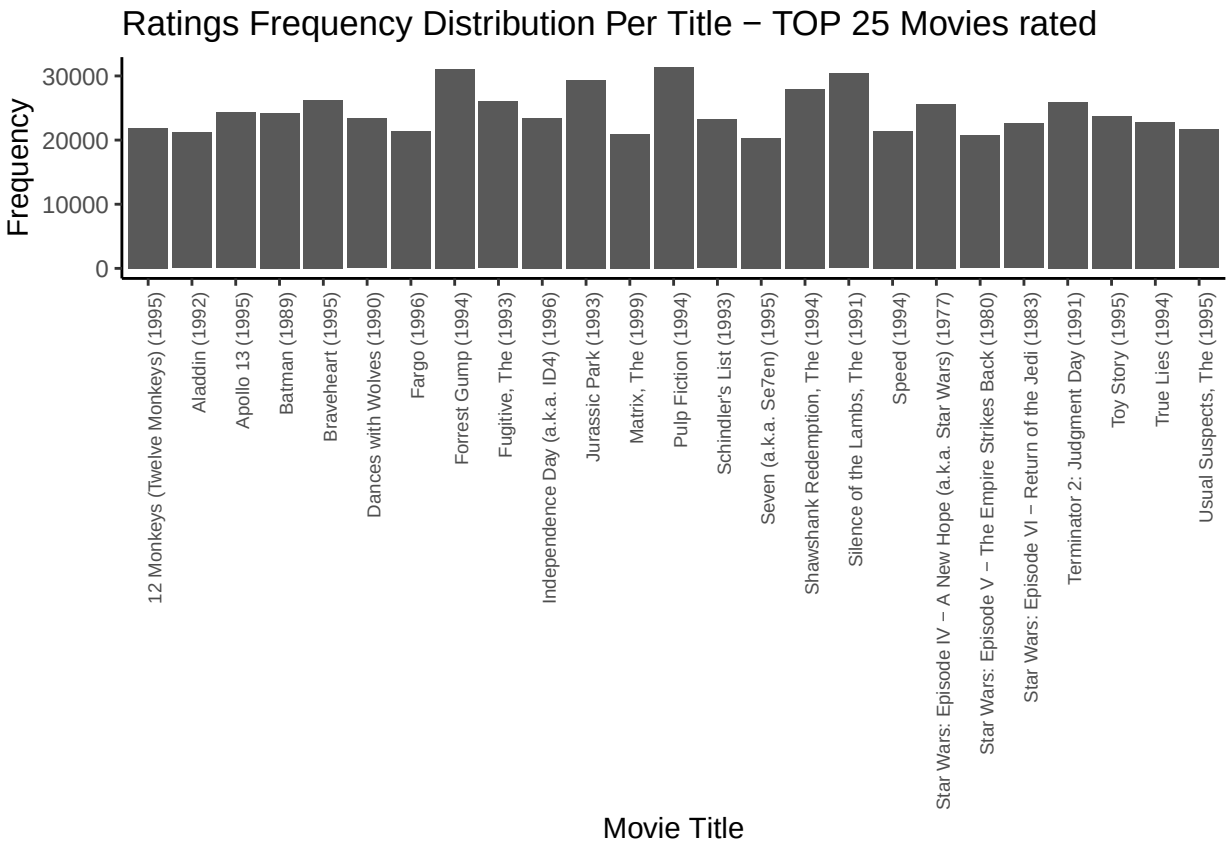
```
## 5     2     711422
```



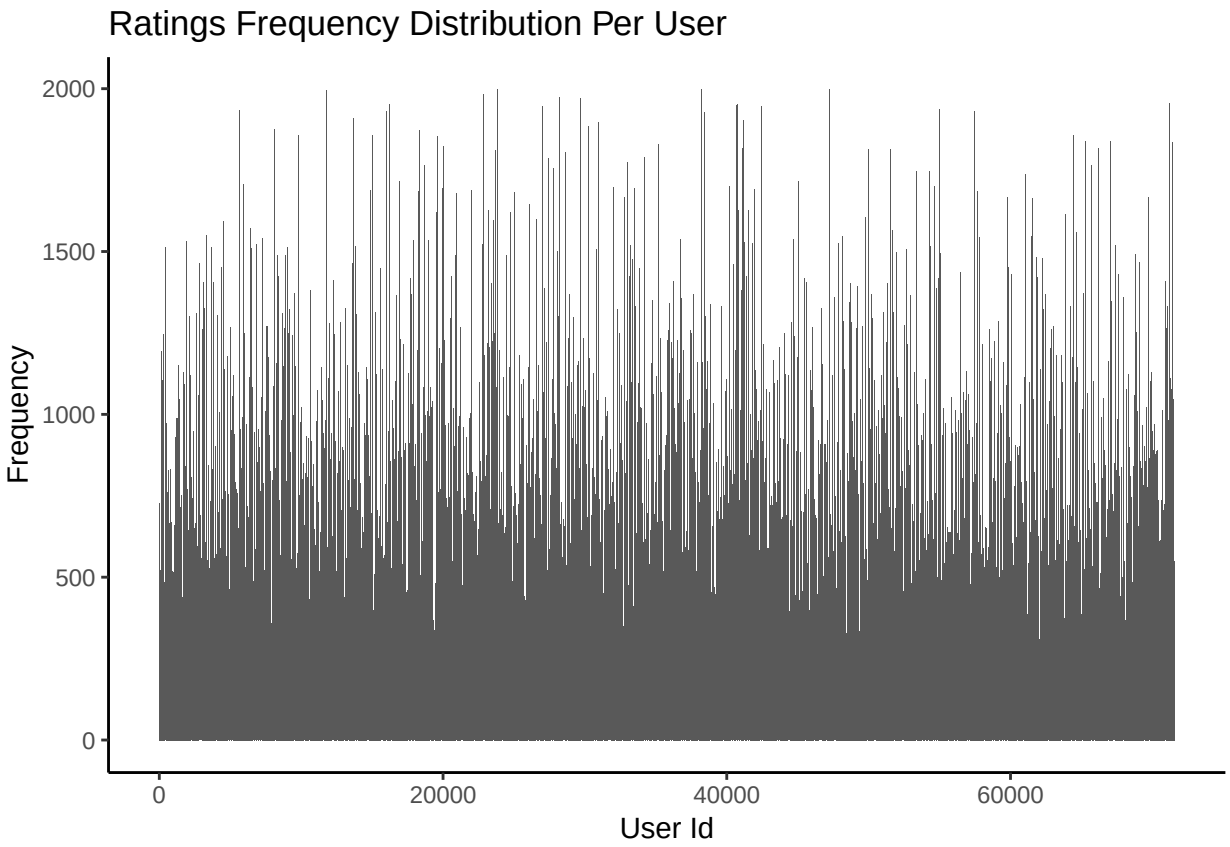
We can see in general, half star ratings are less than whole star ratings. There are fewer ratings of 3.5 or 4.5 than ratings of 3 or 4.
It´s important to explore how many rating per movie have this dataset.
We see the lowers movieId are the most rated:



Then, the top 25 rated movie are:



Also, we can see the rating distribution by User:



In general we see some movies get rated more than others. This should not surprise us given that there are blockbuster movies watched by millions and artsy, independent movies watched by just a few.
Other interesting observation is that some users are more active than others at rating movies.

2.2 Data pre-processing

Now we´ll continue with data pre-processing. . . .We need to separate several genres included in the same row, indicated with "|", because we need analyze if "genres" effect can really improve the predictions in our models.

After many hours of trying **separate_rows() function on Data frame "edx"**(remember:~9 million rows) and get only "error, memory out" message, I take the decision to convert edx dataset in Data.Table. This was really faster an without error message.
Data tables work very fast with large datasets. Now, for the rest of data visualization, our source of data will be **edxDT** Data Table. Extract "yearofRating" from "timestamp" column and remove columns and files that we don´t use anymore. Also, we define the RMSE function.
This is the code:

```
##########################################################################
# Data Pre-processing

##########################################################################
# Separate genre from datasets by "|" and yearofRating to view statistics
#
# After 100 times trying separate_row () on the edx Data Frame to separate the "genre"
# through its separator "|" and after hours of computing ~10 millions of rows
# on WIN10 64bit I5 + 8gb ram + ssd500gb laptop, I get the message "error, memory out",
```

