Movielens_proyect

David Jimeno

14 de noviembre de 2019

MovieLens Project Submission

Introduction

Our analysis focuses on the movielens database, which is a database of movie ratings, in which each user with their id, scores the movies on a specific date collected in a timestamp, these are identified by genre and date of premiere.

The total dimension of movielens is composed of 10000054 observations of 6 variables.

The objective of the project is to create a Recommendation systems use ratings that users have given items to make specific recommendations predicting how many stars a user will give a specific movie. The project improve our recommendation algorithm minimizing the RMSE.

A study is carried out of the different effects that the variables may have to incorporate them into a final algorithm that reduces our RMSE.

The document is structured as follows:

-Preprocessing, exploration and visualization of the database.

-Exercises.

-Application of methods and analysis.

-Results.

-Conclusions.

Preprocessing, exploration and database visualization.

Our analysis have started with a preprocessing already done and we have added the obtaining variable movie premiere year (year) and scored film date (date) this has allowed us to work with median number of ratings, movies average rating, the trend movie rated and its average rating, time effect on average rating and genre average rating.

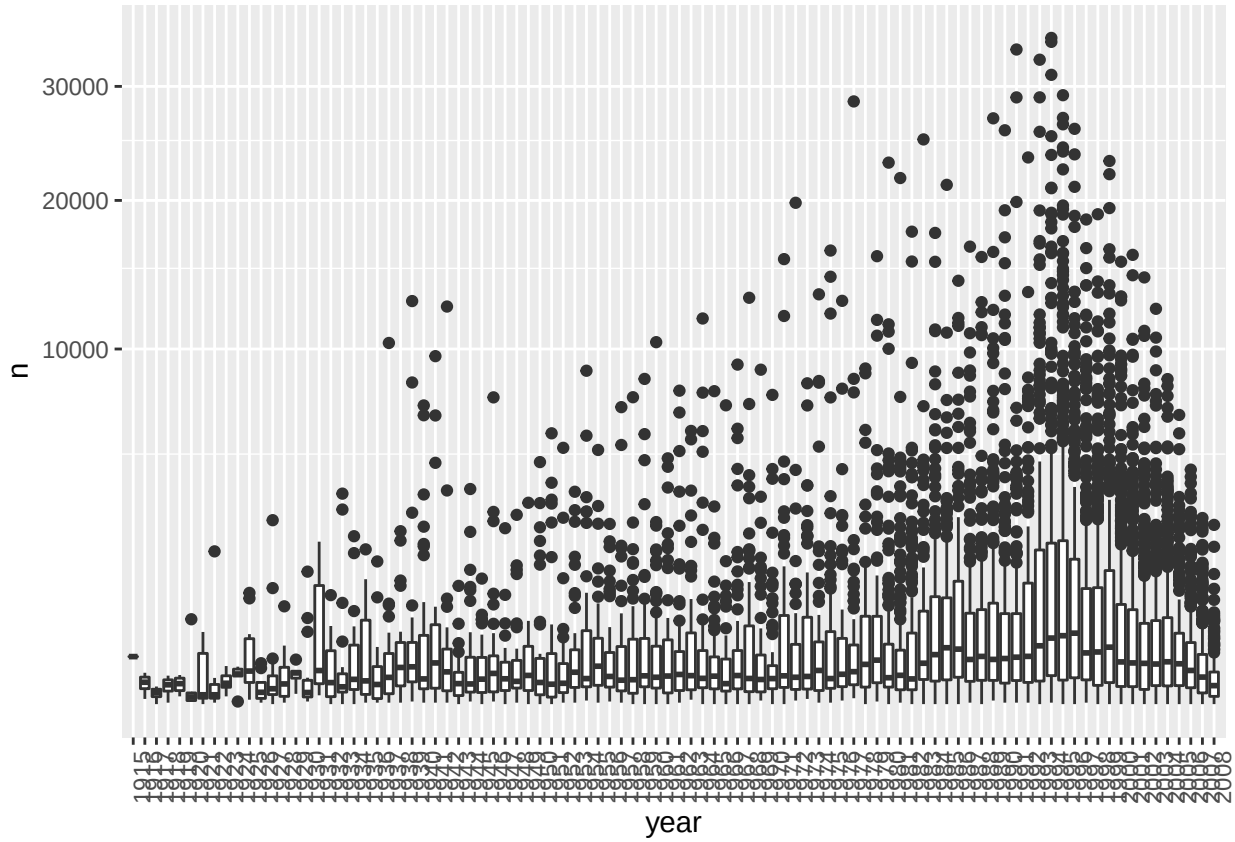
```
################################
# Create edx set, validation set
################################

# Note: this process could take a couple of minutes

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")

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
```

