MovieLens project: Creating a movie recommendation system

Anton Ivanov

2021-11-01

1 Introduction

In this project, we will be creating a movie recommendation system using the 10M version of the MovieLens dataset. This dataset contains 10 million ratings for about 10,000 movies from 72,000 users.

Recommendation systems make specific recommendations based on ratings from users to items. They are of particular importance in the modern world, since they allow companies such as online shops or streaming services to recommend users matching goods or content, providing higher profits.

The goal of the project is to predict a rating for a specific movie from a specific user, based on the available ratings. The used ratings are as following: one star suggests it is not a good movie, whereas five stars suggest it is an excellent movie.

Since in the dataset only 10 million ratings are available out of potential 78 million (if every user would rate every movie), about 86% of ratings remain unknown and have to be predicted.

The recommendation system will be based on a linear model containing movie, user, genre and time effects. Each effect will be considered step-by-step, using a training and test sets for evaluation of the model performance. Regularization of the effects will be then implemented to improve the performance. After final tuning of the model, it will be applied to the validation set to assess the final performance. Finally, the model will be used to recommend movies to a real user.

2 Analysis and Methods

2.1 Data preparation

The 10M Movielens dataset is available for a downloaded as a *.zip file containing two *.dat files: movies.dat and ratings.dat. The files contain data of the following features:

- user id: an entry number specific for each user;
- movie id: an entry number specific for each movie;
- rating: movie ratings in the range from 0.5 to 5 stars with the step 0.5 stars;
- time stamp: an integer presented as number of seconds after 1st January 1970;
- title of the movie with release year;
- genre of the movie.

The features are separated by double colons "::". The values are extracted and saved as a dataframe using the following code.

```
library(tidyverse)
library(data.table)

if(!file.exists("movielens.RData")){

  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")
  save(movielens, file = "movielens.RData")
} else {
  load("movielens.RData")
}
```

In the resulting dataframe, each row represents a rating by one user to one movie. These are the first 10 rows of the dataframe:

```
as_tibble(movielens)
```

```
## # A tibble: 10,000,054 x 6
##    userId movieId rating timestamp title               genres
##     <int>   <dbl>  <dbl>     <int> <chr>               <chr>
##  1      1     122      5 838985046 Boomerang (1992)    Comedy|Romance
##  2      1     185      5 838983525 Net, The (1995)     Action|Crime|Thriller
##  3      1     231      5 838983392 Dumb & Dumber (199~ Comedy
##  4      1     292      5 838983421 Outbreak (1995)     Action|Drama|Sci-Fi|Thri~
##  5      1     316      5 838983392 Stargate (1994)     Action|Adventure|Sci-Fi
##  6      1     329      5 838983392 Star Trek: Generat~ Action|Adventure|Drama|S~
##  7      1     355      5 838984474 Flintstones, The (~ Children|Comedy|Fantasy
##  8      1     356      5 838983653 Forrest Gump (1994) Comedy|Drama|Romance|War
##  9      1     362      5 838984885 Jungle Book, The (~ Adventure|Children|Roman~
## 10      1     364      5 838983707 Lion King, The (19~ Adventure|Animation|Chil~
## # ... with 10,000,044 more rows
```

2.2 Preparation of the training, test and validation sets

For the final evaluation of the recommendation system's performance, a validation dataset containing 10% of data is created. The rest of the data is saved as a edx subset. It is ensured that the userId and movieId values in the validation set are also present in the edx set.

```
library(caret)
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,]

validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)
```

