Data Science 'MovieLens' Capstone Project

András Gelencsér

10 November 2019

Overview

This document describes the result of the capstone project for the Harvard Data Science course based on the MovieLens dataset.

The goal of the project is to develop a movie recommendation algorithm based on data science and machine learning techniques to predict the movie ranking by users based on historical movie rating data. The performnce of the algorithm is messed with the root mean standard error (RMSE) value.

The used dataset conatains movie rating data set collected by GroupLens Research. The full MoviLens dataset include more then 20 millions movie ratings for more than 27,000 movies by 138,000 users. The rated movies cover different time periods. The data sets are avaiable on the https://grouplens.org/datasets/movielens/ url.

Because of the size the full dataset only a subset of the data is used for the project. The subset contains 1,000,000 rating entries for the training purpose (training set) and 100,000 rating entries for evaulation of the algorithm performance (test set).

The following steps were done during the project for developing the algorithm:

1. Download the data from the movielens page
2. Analyse the data structure
3. Define the recommendation model based on the result the data analysis
4. Implement and train the algortihm based on the train set
5. Review the algorithm based on the test set

The final recommendation algorithm achived an RMSE value of 0.8000 on the test set.

Methods & analysis

###Preparation

For the analysis the data we will use the following R packages: * tidyverse * caret * lubridate * stringr

the following R code loads and install the requiered packages if needed:

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

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

```
## -- Attaching packages ---------------------------------------------------------------------------
```

```
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- 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(lubridate)) install.packages("lubridate")
```

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

```
##
## Attaching package: 'lubridate'
```

```
## The following object is masked from 'package:base':
##
##     date
```

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

For the calculation we will use K-fold cross validation. The following R function helps us to execute the K-fold cross validation for a function provided as parameter:

```
#function for cross validation
#parameters:
#     trainset: the train set to use for the cross validation
#     cv_n:     the count of the cross validation
#     FUNC:     the function to call for the actual cross validation train and test set (calculated fro
#     ...:      additional parameter necessary for calling the provided function
#
#return:
#     dataframe with the function result for the cross validations (the data frame has cv_n items)

cross_validation <- function(trainset, cv_n, FUNC,...){

  #get the count of the data rows on the train set
  data_count = nrow(trainset)

  #initialize the data frame for the result
  values_from_cv = data.frame()

  #randomise the trainset.
  #If the train set is ordered (not randomised, like the movielens dataset) the cross validation
  #will not be independent and provide wrong result
  trainset_randomised <- trainset[sample(nrow(trainset)),]

  #create the train- and testset for the cross validation
  #we need cv_n run, therefore we use a loop
  for (i in c(1:cv_n)){
    #evaulate the size of the test set. This will be the 1/cv_n part of the data
    part_count = data_count / cv_n

    #select the data from the parameter train set
    #we get the part_count size elements from the parameter train set
    idx = c(   (trunc((i-1) * part_count) + 1) : trunc(i * part_count) )
```

```
    #tmp holds the new test set
    tmp = trainset_randomised[idx,]
    #train holds the new test set
    train = trainset_randomised[-idx,]

    #we remove the elements from the test set, where either the movie or the user is missing in the tra
    test <- tmp %>%
      semi_join(train, by = "movieId") %>%
      semi_join(train, by = "userId")
    removed <- anti_join(tmp, test, by=c("movieId", "userId"))

    #add the removed elements back to the train set
    train <- rbind(train, removed)

    #call the provided function to the actual train and test set.
    akt_value <- FUNC(train, test,...)

    #add the result to the data frame
    #the column 'cv' contains the idx of the cross validation run
    values_from_cv <- bind_rows(values_from_cv, akt_value %>% mutate(cv = i))
  }

  #return the results of each cross validation
  return(values_from_cv)
}
```

The quality of the developed algorithm will be evaluated be the root mean squered error (RMSE). Therefore we will need a function for calculating the RMSE value for a prediction. The following R function calculates the RMSE value for a prediction:

```
#function for calculating the root mean squared error for a prediction
#parameters:
#     actual_rating:      the real rating values (from the test set)
#     predicted_rating:   the predicted ratings (prediction calculated on the test set with the predict
#return:
#     the root mean squared error of the prediction

RMSE <- function(actual_rating, predicted_rating){
  sqrt(mean((actual_rating - predicted_rating)^2))
}
```

At first we need to prepare the data set for the analysis. The analysis is based on the reduced movielens data set including approximately 1,000,000 rating data.

The following R code downloads the data from the http://grouplens.org site and extract the needed data. To avoid unnecessary data traffic, the data will be downloaded only if not done yet.

```
##########################################################
# Create edx set, validation set, and submission file
##########################################################

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

#flag for marking file download
```

```r
file_downloaded <- FALSE

#download the zip file only if not done yet
if (!dir.exists("ml-10M100K")){
     dl <- tempfile()
     download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
     unzip(dl, "ml-10M100K/ratings.dat")
     unzip(dl, "ml-10M100K/movies.dat")

     #mark the download in the flag
     file_downloaded <- TRUE
}

fl <- file("ml-10M100K/ratings.dat")
ratings <- read.table(text = gsub("::", "\t", readLines(fl)),
                      col.names = c("userId", "movieId", "rating", "timestamp"))
close(fl)

fl <- file("ml-10M100K/movies.dat")
movie_data <- str_split_fixed(readLines(fl), "\\::", 3)
close(fl)

colnames(movie_data) <- c("movieId", "title", "genres")
movie_data <- as.data.frame(movie_data) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

# 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
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = ratings$rating, times = 1, p = 0.1, list = FALSE)
edx <- ratings[-test_index,]
temp <- ratings[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)
```

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

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

#remove temp file if download done
if (file_downloaded) rm(dl)

#remove temporary variables
rm(fl, ratings, test_index, temp, removed, file_downloaded)
```

###Available Data For the analysis we have data about 10,000 movies and 70,000 users. The movies have the following known data:

- Movie ID
- Movie title
- Genres

The ratings have the following known data:

- UserId
- Rating timestamp (from this data we can extract the year and month of the rating)
- Rating value (the rating value is a number between 0.5 and 5)

Unfortunately we don‘t have any information (age, gender, country, spoken languages) about the users For the development the recommendation algorithm we can use only the existing data.

After analyzing the existing data, we can extract some more additional information for our analysis.

####Genres data

The movie data set includes the movie genres. This field contains a text about the genre combination of the movie. It can be a “clear” genre (e.g. ‘Drama’) or a combination of many genres (e.g. ‘Adventure|Crime|Thriller’). We can select all the different genres existing in the movie data set. Additionally we can split the combined genres to the clean genres parts.

The following R code extract the genres information from the movie data set and store it in two new data frame. The data frame ‘genres_data’ contains for all genres combination from the movies data set flags with the clear genres combined in the genres. The data frame ‘clean_genres’ contains all the clean genres that was used in the movie data set.

