Movielens project

Guillermo Pe??a

05/01/2021

Introduction

One of the numerous applications of machine learning in information technology is the ability to make recommendations of items or services to potential to users in general. In the year 2006, Netflix offered a challenge to the data science community. The challenge was to improve Netflix's in house software by 10% with a prize of $1M.

This report contains problem definition, data ingestion, exploratory analysis, modeling and data analysis, results and conclusion. The evaluation criteria for this algorithm is a RMSE expected to be lower than 0.9. The function that computes the RMSE for ratings and their predictors will be:

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

Finally, the best resulting model will be used to predict the movie ratings.

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-pro
ject.org")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages ---------------------------------------------------
------------------------------------------------------------------ tidyverse 1.3.
0 --
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   1.0.0
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------------------------
---------------------------------------------------------- tidyverse_conflicts(
) --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org
")
```

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked _by_ '.GlobalEnv':
##
##     RMSE
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-p
roject.org")
```

```
## Loading required package: 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
```

```
if(!require(lubridate)) install.packages("lubridate")
```

```
## Loading required package: 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
```

```
if(!require(ggthemes)) install.packages("ggthemes")
```

```
## Loading required package: ggthemes
```

```
## Warning: package 'ggthemes' was built under R version 4.0.2
```

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

# 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.da
t"))),
                 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")

# using R 4.0
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

#Joining data
movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of the movieLens data
set.seed(1,sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list =
FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# userId and movieId in validation set as in edx set assurance
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)

# Remove objects we are not using
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

In order to predict accurately the movie rating of the users that haven???t seen the movie yet, the he MovieLens dataset will be splitted into 2 subsets. Named ???edx???, a training subset to train the algorithm, and ???validation???, a subset to test the movie ratings.

Methods and Analysis

Data Analysis

```
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046               Boomerang (1992)
## 2:      1     185      5 838983525                Net, The (1995)
## 3:      1     292      5 838983421                Outbreak (1995)
## 4:      1     316      5 838983392                Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