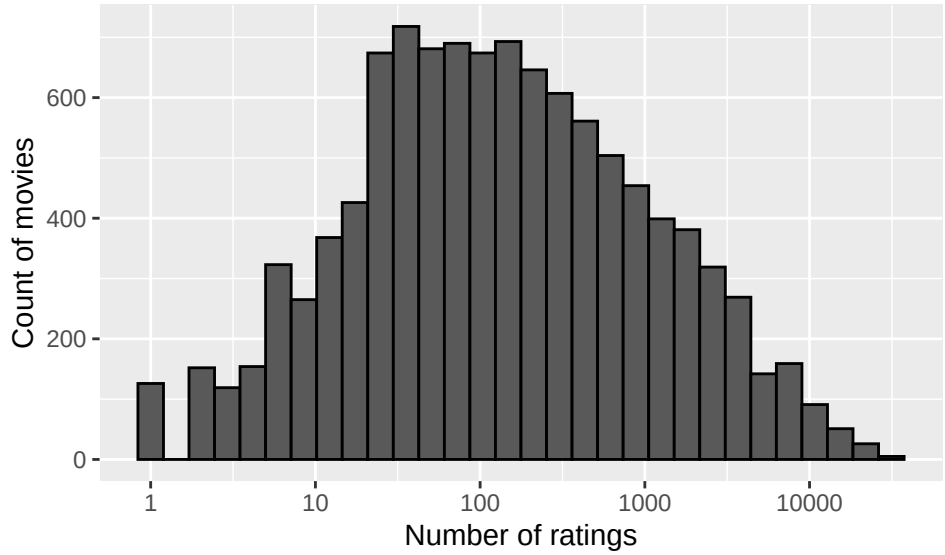
The edx dataset is used for the creation and training of the recommendation system and is split into the training set (80%) and test set (20%) for these needs.

```
set.seed(2)
test_index <- createDataPartition(edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[-test_index]
test_set <- edx[test_index]
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

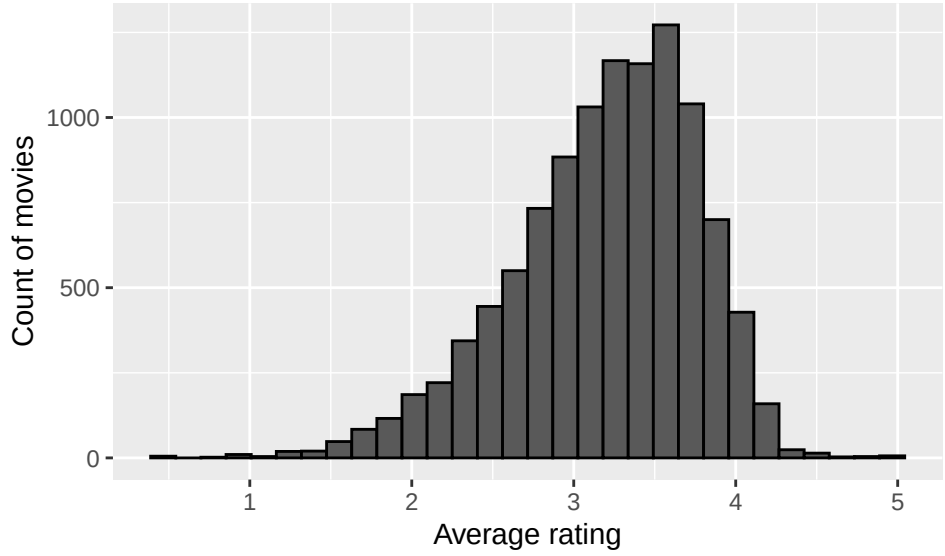
2.3 Data exploration

In the section, the edx dataset will be explored in detail in order to identify relationships between ratings and the rest of the available data.

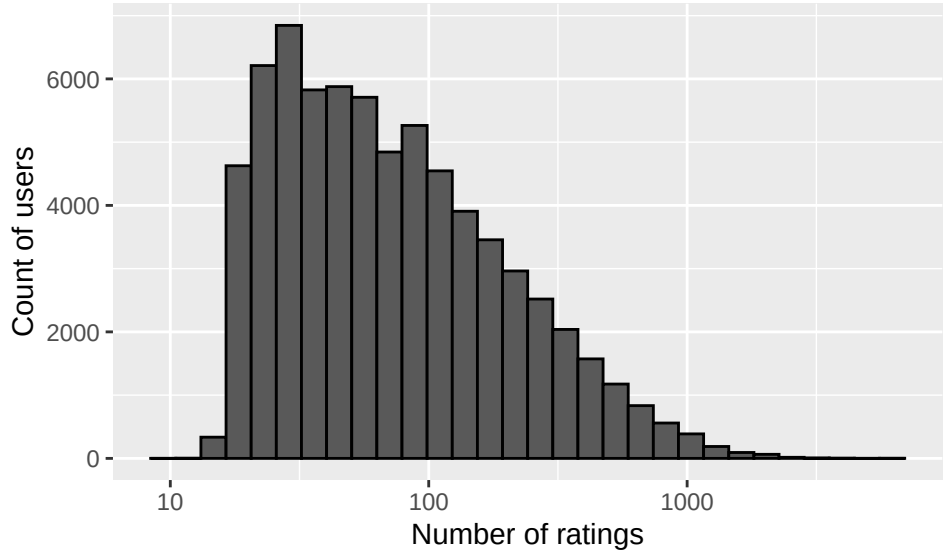
The following histogram shows the number of ratings per movie. As can be seen, most of the movies received roughly between 10 an 1000 ratings. As expected, some movies received only several ratings, whereas other movies received more than 10,000 ratings — some movies are more popular than others.



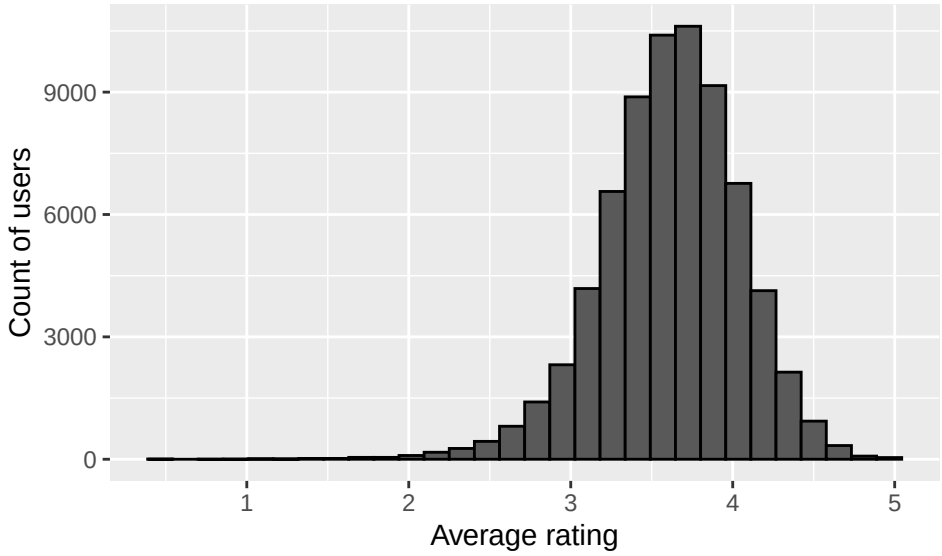
Average ratings per movie are shown in the following histogram. Variability here is also quite high, whereas most of the movies get about 3.5 stars. The distribution is skewed to higher ratings meaning that movies are rated more often positive, than negative.



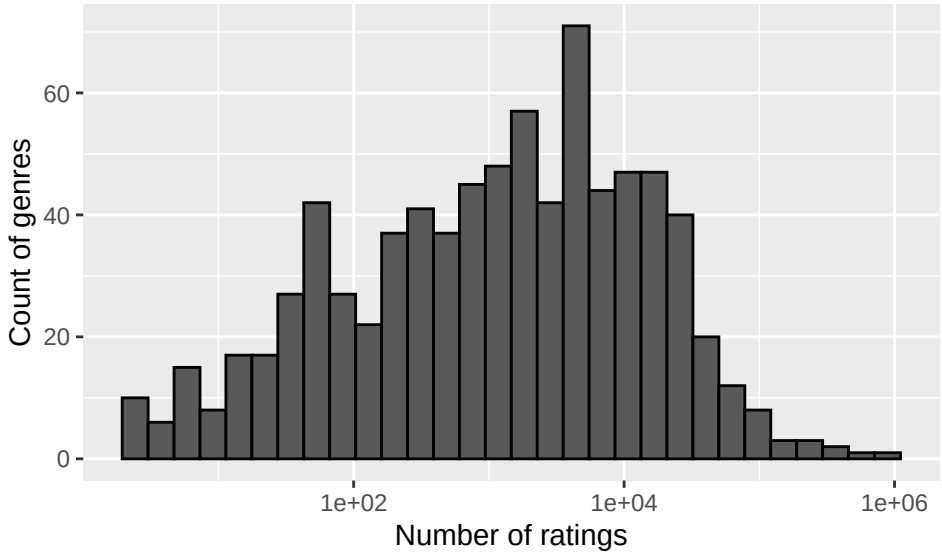
Distribution of ratings per user shows that most of the users gave about 50 ratings. Only a small part of users provided ratings to more than 1000 movies. Again this fact highlights the challenge for the recommendation system to provide predictions based on scarce available ratings.



As can be seen in the following histogram, the distribution of average ratings per user is symmetrical with the majority of average ratings about 3.7 stars. Again, there are some users that on average tend to give more positive ratings than others.



For ratings per movie genre, high variability can be again observed, what is depicted in the following histogram. Some genres are more popular than others.



Additionally, it is easy to show that on average some genres get higher ratings than others. These are the top 5 rated genres:

```
edx %>%
  group_by(genres) %>%
  summarize(mean_g = mean(rating), n = n()) %>%
  filter(n > 500) %>%
  slice_max(mean_g, n = 5) %>%
  knitr::kable()
```

genres	mean_g	n
Drama|Film-Noir|Romance	4.304115	2989
Action|Crime|Drama|IMAX	4.297068	2353
Animation|Children|Comedy|Crime	4.275429	7167
Film-Noir|Mystery	4.239479	5988
Crime|Film-Noir|Mystery	4.216803	4029

And the bottom 5 rated genres:

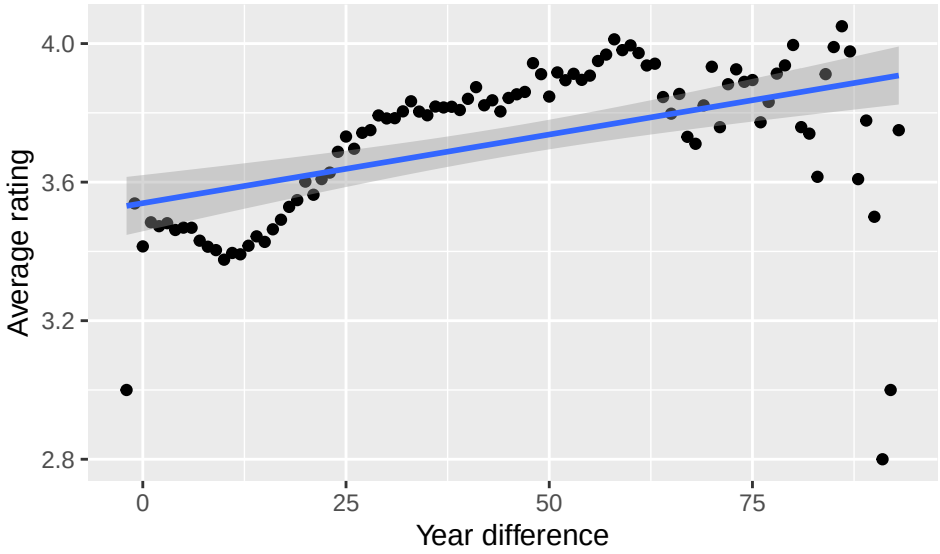
```
edx %>%
  group_by(genres) %>%
  summarize(mean_g = mean(rating), n = n()) %>%
  filter(n > 500) %>%
  slice_min(mean_g, n = 5) %>%
  knitr::kable()
```

genres	mean_g	n
Documentary|Horror	1.449112	619
Action|Children|Comedy	1.910232	518
Action|Adventure|Children	1.915048	824
Adventure|Animation|Children|Fantasy|Sci-Fi	1.924747	691
Action|Children	2.044110	3922

The difference of average rating between best and worst rated genres is significant.

The last bit of available information in the dataset is the year of the movie release (included in the title) and the date when the rating was given. This information itself might not be very helpful. However, according to our experience, the time **between** the movie release and the rating date (*year difference*) might be important. It seems that older movie being rated after many years are rated more positive. To check this hypothesis, the dataset will first be adjusted to include the new columns: movie date, rating date and the difference in years between them. Afterwards, the ratings will be plotted against the time difference for a sample of 1,000,000 entries.

```
library(lubridate)

