capston project

H. Almustafa

2022-09-16

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 uploding

```
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
```

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

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")

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:

to see the number of unique users that provided ratings and how many unique movies were rated:

```
 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
```

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
```

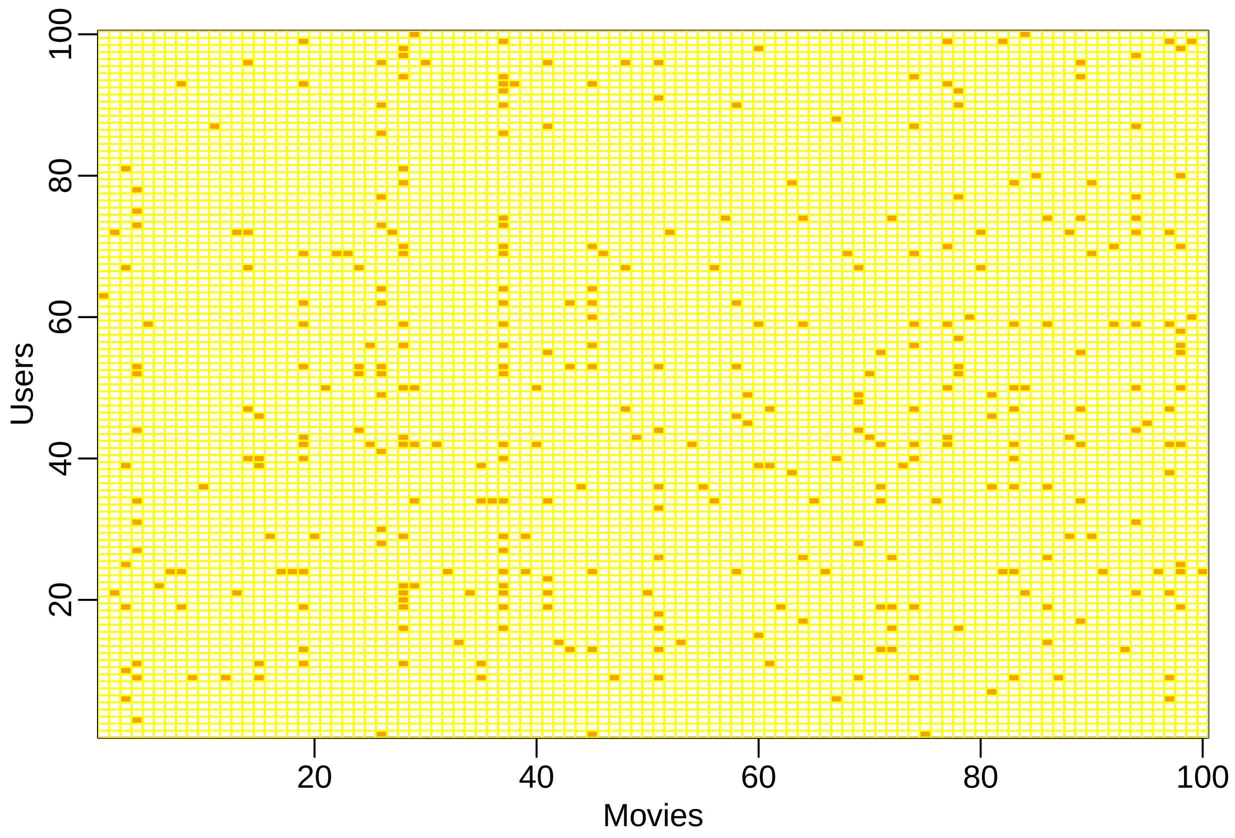
```
tab <- edx %>%
     filter(userId %in% c(34:38)) %>%
     filter(movieId %in% keep) %>%
     select(userId, title, rating) %>%
     spread(title, rating)
tab %>% knitr::kable()
