Predict Movie Rating

Sathish kumar Subbarayelu

Introduction

Create Train and Validation Sets

Introduction You will use the following code to generate your datasets. Develop your algorithm using the edx set. For a final test of your algorithm, predict movie ratings in the validation set as if they were unknown. RMSE will be used to evaluate how close your predictions are to the true values in the validation set.

Important: The validation data should NOT be used for training your algorithm and should ONLY be used for evaluating the RMSE of your final algorithm. You should split the edx data into separate training and test sets to design and test your algorithm.

The data

We download the MovieLens data and run the following code the staff provided to generate the datasets. We use the 10M version of the MovieLens dataset available [here] (https://grouplens.org/datasets/movielens/10m/ (https://grouplens.org/datasets/movielens/10m/)).

Create edx set, validation set

Note: this process could take a couple of minutes

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

```
## Loading required package: tidyverse
```

```
## -- Attaching packages ---------------------------------------------------------------
tidyverse 1.3.0 --
```

```
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------------------------------ tidyv
erse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: caret
```

```
## Warning: package 'caret' was built under R version 3.6.2
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

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

```
## Loading required package: data.table
```

```
## Warning: package 'data.table' was built under R version 3.6.2
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
## The following object is masked from 'package:purrr':
##
##     transpose
```

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

```
## Loading required package: rafalib
```

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

```
## Loading required package: recosystem
```

```
## Warning: package 'recosystem' was built under R version 3.6.2
```

MovieLens 10M dataset: https://grouplens.org/datasets/movielens/10m/ (https://grouplens.org/datasets/movielens/10m/) http://files.grouplens.org/datasets/movielens/ml-10m.zip (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(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

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

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)` instead
```

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

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

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
```

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

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

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

As per course instruction, I randomly split the edx set into training and test data. We use set.seed(1) to provide a reproducible example and the createDataPartition's caret function to select your test index to create two separate data frames called: train and test from the original edx data frame. test contain a randomly selected 10% of the rows of edx,and have train contain the other 90%. We will use these data frames to do the rest of the analyses. In the code, we make sure userId and movieId in test set are also in train set and add rows removed from test set back into train set. After you create train and test, we remove removed,temp,test_index to save space.

```
 set.seed(1, sample.kind="Rounding")  # generate reproducible partition
```

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

```
# if using R 3.5 or earlier, use `set.seed(1)` instead

# train set will be 90% of edx data
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train <- edx[-test_index,]
temp <- edx[test_index,]

# Make sure userId and movieId in test set are also in train set
# test set will be 10% of edx data

test <- temp %>%
  semi_join(train, by = "movieId") %>%
  semi_join(train, by = "userId")

# Add rows removed from test set back into train set

removed <- anti_join(temp, test)
```

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

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

rm(removed, temp, test_index)  # removing used data to save space
```

Data Exploration

```
# let us view the train sets in tidy format with thousands of rows:

train %>% as_tibble()  ## show train set
```

```
## # A tibble: 8,100,065 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     292      5 838983421 Outbreak (1995)       Action|Drama|Sci-Fi|T~
##  3      1     316      5 838983392 Stargate (1994)       Action|Adventure|Sci-~
##  4      1     329      5 838983392 Star Trek: Generation~ Action|Adventure|Dram~
##  5      1     355      5 838984474 Flintstones, The (199~ Children|Comedy|Fanta~
##  6      1     356      5 838983653 Forrest Gump (1994)   Comedy|Drama|Romance|~
##  7      1     362      5 838984885 Jungle Book, The (199~ Adventure|Children|Ro~
##  8      1     364      5 838983707 Lion King, The (1994)  Adventure|Animation|C~
##  9      1     370      5 838984596 Naked Gun 33 1/3: The~ Action|Comedy
## 10      1     377      5 838983834 Speed (1994)          Action|Romance|Thrill~
## # ... with 8,100,055 more rows
```

```
#################################################################################################
# Each row represents a rating given by one user to one movie. We can see the number of unique
# users from train that provided ratings and how many unique movies were rated. How many users
# are there in the train data set? How many movies are in the train data set? What is the
# lowest and highest rating in the # train data set?
#################################################################################################

train %>% summarize(
  n_users=n_distinct(userId),# unique users from train
  n_movies=n_distinct(movieId),# unique movies from train
  min_rating=min(rating),  # the lowest rating
  max_rating=max(rating) # the highest rating
)
```

