# MovieLens Capstone Project 1

*Lukasz Kolodziejek*

*25/09/2019*

## Introduction

### Overview

This is MovieLens Capstone project 1 for online course HarvardX: PH125.9x

Goal of this project is to document, analyse and optimize Root Mean Square Error (RMSE) of movie recommendation algorithm.

In order to achieve this different models will be created to predict ratings given available features.

This project is based on publicly available 10M version of the MovieLens dataset, which can be accessed here: https://grouplens.org/datasets/movielens/10m/

### Data ingestion

Data ingestion part of the code comes from the course HarvardX: PH125.9x and is part of project definition

```
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")

# 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")
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)
```

# Analysis

Training dataset has following columns: userId, movieId, rating, timestamp, title, genres and 9000055 rows. Each user can rate single movie only once, hence pair 'userId, movieId' can be treated as unique identifier. Genres do not come in tidy format, as movie can have multiple genres assign and separated with '|' sign.

## Quiz

### Q1

How many rows and columns are there in the edx dataset?

```
dim(edx)
```

```
## [1] 9000055       6
```

### Q2

Number of zeroes given as ratings in datase

```
nrow(filter(edx, rating==0))
```

```
## [1] 0
```

Number of threes given as ratings in datase

```
nrow(filter(edx, rating==3))
```

```
## [1] 2121240
```

### Q3

How many different movies are there in edx dataset?

```
edx %>% summarize(n_movies = n_distinct(movieId))
```

```
##   n_movies
## 1    10677
```

### Q4

How many different users are there in edx dataset?

```
edx %>% summarize(n_movies = n_distinct(userId))
```

```
##   n_movies
## 1    69878
```

### Q5

How many movie ratings are in each of the following genres in the edx dataset?

```
edx %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
```

```
## # A tibble: 20 x 2
##    genres               count
##    <chr>                <int>
##  1 Drama              3910127
##  2 Comedy             3540930
##  3 Action             2560545
##  4 Thriller           2325899
##  5 Adventure          1908892
##  6 Romance            1712100
##  7 Sci-Fi             1341183
##  8 Crime              1327715
##  9 Fantasy             925637
## 10 Children            737994
## 11 Horror              691485
## 12 Mystery             568332
## 13 War                 511147
## 14 Animation           467168
## 15 Musical             433080
## 16 Western             189394
## 17 Film-Noir           118541
## 18 Documentary          93066
## 19 IMAX                 8181
## 20 (no genres listed)      7
```

### Q6

Which movie has the greatest number of ratings?

```r
edx %>%
  group_by(title) %>%
  summarize(count = n()) %>%
  arrange(desc(count)) %>%
  top_n(1)
```

```
## Selecting by count
```

```
## # A tibble: 1 x 2
##   title               count
##   <chr>               <int>
## 1 Pulp Fiction (1994) 31362
```

### Q7 & Q8

What are the five most given ratings in order from most to least?

```r
edx %>%
  group_by(rating) %>%
  summarize(count = n()) %>%
  arrange(desc(count)) %>%
  top_n(5)
```

```
## Selecting by count
```

```
## # A tibble: 5 x 2
##   rating   count
##    <dbl>   <int>
## 1    4   2588430
## 2    3   2121240
## 3    5   1390114
## 4    3.5  791624
## 5    2    711422
```

## Model Optimization

Let's start with definition of RMSE function as it will be used to assess model's performance

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

### Model 1 - Just the average

First model will be very simple: we will predict all ratings as an average of them

```r
mu <- mean(edx$rating)
model_1_rmse <- RMSE(validation$rating, mu)
```

First model provides following RMSE:

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

### Model 2 - movie effect

In this model we will be predicting average rating and adjusting for movie effect (average rating for a specific movie).

