HarvardX PH125.9xData Science 2022: Capstone Project MovieLens

Jyotirmay Kirtania

Date: 27 September 2022

https://github.com/jk-data/HarvardX-PH125.9xData-Science-2022-Capstone-Project-MovieLens (https://github.com/jk-data/HarvardX-PH125.9xData-Science-2022-Capstone-Project-MovieLens)

ABSTRACT

#This article presents a Machine Learning Project for the partial fulfillment of HarvardX PH125.9xData Science: Capstone Project MovieLens. The objective of this project is to build a predictive model of a movie rating system. The accuracy of the model will be evaluated by measuring the Root Mean Squared Error (RMSE). The MovieLens dataset has about 10 million records and is available online for free (from https://movielens.org (https://movielens.org)). The downloaded dataset was divided in two datasets - the edx dataset (used for training the algorithm), and the validation dataset (used for testing and RMSE evaluation). This Capstone Project was mainly based on the edX lectures on Data Science: Machine Learning.

INTRODUCTION

#Recommendation systems emerged as important factors for e-commerce. Machine learning technology has a wide range of applications in most industries today. The Root Mean Square Error is used to evaluate the closeness of the predictions to the true values in the validation set. Recommendation systems use ratings that users have given to items to make specific recommendations. Companies that sell products to many customers, permit these customers to rate their products. These companies are able to collect massive data-sets that can be used to predict what rating a particular user will give to a specific item. Items for which a high rating is predicted for a given user are then recommended to that user. The same could be done for other items, as movies for instance in our case. Recommendation systems are one of the most used models in machine learning algorithms. In fact the success of Netflix is said to be based on its strong recommendation system. The 2006 Netflix prize was an open competition for the best collaborative filtering algorithm to predict user ratings for films, based on previous customer ratings without any other information about the users or films. This signifies the importance of algorithms for product recommendation systems. For this project we will create a movie recommendation system using the 10M version of MovieLens data-set, collected by GroupLens Research. This 10M MovieLens data-set has about 10 million ratings and 100,000 tags applied to 10,000 movies by 72,000 users.

AIM OF THE PROJECT

#The aim of this project is to train a machine learning algorithm that predicts user ratings (from 0.5 to 5 stars) using the inputs of a provided subset to predict movie ratings in a provided validation set. The value used to evaluate algorithm performance is the Root Mean Square Error, or RMSE. RMSE is one of the measures of the differences between values predicted by a model and the values observed. RMSE is a measure of accuracy, to compare prediction errors of different models for a particular data-set. A lower RMSE is better than a higher one and implies a greater predictive accuracy of the prediction model. The effect of each error on RMSE is proportional to the size of the squared error. Thus larger errors have a disproportionately larger effect on RMSE. Consequently, RMSE is sensitive to outliers. The endpoint of this project is to develop a prediction model which has a much lower RMSE than the naive RMSE. The root mean square error (RMSE) allows us to measure how far predicted values are from observed values in a regression analysis. In other words, how concentrated the data around the line of best fit. The function that computes the RMSE for vectors of ratings and their corresponding predictors will be the following:

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

RMSE <- function(predicted_ratings, true_ratings){sqrt(mean((predicted_ratings - true_ratings)^2))} #RMSE = square root of [sum of squares of (Pi – Oi) / n], where: Pi is the predicted value for the ’i’th observation in the data-set; Oi is the observed value for the ’i’th observation in the data-set; n is the sample size

#What does RMSE indicate? #1) It indicates the absolute fit of the model to the data. #2). It provides average model prediction error in units of the variable of interest. #3). It is a negatively-oriented scores, which means lower values are better.

#Finally, the best fitting model (the model with the lowest RMSE) will be found out to predict the movie ratings.

METHODS

#First, we will begin by preparing the data and loading it from the GroupLens Website. The MovieLens dataset will initially be split into datasets titled “edx” and “validation” before being further partitioned into training and test sets. A preliminary exploratory analysis will be conducted to examine each of the features of the dataset to determine potential “biases” that may confound and reduce the predictive accuracy of our models. Cleaning of the data-set will remove missing values and ensure that the data is in tidy format. Data visualization in the form of histograms will done to facilitate presentation. The Modeling Approach will be based on the insights obtained from the exploratory analysis. This will be refined to create a Final Validation Model with an RMSE less than the naive RMSE.

```
################################################

### 1. Installing essential packages


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-projec
t.org")

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

if(!require(kableExtra)) install.packages("kableExtra", repos = "http://cran.us.r-projec
t.org")

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

if(!require(stringr)) install.packages("stringr", repos = "http://cran.us.r-project.or
g")

library(tidyverse)

library(caret)

library(data.table)

library(knitr)

library(kableExtra)

library(lubridate)

library(stringr)
```

```
  ### 2. Dataset download and preparation

#The MovieLens 10M data-set is described at: https://grouplens.org/datasets/movielens/10
m/

#The data-set is downloaded from: http://files.grouplens.org/datasets/movielens/ml-10m.z
ip

#The following code was used to download the data-set.



dl <- tempfile()
options(timeout=360)
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- read.table(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.da
t"))), 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(levels(movieId))[movieI
d], title = as.character(title), genres = as.character(genres))

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

```
#In order to predict in the most accurate way the movie rating of the users who are yet
to see the movie, the MovieLens data-set will be split into two subsets. One will be the
"edx", a training subset to train the algorithm. The other will be the "validation", a s
ubset to test the movie ratings. The Validation set will be 10% of MovieLens dataset. To
successfully perform validation, we need to make sure all userIds and movieIds in the va
lidation set are also in the edx set.



