Movielens project

Predicting movie ratings by machine learning modeling

Contents

1 Introduction

1.1 Overview

This report present a summary of **Movielens project**, which is part of Data Science Professional Certificate led by HarvardX on edX platform. The Movielens project was part of final graded assessments.

In the following sections some parts of code have been highlighted, but for reference full code has been included in the Appendix. This report, as well as a script file, can be found on GitHub repository.

1.2 Scope of the project

The scope of this project is to find a way to predict user ratings for a movie, based on chosen set of predictors from provided dataset, using possible machine learning techniques. The calculations will be based on GroupLens research lab Movielens 10M dataset, which consists of 10 million ratings on 10 thousand movies, made by 72 thousand users. Each rating has a different set of predictor. The goal of the project is to find a prediction method that would generate residual mean squared error (RMSE) lower than 0.86490.

2 Method of analysis

2.1 Importing and checking movielens data set

Starting the project I've had to import the Movielens data and create useful data set out of it. As a first step I download the zip file form web and transform the two dat files inside it (movies and ratings) using fread and str_split_fixed functions from data.table and stringr packages (part of tidyverse pack). Transformed data frames are saved to *movies* and *ratings* respectively. Both data frames are then joined together by *movieId* to *movielens* data frame that will be used onward. All code for those action was provided in the project description.

After creating *movielens* data set, I inspected it using glimpse function. Our movielens data set consist of 6 parameters:

- userId (integer)
- movieId (double class)
- rating (double class)
- timestamp (integer)
- title (character)
- genres (character)

2.2 Choosing the parameters for modeling

Establishing that *rating* is our outcome, I've considered the influence of the rest of the five parameters on the expected outcome. The PCA analysis couldn't be used to help with cumulative variance explanation, as *movielens* dataset is too big for machine calculations possibilities (and another transformation on character parameters would be needed).

The parameters that have been chosen as primary ones were *userId*, *movieId* and *genres*. *timestamp* and *title* parameters have been used to create additional parameters in different form (see next section).

2.3 Additional parameters

Additional two parameters were created and included in *movielens* data set, which we suspect might be influencing the rating gave by users.

First one is *year_of_release*, indicating in which year the movie has been released. It has been created by taking the years from *title* parameter, as those years are included at the end of character string in the brackets. It has been transformed into double-class using str_sub function from stringr package, and as.numeric function.

Another support parameter created to *movielens* data frame is *rateday*, which is the day of the week the rating has been done (assuming it's happening on the same day the movie is watched), in scale 1-7 where 1 is Monday. This parameter is included based on assumption that users rate differently if they are relaxed over the weekend, or it's middle of stressful week. *rateday* has been added to movielens data set as transformation of *timestamp* parameter by as_datetime and wday functions from lubridate package.

```
movielens <- mutate(movielens,
                    year_of_release = as.numeric(str_sub(title, start=-5, end=-2)),
                    rateday=wday(as_datetime(timestamp),week_start = 1))
```

2.4 Creating training and test sets

For the modeling purposes I've created a training (*edx*) and test (*validation*) sets out of movielens data frame. Test set *validation* is consisting of 10% of original *movielens* set. This part was done with createDataPartition function from the caret package. Code for creating *edx* and *validations* sets was provided as part of project description.

As test set *validation* has to be used only for final check of prepared prediction model, I've created additional test and training sub-sets of training set *edx*, with same approach as used before, thus creating *edx_train* and *edx_test* variables. The test set was created using 50% of *edx* data set. From now on, until final prediction model is created, all trainigs and tests would be done on those data frames.

In both cases I've made sure that *userId*, *movieId*, *year_of_release*, *rateday* and *genres* appearing in test sets are also in training sets. This has been done by using semi_join function on those two sets, by this selected parameters.

2.5 Deciding on modeling approach

Given the size of dataset, more complex algorithms for predicting *ratings* outcome, like random forest, K-nearest neighbors (KNN) or any regression forms couldn't be used (train function was taking too much time on given machine to calculate). I've decided to follow step by step Naive Bayes approach on all 5 chosen features. This could be described by following equation

$$Y_{i,u,y,d,g} = \mu + b_i + b_u + b_y + b_d + b_g + \epsilon_{i,u,y,d,g}$$

where:

- $\mathrm{Y_{i,u,y,d,g}}$ - predicted movie rating
- μ - actual rating for all movies
- $\mathrm{b_i}$ - *movieId* effect
- $\mathrm{b_u}$ - *userId* effect
- $\mathrm{b_y}$ - *year_of_release* effect
- $\mathrm{b_d}$ - *rateday* effect
- $\mathrm{b_g}$ - *genres* effect
- $\epsilon_{\mathrm{i,u,y,d,g}}$ - independent errors

2.6 Naive model

First step to creating a naive prediction model is to estimate μ. This can be done by assuming it's close to the average of all outcomes (*rating*). Effect of *movieId* feature, as well as *userId*, *year_of_release*, *rateday* and *genres* effect has been calculated one by one as the difference of the predicted rating and all effects calculated before (and the independence error). Therefore the equation for *movieId* effect $\mathrm{b_i}$ is as follow

$$b_i \ = Y_{i,u,y,d,g} - \mu - \epsilon_{i,u,y,d,g}$$

and equation for *userId* $\mathrm{b_u}$ effect being

$$b_u = Y_{i,u,y,d,g} - \mu - b_i - \epsilon_{i,u,y,d,g}$$

and all other effects of $\mathrm{b_y}$, $\mathrm{b_d}$ and $\mathrm{b_g}$ created similarly.