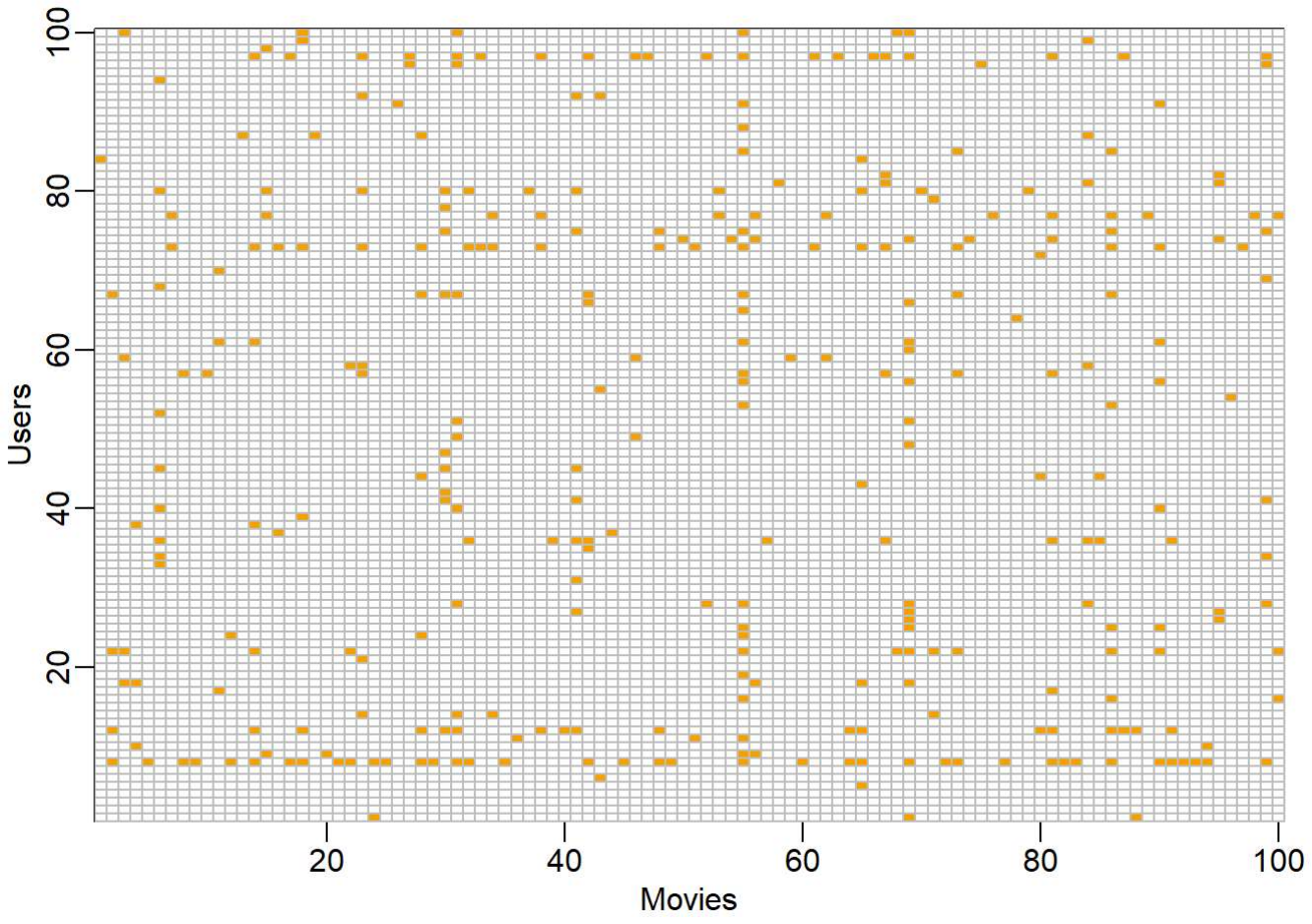
```
##   n_users n_movies min_rating max_rating
## 1   69878    10677        0.5          5
```

```
#################################################################################################
# If we multiply n_users times n_movies, we get a number almost of 750 million, yet our data
# table has about 8,000,000 rows. This implies that not every user rated every movie. So we
# can think of these data as a very large matrix, with users on the rows and movies on the
# columns, with many empty cells.The gather function permits us to convert it to this format,
# but if we try it for the entire matrix,
# it will crash R. Let's show the matrix for seven users and five movies.
#################################################################################################

# matrix for 5 movies and 7 users
keep <- train %>%
  count(movieId) %>%
  top_n(5, n) %>%
  .$movieId

tab <- train %>%
  filter(movieId%in%keep) %>%
  filter(userId %in% c(13:20)) %>%
  select(userId, title, rating) %>%
  mutate(title = str_remove(title, ", The"),
         title = str_remove(title, ":.*")) %>%
  spread(title, rating)
tab %>% knitr::kable()
```

