Movielens. Recommendation system and regularization

Maryna Shut

2023-21-01

Introduction

A recommendation system is a tool that suggests content to users based on their past preferences and ratings (Irizarry 2019). These systems are commonly used by companies to successfully promote themselves on a market: the more useful a platform is, the more time a user spends on the platform, enjoys using it and shares it with friends, the better it is for the business. Therefore, elaborating an solid model that could efficiently predict a user's choice is an important and useful skill and task to perform. In this project I will elaborate a recommendation system based on historical data of users' choice. The accuracy of this model will be evaluated by RMSE.

RMSE is a loss function, which is a technique of establishing the accuracy of a model.

The whole analysis will consist of several parts:

- The inital HarvardX code
- My code for exploring the data
- Building the model with a train set
- Finding the optimal lambda value and building the regularization regression
- Applying the regularization to the final_holdout_test and finding RMSE

HarvardX initial code

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

```
## Loading required package: tidyverse

## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0      v purrr   0.3.5
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: caret
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
```

```
##     lift
```

```
library(tidyverse)
library(caret)

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

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

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

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
# set.seed(1) # if using R 3.5 or earlier
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 final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
```

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

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

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

The analysis

STEP 1: Data preparation

In this part I will prepare the data by splitting it into train and test set.

```
# -- splitting data into train and test

set.seed(1)
index <- createDataPartition(edx$rating, p=0.8, times = 1, list = FALSE)
train_set <- edx %>% slice(index)
test_set <- edx %>% slice(-index)
```

STEP 2: Exploring the data

I will start exploring the data by simply eyeballing the top rows.Then I will run the summary and check the structure of the data. These are the three standard checks I would normally do with a new dataset.

```
head(train_set)
```

```
##   userId movieId rating timestamp                         title
## 2      1     185      5 838983525                Net, The (1995)
## 4      1     292      5 838983421                Outbreak (1995)
## 5      1     316      5 838983392                Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474        Flintstones, The (1994)
## 8      1     356      5 838983653            Forrest Gump (1994)
##                           genres
## 2        Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5      Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7      Children|Comedy|Fantasy
## 8     Comedy|Drama|Romance|War
```

```
summary(train_set)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18123   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35729   Median : 1833   Median :4.000   Median :1.035e+09
##  Mean   :35869   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title                genres
##  Length:7200045     Length:7200045
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
str(train_set)
```

```
## 'data.frame':    7200045 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  185 292 316 329 355 356 362 364 370 377 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838983525 838983421 838983392 838983392 838984474 838983653 838984885 838983707 83
##  $ title    : chr  "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" "Star Trek: Generations (19
##  $ genres   : chr  "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Adventure|Sci-Fi"
```

```
dim(train_set)
```

```
## [1] 7200045       6
```

From the start I can see that the dataset has seven columns containing characters, integers, factors. Additionally I can run describe() function from Hmisc library that is commonly used for data analysis.

```
describe(train_set)
```

```
## train_set
##
##  6  Variables      7200045  Observations
## ---------------------------------------------------------------------------------
## userId
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##  7200045        0    69878        1    35869    23770     3810     7520
##      .25      .50      .75      .90      .95
##    18123    35729    53607    64480    68097
##
## lowest :     1     2     3     4     5, highest: 71563 71564 71565 71566 71567
## ---------------------------------------------------------------------------------
## movieId
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##  7200045        0    10647        1     4120     5533      110      260
##      .25      .50      .75      .90      .95
##      648     1833     3624     6502     8916
##
## lowest :     1     2     3     4     5, highest: 65088 65091 65126 65130 65133
## ---------------------------------------------------------------------------------
## rating
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##  7200045        0       10    0.958    3.512    1.166      1.5      2.0
##      .25      .50      .75      .90      .95
##      3.0      4.0      4.0      5.0      5.0
##
## lowest : 0.5 1.0 1.5 2.0 2.5, highest: 3.0 3.5 4.0 4.5 5.0
##
## Value          0.5     1.0     1.5     2.0     2.5     3.0     3.5     4.0
## Frequency    68184  276851   85179  569658  266610 1696039  632939 2071105
## Proportion   0.009   0.038   0.012   0.079   0.037   0.236   0.088   0.288
```

```
##
## Value          4.5     5.0
## Frequency   421389 1112091
## Proportion   0.059   0.154
## ---------------------------------------------------------------------------------
## timestamp
##          n   missing  distinct      Info      Mean       Gmd       .05       .10
##    7200045         0   5438784         1 1.033e+09 133331395 8.395e+08 8.506e+08
##        .25       .50       .75       .90       .95
## 9.468e+08 1.035e+09 1.127e+09 1.188e+09 1.213e+09
##
## lowest :  789652009  822873600  823185196  823185197  823185198
## highest: 1231131127 1231131132 1231131137 1231131142 1231131303
## ---------------------------------------------------------------------------------
## title
##         n  missing distinct
##   7200045        0    10646
##
## lowest : ...All the Marbles (a.k.a. The California Dolls) (1981)    ...And God Created Woman (Et Dieu
## highest: Zoot Suit (1981)                                           Zorba the Greek (Alexis Zorbas)
## ---------------------------------------------------------------------------------
## genres
##         n  missing distinct
##   7200045        0      797
##
## lowest : (no genres listed)                                 Action
## highest: Thriller|War                                       Thriller|Western
## ---------------------------------------------------------------------------------
```