```
#select all unique genres into a new data frame
all_genres <- movie_data %>% select(genres) %>% unique()

#temporary data frame for the genre flag calculation
genres_data <- data.frame(genres=character(), genre=character(), flag=integer())

#loop for splitting all the existing genre combinations
for (i in 1 : nrow(all_genres)){
  #the list of the genres in the genre combination
  genres = str_split(all_genres$genres[i], "\\|")
  #add all genres separate to the genre combination as a flag
  for (genre in genres[[1]]){
    new_item = data.frame(genres=all_genres$genres[i],genre=genre, flag=1)
    genres_data <- rbind(genres_data, new_item)
  }
}

#remove temporary variables
rm(all_genres, genres, genre, new_item, i)

#select all the 'clean' genres
clean_genres = genres_data %>% select(genre) %>% unique()

##Spread the data to get the genre flags for the genre combinations in the data frame columns
genres_data <- genres_data %>% spread(genre,flag, fill=0)
```

####Movie age

In the movie data set we can find the movie title for each movie. The last part of the title contains the

year of publication. We can write an R function to extract the publication year from the title and add this information to the movie data set:

```
#function to extract the year information from the movie title
#the movie title contains the year at the end of the title in brackets (eg. "Movietitel (1999)")
#parameters:
#     title: the title of the movie containing the year
#
#return:
#     only the year

extract_movie_year <- function(title){
  #cut the year from the end of the title
  as.integer(str_sub(title, str_length(title) - 4, str_length(title)-1))
}

#add movie year as column
movie_data <- movie_data %>% mutate(movie_year = extract_movie_year(title))
```

The rating data set contains the datetime of the rating. If we know the year of rating and the movie publication year, we can calculate the movie age at the rating time and add this information to the rating data. The following R code calculates the movie age at rating time for the train and validation set:

```
#extend data frame with age of the movie at the rating time
edx <- edx %>% left_join(movie_data, by="movieId") %>% mutate(movie_age_by_rating =  year(as_datetime(t
validation <- validation %>% left_join(movie_data, by="movieId") %>% mutate(movie_age_by_rating = year(

#select necessary columns only
edx <- edx %>% select(movieId, genres, userId, rating, movie_age_by_rating)
validation <- validation %>% select(movieId, genres, userId, rating, movie_age_by_rating)
```

####Movies

At first we can calculate the average rating of the movies in the data set.

```
avg_overall = mean(edx$rating)
```

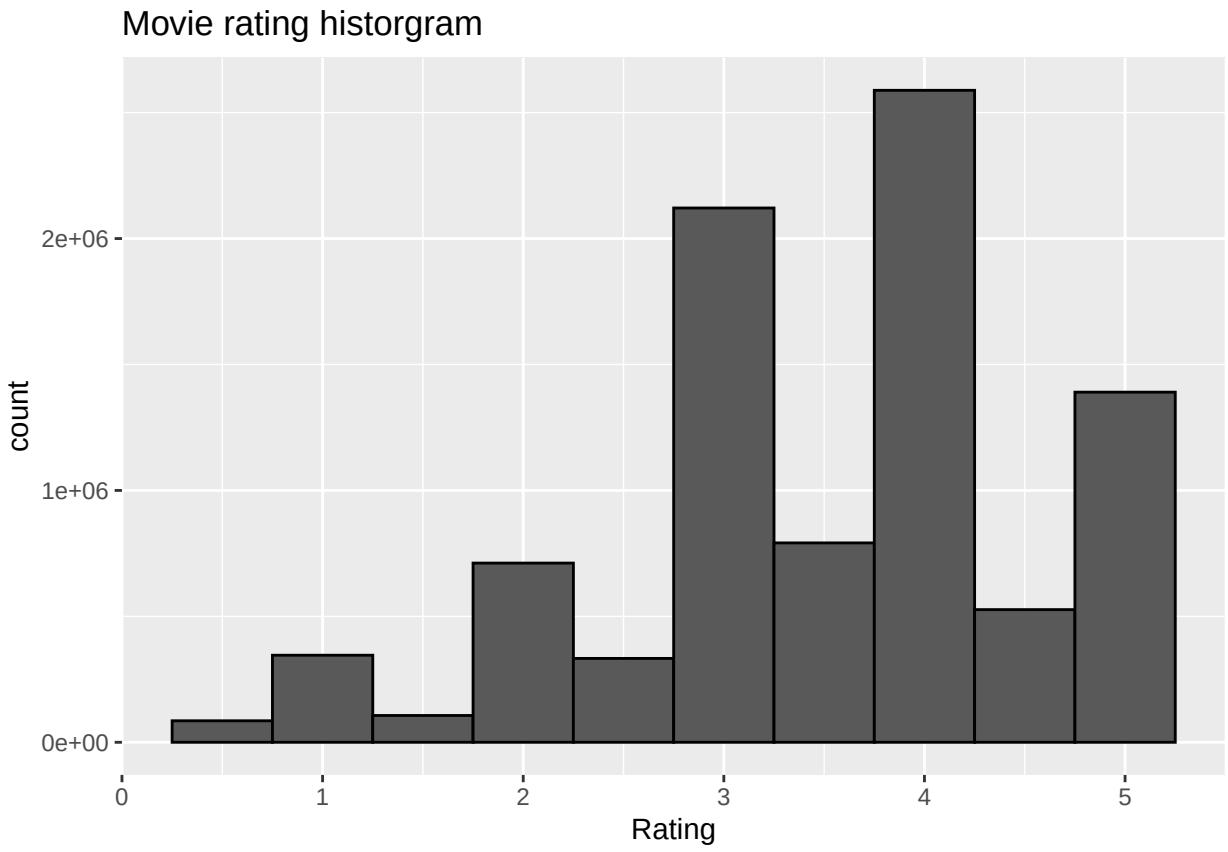
```
#calculate rating count avg rating and sd pro movie
rating_summary_pro_movie <- edx %>% group_by(movieId) %>% summarise(avg = mean(rating), avg_norm=mean(ra
max(rating_summary_pro_movie$n)
```

```
## [1] 31362
```

```
min(rating_summary_pro_movie$n)
```

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

