HarvardX Capstone - MovieLens Report

Leandro Rodrigues Carvalho

09/02/2021

Introduction

This is an R Markdown Report about the "MovieLens" project as part of "HarvardX - Data Science: Capstone" course, lead by Professor Rafael A. Irizarry. The *goal* of this project is to create a movie recommendation system to predict movie ratings using the MovieLens data set. The *key steps* that were performed included Exploratory Data Analysis (EDA), data visualization, data wrangling and training the machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set.

Historic Context

The recommendation system presented at the HarvardX course went through some of the data analysis strategies used by the winning team of the 2006 "Netflix Challenge", when the company challenged the data science community to improve its recommendation algorithm by 10%. The winner would get a million dollars prize. In September 2009, the winners were announced.

Code provided

This code creates a training set and a validation, or test, set (final hold-out test set)

```
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")

library(tidyverse)
library(caret)
library(data.table)
```

#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")

# 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, 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)
```

Saving data as R objects

```
save(edx, file = 'edx.RData')
save(validation, file = 'validation.RData')
```

Dataset description

The data set version of MovieLens used in this report, the 10M version of the original data set, is a small subset of a much larger set with millions of ratings, in order to make the computation faster.

It's important to mention that the Netflix data is not publicly available, but the GroupLens research lab generated their own database with over 20 million ratings for over 27,000 movies by more than 138,000 users.

HarvardX and Professor Rafael A. Irizarry made a small subset of this data via the dslabs package containing 9,000,055 observations of 6 variables, where each row represents a rating given by one user to one movie.

```
str(edx)
```

There is a total of 10,677 different movies and 69,878 different users in the data set.

```
unique_col <- edx %>%
summarize(unique_users = n_distinct(userId),
unique_movies = n_distinct(movieId),
unique_genres = n_distinct(genres))

knitr::kable(unique_col)
```

unique_users	unique_movies	unique_genres
69878	10677	797

```
tot_observation <- length(edx$rating) + length(validation$rating)
tot_observation
```

```
## [1] 10000054
```

Some of the most popular genres in the data set are “Drama”, “Comedy”, “Thriller” and “Romance”.

```
genres = c("Drama", "Comedy", "Thriller", "Romance")
  sapply(genres, function(g) {
    sum(str_detect(edx$genres, g))
  })
```

```
##    Drama   Comedy Thriller  Romance
##  3910127  3540930  2325899  1712100
```

The movie with the greatest number of ratings is “Pulp Fiction”

```
edx %>% group_by(movieId, title) %>%
    summarize(count = n()) %>%
    arrange(desc(count))
```

```
## # 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
```

It’s important to understand that not every user rated every movie. There are users on the rows and movies on the columns with many empty cells.

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

```
## Selecting by n
```

```
tab <- edx %>%
     filter(userId %in% c(13:20)) %>%
     filter(movieId %in% keep) %>%
     select(userId, title, rating) %>%
     spread(title, rating)