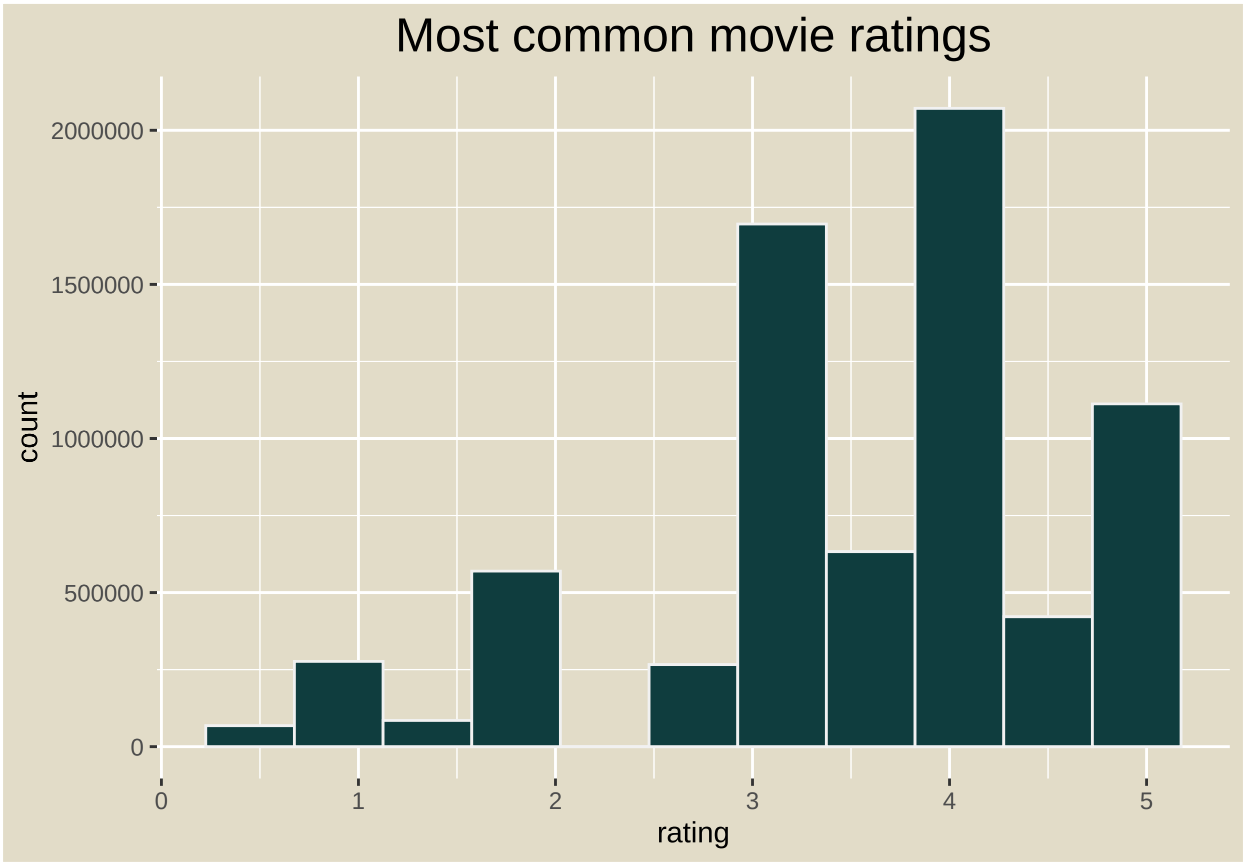
This function shows apart from the data we have already seen with summary(), information on missing values. In this case they are equal to zero. I can additionally check to make sure the dataset has no missing values by running this code:

```
any(is.na(train_set))
```

