Data Science 'MovieLens' Capstone Project

András Gelencsér

10 November 2019

Overview

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

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

The used dataset contains movie rating data set collected by GroupLens Research. The full MoviLens dataset include more than 20 million movie ratings for more than 27,000 movies by 138,000 users. The rated movies cover different time periods. The data sets are available 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 evaluation 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. Analyze the data structure
3. Define the recommendation model based on the result the data analysis
4. Implement and train the model based on the train set
5. Review the model based on the test set

The recommendation model developed in this project achieved an RMSE value of 0.8646461 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 required packages if needed:

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

The quality of the developed algorithm will be evaluated be the root mean squared error (RMSE). Therefore, we will need a function for calculating the RMSE value for a prediction.

The RMSE can calculated with the following formula:

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

, where N is the count of all user-movie rating combinations, $\hat{y}_{u,m}$ is the rating prediction for the user "u" and movie "m" combination and $y_{u,m}$ is the real rating for the user "u" and movie "m" combination.

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
#return:
#     the root mean squared error of the prediction

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

Because we may not use the validation set of data for optimizing the algorithm parameters, we will use K-fold cross validation.

The cross validation uses only the train set und split the data in the train set in two disjunct parts. These two data parts is used as new train and test set for evaluating the model parameter. T

For the cross-fold validation, we used a part of the train set as test set, therefore the result of the cross-fold validation approximates only the true error.

To reduce the noise in the error estimation we use K-fold cross validation. In the K-fold cross validation we repeat the data splitting K times for different part of the data set and calculate the average of the RMSE value of the results.

The following R function implements 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 from the param trainset)
#     ...:      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 train set
    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)
}
```

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/
```

```r
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

#flag for marking file download
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.factor(genres))

# 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)` 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, by=c("userId", "movieId"))
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 genre 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 genre 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)
```

The movielens dataset contains about 700 different genre combinations from 20 genres. Each movie belongs to one of the genre combinations.

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 movie 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(timestamp))- movie_year)

validation <- validation %>% left_join(movie_data, by="movieId") %>%
      mutate(movie_age_by_rating = year(as_datetime(timestamp))-movie_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)
```

Movie data analysis

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

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

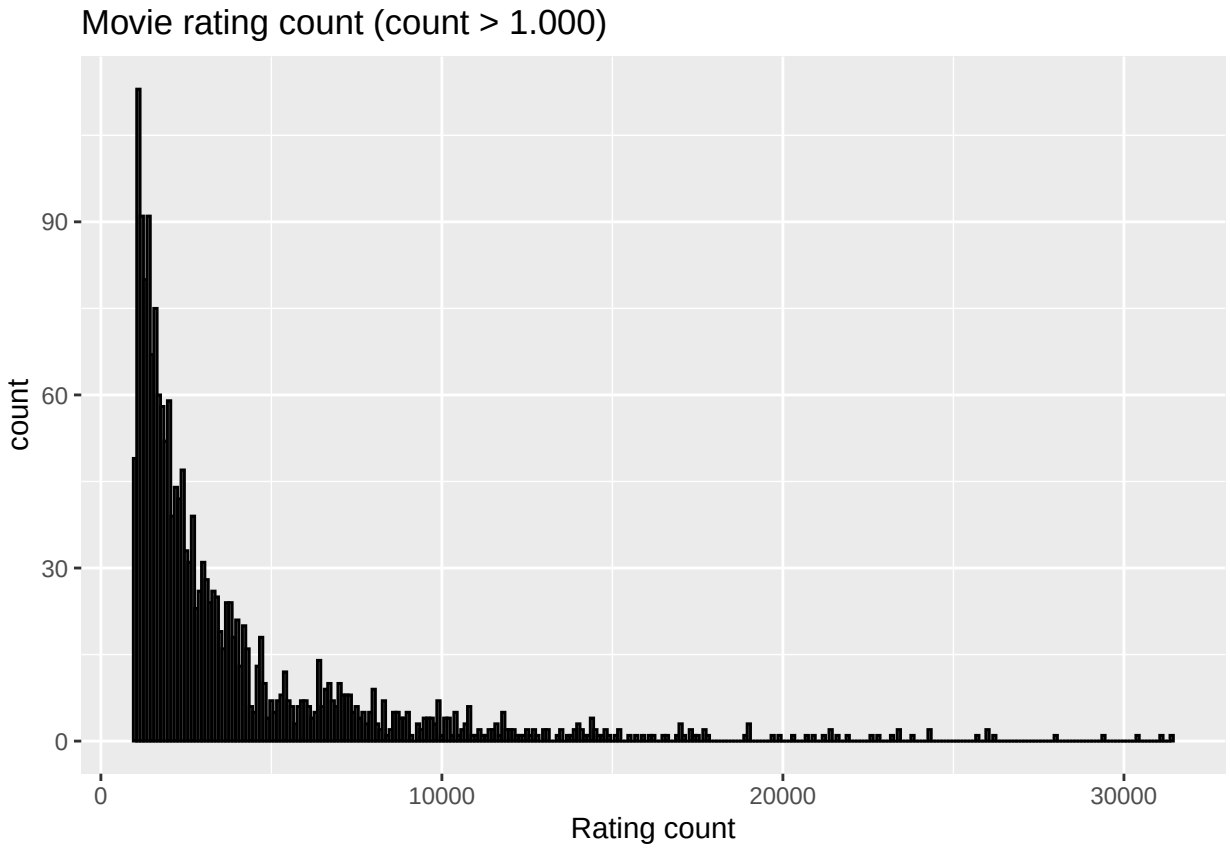
If we don‘t have any additional information about the movies and user, we can use this average for predicting the rating movie rating.

We can calculate the average rating for each movie and the abbreviation of this calculated average from the overall rating average.

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

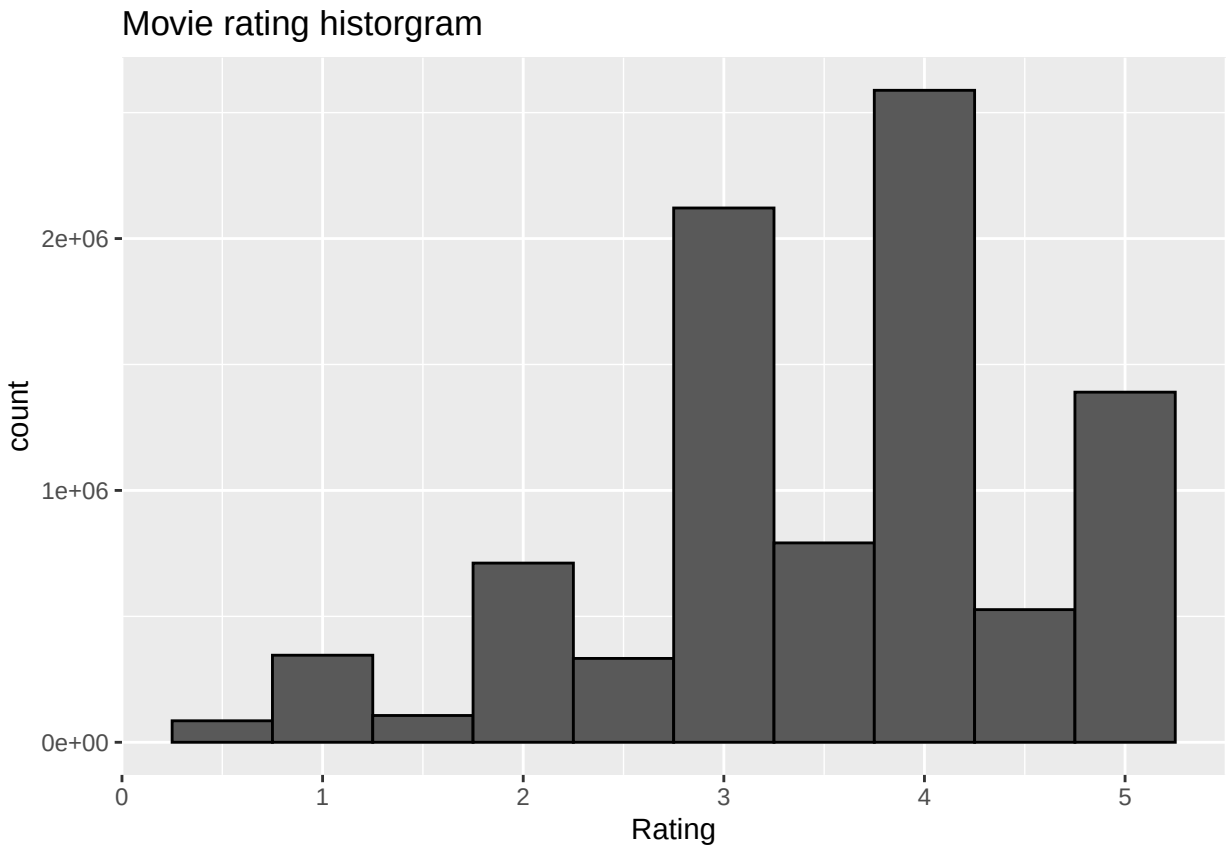
If we examine the histogram of the movies, we see, that some movies were rated more often than other. The following histogram shows the rating count for movies that were rated more than 1000 times:

```
#plot the histogram oft the rating count (for movies with more than 1000 rating)
rating_summary_pro_movie %>% filter(n > 1000) %>% ggplot(aes(n)) +
  geom_histogram(color="black", binwidth = 100) +
  ggtitle("Movie rating count (count > 1.000)") +
  xlab("Rating count") + ylab("count")
```



We can create a histogram about the ratings in the train set

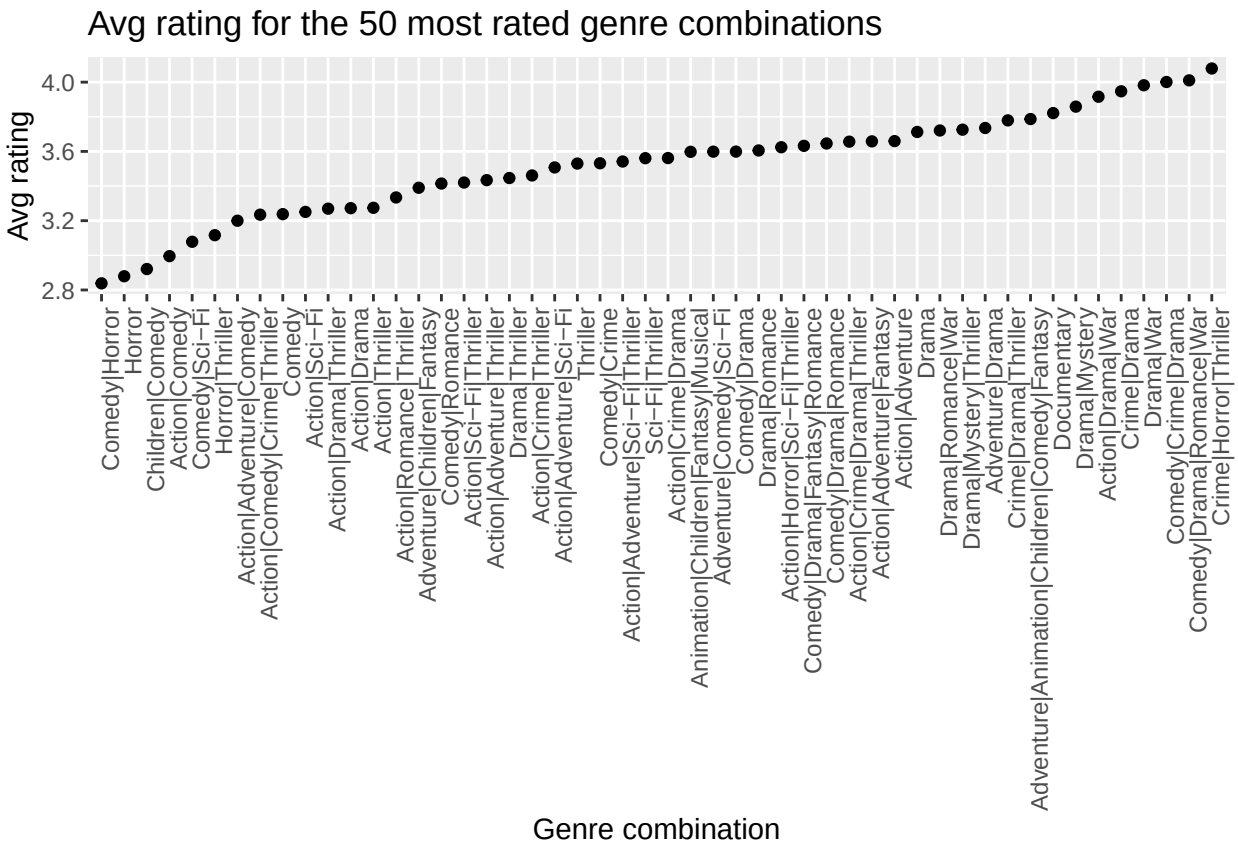
```
#plot the histogram for the rating
edx %>% ggplot(aes(rating)) + geom_histogram(color="black", binwidth = 0.5)+
  ggtitle("Movie rating historgram") +
  xlab("Rating") + ylab("count")
```



In the histogram we see, that half star ratings are rare compare to whole star ratings.

As we saw in the chapter "Genres data", each movie belongs to a specific genre combination. We can calculate the average movie rating for each genre combination in the train dataset. The following R code draws a plot chart for the average rating for the genre combination with more than 25000 ratings ordered by the average rating:

```
#calculate the avg rating and count pro genre
genres_rating <- edx %>%
                 group_by(genres) %>% summarise(avg=mean(rating), sd=sd(rating), n=n())