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

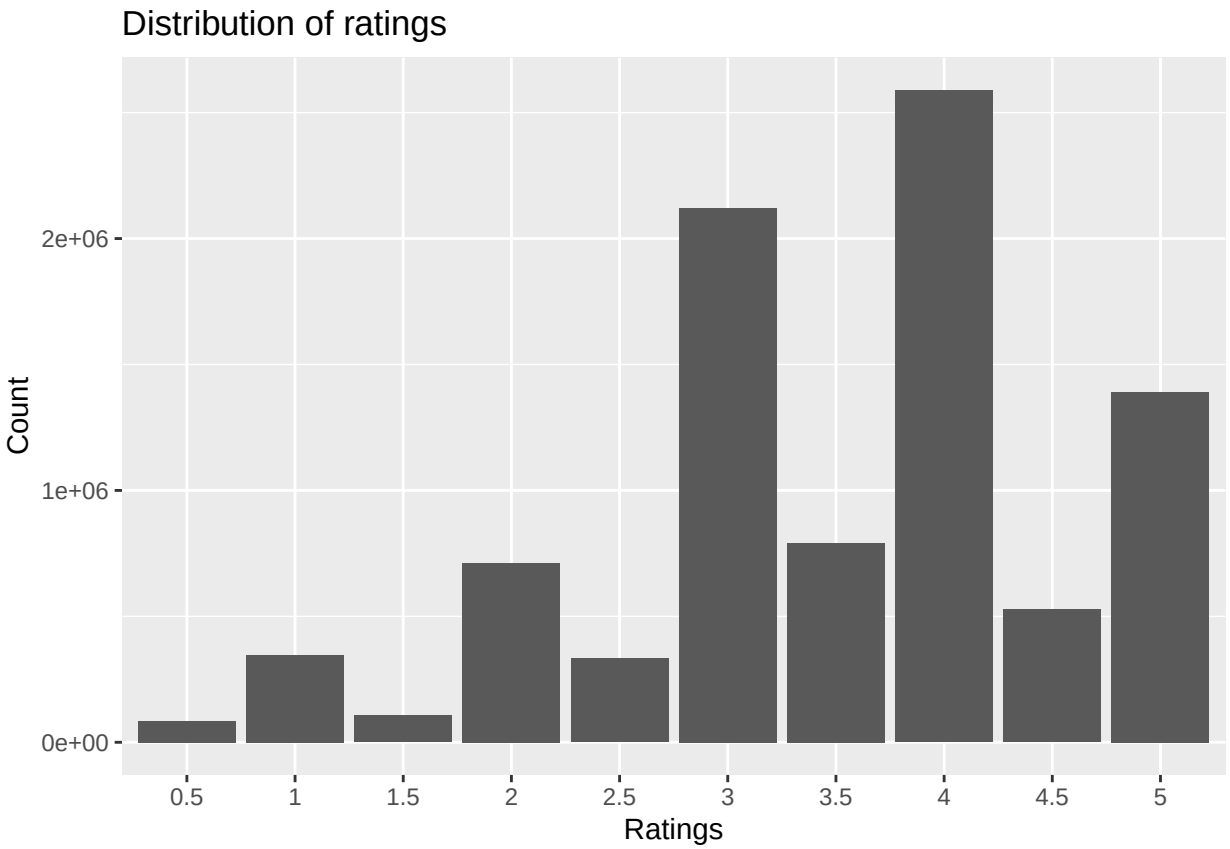
userId	Forrest Gump (1994)	Jurassic Park (1993)	Pulp Fiction (1994)	Shawshank Redemption, The (1994)	Silence of the Lambs, The (1991)
13	NA	NA	4	NA	NA
16	NA	3	NA	NA	NA
17	NA	NA	NA	NA	5
18	NA	3	5	4.5	5
19	4	1	NA	4.0	NA

Ratings profile

```
table_rating <- as.data.frame(table(edx$rating))
colnames(table_rating) <- c('Rating', 'Frequencies')

knitr::kable(table_rating)
```

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

Visualizing the ratings



Methods and Analysis

Initially, we have to create a test set using the **createDataPartition** function to make possible to assess the accuracy of the model we'll create. In this project, 10% of the data is assign to the test set and 90% to the train set.

Residual Mean Squared Error (RMSE)

Part important of this project is to provide the RMSE using only the training set (edx), and experimenting with multiple parameters. That strategy follows the Netflix challenge winning project that based their work on the residual mean squared error (RMSE) on the test set.

As Professor Rafael A. Irizarry explained in the course, *"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 first thing to do is to develop a function to compute the RMSE for vectors of ratings and their corresponding predictors.

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

Building up the model step-by-step

Step One

The ground base model is the simplest possible recommendation system that assumes the same rating for all movies. Considering that the estimate that minimizes the RMSE is the least squares estimate of u or the average of all rating.

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

```
## [1] 3.512465
```

Predicting all unknown ratings with u hat we obtain the following RMSE

```
step_one_rmse <- RMSE(validation$rating, mu_hat)
step_one_rmse
```

```
## [1] 1.061202
```

```
rmse_project_results <- data_frame(Method = "Step One/Base Model", RMSE = step_one_rmse)
```