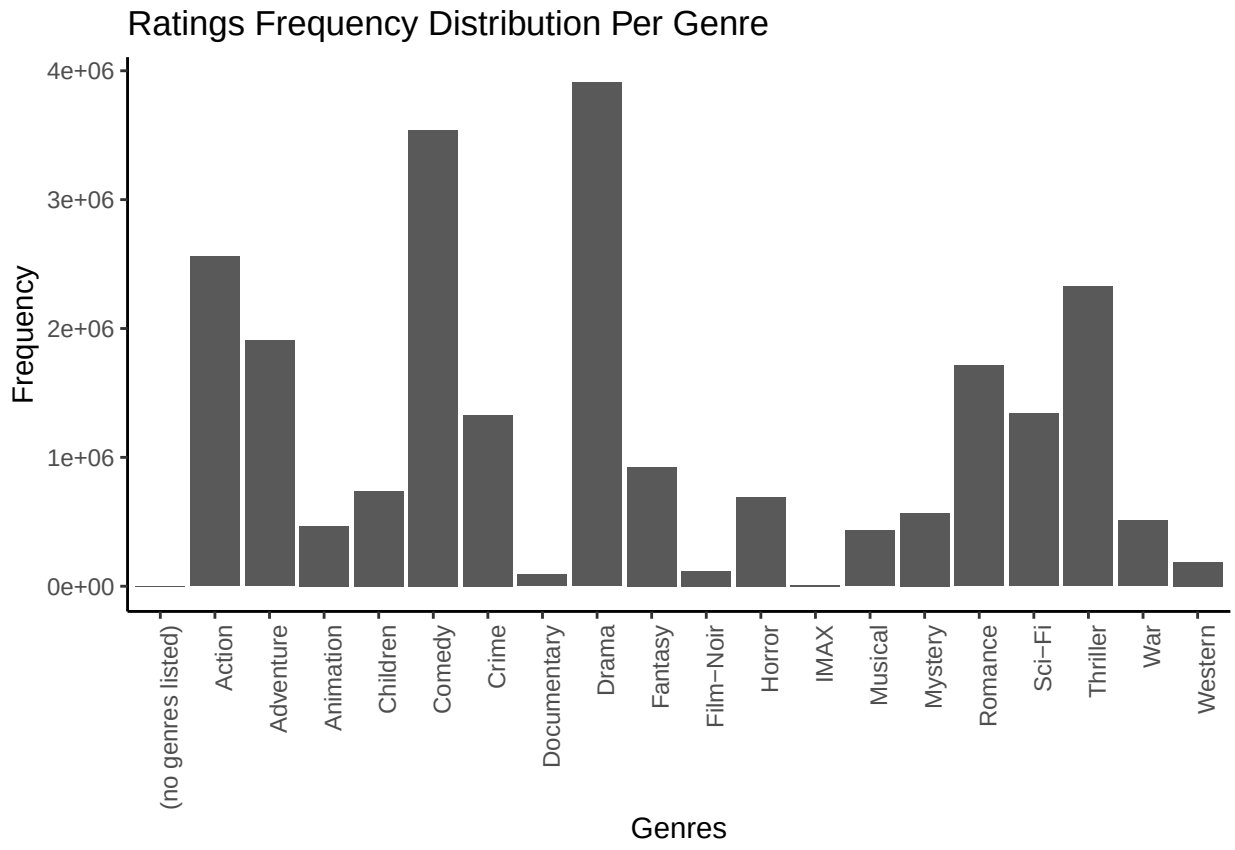
```
# setting memory.limit(64000) on console and after investigating several days,
# I got the following simple solution
# (if you know a better one, please, let me know, thanks:)

# Convert dataframes edx & validation to "data table"-> edxDT & validationDT
# from "edx Data Frame" ~ 900mb(Global Enviroment) to ~ 400mb in "edx Data Table",
# reducing the size of file in Mbytes to a half

memory.limit(64000)
```

```
## [1] 64000
```

```
edxDT <- as.data.table(edx)

# Split in half Data Table "edxDT"

index1 <- as.integer(nrow(edxDT) / 2)
index2 <- nrow(edxDT) - index1
edx1 <- edxDT[1:index1,]
edx2 <- edxDT[index2:nrow(edx),]

# separate "genres" in both data tables generated "edx1" & "edx2".
# It took 2 hours with R 3.5.3 and 2 minutes with Microsoft R Open 3.5.3
# Yes, procesing  ~24millon rows with Microsoft R Open in 2 minutes!!!!

edx1  <- edx1  %>% separate_rows(genres, sep = "\\|")
edx2  <- edx2  %>% separate_rows(genres, sep = "\\|")

# join both "edx_" data tables

edxDT <- bind_rows(edx1, edx2) # now we have a dataset of ~24 million rows

# Extract yearofRating from timestamp in the dataset edxDT.
# We will use it as Year effect

edxDT <- as.data.table(edxDT %>%
  mutate(yearofRating = as.integer(format(as.POSIXct(timestamp, origin="1970-01-01"),
                                          "%Y"))))

# removing big obsolete datasets

rm(edx1, edx2) # release memory Ram!

# removing "timestamp" column from both datasets

edxDT <- select(edxDT, -timestamp)



#############################################################################
# This is the Root Mean Square Error "RMSE" function that will give
# us the scores found with our models.
# The lower this result, the lower the error in our prediction.
# Our goal is obtain a RMSE < 0.86490
```