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

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

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

```
## 'data.frame':    10000054 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983653 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller"
```

```
##Data exploration and visualization

#Lets look to the year with highest median number of ratings
year<-gsub("[^0-9]", "", movielens$title)
library(stringi)
year <- stri_extract_last_regex(year, "\\d{4}$")
year<- as.numeric(year)
movielens1 <-movielens%>% mutate(year)
```

```
movielens1%>% group_by(movieId) %>%
  summarize(n = n(), year = as.character(first(year))) %>%
  qplot(year, n, data = ., geom = "boxplot") +
  coord_trans(y = "sqrt") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



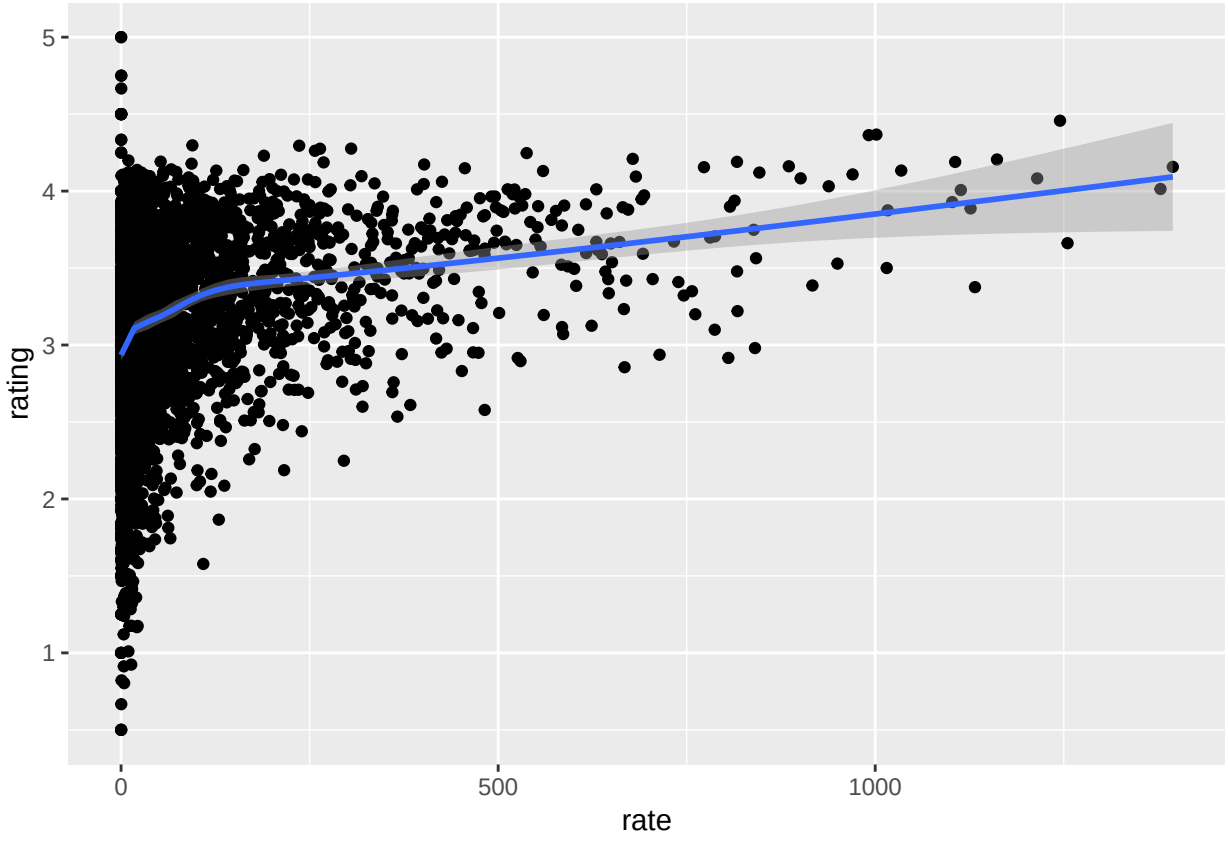

```
#movies after 1993 get more
#ratings. We also see that with newer movies, starting in 1993, the
#number of ratings decreases with year: the more recent a movie is, the
#less time users have had to rate it.
```

```
#Among movies that came out in 1993 or later, here are the 10 movies
#with the most ratings per year, and its average rating of each
#of the top 10 movies.

movielens1 %>%
  filter(year >= 1993) %>%
  group_by(movieId) %>%
  summarize(n = n(), years = 2019 - first(year),
            title = title[1],
            rating = mean(rating)) %>%
  mutate(rate = n/years) %>%
  top_n(10, rate) %>%
  arrange(desc(rate))
```

```
## # A tibble: 10 x 6
##    movieId     n years title                                rating  rate
##      <dbl> <int> <dbl> <chr>                                 <dbl> <dbl>
##  1     296 34864    25 Pulp Fiction (1994)                    4.16 1395.
##  2     356 34457    25 Forrest Gump (1994)                    4.01 1378.
##  3     480 32631    26 Jurassic Park (1993)                   3.66 1255.
##  4     318 31126    25 Shawshank Redemption, The (1994)       4.46 1245.
##  5     110 29154    24 Braveheart (1995)                      4.08 1215.
```

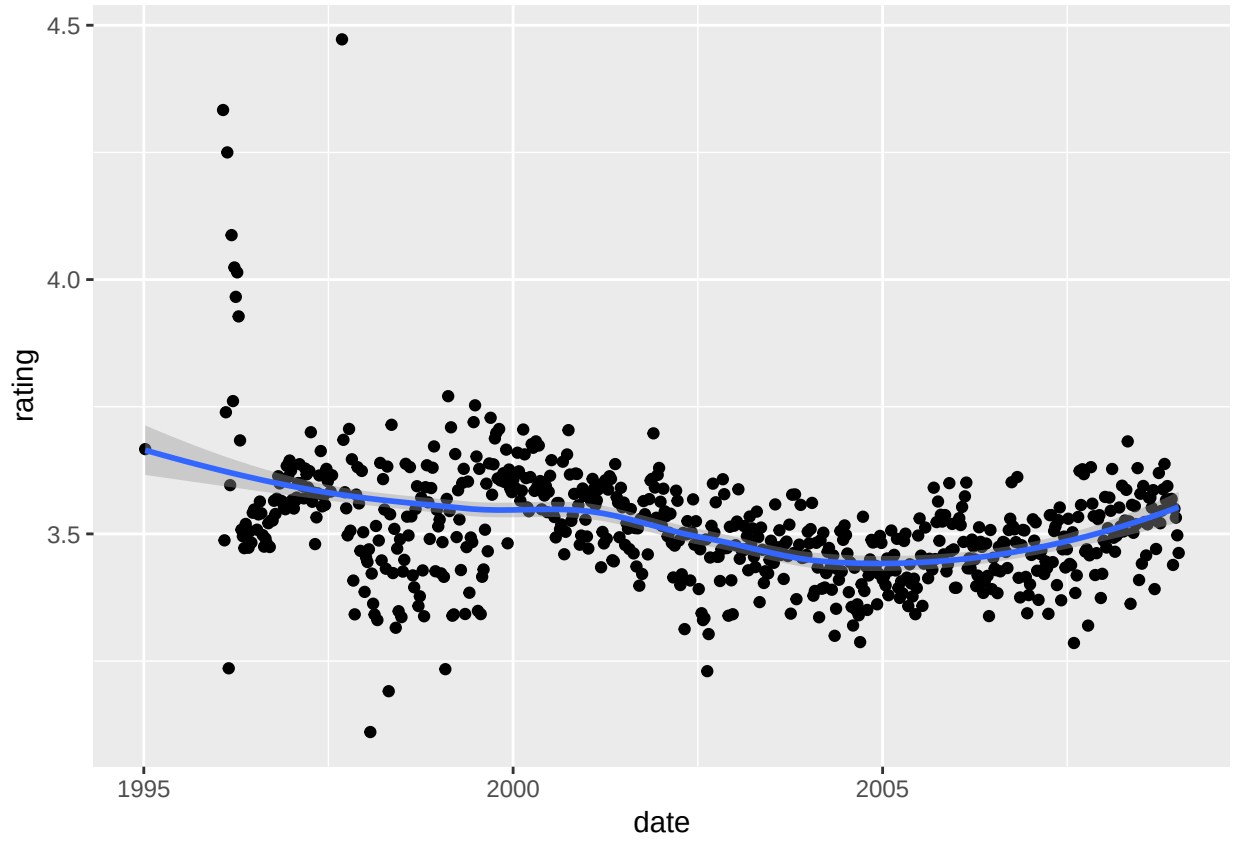
```
##  6    2571 23229    20 Matrix, The (1999)                       4.21 1161.
##  7     780 26042    23 Independence Day (a.k.a. ID4) (1996)     3.38 1132.
##  8     150 27035    24 Apollo 13 (1995)                         3.89 1126.
##  9     457 28951    26 Fugitive, The (1993)                     4.01 1114.
## 10    2858 22120    20 American Beauty (1999)                   4.19 1106
```