All of those effects are calculated on edx_train train set, with code as followed:

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

b_i <- edx_train %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating-mu))

b_u <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u=mean(rating-mu-b_i))

b_y <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(year_of_release) %>%
  summarise(b_y=mean(rating-mu-b_i-b_u))

b_d <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_y, by="year_of_release") %>%
  group_by(rateday) %>%
  summarise(b_d=mean(rating-mu-b_i-b_u-b_y))

b_g <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_y, by="year_of_release") %>%
  left_join(b_d, by="rateday") %>%
  group_by(genres) %>%
  summarise(b_g=mean(rating-mu-b_i-b_u-b_y-b_d))
```

The effects calculated that way are then used to calculate predictions on movie rate $\mathrm{Y_{i,u,y,d,g}}$. For this purpose, I've created *predictions* function, which will be used through all project, also for final testing on *validations* test set.

```r
predictions<- function(x,i,u,y,d,g){
  x %>%
    left_join(i, by="movieId") %>%
    left_join(u, by="userId") %>%
    left_join(y, by="year_of_release") %>%
    left_join(d, by="rateday") %>%
    left_join(g, by="genres") %>%
    mutate(pred=mu+b_i+b_u+b_y+b_d+b_g) %>%
    pull(pred)
}

pred <- predictions(edx_test, b_i, b_u, b_y, b_d, b_g)
```

Creating prediction vector, I was able to test out what would be the residual mean squared error (RMSE) of current model. This could be calculated by following

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

where:

- $y_{i,u,y,d,g}$ - actual rating for a movie with selected features
- $\hat{y}_{i,u,y,d,g}$ - predicted rating for a movie with selected features

equation, which in this project can be translated to code:

```r
rmse <- sqrt(mean((pred-edx_test$rating)^2))
```

The RMSE obtained from naive model, although calculated based on train and test subsets and to proper whole *edx* set, is equal 0.86907, so above 0.86490 threshold required by the project. Therefore current model is not enough, and further modification of it is required.

2.7 Regularized model

To make RMSE of the model smaller, all five features were adjusted by separate regularization parameters. Using regularization parameters (named *alpha*, *lambda delta*, *kappa* and *omega*) allowed to influence total variability of effect sizes. This can be done by transforming RMSE equation with penalized regression (for controlling variety), with focus on minimizing paramter effect.

For example for b_i it would be

$$\frac{1}{N}\sum_{i,u,y,d,g}(\hat{y}_{i,u,y,d,g} - \mu - b_i)^2 + \lambda\sum_i b_i^2$$

transformed into

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

2.7.1 Regularized movie effect

For regularized movie effect (b_i), regularization has been done with α parameter. The best value of α has been chosen using sapply function on α values from 0:10 (in the code it has been narrowed down for the sake of calculation speed).

Value of b_i has been calculated using following code

