MovieLens Recommendation System

Alejandro Oscar BOFFA

28/3/2020

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 $less than 0.86490$. 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 and exploration, **the recommendation system builted on the train dataset (edx) are evaluated on the test dataset (validation) and chosen based on the Root Mean Squared Error that must be lower than 0.86490 (RMSE < 0.86490)**
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+year+genre 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 Movie + User Effects Model we already get RMSE = 0.8633660**, that is less than our priority goal RMSE = 0.86490, so we have met our goal from here. However, we seek to reduce this value testing other models like regularized ones. Our last **Movie+User+Year+Genre model achieve a RMSE = 0.8626426**

2. Methods and Analysis

(NOTE: if you want to run this project, feel free to do it, on my laptop, Intel I5+8gb ram+ssd 500gb, it took about 3 hours to run).

Our first step is create edx(train) and validation(test) 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|Thriller
##  3      1     292      5 838983421 Outbreak (1995)        Action|Drama|Sci-Fi|T~
##  4      1     316      5 838983392 Stargate (1994)        Action|Adventure|Sci-~
##  5      1     329      5 838983392 Star Trek: Generation~ Action|Adventure|Dram~
##  6      1     355      5 838984474 Flintstones, The (199~ Children|Comedy|Fanta~
##  7      1     356      5 838983653 Forrest Gump (1994)    Comedy|Drama|Romance|~
##  8      1     362      5 838984885 Jungle Book, The (199~ Adventure|Children|Ro~
##  9      1     364      5 838983707 Lion King, The (1994)  Adventure|Animation|C~
## 10      1     370      5 838984596 Naked Gun 33 1/3: The~ 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:

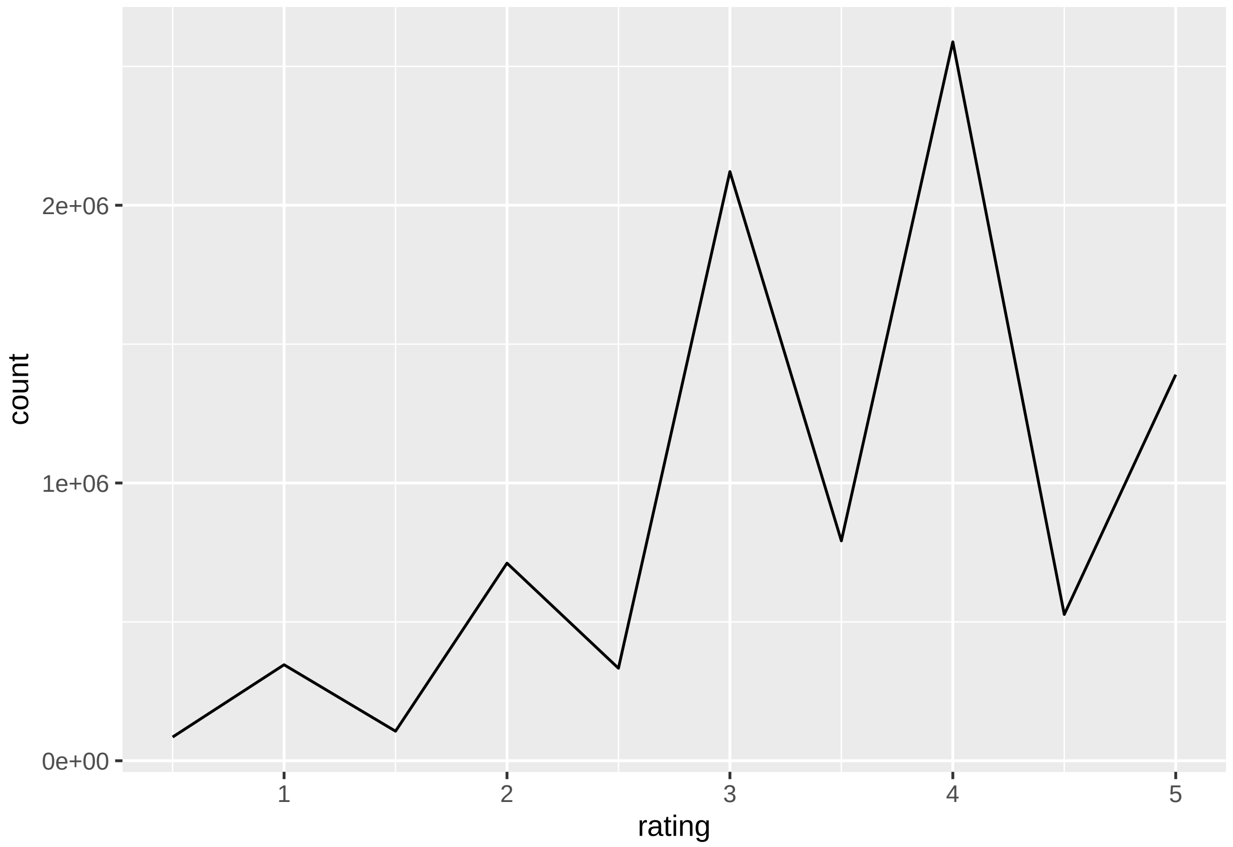
1. **userId**: the unique identifier for the user.
2. **movieId**: the unique identifier for the movie.
3. **rating**: a numeric rating between 0 and 5 for the movie in 0.5 increments.
4. **timestamp**: unix timestamp when the user rating was provided in seconds since Unix Epoch on January 1st, 1970 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:

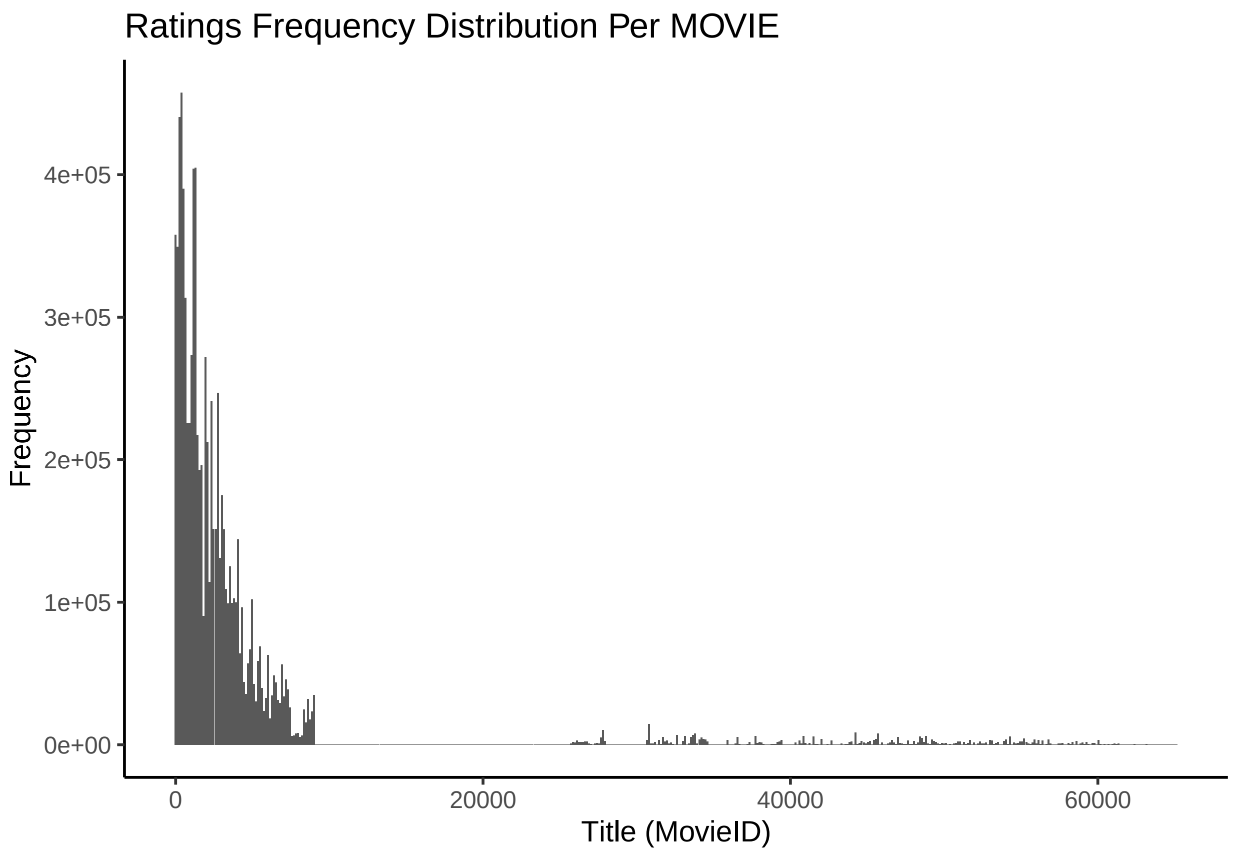
```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                        count
##      <dbl> <chr>                                                        <int>
##  1     296 Pulp Fiction (1994)                                          31362
##  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) (1977) 25672
## 10     150 Apollo 13 (1995)                                             24284
## # ... with 10,667 more rows
```

And the most given ratings and their distribution plot:

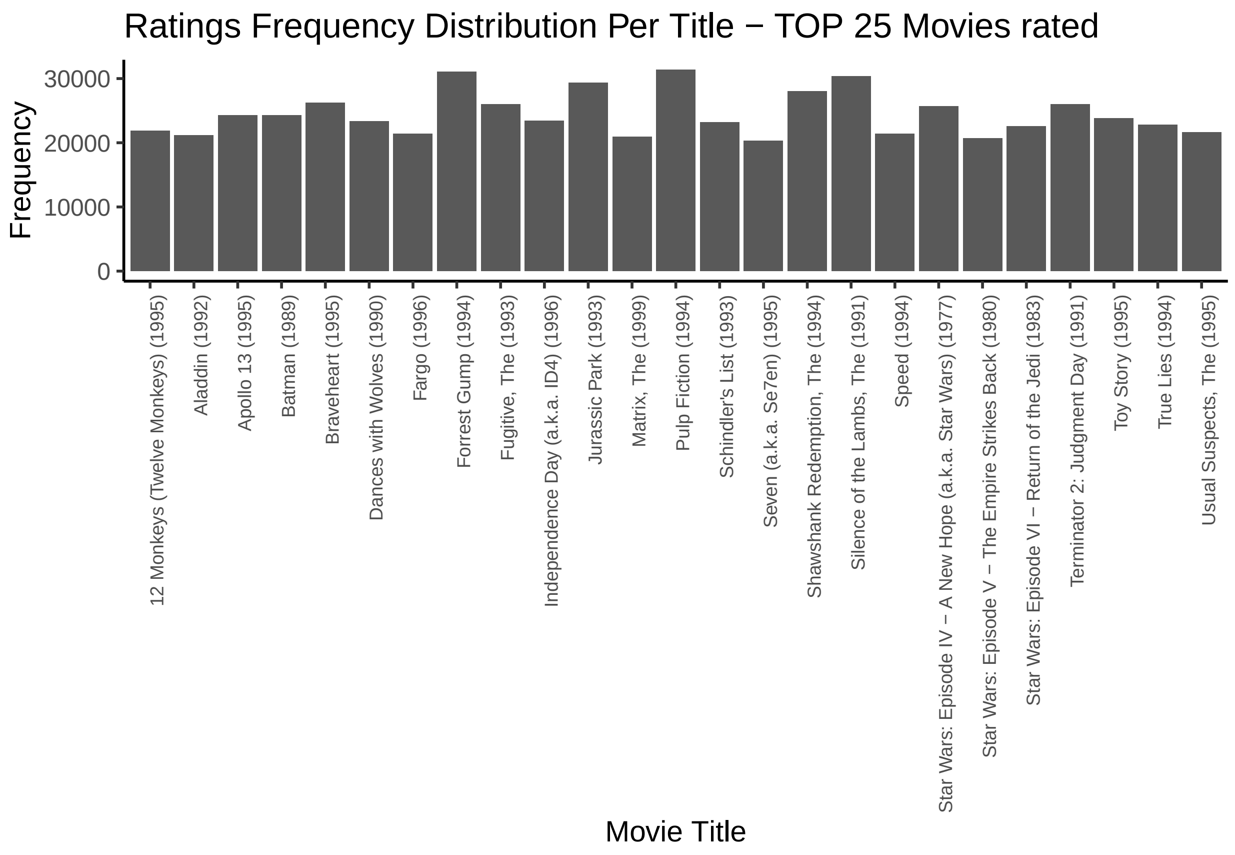
```
## # A tibble: 5 x 2
##   rating   count
##    <dbl>   <int>
## 1    4   2588430
## 2    3   2121240
## 3    5   1390114
## 4    3.5  791624
## 5    2    711422
```



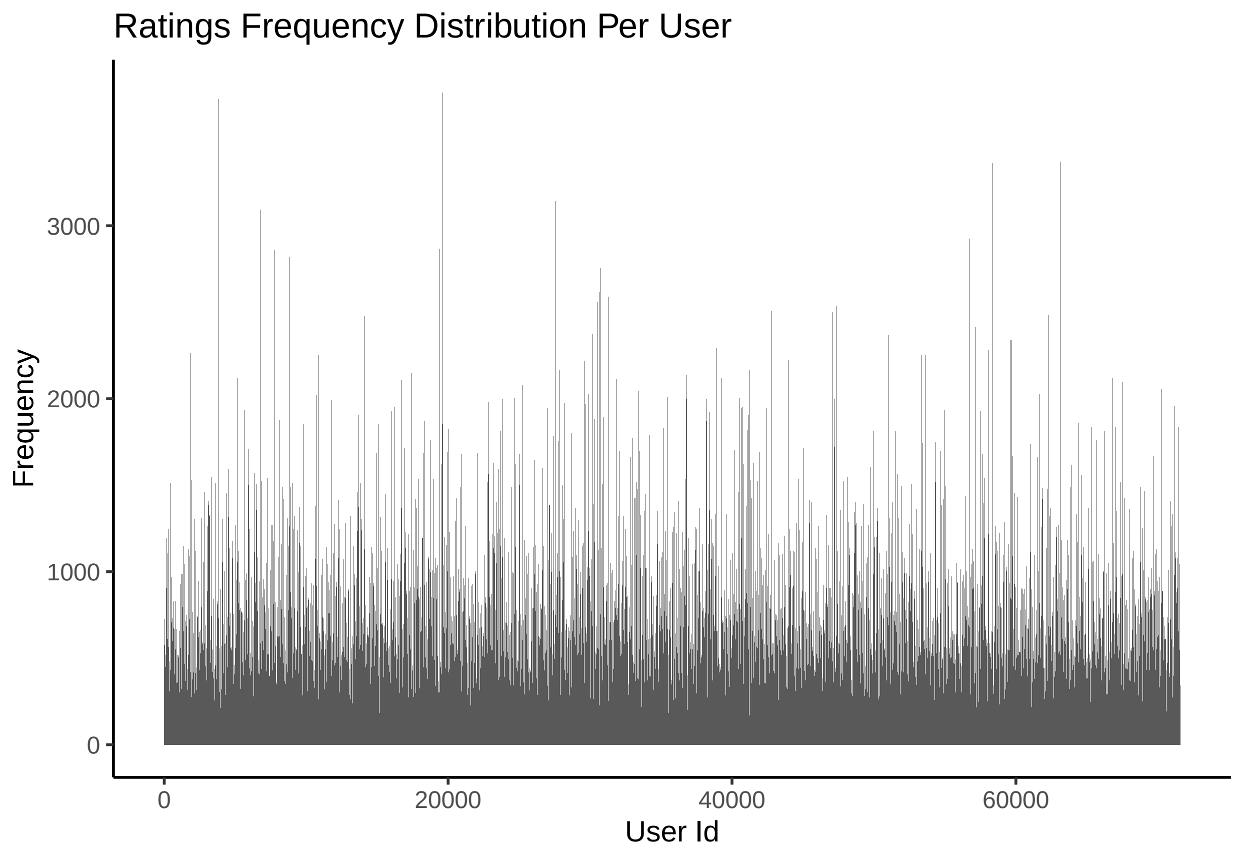
We can see in general, half star ratings are less than whole star ratings. There are fewer ratings of 3.5 or 4.5 than ratings of 3 or 4.
It´s important to explore how many rating per movie have this dataset.
We see the lowers movieId are the most rated:



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 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 and validation datasets in Data.Table. This was really faster an without error message.
Data tables work very fast with large datasets. Now, our source of data will be **edxDT and validationDT** Data Tables. 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:

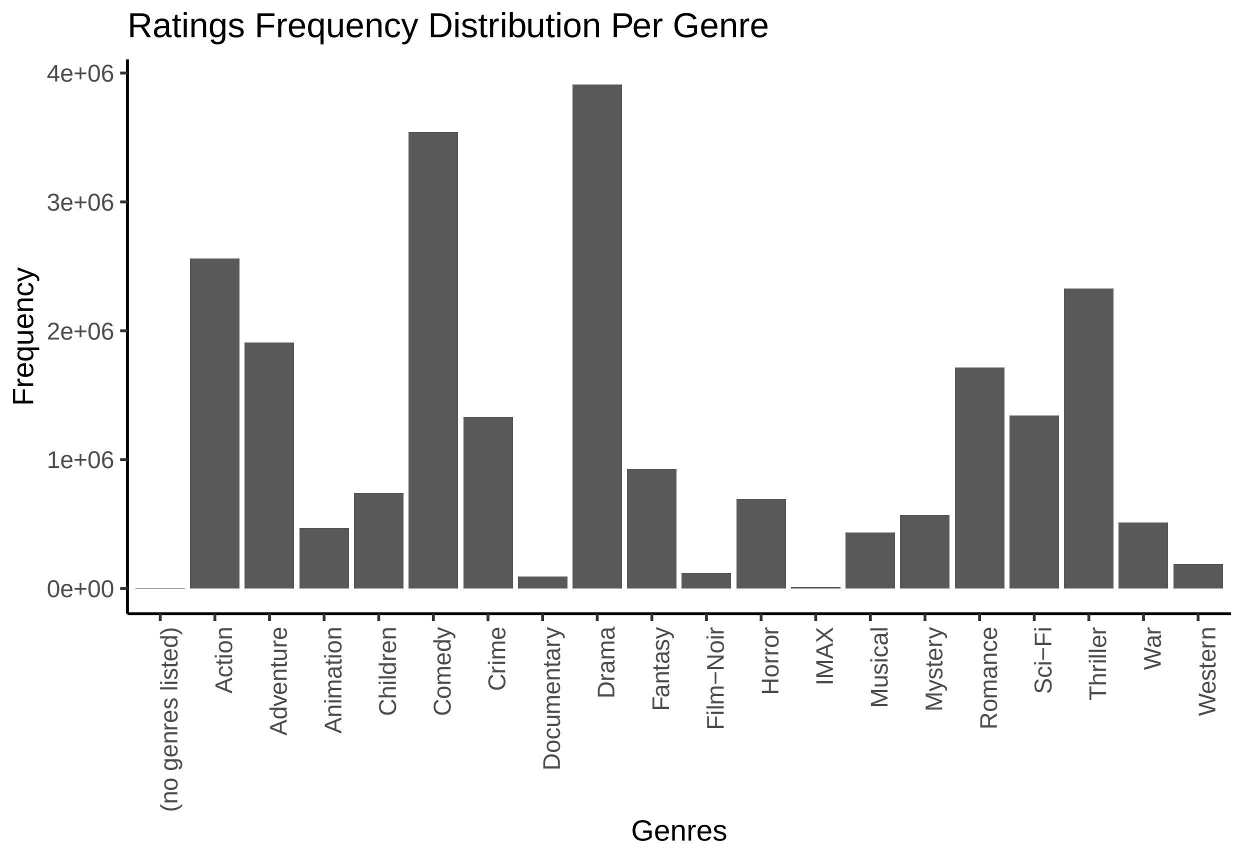
```
########################################################################
# Data Pre-processing

#########################################################
# Separate genre from datasets by "|"
#
# After 100 times trying separate_rows () on the edx Data Frame to separate the "genre"
# through its separator "|" and after hours of computing ~9 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,
```

```r
# I got the following simple solution
# (if you know a better one, please, let me know, thanks:)

# Convert dataframes edx and validation to "data table"-> edxDT and 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
```

```r
edxDT <- as.data.table(edx)
validationDT <- as.data.table(validation)