```
#We see that the trend is that the more often a movie is rated, the higher
# its average rating
movielens1 %>%
  filter(year >= 1993) %>%
  group_by(movieId) %>%
  summarize(n = n(), years = 2019 - first(year),
            title = title[1],
            rating = mean(rating)) %>%
  mutate(rate = n/years) %>%
  ggplot(aes(rate, rating)) +
  geom_point() +
  geom_smooth()
```

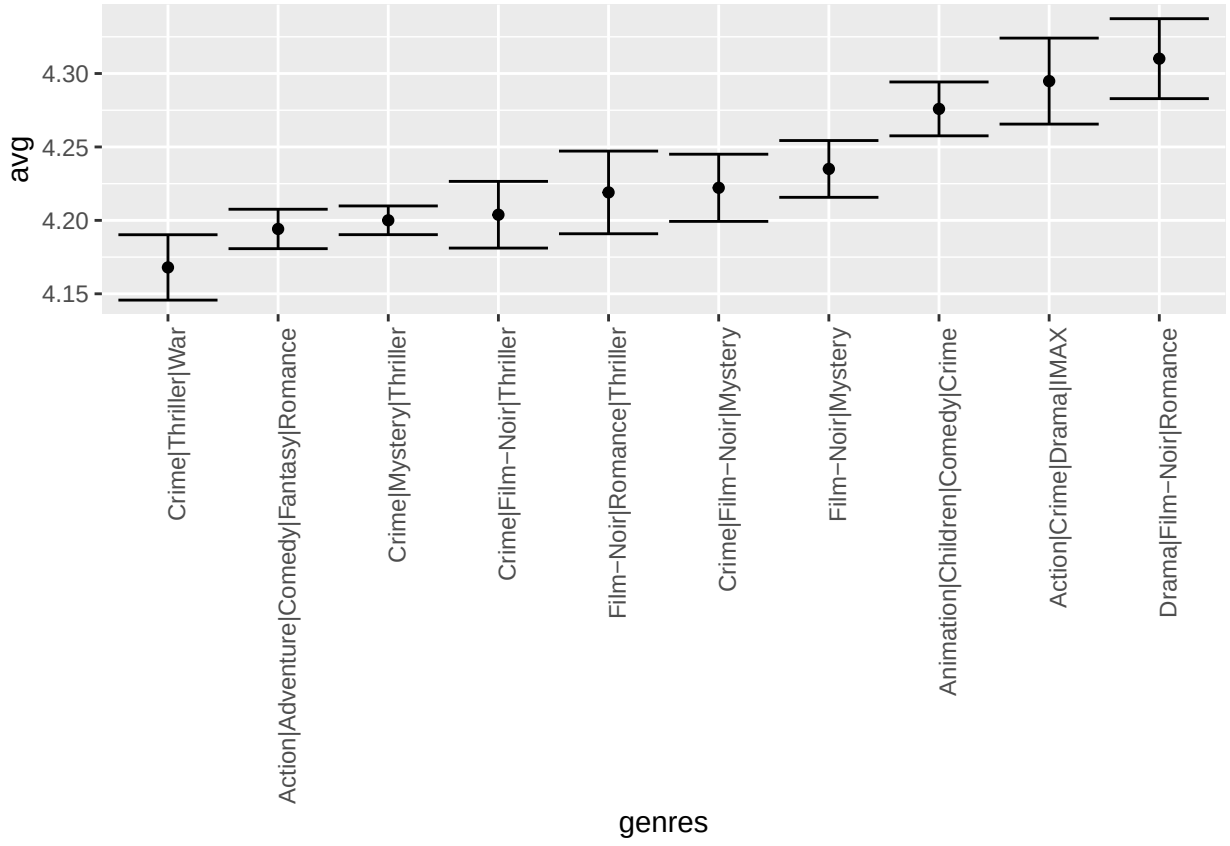



```
#There is some evidence of a time effect on average rating

library(lubridate)
movielens2 <- mutate(movielens, date = as_datetime(timestamp))
movielens2 %>%
  mutate(date = round_date(date, unit = "week")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
```

```
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth()
```




```
#Keeping only categories with more than 1,000 ratings Drama|Film-Noir|Romance
# genre has the highest average rating.

movielens2 %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 1000) %>%
  mutate(genres = reorder(genres, avg)) %>%top_n(10, avg)%>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```




```
# Validation set will be 10% of MovieLens data

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

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

Exercises.