#plot the avg rating the 50 most rated genres combination ordered by avg rating
head(genres_rating %>% arrange(desc(n)),50) %>% mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(genres,avg)) + geom_point() +
  ggtitle("Avg rating for the 50 most rated genre combinations") +
  xlab("Genre combination") + ylab("Avg rating") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

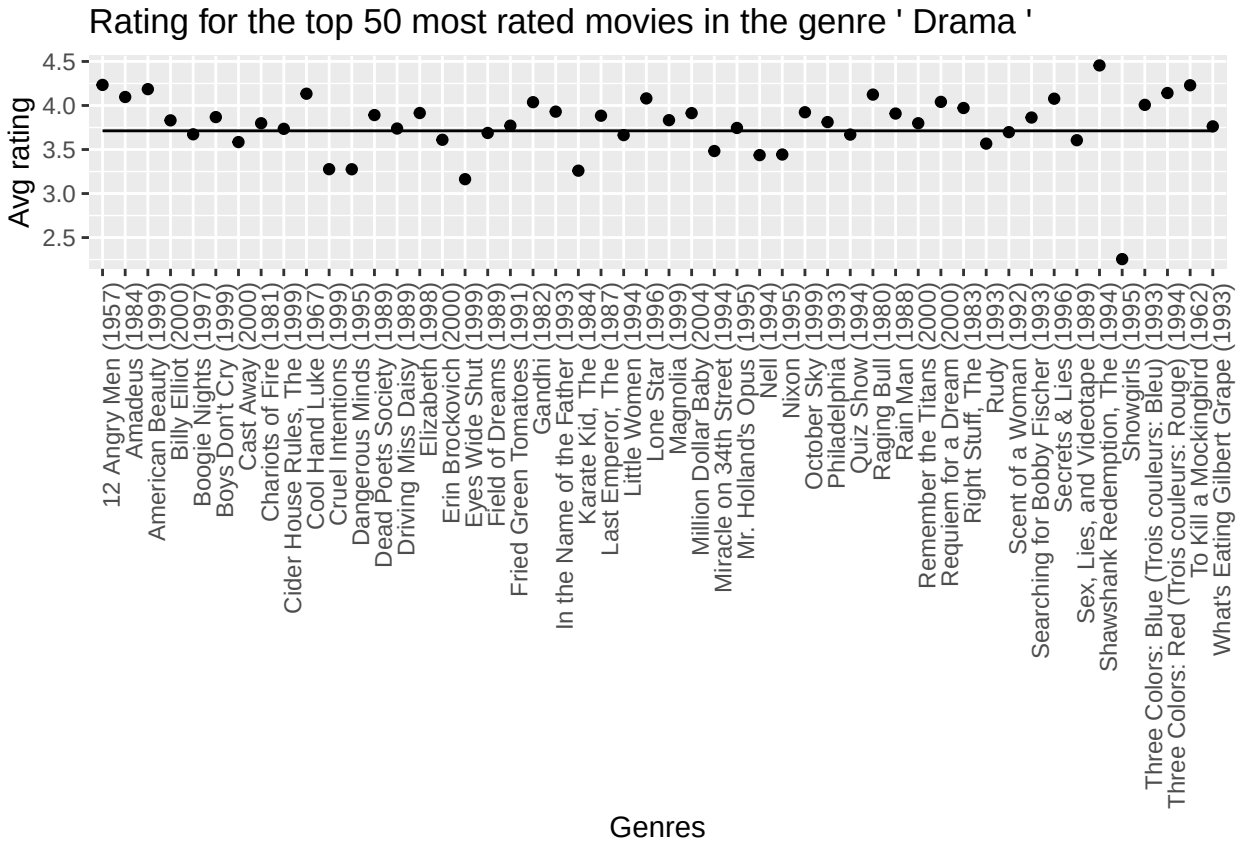


As we can see, there are more than 1-star rating difference between the rating for the genre combination 'Crime|Horror|Thriller' and 'Comedy|Horror'. It looks like, the genre combination of a movie is a relevant factor for the rating, some genre combinations are more popular as other combinations.

We can examine, if there are differences between movies in a specific genre combination. For the analysis we search the genre combination with the most ratings. Afterward we plot the top 50 rated movies from this genre.

The following R code draws us the plot:

```
top_genre = genres_rating[which.max(genres_rating$n),]$genres
top_genre_avg = genres_rating[which.max(genres_rating$n),]$avg

head(rating_summary_pro_movie %>% inner_join(movie_data, by="movieId") %>%
    filter(genres==top_genre) %>% arrange(desc(n)),50) %>%
    ggplot(aes(title,avg)) + geom_point() +
    geom_line(aes(title,top_genre_avg), group=1) +
    ggtitle(paste("Rating for the top 50 most rated movies in the genre '",
                  top_genre, "'")) +
    xlab("Genres") + ylab("Avg rating") +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



The line in the middle of the chart shows the genre rating average. As we see there are some movies under this line. These movies were less preferred as the genre itself. On the other side, there are some movies about the average line. This are the movies that are better rated as the genre average.

We can examine other genres too, and we get the same result: some movies are better than the genre average, some movies are rated worst as the genre average.

That means, the movies have an individual rating factor on the top of the genre rating.

We can say, that a movie rating depends on the genre combination of the movie and on the movie itself.

User data analysis

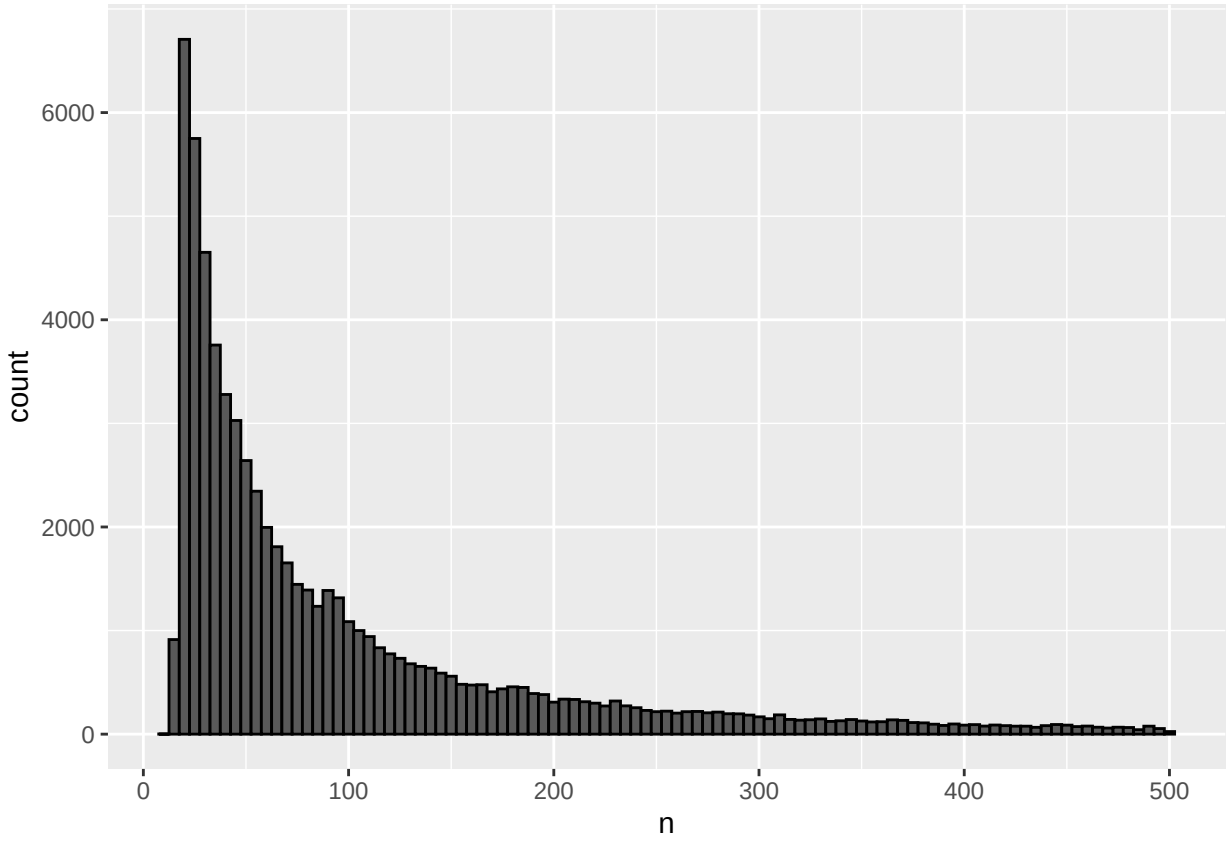
For analyzing the user relevant rating data, we calculate the count of the ratings, the average rating and the standard deviation o the ratings pro user.

The following R code calculates the summarized data pro user:

```
rating_summary_pro_user <- edx %>% group_by(userId) %>%
    summarise(n = n(), avgRating=mean(rating), sdRating=sd(rating))
```

As first, we can look for the rating count pro user. The following R code provides a histogram about the user rating count for the users with less than 500 rating:

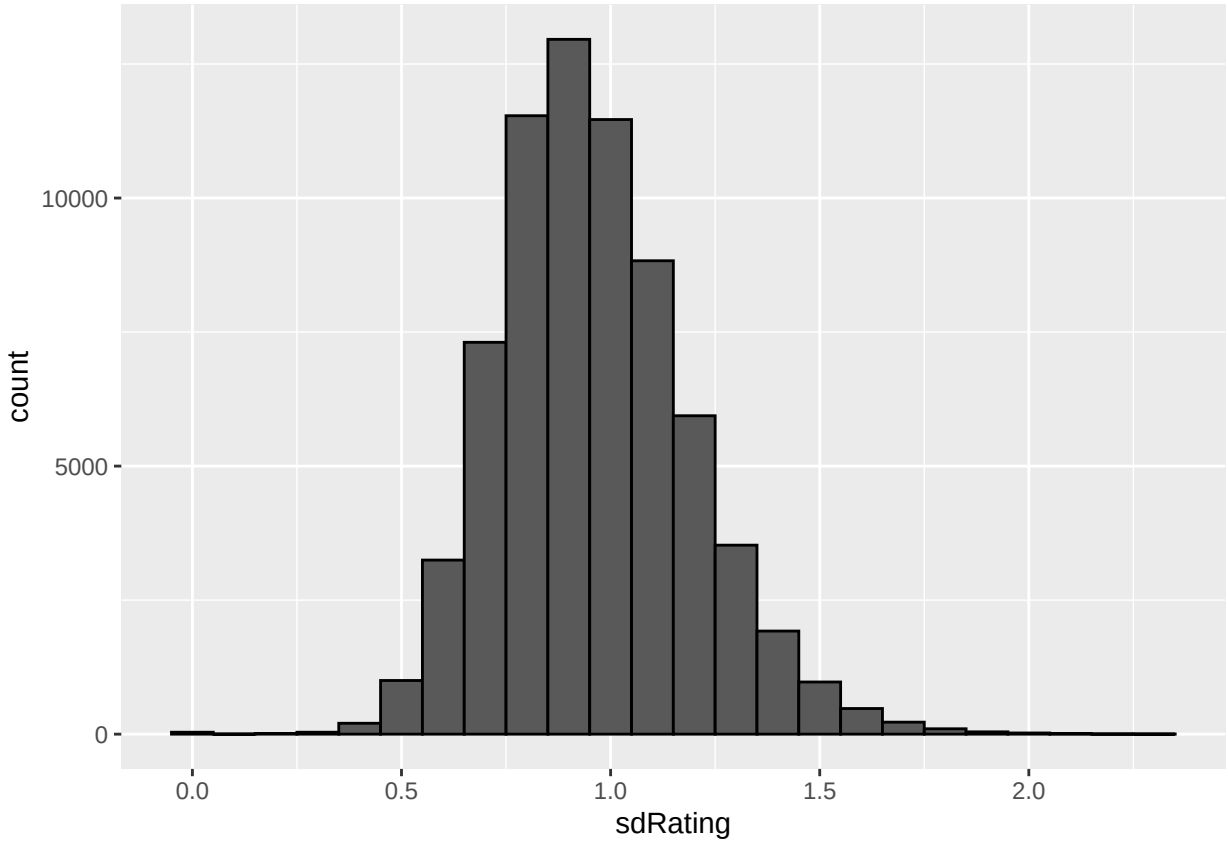
```
rating_summary_pro_user %>% filter(n < 500) %>% ggplot(aes(n)) +
    geom_histogram(color="black", binwidth = 5)
```



As we see, that most of the user have made between 10 and 30 ratings, but there are some users with more than 1000 rated movies.

We can look to the histogram of the user ratings standard deviation:

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



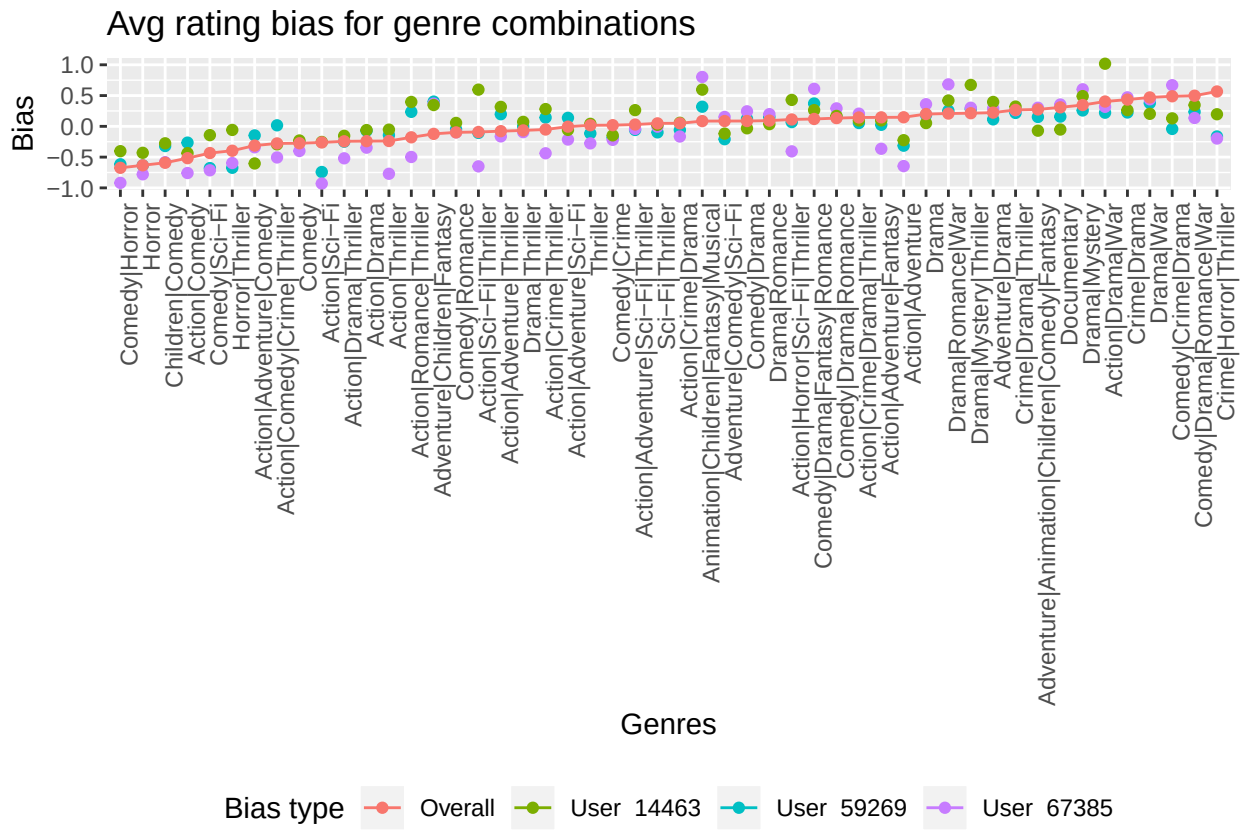
As we see, most of the user give a rating in 0,9 - 1-star standard deviation to the user average rating. Additionally, we see, that there are some entries with zero standard deviation. That means, some user gives always the same rating for every movie they rate. We can use this information for the prediction: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

We saw in the movie analysis chapter, that the genre combination has an impact to the movie rating. We can now examine if the genre combination rating pro user is the same as the overall genre combination average rating.

For this analysis we use the three users with the most ratings in the database. The following R code selects the user id for these users:

```
#summarise the rating information for the user/genre combinatoin for
#the users with more than 500 ratings
user_genre_ratings = rating_summary_pro_user %>% filter(n > 500) %>%
    inner_join(edx,by="userId") %>%
    group_by(userId, genres) %>% summarise(avg= mean(rating - avgRating), n=n())

#the count of users, we want to display in the plot
top_user_count = 3

genres_to_compare <- head(genres_rating %>% arrange(desc(n)),50)$genres
genres_user_rating <- data.frame(genres=character(), type=character(),
                                 avg=numeric(), stringsAsFactors=FALSE)

for(u in head(rating_summary_pro_user %>% arrange(desc(n)),top_user_count)$userId)
{
```

```
      genres_user_rating <- bind_rows(genres_user_rating,user_genre_ratings %>%
                              filter(userId==u & genres %in% genres_to_compare) %>%
                              mutate(type=paste('User ', u)) %>%
                              ungroup() %>% select(genres, type, avg) %>%
                              mutate(genres=as.character(genres)))
}

genres_user_rating <- bind_rows(genres_user_rating,
                                genres_rating %>% filter(genres %in% genres_to_compare) %>%
                                mutate(type='Overall', avg=avg-avg_overall) %>%
                                ungroup() %>% select(genres,type, avg) %>%
                                mutate(genres=as.character(genres)))


genres_user_rating <- genres_user_rating %>%
                      left_join(genres_rating %>% mutate(avg_o=avg) %>%
                      ungroup() %>% select(genres, avg_o)  %>%
                      mutate(genres=as.character(genres)), by="genres")

genres_user_rating <- genres_user_rating %>% mutate(genres = as.factor(genres))