```
summarize(edx)
```

```
## data frame with 0 columns and 1 row
```

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

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

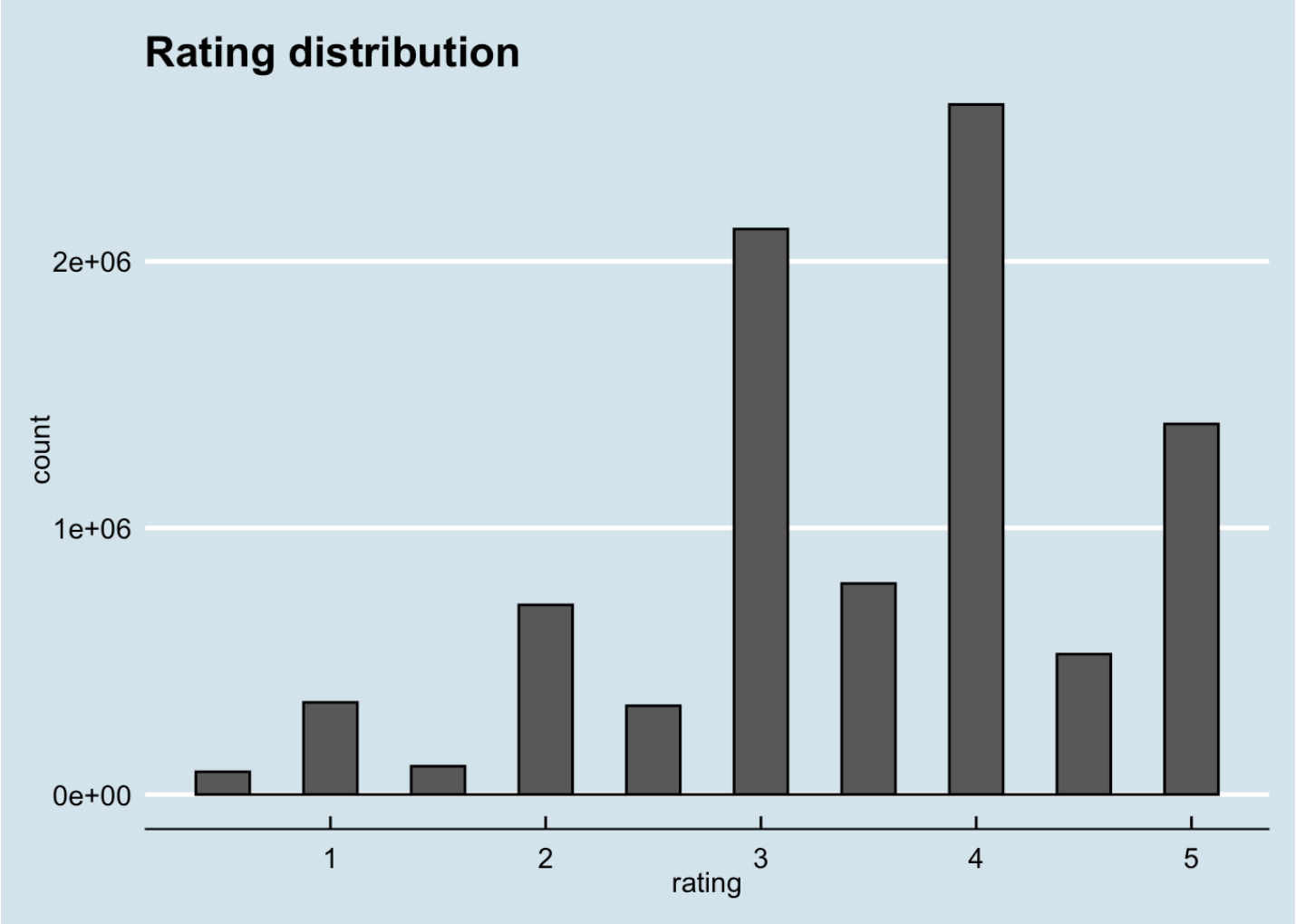
This gives us a good summary of what the data set looks like.

Data pre-processing and exploratory analysis

Some users are positive or negative because of their own personal taste regardless of the movie. The model should include a user bias. The popularity of the movie genre depends on time. So we should also account for the time dependent analysis. The model should find genre popularity over the years.

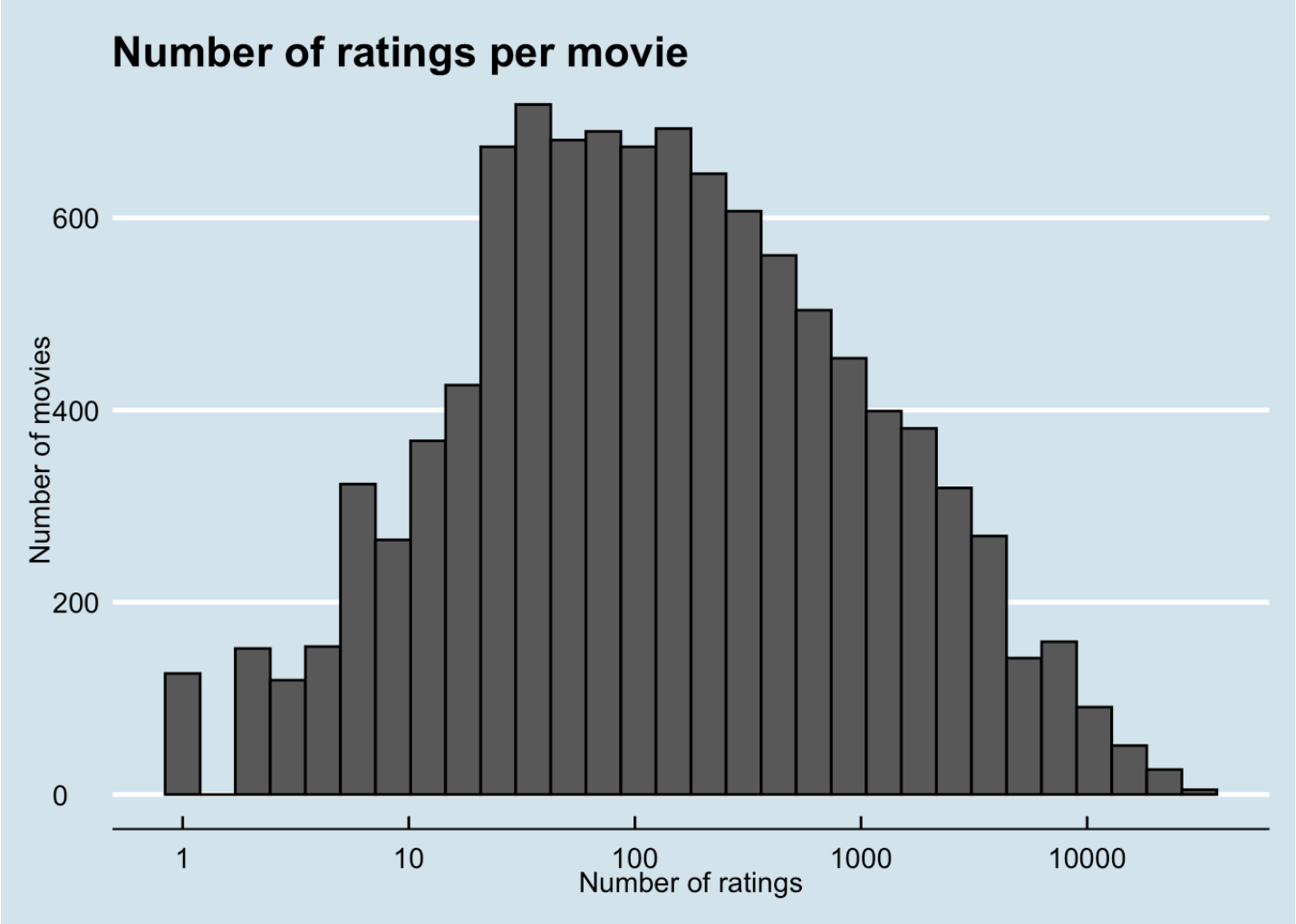
We have a look at the ratings distribution.

```
edx %>%
  ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.25, color = "black") +
  ggtitle("Rating distribution") +
  theme_economist(base_size = 10, base_family = "sans",
  horizontal = TRUE, dkpanel = FALSE)
```



A good visual summary for the expected quality of the analysis can be given by the number of ratings per movie.

```
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  xlab("Number of ratings") +
  ylab("Number of movies") +
  ggtitle("Number of ratings per movie") +
  theme_economist(base_size = 10, base_family = "sans",
  horizontal = TRUE, dkpanel = FALSE)
```



We can see some movies that only have one rating, this fact will compromise the quality of the given recommendations. Fortunately it is a minority.