This model assumes that different movies have different average ratings. It calculates b_i, which is the difference from the mean movie rating:

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

predicted_ratings <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i

model_2_rmse <- RMSE(validation$rating, predicted_ratings)
```

Here is graph presenting movie effect:

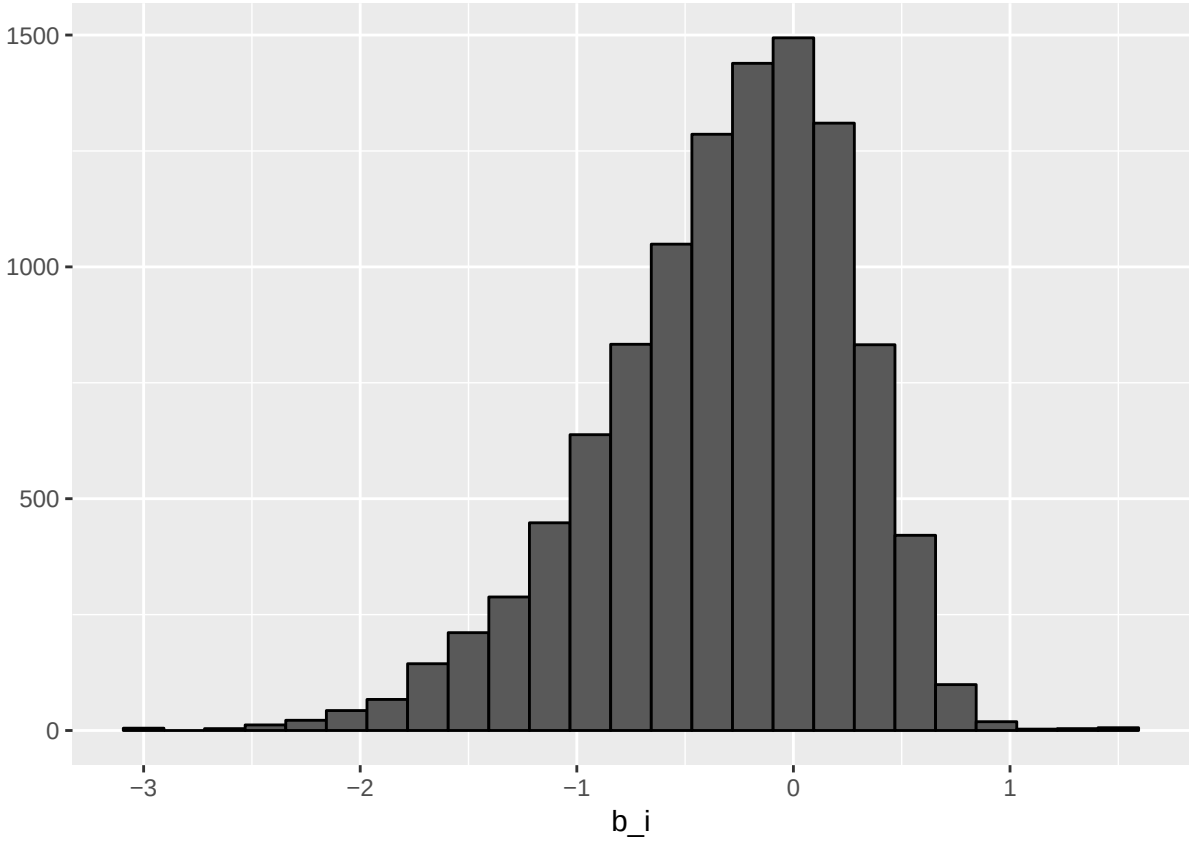


Figure 1: Movie effect histogram

Second model provides following RMSE:

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

### Model 3 - Movie + user effect

This model assumes that different movies have different average ratings and that different users tend to give higher or lower than average ratings. It calculates b_i, which is the difference from the mean movie rating for movies and b_u to reflect user difference from the users' mean rating.

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

predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

model_3_rmse <- RMSE(validation$rating, predicted_ratings)
```

Here is graph presenting user effect:

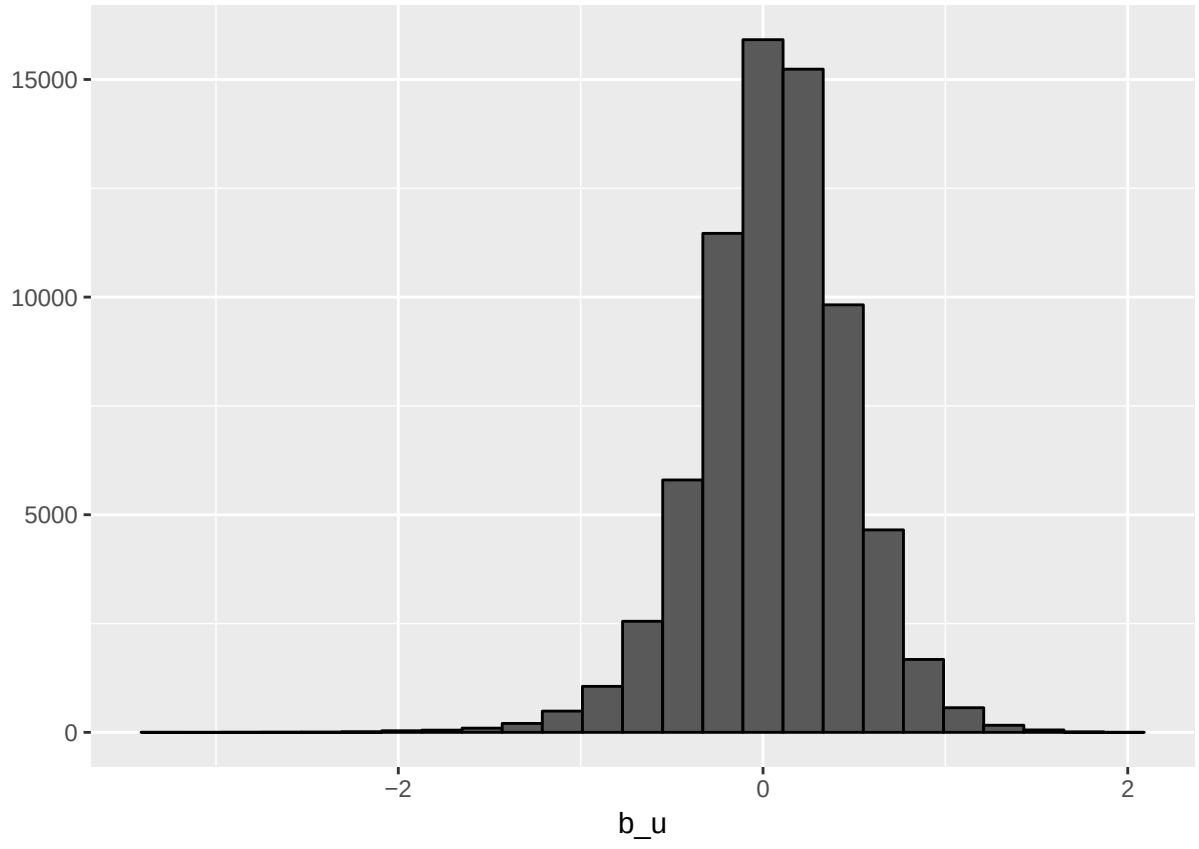


Figure 2: User effect histogram

Third model provides following RMSE:

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

### Model 4 - Regularization + movie + user effect

Additionally to the previous model, now we will be trying to reduce variability of the effect sizes by penalizing large estimates that come from small sample sizes (regularization).

First we need to select optimal lambda parameter to optimize regularization. As validation dataset cannot be used to optimize model parameters, we will divide train set into two subsets 'trainsubset' and 'testsubset'. Trainsubset will be used to train model, while testsubset to calculate MRSE for given Lambda. Once optimal lambda will be selected, final MRSE will be calculated on validation set. This will help us not to overtrain the model.

```
# Initial split

testsubset_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
trainsubset <- edx[-testsubset_index,]
temp <- edx[testsubset_index,]

# Make sure userId and movieId in testsubset set are also in trainsubset

testsubset <- temp %>%
  semi_join(trainsubset, by = "movieId") %>%
  semi_join(trainsubset, by = "userId")

# Add rows removed from testsubset back into trainsubet

removed <- anti_join(temp, testsubset)
trainsubset <- rbind(trainsubset, removed)
```