```
#Quiz: MovieLens Dataset
#Q1 How many rows and columns are there in the edx dataset?
dim(edx)
```

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

```
#Number of rows:9000055 Number of columns:6
```

```
#Q2 How many zeros were given as ratings in the edx dataset?
sum(edx$rating == 0)
```

```
## [1] 0
```

```
#How many threes were given as ratings in the edx dataset?
sum(edx$rating == 3)
```

```
## [1] 2121240
```

```
#Q3 How many different movies are in the edx dataset?
n_distinct(edx$movieId)
```

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

```
#Q4 How many different users are in the edx dataset?
n_distinct(edx$userId)
```

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

```
#Q5 How many movie ratings are in each of the following genres in the edx dataset?
drama <- edx %>% filter(str_detect(genres,"Drama"))
comedy <- edx %>% filter(str_detect(genres,"Comedy"))
thriller <- edx %>% filter(str_detect(genres,"Thriller"))
romance <- edx %>% filter(str_detect(genres,"Romance"))
nrow(drama)
```

```
## [1] 3910127
```

```
nrow(comedy)
```

```
## [1] 3540930
```

```
nrow(thriller)
```

```
## [1] 2325899
```

```
nrow(romance)
```

```
## [1] 1712100
```

```
#Q6 Which movie has the greatest number of ratings?
edx %>% group_by(title) %>% summarise(number = n()) %>%
  arrange(desc(number))
```

```
## # A tibble: 10,676 x 2
##    title                                                     number
##    <chr>                                                      <int>
##  1 Pulp Fiction (1994)                                        31362
##  2 Forrest Gump (1994)                                        31079
##  3 Silence of the Lambs, The (1991)                           30382
##  4 Jurassic Park (1993)                                       29360
##  5 Shawshank Redemption, The (1994)                           28015
##  6 Braveheart (1995)                                          26212
##  7 Fugitive, The (1993)                                       25998
##  8 Terminator 2: Judgment Day (1991)                          25984
##  9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)  25672
## 10 Apollo 13 (1995)                                           24284
```

```
## # ... with 10,666 more rows
```

```
#Answer Pulp Fiction
```

```
#Q7 What are the five most given ratings in order from most to least?
edx%>%group_by(rating)%>% summarise(number = n()) %>%
  arrange(desc(number))
```

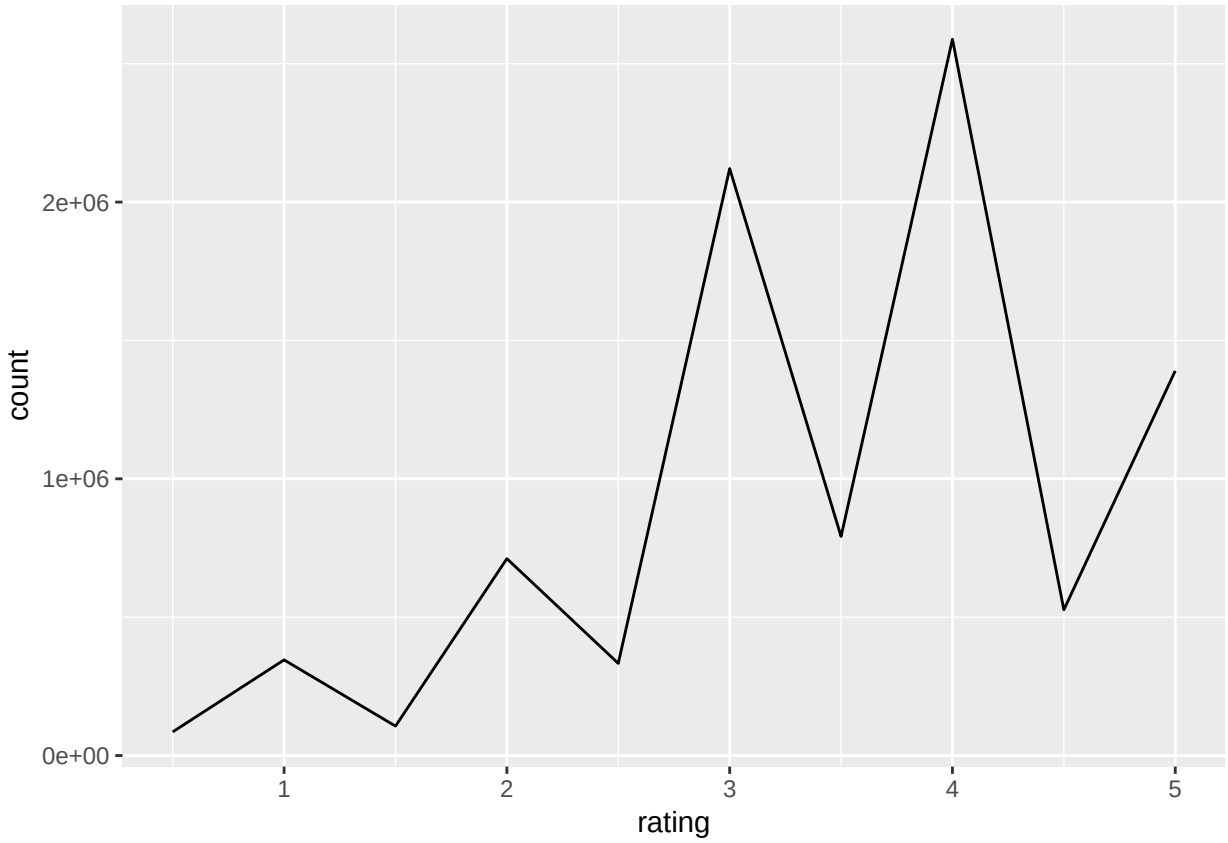
```
## # A tibble: 10 x 2
##    rating  number
##     <dbl>   <int>
##  1    4   2588430
##  2    3   2121240
##  3    5   1390114
##  4    3.5  791624
##  5    2    711422
##  6    4.5  526736
##  7    1    345679
##  8    2.5  333010
##  9    1.5  106426
## 10    0.5   85374
```