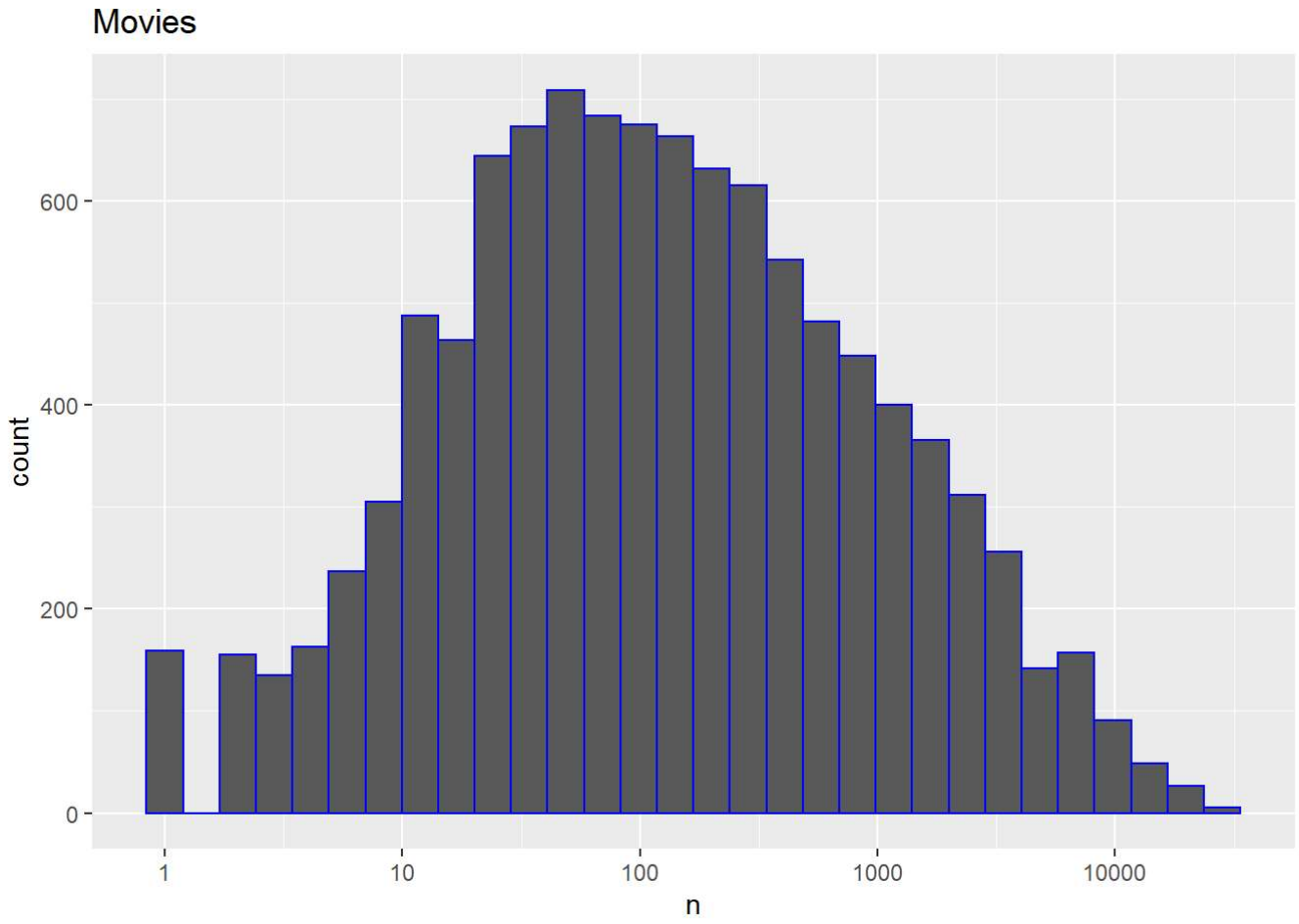
userId	Forrest Gump (1994)	Jurassic Park (1993)	Pulp Fiction (1994)	Shawshank Redemption (1994)	Silence of the Lambs (1991)
13	NA	NA	4	NA	NA
16	NA	3	NA	NA	NA
17	NA	NA	NA	NA	5
18	NA	3	NA	4.5	5

