MovieLensProject

Finn Bartels

01/08/2021

1. Introduction

A look on our data

This project is about using analysis and machine learning methods to predict movie ratings. These movie rating prediction help recommending movies to users. The project is part of the 'HarvardX: PH125.9x Data Science: Capstone'.

To setup the data we install required packages and download the data:

```
##########################################################
# Create edx set, validation set (final hold-out test 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")

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

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
```

```r
# 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 3.6 or earlier:
# movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
#                                            title = as.character(title),
#                                            genres = as.character(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") # if using R 3.5 or earlier, use `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)
edx <- rbind(edx, removed)

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

The data is a set of 10 million ratings from the movielens project. For the final test the data was splitted up into two dataframes: trainingdata: edx (90%), testingdata: validation (10%).
First of all we will analyze the given data, investigating how the ratings are distributed in relation to the users, movies and genres. After that we will apply the **RMSE** (Root Mean Square Error) on several effects that possibly have an influence on the ratings. Finally we will apply regularization.

First we want to review the data we will work with:

```r
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
```

As we can see, in the dataframe, each row consists of user, movie, rating, timestamp, title and genres. Where the genres column is a composion of each genre, in which the movie is in. UserId's and MovieId's can appear in multiple rows as we know a row in the dataset consists of a single rating made by one user for one movie.

Goal

In this project we will use the root mean squared error **(RMSE)** as the loss-function. The goal is to get the **RMSE** as low as possible. To reach this goal we will work with different approaches and by taking the attributes movie, user and genres into account we will define different effects for each of those values. Finally we will work with regularization solving the problems that the other approaches ignore. The goal is to get an **RMSE** < 0.86490.

2. Analysis and methods

Analysis of the data

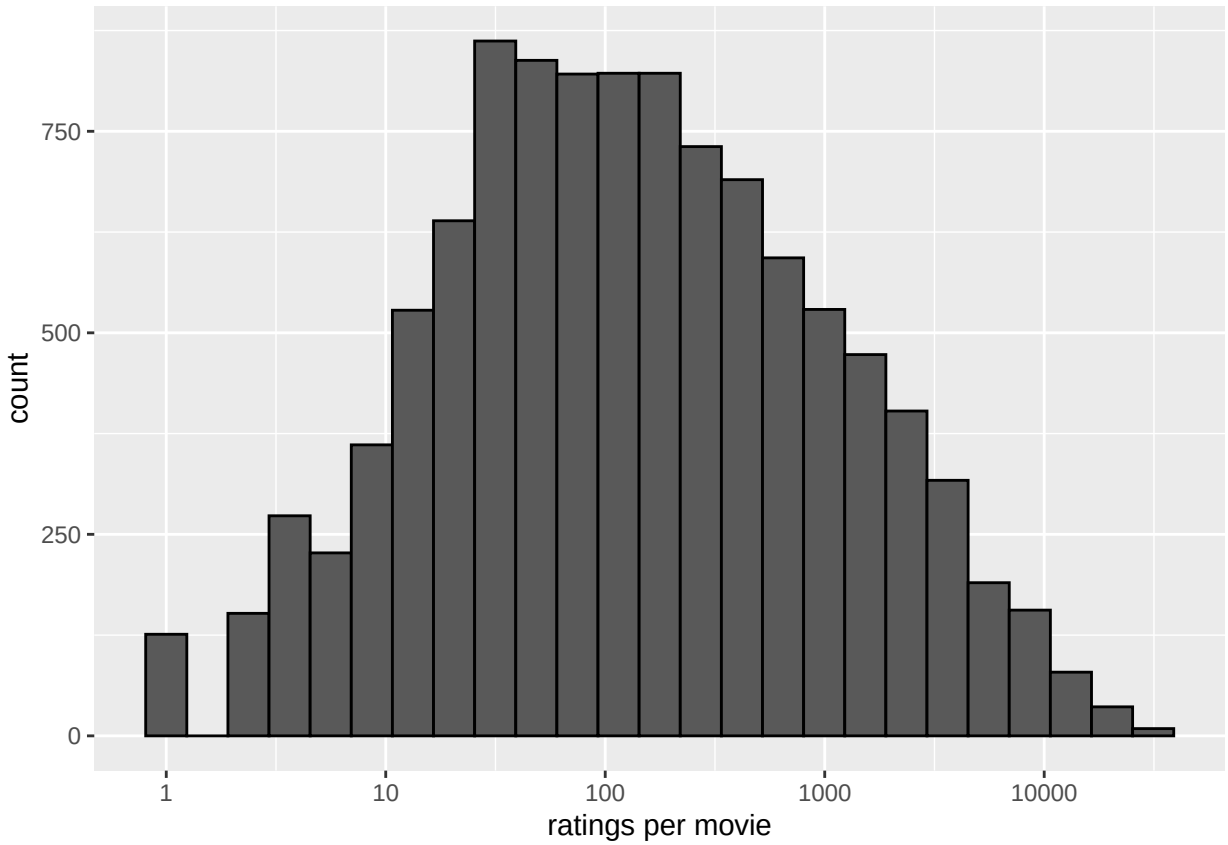
Let us take a look on ratings. As we can see the most movies were rated quite a few times. There are 10677 different movies in this dataframe.