genres_user_rating %>%  mutate(genres = reorder(genres,avg_o)) %>%
  ggplot(aes(genres, avg, color=type))  +
  geom_point() +
  geom_line(data=genres_user_rating%>%filter(type=='Overall') %>%
              mutate(genres = reorder(genres,avg_o)),aes(genres,avg), group=1) +
  ggtitle("Avg rating bias for genre combinations") +
  xlab("Genres") + ylab("Bias") +
  labs(color='Bias type') +
  theme(axis.text.x = element_text(angle = 90, hjust = 1), legend.position = 'bottom')
```



As we see, the users have different preferences for the different genres. For example, the average rating bias of the genre combination 'Action|Sci-Fi|Thriller' is -0.092. The bias of this genre for the user with id '14463' is 0.596 (the user like this genre combination) but the bias for the user with id '67385' is -0.650 (the user doesn't like this genre combination).

Therefore, we can use the user genre combination preference to improve our prediction.

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 minimize the RSME value for unknown movies and user. The overall average can be calculated with the following formula:

$$avg = \frac{1}{N}\sum_{i=1}^{n} r_i$$

If we have rating data about the genre combination to predict, we can assume, that the abbreviation of the genre combination average rating from the overall average shows the quality of the genre combination. We can use this genre bias to tune the prediction value. This genre combination bias can be calculated by the following formula from the train set:

$$b\_genre_g = \frac{1}{N_g}\sum_{i=1}^{N_g} r_i - avg$$

where N_g is the count of the ratings for the genre combination g and r_i is the rating for the genre combination and avg ist the overall average calculated in the previous step.

If we have rating data about the movie to predict we can assume, that the abbreviation of the movie rating from the overall average and the genres combination shows the quality of the movie. We can use this movie bias to improve the prediction value.

The movie bias formula uses the overall average and genres bias values calculated in the previous steps.

$$b\_movie_m = \frac{1}{N_m} \sum_{m}^{N_m} r_m - b\_genre_g - avg$$

where N_m is the count of the ratings for the movie m and r_m is the rating for the movie m, b_genre_g is the genre combination bias for the g genre combination of the movie m and avg ist the overall rating average .

If we have rating data about the user to predict we can assume, that the abbreviation of the user rating from the overall average corrected with the genre and movie bias shows the rating mood of the user. We can add this user bias information to our model too. The user bias formula uses the bias results from the previous steps.

$$b\_user_u = \frac{1}{N_u} \sum_{u}^{N_u} r_{u,m} - b\_genre_g - b\_movie_m - avg$$

where N_u is the count of the ratings for the user u and $r_{u,m}$ is the rating for the user u and movie m, b_genre_g is the genre combination bias for the g genre combination of the movie m and avg ist the overall rating average .

As prediction for the use u and movie m we will use the value

$$\hat{y}_{u,m} = avg + b\_genre_g + b\_movie_m + b\_user_u$$

where avg is the overall rating average, b_genre_g is the genre combination bias for the g genre combination of the movie m, the b_movie_m is the movie bias for the movie m and b_user_u is the user bias for the user u.

We know, that the maximum allowed rating is 5. The minimum allowed rating is 0.5. Because of the different bias, there can be predictions above and under this range. (Think about to predict an excellent movie (movie bias > 1) in a very popular genre combination (genre bias > 0.5) for a very nice user (user bias > 1) at the overall average rating of 3.5. Our prediction model would predict a value above 5. On the other side, we can reach a prediction under the minimum possible rating for some other constellations.)

To prevent over- or underrating, we can use the minimum and maximum range for the prediction. The updated prediction model is:

$$\hat{y}_{u,m} = \begin{cases} 5 & avg + b\_genre_g + b\_movie_m + b\_user_u > 5 \\ 0.5 & avg + b\_genre_g + b\_movie_m + b\_user_u < 0.5 \\ avg + b\_genre_g + b\_movie_m + b\_user_u & otherwise \end{cases}$$

We saw, that some user has a low rating standard deviation. In this case, we can assume, that the user will give the same rating for a new movie as his calculated average rating is, independent from the overall rating average and from the genre and movie bias for the new movie.

Therefore, we can improve our rating model with this information:

$$\hat{y}_{u,m} = \begin{cases} avg_u & sd_u <= sd\_rating\_limit \\ 5 & avg + b\_genre_g + b\_movie_m + b\_user_u > 5 \\ 0.5 & avg + b\_genre_g + b\_movie_m + b\_user_u < 0.5 \\ avg + b\_genre_g + b\_movie_m + b\_user_u & otherwise \end{cases}$$

where avg_u is the average rating for the user u. We must find out the sd_rating_limit for our model. For this calculation we will train our model with different limit values until we find the best RMSE value for the model.

In the model we calculated bias values with an average function. This average function doesn't consider the different count of the ratings. E.g. if we have a genre combination with plenty ratings, the bias for this genre combination will be near to the real genre bias value. If we have only few ratings, we can have a higher error to the real genre bias value.

To correct this error in our model, we use regularization for the different bias. We will train our model with different penalty terms to find out the optimal value for our final model.

Implementation & result

For the regularized genre, movie and user bias calculation we need the lambda penalty term that minimizes the RMSE value. To find the optimum value for the penalty terms we may only use the train set, therefore we will use cross fold validation for this purpose. We will calculate the penalty terms in different steps: first we calculate the penalty term for the genre bias, afterwards for the movie bias and at the end we calculate the penalty term for the user bias.

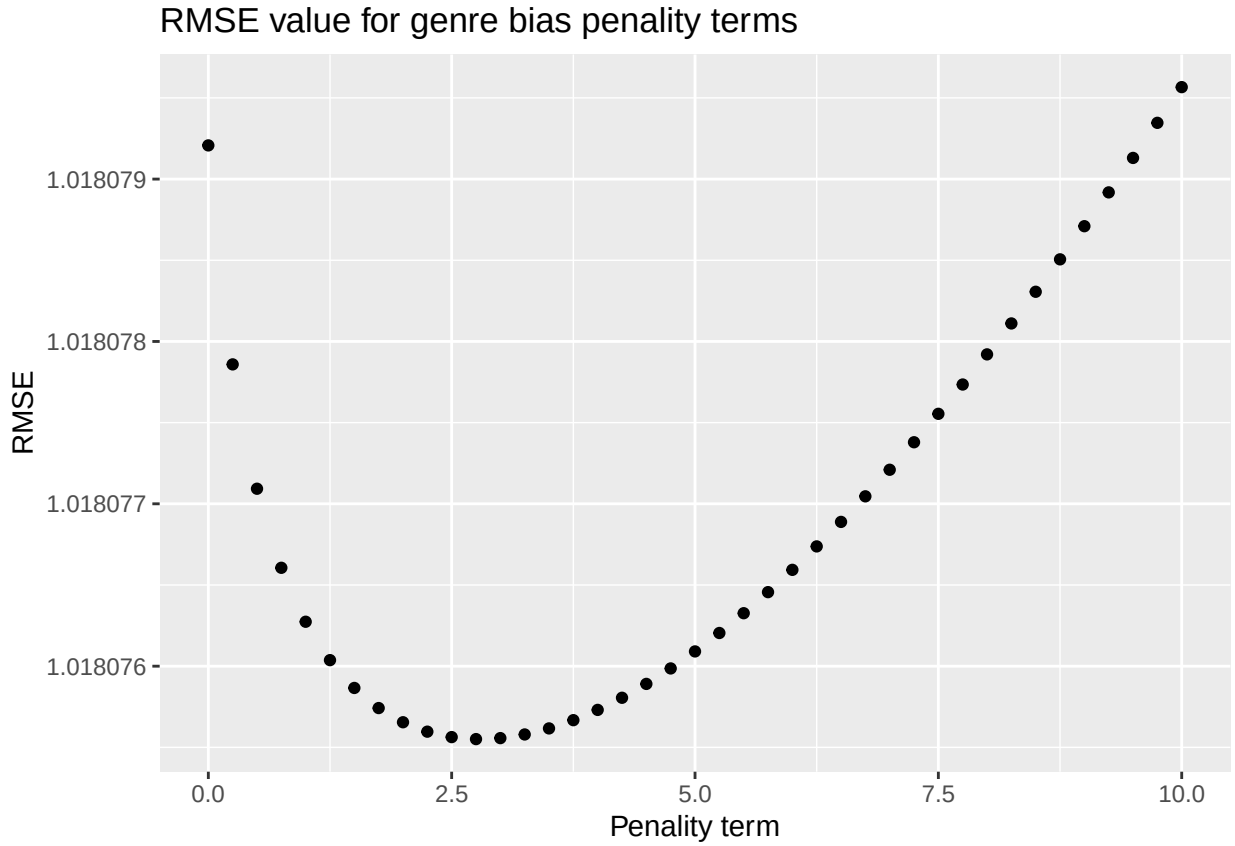
For the first iteration we will search the penalty terms in the range 0 up to 10 (step by 0.25). If we don't find a penalty term that minimizes the RMSE value our model, we will extend the range.

As we mentioned, we may not use the test set for optimizing the penalty terms, therefor we will use K fold cross validation.

The following R code calculates the RMSE value for the genre bias penalty term in 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)
}

res <- cross_validation(edx, 5, rmses_for_genres_lambdas)
res <- res %>% group_by(lambda) %>% summarise(avg_rmse=mean(rmse))
qplot(main=c('RMSE value for genre bias penality terms'), lambdas, res$avg_rmse) +
    xlab("Penality term") + ylab("RMSE")
```




