HarvardX: PH125.9x Data Science: MovieLens Project

Olga Loginova

16/06/2020

Introduction

Overview

The project is part of the HervardX: PH125.9x Data Science: Capstone course. The entire course consists of two parts: MovieLens Project and CYO Project, respectively. The current document is dedicated to MovieLens Project.

The challenge is to create a recommendation system using ratings that users have given to movies according to Netflix data. The Netflix data are not publicly available, but for the task students are offered to create training and test sets with a predifined code extracting records from the [10M version of MovieLens dataset, collected by GroupLens Research] (http://files.grouplens.org/datasets/movielens/ml-10m.zip). Movie ratings in the test set will be predicted as if they were unknown.

Dataset

Let's create training and test sets using the code given for the project in the course:

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

```
## Loading required package: tidyverse

## -- Attaching packages --------------------------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.0     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   0.8.5
## v tidyr   1.0.3     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 from 'package:purrr':
##
##     lift
```

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

```r
# 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")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[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")
```

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

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

# Making sure userId and movieId in validation set are also in edx set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Adding rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
```

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

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

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

In the models *edx* will be used as the training set and *validation* (which comprises 10% of the MovieLens data) as the test set. The validation set will be used only for the final evaluation of each model.

Aim

The Root Mean Square Error (RMSE) will be used to assess how close the predictions are to the true values of the given test set. In gneral, RMSE shows the difference between values predicted by a model and the actual values of a particular dataset.

If we define $y_{u,i}$ as the rating for a movie i by a user u and denote our prediction with $\hat{y}_{u,i}$, then the RMSE will be as follows:

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

with N being the number of user/movie combinations and the sum occurring over all these combinations.

RMSE is the typical error we make when predicting a movie rating. If this number is larger than 1, our typical error is larger than one star, which is not good.

The aim of the project is to get an RMSE lower than 0.86490. Let's write a function that computes the RMSE for vectors of ratings of the validation set and their corresponding predictors:

```
RMSE <- function(predicted_ratings){
  sqrt(mean((validation$rating - predicted_ratings)^2))
}
```

and define the maximum allowed RMSE value as max_rmse

```
max_rmse <- 0.86490
```

The predicting models described below will be compared using their RMSEs against the maximum allowed RMSE.

Data Analysis and Models

Preprocessing

First six rows of the edx dataset give a glimpse at the data and show the column names:

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1                 Comedy|Romance
## 2          Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5        Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7        Children|Comedy|Fantasy
```

The dataset is in tidy format with thousands of rows. Each row represents a rating given by one user to one movie.

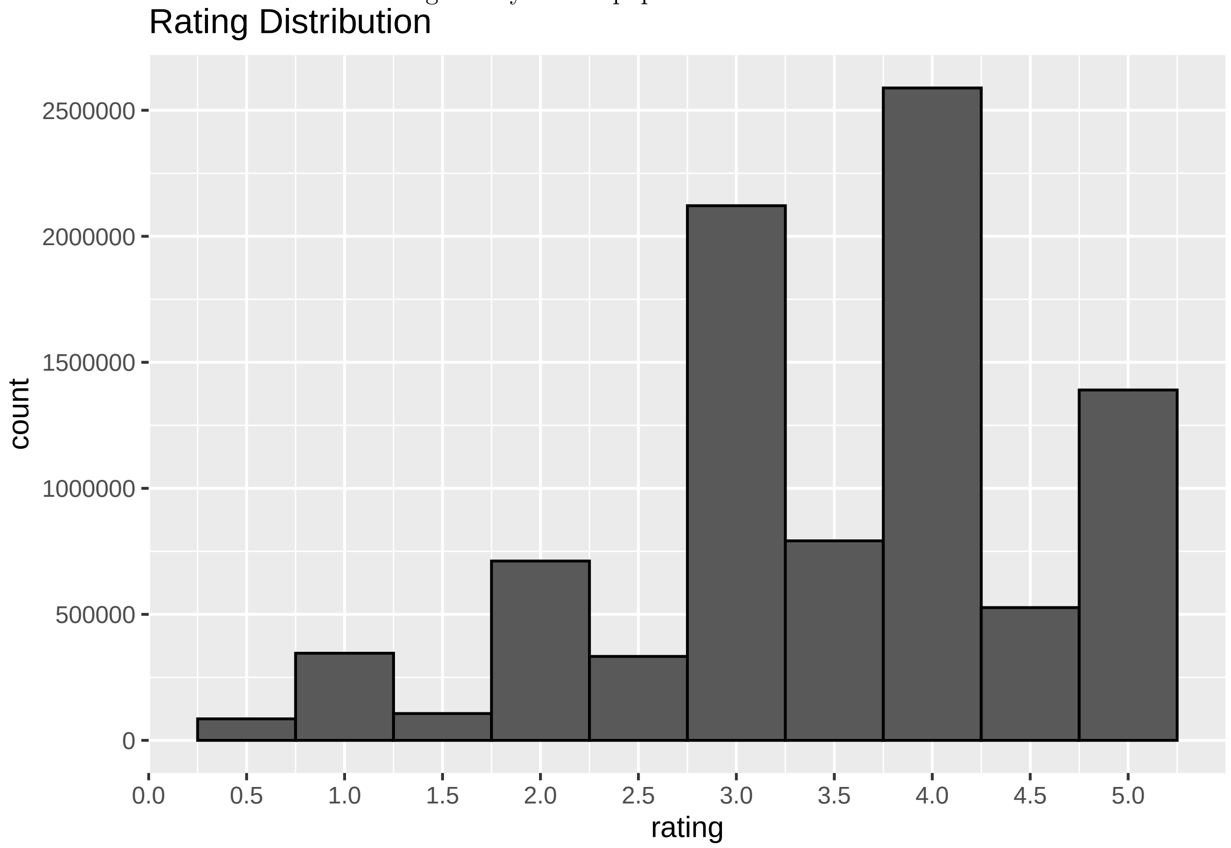
The summarary shows that the ratings are between 0.5 and 5 and there are no zero ratings.