set.seed(1)
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALS
E)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

```
#Make sure userId and movieId in validation set are also in edx subset:

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

```
#Add rows removed from validation set back into edx set

removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

3. Data exploration

```
#### Create edx set, validation set, and submission file to get overall profile of the d
ata-set

#A general overview of the data-set is obtained:


head(edx) %>%
  print.data.frame()
```

```
##   userId movieId rating timestamp title genres
## 1      1     122      5 838985046  <NA>   <NA>
## 2      1     185      5 838983525  <NA>   <NA>
## 3      1     231      5 838983392  <NA>   <NA>
## 4      1     292      5 838983421  <NA>   <NA>
## 5      1     316      5 838983392  <NA>   <NA>
## 6      1     329      5 838983392  <NA>   <NA>
```

```
#A summary of the data-set confirms that there are no missing values. Total unique movie
s and users


summary(edx)
```

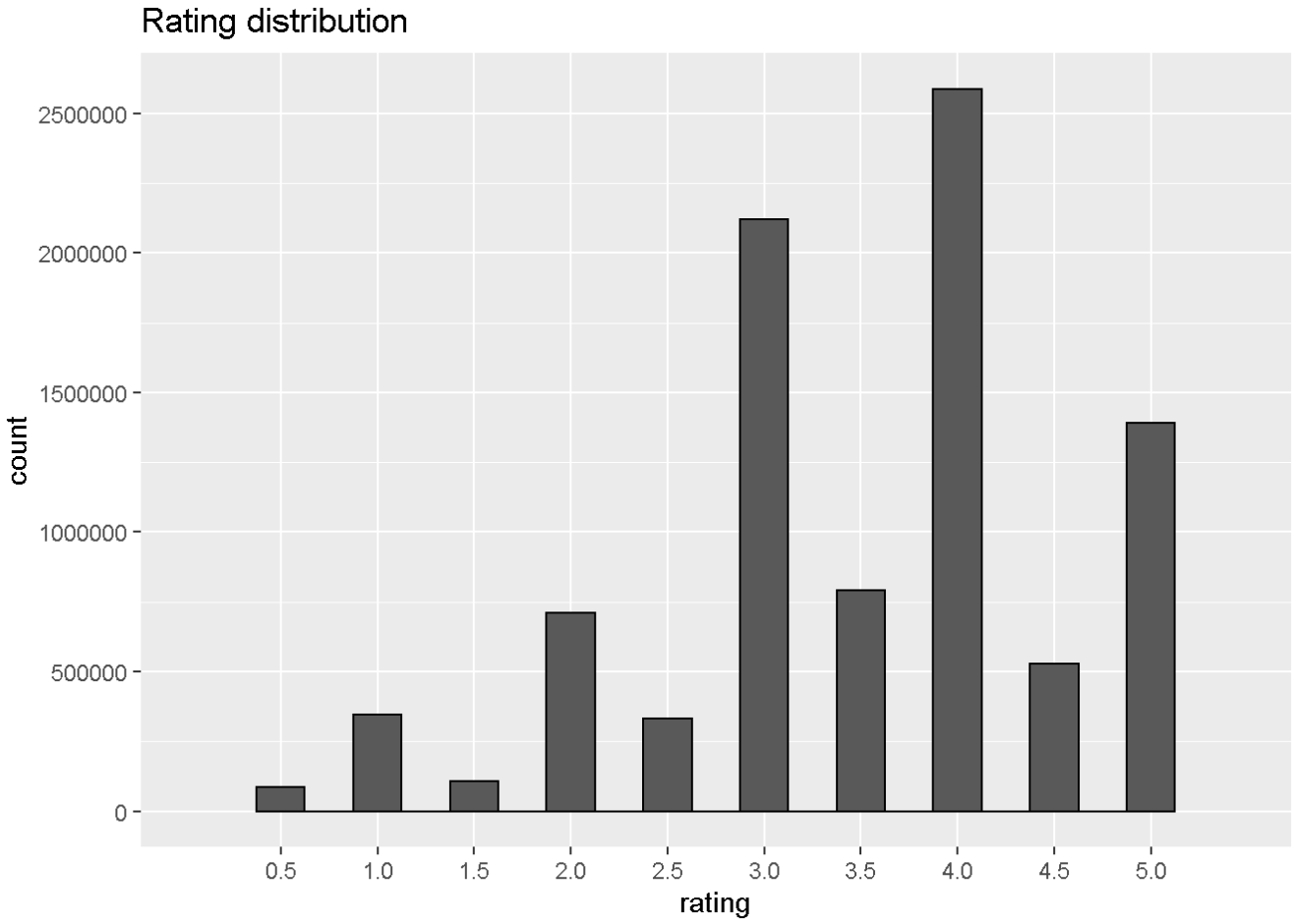
```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18122   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35743   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35869   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53602   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:9000061     Length:9000061
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
#### Number of unique movies and users in the edx dataset

edx %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

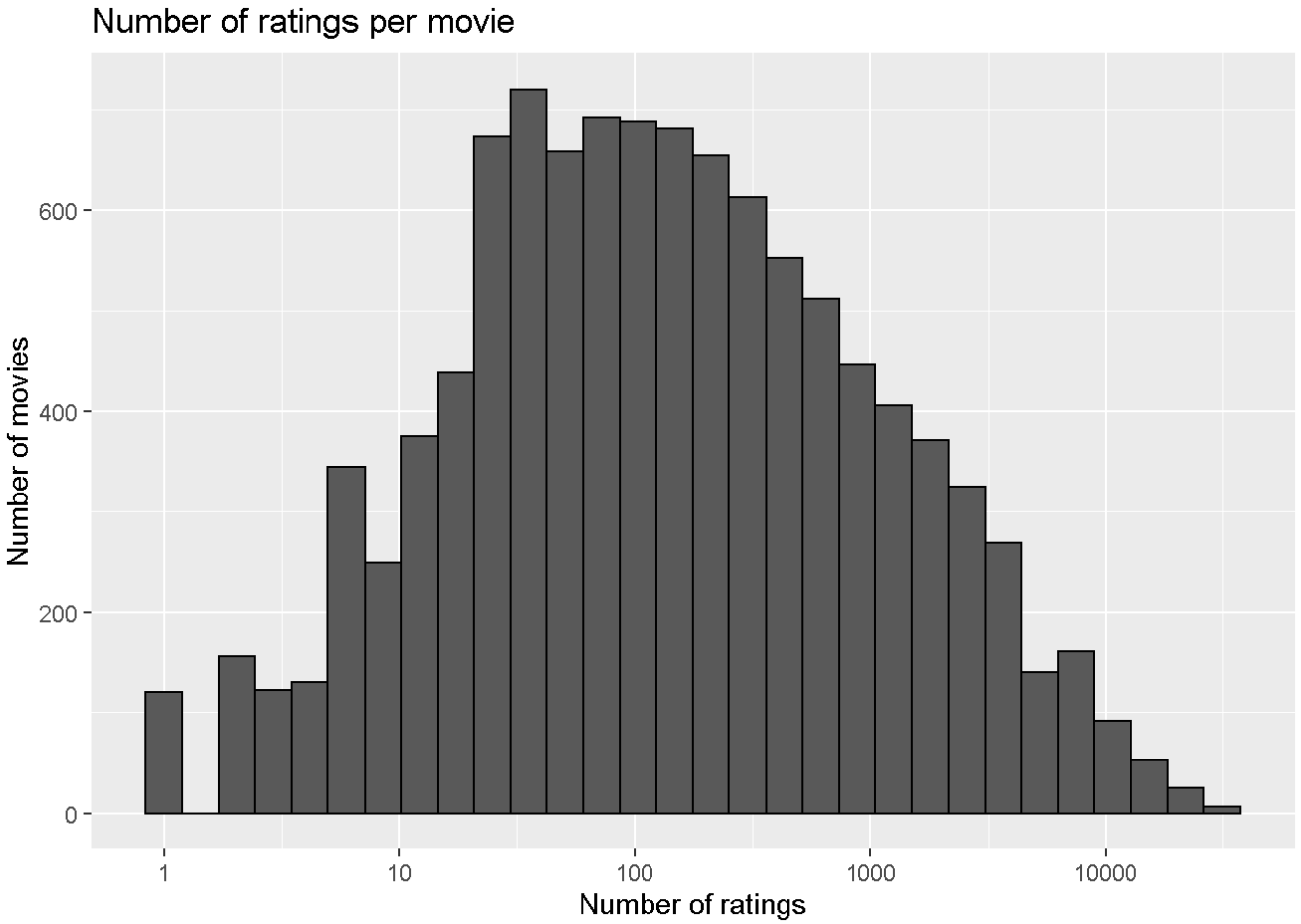
#The edx subset contains about 70000 unique users and about 10700 different movies. If every user rated on every movie, we would have 70000 x 10700 = 749000000 ratings. However, we only have 9000061 ratings, which is only 1.2 percent of the number of all possible ratings.