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

To further explore the data, I can create some visualizations:

```
train_set %>% ggplot() + geom_histogram(aes(rating), binwidth = 0.45, fill = '#0F3D3E', color = '#F1F1F
  ggtitle('Most common movie ratings') +
  theme(plot.background = element_rect(fill = '#E2DCC8'),
        panel.background = element_rect(fill = '#E2DCC8'),
        plot.title = element_text(size = 18, hjust = 0.5))
```



What this plot tells us is that 4, 4 and 5 are the most common ratings, with 4 being the most common.

```
# checking all the unique rating values
sort(unique(train_set$rating))
```

```
##  [1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```
# checking the number of unique
sum(unique(train_set$userId))
```

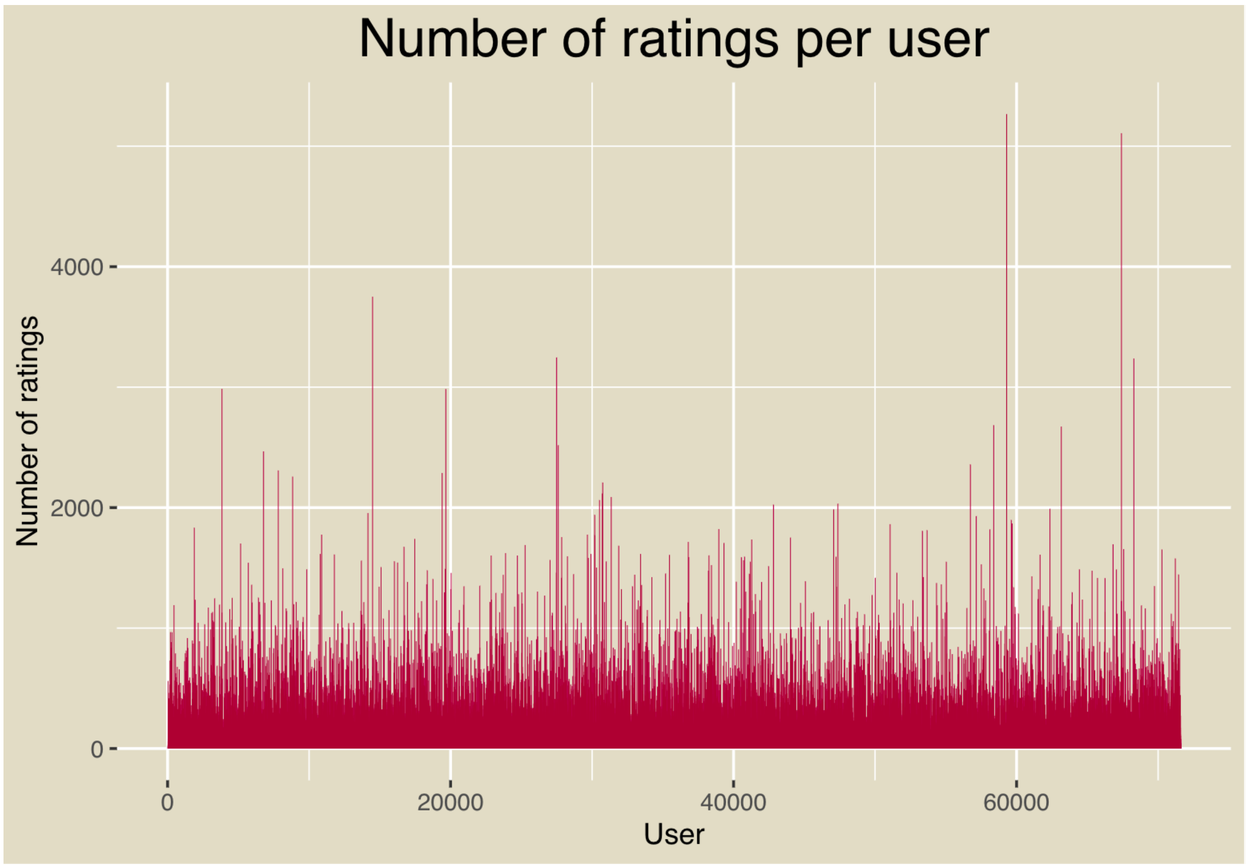
```
## [1] 2500389787
```

```
sum(unique(train_set$movieId))
```

```
## [1] 138571464
```