```
#Answer 4,3,5,3.5,2
```

```
#Q8 True or False: In general, half star ratings are less common than whole star
# ratings (e.g., there are fewer ratings of 3.5 than there are ratings of 3 or 4
# , etc.).
table(edx$rating)
```

```
##
##     0.5       1     1.5       2     2.5       3     3.5       4     4.5
##   85374  345679  106426  711422  333010 2121240  791624 2588430  526736
##       5
## 1390114
```

```
edx %>%
  group_by(rating) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = rating, y = count)) +
  geom_line()
```




```
#Answer True
```

Application of methods and analysis.

Recommendation systems is a type of machine learning algorithm that will be applied outside our control, as users look for movie recommendations.Our algorithm is decided based on the residual mean squared error (RMSE) on a test set, Once the algorithm is decided, we will apply it to the validation set.

We create an aditional partition of training and test sets from the provided edx dataset to experiment with multiple parameters.We will first apply the mean rating for all movies regardless of user and then add terms that represent average ranking for movie i (bi), average rating for user u, we use subsequently regularization to penalize large estimates using cross-validation choosing the penalty terms, finally we use regularized Movie + User Effect Mode.

```
#We can see this table is in tidy format with 9,000,055 rows:
edx %>% as_tibble()
```

```
## # 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
```

```
#Each row represents a rating given by one user to one movie.

#Creating new data Partition.
#Test set will be 10% of Edx data
set.seed(6, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
temp2 <- edx[test_index,]

# We make sure userId and movieId in test set and validation set are also in the training set

test_set <- temp2 %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")


# Add rows removed from test set back into train set

removed <- anti_join(temp2, test_set)
train_set <- rbind(train_set, removed)

rm(test_index, temp2, removed)


#Lets write a 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))
}

#First model:
#we predict the same rating for all movies regardless of user.
#We know that the estimate that minimizes the RMSE is the least squares estimate in this case, is
#the average of all ratings:
mu_hat <- mean(train_set$rating)
mu_hat
```

```
## [1] 3.512348
```

```
#If we predict all unknown ratings with mu_hat we obtain the following RMSE:
naive_rmse <- RMSE(test_set$rating, mu_hat)
naive_rmse
```

```
## [1] 1.059535
```

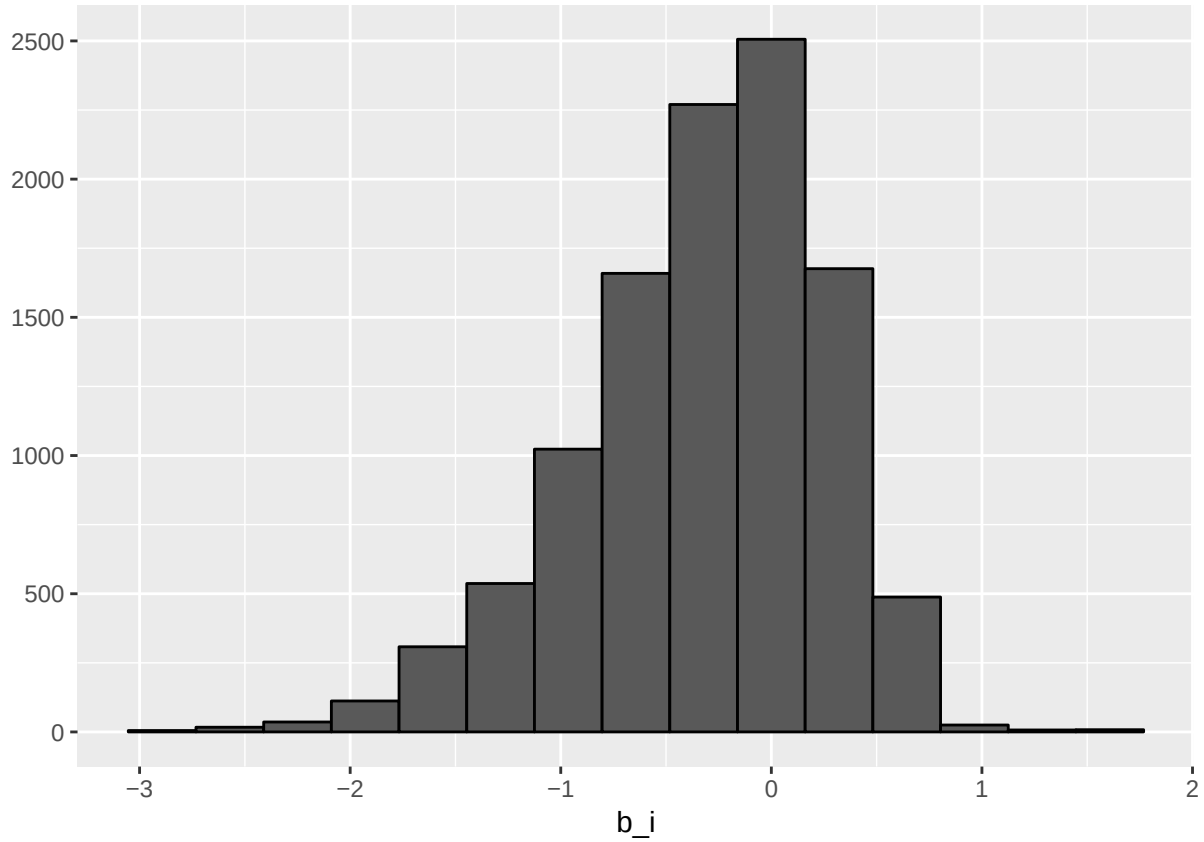
```
#Lets create a results comparing table for different approaches.
rmse_results <- data_frame(method = "Just the average", RMSE = naive_rmse)
rmse_results
```

```
## # A tibble: 1 x 2
```

```
##   method            RMSE
##   <chr>            <dbl>
## 1 Just the average  1.06
```

```
#Second model:
#We augment our previous model by adding the term  bi bias to represent
#average ranking for movie i.