```
#plot the histogram for the rating
edx %>% ggplot(aes(rating)) + geom_histogram(color="black", binwidth = 0.5)+
  ggtitle("Movie rating historgram") +
  xlab("Rating") + ylab("count")
```



####Users

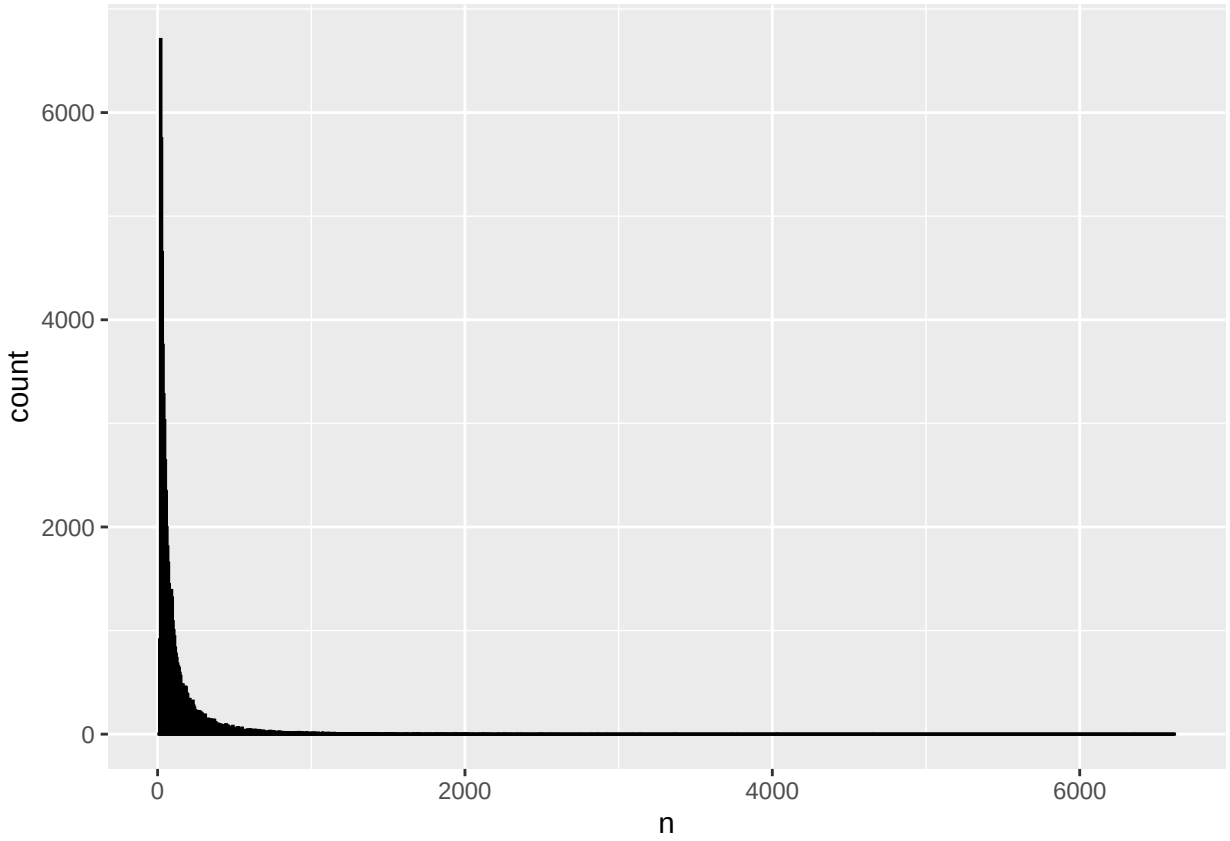
```
rating_summary_pro_user <- edx %>% group_by(userId) %>% summarise(n = n(), avgRating=mean(rating), sdRa
max(rating_summary_pro_user$n)
```

```
## [1] 6616
```

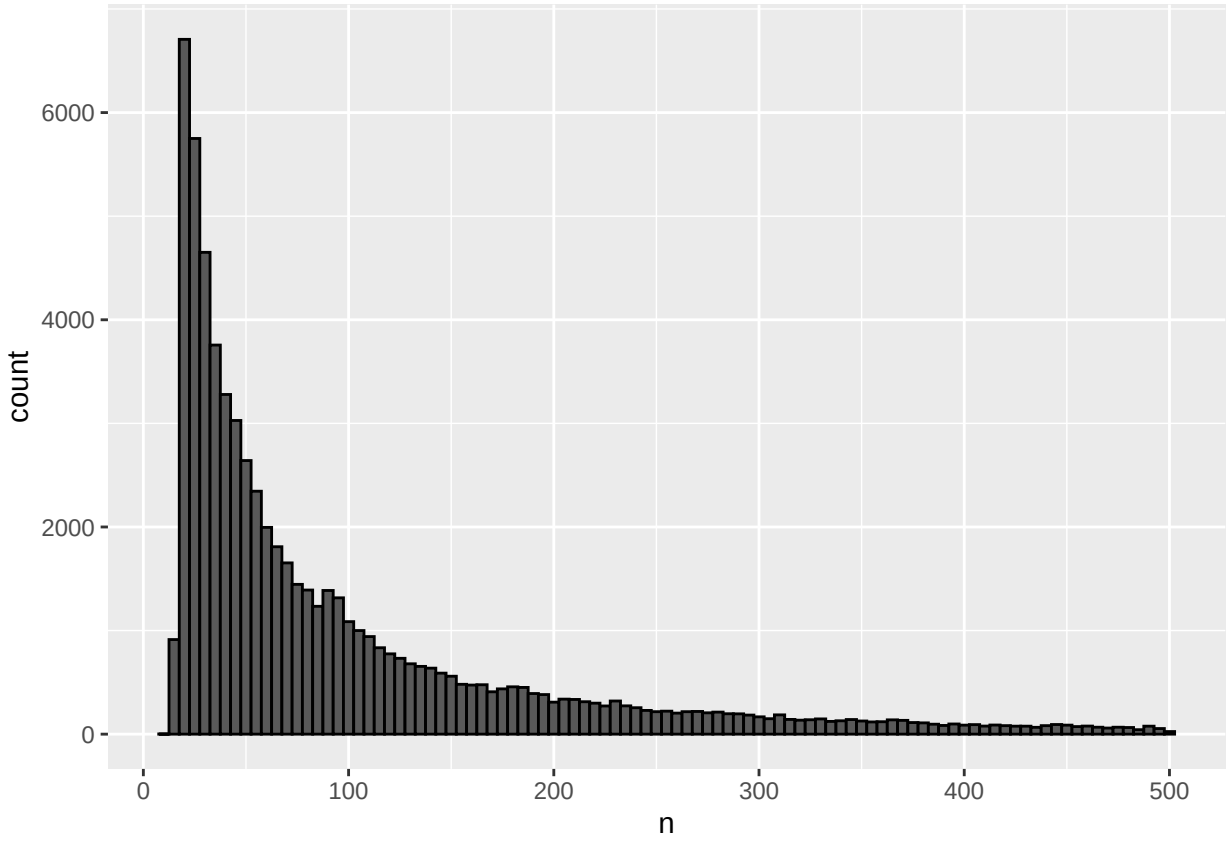
```
min(rating_summary_pro_user$n)
```