```
edx %>%
  group_by(movieId) %>%
  summarize(count = n()) %>%
  filter(count == 1) %>%
  left_join(edx, by = "movieId") %>%
  group_by(title) %>%
  summarize(rating = rating, n_rating = count) %>%
  slice(1:20) %>%
  knitr::kable()
```

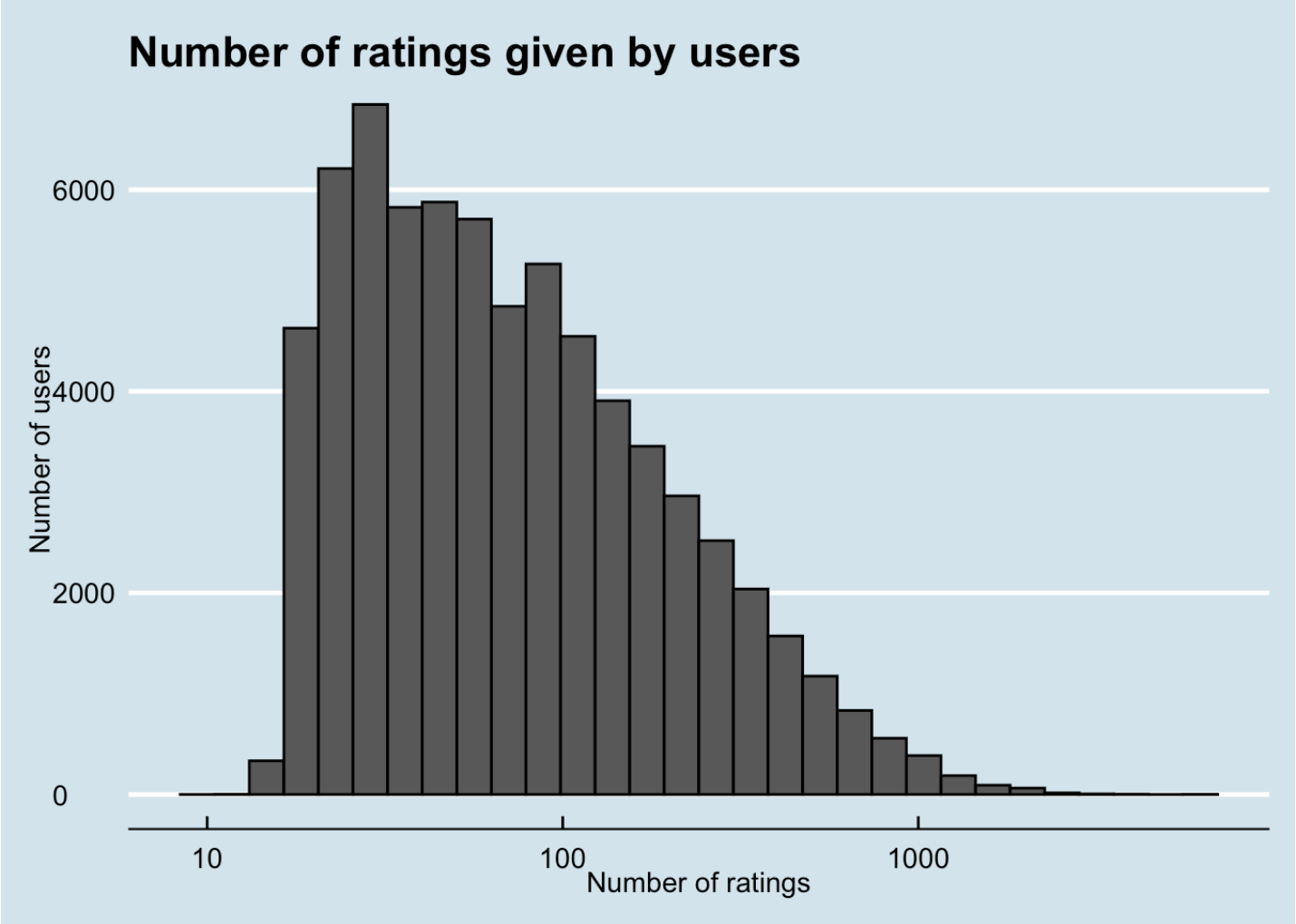
```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
```

title	rating	n_rating
1, 2, 3, Sun (Un, deuz, trois, soleil) (1993)	2.0	1
100 Feet (2008)	2.0	1
4 (2005)	2.5	1

Accused (Anklaget) (2005)	0.5	1
Ace of Hearts (2008)	2.0	1
Ace of Hearts, The (1921)	3.5	1
Adios, Sabata (Indio Black, sai che ti dico: Sei un gran figlio di...) (1971)	1.5	1
Africa addio (1966)	3.0	1
Aleksandra (2007)	3.0	1
Bad Blood (Mauvais sang) (1986)	4.5	1