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

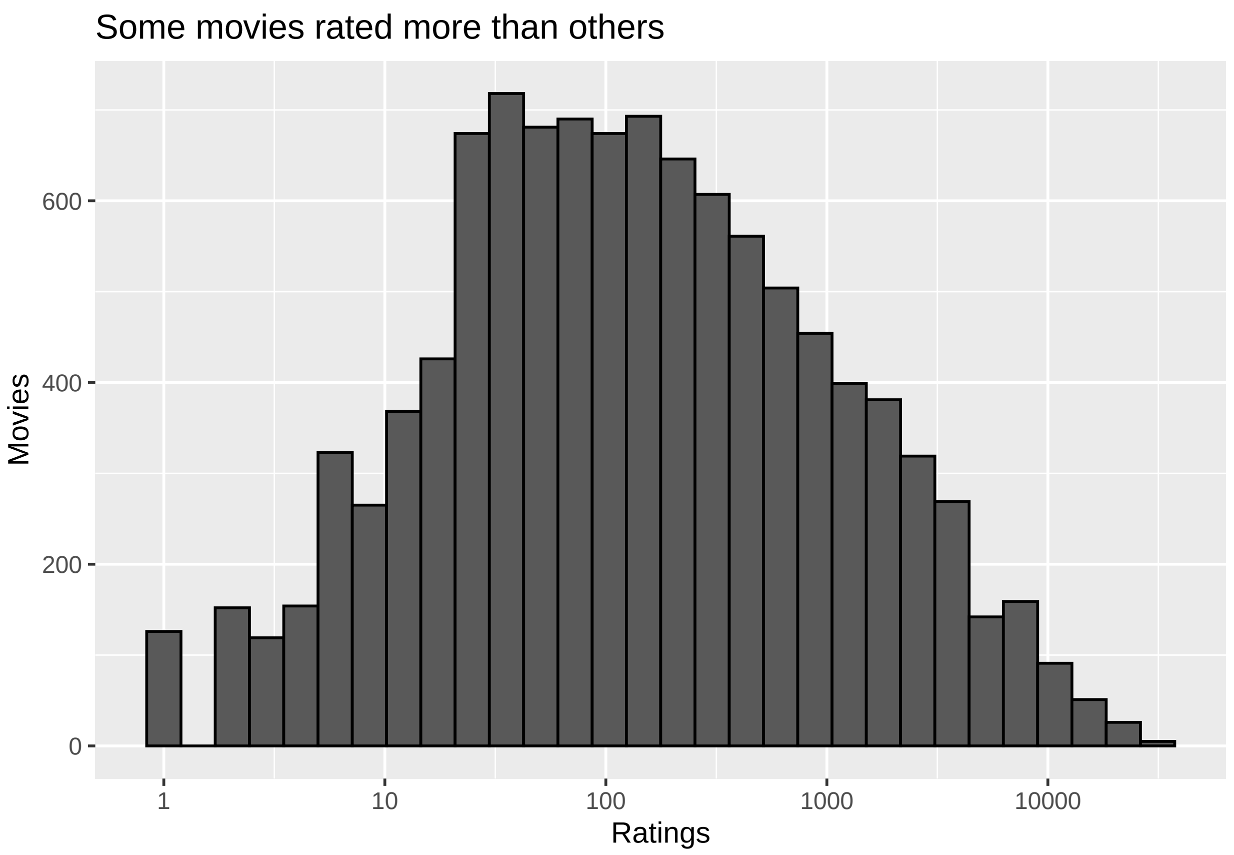
By the number of unique users who rated movies and how many unique movies were rated it is clear that not every user rated every movie (if we multiply numbers, there should be much more than 10 million rows):