```
## [1] 10
```

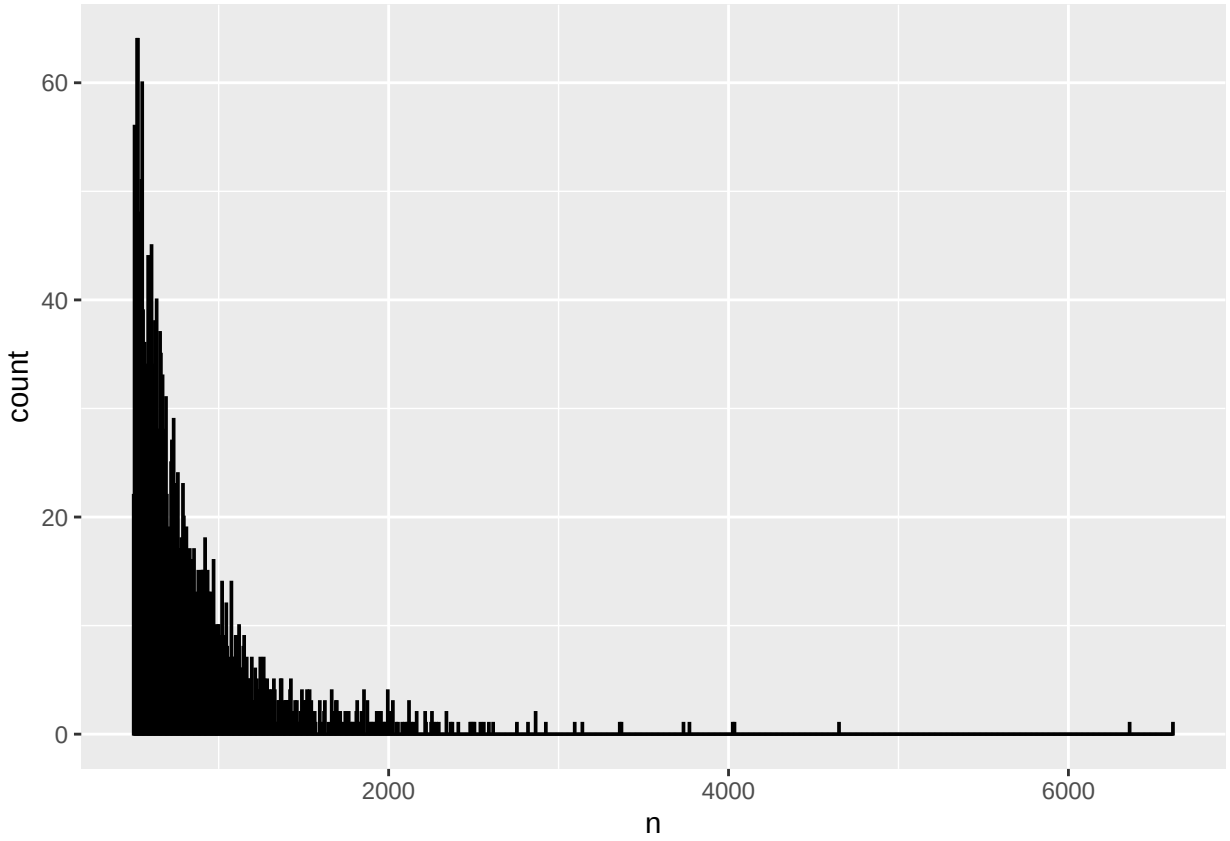
```
rating_summary_pro_user %>% ggplot(aes(n)) + geom_histogram(color="black", binwidth = 5)
```




```
rating_summary_pro_user %>% filter(n < 500) %>% ggplot(aes(n)) + geom_histogram(color="black", binwidth
```

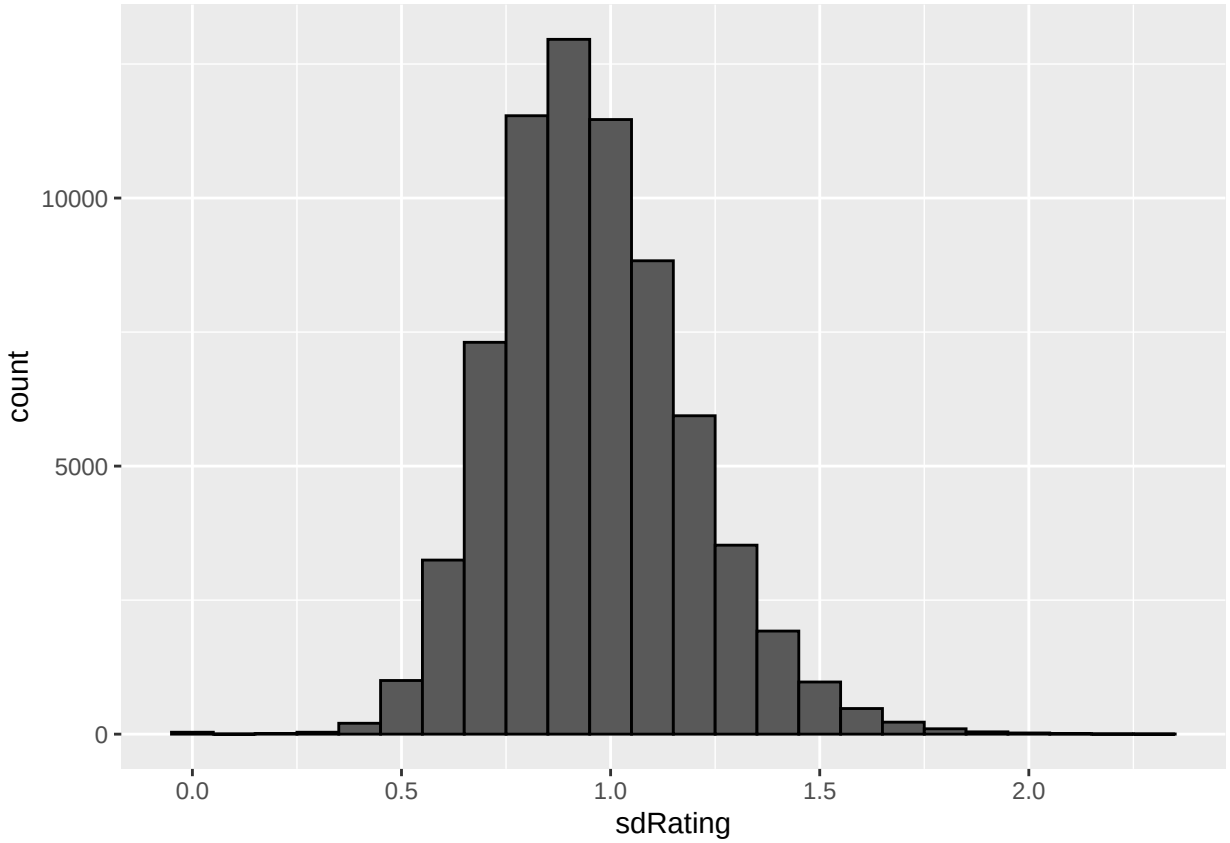



```
rating_summary_pro_user %>% filter(n > 500) %>% ggplot(aes(n)) + geom_histogram(color="black", binwidth
```



We can take a look to the standard deviation of the user ratings.