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

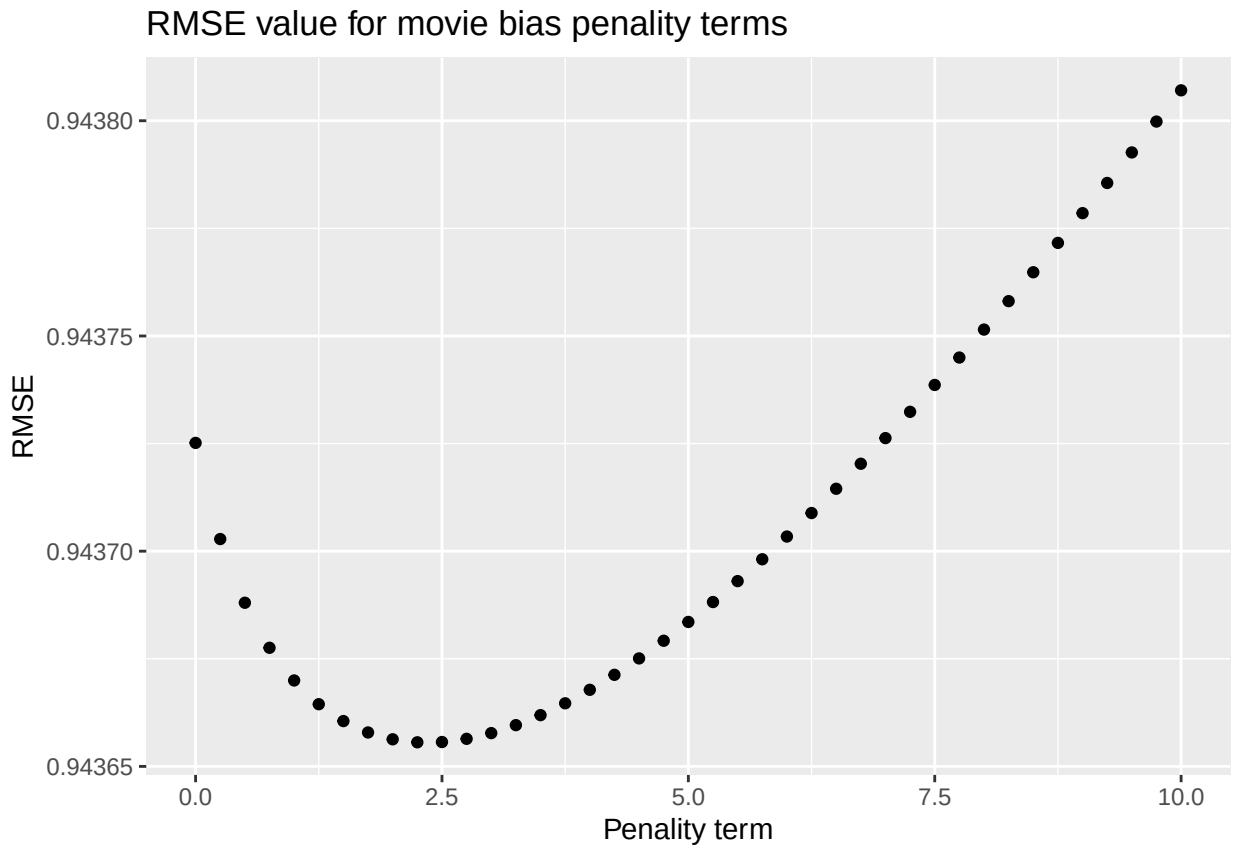
```
## Lambda for genres regularization:  2.75
```

We can see in the plot, which penalty term value minimizes the genre bias RMSE value

The following R code calculates the RMSE value for the movie bias penalty term in the given lambda range using 5-fold cross validation. The function uses the found genre bias penalty value from the previous step.

```
#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)
}

res <- cross_validation(edx, 5,rmses_for_movie_lambdas)
res <- res %>% group_by(lambda) %>% summarise(avg_rmse=mean(rmse))
qplot(main=c('RMSE value for movie bias penality terms'), lambdas, res$avg_rmse)  +
    xlab("Penality term") + ylab("RMSE")
```

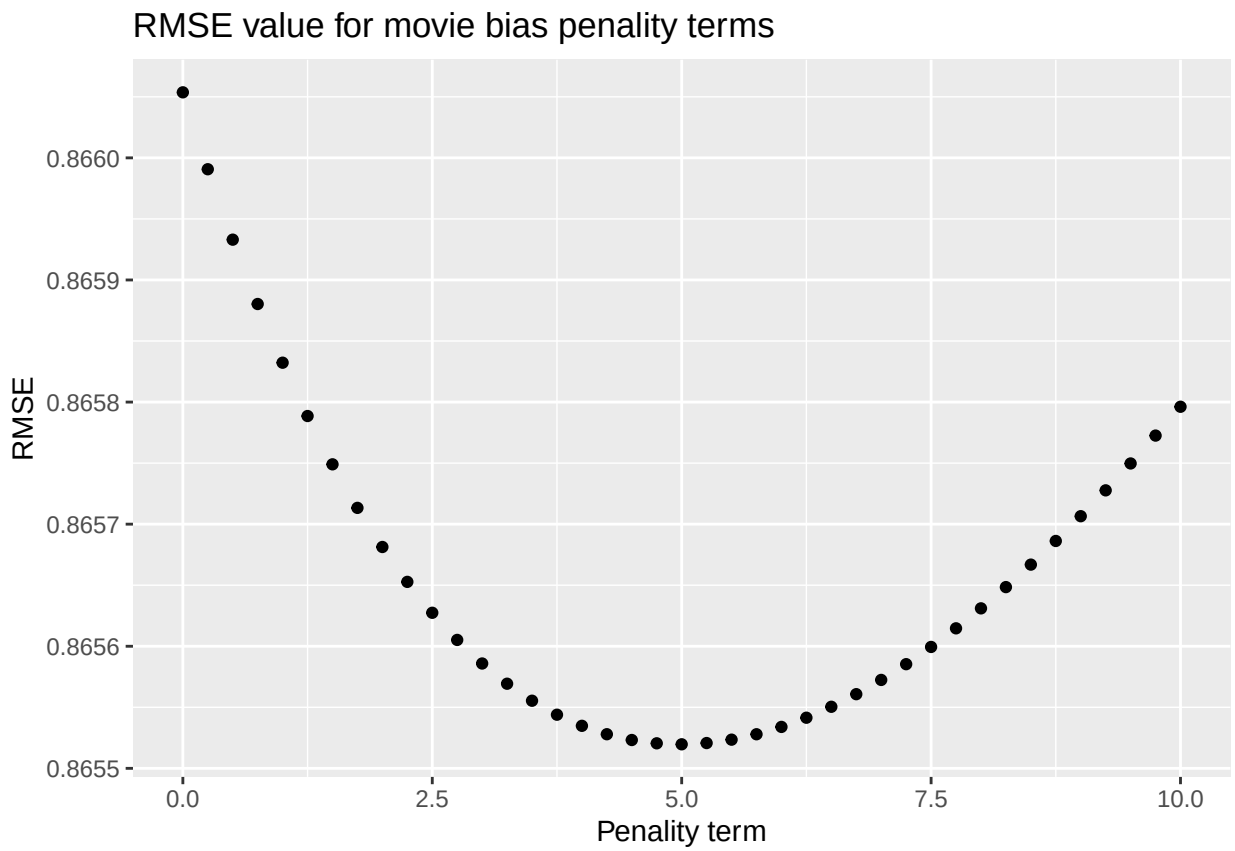



```
movie_lambda = lambdas[which.min(res$avg_rmse)]
cat("Lambda for movie regulaisration: ", movie_lambda)
```