Now we run model on train and test subset to select optimal Lambda:

```
lambdas <- seq(0, 20, 1)


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

  mu <- mean(trainsubset$rating)

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

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

  predicted_ratings <- testsubset %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred
```

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

Here is plot presenting achieved RMSE depending on used Lambda parameter:

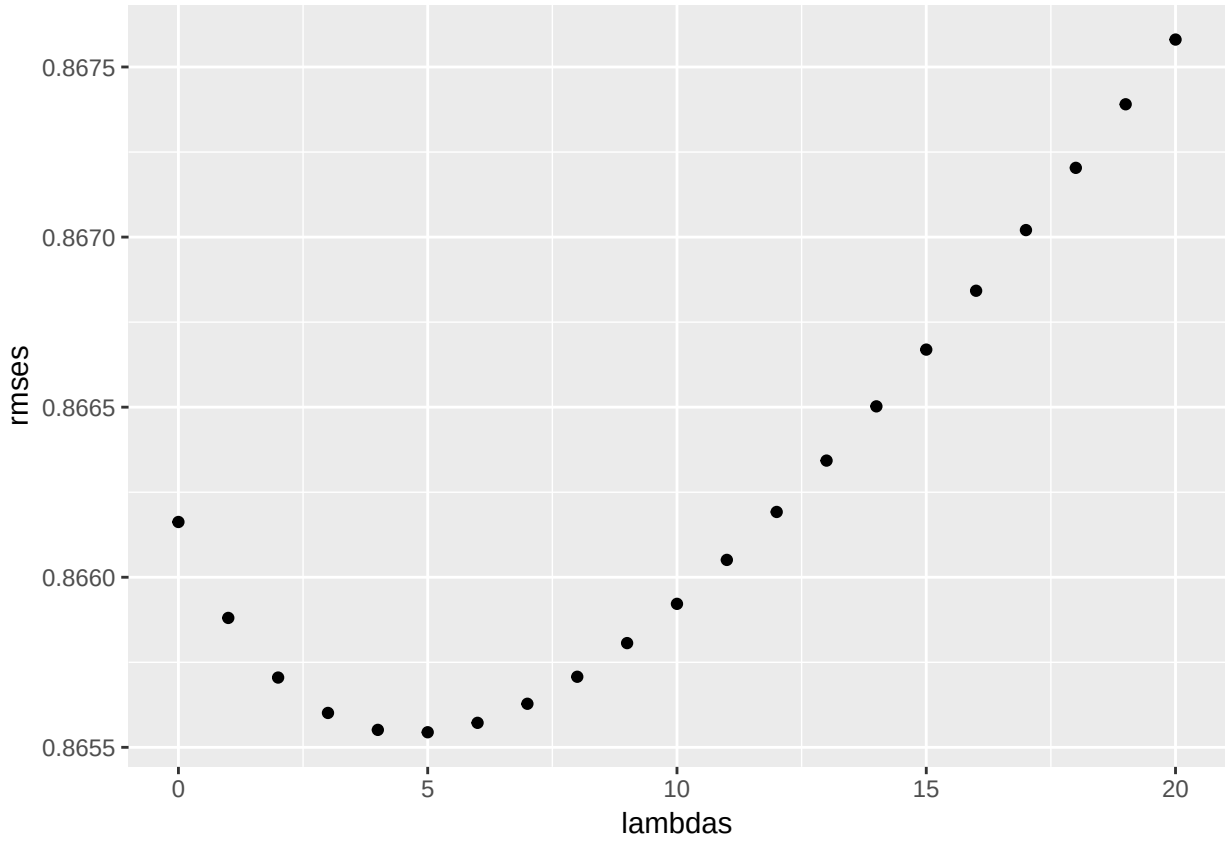


Figure 3: Model 4: RMSE ~ Lambda

We now pick Lambda which minimizes RMSE of testsubset:

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

MRSE calculated on test subset for optimal Lambda equals to:

```
min(rmses)
```

```
## [1] 0.8655444
```

Now we need to calculate final RMSE by training model with fixed Lambda on full training set and calculate results for validation set

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

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

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

predicted_ratings <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
```

Hence fourth model provides following RMSE:

```
model_4_rmse <- RMSE(validation$rating,predicted_ratings)
model_4_rmse
```

```
## [1] 0.8648177
```

### Model 5 - Regularization + movie + user + genre effect

Additionally to the previous model, we will now account for next available feature: genres.

Because genres come not in tidy format, we need first to separate rows.