```

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:

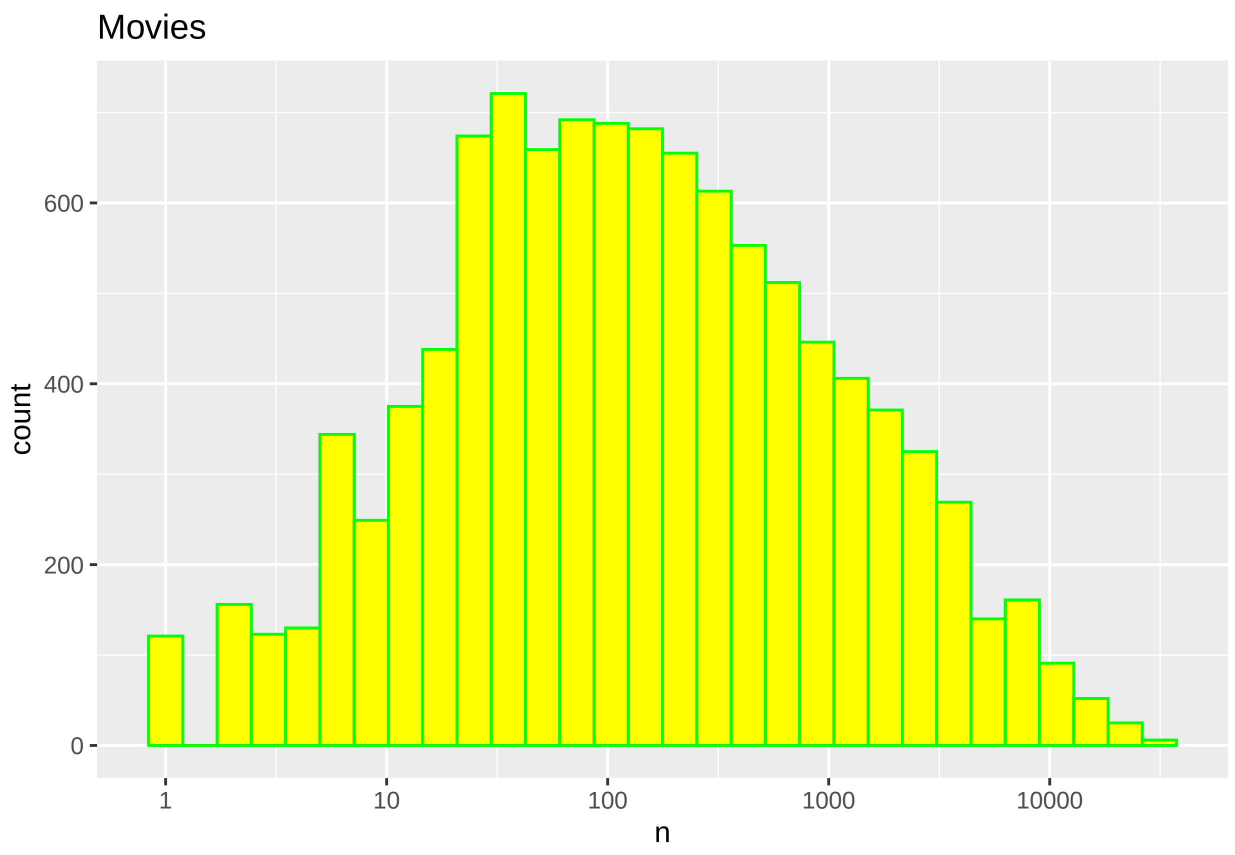
```
edx|>
  summarize(n_genres = n_distinct(genres))
```

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

the general properties of the data:

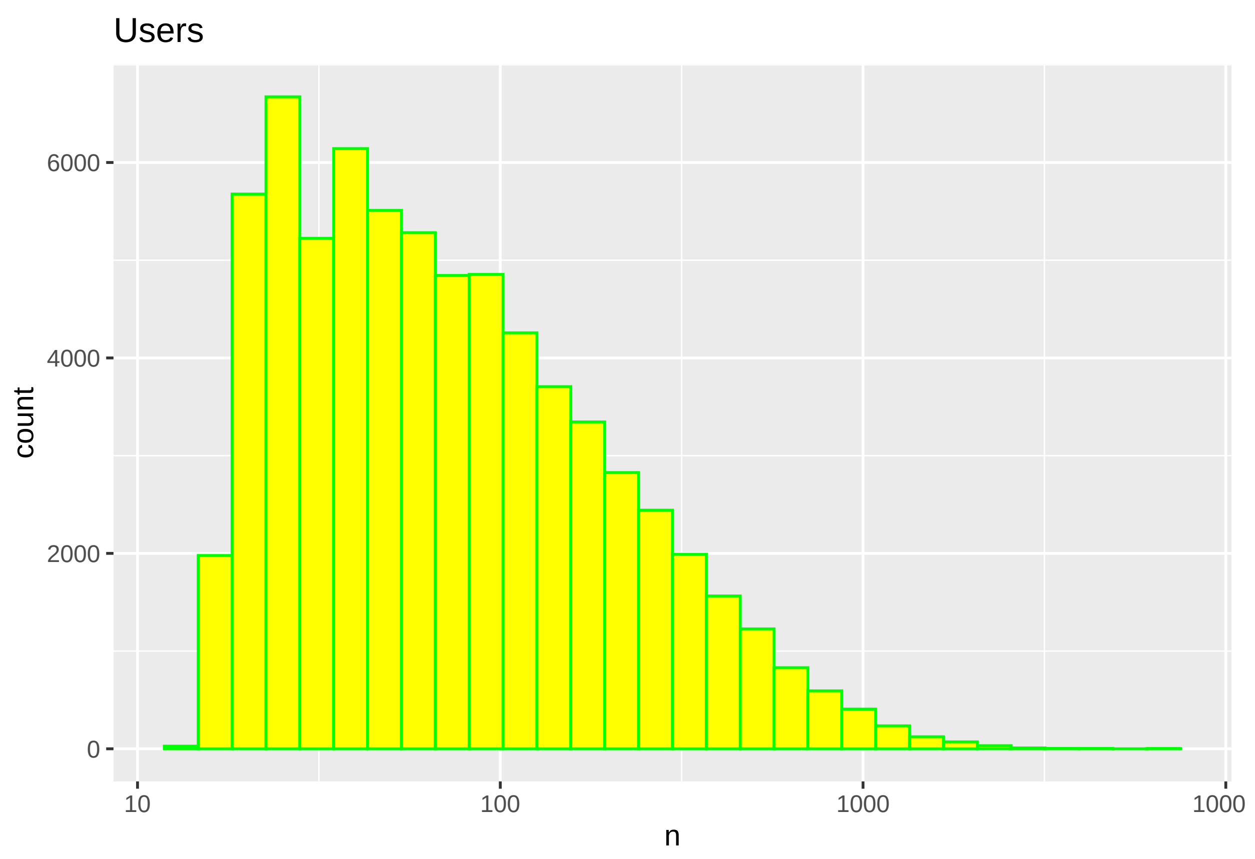
The first thing we notice is that some movies get rated more than others.`plot_1`