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

Now we can explore Distribution of Rating by Genre:



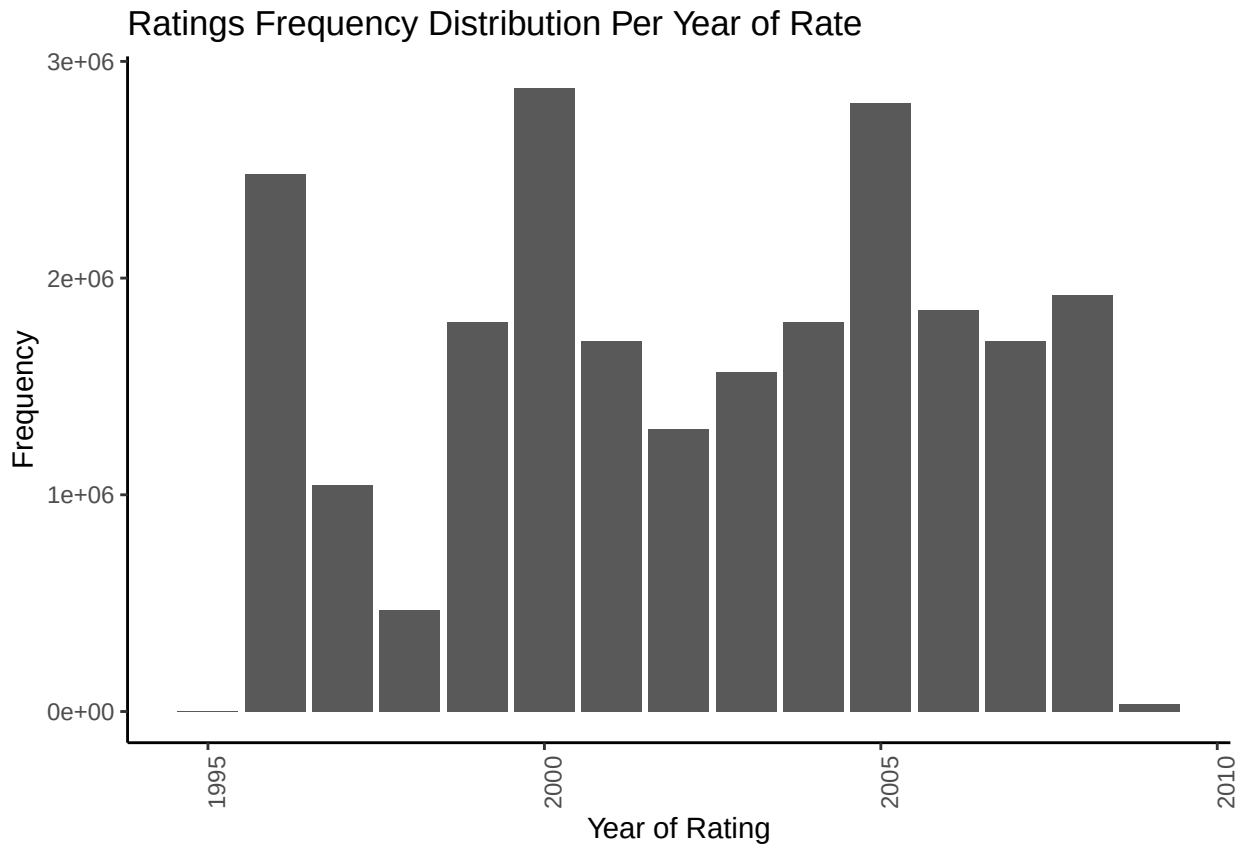
and we can see the number of ratings grouped by separate genres classified:

```
## # A tibble: 20 x 2
##    genres               count
##    <chr>                <int>
##  1 Drama              3910127
##  2 Comedy             3540930
##  3 Action             2560545
##  4 Thriller           2325899
##  5 Adventure          1908892
##  6 Romance            1712100
##  7 Sci-Fi             1341183
##  8 Crime              1327715
##  9 Fantasy             925637
## 10 Children            737994
## 11 Horror              691485
## 12 Mystery             568332
## 13 War                 511147
## 14 Animation           467168
## 15 Musical             433080
```

```
## 16 Western             189394
## 17 Film-Noir           118541
## 18 Documentary          93066
## 19 IMAX                  8181
## 20 (no genres listed)       7
```