Battle of Russia, The (Why We Fight, 5) (1943)	3.5	1
Bellissima (1951)	4.0	1
Big Fella (1937)	3.0	1
Black Tights (1-2-3-4 ou Les Collants noirs) (1960)	3.0	1
Blind Shaft (Mang jing) (2003)	2.5	1
Blue Light, The (Das Blaue Licht) (1932)	5.0	1
Borderline (1950)	3.0	1
Brothers of the Head (2005)	2.5	1
Chapayev (1934)	1.5	1
Cold Sweat (De la part des copains) (1970)	2.5	1

We can have a look on the number of ratings given by users

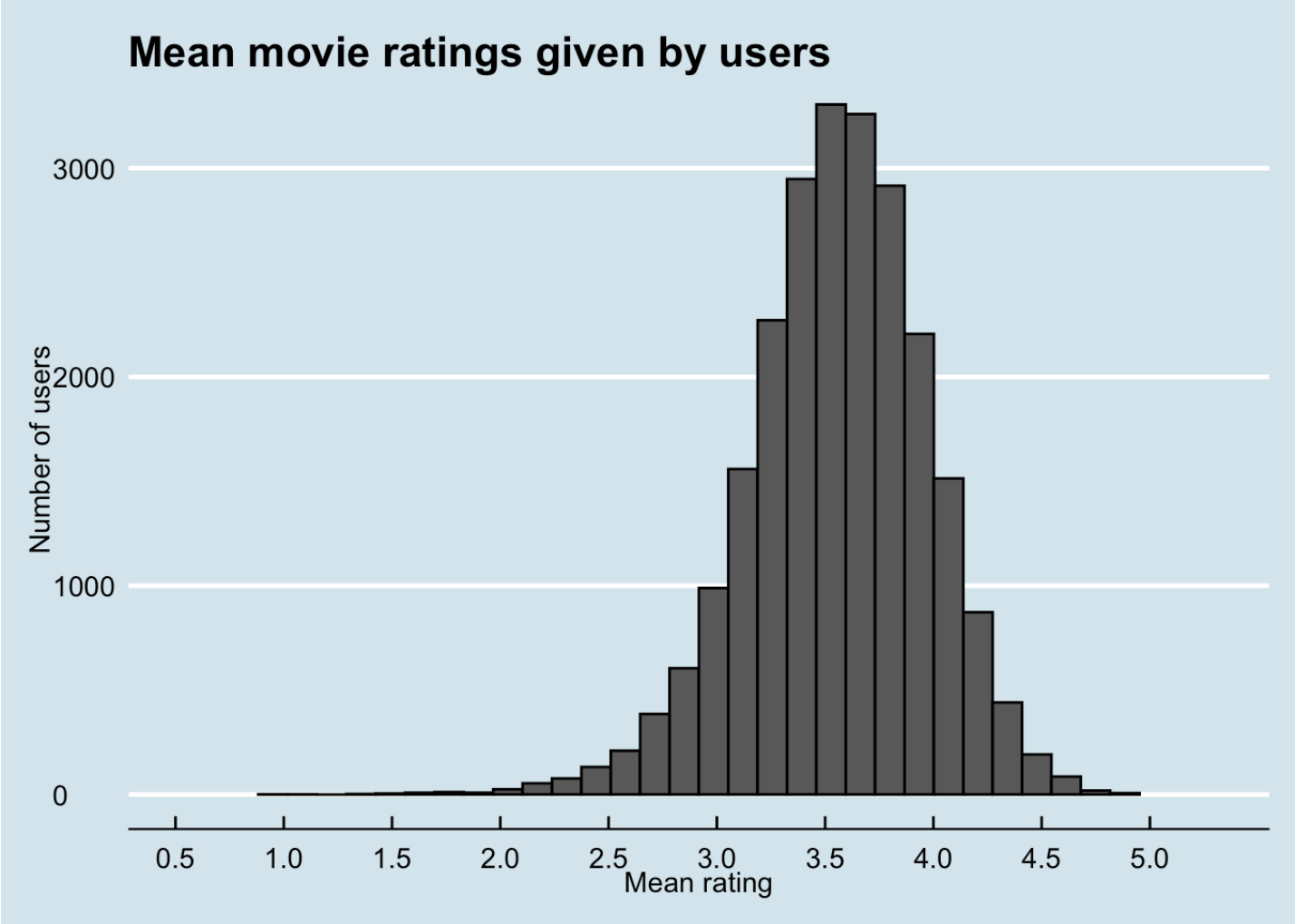
```
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  xlab("Number of ratings") +
  ylab("Number of users") +
  ggtitle("Number of ratings given by users") +
  theme_economist(base_size = 10, base_family = "sans",
  horizontal = TRUE, dkpanel = FALSE)
```