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

The following R code calculates the RMSE value for the user bias penalty term in the given lambda range using 5-fold cross validation. The function uses the found genre and movie bias penalty value from the previous steps.

```
#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_user_lambdas)
res <- res %>% group_by(lambda) %>% summarise(avg_rmse=mean(rmse))
qplot(main=c('RMSE value for movie bias penality terms'), lambdas, res$avg_rmse) +
    xlab("Penality term") + ylab("RMSE")
```



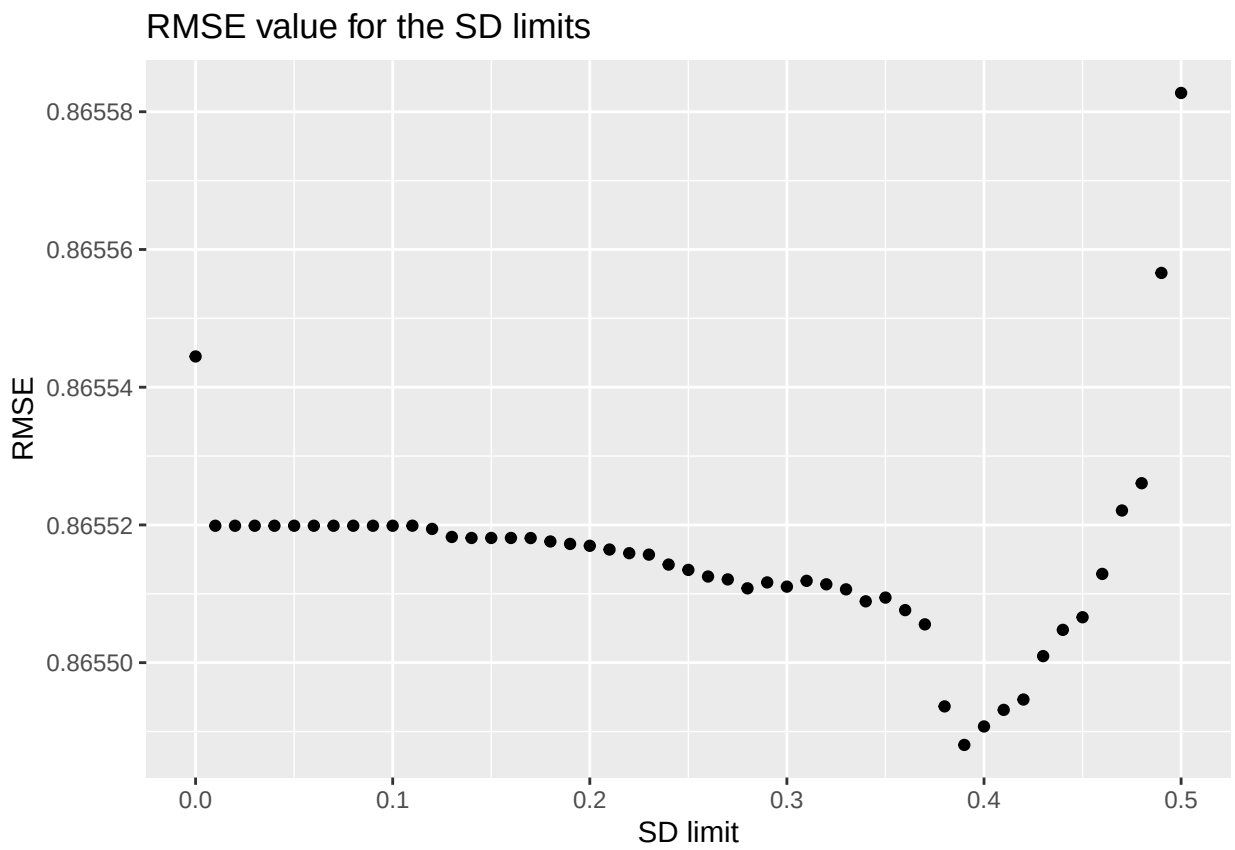

```
user_lambda = lambdas[which.min(res$avg_rmse)]
cat("Lambda for user regularization: ", user_lambda)
```

```
## Lambda for user regularization:  5
```

Our model uses the user average rating for predicting the rating for new movies for a known user if the user rating standard deviation is under a limit. To find the optimum limit for the model, we look for sd limit value in the range 0 up to 0.5 (step by 0.01). If we don't find a penalty term that minimizes the RMSE value our model, we will extend the range. We calculate he optimum value with k-fold cross validation again.

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

  RMSE_all = data.frame(r = 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(rating), avg_user=mean(rating))

  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('RMSE value for the SD limits'), sd_range, res$avg_rmse) +
    xlab("SD limit") + ylab("RMSE")
```



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

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

Now we have all penalty values for our model and can implement the prediction model. Afterwards we can predict the rating values for the validation set.

The following R code implements the model and calculates the rating predictions for the validation set data:

```
avg_rating = mean(edx$rating)

#bias based on the genres
genre_bias_reg <- edx %>% group_by(genres) %>%
    summarise(b_genre = sum(rating - avg_rating)/(n() + genres_lambda))


#bias based on the movies
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))

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(rating), sd_userrating=sd(rating))

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

#prediction based on avg, genre, movie and user bias regulated
prediction <- 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))

prediction[prediction$pred > 5,]$pred = 5
prediction[prediction$pred < 0.5,]$pred = 0.5
```

Now we can calculate the RMSE value for our implemented model on the validation set:

```
#performance of the model
cat("The model prediction RMSE value: ", RMSE(validation$rating, prediction$pred))
```

```
## The model prediction RMSE value:  0.8646461
```

Conclusion

The prediction model uses the information about genre, movie, user bias. The result shows, our model predict the movie rating with an error less as 0,8646 stars.

In the model we haven't used the information about the movie age at the rating time. This information can be useful too, because the rating of a movie can depend on the movie age at the rating time. E.g. there are evergreen movies, that have about the same rating during the time. On the other side, some movies (e.g. sci-fi or horror movies) can become less popular after some decades. This assumption can be verified with the analysis of the movie age at the rating time data.

Additionally, we used the genre combination information for creating a genre bias value for the movies and users. There are some "exotic" combinations that are only few times in the dataset. For this genre combinations we get approximately 0 bias because of the regularization in the algorithm. We can analyses if there is any correlation between the different genre combination. Perhaps there are some major genre combinations that can give us good estimation for exotic genre combination. For this analysis we can use clustering techniques.

As we see, the prediction model can be extended with some additional information that would increase the accuracy the prediction model.