capston- project

H. Almustafa

2022-09-17

Introduction:

Recommendation systems use ratings that users have given items to make specific recommendations. Here we provide the basics of how these recommendations are predicted,motivated by some of the approaches taken by the winners of the Netflix challenge.

my Github repo

Methods :

after downloading the data i,m going to explore it, extract the Information from it ,then split the data into separate training and test sets to design and test my algorithm. then i will use the data to build and train a machine learning algorithm , using the inputs in one subset to predict Movie ratings in the validation set,by building a several models and comparing then using root mean squared error RMSA as loss function.

data loding

Create Train and Final Hold-out Test Sets:

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.7     v dplyr   1.0.8
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

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

```
##
## Attaching package: 'data.table'
##
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
##
## The following object is masked from 'package:purrr':
##
##     transpose
```

```
 library(dplyr)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
##
## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
##
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
dl <- tempfile()
download.file("https://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")


# if using R 4.0 or later:

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")

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

```
set.seed(1)
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)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
edx <- rbind(edx, removed)


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

Data exploration :

```
edx %>% as_tibble()
```

```
## # A tibble: 9,000,061 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|T~
##  3      1     231      5 838983392 Dumb & Dumber (1994)         Comedy
##  4      1     292      5 838983421 Outbreak (1995)              Action|Drama|S~
##  5      1     316      5 838983392 Stargate (1994)              Action|Adventu~
##  6      1     329      5 838983392 Star Trek: Generations (1994) Action|Adventu~
##  7      1     355      5 838984474 Flintstones, The (1994)      Children|Comed~
##  8      1     356      5 838983653 Forrest Gump (1994)          Comedy|Drama|R~
##  9      1     362      5 838984885 Jungle Book, The (1994)      Adventure|Chil~
## 10      1     364      5 838983707 Lion King, The (1994)        Adventure|Anim~
## # ... with 9,000,051 more rows
```

We can see this table is in tidy format with thousands of rows,Each row represents a rating given by one user to one movie.

```
glimpse(edx)
```

```
## Rows: 9,000,061
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ movieId   <dbl> 122, 185, 231, 292, 316, 329, 355, 356, 362, 364, 370, 377, ~
## $ 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, 838983392, 838983421, 838983392, 83898~
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Dumb & Dumber (1994)~
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Comedy", "Action~
```

There is 9,000,061 Row , aand 6 Columns.

edx information:

We can see the number of unique users that provide ratings and for how many unique movies they provided them

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

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

If we multiply those two numbers, we get a very large number , yet our data table has about 100,000 rows. This implies that not every user rated every movie.

So we can think of this data as a very large matrix with users on the rows and movies on the columns with many empty cells.

we can see a very small subset of 5 user and 5 movie:

```
keep <- edx %>%
     dplyr::count(movieId) %>%
     top_n(5) %>%
     pull(movieId)
```

```
## Selecting by n
```

```
 tab <- edx %>%
     filter(userId %in% c(34:38)) %>%
     filter(movieId %in% keep) %>%
     select(userId, title, rating) %>%
     spread(title, rating)
tab %>% knitr::kable()
```

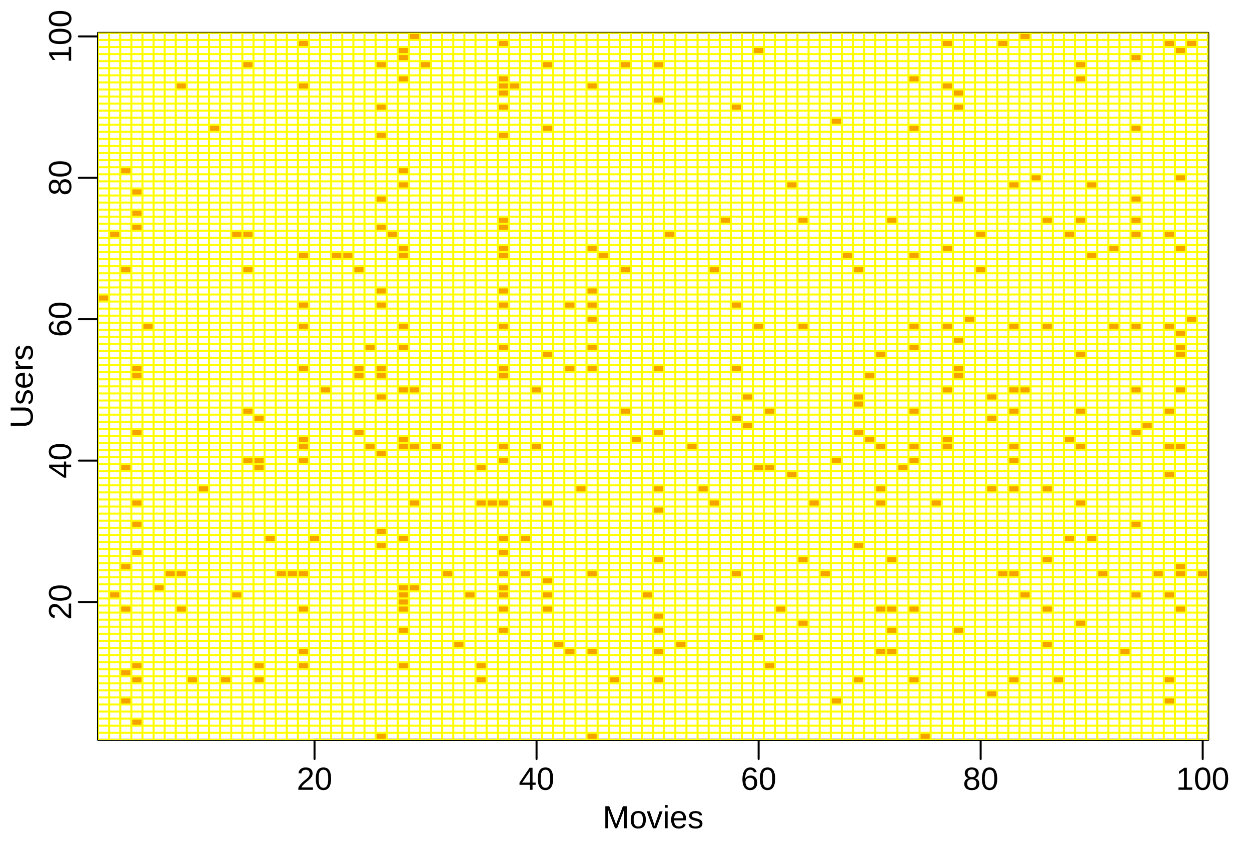
userId	Forrest Gump (1994)	Jurassic Park (1993)	Pulp Fiction (1994)	Shawshank Redemption, The (1994)	Silence of the Lambs, The (1991)
34	1.0	4	1.0	5	5
35	NA	3	NA	NA	NA
36	5.0	4	NA	NA	4
37	NA	NA	5.0	3	NA
38	4.5	3	4.5	4	4