edx <- edx %>%
  mutate(year_m = str_extract(title, "\\(\\d{4}\\)")) %>%
  mutate(year_m = str_extract(year_m, "\\d{4}")) %>%
  mutate(year_m = as.numeric(year_m),
         year_r = year(as.POSIXct(timestamp, origin="1970-01-01")),
         year_diff = year_r - year_m)
```



As can be seen, indeed, there is a positive correlation between the movie rating and the *year difference*.

The discussed above insights will be used in the following to create the recommendation model.

2.4 Creating a linear model

The linear model will be built based on observations made for the edx dataset. We will start with the simplest model and will gradually increase its complexity.

The simplest model would be just assuming all unknown ratings equal to the average rating of all movies. It can be presented with the following equation.

$$Y_{u,i} = \mu + \epsilon_{u,i} \tag{1}$$

where $Y_{u,i}$ is the predicted rating, μ is a mean of all ratings in the data set and $\epsilon_{u,i}$ is an independent error (it is added as a reminder that the prediction is not perfect).

Of course such assumption would not result in an accurate prediction, as will be shown in the Results Chapter. To improve the prediction, an effect of each specific movie on its rating will be considered. It is obvious that each movie has a different "true" rating — there are better and worse movies. This was supported in the previous section by showing the distribution of the avarage movie ratings.

This fact will be used to predict ratings. The model will include the average of **all** movies in the dataset plus the term **specific** for each movie:

$$Y_{u,i} = \mu + b_i + \epsilon_{u,i} \tag{2}$$

where b_i is the item (movie) effect.

Analogously, similar relationships were shown for the overall ratings of one user in the previous section. Each user shows a different overall average rating. This means that on average some users give higher ratings than other users.

Thus, another term can be included in the model — the user effect:

$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i} \tag{3}$$

where b_u is the user effect.

Further, we have shown that different genres are variously rated. Accordingly, the genre effect is also included in the model.

$$Y_{u,i} = \mu + b_i + b_u + b_g + \epsilon_{u,i} \tag{4}$$

where b_g is the genre effect.

Finally, the *year difference* between movie release date and rating date is added to the model via the time effect b_y, since its effect on ratings average was also confirmed:

$$Y_{u,i} = \mu + b_i + b_u + b_g + b_y + \epsilon_{u,i} \tag{5}$$

Now the model includes all useful information available in the dataset presented through the movie, user, genre and time effects. But how do we calculate the effects?

2.5 Calculation of effects

We will perform the calculation of all effects step by step starting with the movie effect b_i. From the equation 2, the b_i specific for each movie can be approximated based on the average rating of all movies μ and the **known** ratings of a movie.

$$\hat{b_i} = \sum_{u=1}^{n_i} (Y_{i,u} - \hat{\mu}) \tag{6}$$

whereas terms with hats are estimated values.

Knowing b_i and rewriting the equation 3 the effect b_u can be found as following.

$$\hat{b_u} = \sum_{u=1}^{n_i} (Y_{i,u} - \hat{\mu} - \hat{b_i}) \tag{7}$$

In the same manner, the values of the last two effects can be found based on equations 4-5.

$$\hat{b_g} = \sum_{u=1}^{n_i} (Y_{i,u} - \hat{\mu} - \hat{b_i} - \hat{b_u}) \tag{8}$$

$$\hat{b_y} = \sum_{u=1}^{n_i} (Y_{i,u} - \hat{\mu} - \hat{b_i} - \hat{b_u} - \hat{b_g}) \tag{9}$$

2.6 Calculation of effects with regularization

As shown in equations 6-9, calculation of effects depends on averages of ratings grouped in different ways (ratings for the same movie, of the same user, of the same genre or of the same movie date and rating date difference). This means that the more ratings of one group are available, the better estimation of an effect is possible. To compensate a strong influence of low number of ratings, a penalty term λ is introduced into the equations for the effects. Penalizing large estimates that are formed using small sample sizes is the concept of **regularization**.

For example, for the movie effect b_i, the modified equation looks as following:

$$\hat{b_i}(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{i,u} - \hat{\mu}) \tag{10}$$

As can be seen, when the sample size n_i is large giving a stable estimate of $\hat{b_i}$, the penalty term λ is ignored and $n_i + \lambda \approx n_i$. By a small sample size λ becomes dominant and $\hat{b_i}(\lambda)$ is shrunken towards 0.

In a similar way, penalty terms will be added to each effect. Optimal penalty terms will be defined by trying out different values and comparing their influence on the performance of the prediction.

2.7 Loss function

The performance of the prediction system will be measured by the typical error loss — the residual mean squared error (RMSE) on a test set:

$$RMSE = \sqrt{\frac{1}{N} \sum_{u,i} (\hat{y}_{u,i} - y_{u,i})^2} \tag{11}$$

where $y_{u,i}$ is an actual rating of user **u** for movie **i**, $\hat{y}_{u,i}$ is the corresponding predicted value and **N** is the number of user/movie combinations.

The function that will compute the RMSE for vectors of ratings and their corresponding predictors:

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

3 Results

In the following, the described model will be built and the effects and regularization penalty terms will be calculated based on the train and test sets. The final performance of the recommendation system will be evaluated with the validation set.

3.1 Model without regularization

First, we will try just to randomly guess the ratings and show the corresponding RMSE:

```
set.seed(3)
pred_random <- sample(seq(0.5, 5, by = 0.5), nrow(test_set), replace = TRUE)
rmse_random <- RMSE(test_set$rating, pred_random)
rmse_random
```

```
## [1] 1.941521
```

Not surprisingly, the RMSE is quite high. Just applying our simplest model — assuming all ratings to be the average of the known ratings — already improves the performance:

```
mu <- mean(train_set$rating)
rmse_mean <- RMSE(test_set$rating, mu)
rmse_mean
```

```
## [1] 1.060273
```

Now we will add the first effect (movie effect) to the model. The effect is calculated according to the equation 6:

```
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/n())

predicted_ratings_b_i <- test_set %>%
  left_join(b_i, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)

rmse_b_i <- RMSE(test_set$rating, predicted_ratings_b_i)
rmse_b_i
```

```
## [1] 0.9440821
```

The addition of the movie effect results in the improvement of the prediction on the test set of 11%. This is a considerable improvement of the performance. The implementation of the other effects in the model and their influence on the model performance are shown in the following.

User effect:

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

predicted_ratings_b_iu <- test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

rmse_b_iu <- RMSE(test_set$rating, predicted_ratings_b_iu)
rmse_b_iu
```

```
## [1] 0.8669336
```

Genre effect:

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

predicted_ratings_b_iug <-
  test_set %>%
  left_join(b_i, by = "movieId") %>%
```

```
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)

rmse_b_iug <- RMSE(test_set$rating, predicted_ratings_b_iug)
rmse_b_iug
```

```
## [1] 0.8665872
```

To calculate and apply the time effect, the *year difference* will be added to the data sets.

```
train_set <- train_set %>%
  mutate(year_m = str_extract(title, "\\(\\d{4}\\)")) %>%
  mutate(year_m = str_extract(year_m, "\\d{4}")) %>%
  mutate(year_m = as.numeric(year_m),
         year_r = year(as.POSIXct(timestamp, origin="1970-01-01")),
         year_diff = year_r - year_m)

test_set <- test_set %>%
  mutate(year_m = str_extract(title, "\\(\\d{4}\\)")) %>%
  mutate(year_m = str_extract(year_m, "\\d{4}")) %>%
  mutate(year_m = as.numeric(year_m),
         year_r = year(as.POSIXct(timestamp, origin="1970-01-01")),
         year_diff = year_r - year_m)
```

Time effect:

```
b_y <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  group_by(year_diff) %>%
  summarize(b_y = sum(rating - b_i - b_u - b_g - mu)/n())

predicted_ratings_b_iugy <-
  test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_y, by = "year_diff") %>%
  mutate(pred = mu + b_i + b_u + b_g + b_y) %>%
  pull(pred)

rmse_b_iugy <- RMSE(test_set$rating, predicted_ratings_b_iugy)
rmse_b_iugy
```

```
## [1] 0.8661955
```

All data are summarized in a table with the following code:

```
results <- tibble(method = c("Random",
                             "Mean",
                             "Movie Effect",
                             "Movie + User Effect",
                             "Movie + User + Genre Effect",
                             "Movie + User + Genre + Time Effect"),
                  RMSE = c(rmse_random,
                           rmse_mean,
                           rmse_b_i,
                           rmse_b_iu,
                           rmse_b_iug,
                           rmse_b_iugy))

results <- results %>%
  mutate(improvement = (1 - RMSE/lag(RMSE))*100)
knitr::kable(results)
```

method	RMSE	improvement
Random	1.9415205	NA
Mean	1.0602732	45.3895445
Movie Effect	0.9440821	10.9585997
Movie + User Effect	0.8669336	8.1718020
Movie + User + Genre Effect	0.8665872	0.0399495
Movie + User + Genre + Time Effect	0.8661955	0.0452112

An improvement of RMSE is observed for the addition of each effect, though in a smaller extent for the last two effects.

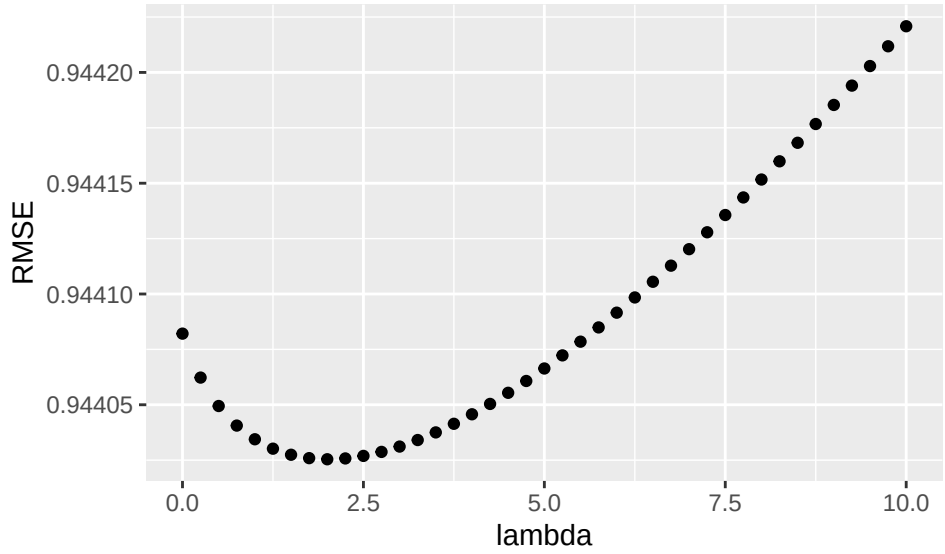
3.2 Model with regularization

In order to decrease RMSE even further, regularization of the effects will be introduced according to the equation 10. Penalty terms will be defined step-by-step for all effects.

Movie effect with regularization:

```
lambdas1 <- seq(0, 10, 0.25)

rmses_b_i <- sapply(lambdas1, function(l){

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

  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)

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

The influence of the penalty term value on RMSE can be shown in the following plot:



The best λ is defined and the lowest RMSE is shown. A vector of movie effects based on the best λ is saved for the final evaluation.

```
lambda1 <- lambdas1[which.min(rmses_b_i)]
lambda1
```

```
## [1] 2
```

```
rmse_b_i_reg <- min(rmses_b_i)
rmse_b_i
```

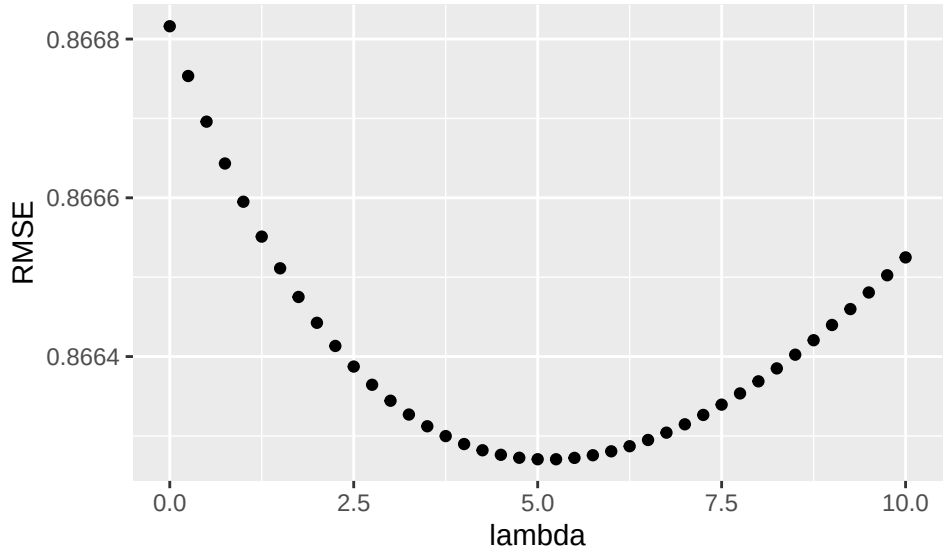
```
## [1] 0.9440821
```

```
b_i_reg <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i_reg = sum(rating - mu)/(n()+lambda1))
```

User effect with regularization:

```
lambdas2 <- seq(0, 10, 0.25)

rmses_b_iu <- sapply(lambdas2, function(l){

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

  predicted_ratings <- test_set %>%
    left_join(b_i_reg, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i_reg + b_u) %>%
```

```
    pull(pred)

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




```
lambda2 <- lambdas2[which.min(rmses_b_iu)]
lambda2
```

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