We can see, grouped users ratings by movie genre, “Drama” and “Comedy” genres were the most rated. Documentary and IMAX genre movies were the least rated by users. Does this mean that people don’t like Documentaries?, We don´t know.

Also we can explore Distribution of Rating by Year of rate:



Now, we have to create our train and test sets from splitted edx dataset, this is to avoid overfitting (too optimistic result). Also, in this part of the report, we start to show the code used, so the solution is easy to understand. Here is the code:

```
############################################################
#  Spliting edx dataset in train_set and test_set
# ############################################################
#
#  The validation data should NOT be used for training the algorithm
#  and should ONLY be used for evaluating the RMSE of final algorithm.
#  You should split the edx data into separate training and test sets
#  to design and test your algorithm.


set.seed(1) # if you are using R 3.5 or Microsoft R Open 3.5.3
# set.seed(1, sample.kind="Rounding") if using R 3.5.3 or later
```

```r
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

# To make sure we don't include users and movies in the test set
# that do not appear in the training set,
# we remove these entries using the semi_join function:

test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

2.3 Model building approach

As we see before, Movie and User are the principal effects to consider in our recommendation system project. Later we´ll consider regularization method that permits us to penalize large estimates that are formed using small sample sizes.

2.3.1 Just the average model

Here we make the simplest possible recommendation system: predict the same rating for all movies regardless of user. A model that assumes the same rating for all movies and users with all the differences explained by random variation is:

$$Yu,i = mu + eu,i$$

with eu,i independent errors sampled from the same distribution centered at 0 and mu the "true" rating for all movies. We know that the estimate that minimizes the RMSE is the least squares estimate of mu and, in this case, is the average of all ratings:

```r
############################
### Starting Naive model ###

# Calculating "just the average" of all movies

mu <- mean(train_set$rating)

# Calculating the RMSE on the test set

naive_rmse <- RMSE(test_set$rating, mu)

# Creating a results dataframe that will contains all RMSE results.
# Here we insert our first RMSE.

rmse_results <- data.frame(method = "Just the average", RMSE = naive_rmse)

# show the RMSE result

rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.060054

We get a RMSE of about 1. To win the Netflix grand prize of $1,000,000, a participating team had to get an RMSE of about 0.857 and in our project, we have to get RMSE < 0.86490. So we have to do it better!

2.3.2 Movie effect model

As we see previously, some movies are just generally rated higher than others. We can augment our previous model by adding the term *bi* to represent average ranking for movie *i*:

$$Yu,i = mu + bi + eu,i$$

Statistics textbooks refer to the *bs* as effects. However, in the Netflix challenge papers, they refer to them as "*bias*", thus the *b* notation.

```
###################################
### Starting Movie Effect Model ###
# Calculating the average by movie

movie_avgs <- train_set %>%
   group_by(movieId) %>%
   summarize(b_i = mean(rating - mu))

# Computing the predicted ratings on test dataset

predicted_ratings <- mu + test_set %>%
   left_join(movie_avgs, by='movieId') %>%
   .$b_i

# Computing Movie effect model

model_1_rmse <- RMSE(predicted_ratings, test_set$rating)

# Adding the results to the rmse_results table

rmse_results <- bind_rows(rmse_results,
                          data.frame(method = "Movie Effect Model",
                                     RMSE = model_1_rmse ))

# show the RMSE result

rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615

We see an improvement but not enough....

2.3.3 User effect model

Because a substantial variability across users, an improvement to our model may be:

$$Yu,i = mu + bi + bu + eu,i$$

where *bu* is a *user* effect, added to our previous model...