```
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`plot_2` - some users have rated over 1,000 movies

```
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 a 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.

Define RMSE

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

# Modeling

** 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.

- compute mu *

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

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

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

We compute this average on the training data. And then we compute the residual mean squared error on the test set data.

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

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

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
```

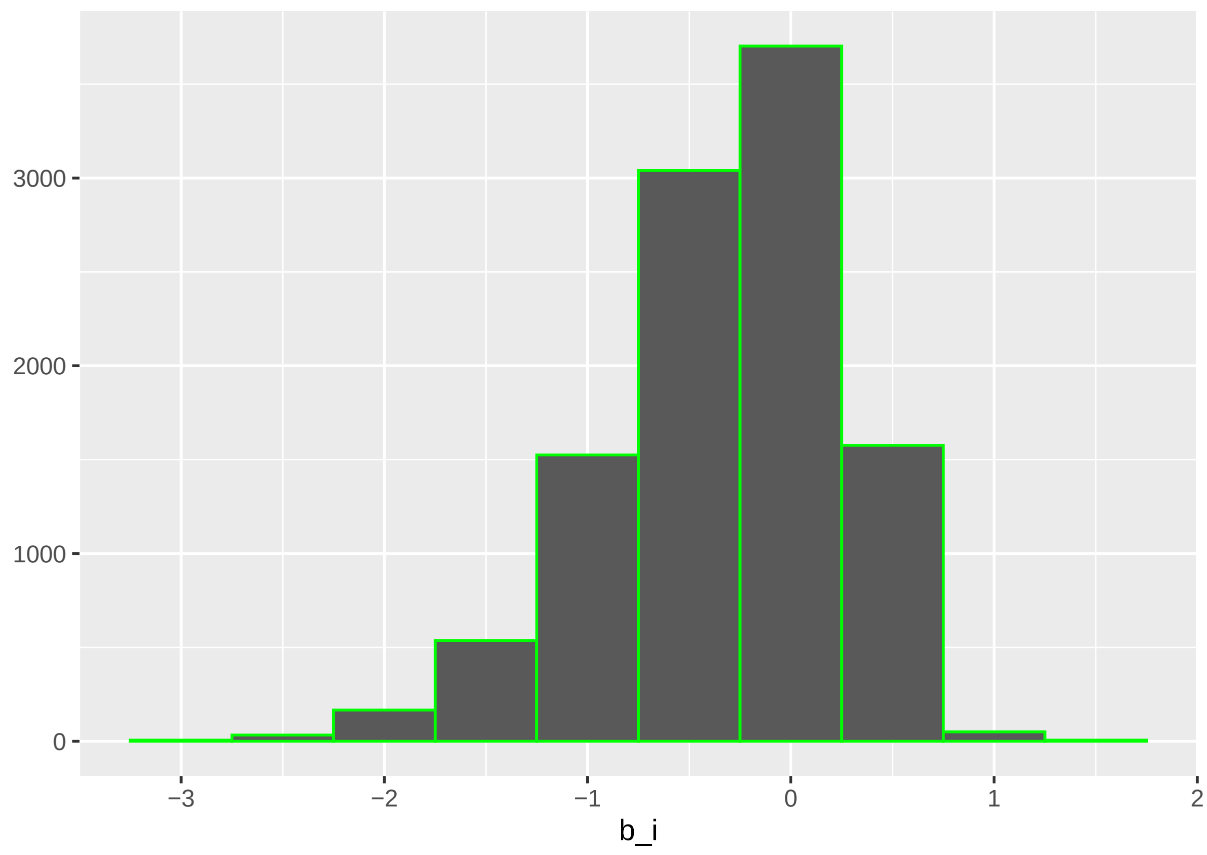
```
## # A tibble: 1 x 2
##   method             RMSE
##   <chr>             <dbl>
## 1 Just the average   1.06
```

model_2 (Movie effect)

We know that some movies are just generally rated higher than others. This intuition, that different movies are rated differently, is confirmed by data. we're going to drop the hat notation in the code to represent the estimates going forward, just to make the code cleaner.

We can see this by making a `plot` of the average rating that each movie got.

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

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



so We can augment our previous model by adding the term b-i to represent average ranking for movie i:

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

we have already the b-i Now we can see how much our prediction improves once we predict

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

```
## [1] 0.9431724
```

results table:

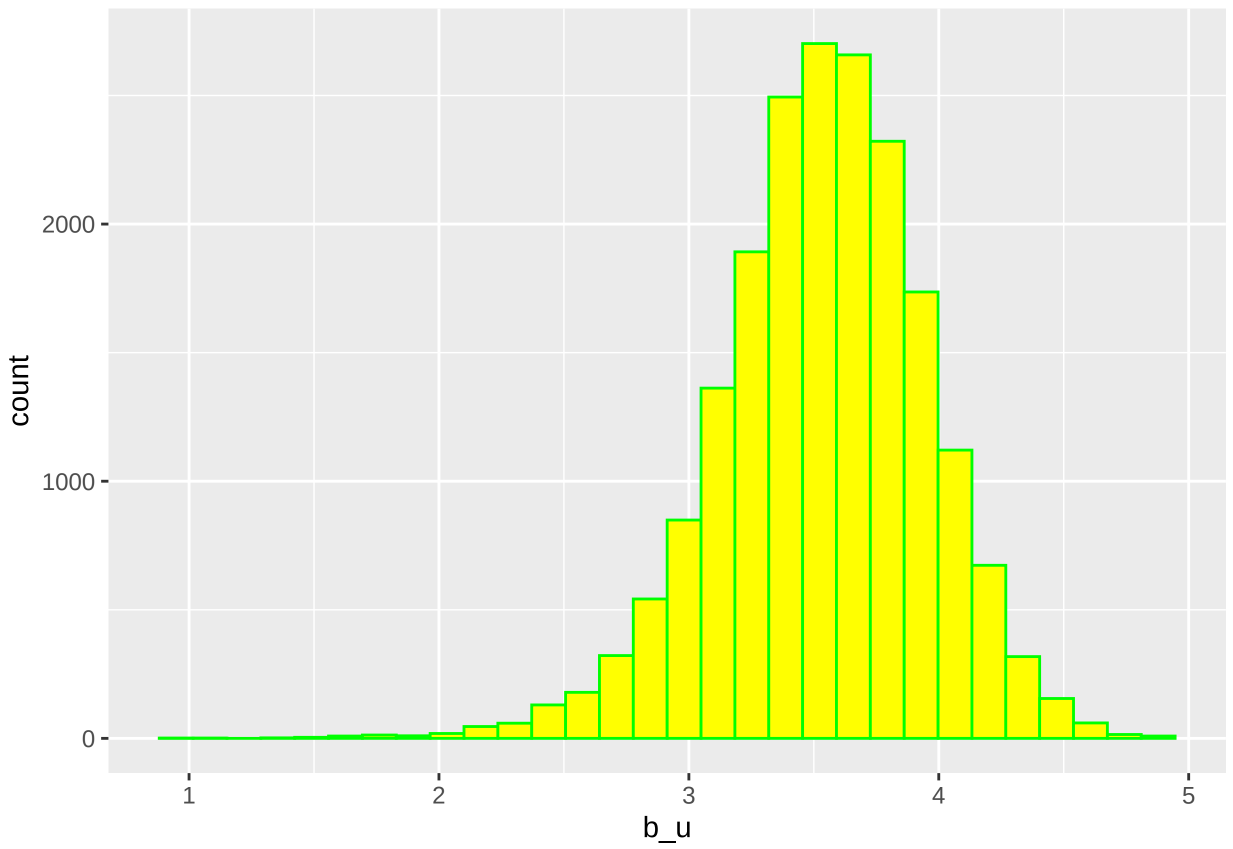
```
model_1_rmse <- RMSE(predicted_ratings,edx_test$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie Effect Model",
                                     RMSE = model_1_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0596429
Movie Effect Model	0.9431724

# model_3 (User Effects):

different users are different in terms of how they rate movies. let's compute the average rating for user, u, for those that have rated over 100 movies. we can see that from this `plot`

```
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")
```



the plot confirm that there is substantial variability across users, as well.

This implies that a further improvement to our model may be something like this.

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

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

will compute an approximation by computing `mu-hat`,`b-i-hat` and estimating `b-u-hat` as the average of y-u,i-mu- hat-b-i-hat

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

We can now construct predictors and see how much the RMSE improves:

```
predicted_ratings <- edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
RMSE(predicted_ratings,  edx_test$rating)
```

```
## [1] 0.8655154
```

and rmse results :

```
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
Movie Effect Model	0.9431724
Movie + User Effects Model	0.8655154

# model_4 (Regularization)

Despite the large movie to movie variation, our improvement in RMSE was only about 5%,why it wasn't bigger? Let's explore where we made mistakes in our first model, using only movie effects `b-i`

```
edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(residual = rating - (mu + b_i)) %>%
  arrange(desc(abs(residual))) %>%
  slice(1:10) %>%
  pull(title)
```

```
##  [1] "From Justin to Kelly (2003)"
##  [2] "Pokémon Heroes (2003)"
##  [3] "Holy Mountain, The (Montaña sagrada, La) (1973)"
##  [4] "Shawshank Redemption, The (1994)"
##  [5] "Shawshank Redemption, The (1994)"
##  [6] "Shawshank Redemption, The (1994)"
##  [7] "Shawshank Redemption, The (1994)"
##  [8] "Shawshank Redemption, The (1994)"
##  [9] "Shawshank Redemption, The (1994)"
## [10] "Carnosaur 3: Primal Species (1996)"
```

most of the movies ,seem to be obscure movies and in our model many of them obtained large predictions.