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

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

Method	RMSE
Step One/Base Model	1.061202

Step Two

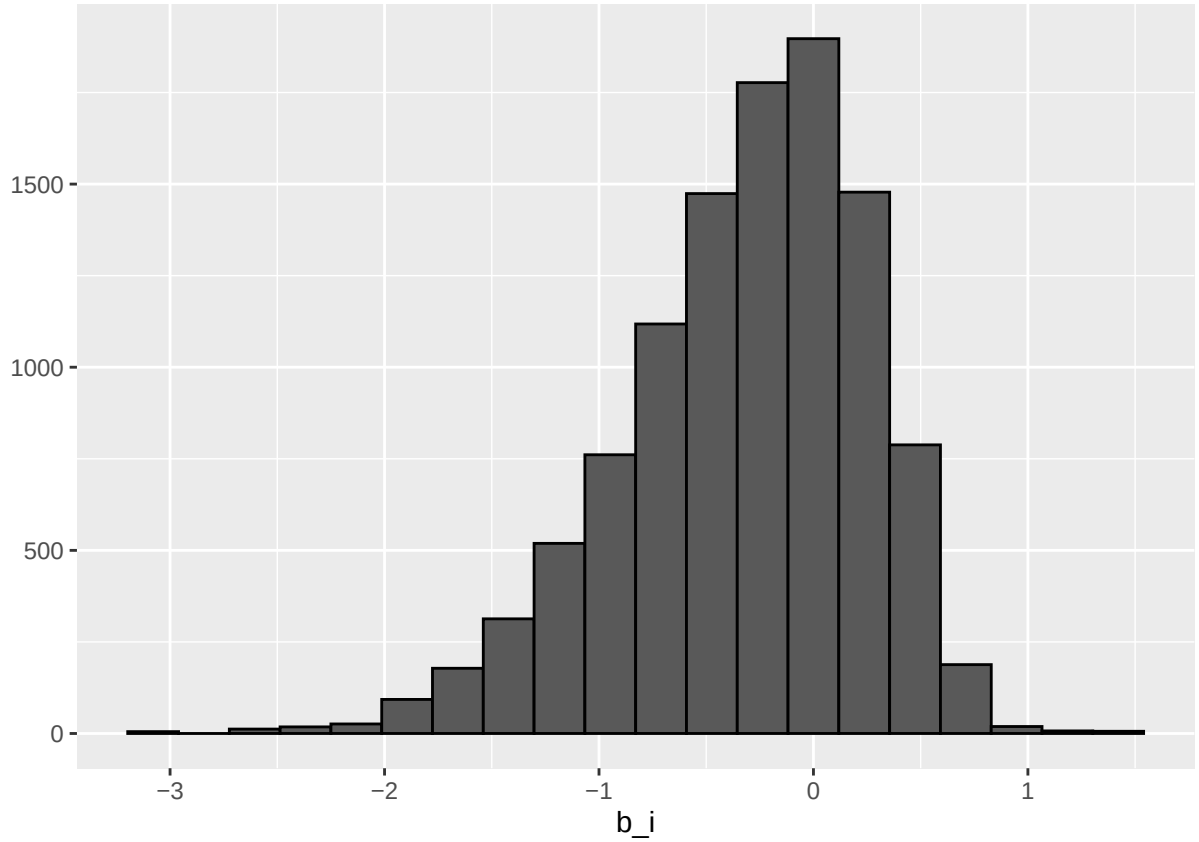
The second step aiming to develop the model is to add an 'effect' (or an 'bias', as referred in the Netflix challenge) to represent **average ranking** for movie i. We can estimate b(i), or the bias of film "i" using least squares.

#lm(rating ~ as.factor(movieID), data = movielens)

The use of linear regression and the lm() function would demand too much computation and time so we will consider that the least squares estimate b(i) is just the average of Y(u,i) - u_hat and opt for the code below

```
mu <- mean(edx$rating)

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

We can visualize the penalty term on the movie effect in a skewed histogram:



When we plug in the base formula of Y_hat(u, i) = U_hat + b_hat(i) we have

```
predicted_ratings_movie_bias <- mu + validation %>%
  left_join(movie_avgs, by = 'movieId') %>%
  pull(b_i)

step_two_rmse <- RMSE(predicted_ratings_movie_bias, validation$rating)
step_two_rmse
```

```
## [1] 0.9439087
```

```
rmse_project_results <- bind_rows(rmse_project_results, data_frame(Method = "Step Two/Movie Bias",
RMSE = step_two_rmse))