```
##########################################
### Starting Movie + User Effect Model ###

# Calculating the average by user
```

```r
user_avgs <- train_set %>%
   left_join(movie_avgs, by='movieId') %>%
   group_by(userId) %>%
   summarize(b_u = mean(rating - mu - b_i))

# Computing the predicted ratings on test dataset

predicted_ratings <- test_set %>%
   left_join(movie_avgs, by='movieId') %>%
   left_join(user_avgs, by='userId') %>%
   mutate(pred = mu + b_i + b_u) %>%
   .$pred

model_2_rmse <- RMSE(predicted_ratings, test_set$rating)

# Adding the results to the results dataset

rmse_results <- bind_rows(rmse_results,
                          data.frame(method = "Movie + User Effects Model",
                                     RMSE = model_2_rmse ))

# show the RMSE result

rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646844

Here we really make an improvement, from RMSE = 0.9429615 to a RMSE = 0.8646844. From here we could say that we reached our goal, but if we choose this as the final model, we should apply the edx and validation dataset to make sure. But we don´t stop here, we investigate a little more....

2.3.4 Regularized movie and user effect model

Analyzing our dataset, "best" and "worst" movies were rated by very few users. These movies were mostly unknown ones. This is because with just a few users, we have more uncertainty. Therefore, larger estimates of b_i, negative or positive, are more likely. These are noisy estimates that we should not trust, especially when it comes to prediction. **Large errors can increase our RMSE**, so regularization permits us to **penalize** large estimates that are formed using small sample sizes...

We are using this equation:

$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i)^2+\lambda\sum_i b_i^2$$

The first term is just least squares and the second is a penalty that gets larger when many b_i are large. Using calculus we can actually show that the values of bi that minimize this equation are:

$$\hat{b_i}(\lambda)=\frac{1}{\lambda+n_i}\sum_{u=1}^{n_i}(Y_{u,i}-\hat{\mu})$$

where n_i is the number of ratings made for movie i.
This is how it work: when our sample size n_i is very large, a case which will give us a stable estimate, then

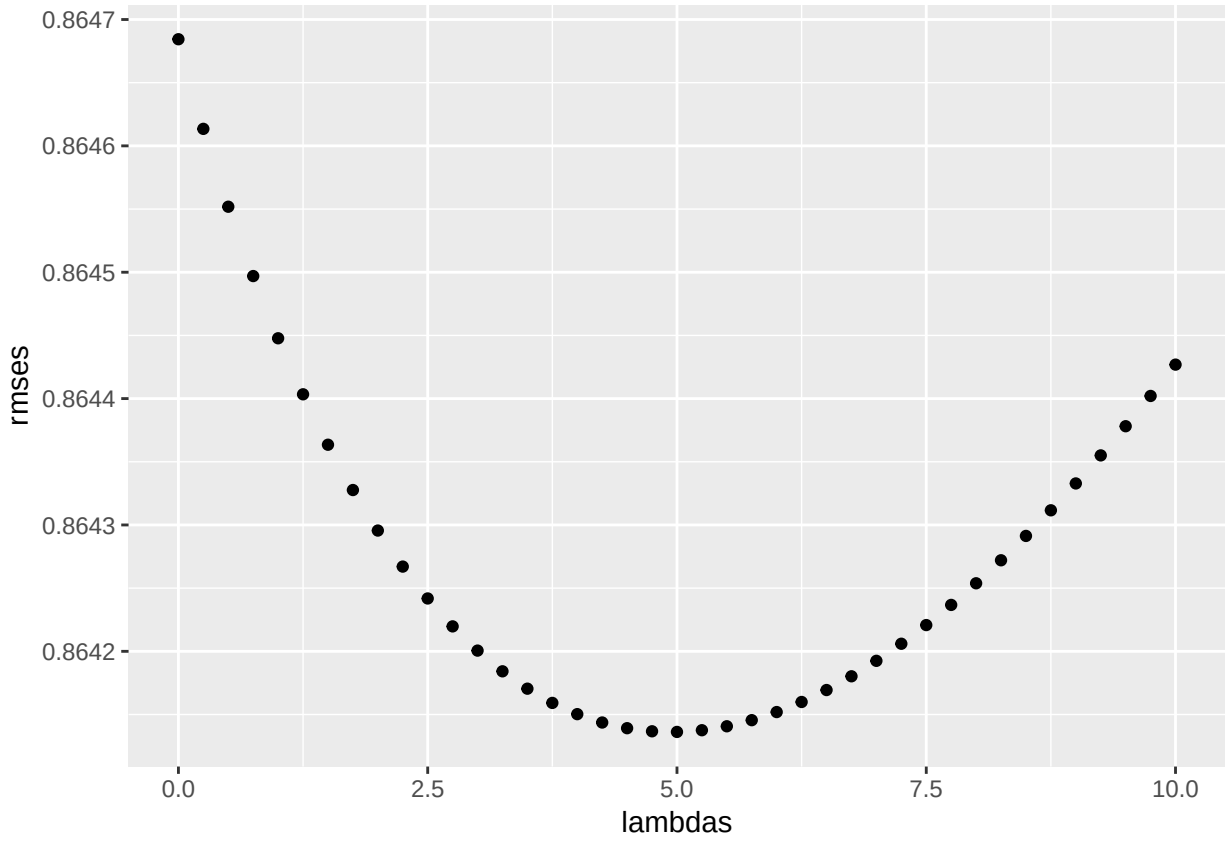
the penalty "lambda" is effectively ignored since n_i + lambda ~ ni. However, when the n_i is small, the estimate $\hat{}b_i(lambda)$ is shrunken towards 0. **The larger lambda, the more we shrink**:

So, we compute this...

```
##########################################
# Starting Regularization of our models.
##########################################

#######################################
# Regularizing Movie + User Effect Model
# Computing the predicted ratings on test dataset using different values of lambda
# b_i is the Movie effect and b_u is User effect.
# Lambda is a tuning parameter.
# We are using cross-validation to choose the best lambda that minimize our RMSE.

lambdas <- seq(0, 10, 0.25)

# function rmses calculate predictions with several lambdas

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

   # Calculating the average by movie

   b_i <- train_set %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating - mu) / (n() + l))

   # Calculating the average by user

   b_u <- train_set %>%
      left_join(b_i, by="movieId") %>%
      group_by(userId) %>%
      summarize(b_u = sum(rating - b_i - mu) / (n() + l))

   # Computing the predicted ratings on test dataset

   predicted_ratings <- test_set %>%
      left_join(b_i, by = 'movieId') %>%
      left_join(b_u, by = "userId") %>%
      mutate(pred = mu + b_i + b_u) %>%
      .$pred

   # Predicting the RMSE on the test set

   return(RMSE(predicted_ratings, test_set$rating))
})
```

This is the plot to choose the best lambda (the lower one):

```
# Getting the best lambda value that minimize the RMSE on reg movie + user effects model

qplot(lambdas, rmses) # visualizing the best lambda
```




```
lambda <- lambdas[which.min(rmses)]
lambda # the best lambda
```

```
## [1] 5
```

```
# We know that our best RMSE is given by: RMSE = min(rmses),
# but as a purpose of clarification,
# we compute again our estimate with best lambda found:

# Computing regularized estimates of b_i using best lambda

movie_avgs_reg <- train_set %>%
   group_by(movieId) %>%
   summarize(b_i = sum(rating - mu) / (n() + lambda), n_i = n())

# Computing regularized estimates of b_u using best lambda

user_avgs_reg <- train_set %>%
   left_join(movie_avgs_reg, by ='movieId') %>%
   group_by(userId) %>%
   summarize(b_u = sum(rating - mu - b_i) / (n() + lambda), n_u = n())

# Predicting ratings

predicted_ratings <- test_set %>%
   left_join(movie_avgs_reg, by = 'movieId') %>%
   left_join(user_avgs_reg, by = 'userId') %>%
```

```
   mutate(pred = mu + b_i + b_u) %>%
   .$pred

# Predicting the RMSE on the test set

model_3_rmse <- RMSE(predicted_ratings, test_set$rating)

# Adding the results to the rmse_results dataset

rmse_results <- bind_rows(rmse_results,
                          data.frame(method = "Regularized Movie + User Effects Model",
                                     RMSE = model_3_rmse ))

# Show the table with the differents Models and their RMSEs results
# using splitted edx dataset

rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646844
Regularized Movie + User Effects Model	0.8641362

The **penalized** estimates provide an improvement over the least squares estimates... Until here, we made our prediction models, obtaining very good **RMSE** values. The last one is **0.8641362**. But, to make sure we have reached the goal, we must run our final model with original edx and validation datasets.

2.3.5 Regularized movie and user effect model using original edx and validation datasets

Now, we can do our best model with these effects together, regularized with the originals edx and validation datasets:

```
##########################################################################################
# Regularizing Movie + User Effect Model using original edx and validation datasets.
# Computing the predicted ratings on validation dataset using the best value
# of lambda.

