HarvardX PH125.9xData Science: Capstone: Movielens recommendation system

Yannick Hermans

1-10-2020

Introduction

In 2006 Netflix launched a competition where participants were asked to optimize Netflix's movie recommendation system with at least 10 %. The winners were promised a prize worth $1M. The challenge was to predict user ratings as accurately as possible based on a dataset containing previous user ratings. There was no additional knowledge on the users or movies. In 2009 BellKor's Pragmatic Chaos team was able to best Netflix's movie recommendation algorithm by 10.06%

In this capstone project, which forms the conclusion to the HarvardX PH125.9xData Science course, the goal is to build towards the recommendation algorithm that the winners of the Netflix challenge developed. This project will represent the knowledge I gained throughout the edX data science course on machine learning algorithms, data processing and data exploration. A reduced version of the open source MovieLens data set with 10M ratings, from 10,000 users on 72,000 movies, has been used for this data science project.

This report contains the data processing steps, data exploratory analysis and a few machine learning algorithms.

Data preprocessing

First the necessary packages are installed and loaded. The 10M MovieLens dataset is loaded and separated into a training ("edx") and a validation "validation" set. The code for these preprocessing steps is based on the already provided code in the edx capstone project module. https://courses.edx.org/courses/course-v1:HarvardX+PH125.9x+2T2018/courseware/dd9a048b16ca477a8f0aaf1d888f0734/e8800e37aa444297a3a2f35bf84ce452/?child=first

```
############################################################
# Create edx set, validation set (final hold_out test set)
############################################################

# Note: this process could take a couple of minutes

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

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

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# 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(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will contain 10% of the MovieLens data set
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)
```

The validation set is used to determine the accuracy of the final movie recommendation algorithm. The other models are tested by splitting up the edx set into a training (edx_training_set) and a test set (edx_test_set).

```
# Split edx data in training (80%) and test set (20%)
set.seed(1992, sample.kind = "Rounding")
edx_test_index <- createDataPartition(y = edx$rating, times = 1,
                                      p = 0.2, list = FALSE)
edx_train_set <- edx[-edx_test_index,]
edx_test_set <- edx[edx_test_index,]

edx_test_set <- edx_test_set %>%
  semi_join(edx_train_set, by = "movieId") %>%
  semi_join(edx_train_set, by = "userId")
```

In addition, certain libraries for the data exploration and machine learning development are loaded

```
library(lubridate)
library(ggplot2)
library(recosystem)
library(knitr)
library(gridExtra)
```

Data exploratory analysis

Overview

First an overview of the data is given.

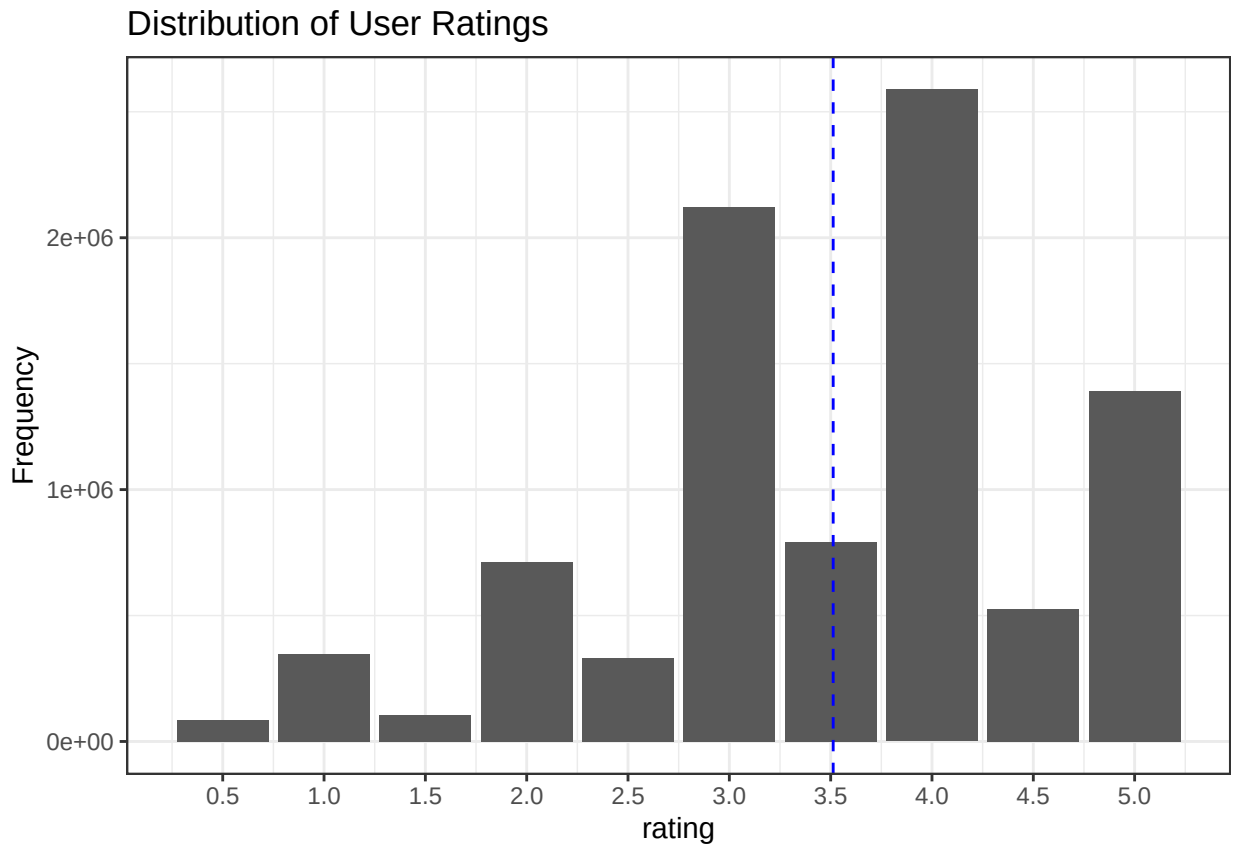
```
kable(head(edx))
```

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

```
# the number of observations in the Movielens data set
length(edx$rating)+length(validation$rating)
```

[1] 10000054

The data contains 10000054 observations and is already in a tidy format, which allows for straightforward data exploration in r. In addition to a rating, additional information can be found in the dataset such as a userID, a movieID, the time at which the rating was given and the genres to which the rated movie belongs. Before analyzing how these variables relate to the rating, an overview of the rating distribution is shown below.

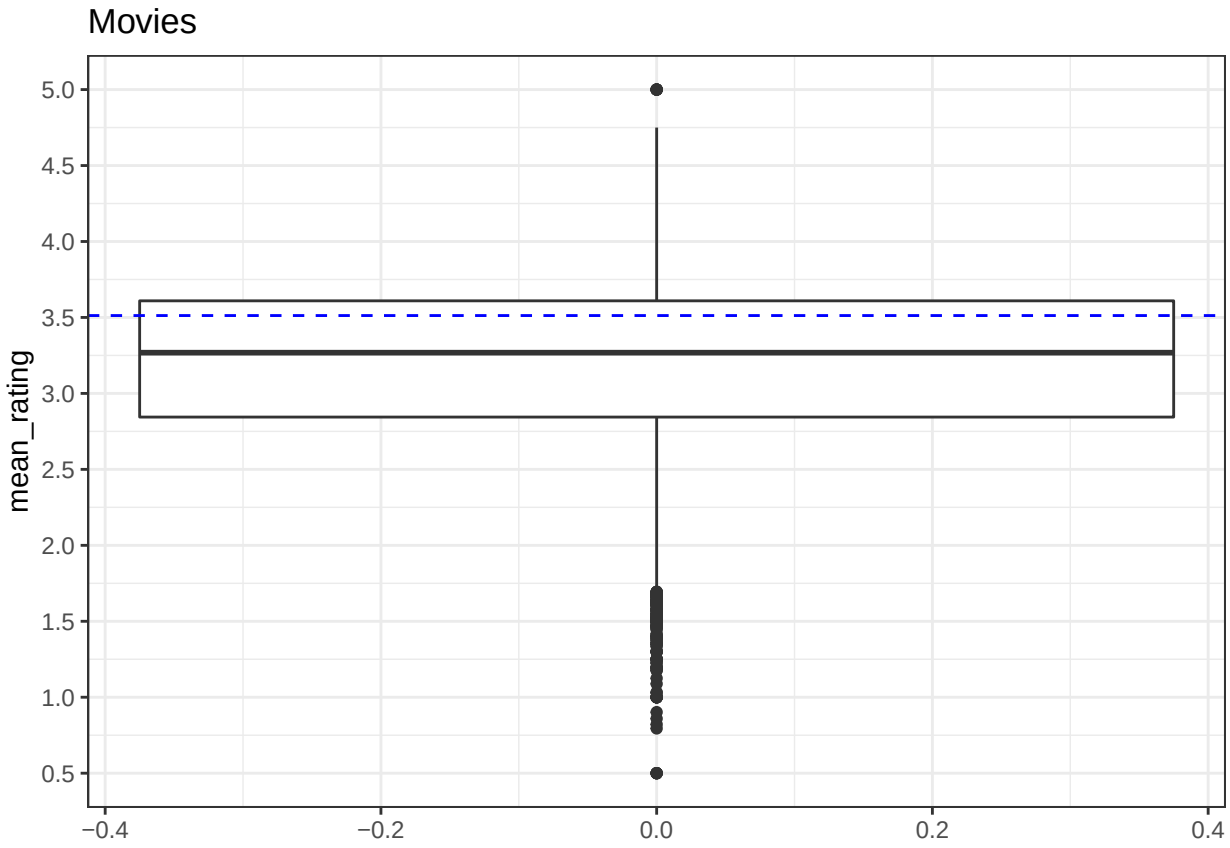


The user ratings' distribution demonstrates that users mainly give a score of 3.0 or 4.0. In addition it is clear

that users are less likely to give half point scores than whole point scores. In average users gave a score of 3.5 which is indicated by the blue dashed line.

For the development of a recommendation algorithm it would be informative to know how the additional variables in the 10M Movielens dataset relate to the ratings. This will be explored in the following.

Relation MovieId(Moviename) to user rating

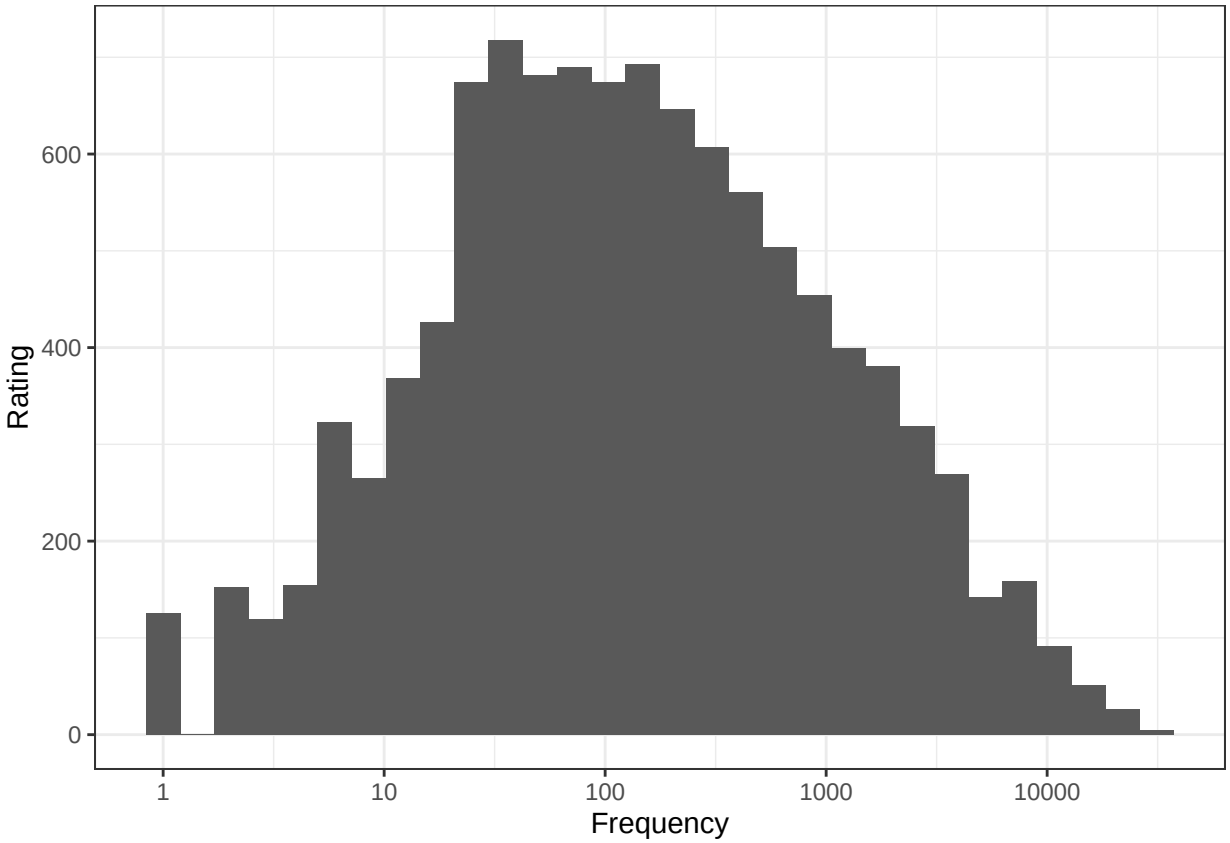


The boxplot above shows the variability of the rating averages for each movie in the edx dataset. On first sight it seems that the movieId has a strong influence on the rating, but let's have a look at the 10 best and the 10 worst rated movies.