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

```
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 15, data = ., color = I("black"))
```

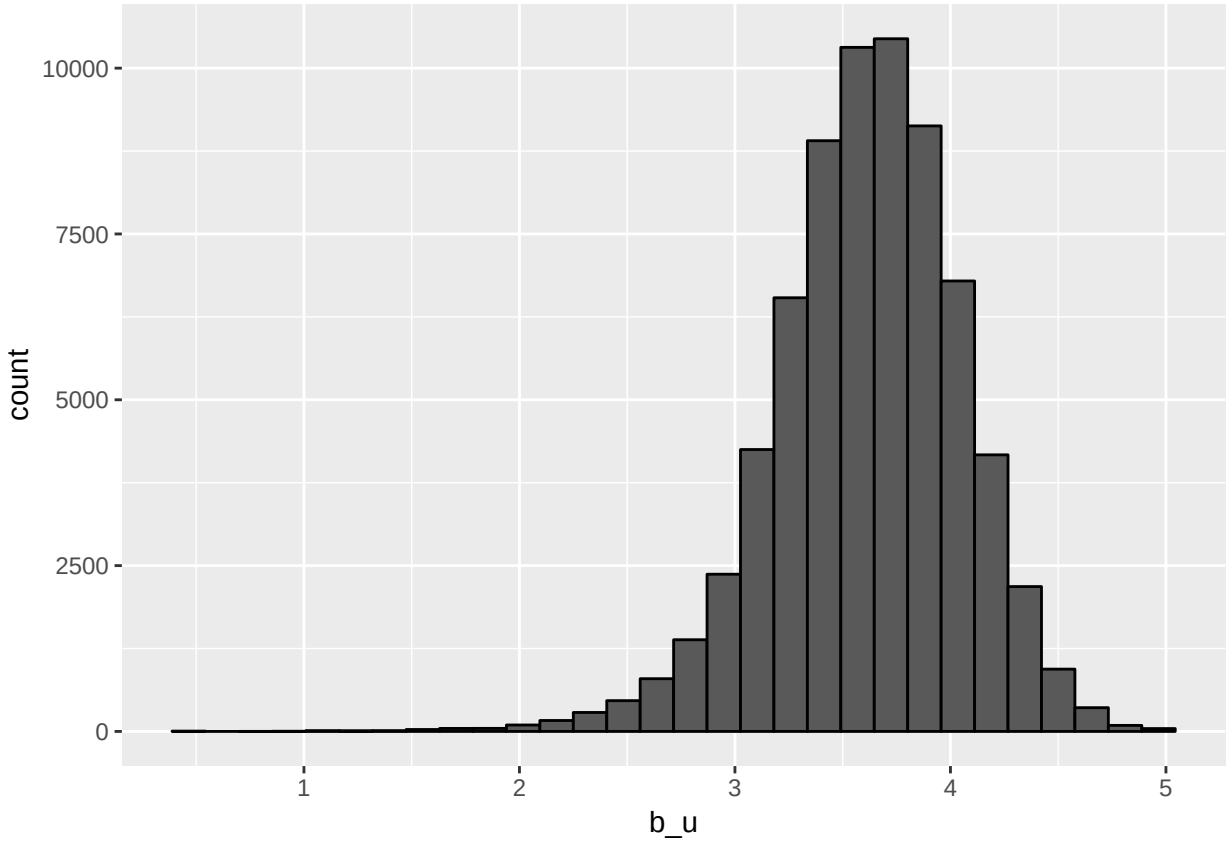



```
# Remember  mu_hat=3.5 so a  bi=1.5 implies a perfect five star rating.
#Lets see how much prediction improves using bi
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = model_2_rmse))
rmse_results
```

```
## # A tibble: 2 x 2
```

```
##   method              RMSE
##   <chr>              <dbl>
## 1 Just the average   1.06
## 2 Movie Effect Model 0.943
```

```r
#Third model. User effects
#Lets compute the average rating for user  u for those that have rated over 200 movies:
train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=200) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```