```
#### Ratings distribution

edx %>%
  ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.25, color = "black") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  scale_y_continuous(breaks = c(seq(0, 3000000, 500000))) +
  ggtitle("Rating distribution")
```



#We observed that users tend to give higher ratings to movies than they give lower ratings. That is number of movies with higher rating are larger than the number of movies with lower ratings. It is also observed that whole number ratings (e.g. 5, 4, 3, 2, 1) by users are more common than fractional ratings (e.g. 4.5, 3.5, 2.5, 1.5, 0.5).

```
#### We plotted the number of ratings per movie

edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  xlab("Number of ratings") +
  ylab("Number of movies") +
  ggtitle("Number of ratings per movie")
```



#We observed that some movies have been rated much frequently than other, while some have very few ratings and sometimes rated only once. This is important for our model, because infrequently rated movies might result in unreliable estimate for our predictions. In fact 125 movies have been rated only once. Thus regularization and a penalty term will be applied to the models in this project. Regularization techniques are that are used to reduce the error by fitting a function appropriately on the given training set and avoid overfitting. Overfitting is an estimate that corresponds very closely or exactly to a particular set of data, and may therefore fail to fit similar data or fail to predict future observations reliably. Regularization is a technique used for tuning the function by adding an additional penalty term in the error function. The additional term controls the excessively fluctuating function such that the coefficients don’t take extreme values.

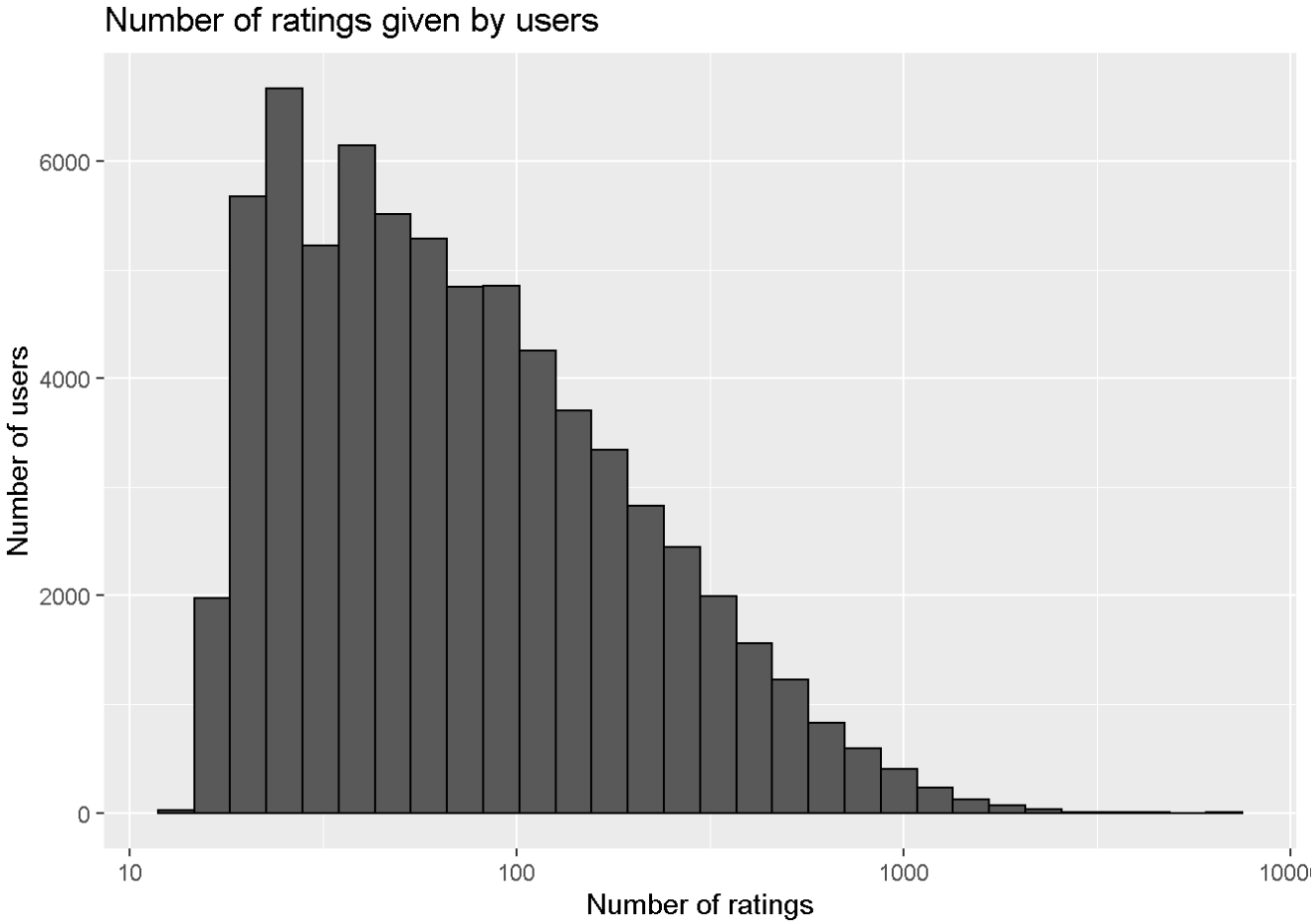
```
#### Table of 20 movies that were rated only once

edx %>%
  group_by(movieId) %>%
  summarize(count = n()) %>%
  filter(count == 1) %>%
  left_join(edx, by = "movieId") %>%
  group_by(title) %>%
  summarize(rating = rating, n_rating = count) %>%
  slice(1:20) %>%
  knitr::kable()
```

titleratingn_rating

NA	5.0	1
NA	3.0	1
NA	3.0	1
NA	1.0	1
NA	3.0	1
NA	1.0	1
NA	1.0	1
NA	2.0	1
NA	1.5	1
NA	2.0	1
NA	1.5	1
NA	1.0	1
NA	3.0	1
NA	3.0	1
NA	3.0	1
NA	2.5	1
NA	2.5	1
NA	3.0	1
NA	4.5	1
NA	2.5	1

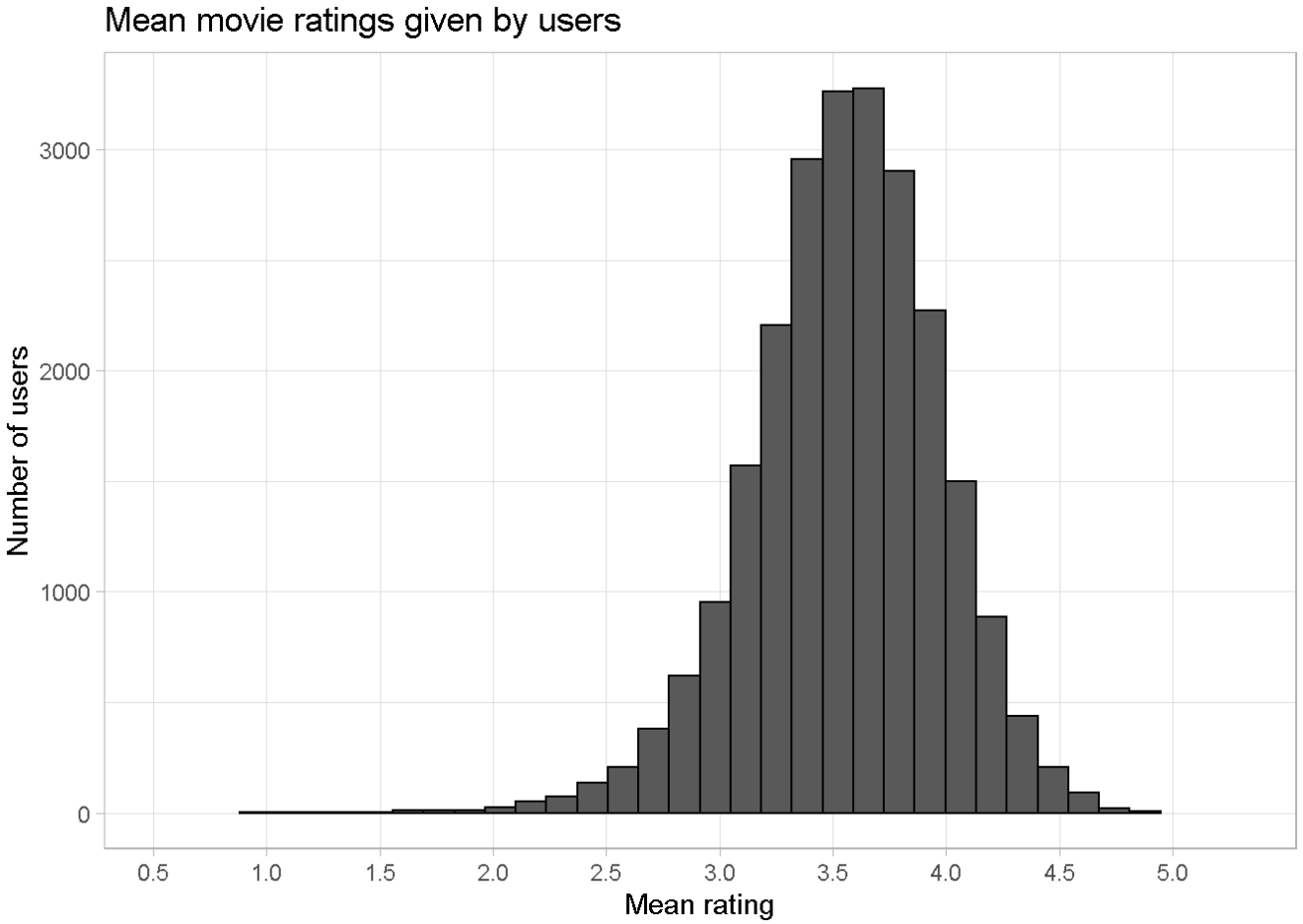
#We observed that 20 movies were rated only once. These movies appear to be obscure. Thus predictions of future ratings for these obscure movies will be difficult.