The number of users is a lot lower than the number of the movies. Now I'm interested to see how frequently users give ratings to movies and whether it is evenly distributed or not.

```
train_set  %>% group_by(userId) %>% summarise (n = n()) %>% ggplot() +
  geom_col(aes(userId, n), fill = "#A10035") + ylab('Number of ratings') +
  xlab("User") + ggtitle("Number of ratings per user") +
  theme(plot.title = element_text(size = 20, hjust = 0.5),
        plot.background = element_rect(fill = "#E2DCC8"),
        panel.background = element_rect(fill = "#E2DCC8"))
```



The plot clearly shows that the number of ratings is uneavenly distributed, which means that there are users who rate lots of movies, and others who rate only some.

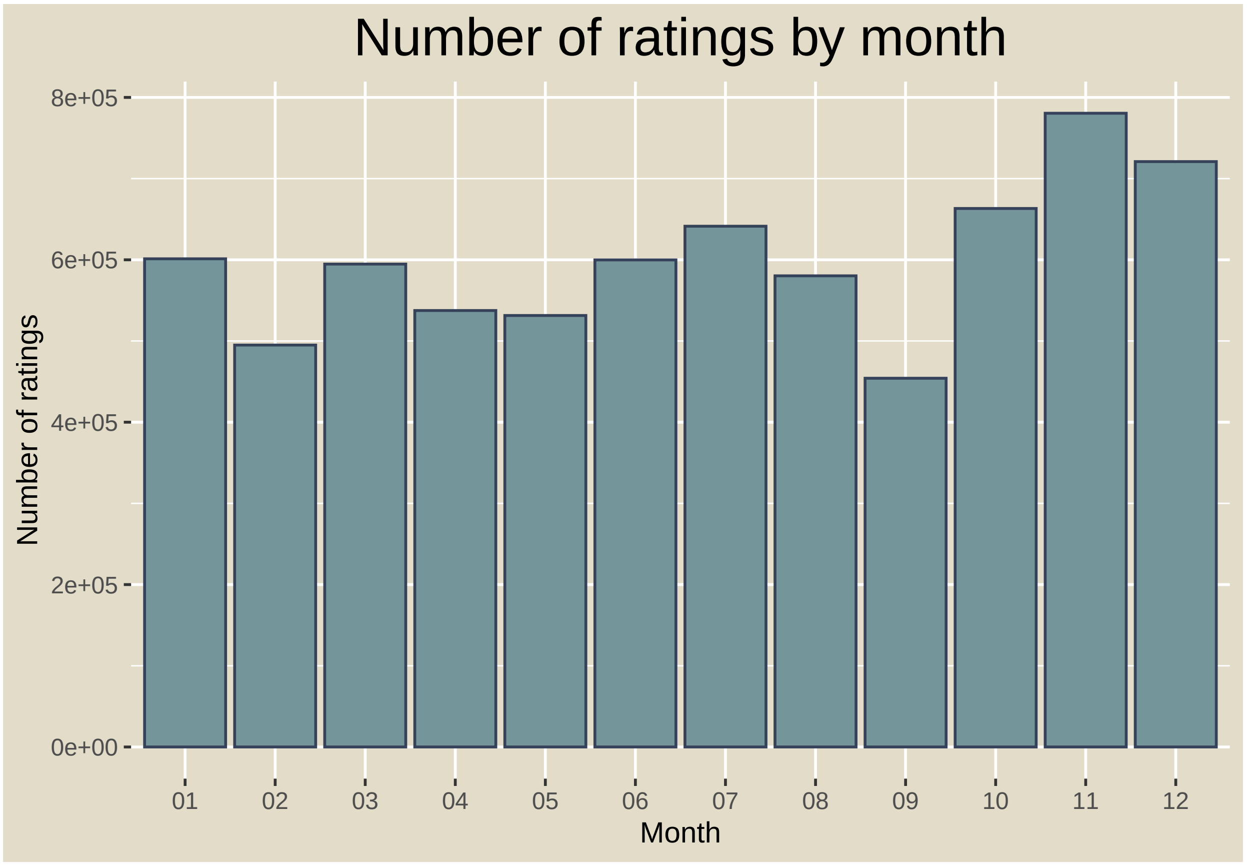
Another thing I can do with the available data is to explore a bit more the timestamp column. First it should be converted into a format where it is clearly understandable what year, month and date a movie received its rating.

```
train_set$timestamp <- as.POSIXct(train_set$timestamp, origin = '1970-01-01')

train_set$month <- sapply(train_set$timestamp, function(x){format(x, '%m')})


head(train_set)
```

```
##   userId movieId rating           timestamp                         title
## 2      1     185      5 1996-08-02 12:58:45                Net, The (1995)
## 4      1     292      5 1996-08-02 12:57:01                Outbreak (1995)
## 5      1     316      5 1996-08-02 12:56:32                Stargate (1994)
## 6      1     329      5 1996-08-02 12:56:32 Star Trek: Generations (1994)
## 7      1     355      5 1996-08-02 13:14:34       Flintstones, The (1994)
## 8      1     356      5 1996-08-02 13:00:53           Forrest Gump (1994)
##                           genres month
## 2         Action|Crime|Thriller    08
## 4  Action|Drama|Sci-Fi|Thriller    08
## 5       Action|Adventure|Sci-Fi    08
## 6 Action|Adventure|Drama|Sci-Fi    08
## 7       Children|Comedy|Fantasy    08
## 8       Comedy|Drama|Romance|War     08
```