```
tibble(unique_movies=nrow(distinct(edx, movieId)))
```

```
## # A tibble: 1 x 1
##   unique_movies
##           <int>
## 1         10677
```

```
edx %>%
  group_by(movieId) %>%
  summarize(ratings_per_movie=n()) %>%
  ggplot(aes(ratings_per_movie)) +
  geom_histogram(color="black", bins = 25) +
  scale_x_log10() +
  xlab("ratings per movie")
```

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



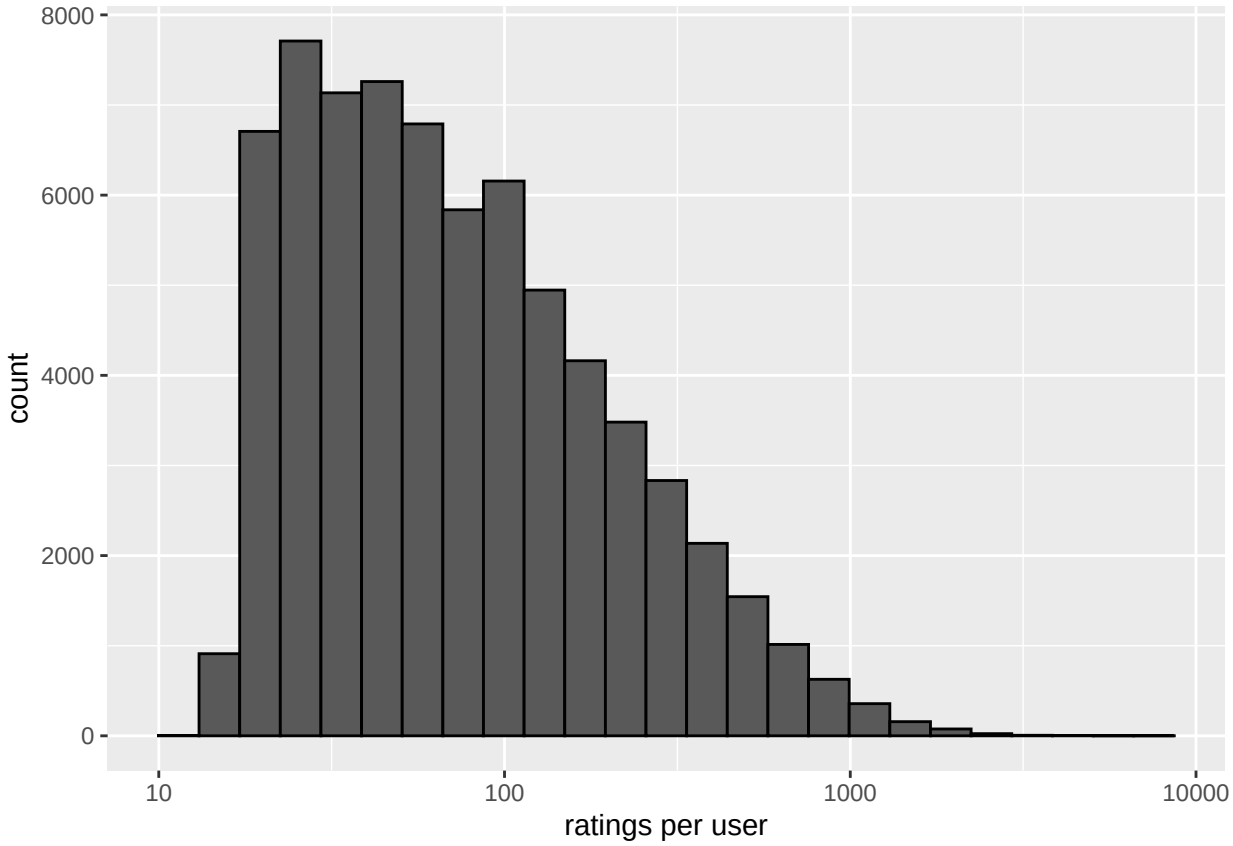
Let us also take a look on the ratings per user and that count. 69878 unique users rated at least once.