The supposed "best" and "worst" movies were rated by very few users, in most cases just 1. This is because with just a few users, we have more uncertainty.

Regularization permits us to penalize large estimates that are formed using small sample sizes. so we are going To estimate the b's instead of minimizing the residual sum of squares

Using calculus we can actually show that the values of bi that minimize this equation are:

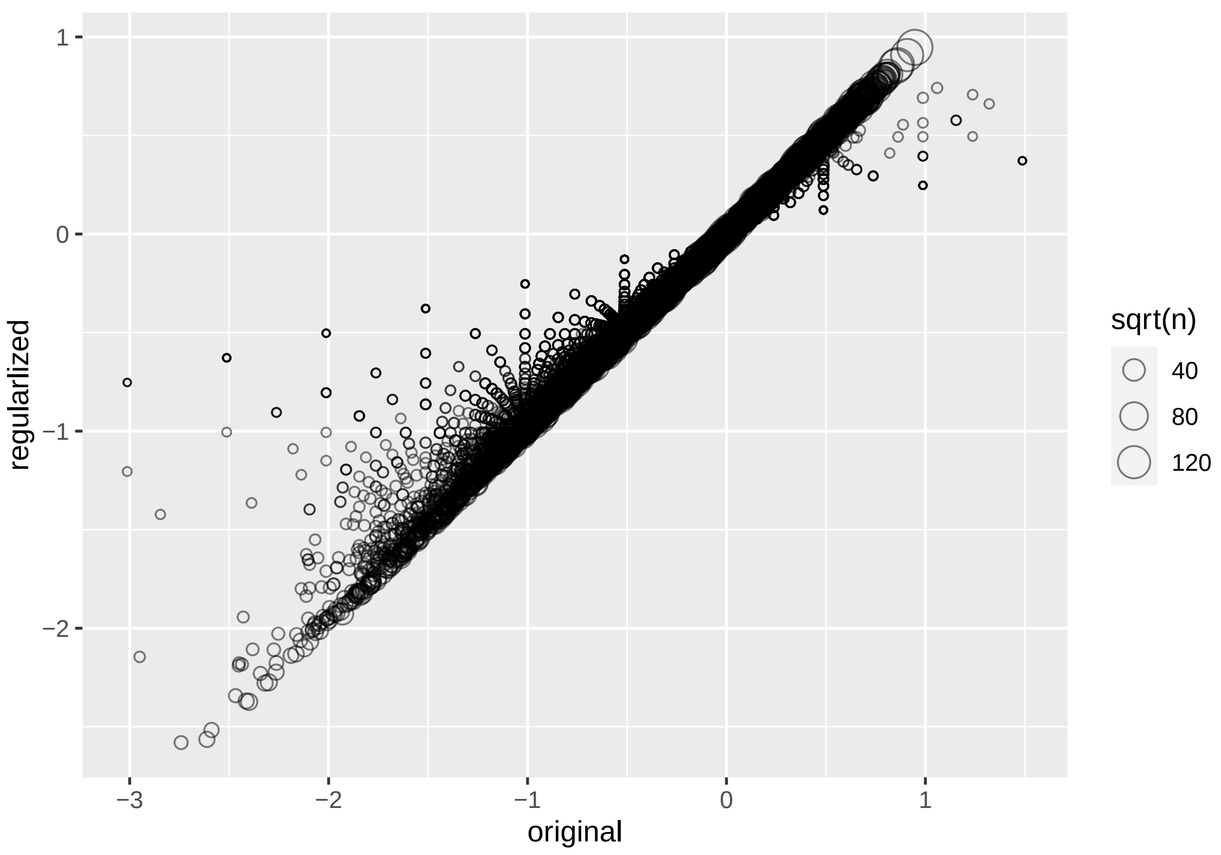
computing these regularized estimates of bi using lambda =3 .

```
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())
```