```
b_i <- edx_train %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating-mu)/(n()+alpha))
```

and the rest of effects has been calculated using basic naive approach.

The best value of α has been determined as 3.85.

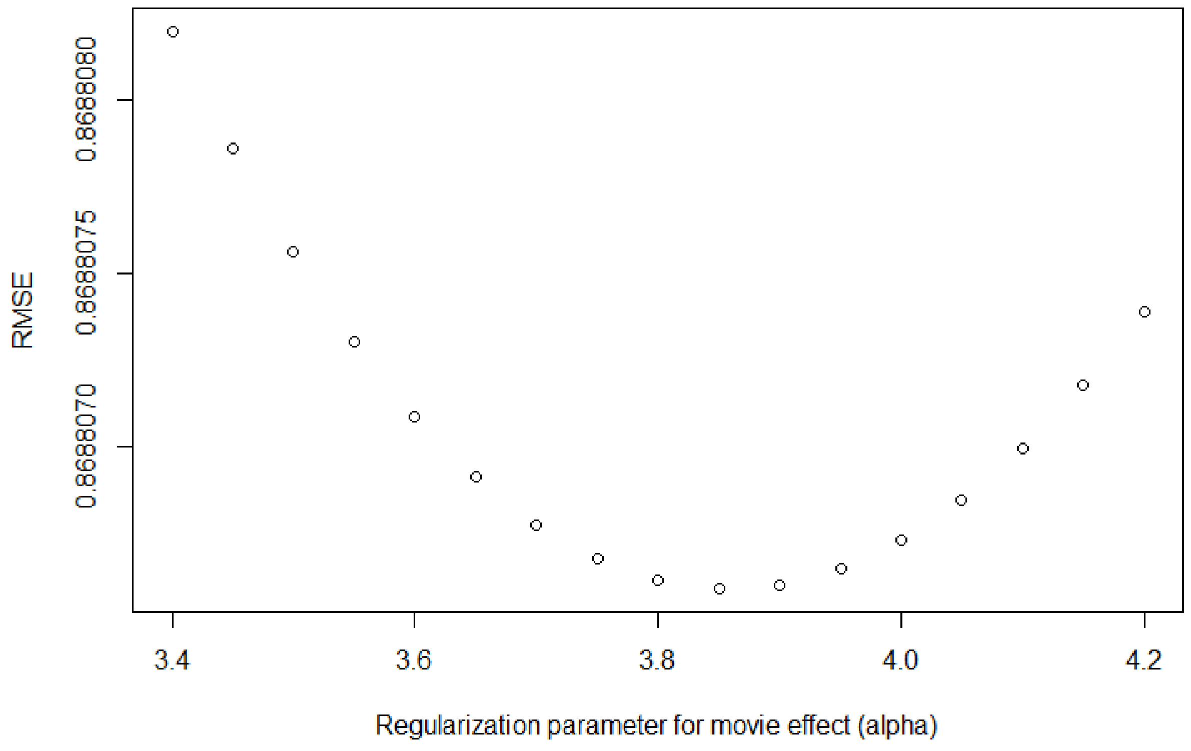


Figure 1: RMSE vs. Alpha plot

For this value of α RMSE has been calculated as 0.86881, which is 0.03% improvement from fully naive approach.

2.7.2 Regularized user effect

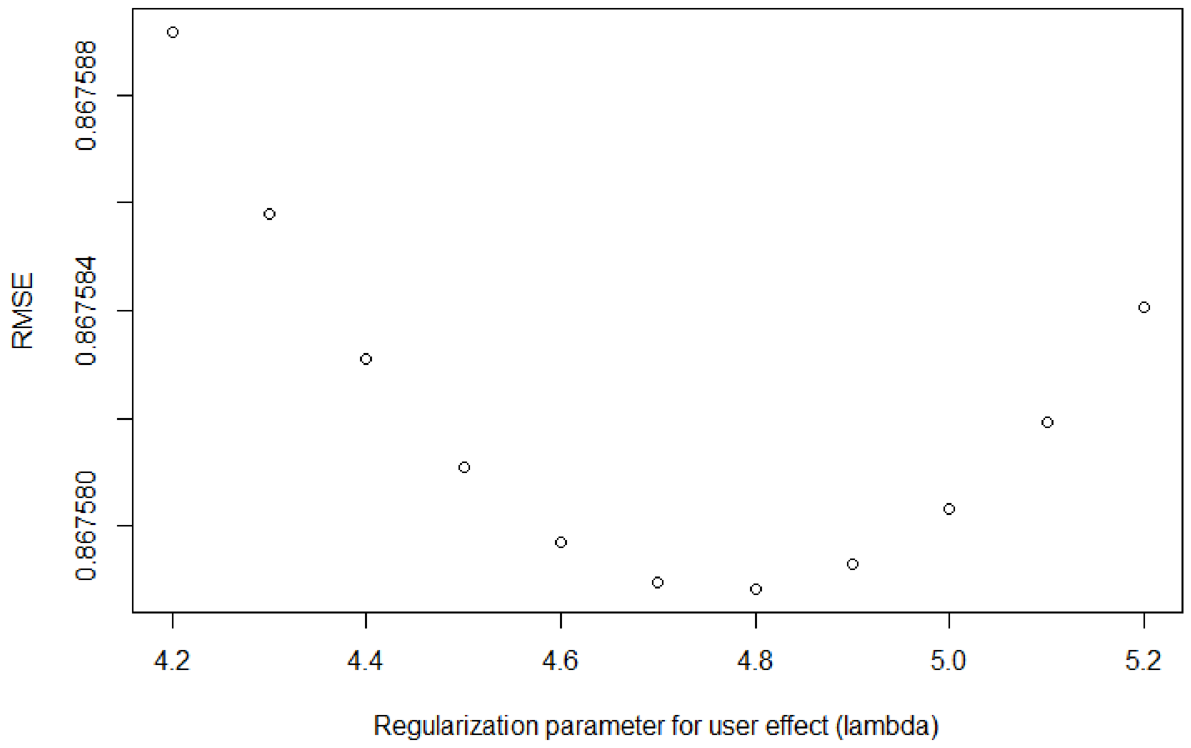
For regularized user effect (b_u), regularization has been done with λ parameter. The best value of λ has been chosen using sapply function on λ values from 0:10 (in the code it has been narrowed down for the sake of calculation speed).

Value of b_u has been calculated using following code

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

and the rest of effects (besides movie effect b_i) has been calculated using basic naive approach. Movie effect used for this regularization has been calculated with previously defined α regularization parameter of 3.85.

The best value of λ has been determined as 4.8.



For this value of λ RMSE has been calculated as 0.86758, which is 0.17% improvement from fully naive approach.

2.7.3 Regularized year of movie release effect

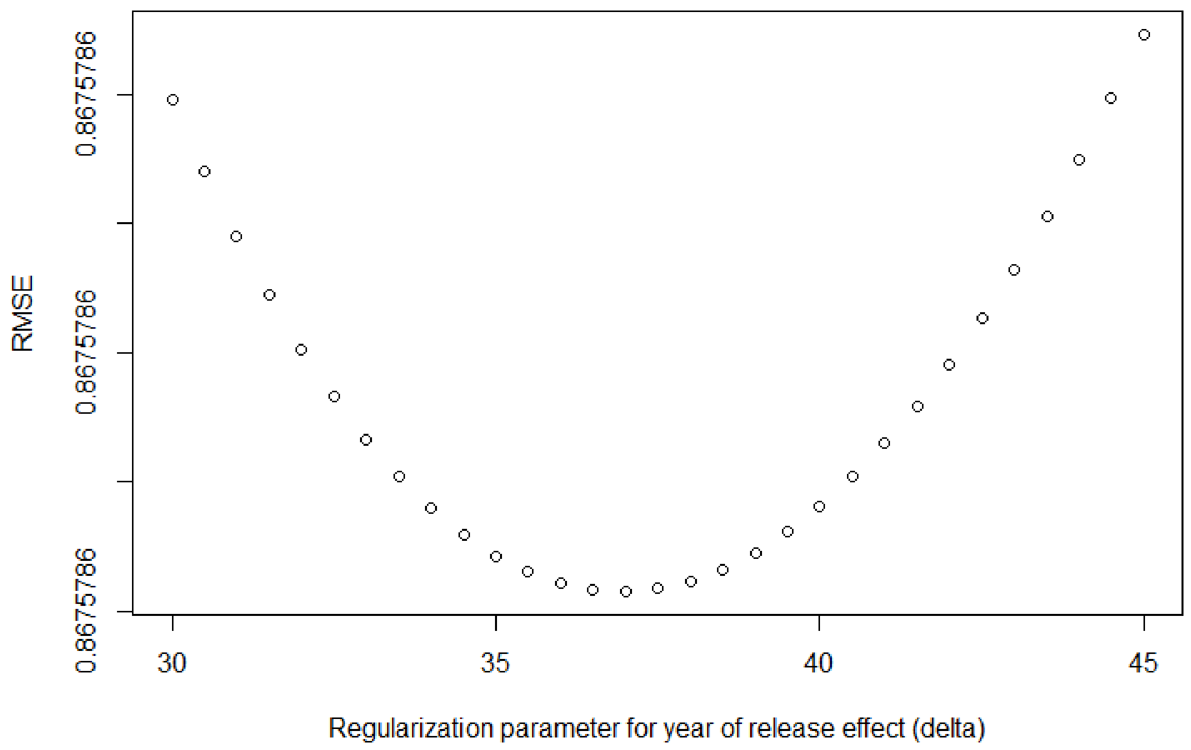
For regularized year of movie release effect (b_y), regularization has been done with δ parameter. The best value of δ has been chosen using sapply function on δ values from 0:50 (in the code it has been narrowed down for the sake of calculation speed).

Value of b_y has been calculated using following code