we can see the ratings that each user gave each movie and we also see NA's for movies that they didn't watch or they didn't rate.

matrix sparse :

To see how sparse the entire matrix is, here the matrix for a random sample of 100 movies and 100 users:

```
users <- sample(unique(edx$userId), 100)
rafalib::mypar()
edx %>% filter(userId %in% users) %>%
     select(userId, movieId, rating) %>%
     mutate(rating = 1) %>%
     spread(movieId, rating) %>% select(sample(ncol(.), 100)) %>%
     as.matrix() %>% t(.) %>%
     image(1:100, 1:100,. , xlab="Movies", ylab="Users")
abline(h=0:100+0.5, v=0:100+0.5, col = "yellow")
```



the genres:

the Genres:

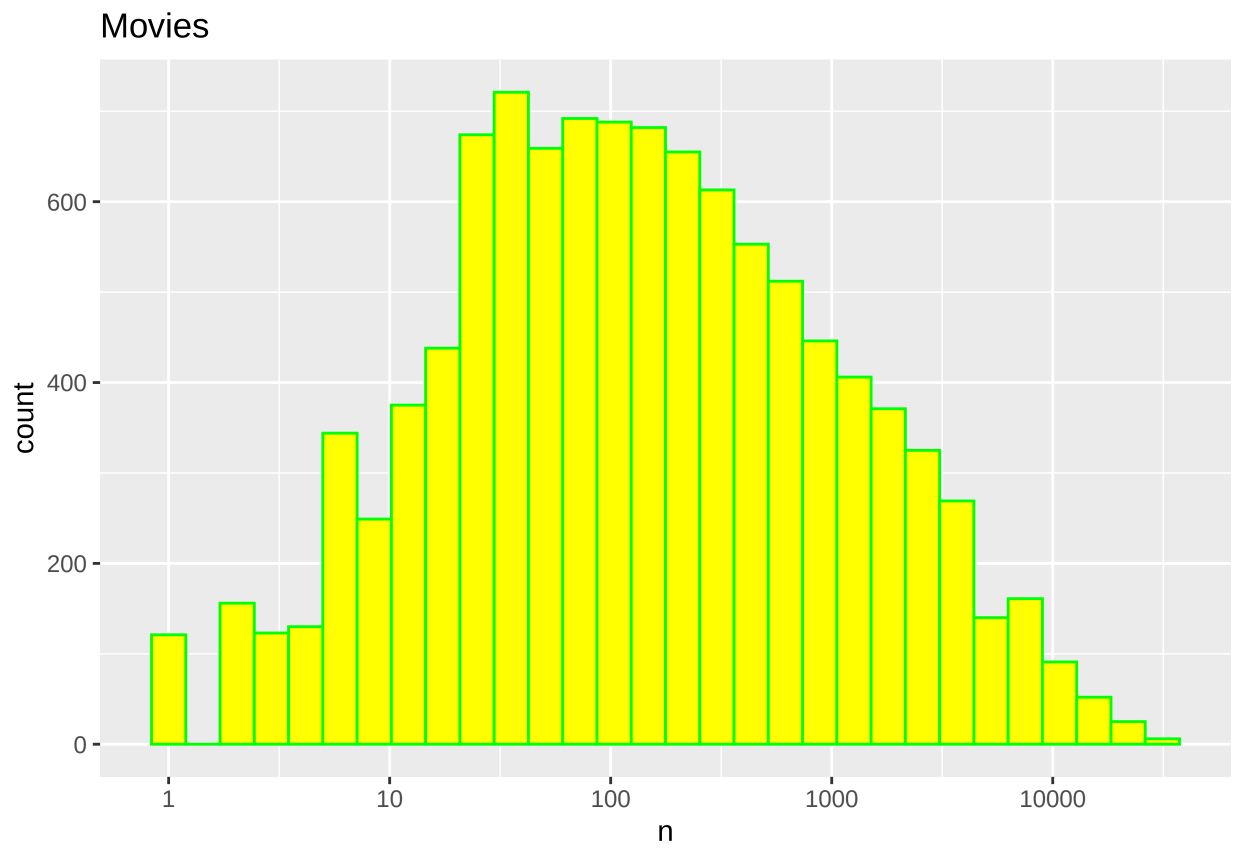
```
edx %>%
  summarize(n_genres = n_distinct(genres))
```

```
##   n_genres
## 1      797
```

the general properties of the data:

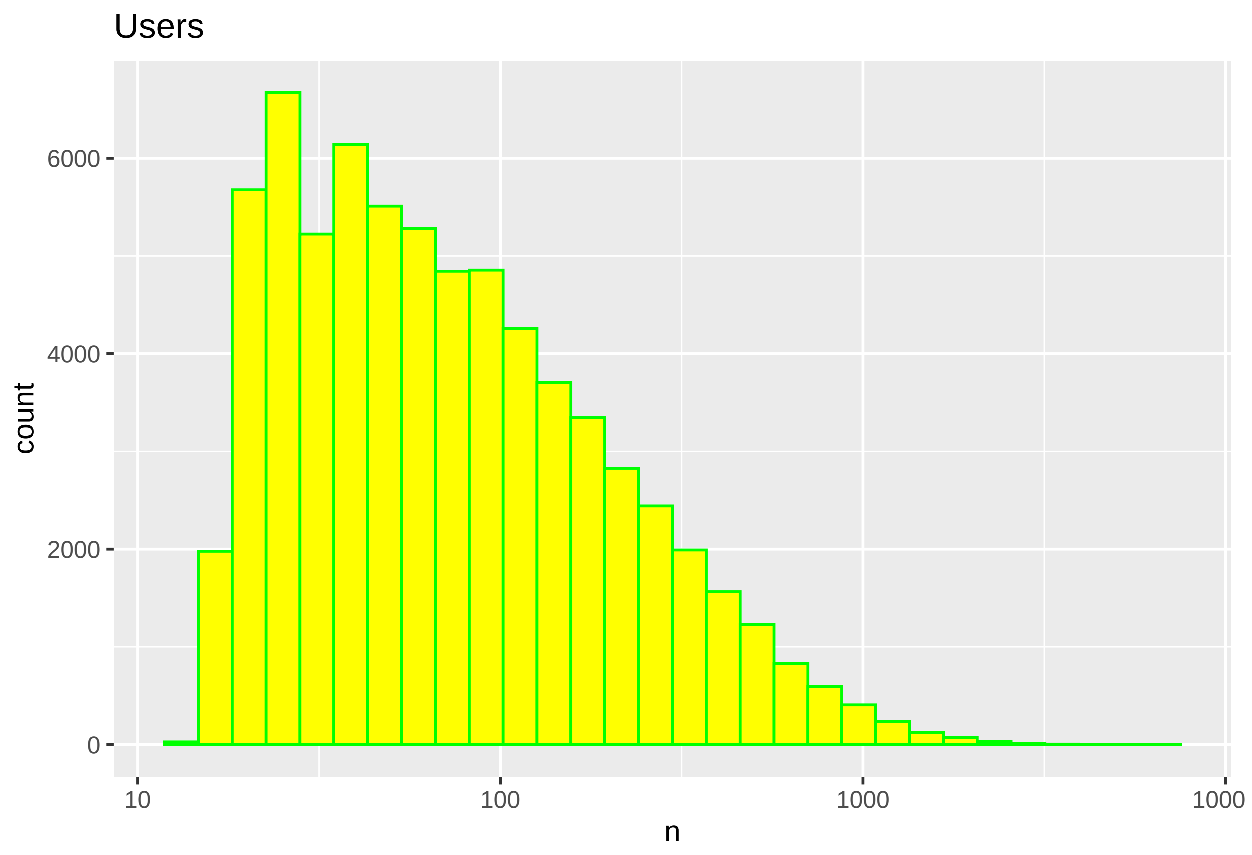
The first thing we notice is that some movies get rated more than others.`plot_1`

```
edx %>%
     dplyr::count(movieId) %>%
     ggplot(aes(n)) +
     geom_histogram(bins = 30, color = "green",fill="yellow") +
     scale_x_log10() +
     ggtitle("Movies")
```



A second observation is that some users are more active than others at rating movies - some users have rated over 1,000 movies- `plot_2`

```
edx %>%
     dplyr::count(userId) %>%
     ggplot(aes(n)) +
     geom_histogram(bins = 30, color = "green", fill="yellow") +
     scale_x_log10() +
     ggtitle("Users")
```



split edx in tow sets and create train and test set to assess the accuracy of the models we implement.

```
library(caret)
set.seed(1)
test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE)
           edx_train <- edx[-test_index,]
           edx_temp <- edx[test_index,]


edx_test <- edx_temp %>%
     semi_join(edx_train, by = "movieId") %>%
     semi_join(edx_train, by = "userId")
```

Loss funktion and compare models:

To compare different models or to see how well we're doing compared to some baseline, we need to quantify what it means to do well. We need a loss function. we are going to use the residual mean squared error on a test set.to To compare different models

Define RMSE

So if we define y__u,i as the rating for movie i by user -u and y-hat__u,i as our prediction, then the residual mean squared error is defined as follows.

$$\sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

n is a number of user movie combinations, and the sum is occurring over all these combinations.

and the function that computes this residual means squared error for a vector of ratings:

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

Building the Recommendation System

our models will be based on an approach called matrix factorization.

model__1 (Just the average):

we start by building the simplest possible recommendation system: we predict the same rating for all movies regardless of user and movie. this model would look something like this:

```
 ‘Y-u,i =mu + epsilon-u,i‘
```

epsilon represents independent errors - mu represents the true rating for all movies and users.

We know that the estimate that minimizes the residual mean squared error is the least squares estimate of mu. And in this case, that’s just the average of all the ratings. So we’re predicting all unknown ratings with this average.

compute mu

```
 mu_hat <- mean(edx_train$rating)
mu_hat
```

```
## [1] 3.51238
```

that is the average rating of all movies across all users.

```
naive_rmse <- RMSE(edx_test$rating, mu_hat)
naive_rmse
```

```
## [1] 1.059643
```

So we get a residual mean squared error of about 1.

RMSE table

Now because as we go along we will be comparing different approaches,we're going to create a table that's going to store the results that we obtain as we go along.We're going to call it RMSE results.

```
rmse_results <- tibble(method = "Just the average", RMSE = naive_rmse)
rmse_results
```

```
## # A tibble: 1 x 2
##   method            RMSE
##   <chr>            <dbl>
## 1 Just the average  1.06
```

model_2 (Movie effect):

We know that some movies are just generally rated higher than others. so We can augment our previous model by adding the term bi to represent the average rating for movie i.