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

Let's look at some properties of the data. The general overview of the ratings shows that users tend to rate movies between 3 and 4 and half ratings are by far less popular than the full ones. 0.5 is the least common rating.

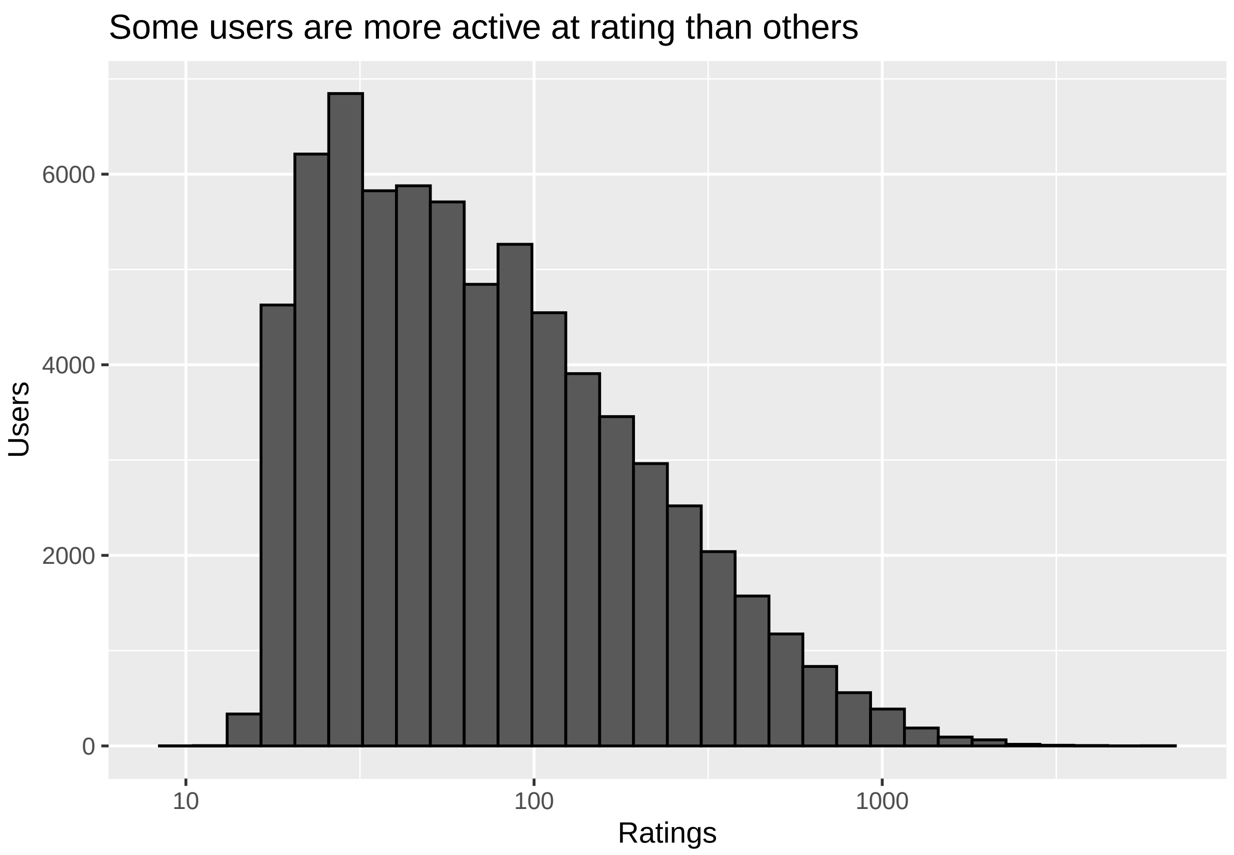


By the distribution below it's clear that some movies get rated more often than others. This is a so-called Movie effect, or movie bias b_i. The Movie Effect Model will take it into account.