```
rating_summary_pro_user %>% ggplot(aes(sdRating)) + geom_histogram(color="black", binwidth = 0.1)
```



As we see, most of the user give a rating in 0,9 - 1 star standard deviation to the user average rating. Additionally we see, that there are some entries with zero standard deviation. That means, some user give always the same rating for every movies they rate. We can use this information for the prediction: if we have to predict the movie rating for a this user, we can predict the average movie rating calculated for the user from the training set without considerate any further information.

The count of the user with 0 standard deviation ratings in the training set is:

```
sum(rating_summary_pro_user$sdRating == 0)
```

```
## [1] 36
```

####Model Based on the movie and user data analysis we can develop a prediction model.

If we don't have any information about the movie and the user in the test set (new movie and new user), we can use the overall average rating from the training set rating data. This prediction will minimase the RSME value for unknown movies and user.

If we have rating data about the movie to predict we can assume, that the abriviation of the movie from the overall average shows the quality of the movie. We can use this information to tune the prediction value.

If we have rating data about the user to predict we can assume, that the abriviation of the user from the overall average corrected with the movie bias shows the rating mood of the user.

Implementation & result

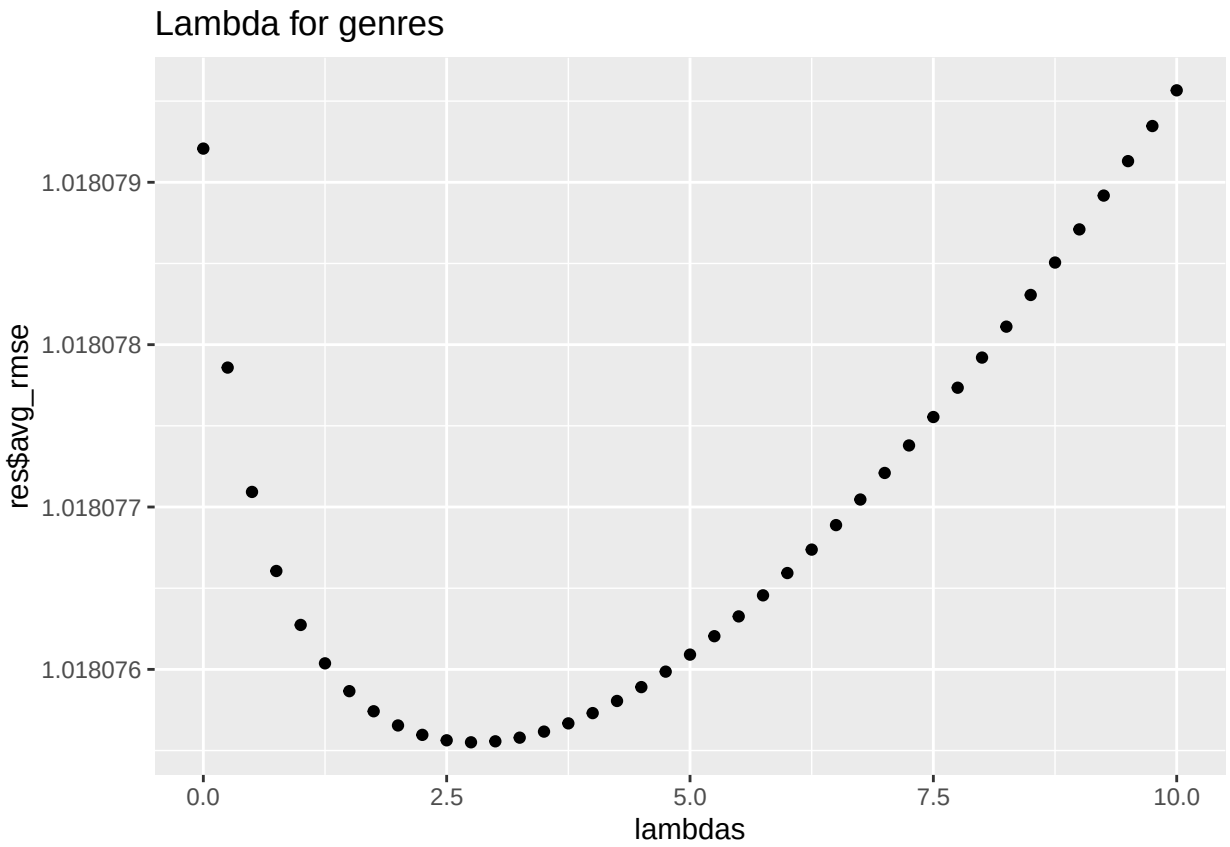
For the regularised movie and user bias calculation we need the lambda penality term that minimises the RMSE value. To find the optimum value for the penality terms we may only use the train set, therefore we will use cross fold validation for this porpuse. We will calculate the penality terms in two step: first we

calculate the penaltiy term for the movie bias and in the second step we calculate the penality term for the user bias using the result from step 1.

The following R code calculates the RMSE value for the given lambda range (0 up to 10, step by 0.25) using 5 fold cross validation:

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

#Movie genres lambda with cross validation
rmses_for_genres_lambdas <- function(train, test){

  RMSE_all = data.frame(lambda = numeric(), rmse = numeric())
  avg <- mean(train$rating)
  genre_sum <- train %>% group_by(genres) %>% summarise(s=sum(rating-avg), n_i=n())


  for (l in lambdas){
    predicted_ratings <- test %>%
      left_join(genre_sum, by='genres') %>%
      mutate(b_g = s / (n_i + l)) %>%
      mutate(pred = avg + b_g) %>%
      pull(pred)

    rmse_akt =RMSE(predicted_ratings, test$rating)
    RMSE_all <- bind_rows(RMSE_all, data.frame(lambda = l, rmse = rmse_akt))
  }
  return(RMSE_all)
}

#Movie lambda with cross validation
rmses_for_movie_lambdas <- function(train, test){

  RMSE_all = data.frame(lambda = numeric(), rmse = numeric())
  avg <- mean(train$rating)
  b_genres <- train %>%
    group_by(genres) %>%
    summarize(b_genres = sum(rating - avg) / (n() + genres_lambda))

  movie_sum <- train %>%
    left_join(b_genres, "genres") %>%
    group_by(movieId) %>%
    summarize(s = sum(rating - b_genres - avg), n_i= n())

  for (l in lambdas){
    predicted_ratings <- test %>%
      left_join(b_genres, by='genres') %>%
      left_join(movie_sum, by='movieId') %>%
      mutate(b_i = s / (n_i + l)) %>%
      mutate(pred = avg + b_genres + b_i) %>%
      pull(pred)

    rmse_akt =RMSE(predicted_ratings, test$rating)
    RMSE_all <- bind_rows(RMSE_all, data.frame(lambda = l, rmse = rmse_akt))
  }
  return(RMSE_all)
```