To see how the estimates shrink, let's make a`plot` of the regularized estimate versus the least square estimates with the size of the circle telling us how large n-i was.

```
data_frame(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)
```

```
## 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.
```



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

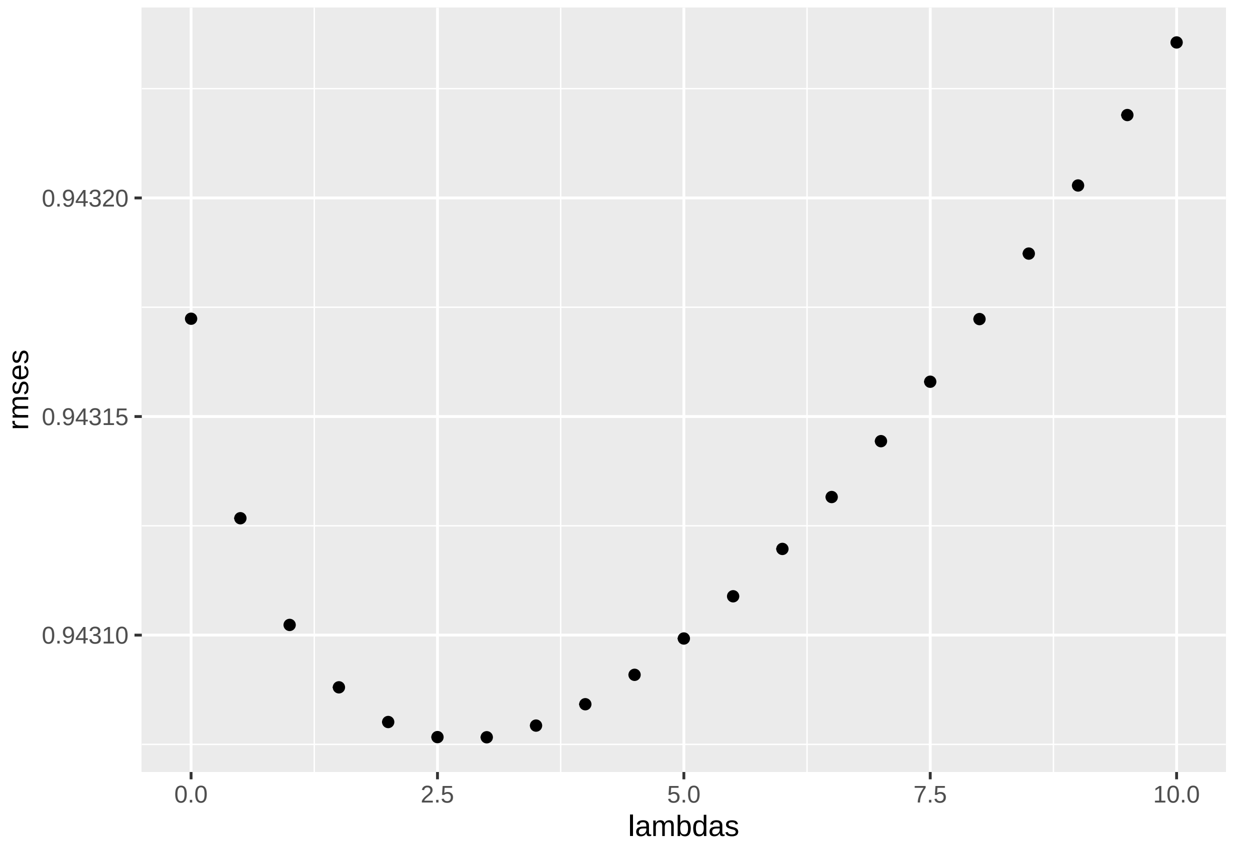
know we check if we improve our results

```
  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,
                          tibble(method="Regularized Movie Effect Model",
                                     RMSE = model_3_rmse ))
rmse_results %>% knitr::kable()
```

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

we use the cross-validation to choose lambda

```
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)
```