```
b_y <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(year_of_release) %>%
    summarise(b_y=sum(rating-mu-b_i-b_u)/(n()+delta))
```

and the rest of effects (besides movie effect b_i and user effect b_u) has been calculated using basic naive approach. Movie effect used for this regularization has been calculated with previously defined α regularization parameter of 3.85, and user effect used with λ of 4.8.

The best value of δ has been determined as 37.



For this value of δ RMSE has been calculated as 0.86758, which is 0.17% improvement from fully naive approach (and not much different from previous step).

2.7.4 Regularized rating day of the week effect

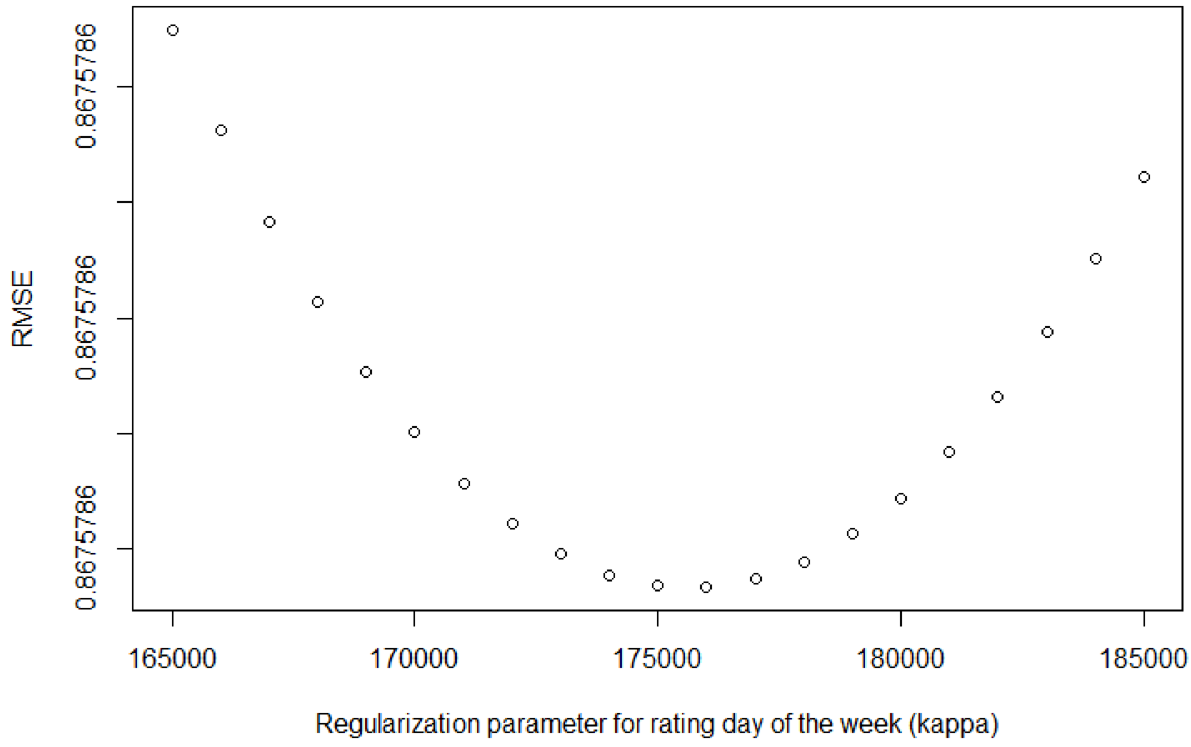
For regularized rating day of the week effect (b_d), regularization has been done with κ parameter. The best value of κ has been chosen using sapply function on κ values from 0:1,000,000 (in the code it has been narrowed down for the sake of calculation speed).

Value of b_d has been calculated using following code