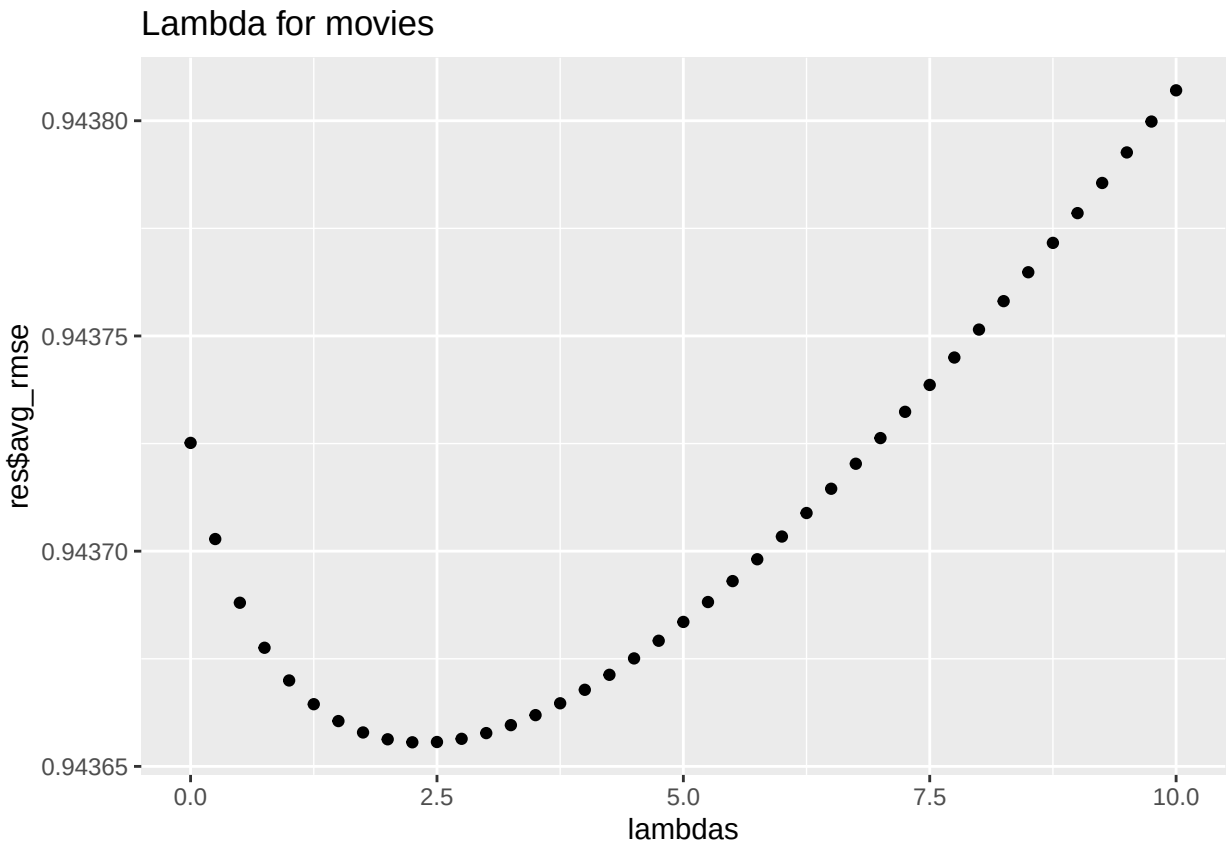
```
}

#User lambda with cross validation
rmses_for_user_lambdas <- function(train, test){

  RMSE_all = data.frame(lambda = numeric(), rmse = numeric())
  avg <- mean(train$rating)
  b_genres <- train %>%
    group_by(genres) %>%
    summarize(b_genres = sum(rating - avg) / (n() + genres_lambda))

  b_movie <- train %>%
    left_join(b_genres, by="genres") %>%
    group_by(movieId) %>%
    summarize(b_movie = sum(rating - b_genres - avg) / (n() + movie_lambda))

  user_sum <- train %>%
    left_join(b_genres, by="genres") %>%
    left_join(b_movie, by="movieId") %>%
    group_by(userId) %>%
    summarize(s = sum(rating - b_movie - b_genres - avg), n_i= n())

  for (l in lambdas){
    predicted_ratings <- test %>%
      left_join(b_genres, by="genres") %>%
      left_join(b_movie, by='movieId') %>%
      left_join(user_sum, by='userId') %>%
      mutate(b_u = s / (n_i + l)) %>%
      mutate(pred = avg + b_genres + b_movie + b_u) %>%
      pull(pred)

    rmse_akt =RMSE(predicted_ratings, test$rating)
    RMSE_all <- bind_rows(RMSE_all, data.frame(lambda = l, rmse = rmse_akt))
  }
  return(RMSE_all)
}

res <- cross_validation(edx, 5, rmses_for_genres_lambdas)
res <- res %>% group_by(lambda) %>% summarise(avg_rmse=mean(rmse))
qplot(main=c('Lambda for genres', 'RMSE', 'Lambda'), lambdas, res$avg_rmse)
```




```
genres_lambda = lambdas[which.min(res$avg_rmse)]
cat("Lambda for genres regulaisration: ", genres_lambda)
```

```
## Lambda for genres regulaisration:  2.75
```