```
edx_separated <- edx %>% separate_rows(genres, sep = "\\|")
validation_separated <- validation %>% separate_rows(genres, sep = "\\|")
trainsubset_separated <- trainsubset %>% separate_rows(genres, sep = "\\|")
testsubset_separated <- testsubset %>% separate_rows(genres, sep = "\\|")
```

As in previous regularization model to select Lambda we will be working on trainsubset and testsubset, here on their separated versions.

With this in mind let's select optimal lambda parameter to optimize regularization:

```
lambdas <- seq(0, 20, 1)

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

  mu <- mean(trainsubset_separated$rating)

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

  b_u <- trainsubset_separated %>%
    left_join(b_i, by="movieId") %>%
```

```
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  b_g <- trainsubset_separated %>%
    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 <- testsubset_separated %>%
    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) %>%
    .$pred

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

Here is plot presenting achieved RMSE depending on used Lambda parameter:

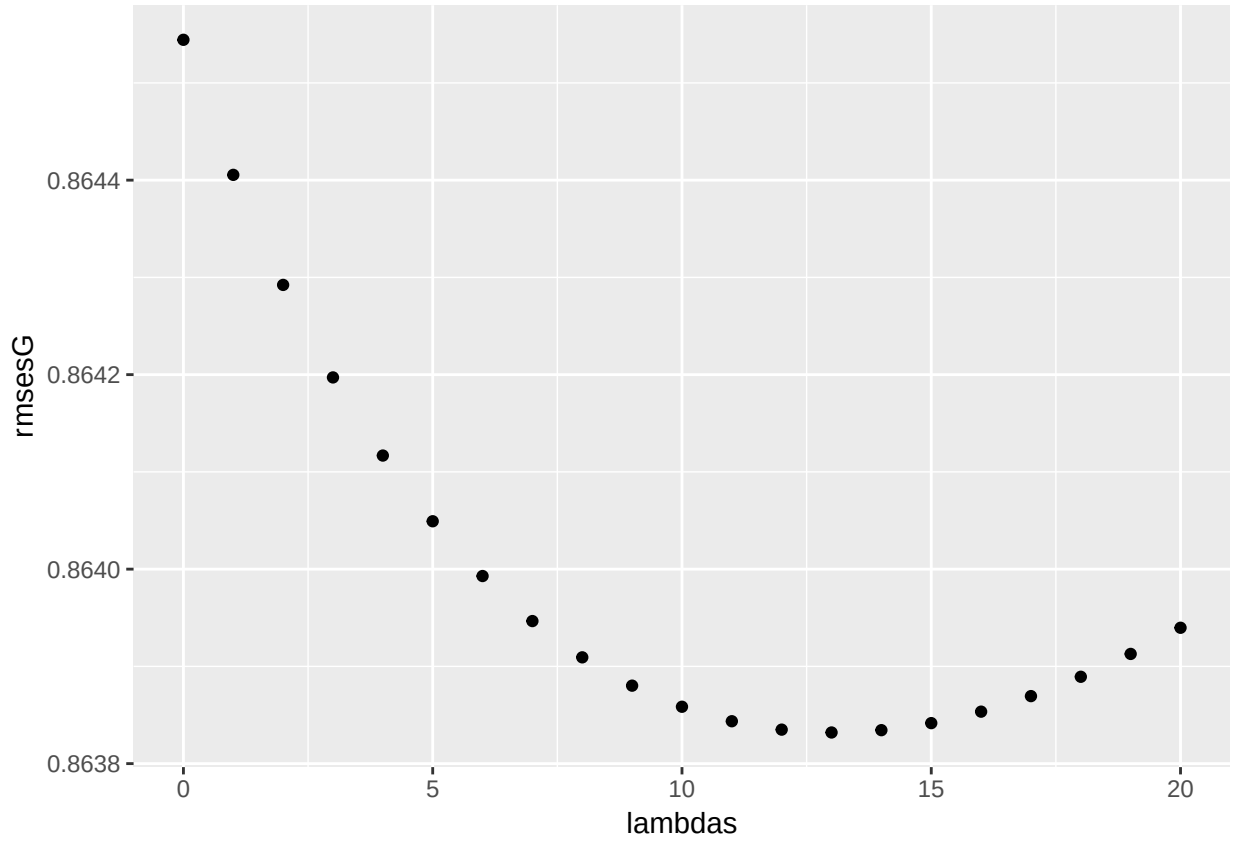


Figure 4: Model 5: RMSE ~ lambda

We now pick Lambda which minimizes RMSE for test subset:

```
## [1] 13
```

And find associated with it minimal RMSE achieved on test subset using following formula:

```
min(rmsesG)
```

```
## [1] 0.863832
```

Now we need to calculate final RMSE by training model with fixed Lambda on full training set and calculate results for validation set:

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

b_i <- edx_separated %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambdaG))

b_u <- edx_separated %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambdaG))

b_g <- edx_separated%>%
  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()+lambdaG))

predicted_ratings <- validation_separated %>%
  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) %>%
  .$pred
```

Fifth model provides following RMSE:

```
model_5_rmse <- RMSE(validation_separated$rating,predicted_ratings)
model_5_rmse
```

```
## [1] 0.8626725
```

## Results

Here is summary of the results achieved for the 5 analysed models (with fixed Lambda parameters for Model 4 and Model 5):

| method | RMSE |
|---|---|
| Model 1 - Just the average | 1.0612018 |
| Model 2 - Movie effect | 0.9439087 |
| Model 3 - Movie + User effects | 0.8653488 |
| Model 4 - Regularized Movie + User effects | 0.8648177 |