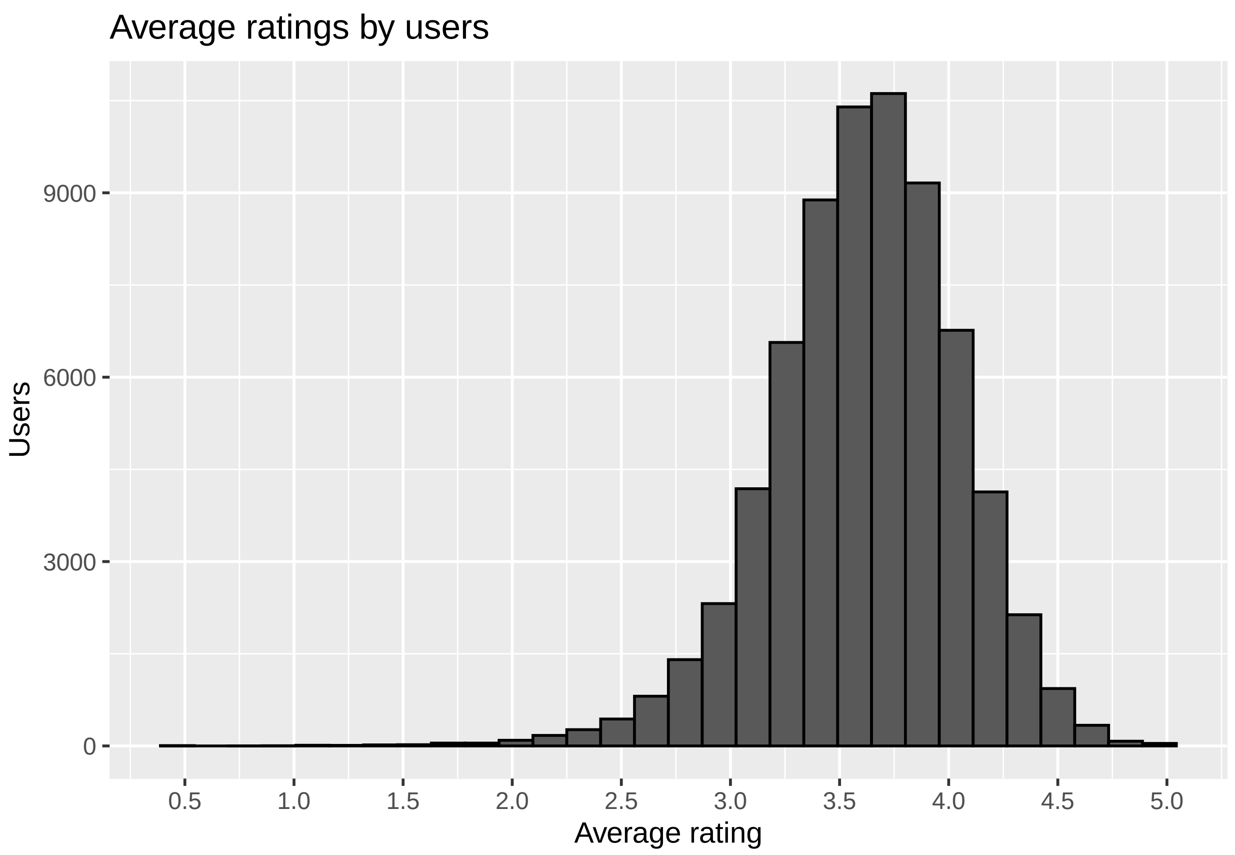


Also, some movies have very few ratings and about 130 movies have only one rating. As long as for RMSEs every error has the effect of a squared error, large errors in such rarely rated movies are able to drastically increase RMSE. This will be counteracted by applying regularisation to the extended Movie Effect Model. Generally, regularisation is aimed for penalising large estimates that are formed using small sample sizes.

A similar bias is seen below in the distribution of users vs ratings: some users are more active at rating than others and the majority of the users rated from about 40 to 150 movies.



Likewise, there is high variability across users. Some users tend to use only low ratings for all movies, whereas others rate more generously. It is a so-called User effect and it will be taken into account as the Movie Effect Model will be extended with the User effect b_u in Movie-User Effect Model.