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

Method	RMSE
Step One/Base Model	1.0612018
Step Two/Movie Bias	0.9439087

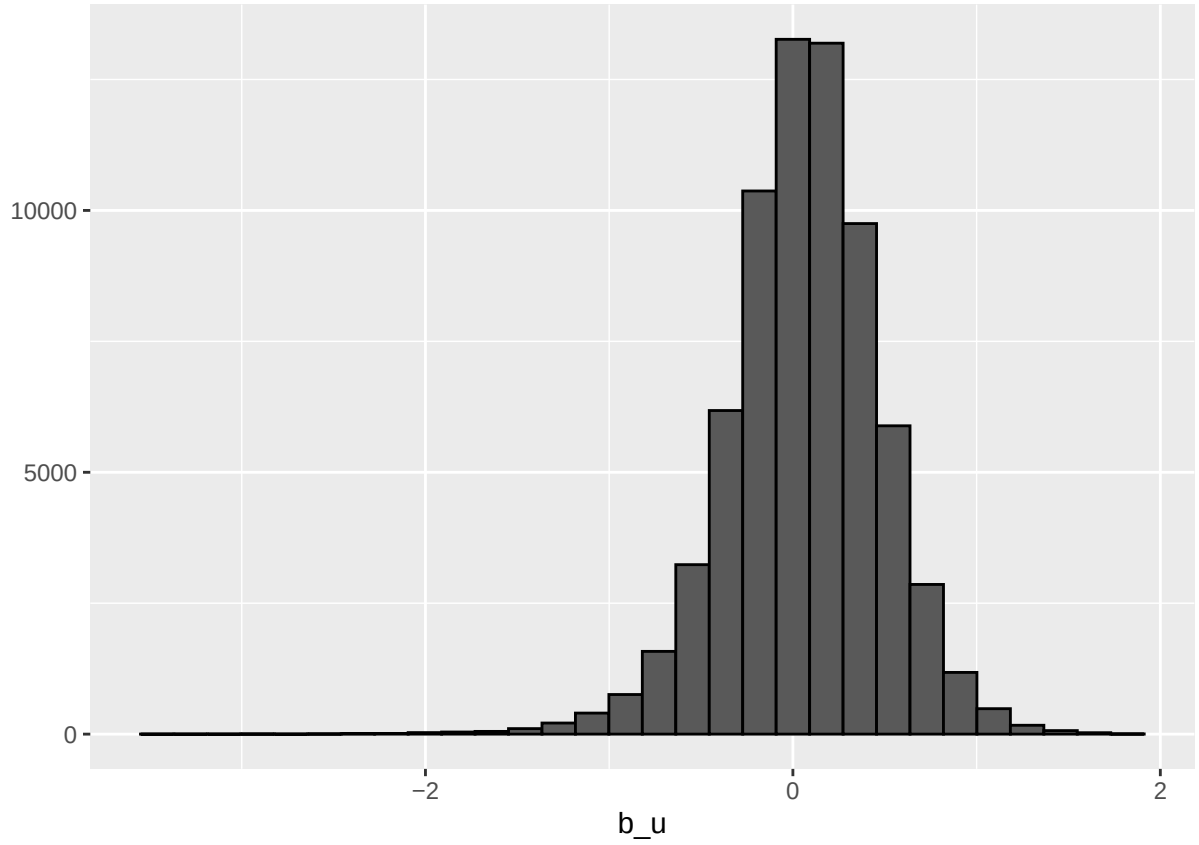
Step Three

To further improve the model, we can do something similar to what we did in *Step Two*, but this time with 'user bias', adding to the formula a "user-specific effect" (b_hat(u)). As before, we won't use linear regression because the lm() function will be very slow due to big number of bias (bs). Instead, we will use an approximation by computing u_hat and b_hat(i), and estimating b_hat(u) as the average of

y(u, i) - u_hat - b_hat(i)

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

We can visualize the penalty term on the user effect in a histogram:



Now we can apply it to build up a predictor and check if it improves the RMSE.

```
predicted_ratings_user_bias <- validation %>%
  left_join(movie_avgs, by = 'movieId') %>%
  left_join(user_avgs, by = 'userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

step_three_rmse <- RMSE(predicted_ratings_user_bias, validation$rating)
step_three_rmse
```

```
## [1] 0.8653488
```