| Model 5 - Regularized Movie + User + Genres effects | 0.8626725 |

As we can see best RMSE was achieved in Model 5 with regularization parameter Lambda set to 13.

Let's double confirm our result. For this we will recalculate final model with fixed Lambda parameter:

Mu

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

Optimal lambda optimizing MRSE has been found to be 13

```
final_lambda <- 13
```

As genres comes not in tidy format, we need first to separate rows for training and validation datasets

```
edx_final <- edx %>% separate_rows(genres, sep = "\\|")
validation_final <- validation  %>% separate_rows(genres, sep = "\\|")
```

Now, when all parameters are known let's define model

```
b_i <- edx_final %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - final_mu)/(n()+final_lambda))

b_u <- edx_final %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - final_mu)/(n()+final_lambda))

b_g <- edx_final %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - b_u - final_mu)/(n()+final_lambda))
```

Let's use model to predict ratings

```
predicted_ratings <- validation_final %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  mutate(pred = final_mu + b_i + b_u + b_g) %>%
  .$pred
```

And finally we calculate RMSE of our predictions on validation dataset:

```
RMSE <- RMSE(validation_final$rating,predicted_ratings)
```

Which confirmes that achieved RMSE is:

```
## [1] 0.8626737
```

# Conclusion

5 models have been created and analysed with aim to predict movie rating with minimal RMSE.

| method | RMSE |
|---|---|
| Model 1 - Just the average | 1.0612018 |
| Model 2 - Movie effect | 0.9439087 |
| Model 3 - Movie + User effects | 0.8653488 |
| Model 4 - Regularized Movie + User effects | 0.8648177 |
| Model 5 - Regularized Movie + User + Genres effects | 0.8626725 |

As best performing in terms of RMSE 5th model has been selected. It accounts for movie, user and genre effects and reduces variability of the effect sizes by penalizing large estimates that come from small sample sizes.

It optimal performance has been observed with regularization parameter Lambda equal to 13.

5th model helped improved accuracy in comparision to the first model by over 18%.

Final model can be improved even further by implementing k-fold validation for finding Lambda and applying Matrix Factorization. First however increases calculation time k-times and latter unfortunately exceeds scope of the course and is not covered in this project.