```
tibble(unique_users=nrow(distinct(edx, userId)))
```

```
## # A tibble: 1 x 1
##   unique_users
##          <int>
## 1        69878
```

```
edx %>%
  group_by(userId) %>%
  summarize(ratings_per_user=n()) %>%
  ggplot(aes(ratings_per_user)) +
  geom_histogram(color="black", bins=25) +
  scale_x_log10() +
  xlab("ratings per user")
```

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



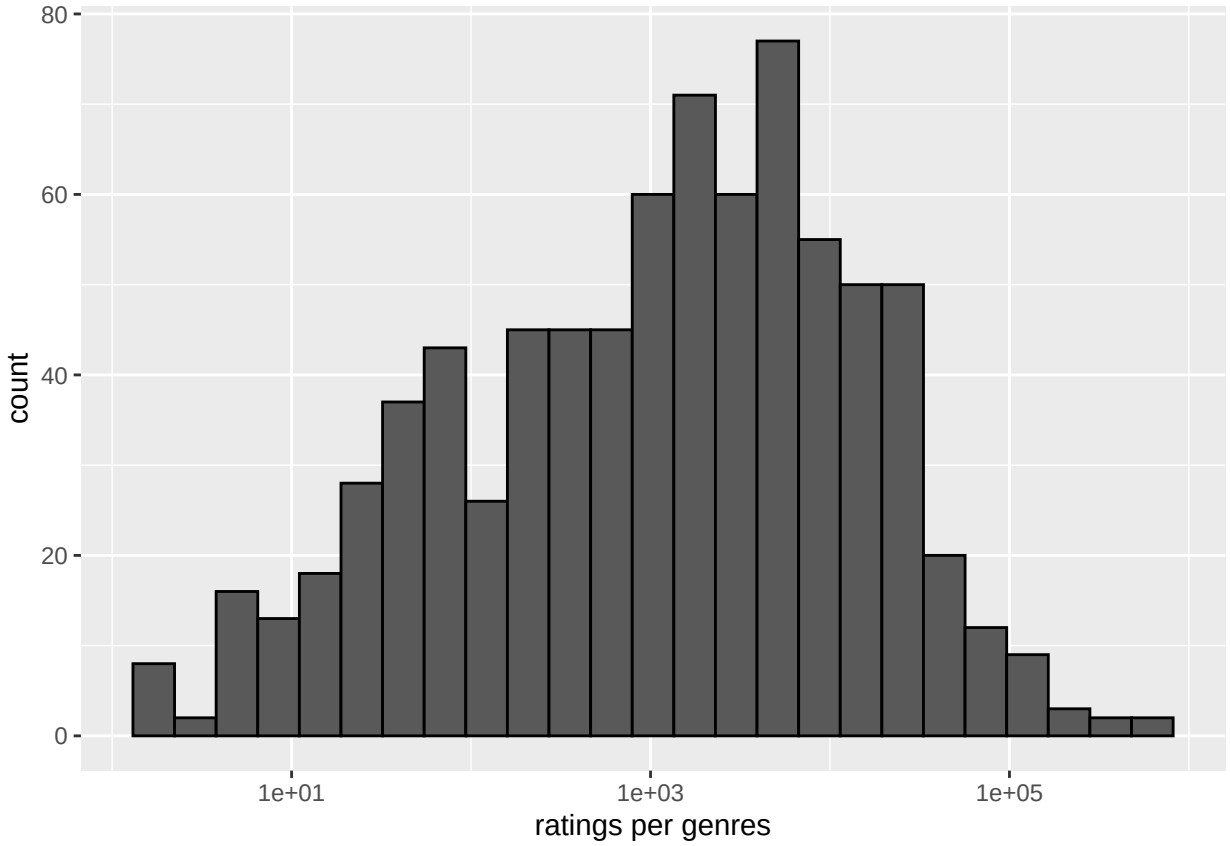
Furthermore we show the ratings for the combinations of genres. The dataframe consists of 797 different combinations of genres.