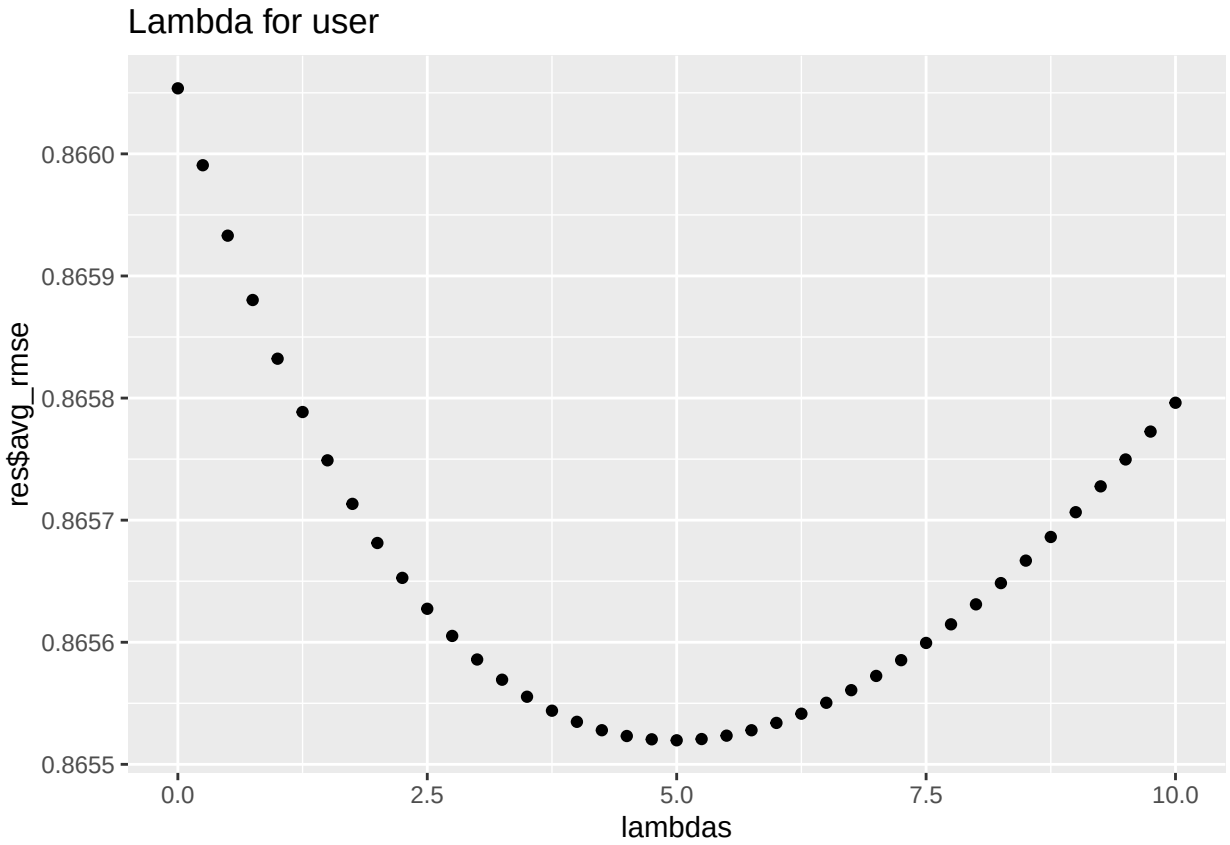
```
res <- cross_validation(edx, 5,rmses_for_movie_lambdas)
res <- res %>% group_by(lambda) %>% summarise(avg_rmse=mean(rmse))
qplot(main=c('Lambda for movies', 'RMSE', 'Lambda'), lambdas, res$avg_rmse)
```




```
movie_lambda = lambdas[which.min(res$avg_rmse)]
cat("Lambda for movie regulaisration: ", movie_lambda)
```

```
## Lambda for movie regulaisration:  2.25
```

```
res <- cross_validation(edx, 5, rmses_for_user_lambdas)
res <- res %>% group_by(lambda) %>% summarise(avg_rmse=mean(rmse))
qplot(main=c('Lambda for user', 'RMSE', 'Lambda'), lambdas, res$avg_rmse)
```

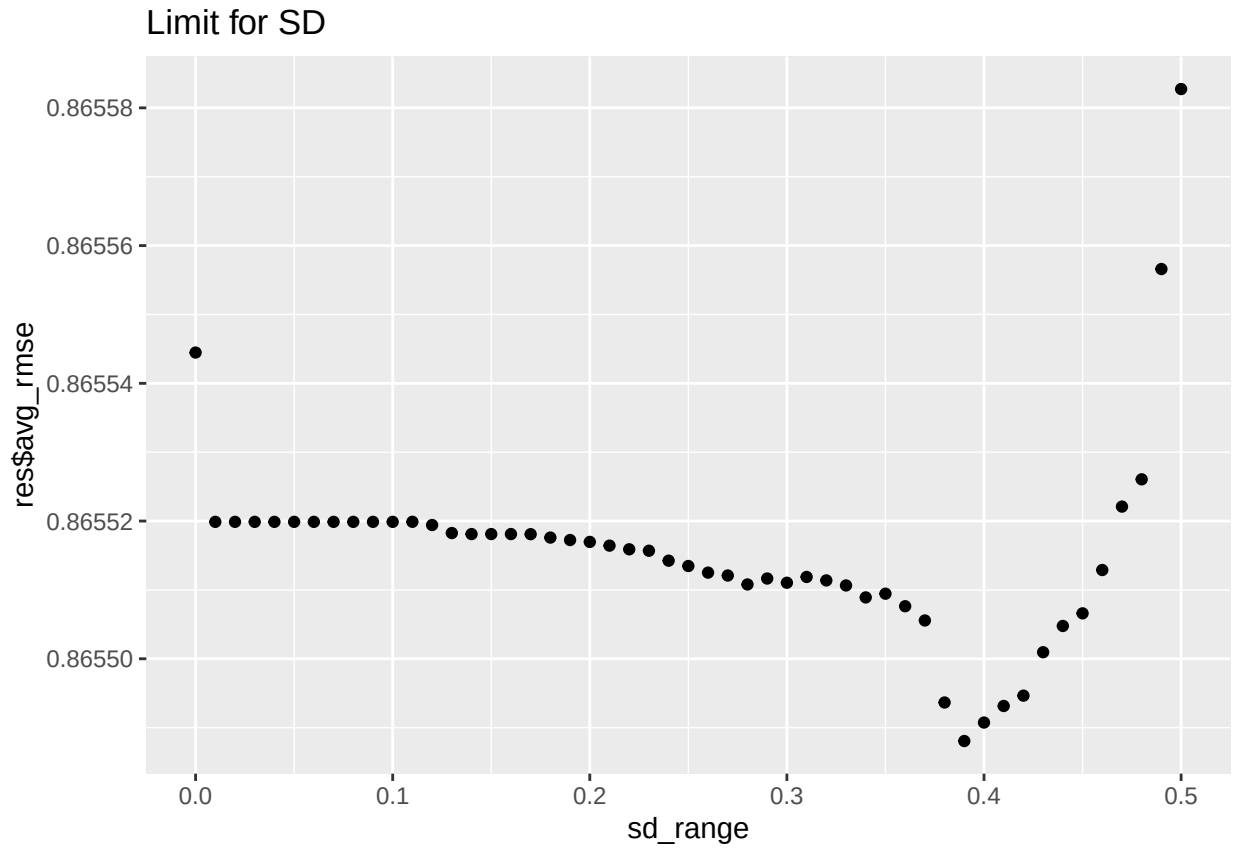



```
user_lambda = lambdas[which.min(res$avg_rmse)]
cat("Lambda for user regulaisration: ", user_lambda)
```

```
## Lambda for user regulaisration:  5
```

```
#SD limit with cross validation
rmses_for_sd_ranges <- function(train, test, range){

  RMSE_all = data.frame(lambda = numeric(), rmse = numeric())
  avg <- mean(train$rating)
  b_genres <- train %>%
    group_by(genres) %>%
    summarize(b_genres = sum(rating - avg) / (n() + genres_lambda))

  b_movie <- train %>%
    left_join(b_genres, by="genres") %>%
    group_by(movieId) %>%
    summarize(b_movie = sum(rating - b_genres - avg) / (n() + movie_lambda))

  b_user <- train %>%
    left_join(b_movie, by="movieId") %>%
    left_join(b_genres, by="genres") %>%
    group_by(userId) %>%
    summarize(b_user = sum(rating - b_movie - b_genres - avg) / (n() + user_lambda), sd_userrating=sd(ra

  for (r in range){
    predicted_ratings <- test %>%
```

```
      left_join(b_genres, by="genres") %>%
      left_join(b_movie, by='movieId') %>%
      left_join(b_user, by='userId') %>%
      mutate(pred = ifelse(sd_userrating < r, avg_user, avg + b_genres + b_movie + b_user)) %>%
      pull(pred)

    rmse_akt =RMSE(predicted_ratings, test$rating)
    RMSE_all <- bind_rows(RMSE_all, data.frame(r = r, rmse = rmse_akt))
  }
  return(RMSE_all)
}

sd_range <- seq(0, 0.5, 0.01)
res <- cross_validation(edx, 5, rmses_for_sd_ranges, sd_range)
res <- res %>% group_by(r) %>% summarise(avg_rmse=mean(rmse))
qplot(main=c('Limit for SD', 'RMSE', 'SD Limit'), sd_range, res$avg_rmse)
```




```
sd = sd_range[which.min(res$avg_rmse)]
cat("SD limit: ", sd)
```

```
## SD limit:  0.39
```