```
rmse_b_iu_reg <- min(rmses_b_iu)
rmse_b_iu_reg
```

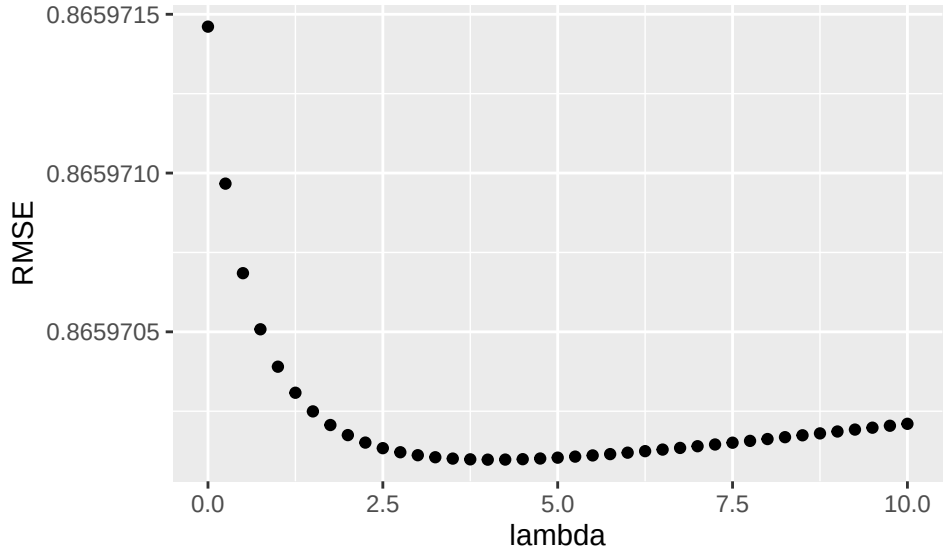
```
## [1] 0.8662706
```

```
b_u_reg <- train_set %>%
  left_join(b_i_reg, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u_reg = sum(rating - b_i_reg - mu)/(n()+lambda2))
```

Genre effect with regularization:

```
lambdas3 <- seq(0, 10, 0.25)

rmses_b_iug <- sapply(lambdas3, function(l){

  b_g <- train_set %>%
    left_join(b_i_reg, by="movieId") %>%
    left_join(b_u_reg, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i_reg - b_u_reg - mu)/(n()+l))
```

```
  predicted_ratings <- test_set %>%
    left_join(b_i_reg, by = "movieId") %>%
    left_join(b_u_reg, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i_reg + b_u_reg + b_g) %>%
    pull(pred)

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




```
lambda3 <- lambdas3[which.min(rmses_b_iug)]
lambda3
```

```
## [1] 4
```

```
rmse_b_iug_reg <- min(rmses_b_iug)
rmse_b_iug_reg
```

```
## [1] 0.8659701
```

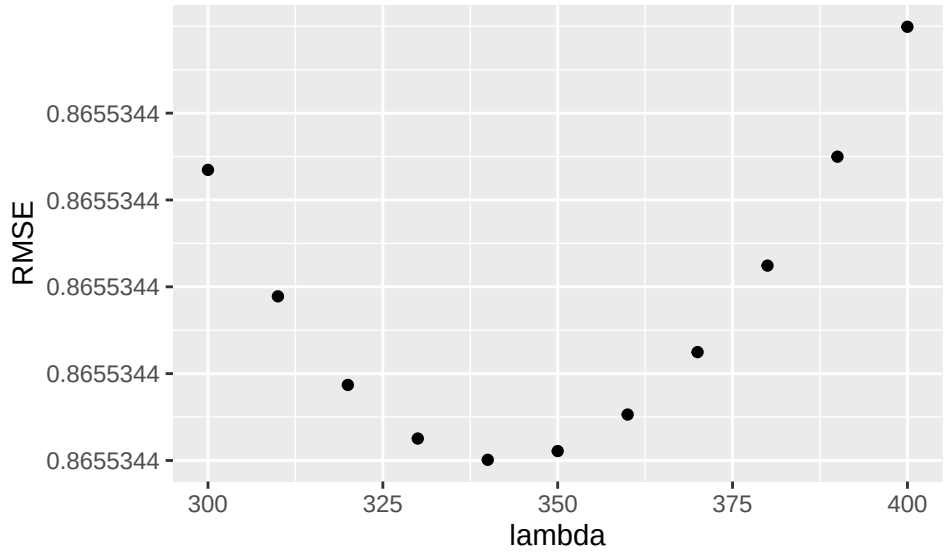
```
b_g_reg <- train_set %>%
  left_join(b_i_reg, by = "movieId") %>%
  left_join(b_u_reg, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g_reg = sum(rating - b_i_reg - b_u_reg - mu)/(n()+lambda3))
```

Time effect with regularization:

```
lambdas4 <- seq(300, 400, 10)

rmses_b_iugy <- sapply(lambdas4, function(l){
```

```
  b_y <- train_set %>%
    left_join(b_i_reg, by="movieId") %>%
    left_join(b_u_reg, by = "userId") %>%
    left_join(b_g_reg, by = "genres") %>%
    group_by(year_diff) %>%
    summarize(b_y = sum(rating - b_i_reg - b_u_reg - b_g_reg - mu)/(n()+l))

  predicted_ratings <-  test_set %>%
    left_join(b_i_reg, by = "movieId") %>%
    left_join(b_u_reg, by = "userId") %>%
    left_join(b_g_reg, by = "genres") %>%
    left_join(b_y, by = "year_diff") %>%
    mutate(pred = mu + b_i_reg + b_u_reg + b_g_reg + b_y) %>%
    pull(pred)

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

Since the parameter *year difference* presents a relatively small number of groups, the sample number of each group is very high. Therefore, the regularization for this parameter is less important and the value of λ is relative high.




```
lambda4 <- lambdas4[which.min(rmses_b_iugy)]
lambda4
```

```
## [1] 340
```

```
rmse_b_iugy_reg <- min(rmses_b_iugy)
rmse_b_iugy_reg
```

```
## [1] 0.8655344
```

```
b_y_reg <- train_set %>%
  left_join(b_i_reg, by = "movieId") %>%
  left_join(b_u_reg, by = "userId") %>%
  left_join(b_g_reg, by = "genres") %>%
  group_by(year_diff) %>%
  summarize(b_y_reg = sum(rating - b_i_reg - b_u_reg - b_g_reg - mu)/(n()+lambda4))
```

Improvement achieved by the introduction of the regularization is shown below (as percent). Though it might not seem significant, the improvement is comparable with the improvement through the introduction of the genre effect and time effect themselves.

```
(1 - rmse_b_iugy_reg/rmse_b_iugy) * 100
```

```
## [1] 0.07631899
```

Now predicted ratings based on all effects can be calculated and used for the fine tunung.