```
b_d <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_y, by="year_of_release") %>%
    group_by(rateday) %>%
    summarise(b_d=sum(rating-mu-b_i-b_u-b_y)/(n()+kappa))
```

and the rest of effects followed previous approach. Genre effect has been calculated using naive approach, and movie, user and year of movie release effects used for this regularization has been calculated with previously defined α (3.85), λ (4.8) and δ (37) regularization parameters.

The best value of κ has been determined as 176,000.



For this value of κ RMSE has been calculated as 0.86758, which is 0.17% improvement from fully naive approach (and only slightly different from two previous steps).

2.7.5 Regularized movie genre effect

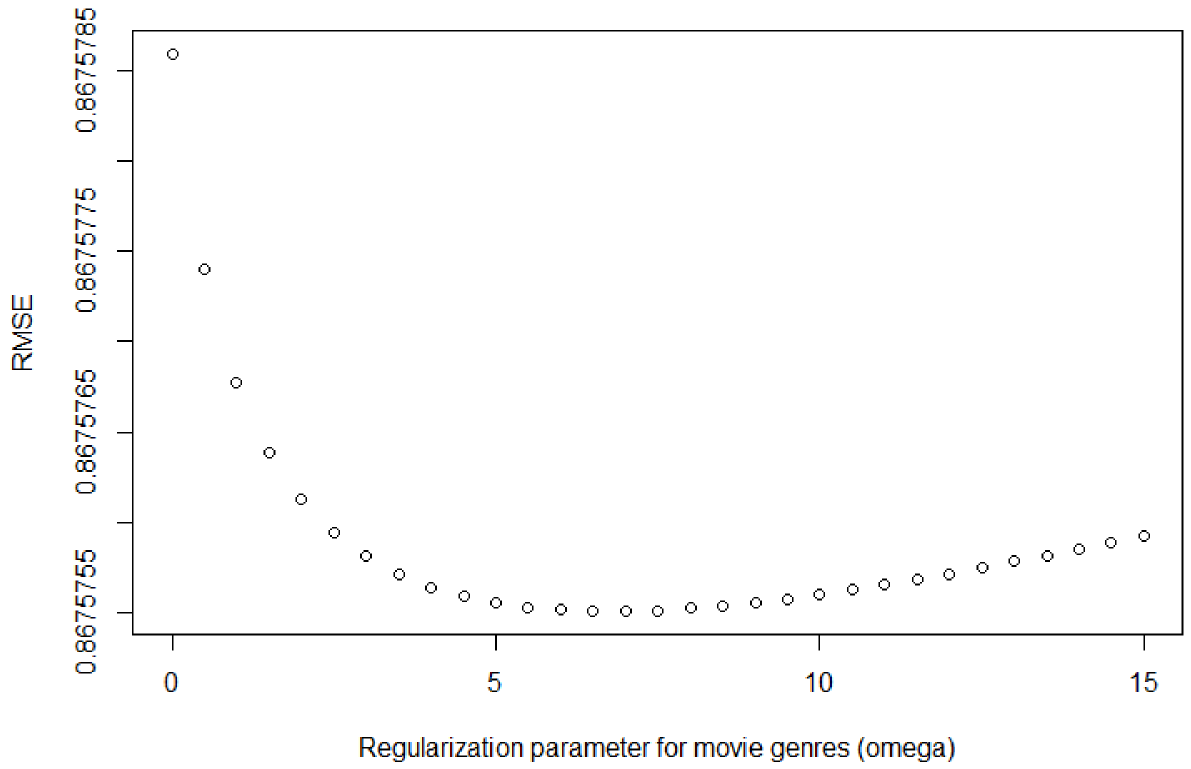
For regularized movie genre effect (b_g), regularization has been done with ω parameter. The best value of ω has been chosen using sapply function on ω values from 0:100 (in the code it has been narrowed down for the sake of calculation speed).

Value of b_g has been calculated using following code