#########################################################
# final model RMSE results with edx and validation set
#########################################################

# Computing regularized estimates of b_i using best lambda

movie_avgs_reg <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu) / (n() + lambda), n_i = n())

# Computing regularized estimates of b_u using best lambda

user_avgs_reg <- edx %>%
  left_join(movie_avgs_reg, by ='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i) / (n() + lambda), n_u = n())
```

```r
# Predicting ratings

predicted_ratings <- validation %>%
  left_join(movie_avgs_reg, by = 'movieId') %>%
  left_join(user_avgs_reg, by = 'userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

# Predicting the RMSE with the final model on the validation set

model_final_rmse <- RMSE(predicted_ratings, validation$rating)

# Creating a results dataframe that will contains FINAL RMSE result.

Final_rmse_result <- data.frame(method = "Final Regularized Movie+User Effects Model",
                                RMSE = model_final_rmse)
```

Here is the resulting RMSE with our Final model and original edx and validation datasets:

```r
Final_rmse_result %>% knitr::kable()
```

method	RMSE
Final Regularized Movie+User Effects Model	0.8648177

3. Results

These are the results that we were able to achieve with our models, trained with train and test sets from splitted edx dataset and the Final Result of our best Model trained with ***edx*** and tested with **validation** datasets:

```r
# results from several models computed with train and test datasets

rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646844
Regularized Movie + User Effects Model	0.8641362

And here is the Final RMSE result from our FINAL Model:

```r
# result from our best Final Model computed with original edx and validation datasets

Final_rmse_result %>% knitr::kable()
```

method	RMSE
Final Regularized Movie+User Effects Model	0.8648177

The First RMSEs table shows an important improvement of the models through the different methods, using splitted edx dataset. As we saw, "Just the average model" give us the RMSE more than 1, which means we could miss the rating by one star. Incorporating 'Movie effect' and 'Movie + User effect', the model got better by ~12% and ~21% respectively.
The last one, the regularized one, have a great improvement:~23%.

We note a little difference between the same Final Model computed with train and test set splitted from edx dataset and on the other hand, computed with original edx and validation datasets: approx. +0.0006 (almost

none).
Now, We are sure that we have the best Final Model for this project.

Next, we see a comparison table between Predicted ratings with our Final model VS true ratings:

```
# comparing 10 first predictions with true ratings

versus <- cbind("Predicted Rating" = predicted_ratings[1:10],
                "True Rating" = validation$rating[1:10])
versus
```

```
##       Predicted Rating True Rating
##  [1,]         4.264512         5.0
##  [2,]         4.992708         5.0
##  [3,]         4.385029         5.0
##  [4,]         3.347321         3.0
##  [5,]         4.232387         2.0
##  [6,]         2.762940         3.0
##  [7,]         3.970675         3.5
##  [8,]         4.133987         4.5
##  [9,]         4.276113         5.0
## [10,]         3.305501         3.0
```

We saw deeper into the data revealing that some features have large effect on errors. So a regularization model was used to penalize that. We got a Final RMSE = 0.8648177, that is ~23% better from "Just the average".

4. Conclusion

As you see, **we reach our goal of RMSE < 0.8490 with the Regularized Movie + User effect Model**. So, **movieId** and **userId** are the most influential variables in rating prediction.

Remembering the regularization is to constrain the total variability of the effect sizes, **the penalized estimates provide a large improvement over the least squares estimates**.

In a next job, we could also consider genres and year of rating effect because we note a little variation through their values, but, is very, very little.

More advanced Machine Learning algorithms surely will improve the RMSE, but with my limited personal laptop, this was all I could do, because the Movielens dataset has a very large size, 10million observations. Also, It would also be interesting to try other machine learning systems such as Keras, Tensor Flow, neural networks, etc.

NOTE: If someone wants, there is **recommenderlab package**, you can see it here:

https://www.rdocumentation.org/packages/recommenderlab/versions/0.2-5

References

https://rafalab.github.io/dsbook

http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/

https://rstudio-pubs-static.s3.amazonaws.com/414477_1dbd5b3ef6854d35bef3402e987695a4.html

https://crantastic.org/packages/data-table

https://mran.microsoft.com/documents/rro/multithread#mt-bench

https://mran.microsoft.com/rro#resources