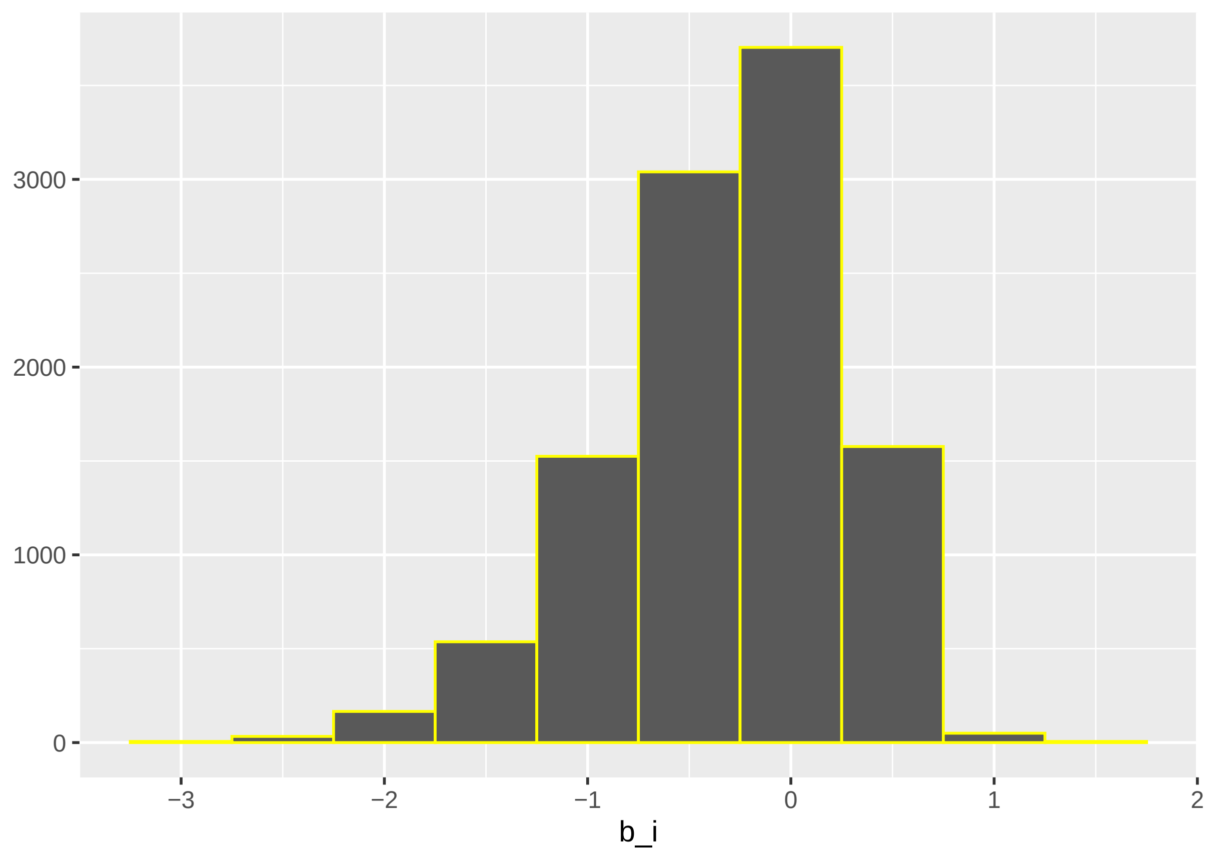
```
      ‘  Y-u,i =mu + epsilon-u,i + b-i‘
```

because b-hat_i, is just the average of y_u,i minus the overall mean for each movie, i. So we can compute them using this code.

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

from this `plot-3` we see that these estimates vary substantially ist (some movie are good Other movies are bad.)

```
 qplot(b_i, data = movie_avgs, bins = 10, color = I("yellow"))
```



Let's see how much our prediction improves once we use

$$\hat{y}_{u,i} = \hat{\mu} + \hat{b}_i$$

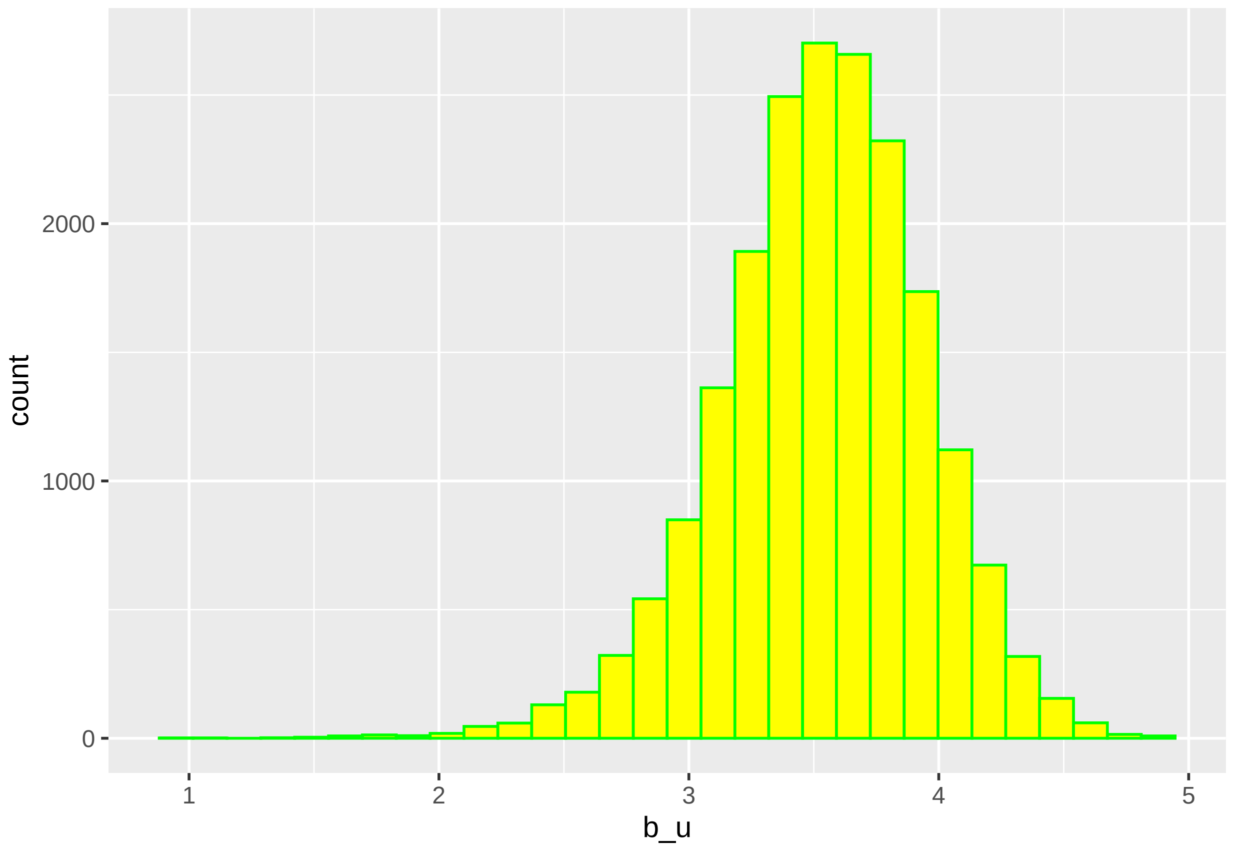
```
 predicted_ratings <- mu +edx_test %>%
     left_join(movie_avgs, by='movieId') %>%
     .$b_i


model_1_rmse <- RMSE(predicted_ratings,edx_test$rating)

rmse_results <- bind_rows(rmse_results,
                    tibble(method="MovieEffectModel",
RMSE= model_1_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0596429
MovieEffectModel	0.9431724

model_3 (User Effects):

different users are different in terms of how they rate movies we compute the average rating for user u for those that have rated 100 or more movies `plot_4`:

```
edx_train %>%
  group_by(userId) %>%
  filter(n()>=100) %>%
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "green",fill="yellow")
```



we can see that there is substantial variability across users as well: some users are very cranky and others love every movie. This implies that a further improvement to our model may be:

```
‘ Y-u,i =mu + epsilon-u,i + b-i + b-u‘
```

We include a term, b_u, which is the user-specific effect.

we will compute our approximation by computing the overall mean, u-hat, the movie effects, b-hat_i, and then estimating the user effects, b_u-hat, by taking the average of the residuals obtained after removing the overall mean and the movie effect from the ratings y_u,i.

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

```
predicted_ratings <- edx_test %>%
     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, edx_test$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie + User Effects Model",
           RMSE = model_2_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0596429
MovieEffectModel	0.9431724
Movie + User Effects Model	0.8655154

model_4 (Regularization):

our improvement in residual mean square error when we just included the movie effect was only about 5%.

to see what's going on, let's look at the top 10 best movies based on the estimates of the movie effect `b-hat_i`.

```
set.seed(755)
movie_titles <- edx %>%
     select(movieId, title) %>%
     distinct()
movie_avgs %>% left_join(movie_titles, by="movieId") %>%
     arrange(desc(b_i)) %>%
     select(title, b_i) %>%
     slice(1:10) %>%
     knitr::kable()
```

title	b_i
Shanghai Express (1932)	1.487620
Satan's Tango (Sátántangó) (1994)	1.487620
Fighting Elegy (Kenka erejii) (1966)	1.487620
Sun Alley (Sonnenallee) (1999)	1.487620
Constantine's Sword (2007)	1.487620
Human Condition II, The (Ningen no joken II) (1959)	1.320954
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.237620
Life of Oharu, The (Saikaku ichidai onna) (1952)	1.237620
I'm Starting From Three (Ricomincio da Tre) (1981)	1.154287
Human Condition III, The (Ningen no joken III) (1961)	1.154287

here are the best 10 movies according to our estimates.

we can see tthat they all have something in common. They're all quite obscure. So let's look at how often they were rated.

Here is the same table, but now we include the number of ratings

```r
edx_train %>% dplyr::count(movieId) %>%
    left_join(movie_avgs) %>%
    left_join(movie_titles, by="movieId") %>%
    arrange(desc(b_i)) %>%
    select(title, b_i, n) %>%
    slice(1:10) %>%
    knitr::kable()
```

```
## Joining, by = "movieId"
```

title	b_i	n
Shanghai Express (1932)	1.487620	1
Satan's Tango (Sátántangó) (1994)	1.487620	1
Fighting Elegy (Kenka erejii) (1966)	1.487620	1
Sun Alley (Sonnenallee) (1999)	1.487620	1
Constantine's Sword (2007)	1.487620	1
Human Condition II, The (Ningen no joken II) (1959)	1.320954	3
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.237620	4
Life of Oharu, The (Saikaku ichidai onna) (1952)	1.237620	2
I'm Starting From Three (Ricomincio da Tre) (1981)	1.154287	3
Human Condition III, The (Ningen no joken III) (1961)	1.154287	3

So the supposed best and worst movies were rated by very few users, in most cases just one. These movies were mostly obscure 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 when fewer users rate the movies.

For this, we introduce the concept of regularization. Regularization permits us to penalize large estimates that come from small sample sizes.

The general idea is to add a penalty for large values of b to the sum of squares equations that we minimize. So having many large b's makes it harder to minimize the equation that we're trying to minimize.

so we are going to estimate the b's instead of minimizing the residual sum of squares as is done by least squares, we now minimize 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 the mean squared error and the second is a penalty term that gets larger when many 's are large.

The values of that minimize this equation are given by:

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

where `n-i` is a number of ratings `b` for movie`i` .