```
b_g <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_y, by="year_of_release") %>%
    left_join(b_d, by="rateday") %>%
    group_by(genres) %>%
    summarise(b_g=sum(rating-mu-b_i-b_u-b_y)/(n()+omega))
```

The rest of effects were calculatedusing their regularized versions. Movie, user, year of movie release and rating day of the week effects were using previously defined for them the best regularization parameters values, with α as 3.85, λ as 4.8, δ as 37 and κ as 176,000.

The best value of ω has been determined as 6.5.



For this value of ω RMSE has been calculated as 0.86758, which is 0.17% improvement from fully naive approach (and only slightly different from last three steps).

2.8 Results on edX test set

Seeing that regularization attempts on last three effects didn't improve RMSE significantly, I've decided to stop at this point. Before calculating final RMSE based on prediction from actual test sets (*validation*) from the model trained on actual train set (full *edx*), below is the summary of all RMSEs calculated so far on sub-sets of edx.

Table 1: RMSE calculated on edx sub-sets, depending on the method used

Method	RMSE
Naive model	0.86907
Naive model + regularized movie effect	0.86881
Naive model + regularized movie and user effects	0.86758
Naive model + regularized movie, user and year of release effects	0.86758
Naive model + regularized movie, user, year of release and rating day of the week effects	0.86758
All 5 effects regularized	0.86758

3 Results

Now that I've created my model, I use the same approach as with 5 regularized effects and their corresponding regularization parameters, to train it on full *edx* data set.

After training the model, I've calculated predictions based on *validation* as a test set. Having that calculated, I could obtain proper and final RMSE, which was equal to:

$$RMSE = 0.86426$$

As it is below targeted 0.86490 the project is completed at this stage.

4 Conclusions

The biggest difference in obtaining lower RMSE has been regularizing *movieId* effect on top of already created naive model with other 4 effects. Regularizing other effects was still improving model, although not significantly, as their variances has not been vastly different from the beginning.

Further learning methods (like matrix factorization or hierarchical clustering) might be considered to bring RMSE even lower, but this would need to be treated carefully not to risk overtraining.

5 Appendix

5.1 Session info

```
sessionInfo()
```

```
## R version 4.0.1 (2020-06-06)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 18362)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=Polish_Poland.1250  LC_CTYPE=Polish_Poland.1250
## [3] LC_MONETARY=Polish_Poland.1250 LC_NUMERIC=C
## [5] LC_TIME=Polish_Poland.1250
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] compiler_4.0.1  magrittr_1.5    tools_4.0.1     htmltools_0.5.0
##  [5] yaml_2.2.1      stringi_1.4.6   rmarkdown_2.3   highr_0.8
##  [9] knitr_1.29      stringr_1.4.0   xfun_0.16       digest_0.6.25
## [13] rlang_0.4.7     evaluate_0.14
```

5.2 Full code

```
############################################
# Download and prepare the movielens dataset
```

```r
#############################################

# 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)
library(lubridate)
options(digits=5)

# 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")

##############################################################################################
# Creating additonal columns in movielens (movie release year and weekday of the rating time)
##############################################################################################

glimpse(movielens)

movielens <- mutate(movielens, year_of_release = as.numeric(str_sub(title, start=-5, end=-2)),
                    rateday=wday(as_datetime(timestamp),week_start = 1))

##########################################################
# Create edx set, validation set (final hold-out test set)
##########################################################

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")

test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId, movieId, year_of_release and rateday in validation set are also in edx set
```

```r
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId") %>%
  semi_join(edx, by="year_of_release") %>%
  semi_join(edx, by="rateday") %>%
  semi_join(edx, by="genres")

# 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)

##################################################################################
# Creating train and test sets out of edX set, for sake of modelling aproach tests
##################################################################################

# Creating a test and training set for edx set
edx_index <- createDataPartition(y = edx$rating, times = 1, p = 0.5, list = FALSE)
edx_test <- edx %>% slice(edx_index)
edx_train <- edx %>% slice(-edx_index)

# Making sure that both edx_test and edx_train have same set of userId, movieId, year_of_release and ra
edx_test <- edx_test %>%
  semi_join(edx_train, by = "movieId") %>%
  semi_join(edx_train, by = "userId") %>%
  semi_join(edx_train, by="year_of_release") %>%
  semi_join(edx_train, by="rateday") %>%
  semi_join(edx_train, by="genres")

######################################
# Model creation - basic naive model
######################################

# Calculating average of all ratings, equal to mu
mu <- mean(edx_train$rating)

# Estimating movie effect
b_i <- edx_train %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating-mu))

# Estimating user effect (+ left joining with movie_ef to have b_i accessible)
b_u <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u=mean(rating-mu-b_i))

# Estimating release year effect (+ left joining with movie_ef and user_ef to have b_i and b_u accessib
b_y <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(year_of_release) %>%
  summarise(b_y=mean(rating-mu-b_i-b_u))
```

```r
# Estimating rating day of the week effect (+ left joining to have b_i, b_u and b_y accessible)
b_d <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_y, by="year_of_release") %>%
  group_by(rateday) %>%
  summarise(b_d=mean(rating-mu-b_i-b_u-b_y))

# Estimating rating day of the genres (+ left joining to have b_i, b_u, b_y and b_d accessible)
b_g <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_y, by="year_of_release") %>%
  left_join(b_d, by="rateday") %>%
  group_by(genres) %>%
  summarise(b_g=mean(rating-mu-b_i-b_u-b_y-b_d))

# Creating a predictions function
predictions<- function(x,i,u,y,d,g){
  x %>%
    left_join(i, by="movieId") %>%
    left_join(u, by="userId") %>%
    left_join(y, by="year_of_release") %>%
    left_join(d, by="rateday") %>%
    left_join(g, by="genres") %>%
    mutate(pred=mu+b_i+b_u+b_y+b_d+b_g) %>%
    pull(pred)
}