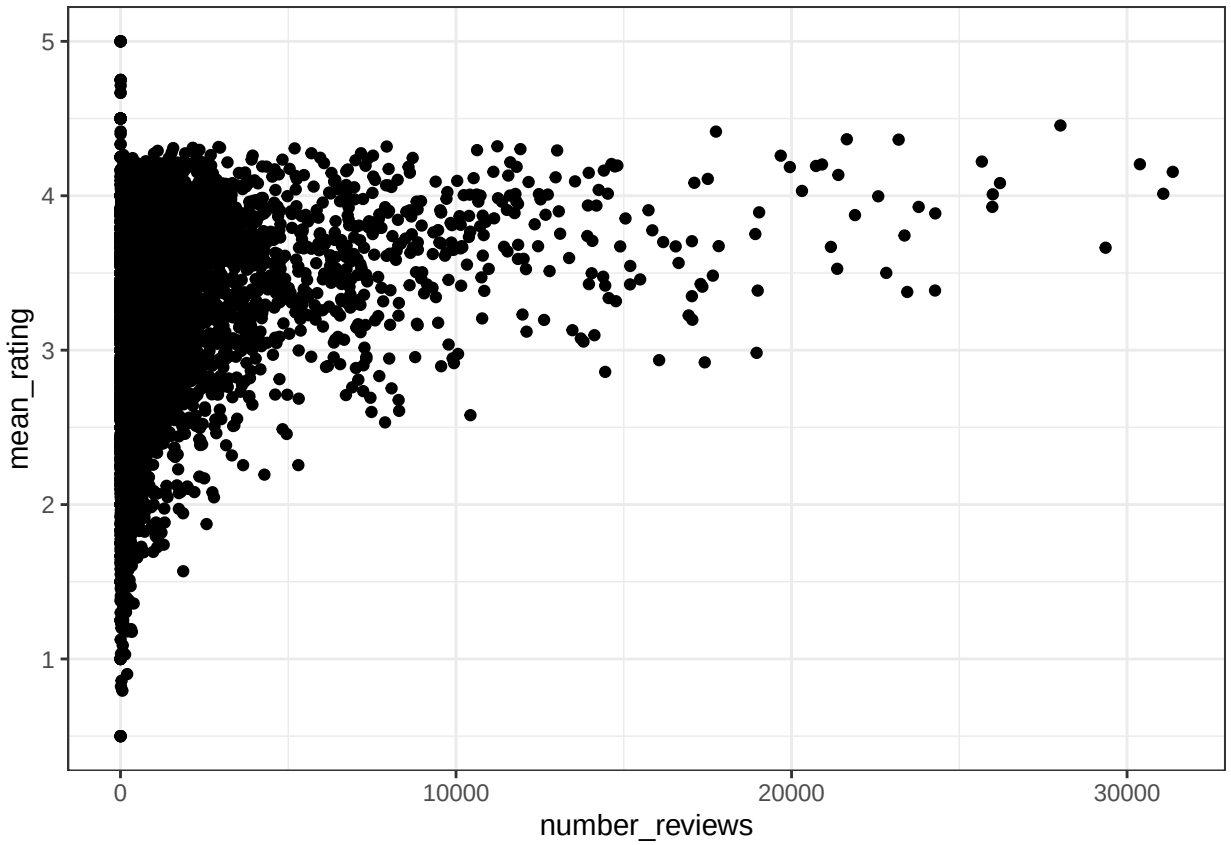
title	movieId	mean_rating
Blue Light, The (Das Blaue Licht) (1932)	64275	5.00
Fighting Elegy (Kenka erejii) (1966)	51209	5.00
Hellhounds on My Trail (1999)	3226	5.00
Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)	33264	5.00
Shadows of Forgotten Ancestors (1964)	42783	5.00
Sun Alley (Sonnenallee) (1999)	53355	5.00
Constantine's Sword (2007)	65001	4.75
Human Condition II, The (Ningen no joken II) (1959)	26048	4.75
Human Condition III, The (Ningen no joken III) (1961)	26073	4.75
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	5194	4.75

title	movieId	mean_rating
Accused (Anklaget) (2005)	61768	0.5000000
Besotted (2001)	5805	0.5000000
Confessions of a Superhero (2007)	63828	0.5000000
Hi-Line, The (1999)	8394	0.5000000
War of the Worlds 2: The Next Wave (2008)	64999	0.5000000
SuperBabies: Baby Geniuses 2 (2004)	8859	0.7946429
Hip Hop Witch, Da (2000)	7282	0.8214286
Disaster Movie (2008)	61348	0.8593750
From Justin to Kelly (2003)	6483	0.9020101
Criminals (1996)	604	1.0000000

The top 10 and bottom 10 movies are,, however, rather obscure. Furthermore, many of these movies have scores which are multiples of 0.5, indicating that only few people have rated those movies. Thus, the ratings already given for these movies will not be very predictive for how other users may rate these movies.



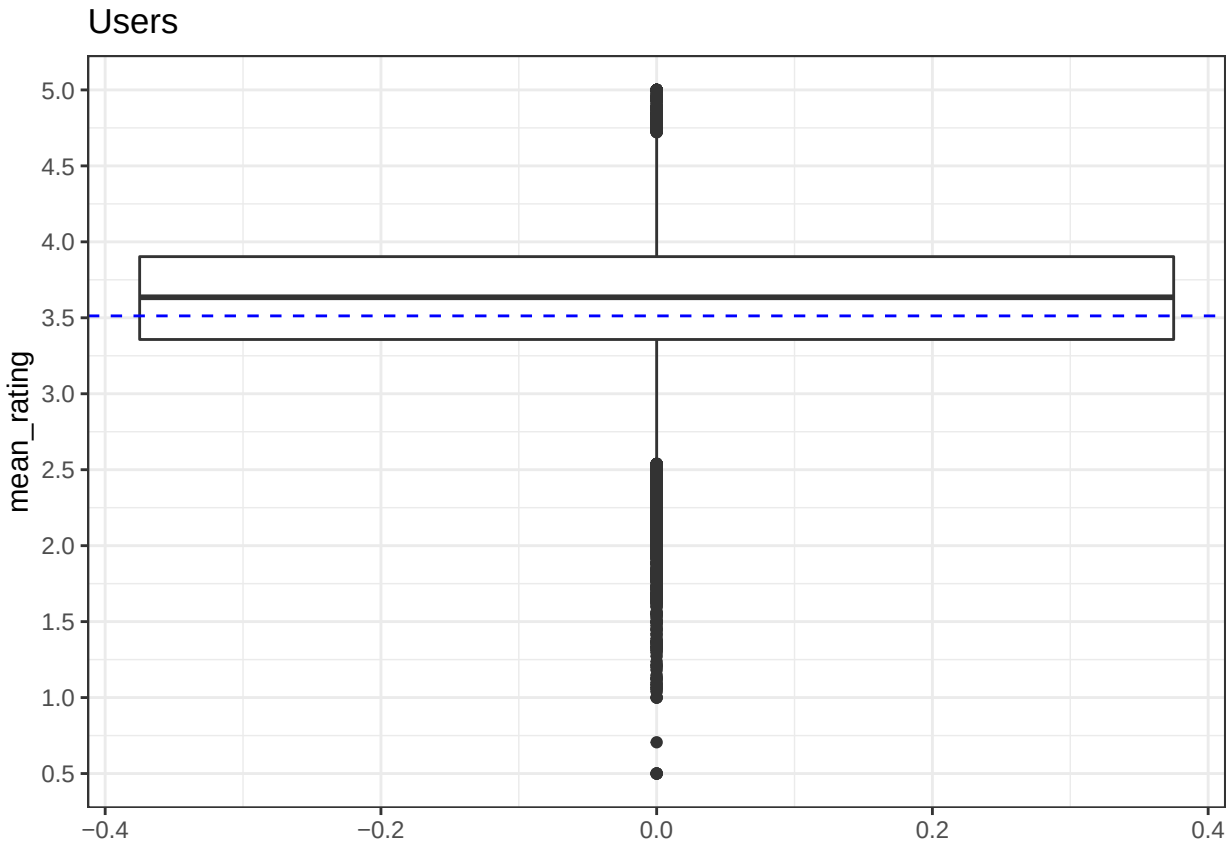
Indeed, the histogram above shows that only few movies have more than 1000 ratings and many have less than 100 reviews.



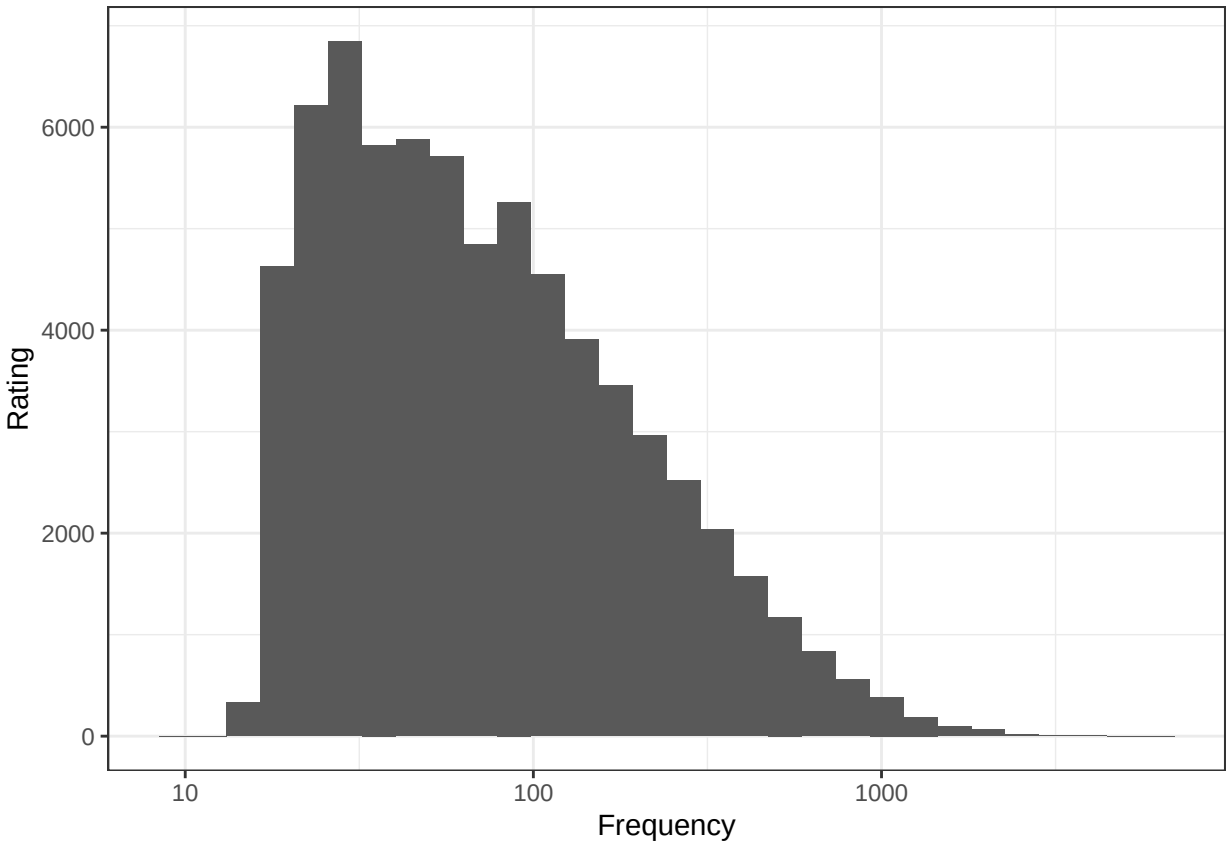
How the average ratings per movie depend on the number of reviews is even better visualized in the scatter plot above. It shows that movies which have been rated often, generally have a higher mean rating. Additionally, the lower the number of ratings, the stronger the variability of the rating. After a certain number of ratings, the average rating of a movie, is likely a strong predictor for what an unknown user would rate this movie.