```
#### We plotted the number of ratings given by users

edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  xlab("Number of ratings") +
  ylab("Number of users") +
  ggtitle("Number of ratings given by users")
```



#We observed that the majority of users have rated between 30 and 100 movies. So, a user penalty term need to be included later in our models.

```
#### We plotted the mean movie ratings given by users

edx %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black") +
  xlab("Mean rating") +
  ylab("Number of users") +
  ggtitle("Mean movie ratings given by users") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  theme_light()
```



#We also observed that users differ greatly how they rate a movie. Some users tend to give much lower star ratings and some users tend to give higher star ratings than average. The visualization above includes only users who have rated at least 100 movies.

4. Data Modelling

#The loss-function, previously anticipated, that compute the RMSE, is defined as follows:

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

#With N being the number of user/movie combinations and the sum occurring over all these combinations. The RMSE is the measure of model accuracy. The RMSE is the typical error we make when predicting a movie rating. If its result is larger than 1, it means that our typical error is larger than one star, which is not a good prediction. The written function to compute the RMSE for vectors of ratings and their corresponding predictions is:

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

A. Average movie rating model

#The first basic model (naive model) predicts the same rating for all movies. So we compute the dataset's mean rating. The expected rating of the data set is between 3 and 4. We start by building the simplest possible recommendation system by predicting the same rating for all movies regardless of user who give it. A model based approach assumes the same rating for all movies with all differences explained by random variation:

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

#With $\epsilon_{u,i}$ being independent error sample from the same distribution centered at 0 and μ the “true” rating for all movies. This very simple model makes the assumption that all differences in movie ratings are due to random variations alone. We know that the estimate that minimize the RMSE is the least square estimate of $Y_{u,i}$, in this case, is the average of all ratings: The expected rating of the dataset is between 3 and 4.