# Making predictions on average + movie effect + user effect + release year effect + rate day of the we
pred <- predictions(edx_test, b_i, b_u, b_y, b_d, b_g)
rmse_naive <- sqrt(mean((pred-edx_test$rating)^2))
print(c("The RMSE for naive model is:", rmse_naive), quote=FALSE, digits = 5)

############################################
# Model creation - regularization parameters
############################################

# Regularizing movie effect, with finding alpha that minimizes rmse. Previously narrowed it down from 0
alpha <- seq(3.4,4.2,0.05)

rmse_reg_a <- sapply(alpha, function(a){
  b_i <- edx_train %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating-mu)/(n()+a))
  b_u <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarise(b_u=mean(rating-mu-b_i))
  b_y <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(year_of_release) %>%
    summarise(b_y=mean(rating-mu-b_i-b_u))
  b_d <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_y, by="year_of_release") %>%
    group_by(rateday) %>%
    summarise(b_d=mean(rating-mu-b_i-b_u-b_y))
  b_g <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_y, by="year_of_release") %>%
    left_join(b_d, by="rateday") %>%
    group_by(genres) %>%
    summarise(b_g=mean(rating-mu-b_i-b_u-b_y-b_d))
  pred <- predictions(edx_test, b_i, b_u, b_y, b_d, b_g)
  sqrt(mean((pred-edx_test$rating)^2))
})

plot(alpha,rmse_reg_a, xlab ="Regularization parameter for movie effect (alpha)", ylab="RMSE")

alpha <- alpha[which.min(rmse_reg_a)]

print(c("The alpha parameter to get the smallest RMSE is",alpha), quote = FALSE)

b_i <- edx_train %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating-mu)/(n()+alpha))

b_u <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u=mean(rating-mu-b_i))

b_y <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(year_of_release) %>%
  summarise(b_y=mean(rating-mu-b_i-b_u))

b_d <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_y, by="year_of_release") %>%
  group_by(rateday) %>%
  summarise(b_d=mean(rating-mu-b_i-b_u-b_y))

b_g <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_y, by="year_of_release") %>%
  left_join(b_d, by="rateday") %>%
  group_by(genres) %>%
  summarise(b_g=mean(rating-mu-b_i-b_u-b_y-b_d))
```

```
pred <- predictions(edx_test, b_i, b_u, b_y, b_d, b_g)

rmse_moviereg <- sqrt(mean((pred-edx_test$rating)^2))

print(c("The RMSE for regularized movie effect + simple user, year of release and rating day of the weel

# Regularizing user effect, with finding lambda that minimizes rmse. Previously narrowed it down from 0
lambda <- seq(4.2,5.2,0.1)
rmse_reg_l <- sapply(lambda, function(l){
  b_u <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarise(b_u=sum(rating-mu-b_i)/(n()+l))
  b_y <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(year_of_release) %>%
    summarise(b_y=mean(rating-mu-b_i-b_u))
  b_d <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_y, by="year_of_release") %>%
    group_by(rateday) %>%
    summarise(b_d=mean(rating-mu-b_i-b_u-b_y))
  b_g <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_y, by="year_of_release") %>%
    left_join(b_d, by="rateday") %>%
    group_by(genres) %>%
    summarise(b_g=mean(rating-mu-b_i-b_u-b_y-b_d))
  pred <- predictions(edx_test, b_i, b_u, b_y, b_d, b_g)
  sqrt(mean((pred-edx_test$rating)^2))
})
plot(lambda,rmse_reg_l, xlab ="Regularization parameter for user effect (lambda)", ylab="RMSE")
lambda <- lambda[which.min(rmse_reg_l)]
print(c("The lambda parameter to get the smallest RMSE is",lambda), quote = FALSE)

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

b_y <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(year_of_release) %>%
  summarise(b_y=mean(rating-mu-b_i-b_u))

b_d <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_y, by="year_of_release") %>%
  group_by(rateday) %>%
  summarise(b_d=mean(rating-mu-b_i-b_u-b_y))

b_g <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_y, by="year_of_release") %>%
  left_join(b_d, by="rateday") %>%
  group_by(genres) %>%
  summarise(b_g=mean(rating-mu-b_i-b_u-b_y-b_d))

pred <- predictions(edx_test, b_i, b_u, b_y, b_d, b_g)

rmse_userreg <- sqrt(mean((pred-edx_test$rating)^2))
print(c("The RMSE for regularized movie and user effect + simple year of release and rating day of the