# 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" and "edx2". It took 2 hours:~24millon rows

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

# separate "genres" in "validation" data table

validationDT  <- validationDT  %>% separate_rows(genres, sep = "\\|") # dataset with ~2.6 million rows

# join both "edx_" data tables

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

# Extract yearofRating from timestamp in both datasets: edxDT and validationDT. We will use it as Year

edxDT <- as.data.table(edxDT %>%
                         mutate(yearofRating = as.integer(format(as.POSIXct(timestamp, origin="1970-01-
validationDT <- as.data.table(validationDT %>%
                                mutate(yearofRating = as.integer(format(as.POSIXct(timestamp, origin="1

# removing big obsolete datasets

rm(edx, edx1, edx2) # release ~2gb memory Ram!

# removing "timestamp" column from both datasets

edxDT <- select(edxDT, -timestamp)
validationDT <- select(validationDT, -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

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

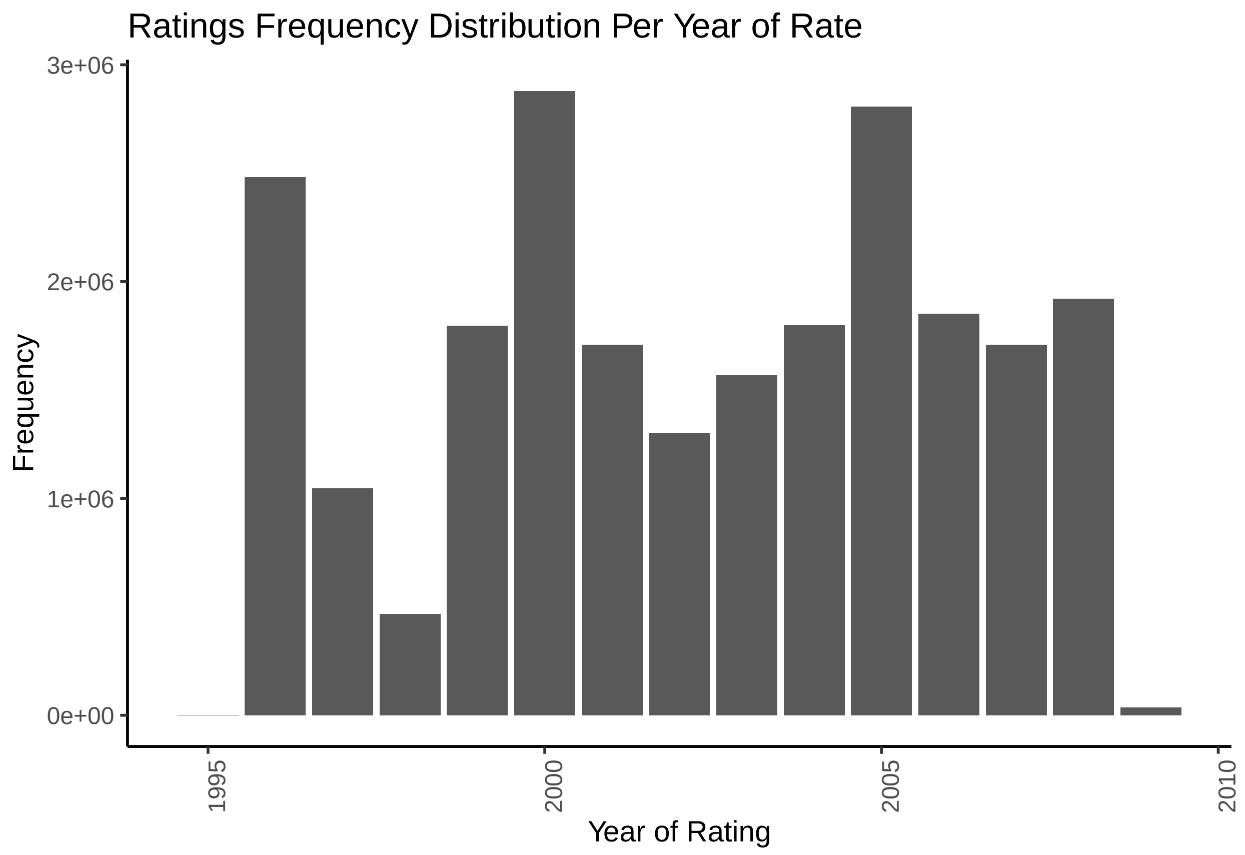
Now we can explore Distribution of Rating by Genre:



and we can count top 10 Separated Genres Rated:

```
## # A tibble: 10 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
```

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



2.3 Model building approach

As we see, movie and user are the principal effects to consider in our recommendation system project. Also we consider genres and year of rating effect because we note a variation through their values.
Later we´ll consider regularization method that permits us to penalize large estimates that are formed using small sample sizes.
Also, in this part of the report, we show the code used, so the solution is easy to understand.

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:

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

# Calculating "just the average" of all movies

mu <- mean(edxDT$rating)

# Calculating the RMSE on the validation set
```

```r
naive_rmse <- RMSE(validationDT$rating, mu)

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

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

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

method	RMSE
Just the average	1.052558

We get a RMSE of about 1. To win the 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.

```r
###################################
### Starting Movie Effect Model ###

# Calculating the average by movie

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

# Computing the predicted ratings on validation dataset

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

# Computing Movie effect model

model_1_rmse <- RMSE(predicted_ratings, validationDT$rating)

# Adding the results to the rmse_results table

rmse_results <- bind_rows(rmse_results,
                          data.table(method = "Movie Effect Model",
                                     RMSE = model_1_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.052558
Movie Effect Model	0.941070

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...

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

# Calculating the average by user

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

# Computing the predicted ratings on validation dataset

predicted_ratings <- validationDT %>%
  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, validationDT$rating)

# Adding the results to the results dataset

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

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

method	RMSE
Just the average	1.052558
Movie Effect Model	0.941070
Movie + User Effects Model	0.863366

Here we really make an improvement, from RMSE = 0.9410 to a RMSE = 0.8633. With that, we reach our goal RMSE < 0.86490. 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:

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

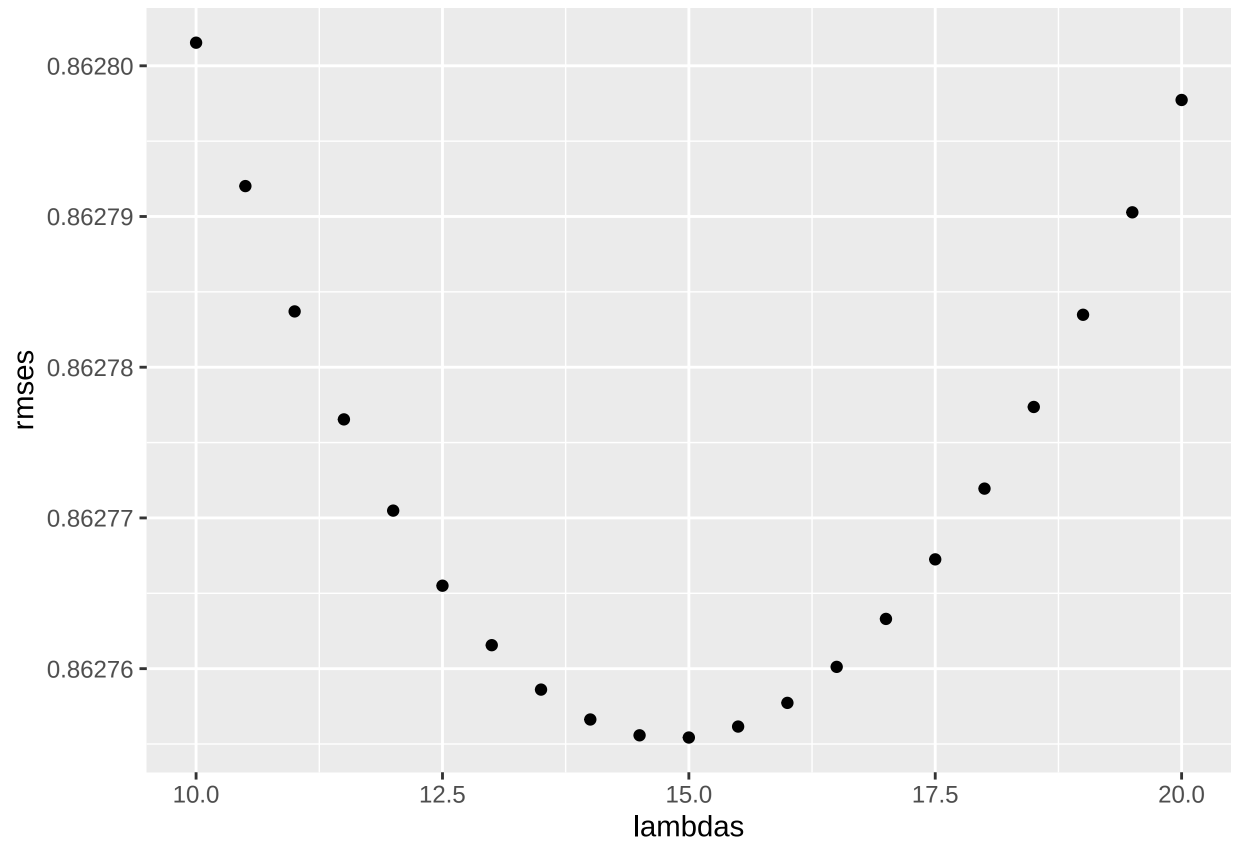
The first term is just least squares and the second is a penalty that gets larger when many b_i are large. Using calculus we can actually show that the values of bi that minimize this equation are:

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

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 validation 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.

# after testing several different intervals, I choose that one that minimize RMSEs.

lambdas <- seq(10, 20, 0.5)

# function rmses calculate predictions with several lambdas

mu <- mean(edxDT$rating)
rmses <- sapply(lambdas, function(l) {

  # Calculating the average by movie

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

  # Calculating the average by user

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

  # Computing the predicted ratings on validation dataset

  predicted_ratings <- validationDT %>%
    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 validation set
  return(RMSE(predicted_ratings, validationDT$rating))
})

# 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)] # choosing the best lambda

print("the best lambda is:")
```

```
## [1] "the best lambda is:"
```

```
lambda
```

```
## [1] 15
```

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

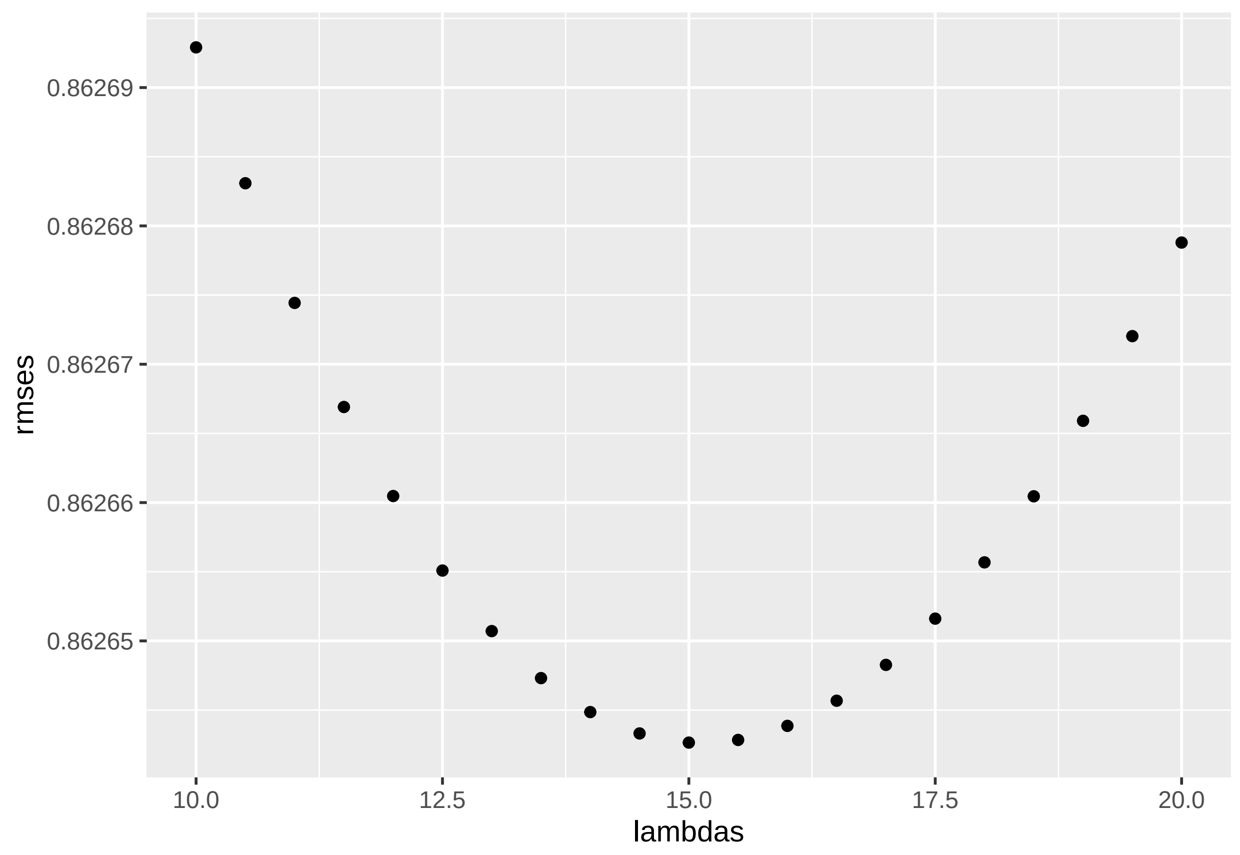
```
# Computing regularized estimates of b_u using best lambda

user_avgs_reg <- edxDT %>%
  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 <- validationDT %>%
  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 validation set

model_3_rmse <- RMSE(predicted_ratings, validationDT$rating)