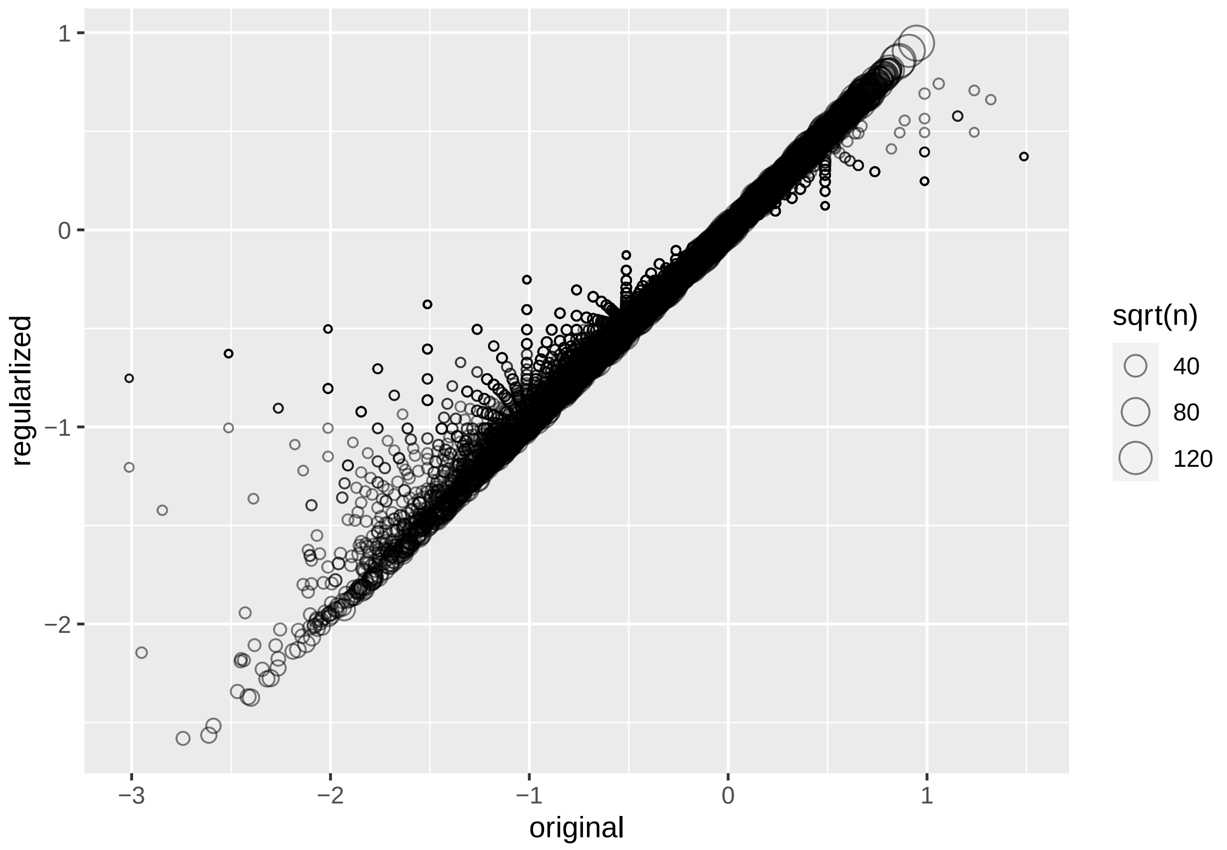
when n_i is small, then the estimate of b_i is shrunken towards zero.The larger lambda, the more we shrink.

compute these regularized estimates of b_i using lambda equals to 3

```
lambda <- 3
mu <- mean(edx_train$rating)
movie_reg_avgs <- edx_train %>%
     group_by(movieId) %>%
     summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
```

here we can see how the estimates shrink `plot5`:

```
tibble(original = movie_avgs$b_i,
           regularlized = movie_reg_avgs$b_i,
           n = movie_reg_avgs$n_i) %>%
     ggplot(aes(original, regularlized, size=sqrt(n))) +
     geom_point(shape=1, alpha=0.5)
```



wee see that when n is small, the values are shrinking more towards zero.

know let's look at our top 10 best movies based on the estimates we got when using regularization.

```
edx_train %>%
     dplyr::count(movieId) %>%
     left_join(movie_reg_avgs) %>%
     left_join(movie_titles, by="movieId") %>%
     arrange(desc(b_i)) %>%
     select(title, b_i, n) %>%
     slice(1:10) %>%
     knitr::kable()
```

```
## Joining, by = "movieId"
```

title	b_i	n
Shawshank Redemption, The (1994)	0.9471172	22340
Godfather, The (1972)	0.9082163	14322
Usual Suspects, The (1995)	0.8589670	17163
Schindler's List (1993)	0.8505500	18531
Double Indemnity (1944)	0.8149149	1702
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	0.8129168	2315
Casablanca (1942)	0.8099059	9000
Seven Samurai (Shichinin no samurai) (1954)	0.8034630	4159
Rear Window (1954)	0.7967186	6343
Godfather: Part II, The (1974)	0.7919178	9616

So do we improve our results?

```
predicted_ratings <- edx_test %>%
    left_join(movie_reg_avgs, by='movieId') %>%
    mutate(pred = mu + b_i) %>%
    .$pred
```

```
model_3_rmse <- RMSE(predicted_ratings, edx_test$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie Effect Model",
                                     RMSE = model_3_rmse ))
```