```
predicted_ratings_reg <-  test_set %>%
  left_join(b_i_reg, by = "movieId") %>%
  left_join(b_u_reg, by = "userId") %>%
  left_join(b_g_reg, by = "genres") %>%
  left_join(b_y_reg, by = "year_diff") %>%
  mutate(pred = mu + b_i_reg + b_u_reg + b_g_reg + b_y_reg) %>%
  pull(pred)
```

3.3 Fine tuning

Finally, some small corrections can be applied to further improve the predictions. Since the model calculates ratings as a sum of several parameters, the ratings lie ipso facto not in the range between 0.5 and 5.0 stars. The range for the current calculation is: -0.6410348, 5.9877228. To correct this, the ratings below 0.5 will be substituted with 0.5 and the ratings above 5.0 with 5.0 with the following code.

```
predicted_ratings_reg_corr <- predicted_ratings_reg
predicted_ratings_reg_corr[predicted_ratings_reg_corr > 5] <- 5
predicted_ratings_reg_corr[predicted_ratings_reg_corr < 0.5] <- 0.5

rmse_b_iugy_reg_final <- RMSE(test_set$rating, predicted_ratings_reg_corr)
rmse_b_iugy_reg_final
```

```
## [1] 0.8654066
```

As can be seen, this correction results in an improved RMSE.

Moreover, since the actual ratings are presented between 0.5 and 5.0 stars with the step 0.5, one can assume that rounding the predicted ratings to the nearest .5 or .0 values could also result in an improvement. However, this approach shows a considerable increase of RMSE and will not be applied in the final fine tuning.

```
predicted_ratings_reg_corr2 <- plyr::round_any(predicted_ratings_reg_corr, 0.5)

RMSE(test_set$rating, predicted_ratings_reg_corr2)
```

```
## [1] 0.8772618
```

All new results including the effect regularization and the fine tuning are added to the summary table.

```
results_reg <- tibble(method = c("Movie Effect Reg",
                                 "Movie + User Effect Reg",
                                 "Movie + User + Genre Effect Reg",
                                 "Movie + User + Genre + Time Effect Reg",
                                 "All Reg + Correction"),
                      RMSE = c(rmse_b_i_reg,
                               rmse_b_iu_reg,
                               rmse_b_iug_reg,
                               rmse_b_iugy_reg,
                               rmse_b_iugy_reg_final))

results_reg <- results_reg %>%
  mutate(improvement = (1 - RMSE/lag(RMSE))*100)

results <- rbind(results, results_reg)
knitr::kable(results)
```

method	RMSE	improvement
Random	1.9415205	NA
Mean	1.0602732	45.3895445
Movie Effect	0.9440821	10.9585997
Movie + User Effect	0.8669336	8.1718020
Movie + User + Genre Effect	0.8665872	0.0399495
Movie + User + Genre + Time Effect	0.8661955	0.0452112
Movie Effect Reg	0.9440254	NA
Movie + User Effect Reg	0.8662706	8.2365129
Movie + User + Genre Effect Reg	0.8659701	0.0346937
Movie + User + Genre + Time Effect Reg	0.8655344	0.0503153
All Reg + Correction	0.8654066	0.0147581

3.4 Model evaluation

Now everything is ready to apply the developed model to the validation set and to find out the final performance of the recommendation system.

First, the validation set is amended with the *year difference* necessary for the calculation of the time effect.

```
validation <- validation %>%
  mutate(year_m = str_extract(title, "\\(\\d{4}\\)")) %>%
  mutate(year_m = str_extract(year_m, "\\d{4}")) %>%
  mutate(year_m = as.numeric(year_m),
         year_r = year(as.POSIXct(timestamp, origin="1970-01-01")),
         year_diff = year_r - year_m)
```

Then the effects are recalculated using the whole edx set.

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

b_i_edx <- edx %>%
  group_by(movieId) %>%
  summarize(b_i_edx = sum(rating - mu_edx)/(n()+lambda1))

b_u_edx <- edx %>%
  left_join(b_i_edx, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u_edx = sum(rating - b_i_edx - mu_edx)/(n()+lambda2))

b_g_edx <- edx %>%
  left_join(b_i_edx, by = "movieId") %>%
  left_join(b_u_edx, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g_edx = sum(rating - b_i_edx - b_u_edx - mu_edx)/(n()+lambda3))

b_y_edx <- edx %>%
  left_join(b_i_edx, by = "movieId") %>%
  left_join(b_u_edx, by = "userId") %>%
  left_join(b_g_edx, by = "genres") %>%
  group_by(year_diff) %>%
  summarize(b_y_edx = sum(rating - b_i_edx - b_u_edx - b_g_edx - mu_edx)/(n()+lambda4))
```

Finally, the ratings of the validation set are predicted based on the developed model, corrected as described above and the final RMSE is calculated.

```
predicted_ratings_val <-  validation %>%
  left_join(b_i_edx, by = "movieId") %>%
  left_join(b_u_edx, by = "userId") %>%
  left_join(b_g_edx, by = "genres") %>%
  left_join(b_y_edx, by = "year_diff") %>%
  mutate(pred = mu_edx + b_i_edx + b_u_edx + b_g_edx + b_y_edx) %>%
  pull(pred)

predicted_ratings_val[predicted_ratings_val > 5] <- 5
predicted_ratings_val[predicted_ratings_val < 0.5] <- 0.5

rmse_val <- RMSE(validation$rating, predicted_ratings_val)
rmse_val
```

```
## [1] 0.8639311
```

The RMSE 0.8639311 of the prediction on the validation set is close to that of the test set and even smaller. This supports the validity of the developed model and of the applied approach.

3.5 Movie recommendation to a real user

The developed recommendation system was tried out with a real user in order to subjectively evaluate its performance additionally to the rmse value. The user received an excel table with 1902 movies (filtered by number of ratings more than 1000), including movie id and title. 168 movies (9%) were rated and the data were used to make the prediction. Export and import of *.xlsx files were performed with the package *xlsx*.