```
##### We computed the mean rating of the dataset

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

```
## [1] 3.512464
```

Following is the test results (naive RMSE) based on simple prediction

```
#If we predict all unknown ratings with $\mu$ or mu, we obtain the first naive RMSE:

naive_rmse <- RMSE(validation$rating, mu)
```

```
##### We checked the results

naive_rmse
```

```
## [1] 1.060651
```

```
##### The prediction in data frame is saved.

rmse_results <- data_frame(method = "Average movie rating model", RMSE = naive_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Average movie rating model	1.060651

#This gives us our baseline (naive) RMSE to compare with #next modelling approaches.

#In order to do better than simply predicting the average rating, we have incorporated some of the insights we gained during the exploratory data analysis

B. Movie effect model

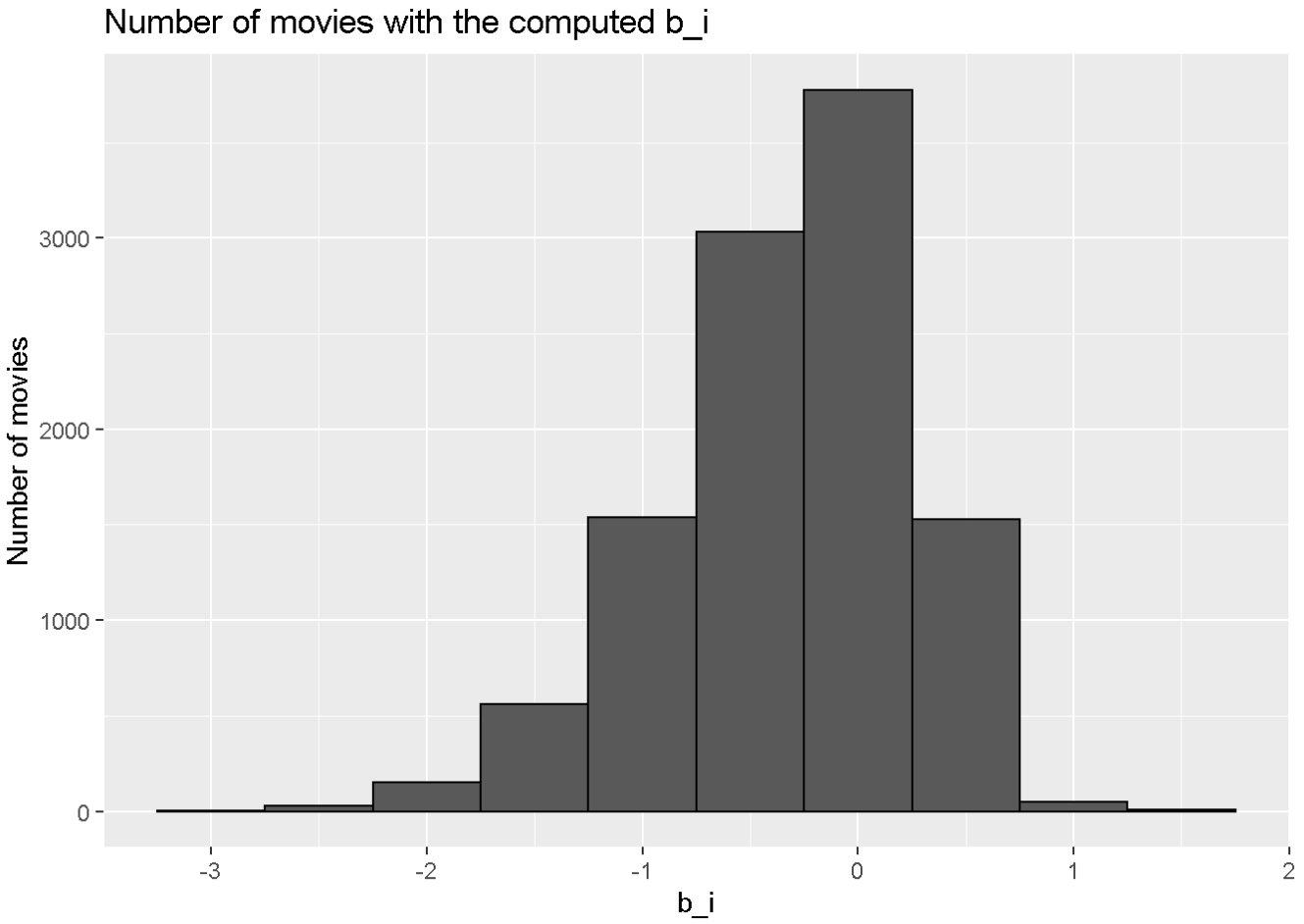
#To improve above model we focus on the fact that, some movies generally receive a higher rating than others. Higher ratings are mostly linked to popular movies among users and the opposite is true for unpopular movies. We compute the estimated deviation of mean rating for each movie from the total mean of all movies μ. The resulting variable is called “b” (bias) for each movie “i” b_i, #that represents average ranking for movie i:

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

Simple model taking into account the movie effect b_i

Subtract the rating minus the mean for each rating the movie received

```
##### Plot number of movies with the computed b_i

movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"),
                     ylab = "Number of movies", main = "Number of movies with the comput
ed b_i")
```



#The histogram is left skewed. This implies that more movies have negative effects. This is called the penalty term movie effect. Our prediction improves once we use this model.

```
##### We tested and saved the RMSE results

predicted_ratings <- mu +  validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
model_1_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie effect model",
                                     RMSE = model_1_rmse ))
```