Relation UserId(Users) to user rating

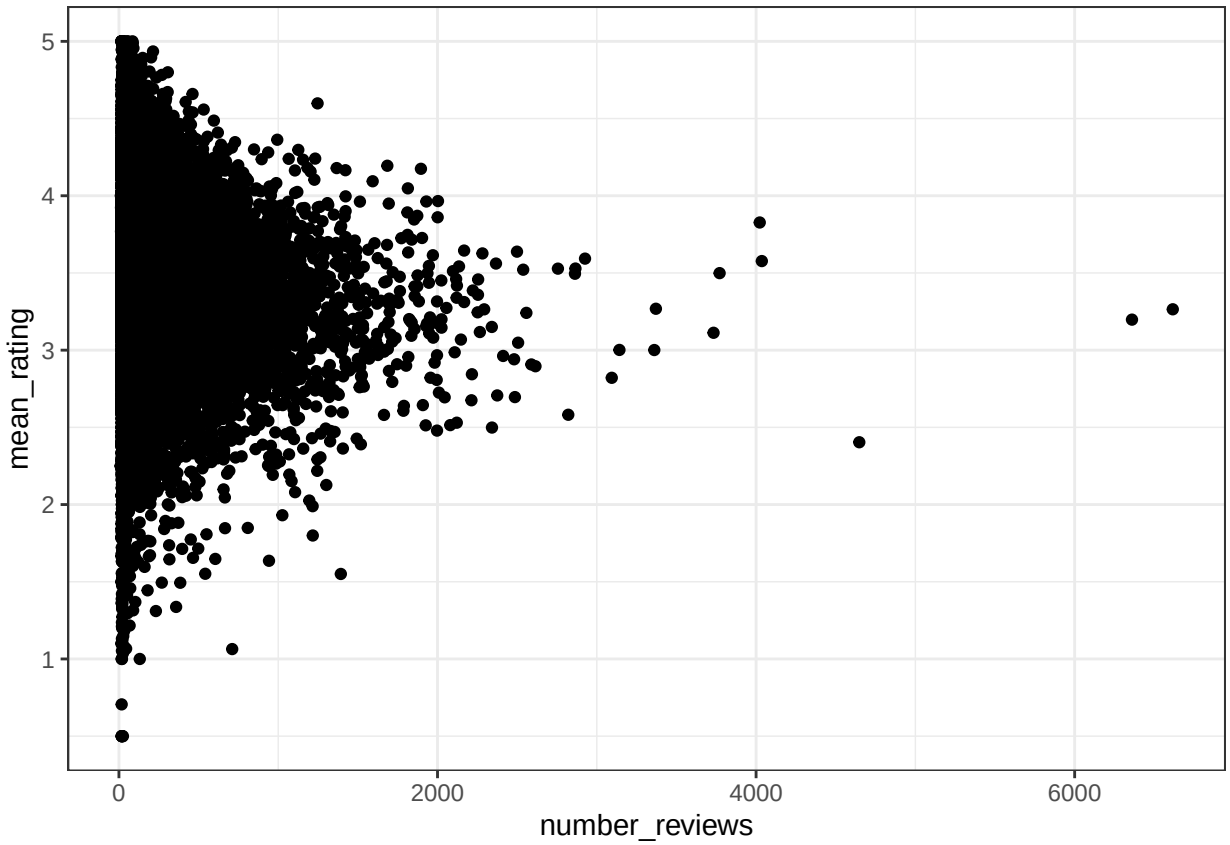
Now the relation between the userId and the rating will be shown.



The boxplot above resembles the one for the movie - average rating relationship. The variability here is slightly lower though, since the interquartile range is narrower. However, there are still many outliers in the lower and higher rating range. These outliers can be due to users rating every movie highly/lowly or it could be users which do not have rated many movies yet.



The histogram for the number of ratings per user indeed shows that most users have rated less than 100 movies. Their rating will be off course less predictive than for users who predicted more than 1000 movies.



The relationship between a user's average rating and the number of movies the user has rated is shown above. The variability in the average rating indeed goes down with the number of reviews a user has posted. It is, however, clear that some users give generally higher scores than others, making the UserId a good predictor for the rating.

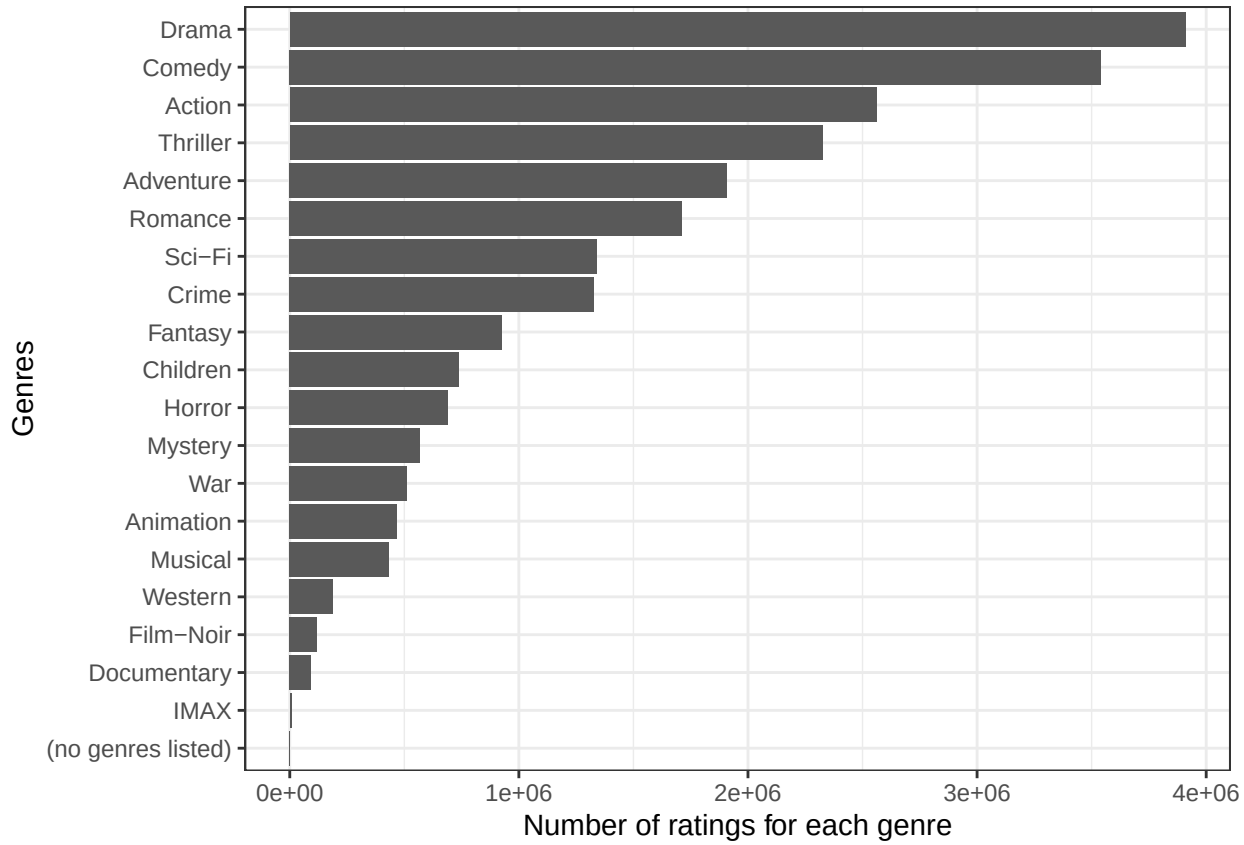
Relation Genres to user rating

Another variable which is contained in the 10M movielens dataset are the genres to which a movie belongs. The genre information is, however, compacted into one string per movie. To allow for further investigation each individual genre has to be extracted from the string

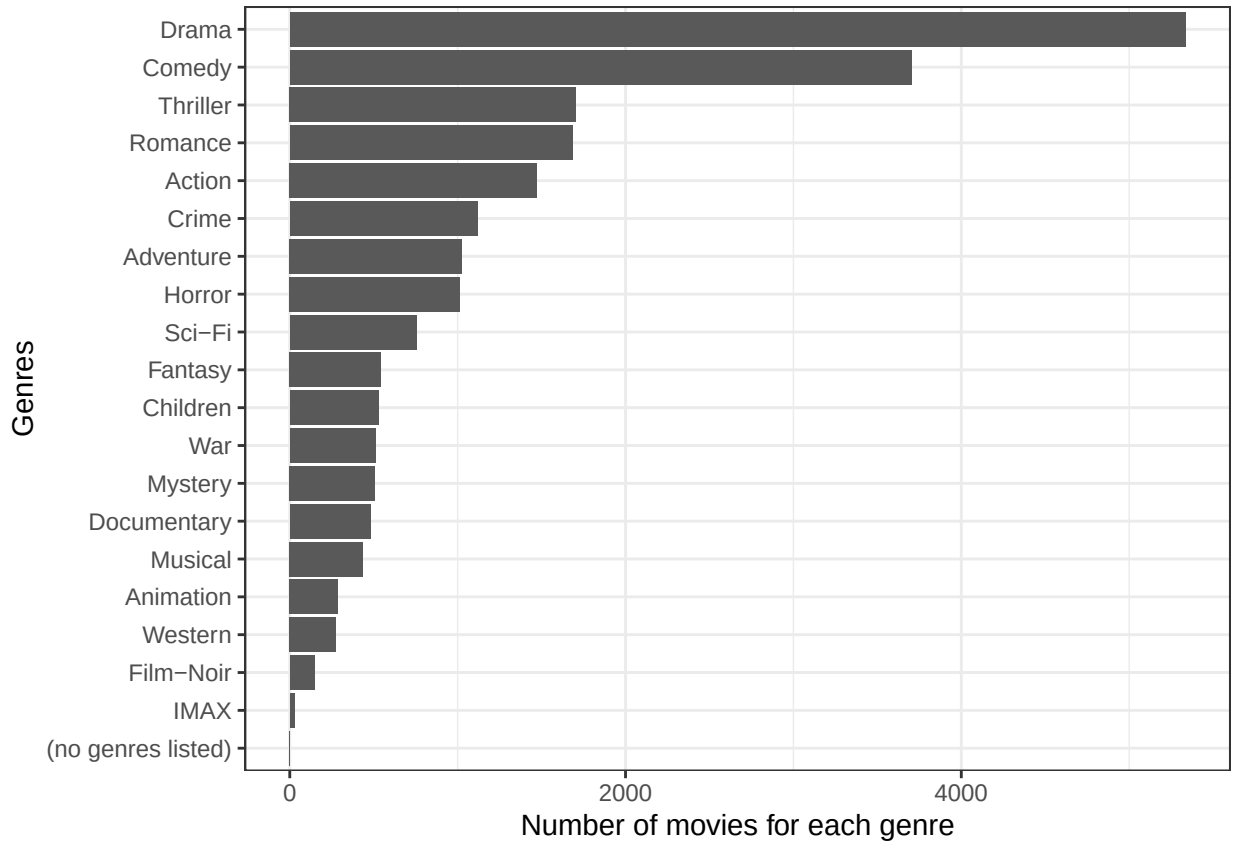
```
# number of ratings for each genre
edx_split_genres  <- edx  %>% separate_rows(genres, sep = "\\|")
kable(head(edx_split_genres))
```

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy
1	122	5	838985046	Boomerang (1992)	Romance
1	185	5	838983525	Net, The (1995)	Action
1	185	5	838983525	Net, The (1995)	Crime
1	185	5	838983525	Net, The (1995)	Thriller
1	292	5	838983421	Outbreak (1995)	Action

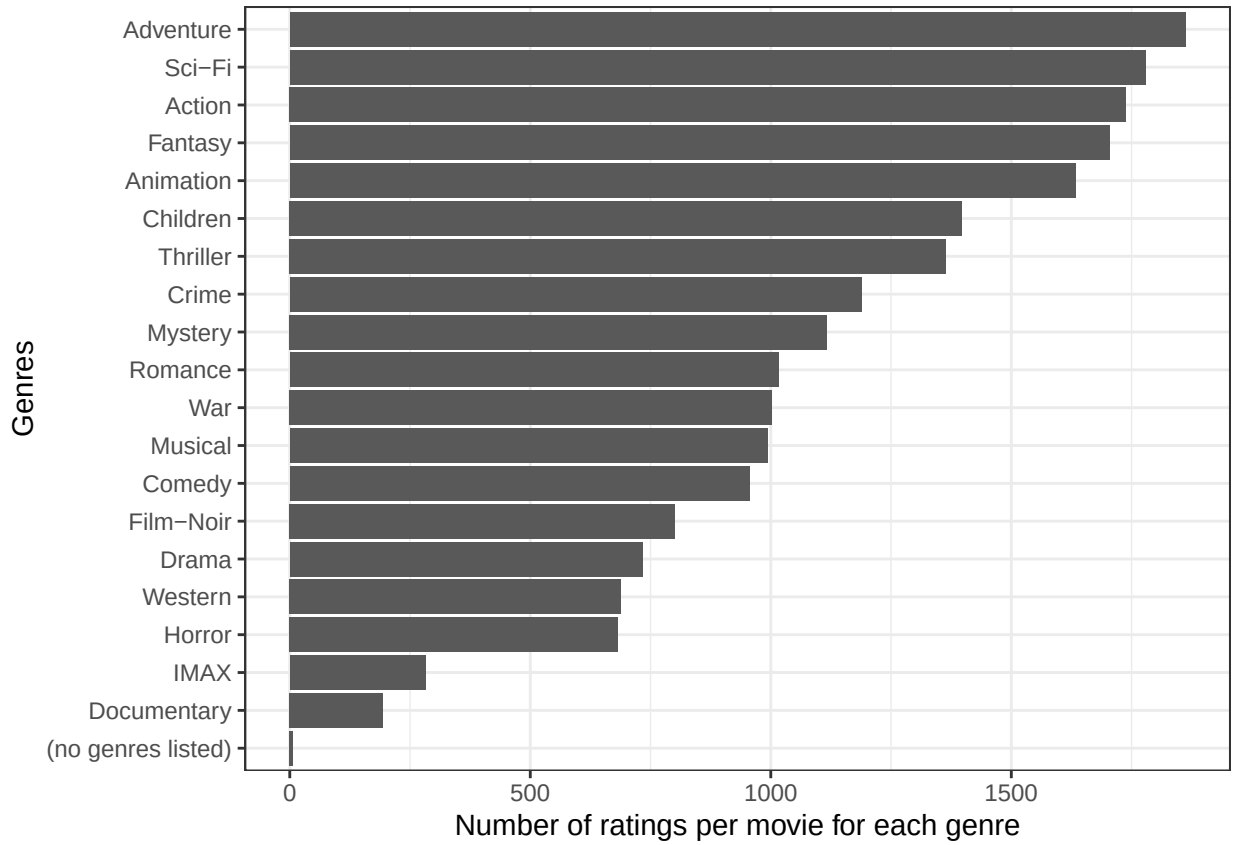
Now the rows only contain one genre.