Yes, it does!

```
rmse_project_results <- bind_rows(rmse_project_results, data_frame(Method = "Step Three/User Bias",
RMSE = step_three_rmse))

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

Method	RMSE
Step One/Base Model	1.0612018
Step Two/Movie Bias	0.9439087
Step Three/User Bias	0.8653488

Step Four

Despite the latest improvements, the performance of the model, so far, if applied to the test set, would rate unknown movies on the top, showing that we haven't achieved a good result yet.This happens because some movies have only a few ratings, from a few users, implying more uncertainty. As Professor Irizarry explained, 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 we would rather be conservative when unsure".*

Regularization

Here is when he introduces the concept of **regularization**, that permits us to penalize large estimates that are formed using small sample sizes, constraining the total variability of the effect sizes.

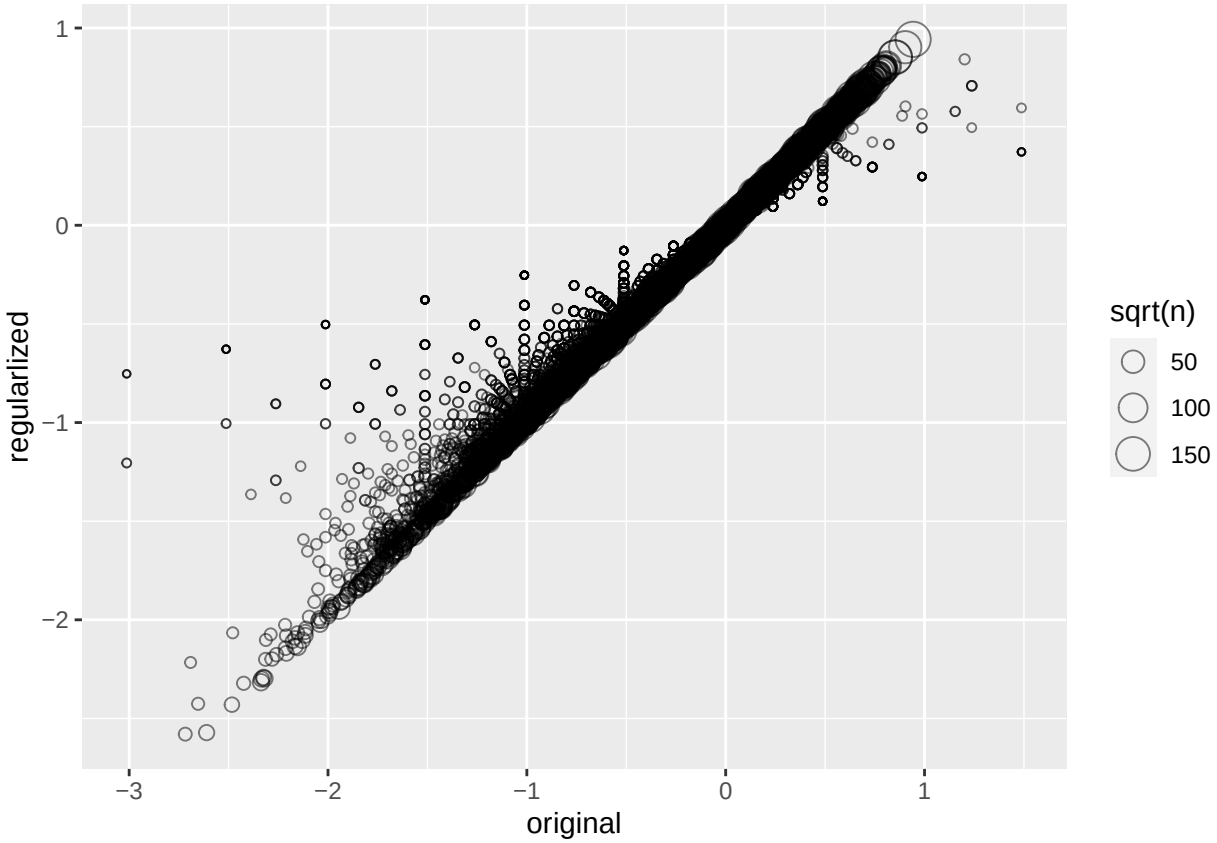
"The general idea of penalized regression is to control the total variability of the movie effects. Specifically, instead of minimizing the least squares equation, we minimize an equation that adds a penalty".

When our sample size is large, a case which will give us a stable estimate, then the penalty *lambda* is effectively ignored since n(i) + lambda = n. However, when the n(i) is small, then the estimate b_hat(i)(lambda) is shrunken towards 0. The larger lambda,the more we shrink.

Computing the regularized estimates of b(i) using lambda = 3 we have

```
lambda <- 3

mu <- mean(edx$rating)