```
tibble(unique_genres=nrow(distinct(edx, genres)))
```

```
## # A tibble: 1 x 1
##   unique_genres
##           <int>
## 1           797
```

```
edx %>%
  group_by(genres) %>%
  summarize(ratings_per_genres=n()) %>%
  ggplot(aes(ratings_per_genres)) +
  geom_histogram(color="black", bins=25) +
  scale_x_log10() +
  xlab("ratings per genres")
```

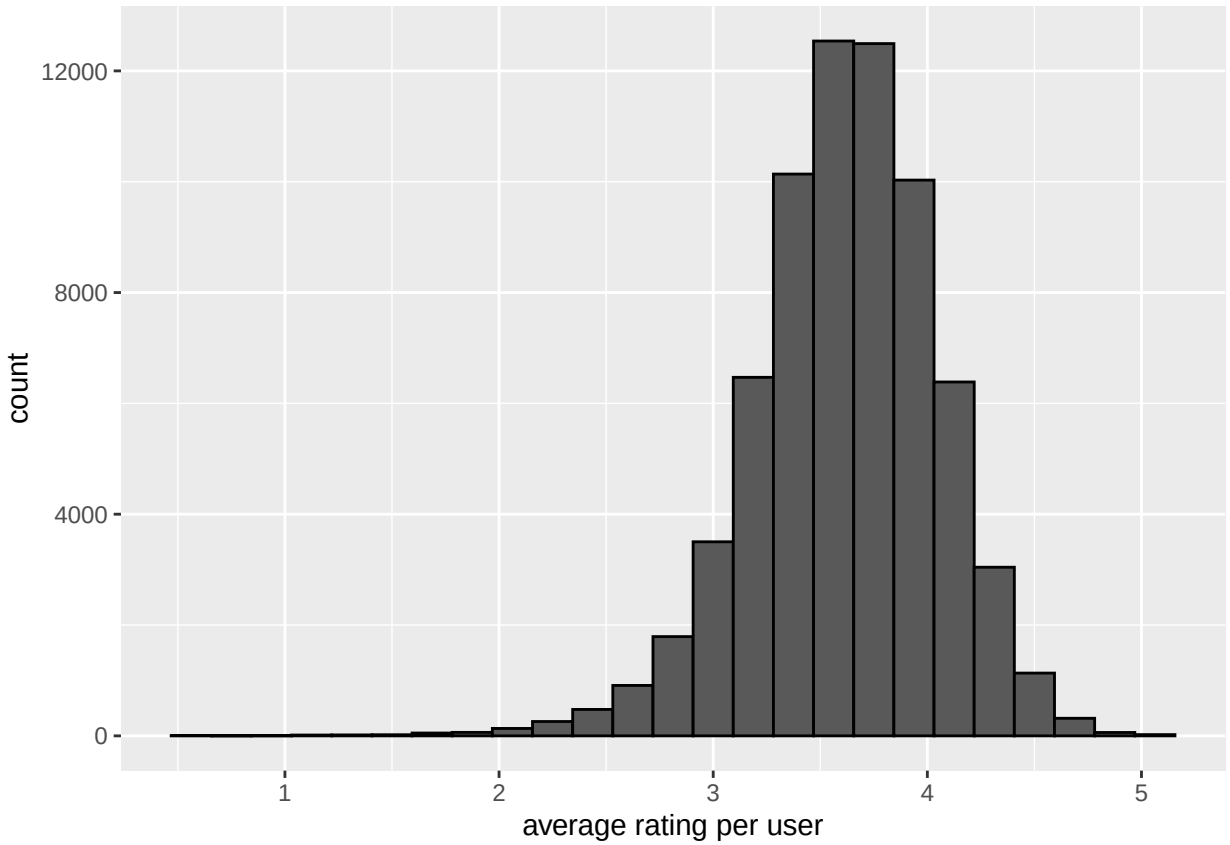
```
## `summarise()` ungrouping output (override with `.groups` argument)
```



We also take a quick look on the average ratings per user:

```
edx %>%
  group_by(userId) %>%
  summarize(mean_user_ratings=mean(rating)) %>%
  ggplot(aes(mean_user_ratings)) +
  geom_histogram(color="black", bins=25) +
  xlab("average rating per user")
```

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



This is close to what we see from the **average rating** from every review:

```
mean(edx$rating)
```

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

Modeling

We will use the average rating in the first method too, but before that, we will split the edx dataframe into a train and test set:

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

As I mentioned before,I use the **RMSE** to see how well we are doing and trying to get it as low as possible. We use a formula of the form:

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

with **N** as the number of user-movie combinations, $y_{u,i}$ as rating for movie i by user u and $\hat{y}_{u,i}$ as the prediction.
The function for **RMSE** is defined as follows:

```
RMSE <- function(true_ratings, predicted_ratings){
  s <- sqrt(mean((true_ratings - predicted_ratings)^2, na.rm=TRUE))
  s
}
```

First approach Now it is time for the first approach, in which we assume to have the same rating for all movies and users:

$$Y_{u,i} = \mu + \epsilon_{u,i}$$

μ represents the true rating for all movies and users and ϵ represents independent errors sampled from the same distribution centered at zero. Least squares estimate of μ is the average rating of all movies across all users here.
The expected rating of the train_set is:

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

```
## [1] 3.512509
```

Prediction gives us the following RMSE:

```
naive_rmse <- RMSE(test_set$rating, mu)
tibble(method="naive rmse",
       RMSE=naive_rmse) %>%
  knitr::kable()
```

method	RMSE
naive rmse	1.061135

Movie effect To improve the model, we use the movie effect by adding

$$b_i$$

which is the average ranking for movie i.
The formula now looks like this:

$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

By using:

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

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

we get the b_i of each movie.

Now our prediction shows an improvement compared to the **naive RMSE**:

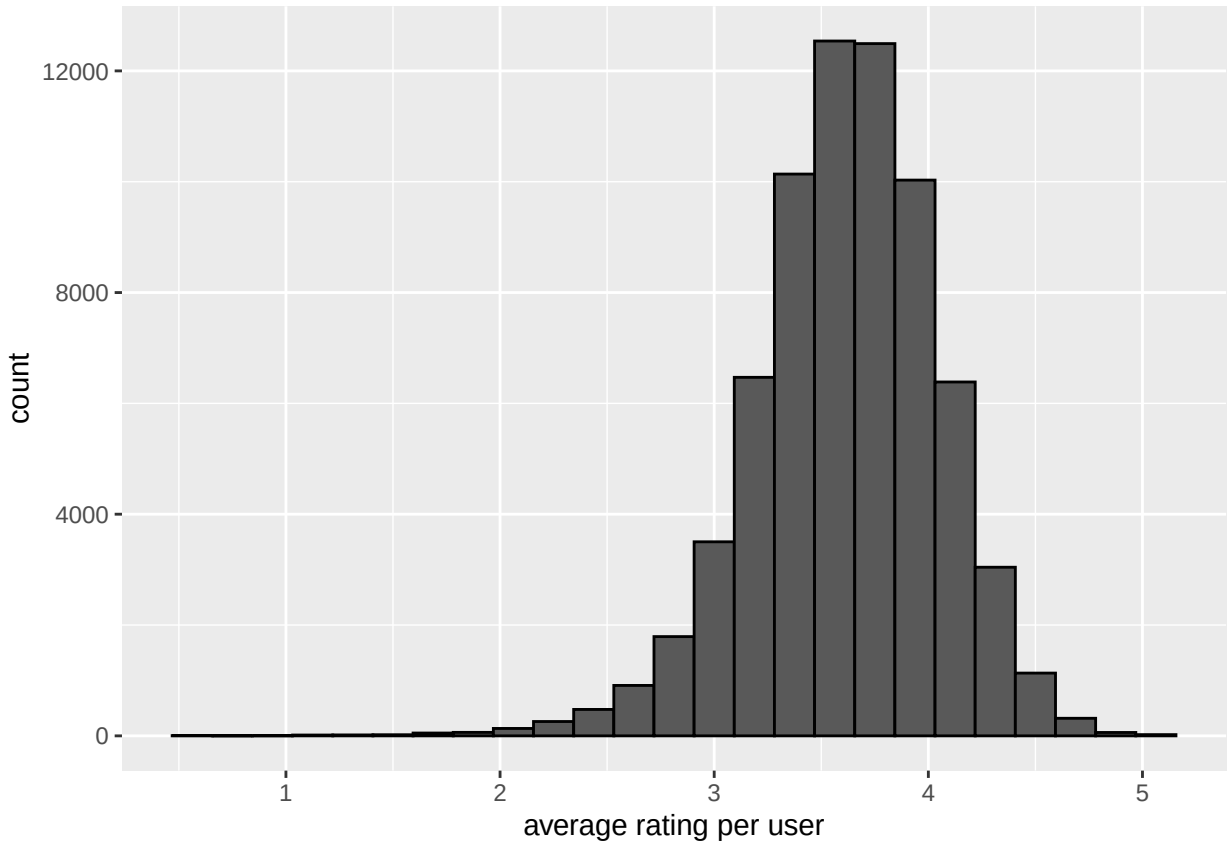
```
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