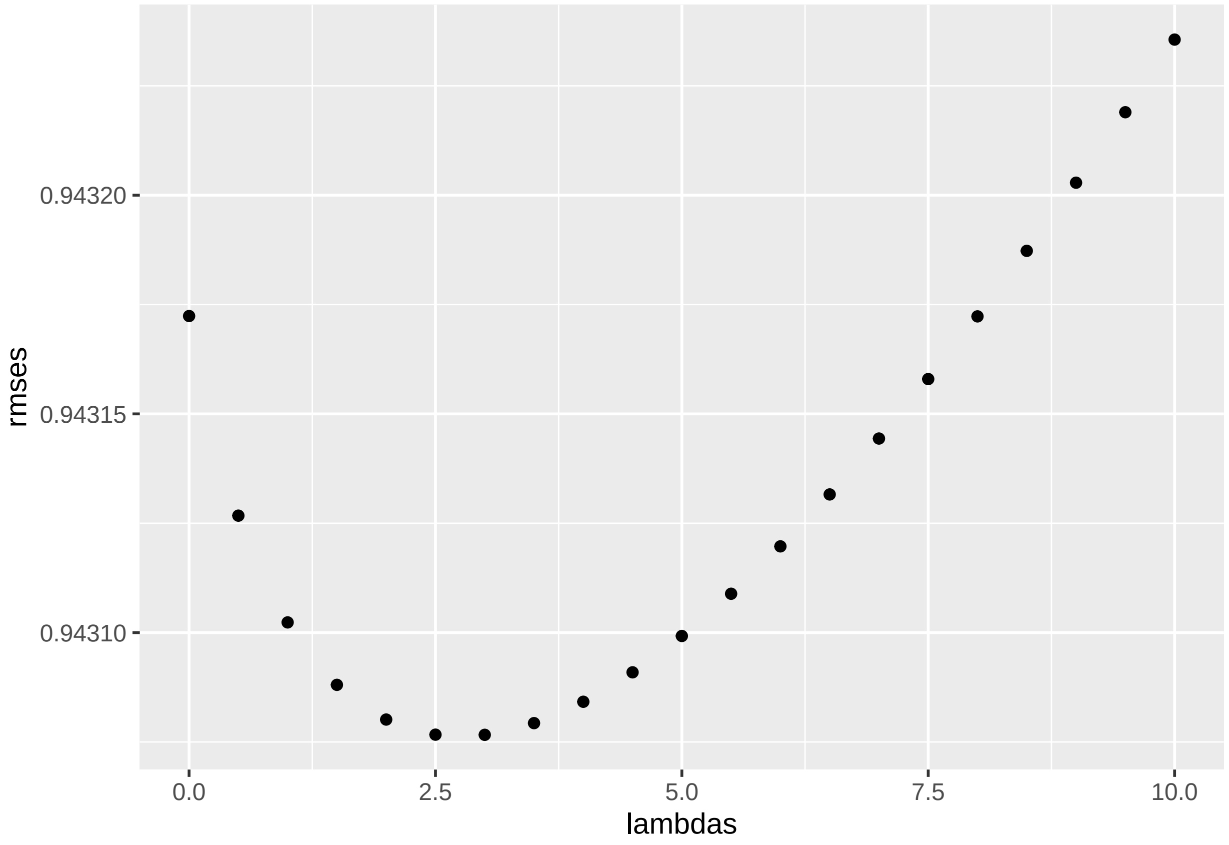
```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```

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

method	RMSE
Just the average	1.0596429
MovieEffectModel	0.9431724
Movie + User Effects Model	0.8655154
Regularized Movie Effect Model	0.9430766

lambda is a tuning parameter. We can use cross-validation to choose it.

```r
lambdas <- seq(0, 10, 0.5)
mu <- mean (edx_train$rating)
just_the_sum <- edx_train %>%
     group_by(movieId) %>%
     summarize(s = sum(rating - mu), n_i = n())
rmses <- sapply(lambdas, function(l){
     predicted_ratings <- edx_test %>%
          left_join(just_the_sum, by='movieId') %>%
          mutate(b_i = s/(n_i+l)) %>%
          mutate(pred = mu + b_i) %>%
          .$pred
     return(RMSE(predicted_ratings, edx_test$rating))
})
qplot(lambdas, rmses)
```



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

```
## [1] 3
```

useing regularization to estimate the user effect.

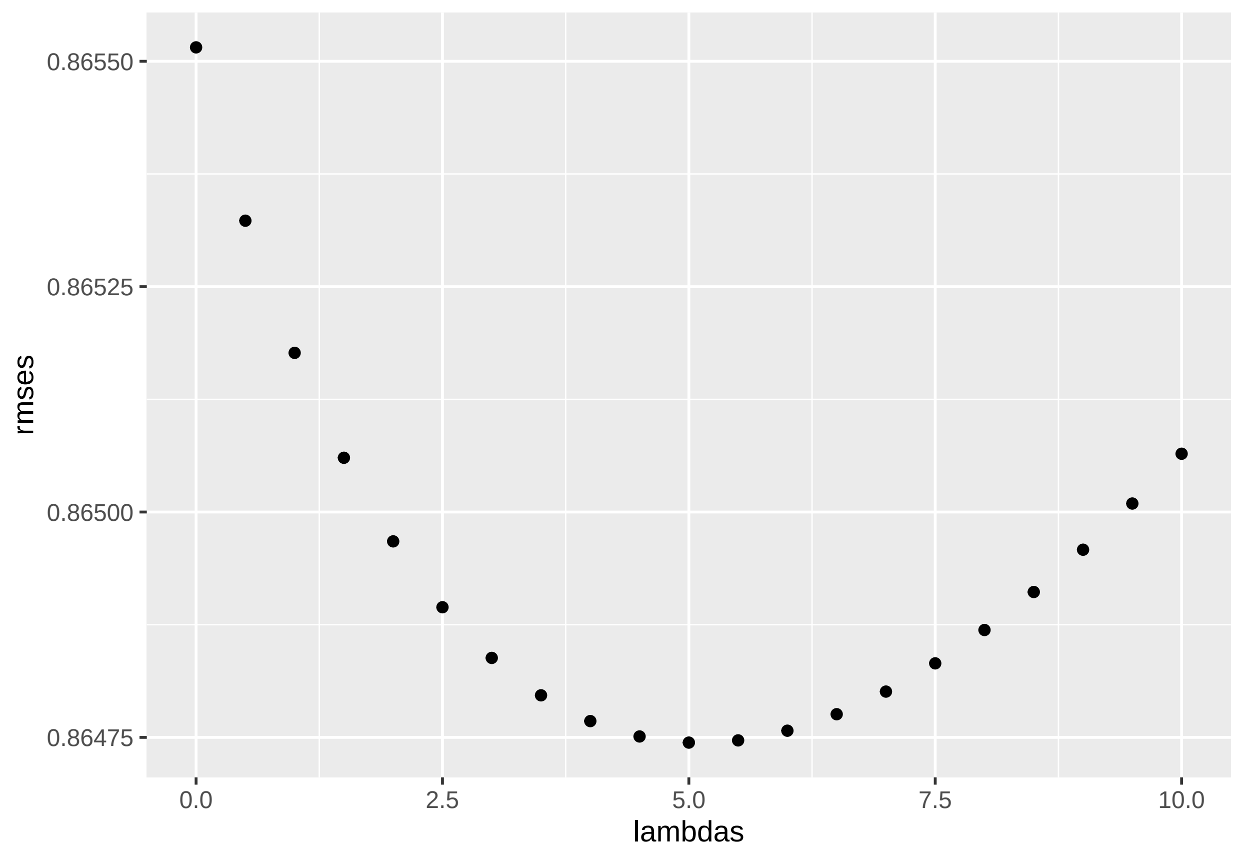
The equation we would minimize would be this one now.

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

It includes the parameters for the user effects as well.

Here we again use cross-validation to pick lambda.