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

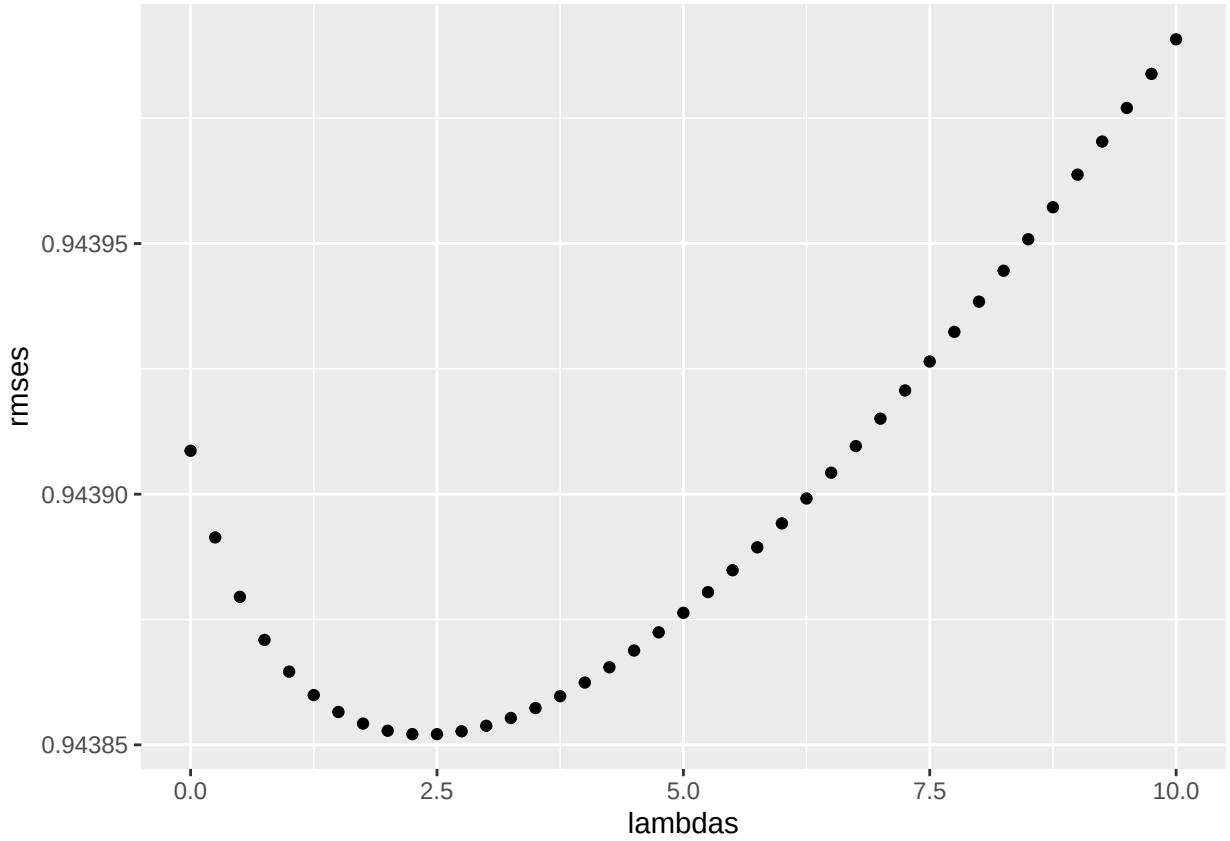
The following plot of the regularized estimates versus the least squares estimates show how the estimates shrink.



To further explore lambda as a tuning parameter we can use **cross-validation**.

```
lambdas <- seq(0, 10, 0.25)

mu <- mean(edx$rating)

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

rmses <- sapply(lambdas, function(l){
  predicted_ratings <- validation %>%
    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, validation$rating))
})
```