rmse_movie_effect <- RMSE(test_set$rating, predicted_ratings)

tibble(method=c("naive rmse",
                "movie effect"),
       RMSE=c(naive_rmse,
              rmse_movie_effect)) %>%
  knitr::kable()
```

method	RMSE
naive rmse	1.0611352
movie effect	0.9441568

User-specific effect Taking a view on the average rating from each user that we reviewed before:

```
edx %>%
  group_by(userId) %>%
  summarize(mean_user_ratings=mean(rating)) %>%
  ggplot(aes(mean_user_ratings)) +
  geom_histogram(color="black", bins=25) +
  xlab("average rating per user")
```

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



We can see that the variability across users is considerable.
Because of what we have seen in the last plot, we also add b_u as the user-specific effect:

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

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

The prediction shows that the **RMSE** got an improvement again:

```
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)
rmse_user_effect <- RMSE(test_set$rating, predicted_ratings)
tibble(method=c("naive rmse",
                "movie effect",
                "user effect"),
       RMSE=c(naive_rmse,
              rmse_movie_effect,
```

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

method	RMSE
naive rmse	1.0611352
movie effect	0.9441568
user effect	0.8659736

Genres effect We add the genres effect to get the average ranking of each combination of genres. This is the difference of ratings of each genre to the mean rating:

```
genres_avgs <- train_set %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(user_avgs, by="userId") %>%
  group_by(genres) %>%
  summarize(b_g=mean(rating-mu-b_i-b_u))
```

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

Once more our prediction shows an improvement compared to the approaches before:

```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(user_avgs, by="userId") %>%
  left_join(genres_avgs, by="genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)
rmse_genres_effect <- RMSE(test_set$rating, predicted_ratings)
tibble(method=c("naive rmse",
                "movie effect",
                "user effect",
                "genres effect"),
       RMSE=c(naive_rmse,
              rmse_movie_effect,
              rmse_user_effect,
              rmse_genres_effect)) %>%
  knitr::kable()
```

method	RMSE
naive rmse	1.0611352
movie effect	0.9441568
user effect	0.8659736
genres effect	0.8656019

Regularization Regularization is a method we use to get the **RMSE** improved by penalizing large estimates that are formed using small samples. This means for example that we will penalize movies that have

only a few user-ratings. To prevent overtraining of small sized samples we add a penalty to the equation, which gets bigger if many b's of its predictor are large. If the samplesize is large enough, the penalty effect is irrelevant.

We are minimizing:

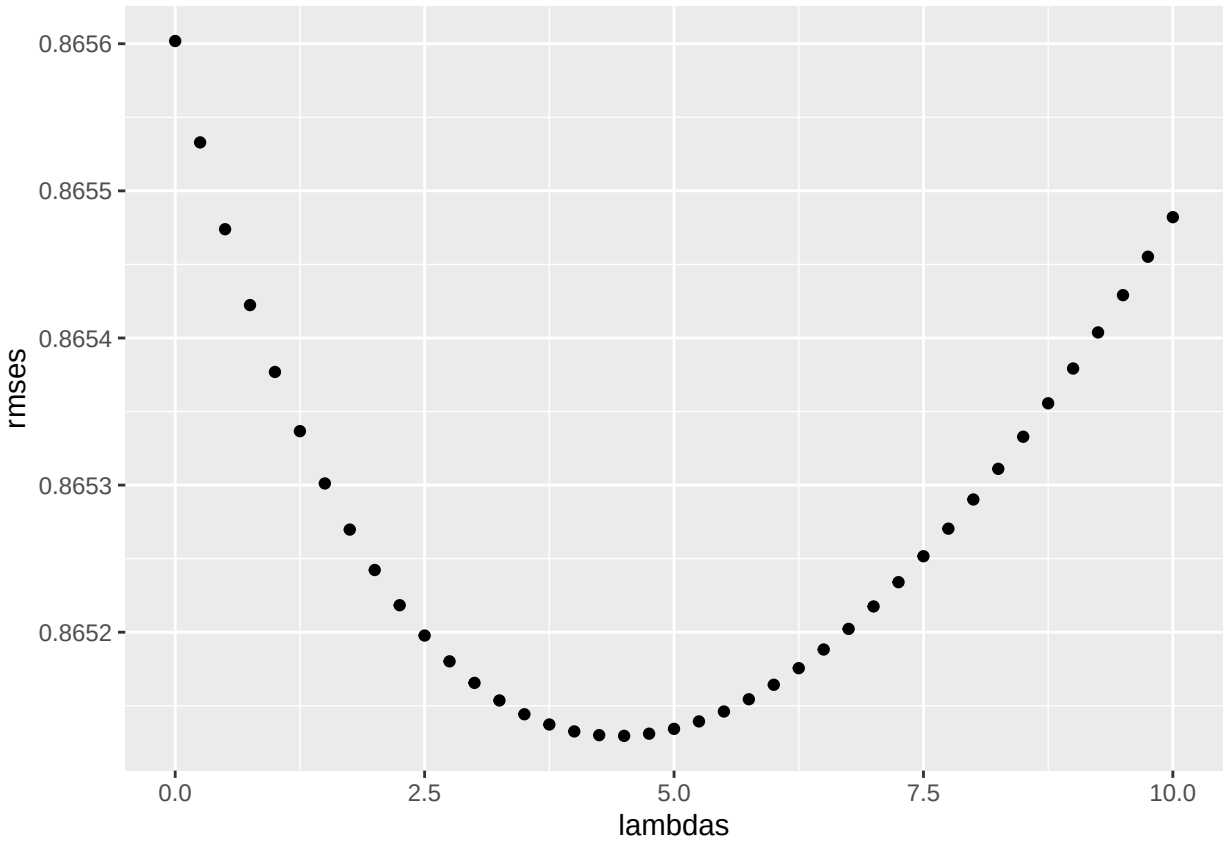
$$\sum_{u,i,g}(y_{u,i,g} - \mu - b_i - b_u - b_g)^2 + \lambda(\sum_i b_i^2 + \sum_u b_u^2 + \sum_g b_g^2)$$

To define the tuning parameter of the penalty λ we will use cross-validation to get the optimal tuning parameter (this takes a few moments):

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

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))

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

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

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

Now we get to see the RMSE values of each lambda from the lambdas sequence:

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



3. Result

As we see from the previous plot, the optimal λ for minimizing the **RMSE** is:

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

```
## [1] 4.5
```

Finally we run the rmses-function again, but only with the optimal lambda which brings the minimum **RMSE**. For this last approach we will use the validation set instead of the test test for testing.
The modeling of regularization shows an improvement on the approaches we made before:

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

#movie effect with regularization on edx set
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))
```

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

```
#user effect with regularization on edx set
b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
```

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

```
#genres effect with regularization on edx set
b_g <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+lambda))
```

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

```
#predictions on validation set
predicted_ratings <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)
```

```
regularization_edx_validation <- RMSE(predicted_ratings, validation$rating)

tibble(method=c("naive rmse",
                "movie effect",
                "user effect",
                "genres effect",
                "regularized movie, user and genres effect (train and test)",
                "regularized movie, user and genres effect (edx and validation)"),
       RMSE=c(naive_rmse,
              rmse_movie_effect,
              rmse_user_effect,
              rmse_genres_effect,
              regularization_train_test,
              regularization_edx_validation)) %>% knitr::kable()
```

method	RMSE
naive rmse	1.0611352
movie effect	0.9441568
user effect	0.8659736
genres effect	0.8656019
regularized movie, user and genres effect (train and test)	0.8651296
regularized movie, user and genres effect (edx and validation)	0.8644543

4. Conclusion

With the final regularization estimate we reached the goal of an **RMSE** < 0.86490.
As we could see, the more predictors we use for predicting the RMSE, the more we get to improve it.
To prevent overfitting, we used the regularization estimate by penalizing least squares, so that movies, users and genres with only a few ratings did not affect the **RMSE** mistakenly.