To complete the task, we will use linear models, but first, let's remove unused columns from the edx dataset.

```
edx <- edx %>% select(-timestamp, -title, -genres)
head(edx)
```

```
##   userId movieId rating
## 1      1     122      5
## 2      1     185      5
## 4      1     292      5
## 5      1     316      5
## 6      1     329      5
## 7      1     355      5
```

Models

Naive Model

Let's start to build a model with the assumption that we have the same rating for all movies and users as if all differences in ratings are explained by some random variation. It will be as follows:

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

with $\varepsilon_{u,i}$ independent error sample from the same distribution centered at 0 and μ the universal rating for all movies. The estimate that minimises the RMSE (through minimising the residual sum of squares, RSS) is the least squares estimate of μ and, in this case, is the average of all ratings:

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

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

```r
naive_rmse <- RMSE(mu)
naive_rmse
```

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

The RMSE is too high and goes above the max_rmse:

```r
naive_rmse < max_rmse
```

```
## [1] FALSE
```

Movie Effect Model

Knowing that some movies are just generally rated higher than others, we can modify the previous model by adding b_i to represent average ranking for a movie i:

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

To estimate b_i we will use the fact that the least squares estimate $\hat{b}_i$ is just the average of $Y_{u,i} - \mu$ for each movie i.

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

The RMSE of the prediction with $\hat{y}_{u,i} = \mu + \hat{b}_i$ is still higher than the maximum allowed RMSE.

```r
predicted_ratings <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
movie_rmse <- RMSE(predicted_ratings)
movie_rmse < max_rmse
```

```
## [1] FALSE
```

Movie-User Effect Model

A further improvement to the previous model may be done by adding b_u, the user effect.

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

Likewise, the estimate $\hat{b}_u$ will be calculated as the average of $Y_{u,i} - \mu - b_i$

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

This enhancement proved to be not enough to complete the challenge:

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

```
## [1] FALSE
```

Regularised Movie-User Model

Let's take a closer look at the Movie Effect linear model

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

with b_i to be the average ranking for movie i. The least squares equation of n observations will be

$$RSS = \sum_{i=1}^{n} \{y_i - (\mu + b_i + \varepsilon_{u,i})\}^2$$

Once again, our goal is to find the least squares estimate $\hat{b}_i$ that minimises the RSS and then RMSE.

We know that some movies are rated more than others. Suppose we have k movies with 100 ratings and m movies with just one rating. In our case, the least squares estimate $\hat{b}_i$ for the k movies is the average of the 100 user ratings, $1/100 \sum_{i=1}^{100}(Y_{i,1} - \mu)$, while the estimate for the m movies is simply the deviation from the average rating μ: $Y_{u,i} - \mu$. Since the latter depends on only one value, it cannot be reliable. The greater m and the smaller k in the training set, the less precise the estimate $\hat{b}_i$ would be.

In order to control the total variability of the Movie effects $\sum_{i=1}^{k+m} b_i^2$, we will impose a penalty with a tuning parameter λ. The penalty should become larger if we have many large b_i.

Moreover, instead of minimising the least squares equation, we will now minimise the equation that adds the penalty:

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

So the values of the new $\hat{b}_i$ estimate will be as follows:

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

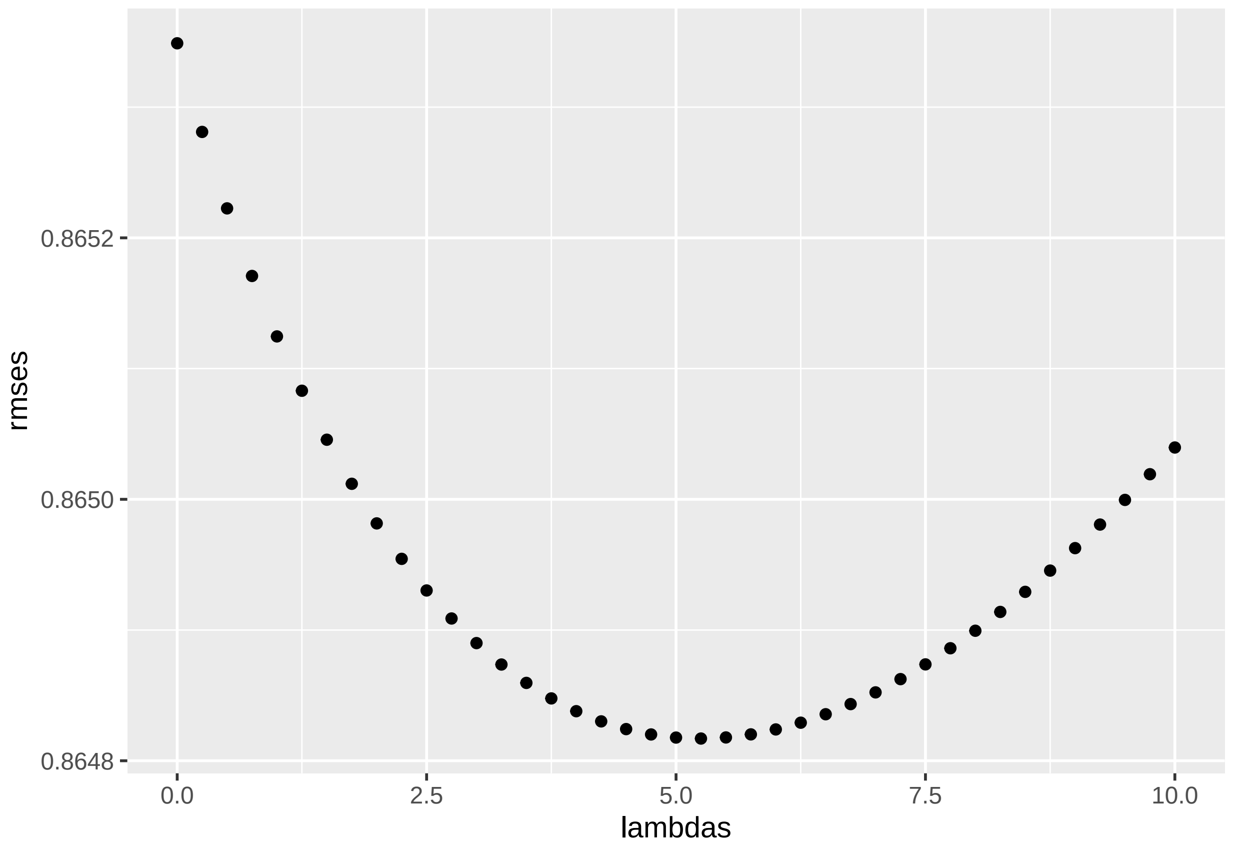
where n_i is the amount of user ratings for a movie i. The greater n_i (the more ratings a movie has), the less important λ is (as $n_i + \lambda$ becomes closer to n_i); and the smaller n_i (the fewer ratings a movie has), the smaller $\hat{b}_i(\lambda)$ becomes and therefore less significant (it shifts towards 0 and gets even closer to 0 with a larger λ).

In order to find the optimal λ (with the minimal RMSE) we will use cross-validation.

```r
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  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))
})
```

The plot rmses vs lambdas will reveal the best lambda.

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



Thus, the lambda with the lowest RMSE is:

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

The RMSE is:

```
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n() + lambda), n_i = n())
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n() + lambda), n_u = n())
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(prediction = mu + b_i + b_u) %>%
  pull(prediction)
reg_movie_user_rmse <- RMSE(predicted_ratings)
reg_movie_user_rmse
```

```
## [1] 0.864817
```

and now it seems to go below the maximum allowed RMSE value max_rmse:

```
reg_movie_user_rmse < max_rmse
```

```
## [1] TRUE
```

Results

The results of each model are summarised in the tibble:

```
##                                   Method     RMSE BelowMaxRMSE
## 1:                      Just the Average 1.061202        FALSE
## 2:                    Movie Effect Model 0.943909        FALSE
## 3:            Movie + User Effects Model 0.865349        FALSE
## 4: Regularised Movie + User Effect Model 0.864817         TRUE
```

We started with a primitive model that gave us the RMSE equal to the typical error larger than one star. Taking into account both movie and user bias, we managed to improve the RMSE. Finally, the regularised model that made a very small improvement turned out to be enough to complete the task and get down below the maximum allowed by the task RMSE value.

Conclusion

To complete the task a relatively simple regularised linear model with movie and user effects proved to be sufficient.

There could be further enhancements of the model like adding the genre effect (i.e. using the split genres column). Perhaps, we could discover and use the fact that movies of a particular genre are more popular within a particular period of time (i.e. the converted into the date format timestamp column with the year used as.numeric can be plotted against split genres). Also, it's worth exploring whether the date of release (last 6 characters without brackets of the data in the title column) affect the RMSE. Other machine learning algorithms may also come in handy here.

However, with the further complications using any other machine learning approach one should be aware of the hardware computing power (using RAM), as the database of 1M records may significantly slower down the training process.