Now I can use this information to see on which month on average users tend to rate movies more or less often.

```
train_set %>% group_by(month) %>% summarise (n = n()) %>%
  ggplot() + geom_col(aes(month, n), fill = '#74959A',color = "#354259") +
  ggtitle("Number of ratings by month") + xlab('Month') + ylab('Number of ratings') +
  theme(plot.title = element_text(size = 20, hjust = 0.5),
        plot.background = element_rect(fill = "#E2DCC8"),
        panel.background = element_rect(fill = "#E2DCC8"))
```



What this graph shows is that November happens to be the month when there's on average the biggest amount of ratings. If I don't need this column later I can also remove it like this:

```
train_set <- subset(train_set, select = - c(month))
```

STEP 3: Building the model

Structure:

- Model 1 (Naive RMSE)
- Model 2 (introducing bias)
- Model 3 (adding the user bias)
- Regularization
- Result and conclusion

To start we can find a Naive RMSE that refers to calculating it against the mean of the data that is known to us, using our train set. To find it I need to first define the mean like this:

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

```
## [1] 3.51235
```

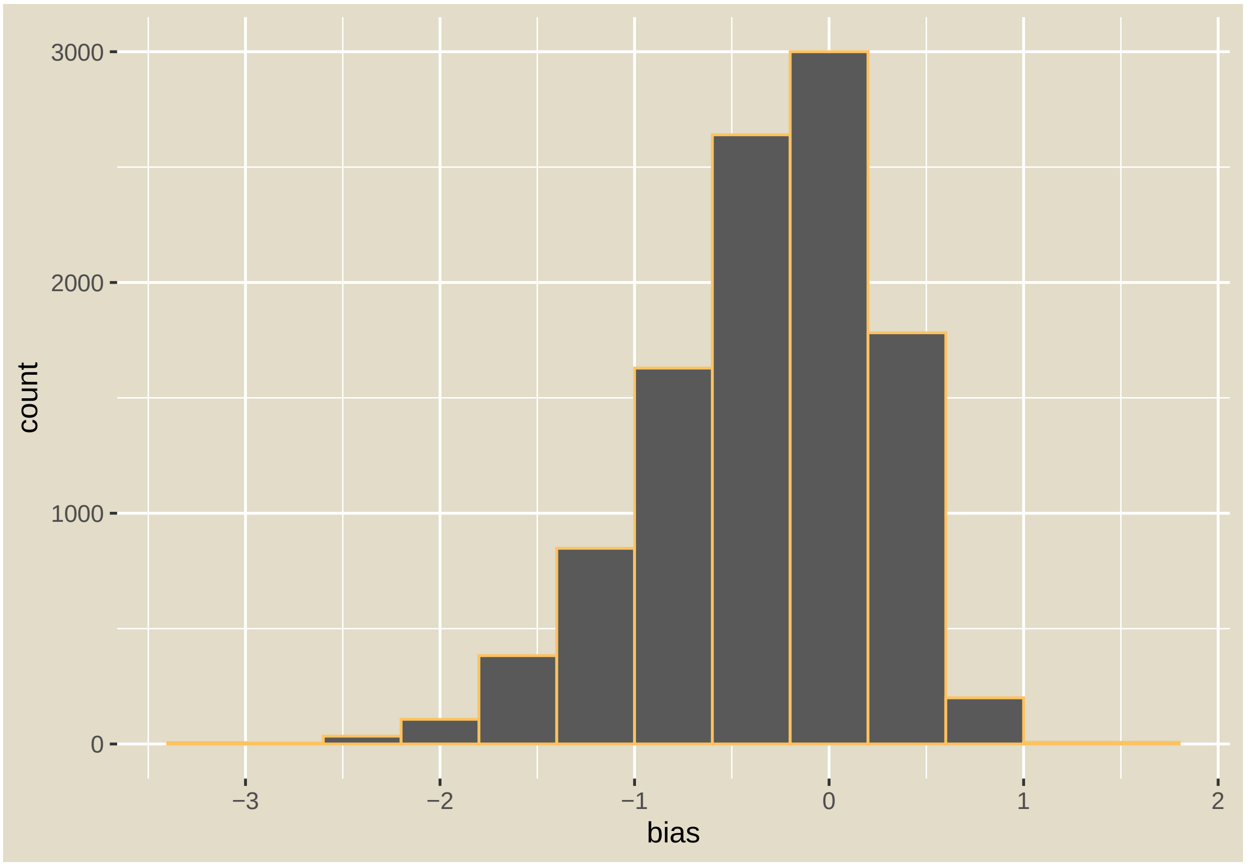
Now I can find the Naive RMSE:

```
naive_rmse <- sqrt(mean((train_set$rating - mu)^2))
naive_rmse
```

```
## [1] 1.060481
```

Knowing this number is useful as a base value to which I will be comparing my model and see how it is improving. To build a better model I will introduce a bias.

```
mov_avg <- train_set%>% group_by(movieId) %>% summarise(bias = mean(rating - mu))

mov_avg %>% ggplot() + geom_histogram(aes(bias), bins = 15, binwidth = .4, color = "#FEC260") +
  theme(plot.background = element_rect(fill = "#E2DCC8"),
        panel.background = element_rect(fill = "#E2DCC8"))
```



```
# Now using the mean I'm building  a predictor
r_hat <- mu + train_set %>%
  left_join(mov_avg, by = 'movieId') %>% pull(bias)

# And I can build the second model, using the bias:

biased_rmse <- sqrt(mean((r_hat - train_set$rating)^2))
```

```
table_rmse <- tibble(naive_rmse, biased_rmse)
table_rmse
```

```
## # A tibble: 1 x 2
##   naive_rmse biased_rmse
##        <dbl>       <dbl>
## 1       1.06       0.942
```