```
# Checked the results

rmse_results %>% knitr::kable()
```

method	RMSE

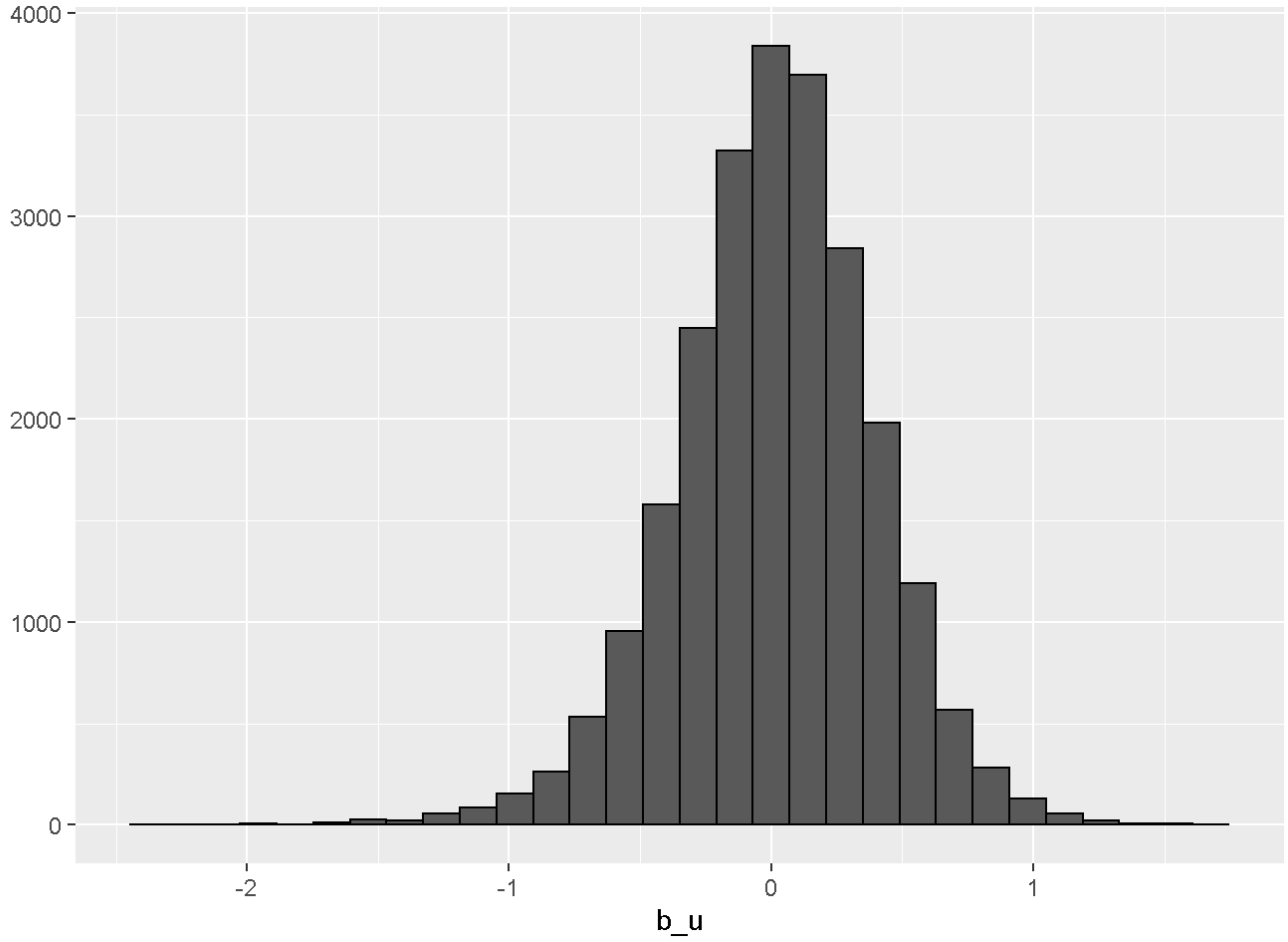
method	RMSE
Average movie rating model	1.0606506
Movie effect model	0.9437046

#Thus we have predicted movie rating considering the fact that movies are rated differently, if we add the computed b_i to μ. If an individual movie is on average rated worse that the average rating of all movies μ, we predict that it will rated lower that μ by b_i, the difference of the individual movie average from the total average. We can see an improvement. But this model does not consider the individual user rating effect.

C. Movie and user effect model

#We compute the average rating for user μ, for those that have rated over 100 movies, said penalty term user effect. In fact users affect the movie ratings positively or negatively.

```
##### We plotted the penalty term user effect

user_avgs<- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating - mu - b_i))
user_avgs%>% qplot(b_u, geom ="histogram", bins = 30, data = ., color = I("black"))
```



#There is significant variability across users as well: some users are very choosy while others like almost every movie. This implies that further improvement to our model my be:

#

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

where b_u is a user-specific effect. If a choosy user (negative b_u rates a great movie (positive b_i), the effects nullify each other and we may be able to correctly predict that this user gave this great movie a 3 rather than a 5. We compute an approximation by computing μ and b_i, and estimating b_u, as the average of

$$Y_{u,i} - \mu - b_i$$

```
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

```
#Now We can construct predictors and see if RMSE improves:

predicted_ratings <- validation%>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
```

```
##### We tested and saved the RMSE results

model_2_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie and user effect model",
                                     RMSE = model_2_rmse))

##### We checked the result

rmse_results %>% knitr::kable()
```

method	RMSE
Average movie rating model	1.0606506
Movie effect model	0.9437046
Movie and user effect model	0.8655329

#Our rating predictions further reduced the RMSE. But we still can make mistakes with our first model (using only movies). The supposedly “best” and “worst” movie were rated by few users, in most cases just one user. These movies were mostly obscure ones. This is because with a few users, we have more uncertainty. Therefore larger estimates of b_i, negative or positive, are more likely. Large errors can increase the RMSE of this model. Until now, we computed standard error and constructed confidence intervals to account for different levels of uncertainty. However, when making predictions, we need one prediction, instead of an interval. For this we shall use the concept of regularization, that permits to penalize large estimates that come from small sample sizes. The general idea is to add a penalty for large values of b_i to the sum of squares equation that we minimize. So having many large b_i, make it harder to minimize. Regularization is a method used to reduce the effect of overfitting.