# Adding the results to the rmse_results dataset

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

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

method	RMSE
Just the average	1.0525579
Movie Effect Model	0.9410700
Movie + User Effects Model	0.8633660
Regularized Movie + User Effects Model	0.8627554

The **penalized** estimates provide an improvement over the least squares estimates…

2.3.5 Regularized movie, user, genre and year of rating effect model Now, we can do our best model with all effects together, regularized:

```
#################################################################
# Regularizing Movie + User + YearofRating + Genres Effect Model.
# Computing the predicted ratings on validation dataset using different values
# of lambda, searching for the best one.
# b_i is the Movie effect and b_u is User effect.
# b_y is Year effect and b_g is Genre effects.

# after testing several different intervals, I choose that one that minimize RMSEs.

lambdas <- seq(10, 20, 0.5)

# function rmses calculate predictions with several lambdas

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

  # Calculating the average by movie
  b_i <- edxDT %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu) / (n() + l))

  # Calculating the average by user

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

  # Calculating the average by year of rating

  b_y <- edxDT %>%
    left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    group_by(yearofRating) %>%
    summarize(b_y = sum(rating - mu - b_i - b_u) / (n() + l), n_y = n())

  # Calculating the average by genre

  b_g <- edxDT %>%
    left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    left_join(b_y, by = 'yearofRating') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u - b_y) / (n() + l), n_g = n())

  # Computing the predicted ratings on validation dataset

  predicted_ratings <- validationDT %>%
    left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    left_join(b_y, by = 'yearofRating') %>%
    left_join(b_g, by = 'genres') %>%
    mutate(pred = mu + b_i + b_u + b_y + b_g) %>%
    .$pred

  return(RMSE(predicted_ratings, validationDT$rating))
})

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

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



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

print("the best lambda is:")
```

```
## [1] "the best lambda is:"
```

```r
lambda
```

```
## [1] 15
```

```r
# Compute regularized estimates of b_i (movie effect) using best lambda

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

# Compute regularized estimates of b_u (user effect) using best lambda

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

# Compute regularized estimates of b_y (year of rating effect) using best lambda

year_reg_avgs <- edxDT %>%
  left_join(movie_reg_avgs, by = 'movieId') %>%
  left_join(user_reg_avgs, by = 'userId') %>%
  group_by(yearofRating) %>%
  summarize(b_y = sum(rating - mu - b_i - b_u) / (n() + lambda), n_y = n())

# Compute regularized estimates of b_g (genre effect) using best lambda

genre_reg_avgs <- edxDT %>%
  left_join(movie_reg_avgs, by = 'movieId') %>%
  left_join(user_reg_avgs, by = 'userId') %>%
  left_join(year_reg_avgs, by = 'yearofRating') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - b_i - b_u - b_y) / (n() + lambda), n_g = n())

# Predict ratings

predicted_ratings <- validationDT %>%
  left_join(movie_reg_avgs, by = 'movieId') %>%
  left_join(user_reg_avgs, by = 'userId') %>%
  left_join(year_reg_avgs, by = 'yearofRating') %>%
  left_join(genre_reg_avgs, by = 'genres') %>%
  mutate(pred = mu + b_i + b_u + b_y + b_g) %>%
  .$pred

model_4_rmse <- RMSE(predicted_ratings, validationDT$rating)

# Adding the results to the rmse_results dataset

rmse_results <- bind_rows(rmse_results,
                          data.table(method="Reg Movie + User + Year + Genre Effect Model",
                                     RMSE = model_4_rmse))

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

method	RMSE
Just the average	1.0525579
Movie Effect Model	0.9410700
Movie + User Effects Model	0.8633660
Regularized Movie + User Effects Model	0.8627554
Reg Movie + User + Year + Genre Effect Model	0.8626426

Until here, we made our prediction models, obtaining very good **RMSE** values. The last one is **0.8626426**

3. Results

These are the results that we were able to achieve with our models, trained with ***edx*** and tested with **validation** datasets:

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

method	RMSE
Just the average	1.0525579
Movie Effect Model	0.9410700
Movie + User Effects Model	0.8633660
Regularized Movie + User Effects Model	0.8627554
Reg Movie + User + Year + Genre Effect Model	0.8626426

The RMSEs table shows an important improvement of the models through the different methods. 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 ~11% and ~22% respectively. The last one is a great improvement. 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 = 0.8626426, that is ~23% better from "Just the average".

4. Conclusion

As we see, **we reach our goal of RMSE < 0.8490 with the Movie + User effect Model**. So, **movieId** and **userId** are the most influential variables in rating prediction. But we decided make predictions with Regularized models, getting a very good value of **RMSE= 0.862642 with Regularized Movie + User + Year + Genre effect Model**. 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**.

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