The model has improved a bit, but not significantly. In the third model I will add the user bias.

```
user_b <- edx %>%
  left_join(mov_avg, by = 'movieId') %>%
  group_by(userId) %>% summarise(user_bias = mean(rating - mu - bias))



nrow(user_b)
```

```
## [1] 69878
```

```
pred_ratings <- train_set %>%
  left_join(mov_avg, by = 'movieId') %>%
  left_join(user_b, by = 'userId') %>%
  mutate(prediction = mu + bias + user_bias) %>%
  pull(prediction)




user_b_rmse <- sqrt(mean((pred_ratings - train_set$rating)^2))



table_rmse <- tibble(naive_rmse, biased_rmse, user_b_rmse)
table_rmse
```

```
## # A tibble: 1 x 3
##   naive_rmse biased_rmse user_b_rmse
##        <dbl>       <dbl>       <dbl>
## 1       1.06       0.942          NA
```

The model has been improved again. The new result of RMSE is lower than that of our first and second model. Let's see how much more we can improve further by using regularization.

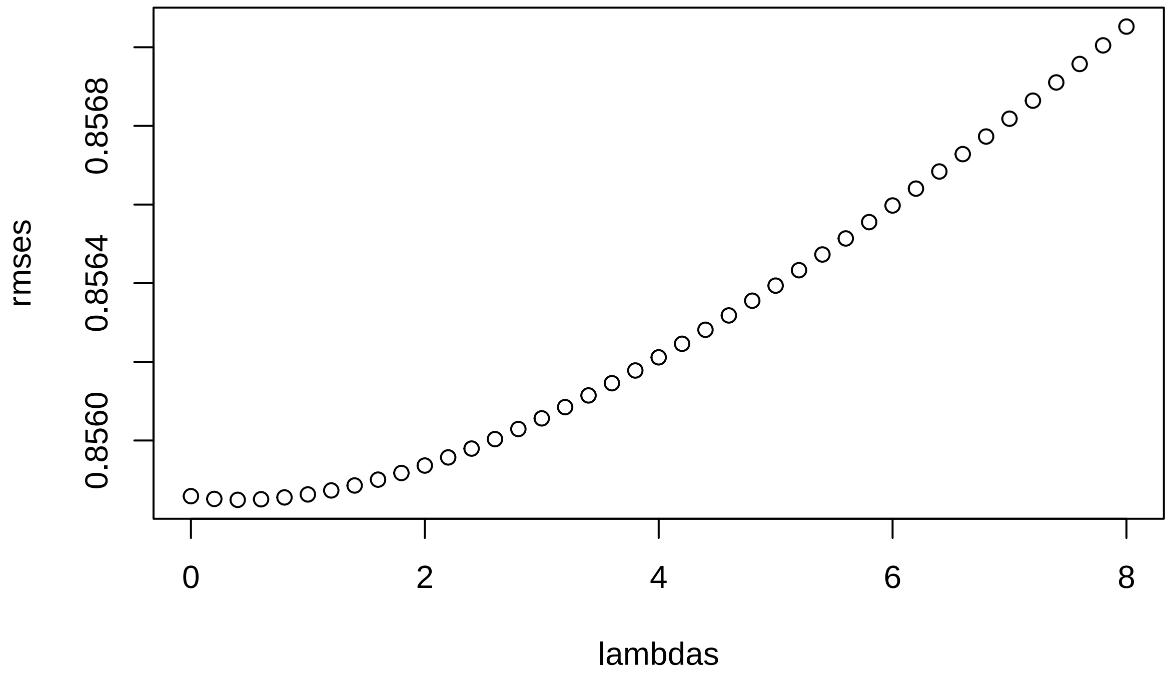
STEP 4 Regularization

Regularization refers to several techniques that are used to perform a calibration on a model to avoid over - or underfitting through a penalty (a bias). Regularization also brings the value of the RMSE down, which is generally useful and also needed for the very purpose of this analysis. The idea is to find the lambda value that will minimize RMSE.