```r
#Notice that there is substantial variability across users as well.
```

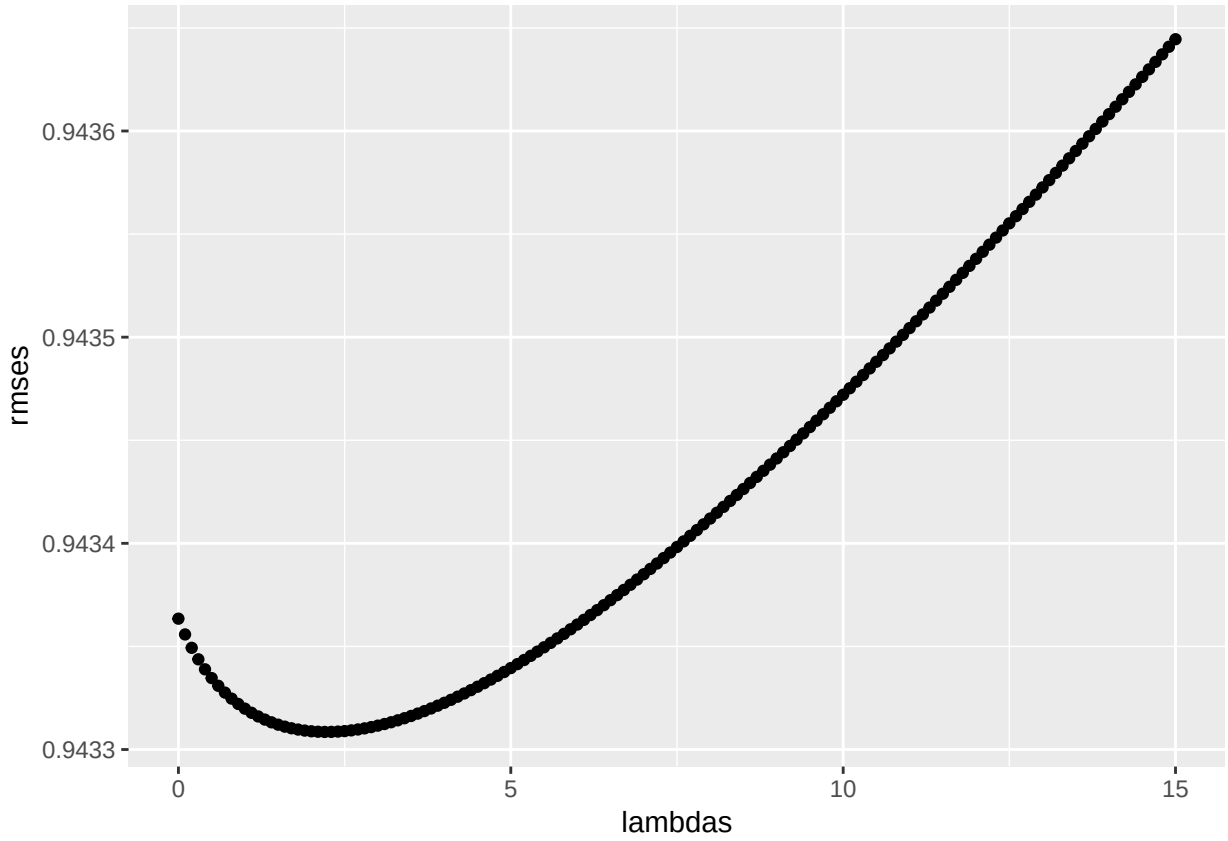
```r
#To fit this model for the reasons of computing time, we will compute an approximation of lm by
#computing
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
#We construct predictors and see how RMSE improves:
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
```

```
  pull(pred)


model_3_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effects Model",
                                     RMSE = model_3_rmse))
rmse_results
```

```
## # A tibble: 3 x 2
##   method                     RMSE
##   <chr>                     <dbl>
## 1 Just the average          1.06
## 2 Movie Effect Model        0.943
## 3 Movie + User Effects Model 0.865
```

```
# Fourth model, Regularized Movie Effect Model
#we use regularization that permits us to penalize large estimates that are formed using small
#sample sizes.

#We use cross-validation choosing the penalty terms (lambdas).
lambdas <- seq(0, 15, 0.10)

mu <- mean(train_set$rating)
just_the_sum <- train_set %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())

rmses <- sapply(lambdas, function(l){
  predicted_ratings <- test_set %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})
```

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



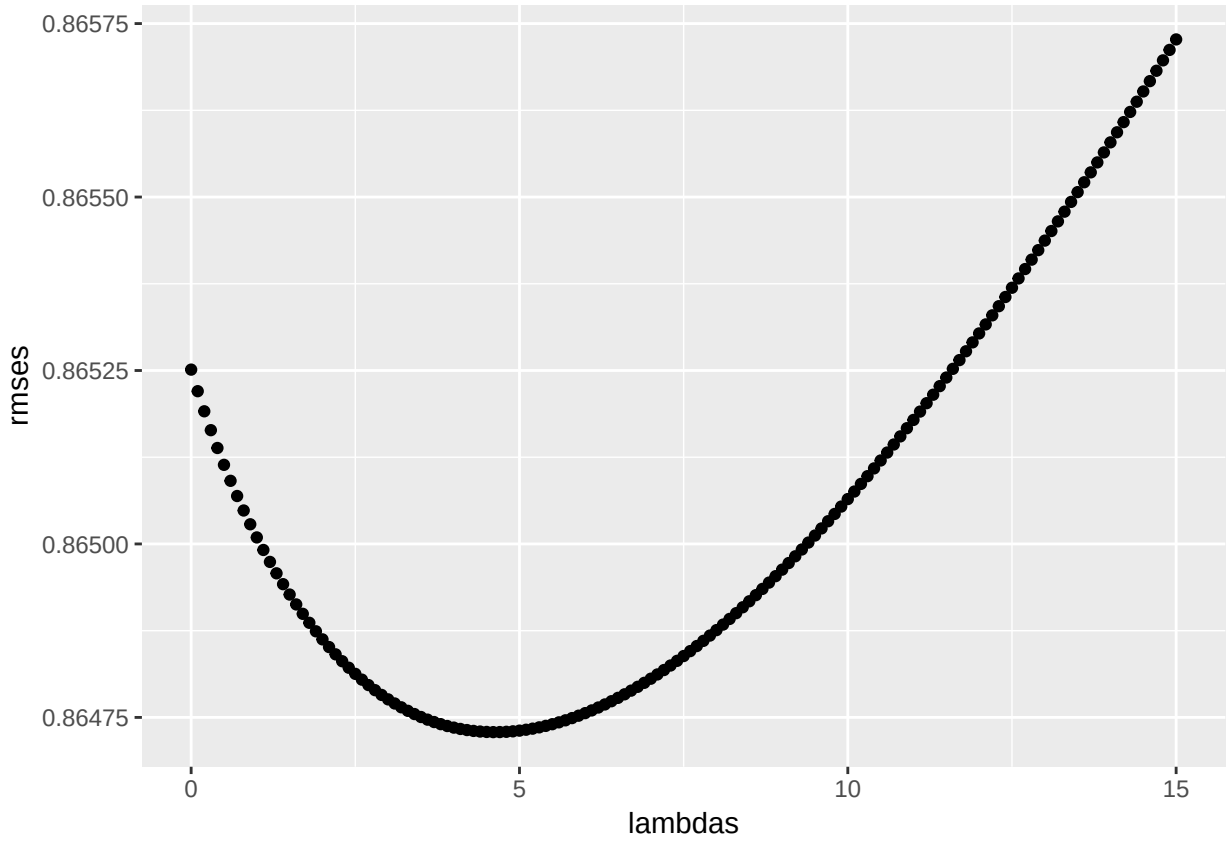