userId	Forrest Gump (1994)	Jurassic Park (1993)	Pulp Fiction (1994)	Shawshank Redemption (1994)	Silence of the Lambs (1991)
19	4	1	NA	4.0	NA

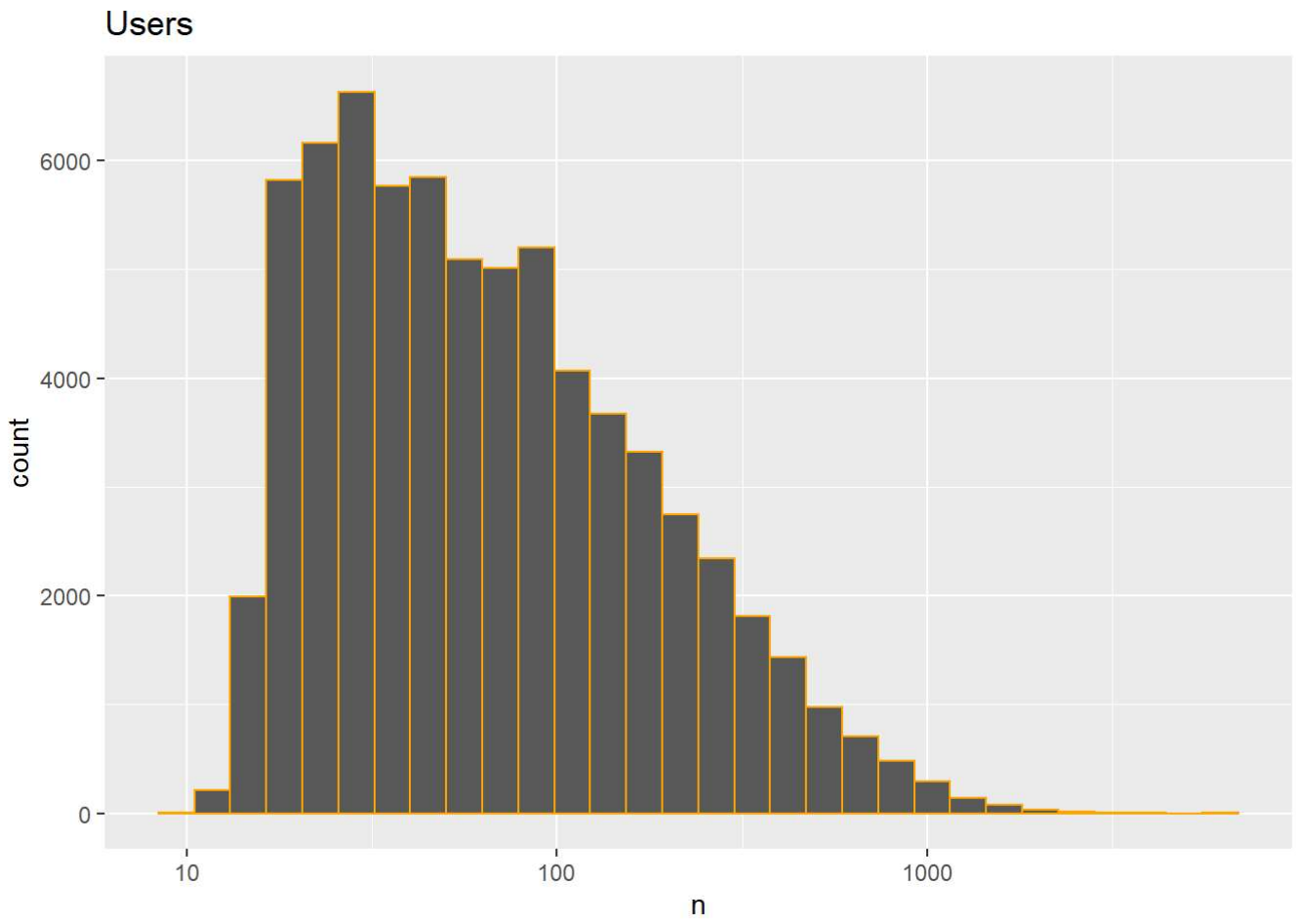
```
#########################################################################################################
# You can think of the task of a recommendation system as filling in the NAs in the table above.
# To see how sparse the matrix is, here is the matrix for a random sample of 100 movies and 100
# users with yellow indicating a user/movie combination for which we have a rating.
# matrix for a random sample of 100 movies and 100 users with yellow
# indicating a user/movie combination for which we have a rating.
#########################################################################################################

users <- sample(unique(train$userId), 100)
mypar()
train %>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>% select(sample(ncol(.), 100)) %>%
  as.matrix() %>% t(.) %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users")
abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
```



```
# The first thing we notice is that some movies get rated more than others. Here is the distribu
tion:

# plot count rating by movie
train %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "blue") +
  scale_x_log10() +
  ggtitle("Movies")
```




```
# Our second observation is that some users are more active than others at rating movies:

# plot count rating by user
train %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "orange") +
  scale_x_log10() +
  ggtitle("Users")
```




```
rm(tab, keep, users) # removing used data to save space
```

```
#############################################################################################
# Methods and Analysis
# More simplest model
# Lets start by building the simplest possible recommendation system:
# we predict the same rating for all movies regardless of user. What number should this
# prediction be?
# We can use a model based approach to answer this. A model that assumes the same rating
# for all movies and users with all the differences explained by random variation would
# look like this:
#   Yu,i=μ+εu,i
#############################################################################################

mu <- mean(train$rating) # compute mean rating
mu
```

```
## [1] 3.512456
```