```
# Plot mean movie ratings given by users
edx %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black") +
  xlab("Mean rating") +
  ylab("Number of users") +
  ggtitle("Mean movie ratings given by users") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  theme_economist(base_size = 10, base_family = "sans",
  horizontal = TRUE, dkpanel = FALSE)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## Warning: Continuous limits supplied to discrete scale.
## Did you mean `limits = factor(...)` or `scale_*_continuous()`?
```



We are going to compute different modelling approaches and see which one performs best.

Firstly, we are going to predict ratings using just the average rating which is 3.512465.

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

model_avg_rmse <- RMSE(validation$rating, avg)

model_avg_rmse
```

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

```
rmse_results_df <- data_frame(method_used = "Average movie rating model", RMSE = mo
del_avg_rmse)
```

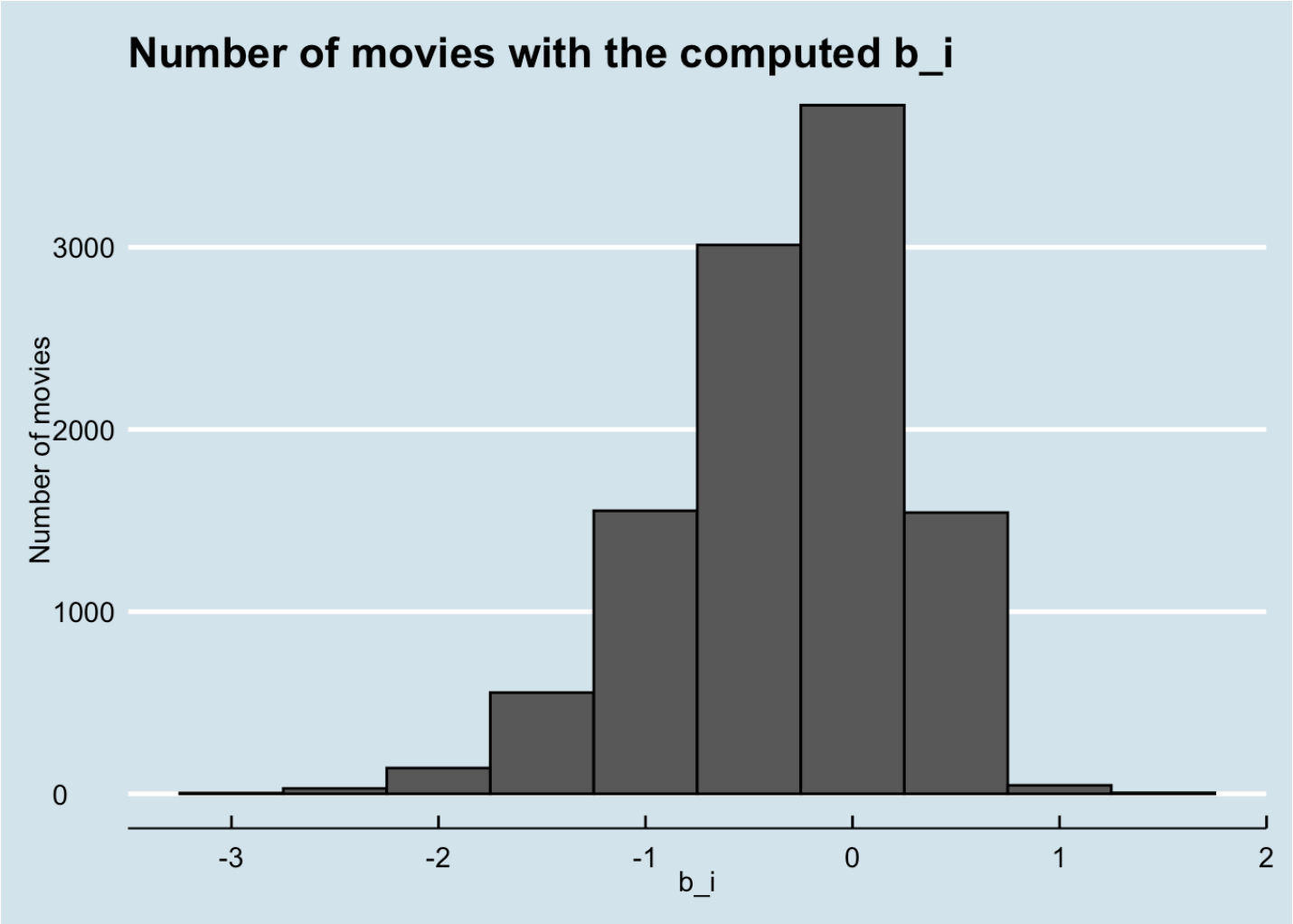
```
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

Different movies are rated differently. As shown, the histogram is not symmetric. It is skewed towards a negative rating. So the movie effect can be taken into account with the difference from the average rating.

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

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"
), ylab = "Number of movies", main = "Number of movies with the computed b_i") +
                theme_economist(base_size = 10, base_family = "sans", horizontal =
TRUE, dkpanel = FALSE)
```




```
predicted_ratings <- avg +  validation %>% left_join(movie_avgs, by='movieId') %>%
pull(b_i)
model_moviebias_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results_df <- bind_rows(rmse_results_df, data_frame(method_used="Movie bias ef
fect model", RMSE = model_moviebias_rmse ))
model_moviebias_rmse
```