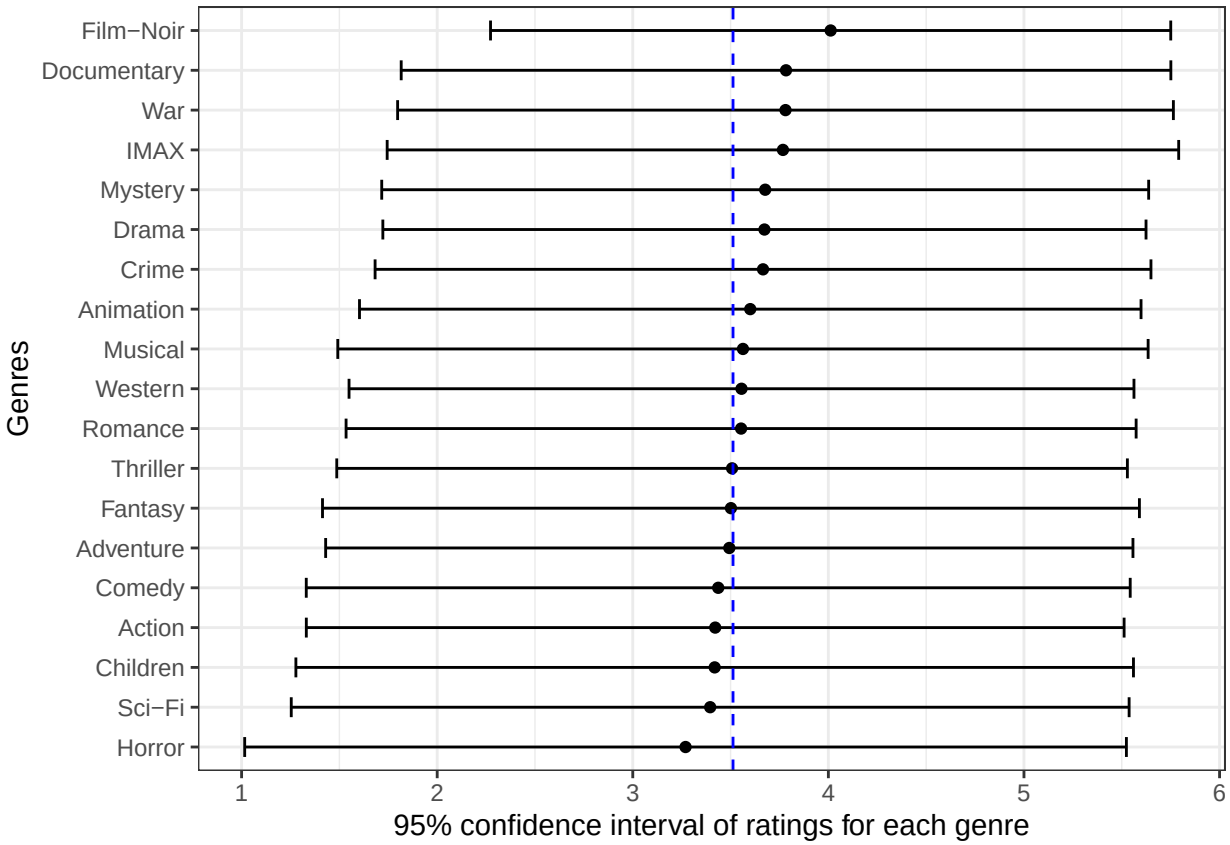
From the above plot one can see that particular genres were rated more often than others. This can be due to some genres being more popular than others or due to more movies having been made for that particular genre.



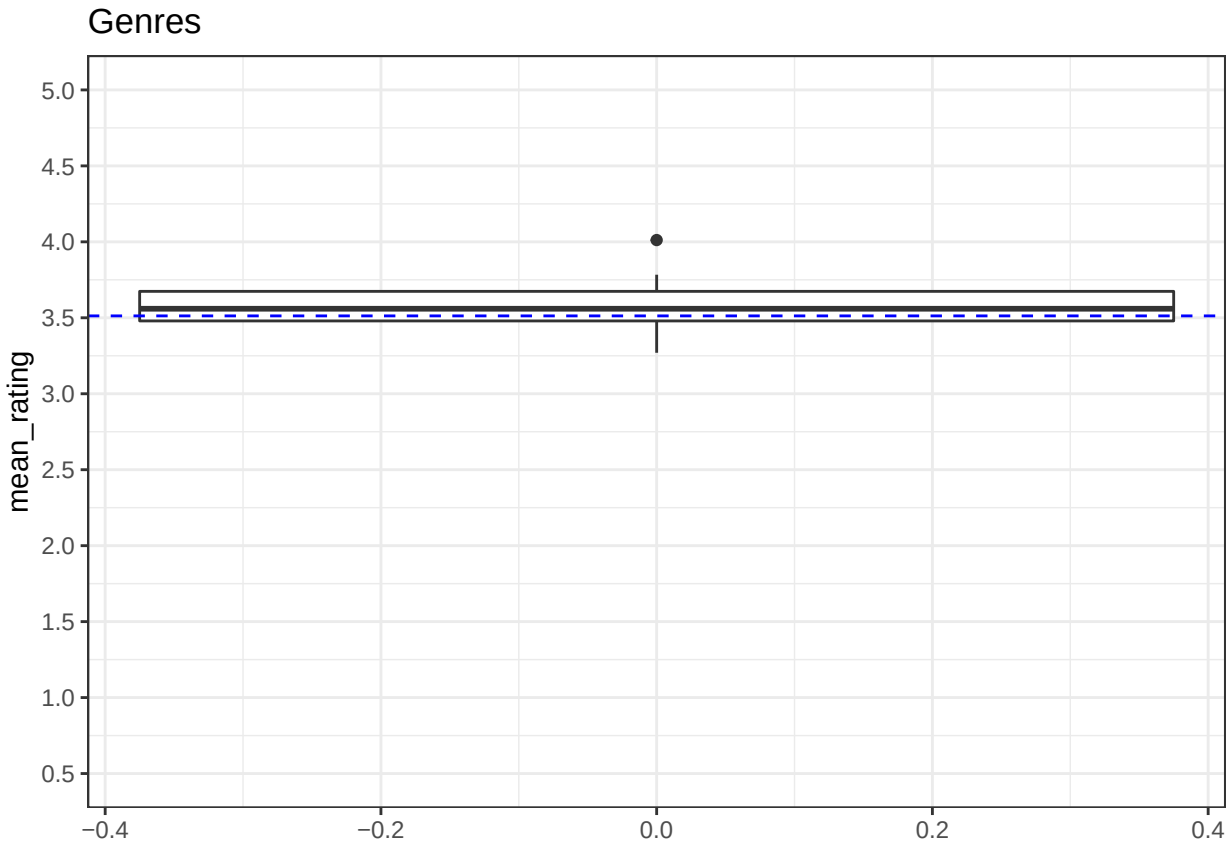
The barplot above depicting the number of movies for each genre is very similar to the barplot for the number of ratings for each genre. This demonstrates that the number of reviews for a genre depends in large on how many movies have been made for that genre. To see which genres users like to rate more often, the average number of reviews per movie for each genre is shown below.



It can be seen that users like to rate Adventure and Sci-Fi movies more often than documentaries and IMAX movies. This could be an indication that some genres are more popular than others.



However, the plot above showing the average rating and the 95% confidence interval per genre is quite different from the previous bar plot. Thus, movies which users like to rate do not automatically yield higher average ratings. In addition, the average ratings per genre do not differ strongly from the overall rating average (blue dashed line). Thus, the genre to which a movie belongs does not influence the rating much. This is also clearly visualized in the boxplot below.



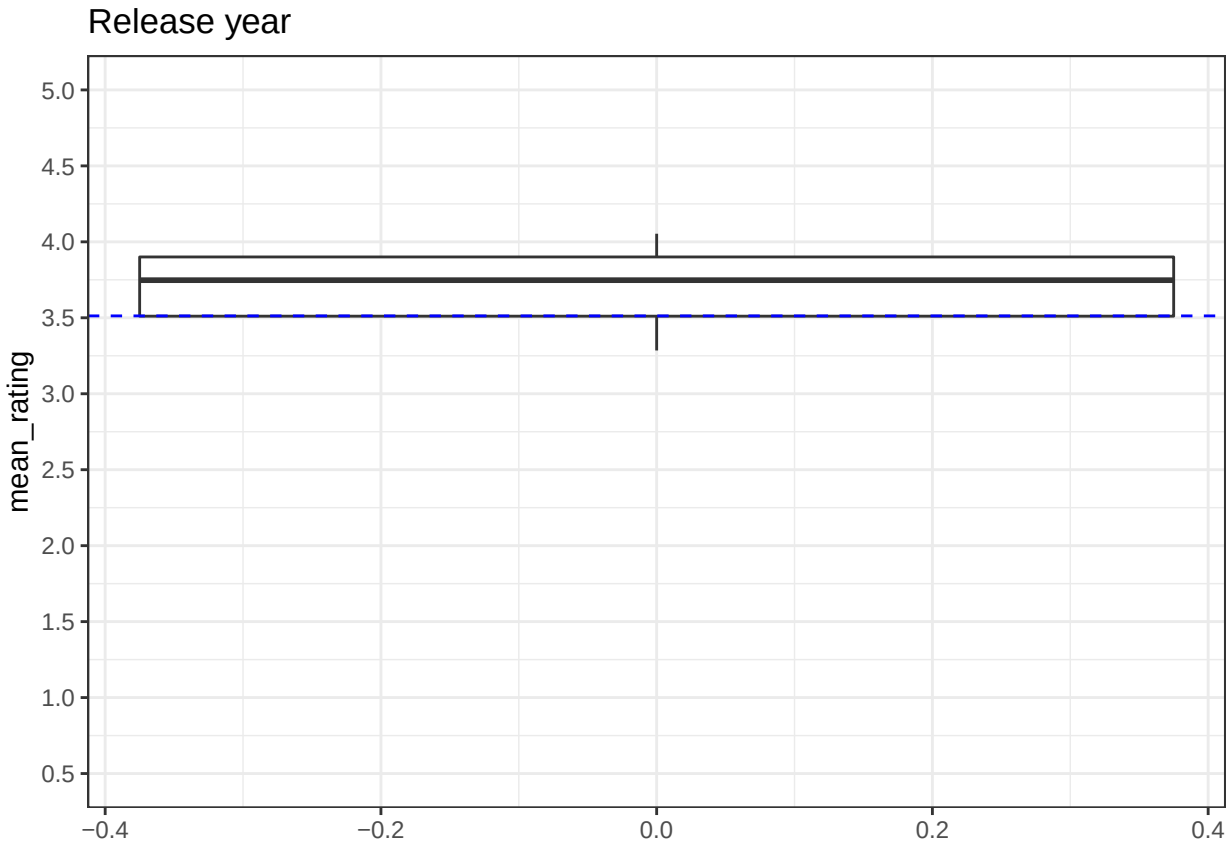
Relation time of release and age at rating to user rating

The last features in the dataset are the date of release of the movie (mentioned in the title) and the timestamp of when the movie was rated. The timestamp first has to be transformed into a year of rating and the year of release has to be extracted from the title:

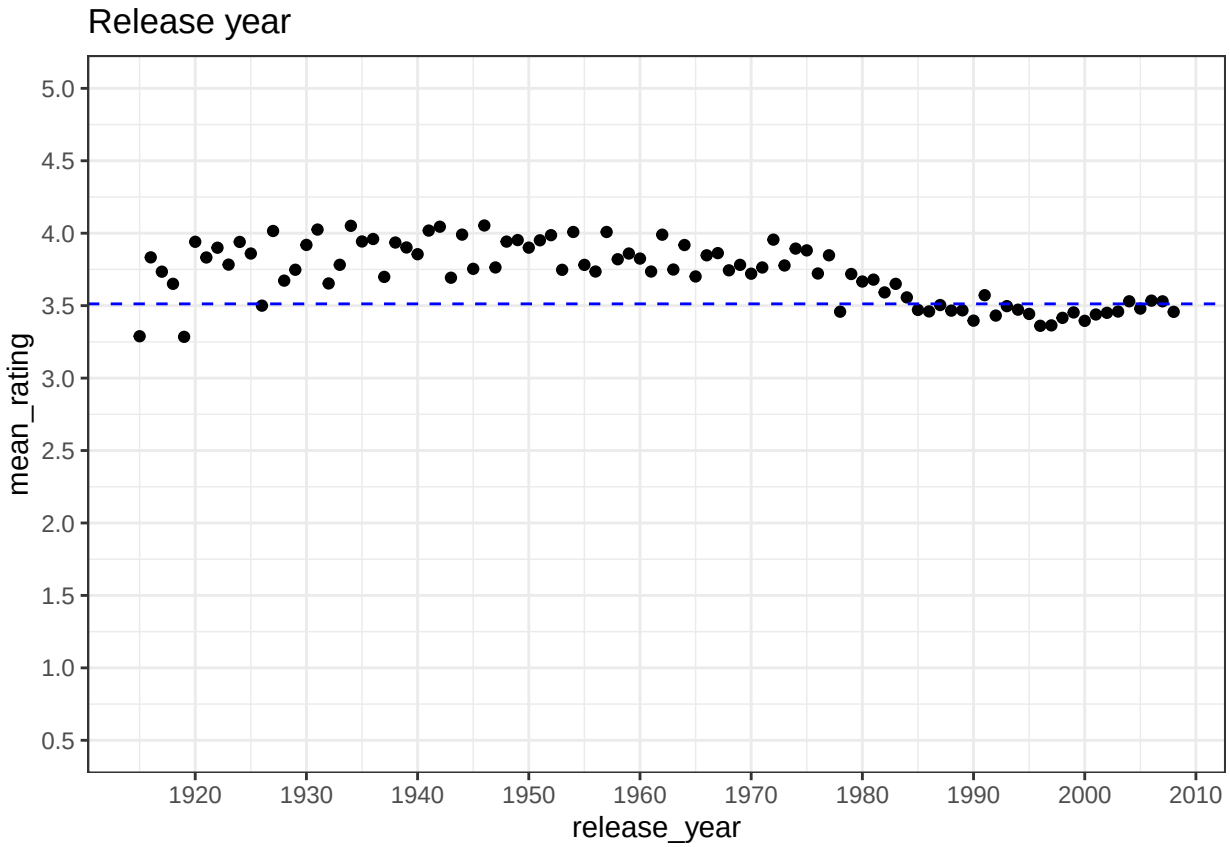
```
edx_year <- edx %>% mutate(rate_year = year(as_datetime(timestamp)))
edx_year <- edx_year %>% mutate(title = str_replace(title,"^(.+)\\s\\((\\d{4})\\)$","\\1__\\2" )) %>%
  separate(title,c("title","release_year"),"__") %>%
  select(-timestamp)
edx_year <- edx_year %>% mutate(movieage_at_rating= as.numeric(rate_year)-as.numeric(release_year), rel
kable(head(edx_year))
```

userId	movieId	rating	title	release_year	genres	rate_year	movieag
1	122	5	Boomerang	1992	Comedy|Romance	1996	
1	185	5	Net, The	1995	Action|Crime|Thriller	1996	
1	292	5	Outbreak	1995	Action|Drama|Sci-Fi|Thriller	1996	
1	316	5	Stargate	1994	Action|Adventure|Sci-Fi	1996	
1	329	5	Star Trek: Generations	1994	Action|Adventure|Drama|Sci-Fi	1996	
1	355	5	Flintstones, The	1994	Children|Comedy|Fantasy	1996	

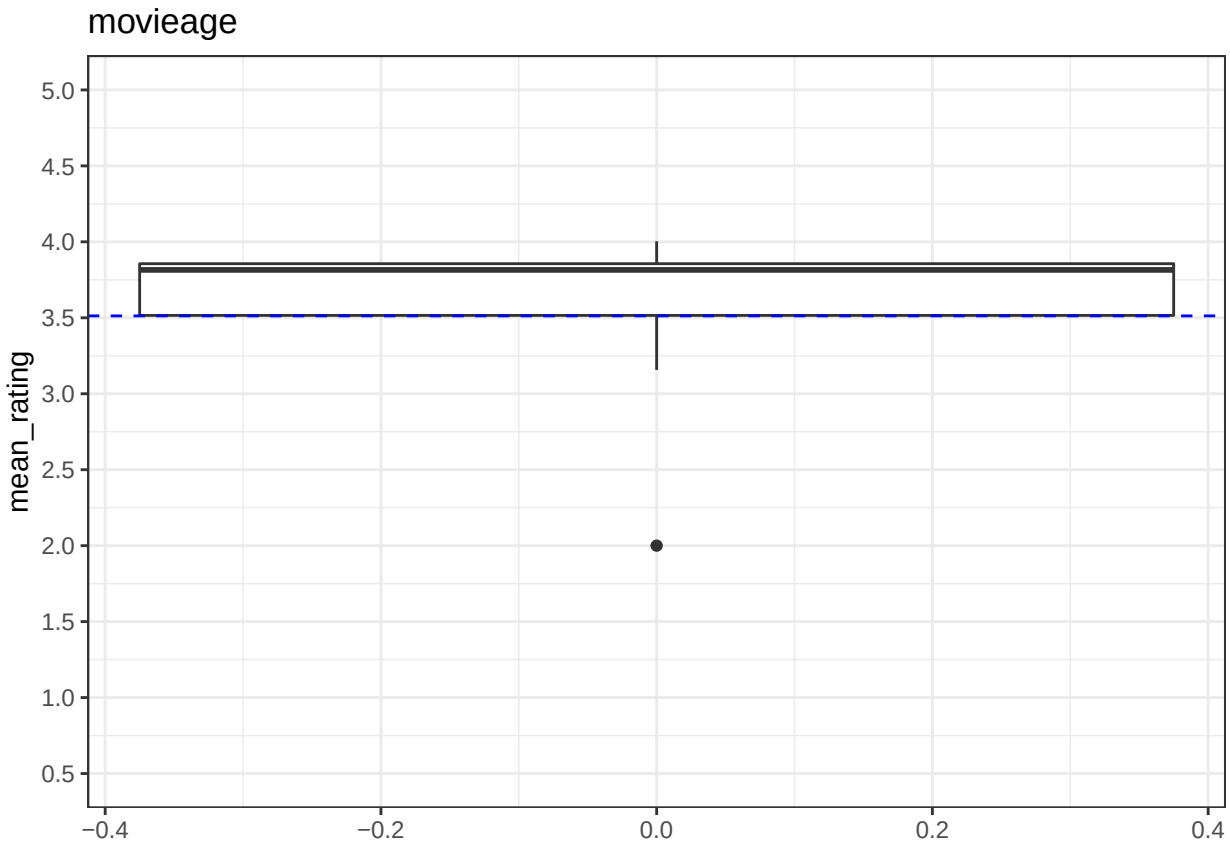
Now the relation of the rating to the year of release and year of rating can be visualized



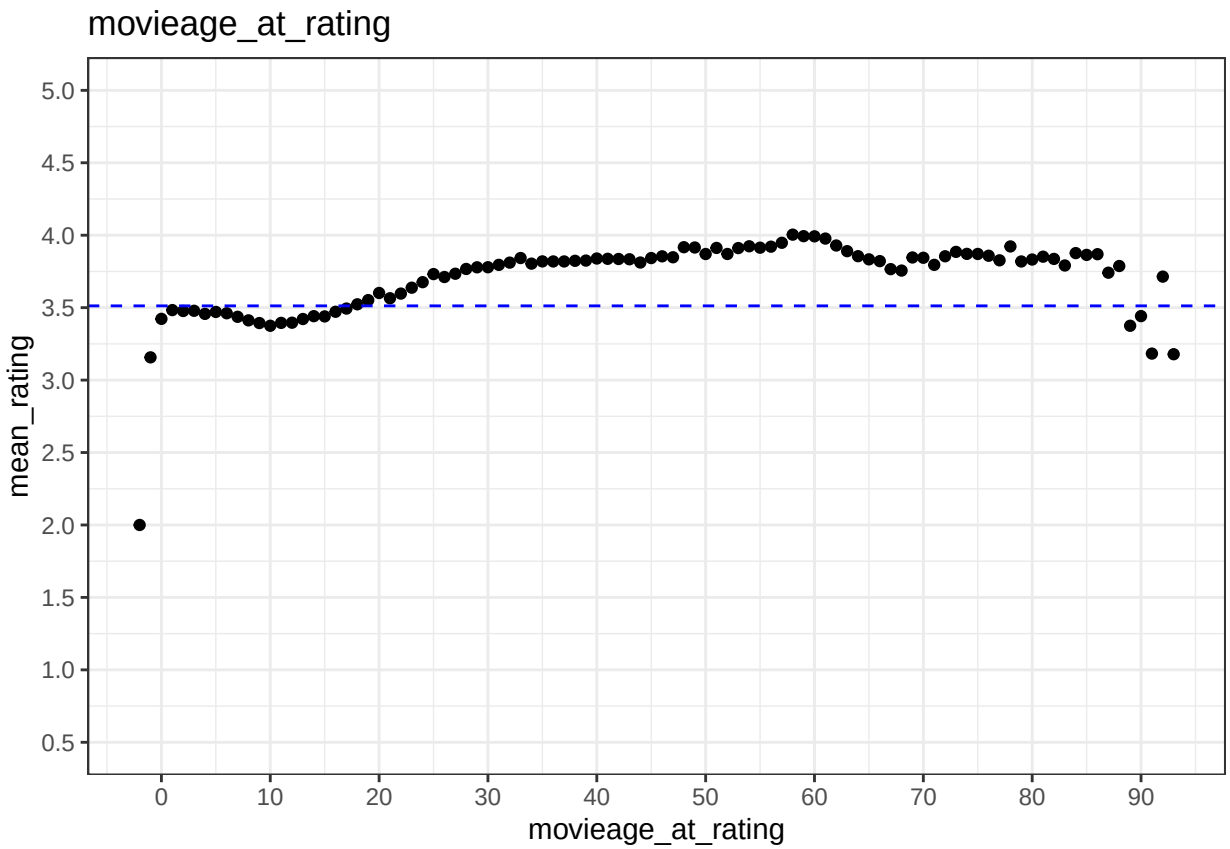
The above boxplot indicates that the release year cause more variability in the rating than the movie genres do. Thus, the release year could be an indicator for the rating.



The lineplot shows more clearly the relationship between the average rating for every release year and the release year itself. Movies before 1980 have been rated higher in average than movies after 1980.



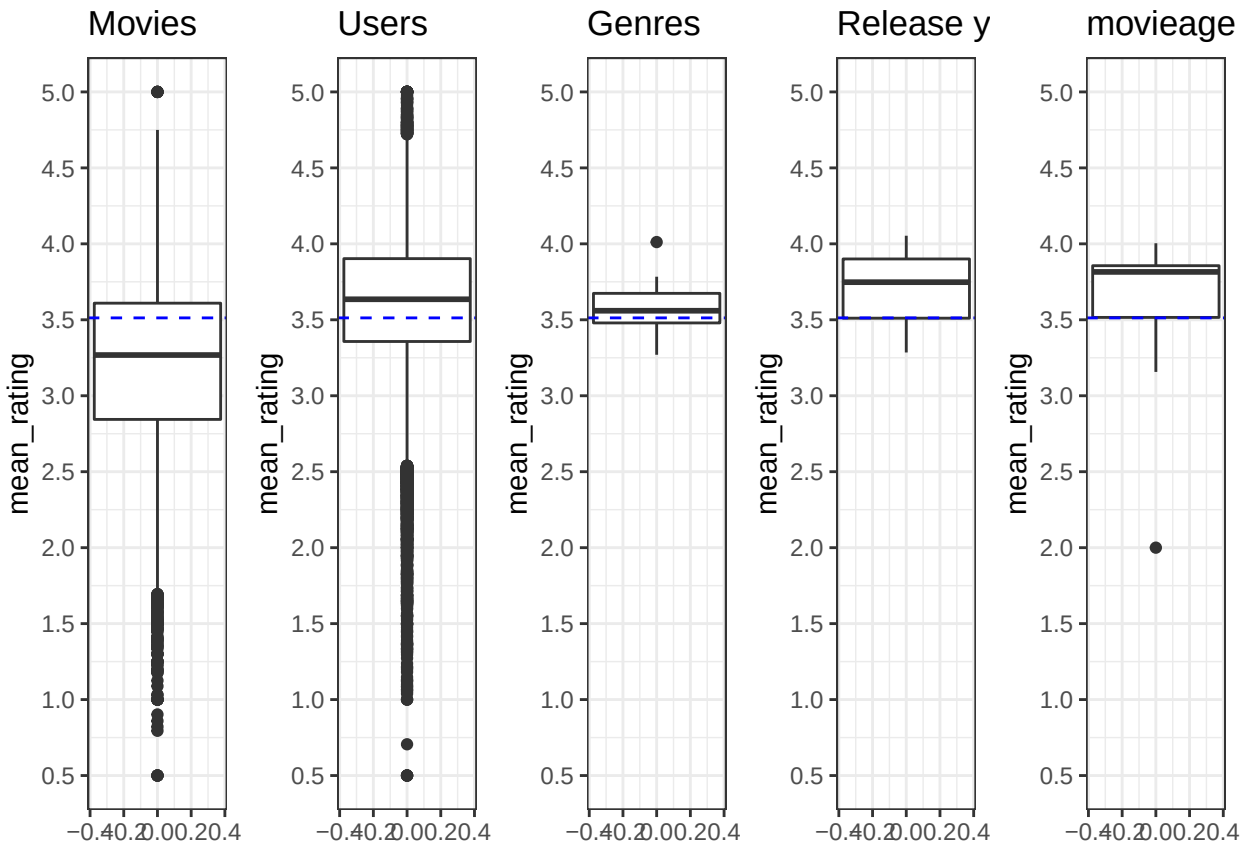
Also the age of a movie influences how users rate that movie. The scatter plot underneath shows the relationship.