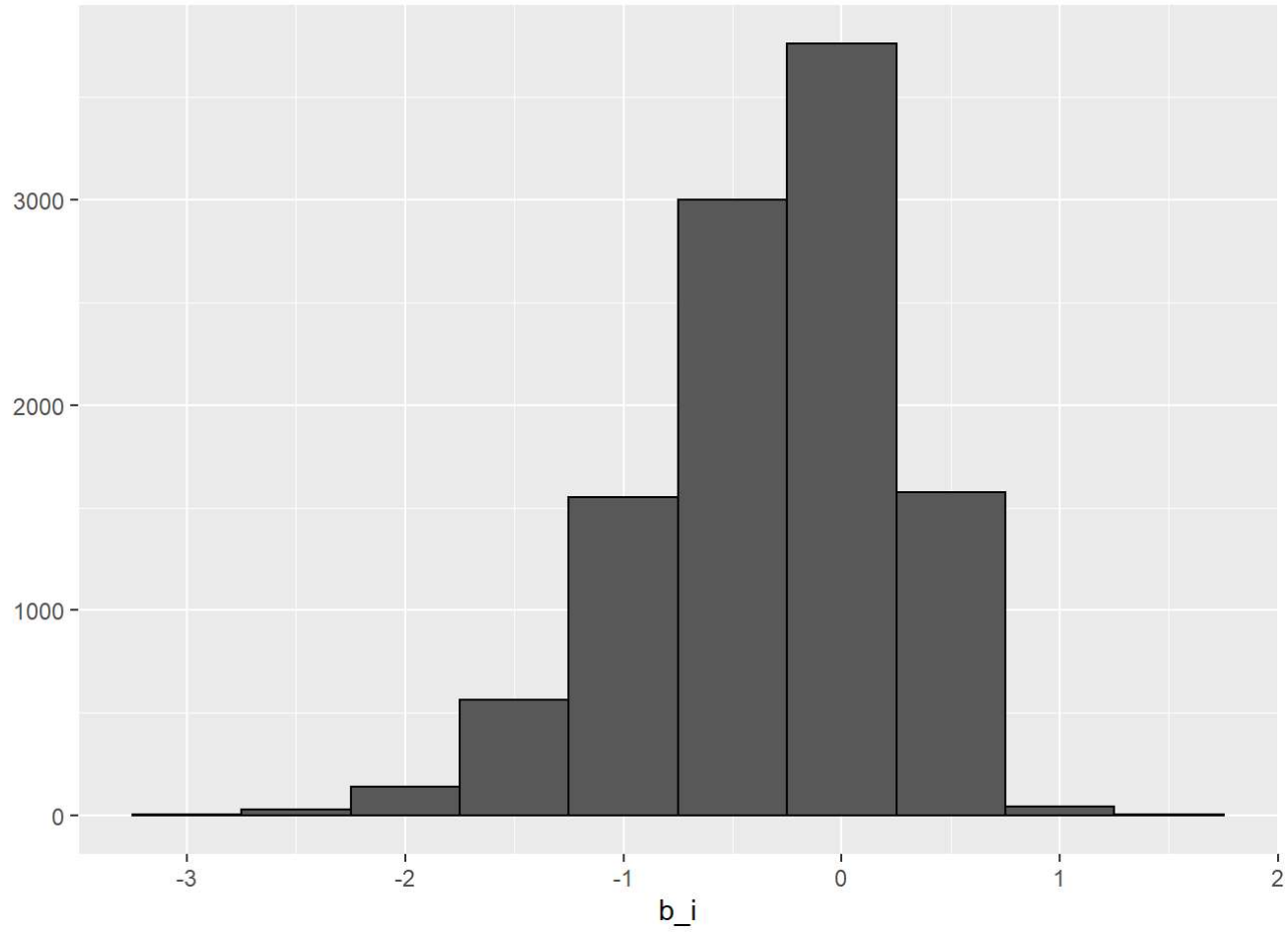
```
naive_rmse <- RMSE(test$rating, mu) # compute root mean standard error
naive_rmse
```

```
## [1] 1.060054
```

```
#############################################################################################
# From looking at the distribution of ratings, we can visualize that this is the standard
# deviation of that distribution. We get a RMSE about 1.06. To get a 25% score in movielens
# project, i have to get an RMSE of about 0.857. So we can definitely do much better!
# As we will be comparing different approaches, we create a results table with this naive
# approach:
#############################################################################################