Plotting the result



And figuring out the which.min rmse

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

```
## [1] 2.5
```

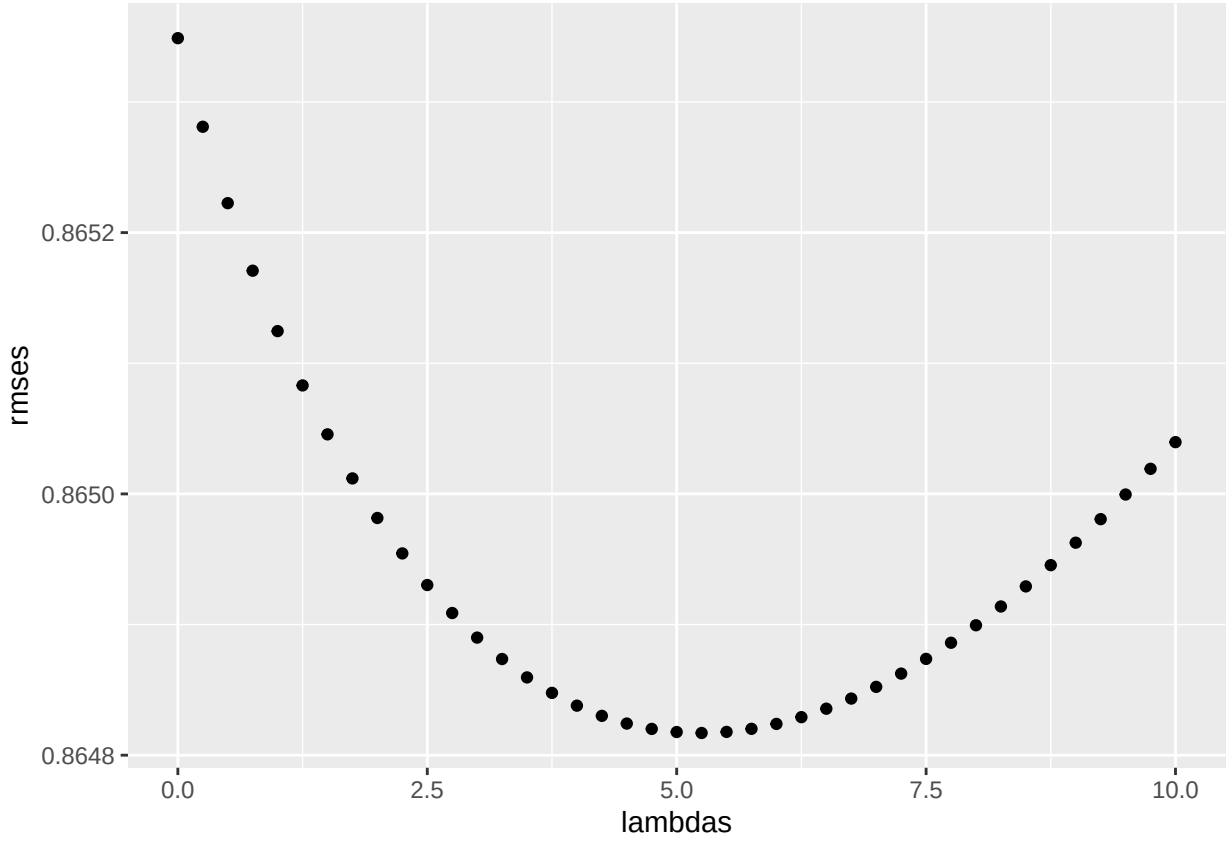
As this is shown in the course as a illustrative procedure, it's important to note that we should always use full cross-validation just on the train set, without using the test set until the final assessment. **The test set should never be used for tuning.**

We can use regularization for the estimate *user effects* as well. The estimates that minimizes this can be found similarly to what we did above. Here we use cross-validation to pick a lambda.

```
lambdas <- seq(0, 10, 0.25)

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))
})
```

To visualize



For the full model, the optimal lambda is:

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

```
## [1] 5.25
```

Results and Conclusion

RMSE Project Results

```
rmse_project_results <- bind_rows(rmse_project_results,
                          data_frame(Method = "Step Four/Regularized Movie + User Effect Model",
                                     RMSE = min(rmses)))

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

Method	RMSE
Step One/Base Model	1.0612018
Step Two/Movie Bias	0.9439087
Step Three/User Bias	0.8653488
Step Four/Regularized Movie + User Effect Model	0.8648170

The base model assumes the same rating for all movies and doesn't achieve a good enough result: its RMSE is more than 1, or in other words, an error of an entire star! To improve this result we added an 'effect' (or bias) representing the average ranking for *movie* i on Step Two and, following the same logic, an effect for *user* u on Step Three.

Step Three achieved a good improvement on the performance of the model with an RMSE of 0.8653488.

The problem here, though,is that when we apply it to the test set, it rates unknown movies on the top, showing that something wasn't quite right.

To sort this out, we've moved ahead to Step Four, applying the concept of **regularization**, penalizing large estimates that are formed using small sample sizes, and adjusting for noisy estimates that were withholding the model to perform better, reaching a final RMSE of 0.864817.