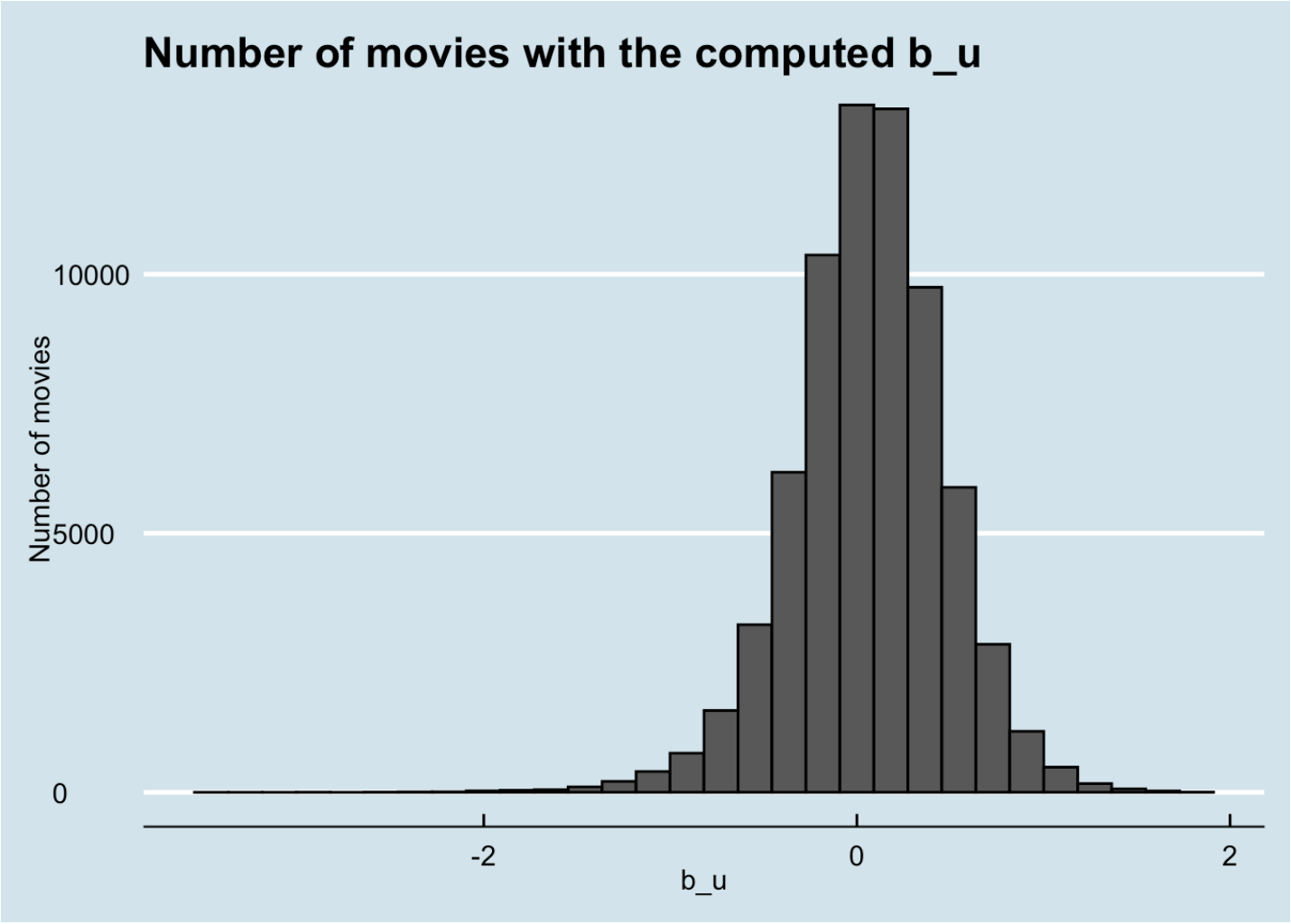
```
## [1] 0.9439087
```

Some movies are rated more often than others. The model should incorporate a movie bias aspect into it. This is a model taking into account the movie effect b_i, as said before, we achieve this by subtracting the rating minus the mean for each rating that the movie received. We already see an improved rmse factor with this simple approach.

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

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
user_avgs %>% qplot(b_u, geom ="histogram", bins = 30, data = ., color = I("black")
, ylab = "Number of movies", main = "Number of movies with the computed b_u") +
              theme_economist(base_size = 10, base_family = "sans", horizontal = TR
UE, dkpanel = FALSE)
```



```
predicted_ratings <- validation%>% left_join(movie_avgs, by='movieId') %>% left_joi
n(user_avgs, by='userId') %>% mutate(pred = avg + b_i + b_u) %>% pull(pred)

model_movieuser_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results_df <- bind_rows(rmse_results_df, data_frame(method_used="Movie and use
r bias effect model", RMSE = model_movieuser_rmse))
```

```
model_movieuser_rmse
```

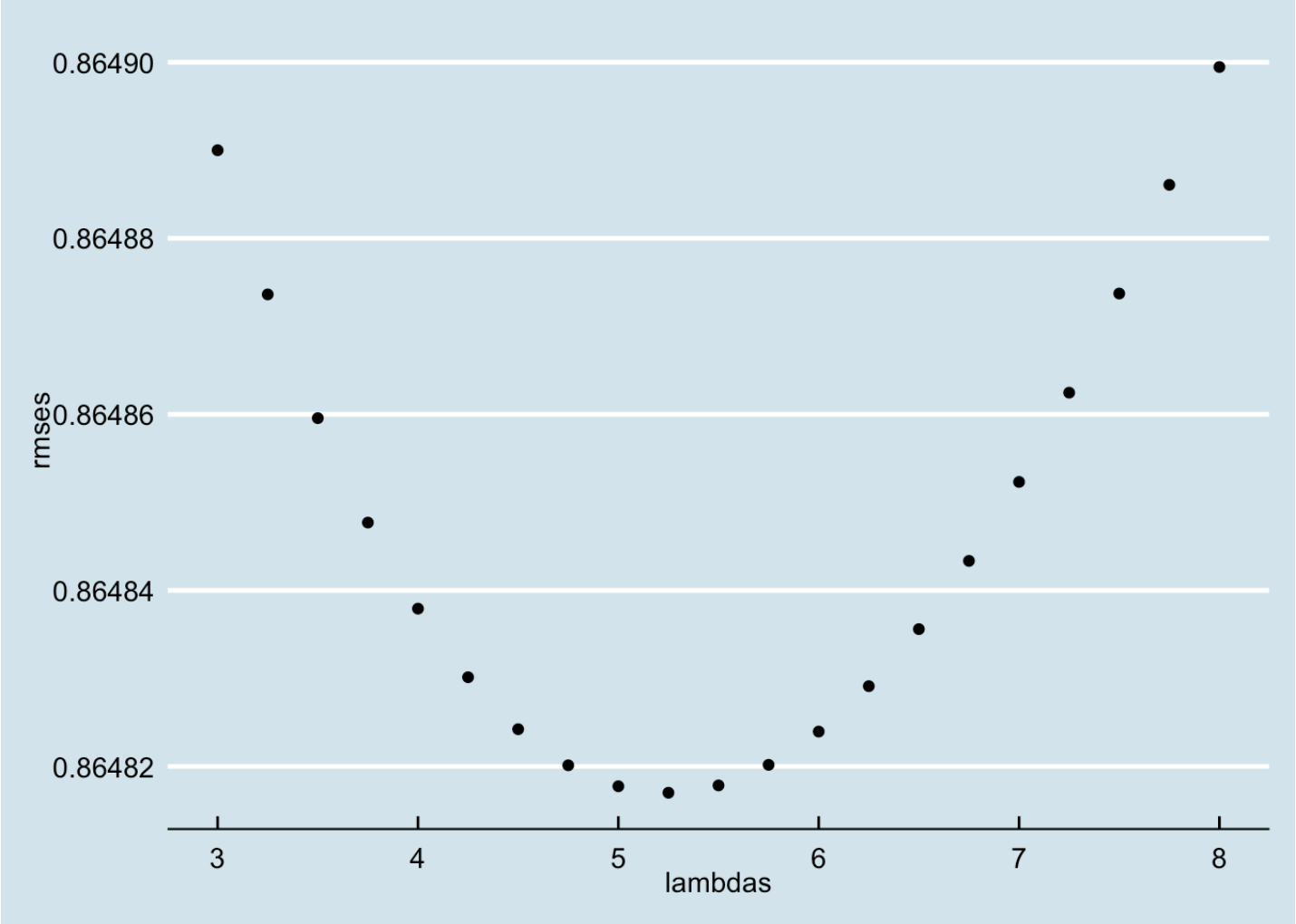
```
## [1] 0.8653488
```

Now, we regularize the user and movie effects adding a penalty factor, named lambda (the tuning parameter). We define a number of values for lambda and for each lambda we will find b_i & b_u, which will be used to find predicted values. The lambda that minimizes the RMSE should be chosen.