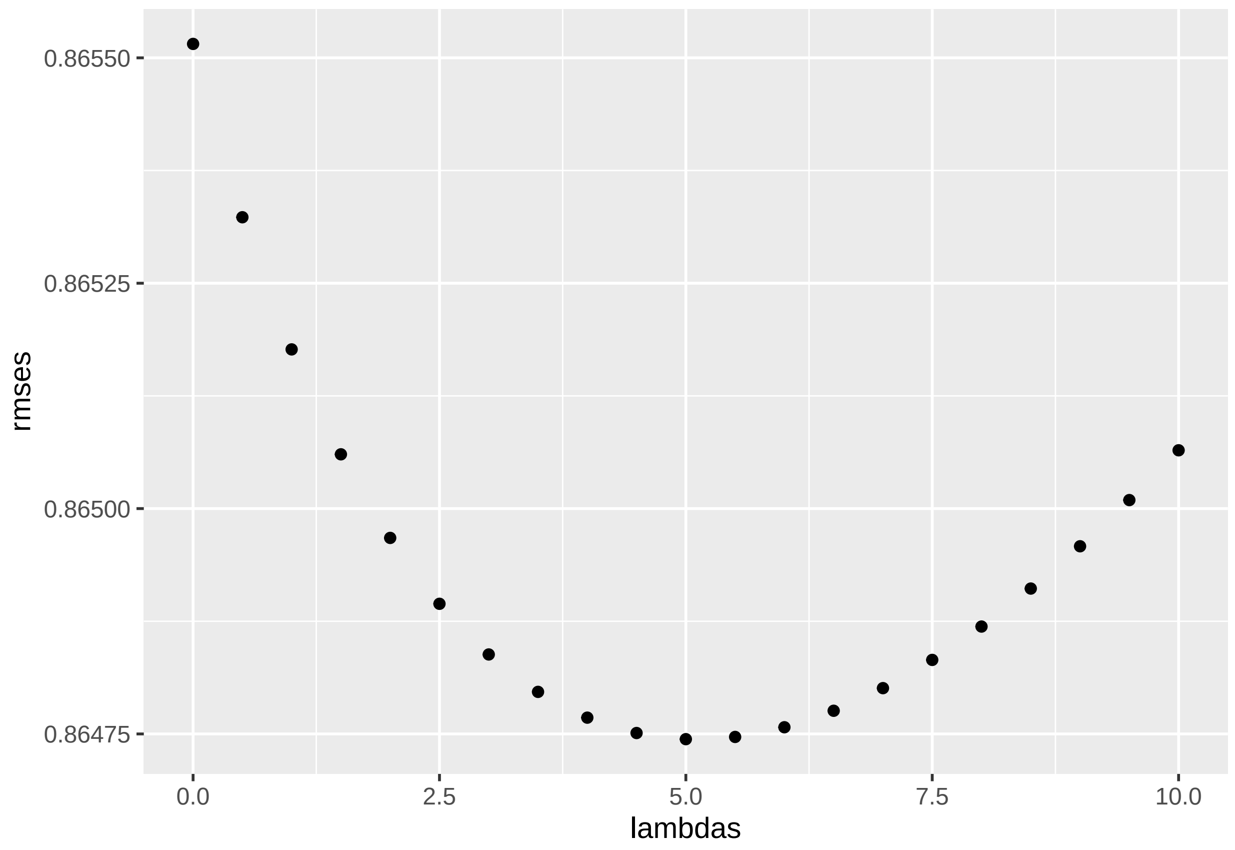
```
lambdas[which.min(rmses)]
```

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

We can also use regularization to estimate the user effect. The equation we would minimize would be this one now.

Here we use cross-validation again . to pick a 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
```

over here a lambda=5 is the best tune.

```
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
Movie Effect Model	0.9431724
Movie + User Effects Model	0.8655154
Regularized Movie Effect Model	0.9430766
Regularized Movie + User Effect Model	0.8647442

using matrix factorization , we have achieved the best accuracy / 0.864 /

## References Rafael A. Irizarry " Introduction to Data Science"