First I'll create a vector of possible lambda values, then it will be fed to the model to see which lambda value brings the RMSE for each of the model I created down.

```
lambdas <- seq(0, 8, 0.2)



# creating a function that will add each value of lambda from the vector
# to an existing model.
```

```r
rmses <- sapply(lambdas, function(l) {

  mu <- mean(train_set$rating)

  bias <- train_set %>%
    group_by(movieId) %>% summarise(b = sum(rating - mu) / (n() + l))

  user_bias <- train_set %>%
    left_join(bias, by = "movieId") %>%
    group_by(userId) %>% summarise(ub = sum(rating - b - mu) / (n() + l))

  pred_ratings <- train_set %>%
    left_join(bias, by = "movieId") %>%
    left_join(user_bias, by = "userId") %>%
    mutate(pred = mu + b + ub) %>%
    pull(pred)

  return(sqrt(mean((pred_ratings - train_set$rating)^2)))
})


# plotting the result
plot(lambdas,rmses)
```



```r
# I can find the value of the optimal lambda like this:

lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 0.4
```

Now that I have my optimal lambda value for the penalty, I can do the next step and add it to the model to later calculate the RMSE.

```r
# applying lamba value to the train_set and defining RMSE

bias <- train_set %>%
  group_by(movieId) %>%
  summarise(b = sum(rating - mu)/(n()+lambda), n = n())



user_bias <- train_set %>%
  left_join(bias, by="movieId") %>%
  group_by(userId) %>%
  summarise(ub = sum(rating - b - mu)/(n()+lambda), n = n())



# applying the result to the test set
train_pred_ratings <- train_set %>%
  left_join(bias, by = "movieId") %>%
  left_join(user_bias, by = "userId") %>%
  mutate(prediction = mu + b + ub) %>%
  pull(prediction)

final_train_rmse <- sqrt(mean((train_pred_ratings - train_set$rating)^2))
final_train_rmse
```

```
## [1] 0.8558491
```

```r
# applying my lambda value to the test_set
bias <- test_set %>%
  group_by(movieId) %>%
  summarise(b = sum(rating - mu)/(n()+lambda), n = n())



user_bias <- test_set %>%
  left_join(bias, by="movieId") %>%
  group_by(userId) %>%
  summarise(ub = sum(rating - b - mu)/(n()+lambda), n = n())



# applying the result to the test set
test_pred_ratings <- test_set %>%
  left_join(bias, by = "movieId") %>%
  left_join(user_bias, by = "userId") %>%
  mutate(prediction = mu + b + ub) %>%
  pull(prediction)
```

I can now calculate the RMSE of the test set:

```r
# calculating the RMSE


reg_rmse <- sqrt(mean((test_pred_ratings - test_set$rating)^2))
reg_rmse
```

```
## [1] 0.8399211
```

Testing the algorithm on the final_holdout_test

```
bias <- final_holdout_test %>%
  group_by(movieId) %>%
  summarise(b = sum(rating - mu)/(n()+lambda), n = n())


user_bias <- final_holdout_test %>%
  left_join(bias, by="movieId") %>%
  group_by(userId) %>%
  summarise(ub = sum(rating - b - mu)/(n()+lambda), n = n())



final_ratings <- final_holdout_test %>%
  left_join(bias, by = "movieId") %>%
  left_join(user_bias, by = "userId") %>%
  mutate(prediction = mu + b + ub) %>%
  pull(prediction)

RMSE <- sqrt(mean((final_ratings - final_holdout_test$rating)^2))
RMSE
```

```
## [1] 0.825615
```

Conclusion

In this analysis I explored the movielens dataset and built a regularization regression model that included the optimal lambda value. The final task of the analysis, which is finding the RMSE value of the final_holdout_test, was fulfilled, with RMSE = 0.825615. During my preparation stage for this analysis, I was able to read through interesting references to the topic of recommendation systems. Some helped me build the current model, others showed that there are even more efficient ways of doing it with even better results, such as NI and KNI (Qu et al. 2019, p. 9).

References

Irizarry, R A 2019, 'Introduction to Data Science', CRC Press, Boca Raton.

Qu, Y, Bai, T, Zhang, W, Nie, J & Tang, J 2019, *An End-to-End Neighborhood-based Interaction Model for Knowledge-enhanced Recommendation*, viewed 21 January 2023, <https://arxiv.org/pdf/1908.04032v2.pdf>.