```
lambdas <- seq(3, 8, 0.25)

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

                  avg <- mean(edx$rating)

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

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

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

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

```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
qplot(lambdas, rmses) + theme_economist(base_size = 10, base_family = "sans", horiz
ontal = TRUE, dkpanel = FALSE)
```




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

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

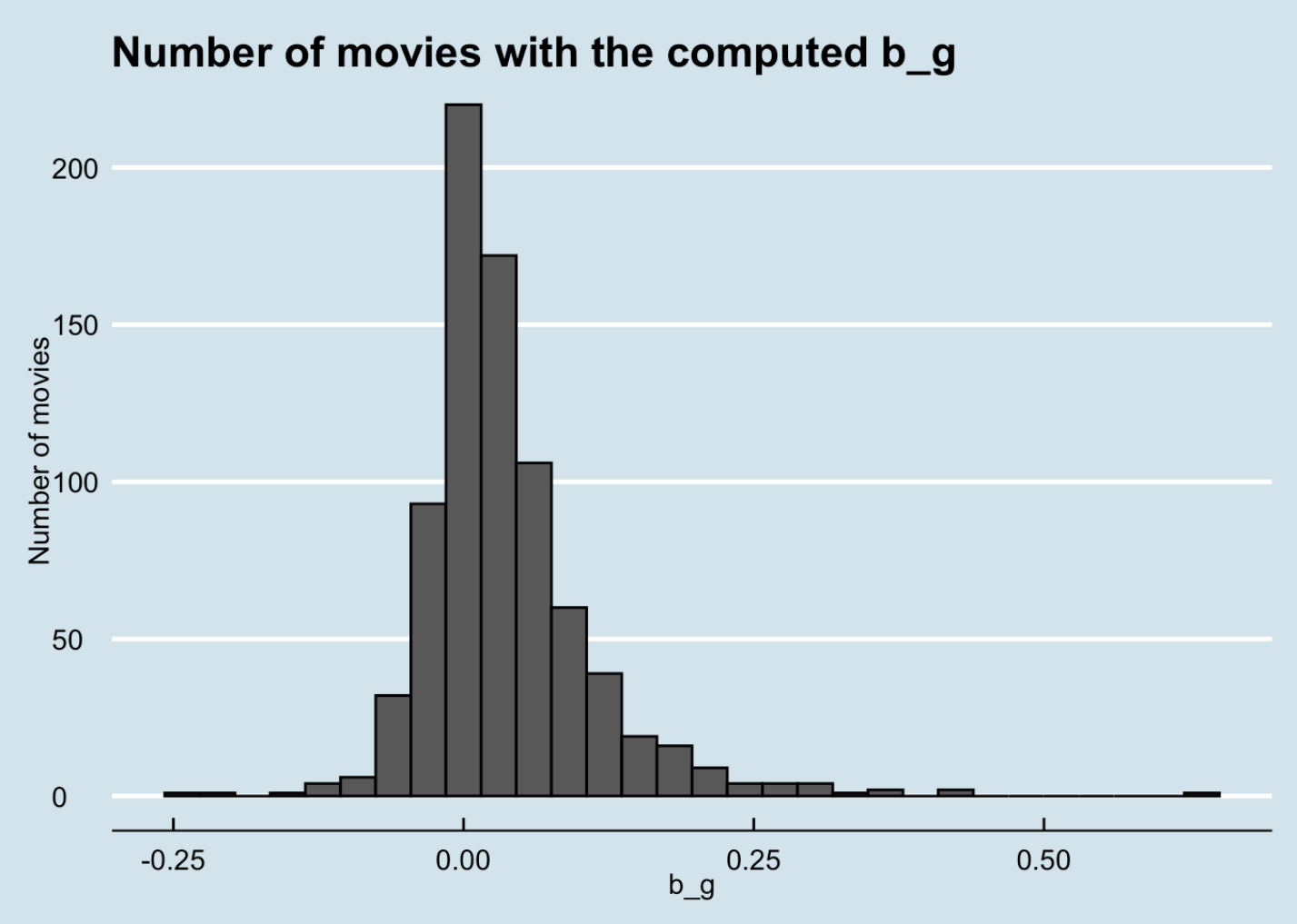
```
rmse_results_df <- bind_rows(rmse_results_df, data_frame(method_used="Regularized m
ovie and user bias effect model", RMSE = min(rmses)))
```

Now let's see if there is a genre bias that we can add to our model.

```
genre_avgs <- edx %>% left_join(movie_avgs, by='movieId') %>% left_join(user_avgs,
by='userId') %>% group_by(genres)  %>% summarize(b_g = mean(rating - avg - b_i - b_
u))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
genre_avgs %>% qplot(b_g, geom ="histogram", bins = 30, data = ., color = I("black"
), ylab = "Number of movies", main = "Number of movies with the computed b_g") +
              theme_economist(base_size = 10, base_family = "sans", horizontal = TR
UE, dkpanel = FALSE)
```