For the prediction, first, the movielens dataset was amended with the year of movie release.

```
movielens <- movielens %>%
  mutate(year_m = str_extract(title, "\\(\\d{4}\\)")) %>%
  mutate(year_m = str_extract(year_m, "\\d{4}")) %>%
  mutate(year_m = as.numeric(year_m))
```

Then, the following function was used in order to process the user modified table. It takes name of the user excel table, number of returned rows and year of rating as an input and returns a list with rmse of the prediction for the new user and with a dataframe with recommended movies arranged by predicted rating.

```
recommend <- function(user_table, output = 10, year = 2021){

  library(tidyverse)
  library(xlsx)

  user_df_raw <- read.xlsx(user_table, 1) # read the xlsx user table

  # extract genre and year columns for all movies from movielens dataset
  movielens_g_y <- movielens %>%
    select(movieId, genres, year_m) %>%
    group_by(movieId) %>%
    summarize(genres = first(genres),
              year_m = first(year_m))

  # prepare dataframe with user ratings to include all needed information
  user_df <- user_df_raw %>%
    mutate(userId = max(movielens$userId) + 1,
           rating = as.numeric(rating),
           year_r = year) %>%
    left_join(movielens_g_y, by = "movieId") %>%
    mutate(year_diff = year_r - year_m)

  # user dataframe including only rated movies
  user_df_rated <- user_df %>%
    filter(!is.na(rating))

  # add new user ratings to movielens dataset
  movielens_plus <- rbind(movielens, user_df_rated, fill = TRUE)

  # Update model parameters (effects)

  lambda1 <- 2 # lambdas stay unchanged
  lambda2 <- 5.25
  lambda3 <- 4
  lambda4 <- 340

  mu_movielens_plus <- mean(movielens_plus$rating)

  b_i_movielens_plus <- movielens_plus %>%
    group_by(movieId) %>%
    summarize(b_i_movielens_plus = sum(rating - mu_movielens_plus)/(n()+lambda1))

  b_u_movielens_plus <- movielens_plus %>%
```

```
  left_join(b_i_movielens_plus, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u_movielens_plus = sum(rating - b_i_movielens_plus -
                                       mu_movielens_plus)/(n()+lambda2))

b_g_movielens_plus <- movielens_plus %>%
  left_join(b_i_movielens_plus, by = "movieId") %>%
  left_join(b_u_movielens_plus, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g_movielens_plus = sum(rating - b_i_movielens_plus - b_u_movielens_plus -
                                       mu_movielens_plus)/(n()+lambda3))

b_y_movielens_plus <- movielens_plus %>%
  left_join(b_i_movielens_plus, by = "movieId") %>%
  left_join(b_u_movielens_plus, by = "userId") %>%
  left_join(b_g_movielens_plus, by = "genres") %>%
  group_by(year_diff) %>%
  summarize(b_y_movielens_plus = sum(rating - b_i_movielens_plus - b_u_movielens_plus -
                                       b_g_movielens_plus - mu_movielens_plus)/(n()+lambda4))

# Updated parameters are used to make new predictions

predicted_ratings <-  movielens_plus %>%
  left_join(b_i_movielens_plus, by = "movieId") %>%
  left_join(b_u_movielens_plus, by = "userId") %>%
  left_join(b_g_movielens_plus, by = "genres") %>%
  left_join(b_y_movielens_plus, by = "year_diff") %>%
  mutate(pred = mu_movielens_plus + b_i_movielens_plus + b_u_movielens_plus +
           b_g_movielens_plus + b_y_movielens_plus)

# predictions sorted out for the new user
predicted_ratings_user <- predicted_ratings %>%
  filter(userId == max(movielens$userId) + 1)

# rmse for the new user
rmse_new_user <- RMSE(user_df_rated$rating, predicted_ratings_user$pred)

# now prediction of rating for the not rated movies is made
rec_new_user <- user_df %>%
  left_join(b_i_movielens_plus, by = "movieId") %>%
  left_join(b_u_movielens_plus, by = "userId") %>%
  left_join(b_g_movielens_plus, by = "genres") %>%
  left_join(b_y_movielens_plus, by = "year_diff") %>%
  mutate(pred = mu_movielens_plus + b_i_movielens_plus + b_u_movielens_plus +
           b_g_movielens_plus + b_y_movielens_plus)

rec_new_user$pred[rec_new_user$pred > 5] <- 5
rec_new_user$pred[rec_new_user$pred < 0.5] <- 0.5

# recommended movies are arranged by predicted rating
rec_ratings <- rec_new_user %>%
  # defined number of rows is shown
  top_n(wt = pred, n = output) %>%
```

```
    arrange(desc(pred)) %>%
    select(title, rating, pred)

  result <- list(rec_ratings, rmse_new_user)

  return(result)
}
```

The function is then applied to the user table for the ratings provided in year 2021 and should return top ten recommended movies:

```
result <- recommend("Films_rm.xlsx", output = 10, year = 2021)
```

```
## Warning: package 'xlsx' was built under R version 4.1.1
```

```
knitr::kable(result[[1]])
```

title	rating	pred
Shawshank Redemption, The (1994)	NA	4.531073
Schindler's List (1993)	NA	4.440061
Usual Suspects, The (1995)	NA	4.433511
Seven Samurai (Shichinin no samurai) (1954)	NA	4.381767
Godfather: Part II, The (1974)	NA	4.377313
Rear Window (1954)	NA	4.376538
Casablanca (1942)	NA	4.373690
Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)	NA	4.372999
Lives of Others, The (Das Leben der Anderen) (2006)	NA	4.363551
One Flew Over the Cuckoo's Nest (1975)	4	4.361771

RMSE calculated for the ratings provided by the real user is:

```
result[[2]]
```

```
## [1] 0.839908
```

The real user ratings were predicted with RMSE smaller than that of the overall model performance. It can be explained by a relative low number of compared predictions. However, the top ten recommended movies seem to be quite generic, rather than user-tailored. In fact, the list is almost identical with the initial movielens top ten rated movie selection. The real user does not expect to give high ratings to the suggested movies based on the available information. The recommendation model should take more into account user preferences.

4 Conclusion

In this project, we have created a recommendation system for prediction of movie ratings. Information used for predictions is as following: user id, movie id, known ratings, year of rating, release year and genre of the movie.

The recommendation system is based on a linear model including movie, user, genre and time effects. The influence of the effects on the predicted rating is controlled via regularization. The effects were defined using the edx set, the penalty terms were defined using the training and test sets. The final evaluation was performed using the separate validation set.

The recommendation system provided prediction with the performance defined by RMSE: 0.8639311. Furthermore, it was used to make movie suggestions for a real user, however, the recommendation was not quite satisfying.

To further improve the recommendation system, more profound methods such as matrix factorization and hierarchical clustering could be applied in the future.