# Regularizing year_of_release effect, with finding delta that minimizes rmse. Previously narrowed it d
delta <- seq(30,45,0.5)
rmse_reg_d <- sapply(delta, function(d){
  b_y <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(year_of_release) %>%
    summarise(b_y=sum(rating-mu-b_i-b_u)/(n()+d))
  b_d <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_y, by="year_of_release") %>%
    group_by(rateday) %>% summarise(b_d=mean(rating-mu-b_i-b_u-b_y))
  b_g <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_y, by="year_of_release") %>%
    left_join(b_d, by="rateday") %>%
    group_by(genres) %>%
    summarise(b_g=mean(rating-mu-b_i-b_u-b_y-b_d))
  pred <- predictions(edx_test, b_i, b_u, b_y, b_d, b_g)
  sqrt(mean((pred-edx_test$rating)^2))
})
plot(delta,rmse_reg_d, xlab ="Regularization parameter for year of release effect (delta)", ylab="RMSE"
delta <- delta[which.min(rmse_reg_d)]
print(c("The delta parameter to get the smallest RMSE is",delta), quote = FALSE)

b_y <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(year_of_release) %>%
  summarise(b_y=sum(rating-mu-b_i-b_u)/(n()+delta))

b_d <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_y, by="year_of_release") %>%
  group_by(rateday) %>% summarise(b_d=mean(rating-mu-b_i-b_u-b_y))
```

```
b_g <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_y, by="year_of_release") %>%
  left_join(b_d, by="rateday") %>%
  group_by(genres) %>%
  summarise(b_g=mean(rating-mu-b_i-b_u-b_y-b_d))

pred <- predictions(edx_test, b_i, b_u, b_y, b_d, b_g)

rmse_yearreg <- sqrt(mean((pred-edx_test$rating)^2))
print(c("The RMSE for regularized movie, user and year of release effect + simple rating day of the weel

# Regularizing rating day of the week effect, with finding kappa that minimizes rmse. Previously narrow
kappa <- seq(165000,185000,1000)
rmse_reg_k <- sapply(kappa, function(k){
  b_d <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_y, by="year_of_release") %>%
    group_by(rateday) %>%
    summarise(b_d=sum(rating-mu-b_i-b_u-b_y)/(n()+k))
  b_g <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_y, by="year_of_release") %>%
    left_join(b_d, by="rateday") %>%
    group_by(genres) %>%
    summarise(b_g=mean(rating-mu-b_i-b_u-b_y-b_d))
  pred <- predictions(edx_test, b_i, b_u, b_y, b_d, b_g)
  sqrt(mean((pred-edx_test$rating)^2))
})
plot(kappa,rmse_reg_k, xlab ="Regularization parameter for rating day of the week (kappa)", ylab="RMSE"
kappa <- kappa[which.min(rmse_reg_k)]
print(c("The kappa parameter to get the smallest RMSE is",delta), quote = FALSE)

b_d <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_y, by="year_of_release") %>%
  group_by(rateday) %>%
  summarise(b_d=sum(rating-mu-b_i-b_u-b_y)/(n()+kappa))

b_g <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_y, by="year_of_release") %>%
  left_join(b_d, by="rateday") %>%
  group_by(genres) %>%
  summarise(b_g=mean(rating-mu-b_i-b_u-b_y-b_d))

pred <- predictions(edx_test, b_i, b_u, b_y, b_d, b_g)

rmse_ratedayreg <- sqrt(mean((pred-edx_test$rating)^2))
```

```
print(c("The RMSE for regularized movie, user, year of release effect and rating day of the week effect

# Regularizing rating day of the week effect, with finding omega that minimizes rmse. Previously narrow
omega <- seq(0,15,0.5)
rmse_reg_o <- sapply(omega, function(o){
  b_g <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_y, by="year_of_release") %>%
    left_join(b_d, by="rateday") %>%
    group_by(genres) %>%
    summarise(b_g=sum(rating-mu-b_i-b_u-b_y)/(n()+o))
  pred <- predictions(edx_test, b_i, b_u, b_y, b_d, b_g)
  sqrt(mean((pred-edx_test$rating)^2))
})
plot(omega,rmse_reg_o, xlab ="Regularization parameter for movie genres (omega)", ylab="RMSE")
omega <- omega[which.min(rmse_reg_o)]
print(c("The omega parameter to get the smallest RMSE is",omega), quote = FALSE)

b_g <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_y, by="year_of_release") %>%
  left_join(b_d, by="rateday") %>%
  group_by(genres) %>%
  summarise(b_g=sum(rating-mu-b_i-b_u-b_y)/(n()+omega))

pred <- predictions(edx_test, b_i, b_u, b_y, b_d, b_g)

rmse_genrereg <- sqrt(mean((pred-edx_test$rating)^2))
print(c("The RMSE for regularized movie, user, year of release effect and rating day of the week effect

#####################################
# Final test of the prediction model
#####################################

# Training the model on edx set
mu <- mean(edx$rating)
b_i <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating-mu)/(n()+alpha))
b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u=sum(rating-mu-b_i)/(n()+lambda))
b_y <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(year_of_release) %>%
  summarise(b_y=sum(rating-mu-b_i-b_u)/(n()+delta))
b_d <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_y, by="year_of_release") %>%
  group_by(rateday) %>%
  summarise(b_d=sum(rating-mu-b_i-b_u-b_y)/(n()+kappa))
b_g <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_y, by="year_of_release") %>%
  left_join(b_d, by="rateday") %>%
  group_by(genres) %>%
  summarise(b_g=sum(rating-mu-b_i-b_u-b_y)/(n()+omega))

# Generating predictions on validation set
pred <- predictions(validation, b_i, b_u, b_y, b_d, b_g)

# Calculating final RMSE
rmse_val <- sqrt(mean((pred-validation$rating)^2))
print(c("Final RMSE is:", rmse_val), quote=FALSE, digits = 5)
```