```
lambdas <- seq(0, 10, 0.5)
rmses <- sapply(lambdas, function(l){
     mu <- mean(edx_train$rating)
     b_i <- edx_train %>%
          group_by(movieId) %>%
          summarize(b_i = sum(rating - mu)/(n()+l))
     b_u <- edx_train %>%
          left_join(b_i, by="movieId") %>%
          group_by(userId) %>%
          summarize(b_u = sum(rating - b_i - mu)/(n()+l))
     predicted_ratings <-
          edx_test %>%
          left_join(b_i, by = "movieId") %>%
          left_join(b_u, by = "userId") %>%
          mutate(pred = mu + b_i + b_u) %>%
          .$pred
     return(RMSE(predicted_ratings, edx_test$rating))
})

qplot(lambdas, rmses)
```



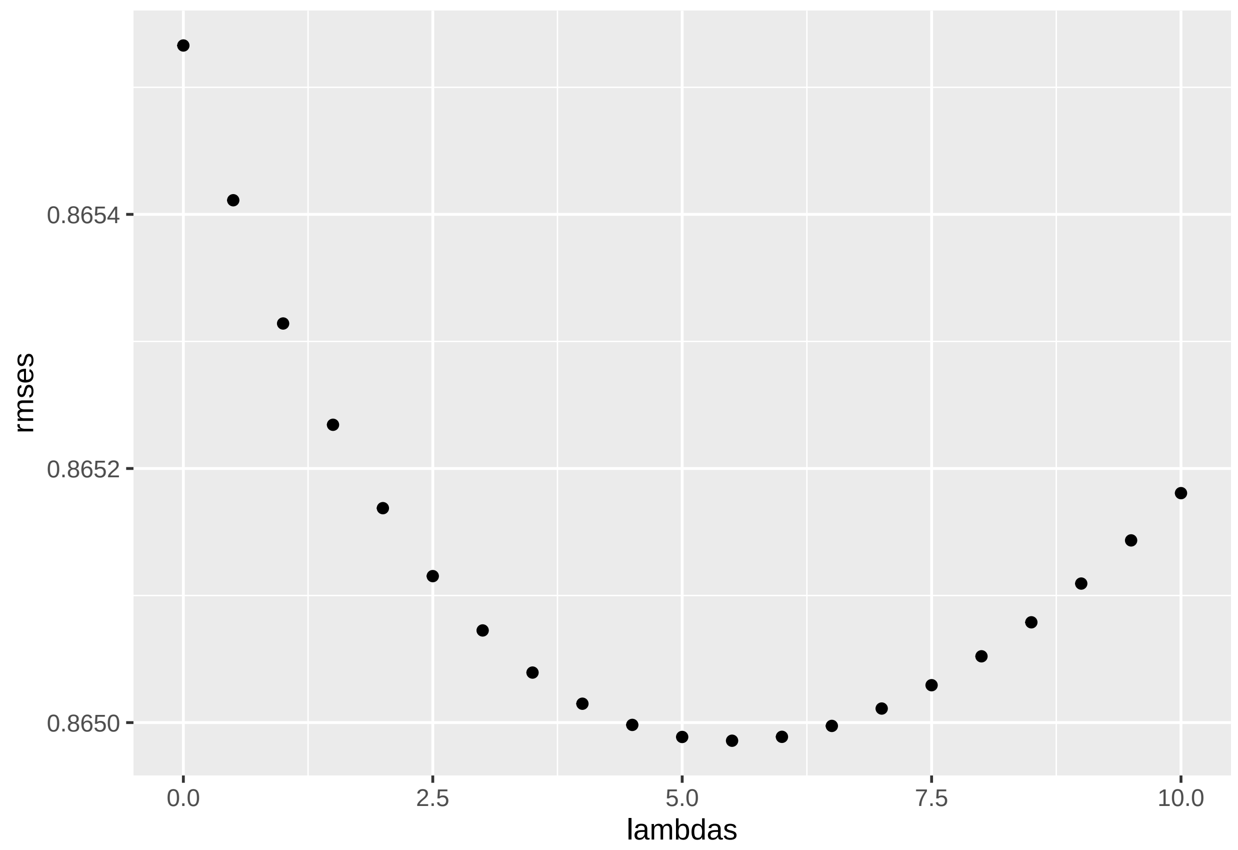
```
lambda <- lambdas[which.min(rmses)]
lambda
```

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

```
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Regularized Movie + User Effect Model",
                                 RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0596429
MovieEffectModel	0.9431724
Movie + User Effects Model	0.8655154
Regularized Movie Effect Model	0.9430766
Regularized Movie + User Effect Model	0.8647442

```
lambdas <- seq(0, 10, 0.5)
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) %>%
          .$pred
     return(RMSE(predicted_ratings, validation$rating))
})

qplot(lambdas, rmses)
```



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

```
## [1] 5.5
```

```
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Regularized Movie + User Effect Model",
                                 RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0596429
MovieEffectModel	0.9431724
Movie + User Effects Model	0.8655154
Regularized Movie Effect Model	0.9430766
Regularized Movie + User Effect Model	0.8647442
Regularized Movie + User Effect Model	0.8649857

Citations:

Irizarry, Rafael A. -Introduction to Data Science-