The plot shows that older movies generally get higher ratings than younger movies.

Summary

The above data exploration of the edx dataset fraction of the 10M movielens dataset has shown that certain features in the dataset have an influence on the rating. Thus, an algorithm which takes the relationships between the ratings and the other features into account should predict unknown ratings better in comparison to using the overall rating average as the only predictor.



The above plot summarizes the 10M Movielens data exploration. It shows that the rating variability is influenced much stronger by the UserId and the MovieId than by genre, movieage or release year. Therefore, the UserId and MovieId should certainly be taken into account when developing a recommendation algorithm.

An important side note has to be made to the above data analysis. Namely that only relationships between the ratings and the features explicitly mentioned in the dataset have been investigated. It is, however, likely that other unknowns features also have a strong influence on the ratings.

Recommendation algorithm development

Strategy

The data exploration has shown that UserId and MovieId have a strong influence on the rating. In contrast, movieage at rating, release year and the genre have a smaller effect on the rating variability. Thus, in the first place a baseline algorithm model will be constructed, which takes the movie and user effect into account. In this baseline model the movie and user effect will be regularized to ensure that users and movies with few ratings do not have a too strong contribution in the model. The first few steps in the development of the baseline model are based on the instructions in the "Recommendation Systems" chapter of the text book (https://rafalab.github.io/dsbook/large-datasets.html#recommendation-systems).

The residuals remaining after the baseline model will be the basis for matrix factorization. This method has been used by the winners of the Netflix challenge to identify hidden patterns between movies and users. In general, matrix factorization tries to separate the rating matrix into a user embedded matrix and a movie embedded matrix. Using a training set, the embeddings of the user embedded and movie embedded matrices are learned to best represent the rating matrix again. In this way, effects such as movieage at rating, release year and genre information, as well as many other unknown features are included in the recommender model if not already implicitly taken into account in the baseline model. A more thorough description of matrix factorization can be found here (https://developers.google.com/machine-learning/recommendation/collaborative/matrix).

There are a few r packages which can perform matrix factorization, but here the "recosystem" package (https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html) was chosen due to its ease of use, reduced memory usage and the relatively low computational power it requires. The r recosystem package is a wrapper for the high-performance C++ LIMBF library which uses the parallel stochastic gradient descent algorithm for optimizing the user embedded matrix and the movie embedded matrix.

The edx dataset has been split up in a training set (edx_train_set) and a test set (edx_test_set). The training set will be used to train the models during the recommendation algorithm development, while the test set will test these developed models. The testing will be performed by taking the root-mean-square error (RMSE) between the predicted ratings and the true ratings for the test set. The lower the RMSE, the better the recommender algorithm.

```
# RMSE function
RMSE <- function(ratings_true, ratings_predicted){
  sqrt(mean((ratings_true - ratings_predicted)^2))
}
```

Model 1: average ratings for each movie

As a first model just the ratings themselves are taken into account. The RMSE can then be minimized by using the overall average of all ratings (3.51248), which yield an RMSE of 1.0606.

```
# Model 1: average rating
# average rating of all movies for all users to predict the rating for user u and movie i
model_1_prediction <- mean(edx_train_set$rating)
model_1_prediction
```

```
## [1] 3.51248
```

```
model_1_RMSE <- RMSE(edx_test_set$rating, model_1_prediction)
model_1_RMSE
```

```
## [1] 1.060635
```

```
all_RMSEs <- data_frame(method = "Model 1 : overall average", RMSE = model_1_RMSE)
all_RMSEs %>% knitr::kable()
```

method	RMSE
Model 1 : overall average	1.060635

Model 2: modeling the movie effect by adding a movie bias (m_bias)

The characteristics of a movie have a certain influence on the ratings of that movie. In the recommendation model the movie effect can be taken into account by adding a bias for each movie (m_bias). This bias corresponds to the difference between the average rating of that specific movie to the overall average rating of all movies.

```
# Model 2: average rating + movie bias
# inclusion of a movie bias (m_bias) since movies can in average be rated higher or lower than the over
avg_rating <- mean(edx_train_set$rating)

movie_avgs <- edx_train_set %>%
  group_by(movieId) %>%
  summarize(m_bias = mean(rating - avg_rating))

model_2_prediction <- avg_rating + edx_test_set %>%
  left_join(movie_avgs, by='movieId') %>%
```

```
  .$m_bias

model_2_RMSE <- RMSE(model_2_prediction, edx_test_set$rating)
all_RMSEs <- bind_rows(all_RMSEs,
                           data_frame(method="Model 2: movie bias",
                                      RMSE = model_2_RMSE))
all_RMSEs %>% knitr::kable()
```

method	RMSE
Model 1 : overall average	1.0606351
Model 2: movie bias	0.9440709

Model 3: modeling the user effect by adding a user bias in addition to the movie bias (m_bias + u_bias)

The characteristics of a user can also influence the rating of a particular movie. There are for example optimistic users who like to give higher scores in comparison to other users. The user effect is accommodated by including a user bias (u_bias) to model 2.

```
# Model 3: average rating + movie bias + user bias
# inclusion of a user bias (u_bias) since users can in average rate movies higher or lower than the ove
avg_rating <- mean(edx_train_set$rating)

user_avgs <- edx_train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(u_bias = mean(rating - avg_rating - m_bias))

model_3_prediction <- edx_test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(prediction = avg_rating + m_bias + u_bias) %>%
  .$prediction


model_3_RMSE <- RMSE(model_3_prediction, edx_test_set$rating)

all_RMSEs <- bind_rows(all_RMSEs,
                       data_frame(method="Model 3: movie bias + user bias",
                                  RMSE = model_3_RMSE))
all_RMSEs %>% knitr::kable()
```

method	RMSE
Model 1 : overall average	1.0606351
Model 2: movie bias	0.9440709
Model 3: movie bias + user bias	0.8664802

Model 4: modeling the movie effect by regularizing the movie bias (m_bias) using a parameter lambda (m_lambda)

Preliminary data exploration has shown tha many obscure movies were only rated one or a few times. This led to obscure movies being very highly or very lowly rated in average. However, one or a few ratings are

generally not sufficient to confidently predict future ratings.

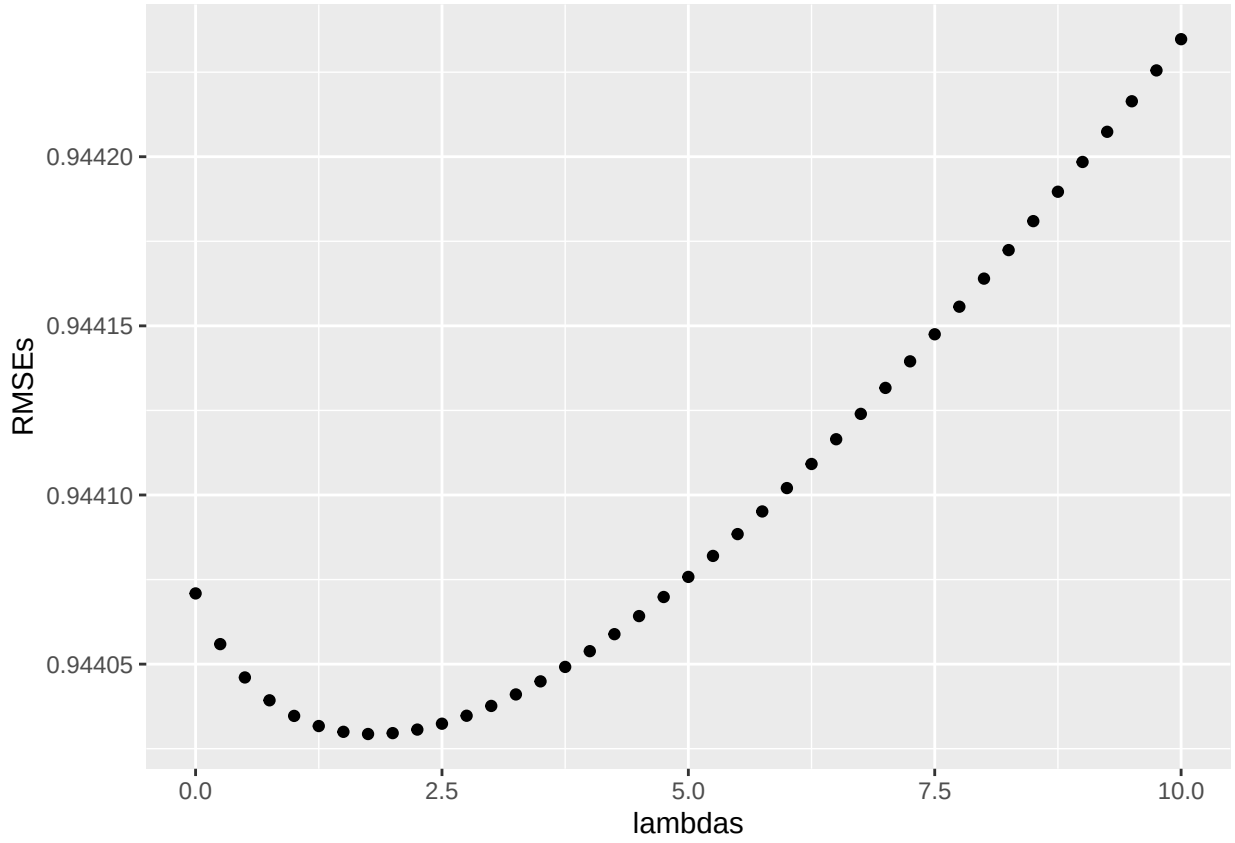
If such small sample sizes are not taken into account, the recommendation algorithm can become overtrained. Small sample sizes can be accounted for by adding a penalty to the movie bias estimates, which increases with smaller sample sizes. This method is called regularization.

How the penalty should vary with the sample size can be modeled using the parameter lambda (m_lambda). In model 4 depicted below a sequence of lambdas is tested to verify which lambda results in the lowest RMSE.

```
# Model 4: average rating + regularized movie bias
# inclusion of a regularized movie bias (reg_m_bias) to account for movies which were not rated often

# Optimize the lambda parameter in the regularization formula
lambdas <- seq(0, 10, 0.25)
avg_rating <- mean(edx_train_set$rating)
sum_rating_minus_avg <- edx_train_set %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - avg_rating), n_i = n())

RMSEs <- sapply(lambdas, function(l){
  predicted_ratings <- edx_test_set %>%
    left_join(sum_rating_minus_avg, by='movieId') %>%
    mutate(m_bias = s/(n_i+l)) %>%
    mutate(prediction = avg_rating + m_bias) %>%
    .$pred
  return(RMSE(predicted_ratings, edx_test_set$rating))
})
qplot(lambdas, RMSEs)
```



```
m_lambda <- lambdas[which.min(RMSEs)]

m_lambda
```

```
## [1] 1.75
```

```
all_RMSEs <- bind_rows(all_RMSEs,
                       data_frame(method="Model 4: regularized movie bias",
                                  RMSE = min(RMSEs)))
all_RMSEs %>% knitr::kable()
```

method	RMSE
Model 1 : overall average	1.0606351
Model 2: movie bias	0.9440709
Model 3: movie bias + user bias	0.8664802
Model 4: regularized movie bias	0.9440293

Model 5: modeling the user effect by regularizing the user bias (u_bias) using a parameter lambda (u_lambda)