```
lambdas[which.min(rmses)]
```

```
## [1] 2.2
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie Effect Model",
                                     RMSE = min(rmses)))

#We use regularization for the estimate user effects.
#The estimates that minimize this can be found using cross-validation to pick a  lambda:
lambdas <- seq(0, 15, 0.10)

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

  mu <- mean(train_set$rating)

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

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

  predicted_ratings <-
    test_set %>%
```

```
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

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

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




```
#The optimal  lambda is:
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 4.6
```

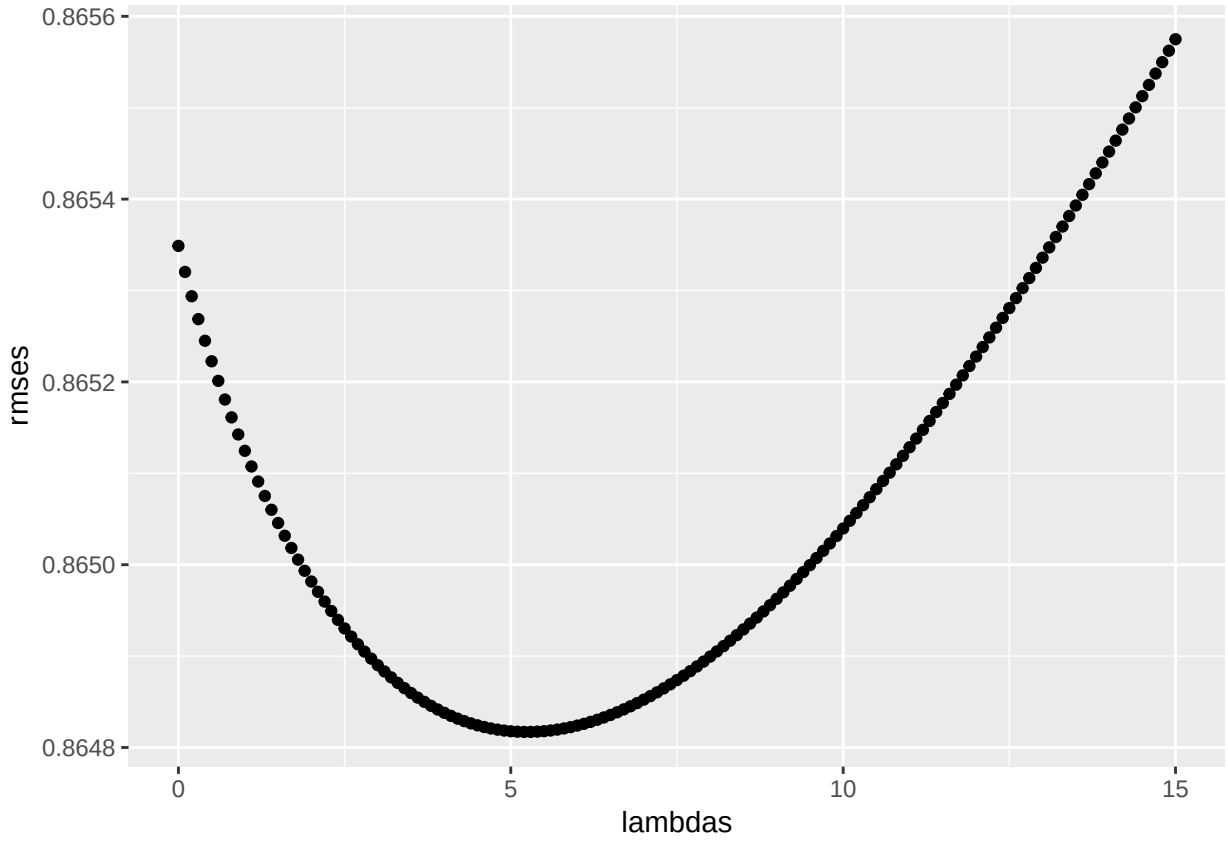
```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie + User Effect Model",
                                     RMSE = min(rmses)))
rmse_results
```

```
## # A tibble: 5 x 2
##   method                              RMSE
##   <chr>                              <dbl>
## 1 Just the average                   1.06
## 2 Movie Effect Model                 0.943
## 3 Movie + User Effects Model         0.865
## 4 Regularized Movie Effect Model     0.943
```

5 Regularized Movie + User Effect Model 0.865

```
#We used validation set to test of the final model Regularized Movie + User Effect Model.

lambdas <- seq(0, 15, 0.10)

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

  mu <- mean(edx$rating)

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

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

  predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

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

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




```
#the optimal  lambda is:
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 5.2
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie + User Effect Model (validation set)",
                                     RMSE = min(rmses)))

rmse_results
```

```
## # A tibble: 6 x 2
##   method                                              RMSE
##   <chr>                                              <dbl>
## 1 Just the average                                   1.06
## 2 Movie Effect Model                                 0.943
## 3 Movie + User Effects Model                         0.865
## 4 Regularized Movie Effect Model                     0.943
## 5 Regularized Movie + User Effect Model              0.865
## 6 Regularized Movie + User Effect Model (validation set) 0.865
```

Results

The report finds the model Regularized Movie + User Effect Model as the one that minimize RMSE with a 0.8647288 applied to the test set and a 0.8648170 to validation set.

Conclusions.

When we take into account more variables our conclusions are more precise.Probably further investigation using recommenderlab package give us a new model.