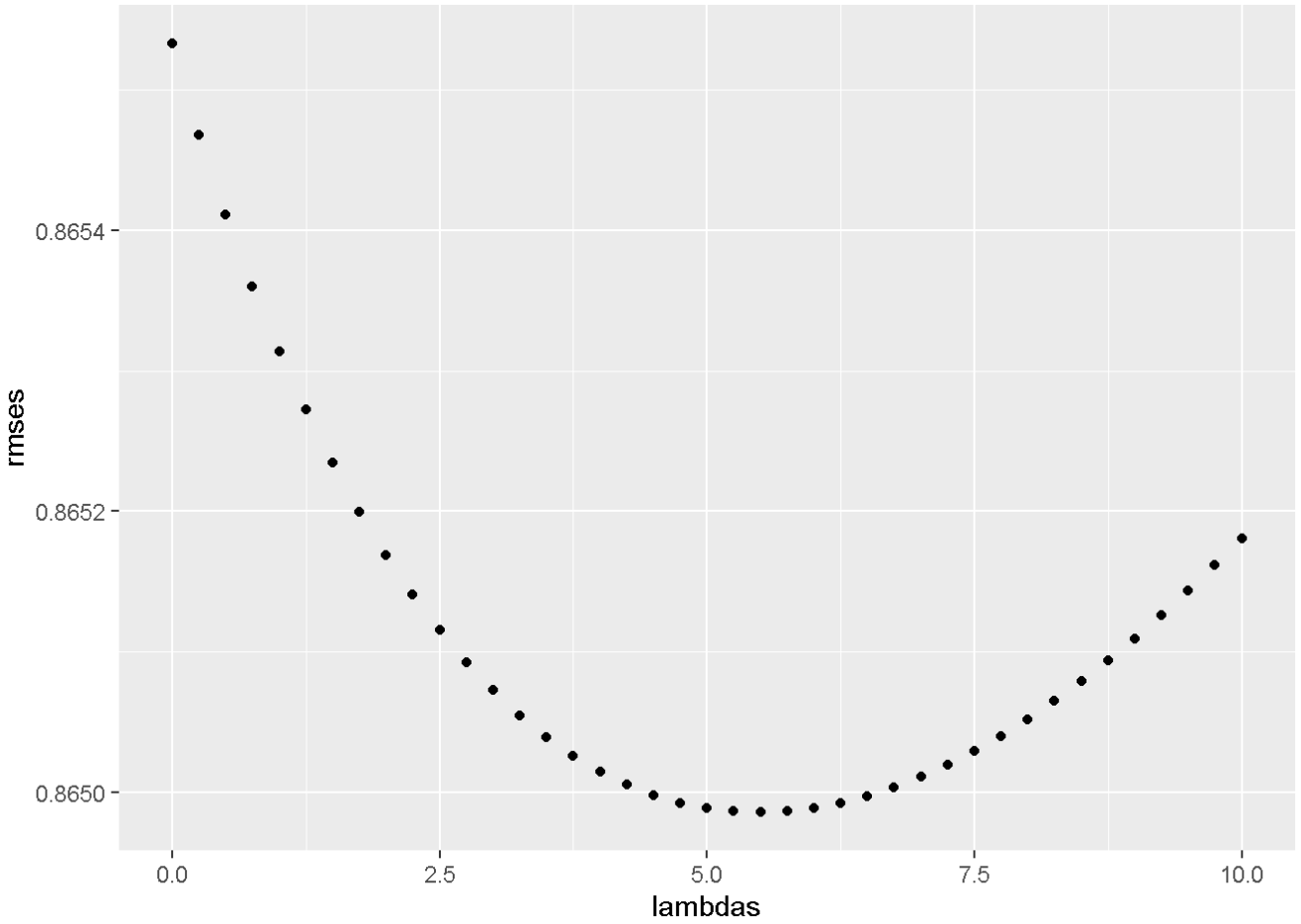
D. Regularized movie and user effect model

##There are also some users who rated a very small number of movies. These ratings can strongly influence the predictions. The use of the regularization permits to penalize these aspects. We should find the value of lambda which is a tuning parameter that will minimize the RMSE.

```
# Use cross-validation to choose the lambda

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

```
##### For each lambda, find b_i & b_u, followed by rating prediction & testing

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

  mu <- mean(edx$rating)

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

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

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

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

```
##### We plotted the RMSEs versus the lambdas to select the optimal lambda

qplot(lambdas, rmses)
```




```
##### The optimal lambda for the the regularized prediction model

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

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

```
##### We tested the RMSEs of the various prediction models and saved the results

rmse_results <- bind_rows(rmse_results,
data_frame(method="Regularized movie and user effect model",
RMSE = min(rmses)))
```

RESULTS

```
#### The summary of the results comparing the various models - from the worst to the bes
t RMSEs are:

rmse_results %>% knitr::kable()
```

method	RMSE
Average movie rating model	1.0606506
Movie effect model	0.9437046
Movie and user effect model	0.8655329
Regularized movie and user effect model	0.8649857

Thus we saw that the regularized movie and user effect model has the lowest RMSE of 0.8649857, among the four prediction models we compared in this project.

DISCUSSION

#We have come to know the regularized movie and user effect model has the lowest RMSE of 0.8649857. So this is the final prediction model for our project. It is expressed by the following:

#

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

#This model will work well if the average user does not rate a particularly popular movie with a large positive rating or rate a disliked movie with a very low rating.

CONCLUSION

#We have successfully built a machine learning algorithm to predict movie ratings with the MovieLens data-set. The regularized model with user effect is characterized by a much lower RMSE value than the average movie rating model (naive model). Thus the final model is the optimum model to use for the present project. We can conclude that improvements in the RMSE can be achieved by adding other effects (genre, year, age, …). Other machine learning models can further improve the RMSE, but PC hardware limitations, as the RAM, CPU clock speed, may pose as constraints.

```
## COMPUTATION ENVIRONMENT

print("Operating System:")
```

```
## [1] "Operating System:"
```

```
version
```

```
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## crt            ucrt
## system         x86_64, mingw32
## status
## major          4
## minor          2.1
## year           2022
## month          06
## day            23
## svn rev        82513
## language       R
## version.string R version 4.2.1 (2022-06-23 ucrt)
## nickname       Funny-Looking Kid
```