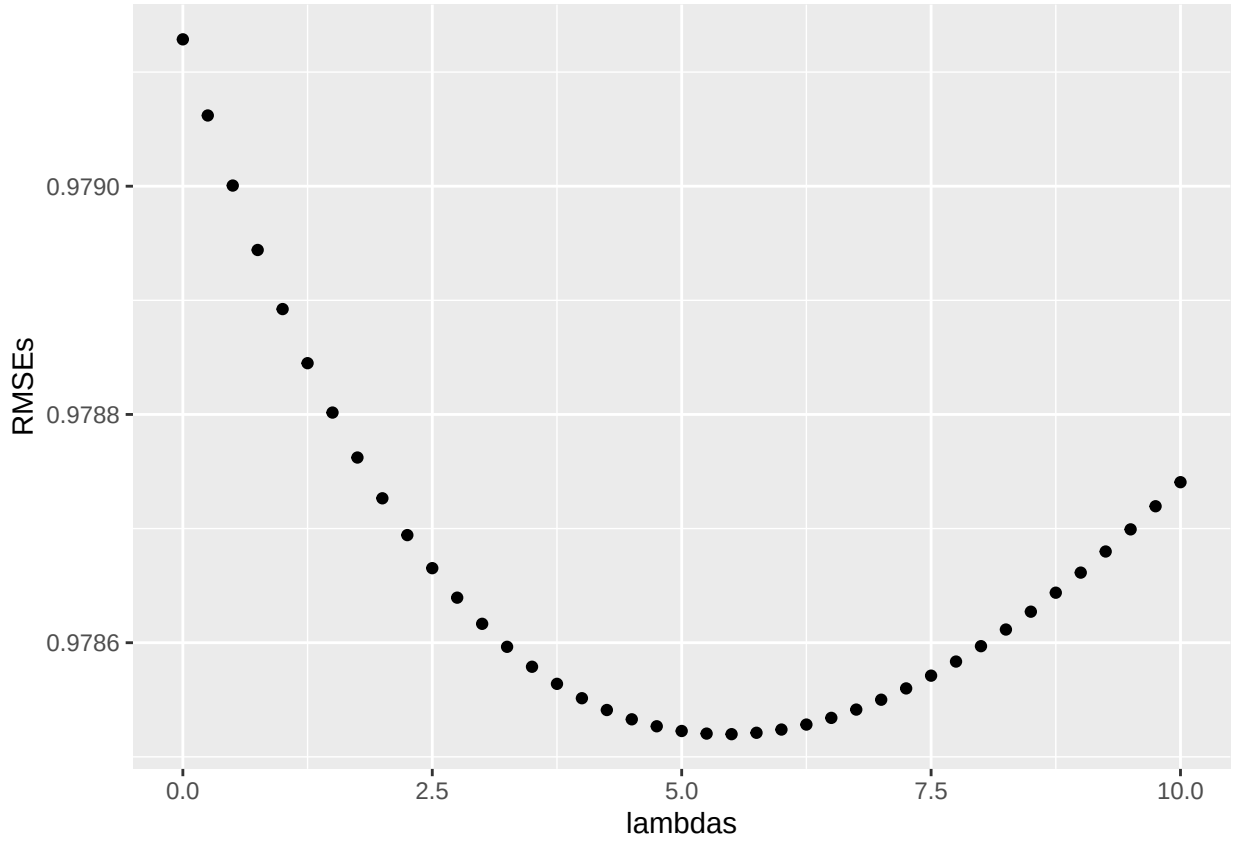
Regularization can be applied as well to the user bias in a similar way to the movie bias. Indeed, a user who has rated a few movies very highly, will not necessarily rate every movie very highly. He may still rate movies in average highly but this cannot be derived with high certainty from the small sample of movies the user has rated.

The lambda parameter was here also used to optimize the penalty to the user bias. The optimized lambda for the user effect is denoted as u_lambda.

```
# Model 5: regularized user bias
#Include a regularized user bias to lower the importance of the user bias for users who did not yet rat

#Optimize the lambda parameter in the regularization formula
lambdas <- seq(0, 10, 0.25)
avg_rating <- mean(edx_train_set$rating)
sum_rating_minus_avg <- edx_train_set %>%
  group_by(userId) %>%
  summarize(s = sum(rating - avg_rating), n_i = n())

RMSEs <- sapply(lambdas, function(l){
  predicted_ratings <- edx_test_set %>%
    left_join(sum_rating_minus_avg, by='userId') %>%
    mutate(u_bias = s/(n_i+l)) %>%
    mutate(prediction = avg_rating + u_bias) %>%
    .$pred
  return(RMSE(predicted_ratings, edx_test_set$rating))
})
qplot(lambdas, RMSEs)
```




```
u_lambda <- lambdas[which.min(RMSEs)]

u_lambda
```

```
## [1] 5.5
```

```
all_RMSEs <- bind_rows(all_RMSEs,
                       data_frame(method="Model 5: regularized user bias",
                                  RMSE = min(RMSEs)))
all_RMSEs %>% knitr::kable()
```

method	RMSE
Model 1 : overall average	1.0606351
Model 2: movie bias	0.9440709
Model 3: movie bias + user bias	0.8664802
Model 4: regularized movie bias	0.9440293
Model 5: regularized user bias	0.9785199

Model 6: modeling movie effect and user effect using the regularized biases and lambdas from the previous models

In model 6, model 4 and 5 are combined so that a regularized bias for the movie and user effect is used. The lambdas optimized in each model are being used: 5.5 for the movie bias and 1.75 for the user bias.

```
# Model 6: regularized movie bias + user bias
#Now include both a regularized user bias with lambda 5.5 and a regularized movie bias with lambda 1.75
```

```
u_lambda <- 5.5
m_lambda <- 1.75

avg_rating <- mean(edx_train_set$rating)

# regularized movie bias based on the lambda calculated in model 4
reg_movie_bias <- edx_train_set %>%
  group_by(movieId) %>%
  summarize(reg_m_bias = (sum(rating - avg_rating))/(n()+m_lambda))

# regularized user bias based on the lambda calculated in model 5
reg_user_bias <- edx_train_set %>%
  left_join(reg_movie_bias, by='movieId') %>%
  group_by(userId) %>%
  summarize(reg_u_bias = (sum(rating - avg_rating - reg_m_bias))/(n()+u_lambda))

model_6_prediction <- edx_test_set %>%
  left_join(reg_movie_bias, by='movieId') %>%
  left_join(reg_user_bias, by='userId') %>%
  mutate(prediction = avg_rating + reg_m_bias + reg_u_bias) %>%
  .$prediction

model_6_RMSE <- RMSE(model_6_prediction, edx_test_set$rating)

all_RMSEs <- bind_rows(all_RMSEs,
                       data_frame(method="Model 6: regularized movie bias + regularized user bias",
                                  RMSE = model_6_RMSE))
all_RMSEs %>% knitr::kable()
```

method	RMSE
Model 1 : overall average	1.0606351
Model 2: movie bias	0.9440709
Model 3: movie bias + user bias	0.8664802
Model 4: regularized movie bias	0.9440293
Model 5: regularized user bias	0.9785199
Model 6: regularized movie bias + regularized user bias	0.8658766

Model 7: modeling movie effect and user effect using regularized biases by optimizing lambda simultaneously for both effects.

In model 7 the same lambda will be used for regularizing both the movie and user effect.

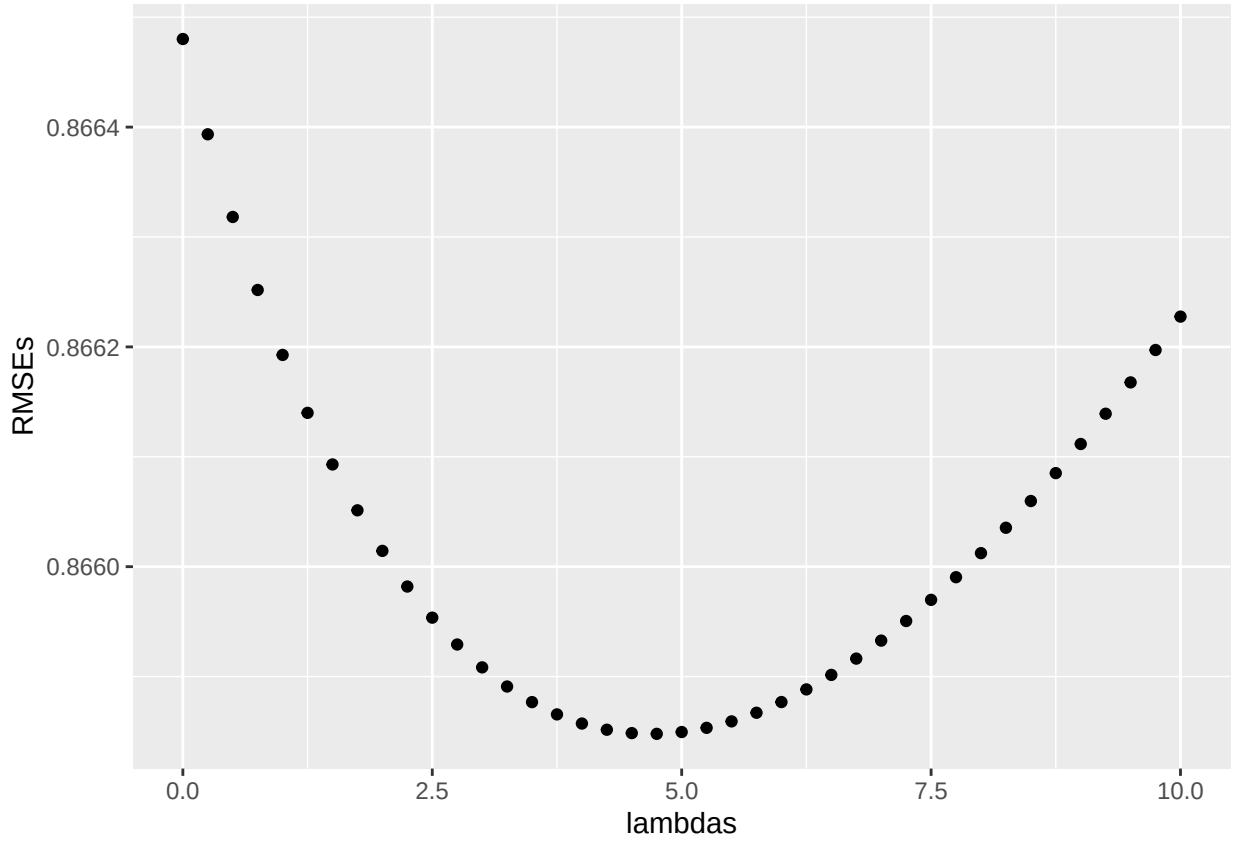
```
# Model 7: regularized movie bias + user bias 2
# In the second type the lambda is simultaneously optimized for the movie bias and the user bias

# First optimize the lambda
avg_rating <- mean(edx_train_set$rating)
lambdas <- seq(0, 10, 0.25)

RMSEs <- sapply(lambdas, function(um_lambda){
  reg_movie_bias <- edx_train_set %>%
    group_by(movieId) %>%
    summarize(reg_m_bias = (sum(rating - avg_rating))/(n()+um_lambda))
```

```
  reg_user_bias <- edx_train_set %>%
    left_join(reg_movie_bias, by='movieId') %>%
    group_by(userId) %>%
    summarize(reg_u_bias = (sum(rating - avg_rating - reg_m_bias))/(n()+um_lambda))
  predicted_ratings <- edx_test_set %>%
    left_join(reg_movie_bias, by='movieId') %>%
    left_join(reg_user_bias, by='userId') %>%
    mutate(prediction = avg_rating + reg_m_bias + reg_u_bias) %>%
    .$prediction
  return(RMSE(predicted_ratings, edx_test_set$rating))
})

qplot(lambdas, RMSEs)
```




```
um_lambda <- lambdas[which.min(RMSEs)]
um_lambda
```

```
## [1] 4.75
```

```
model_7_RMSE <- min(RMSEs)
all_RMSEs <- bind_rows(all_RMSEs,
                       data_frame(method="Model 7: regularized movie bias + regularized user bias 2",
                                  RMSE = model_7_RMSE))
all_RMSEs %>% knitr::kable()
```

method	RMSE

method	RMSE
Model 1 : overall average	1.0606351
Model 2: movie bias	0.9440709
Model 3: movie bias + user bias	0.8664802
Model 4: regularized movie bias	0.9440293
Model 5: regularized user bias	0.9785199
Model 6: regularized movie bias + regularized user bias	0.8658766
Model 7: regularized movie bias + regularized user bias 2	0.8658479

Model 8: Matrix factorization on residuals of model 7 (baseline model)

Since model 7 exhibited the lowest RMSE the residuals from that model were used for matrix factorization. The residuals are obtained after subtracting the average rating, movie bias and user bias from each rating.

```
# Model 8: Matrix factorization
# To further enhance the recommendation the residuals have to be modeled using matrix factorization,
# The matrix factorization will be performed using the recosystem package
# Model 7 will be used as baseline system to calculate the residuals

# Obtain the residuals using the lambda of model 7
um_lambda <- 4.75
avg_rating <- mean(edx_train_set$rating)

reg_movie_bias <- edx_train_set %>%
  group_by(movieId) %>%
  summarize(reg_m_bias = (sum(rating - avg_rating))/(n()+um_lambda))
reg_user_bias <- edx_train_set %>%
  left_join(reg_movie_bias, by='movieId') %>%
  group_by(userId) %>%
  summarize(reg_u_bias = (sum(rating - avg_rating - reg_m_bias))/(n()+um_lambda))

predicted_ratings_m7 <-
  edx_test_set %>%
  left_join(reg_movie_bias, by = "movieId") %>%
  left_join(reg_user_bias, by = "userId") %>%
  mutate(pred = avg_rating + reg_m_bias + reg_u_bias)%>%
  pull(pred)

residuals_train_set <- edx_train_set %>%
  left_join(reg_movie_bias, by = "movieId") %>%
  left_join(reg_user_bias, by = "userId") %>%
  mutate(residual = rating - avg_rating - reg_m_bias - reg_u_bias) %>%
  select(userId, movieId, residual)
head(residuals_train_set)
```

```
##   userId movieId   residual
## 1      1     185  0.6167235
## 2      1     292  0.3273172
## 3      1     316  0.3990895
## 4      1     329  0.4071122
## 5      1     356 -0.2640140
```

```
## 6      1     362  0.3091414
```

These residuals are the basis for matrix factorization. For the recosystem package both edx training and test set need to be transformed into three-column matrices with users, movies and residuals/ratings as column identifiers. These matrices are then temporarily written onto the hard disk, using less RAM memory. Then, the Reco() function in the recosystem package will be used to built a recommender object of which a set of parameters will be trained using the edx training matrix.