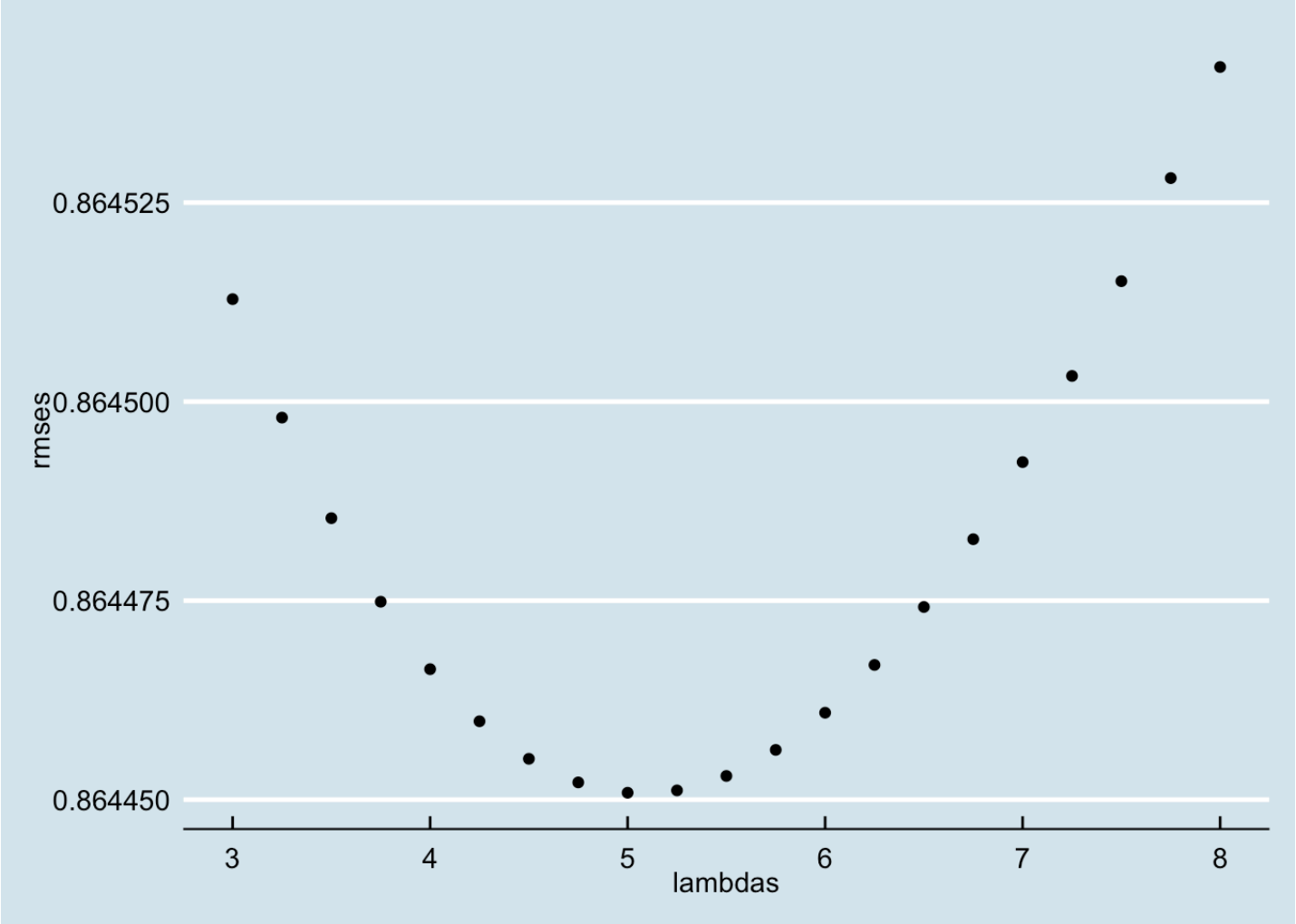
We can see a positive bias when we compute genre. Which makes sense since there are genres more liked than others in users.

```
lambdas <- seq(3, 8, 0.25)

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

                  avg <- mean(edx$rating)

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

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

                  b_g <- edx %>% left_join(b_i, by='movieId') %>% left_join(b_u, by
='userId') %>% group_by(genres)  %>% summarize(b_g = mean(rating - avg - b_i - b_u)
)

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

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

```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
qplot(lambdas, rmses) + theme_economist(base_size = 10, base_family = "sans", horiz
ontal = TRUE, dkpanel = FALSE)
```




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

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

```
rmse_results_df <- bind_rows(rmse_results_df, data_frame(method_used="Regularized w
ith genre, movie and user bias effect model", RMSE = min(rmses)))
```

We see that we get another step towards minimizing rmse if we incorporate in the model a feature accounting the genre bias.

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

method_used	RMSE
Average movie rating model	1.0612018
Movie bias effect model	0.9439087
Movie and user bias effect model	0.8653488

Regularized movie and user bias effect model	0.8648170
Regularized with genre, movie and user bias effect model	0.8644509

In conclusion, we have seen that indeed it's farily simple to build a successful recommendation system.

As shown by the results, when we include more insights into the model, the rmse lowers.

The regularized with genre, movie and user effect model turns to be the best model in terms of delivered RMSE.

In future works it has been suggested by other students to use the "recosystem" package, which seems to have specific tools that optimizes the type of analysis that recomendation systems work on. The results that some of them are achieving are quite impressing and look like the way to go forward.

```
version
```

```
##                _
## platform       x86_64-apple-darwin17.0
## arch           x86_64
## os             darwin17.0
## system         x86_64, darwin17.0
## status
## major          4
## minor          0.0
## year           2020
## month          04
## day            24
## svn rev        78286
## language       R
## version.string R version 4.0.0 (2020-04-24)
## nickname       Arbor Day
```