The rating prediction can be calculated with th following R code:

```
avg_rating = mean(edx$rating)

#bias based on the genres
```

```r
genre_bias <- edx %>% group_by(genres) %>% summarise(b_genre = mean(rating - avg_rating))
genre_bias_reg <- edx %>% group_by(genres) %>% summarise(b_genre = sum(rating - avg_rating)/(n() + genre

#prediction based on avg und genres bias
prediction0 <- validation %>% left_join(genre_bias_reg, by="genres")  %>% mutate(pred = avg_rating + b_g


#bias based on the movies
movie_bias <- edx %>% left_join(genre_bias, by="genres") %>%
      group_by(movieId) %>% summarise(b_movie = mean(rating - avg_rating - b_genre))

movie_bias_reg <- edx %>% left_join(genre_bias_reg, by="genres") %>%
      group_by(movieId) %>% summarise(b_movie = sum(rating - avg_rating - b_genre) / (n() + movie_lambda

#prediction based on avg, genre and movie bias
prediction1 <- validation %>% left_join(genre_bias, by="genres") %>%
      left_join(movie_bias,by="movieId") %>% mutate(pred = avg_rating + b_genre + b_movie)

#prediction based on avg, genre and movie bias regularised
prediction2 <- validation %>% left_join(genre_bias_reg, by="genres") %>%
      left_join(movie_bias_reg,by="movieId") %>% mutate(pred = avg_rating + b_genre + b_movie)

#bias based on the user
user_bias <- edx %>% left_join(genre_bias, by="genres") %>% left_join(movie_bias, by="movieId") %>%
      group_by(userId) %>%
      summarise(b_user = mean(rating - avg_rating - b_genre - b_movie), avg_user=mean(rating), sd_userra

user_bias_reg <- edx %>% left_join(genre_bias_reg, by="genres") %>%
      left_join(movie_bias_reg, by="movieId") %>%  group_by(userId) %>%
      summarise(b_user = sum(rating - avg_rating - b_genre - b_movie)/(n()+user_lambda), avg_user=mean(

user_avg_rating <- edx %>% group_by(userId) %>% summarise(avg_rating = mean(rating))

#prediction based on avg, genre, movie and user bias
prediction3 <- validation %>% left_join(genre_bias, by="genres") %>%
      left_join(movie_bias,by="movieId") %>% left_join(user_bias, by = "userId")  %>%
      mutate(pred = avg_rating + b_genre + b_movie + b_user)

#prediction based on avg, movie and user bias regularised
prediction4 <- validation %>% left_join(genre_bias_reg, by="genres") %>%
      left_join(movie_bias_reg,by="movieId") %>% left_join(user_bias_reg, by = "userId")  %>%
      mutate(pred = avg_rating + b_genre + b_movie + b_user)

#prediction based on avg, movie and user bias regularised with min/max
prediction4b <- prediction4
prediction4b[prediction4b$pred > 5,]$pred = 5
prediction4b[prediction4b$pred < 0.5,]$pred = 0.5

#prediction based on avg, genre, movie and user bias regulated
prediction5 <- validation %>% left_join(genre_bias_reg, by="genres") %>%
      left_join(movie_bias_reg,by="movieId") %>% left_join(user_bias_reg, by = "userId")  %>%
      mutate(pred = ifelse(sd_userrating < 0.4, avg_user, avg_rating + b_genre + b_movie + b_user))
```

```r
prediction5b <- prediction5
prediction5b[prediction5b$pred > 5,]$pred = 5
prediction5b[prediction5b$pred < 0.5,]$pred = 0.5


#performance of the avarage
print("Avarage")
```

```
## [1] "Avarage"
```

```r
RMSE(validation$rating, avg_rating)
```

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

```r
print("RMSE for genre effect reg")
```

```
## [1] "RMSE for genre effect reg"
```

```r
RMSE(validation$rating, prediction0$pred)
```

```
## [1] 1.018406
```

```r
print("RMSE for genre and movie effect")
```

```
## [1] "RMSE for genre and movie effect"
```

```r
RMSE(validation$rating, prediction1$pred)
```

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

```r
print("RMSE for genre and movie effect regulasired")
```

```
## [1] "RMSE for genre and movie effect regulasired"
```

```r
RMSE(validation$rating, prediction2$pred)
```

```
## [1] 0.9438527
```

```r
print("RMSE for genre, movie and usert effect")
```

```
## [1] "RMSE for genre, movie and usert effect"
```

```r
RMSE(validation$rating, prediction3$pred)
```

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

```r
print("RMSE for genre, movie and usert effect regularised")
```

```
## [1] "RMSE for genre, movie and usert effect regularised"
```

```r
RMSE(validation$rating, prediction4$pred)
```

```
## [1] 0.8648431
```

```r
print("RMSE for genre, movie and usert effect regularised mit min&max")
```

```
## [1] "RMSE for genre, movie and usert effect regularised mit min&max"
```

```r
RMSE(validation$rating, prediction4b$pred)
```

```
## [1] 0.8647318
```

```r
print("RMSE for genre, movie and usert effect regularised with sd")
```

```
## [1] "RMSE for genre, movie and usert effect regularised with sd"
```

```
RMSE(validation$rating, prediction5$pred)
```

```
## [1] 0.8647521
```

```
print("RMSE for genre, movie and usert effect regularised with sd and min&max")
```

```
## [1] "RMSE for genre, movie and usert effect regularised with sd and min&max"
```

```
RMSE(validation$rating, prediction5b$pred)
```

```
## [1] 0.8646461
```

Conclusion