Using the optimized parameters and test data a prediction model is made. The prediction is made with the base prediction of model 7 and the residuals predicted here. Finally, an RMSE is calculated.

```
# Use recosystem package to perform matrix factorization
install.packages("recosystem")
library(recosystem)
matrix_train_residuals <- residuals_train_set %>% as.matrix()
matrix_test <- edx_test_set %>%
  select(userId, movieId, rating) %>%
  as.matrix()

# write the matrices on disk and assign them to a variable
write.table(matrix_train_residuals , file = "matrixtrainresiduals.txt" , sep = " " , row.names = FALSE,
write.table(matrix_test, file = "matrix_test.txt" , sep = " " , row.names = FALSE, col.names = FALSE)

set.seed(1992, sample.kind = "Rounding")
trainset <- data_file("matrixtrainresiduals.txt")
testset <- data_file("matrix_test.txt")

# make a recommender object
recommender <-Reco()

# tuning the recommender with the training data
opts <- recommender$tune(trainset, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                        costp_l1 = 0, costq_l1 = 0,
                                        nthread = 1, niter = 10))
opts
```

```
## $min
## $min$dim
## [1] 30
##
## $min$costp_l1
## [1] 0
##
## $min$costp_l2
## [1] 0.01
##
## $min$costq_l1
## [1] 0
##
## $min$costq_l2
## [1] 0.1
##
## $min$lrate
## [1] 0.1
##
## $min$loss_fun
```

```
## [1] 0.8012723
##
##
## $res
##    dim costp_l1 costp_l2 costq_l1 costq_l2 lrate  loss_fun
## 1   10        0     0.01        0     0.01   0.1 0.8177840
## 2   20        0     0.01        0     0.01   0.1 0.8283626
## 3   30        0     0.01        0     0.01   0.1 0.8385643
## 4   10        0     0.10        0     0.01   0.1 0.8127029
## 5   20        0     0.10        0     0.01   0.1 0.8133243
## 6   30        0     0.10        0     0.01   0.1 0.8157175
## 7   10        0     0.01        0     0.10   0.1 0.8087027
## 8   20        0     0.01        0     0.10   0.1 0.8038228
## 9   30        0     0.01        0     0.10   0.1 0.8012723
## 10  10        0     0.10        0     0.10   0.1 0.8249721
## 11  20        0     0.10        0     0.10   0.1 0.8234454
## 12  30        0     0.10        0     0.10   0.1 0.8246958
## 13  10        0     0.01        0     0.01   0.2 0.8244819
## 14  20        0     0.01        0     0.01   0.2 0.8407193
## 15  30        0     0.01        0     0.01   0.2 0.8587363
## 16  10        0     0.10        0     0.01   0.2 0.8133382
## 17  20        0     0.10        0     0.01   0.2 0.8201987
## 18  30        0     0.10        0     0.01   0.2 0.8229976
## 19  10        0     0.01        0     0.10   0.2 0.8094115
## 20  20        0     0.01        0     0.10   0.2 0.8086058
## 21  30        0     0.01        0     0.10   0.2 0.8103958
## 22  10        0     0.10        0     0.10   0.2 0.8223475
## 23  20        0     0.10        0     0.10   0.2 0.8216572
## 24  30        0     0.10        0     0.10   0.2 0.8213752
```

```
# training the matrix factorization model

recommender$train(trainset, opts = c(opts$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.8595   5.5650e+06
##    1       0.8365   5.1695e+06
##    2       0.8194   5.0296e+06
##    3       0.8025   4.8963e+06
##    4       0.7872   4.7806e+06
##    5       0.7737   4.6833e+06
##    6       0.7620   4.6006e+06
##    7       0.7520   4.5319e+06
##    8       0.7433   4.4752e+06
##    9       0.7357   4.4254e+06
##   10       0.7289   4.3812e+06
##   11       0.7229   4.3452e+06
##   12       0.7176   4.3110e+06
##   13       0.7128   4.2832e+06
##   14       0.7085   4.2564e+06
##   15       0.7047   4.2336e+06
##   16       0.7012   4.2127e+06
##   17       0.6980   4.1945e+06
##   18       0.6951   4.1774e+06
##   19       0.6925   4.1620e+06
```

```
# prediction using trained model

prediction <- tempfile()
recommender$predict(testset, out_file(prediction))
```

```
## prediction output generated at C:\Users\yanni\AppData\Local\Temp\Rtmpis4QB9\file884481942c6
```

```
prediction_resid_matfac <- scan(prediction)
prediction_ratings_matfac <- predicted_ratings_m7 + prediction_resid_matfac
model_8_RMSE <- RMSE(prediction_ratings_matfac, edx_test_set$rating)
model_8_RMSE
```

```
## [1] 0.7968553
```

```
all_RMSEs <- bind_rows(all_RMSEs,
                       data_frame(method="Model 8: Matrix factorization",
                                  RMSE = model_8_RMSE))
all_RMSEs %>% knitr::kable()
```

method	RMSE
Model 1 : overall average	1.0606351
Model 2: movie bias	0.9440709
Model 3: movie bias + user bias	0.8664802
Model 4: regularized movie bias	0.9440293
Model 5: regularized user bias	0.9785199
Model 6: regularized movie bias + regularized user bias	0.8658766
Model 7: regularized movie bias + regularized user bias 2	0.8658479
Model 8: Matrix factorization	0.7968553

Final validation on model 8

The RMSE of model 8 is the lowest of all models. However, the RMSE was calculated using the edx test set, which was also used to optimize the lambda in model 7. It is, however, not done in machine learning to use (part of) a data set for training and validation! Therefore, the recommendation algorithm of model 8 will be validated using the validation set separated from the Movielens 10M data set. This data set has not been used for training the algorithms and is thus perfect for validating model 8.

For the baseline model the same steps as for the development of model 7 are used, while for the residuals the matrix factorization method from model 8 is used. Here the complete edx set is used as a training set and the lambda derived in model 7 is used as regularization parameter for the movie and user effect.

```
## Final validation with model 8
# Now use complete edx set for training
#first use model 7 for regularized user and movie bias
avg_rating <- mean(edx$rating)
lambdas <- seq(0, 10, 0.25)

um_lambda <- 4.75
avg_rating <- mean(edx$rating)

reg_movie_bias_valid <- edx %>%
  group_by(movieId) %>%
  summarize(reg_m_bias = (sum(rating - avg_rating))/(n()+um_lambda))
reg_user_bias_valid <- edx %>%
  left_join(reg_movie_bias_valid, by='movieId') %>%
```

```
  group_by(userId) %>%
  summarize(reg_u_bias = (sum(rating - avg_rating - reg_m_bias))/(n()+um_lambda))

predicted_ratings_validation <-
  validation %>%
  left_join(reg_movie_bias_valid, by = "movieId") %>%
  left_join(reg_user_bias_valid, by = "userId") %>%
  mutate(pred = avg_rating + reg_m_bias + reg_u_bias)%>%
  pull(pred)

residuals_edx_set <- edx %>%
  left_join(reg_movie_bias_valid, by = "movieId") %>%
  left_join(reg_user_bias_valid, by = "userId") %>%
  mutate(residual = rating - avg_rating - reg_m_bias - reg_u_bias) %>%
  select(userId, movieId, residual)
head(residuals_edx_set)
```

```
##   userId movieId  residual
## 1      1     122 0.7967788
## 2      1     185 0.5273185
## 3      1     292 0.2387459
## 4      1     316 0.3070651
## 5      1     329 0.3192734
## 6      1     355 1.1679938
```

```
matrix_edx_residuals <- residuals_edx_set %>% as.matrix()
matrix_validation <- validation %>%
  select(userId, movieId, rating) %>%
  as.matrix()

write.table(matrix_edx_residuals , file = "matrixedxresiduals.txt" , sep = " " , row.names = FALSE, col
write.table(matrix_validation, file = "matrix_validation.txt" , sep = " " , row.names = FALSE, col.names

set.seed(1992, sample.kind = "Rounding")
edxset <- data_file("matrixedxresiduals.txt")
validationset <- data_file("matrix_validation.txt")

# make a recommender object
recommender_valid <-Reco()

# tuning the recommender with the training data
opts_valid <- recommender_valid$tune(edxset, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                                         costp_l1 = 0, costq_l1 = 0,
                                                         nthread = 1, niter = 10))
opts_valid
```

```
## $min
## $min$dim
## [1] 30
##
## $min$costp_l1
## [1] 0
##
## $min$costp_l2
```

```
## [1] 0.01
##
## $min$costq_l1
## [1] 0
##
## $min$costq_l2
## [1] 0.1
##
## $min$lrate
## [1] 0.1
##
## $min$loss_fun
## [1] 0.7941609
##
##
## $res
##    dim costp_l1 costp_l2 costq_l1 costq_l2 lrate  loss_fun
## 1   10        0     0.01        0     0.01   0.1 0.8072272
## 2   20        0     0.01        0     0.01   0.1 0.8114371
## 3   30        0     0.01        0     0.01   0.1 0.8200044
## 4   10        0     0.10        0     0.01   0.1 0.8049610
## 5   20        0     0.10        0     0.01   0.1 0.8018715
## 6   30        0     0.10        0     0.01   0.1 0.8034186
## 7   10        0     0.01        0     0.10   0.1 0.8038635
## 8   20        0     0.01        0     0.10   0.1 0.7973237
## 9   30        0     0.01        0     0.10   0.1 0.7941609
## 10  10        0     0.10        0     0.10   0.1 0.8242639
## 11  20        0     0.10        0     0.10   0.1 0.8235046
## 12  30        0     0.10        0     0.10   0.1 0.8228670
## 13  10        0     0.01        0     0.01   0.2 0.8106341
## 14  20        0     0.01        0     0.01   0.2 0.8233611
## 15  30        0     0.01        0     0.01   0.2 0.8368961
## 16  10        0     0.10        0     0.01   0.2 0.8062928
## 17  20        0     0.10        0     0.01   0.2 0.8064956
## 18  30        0     0.10        0     0.01   0.2 0.8082813
## 19  10        0     0.01        0     0.10   0.2 0.8034891
## 20  20        0     0.01        0     0.10   0.2 0.7991972
## 21  30        0     0.01        0     0.10   0.2 0.7990707
## 22  10        0     0.10        0     0.10   0.2 0.8227886
## 23  20        0     0.10        0     0.10   0.2 0.8215954
## 24  30        0     0.10        0     0.10   0.2 0.8219777
```

```
# training the matrix factorization model

recommender_valid$train(edxset, opts = c(opts_valid$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.8590   6.9701e+06
##    1       0.8343   6.4465e+06
##    2       0.8159   6.2616e+06
##    3       0.7986   6.0910e+06
##    4       0.7840   5.9561e+06
##    5       0.7717   5.8478e+06
##    6       0.7614   5.7569e+06
##    7       0.7528   5.6832e+06
```

```
##    8       0.7455   5.6253e+06
##    9       0.7391   5.5727e+06
##   10       0.7335   5.5290e+06
##   11       0.7287   5.4934e+06
##   12       0.7243   5.4604e+06
##   13       0.7204   5.4312e+06
##   14       0.7169   5.4046e+06
##   15       0.7138   5.3828e+06
##   16       0.7109   5.3620e+06
##   17       0.7083   5.3433e+06
##   18       0.7058   5.3264e+06
##   19       0.7037   5.3123e+06
```

```
prediction_validation <- tempfile()
recommender_valid$predict(validationset, out_file(prediction_validation))
```

```
## prediction output generated at C:\Users\yanni\AppData\Local\Temp\Rtmpis4QB9\file88446d0429d8
```

```
prediction_validation_resid <- scan(prediction_validation)
prediction_validation_ratings <- predicted_ratings_validation + prediction_validation_resid
RMSE_validation <- RMSE(prediction_validation_ratings, validation$rating)
RMSE_validation
```

```
## [1] 0.7865076
```

Conclusion

This cap stone project has shown how a movie recommendation algorithm can be developed using data exploration and machine learning. This data science challenge was carried out using part of the classical MovieLens dataset. RMSE was used as the parameter to evaluate the different recommendation models. A summary of the RMSEs obtained during the development of the recommendation model are shown underneath.

method	RMSE
Model 1 : overall average	1.0606351
Model 2: movie bias	0.9440709
Model 3: movie bias + user bias	0.8664802
Model 4: regularized movie bias	0.9440293
Model 5: regularized user bias	0.9785199
Model 6: regularized movie bias + regularized user bias	0.8658766
Model 7: regularized movie bias + regularized user bias 2	0.8658479
Model 8: Matrix factorization	0.7968553
Model 8 Validation	0.7865076

The final RMSE table shows that model 8 has by far the lowest RMSE. This shows that matrix factorization is a very strong technique to discover patterns between users and movies based on a large and sparse set of ratings. These patterns could not be derived from purely analyzing the explicit information in the dataset. The final validation also confirms the strength of the recommendation algorithm of model 8. The baseline model could be further optimized by also adding a genre bias, movieage bias and a release year bias. In addition, using cross validation instead of a single test set would be more reliable for determining the regularization parameter.