# create a results table with this approach
rmse_results <- tibble(method = "Just the average", RMSE = naive_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.060054

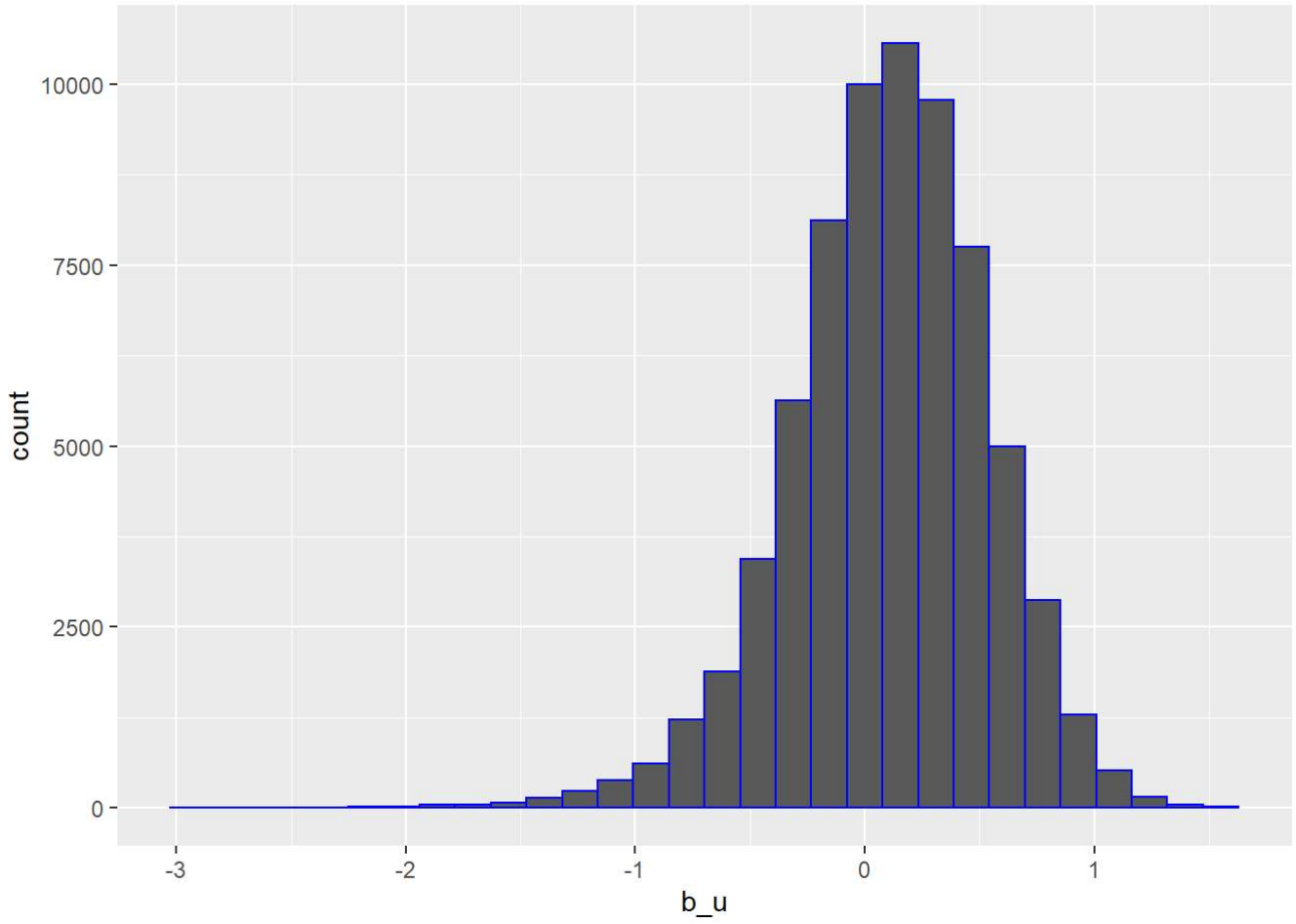
```
#############################################################################################
# Modeling movie effects
# We know from experience that some movies are just generally rated higher than others.
# This intuition, that different movies are rated differently, is confirmed by data.
# We can augment our previous model by adding the term bi to represent average ranking for movie
# i:
#
#   Yu,i=μ+bi+εu,i
#
# Statistics textbooks refer to to the bs as effects. However, in the Netflix challenge papers,
# they refer to them as “bias”, thus the b notation.
#
# We can again use least squares to estimate the bi in the following way,

# fit <- lm(rating ~ as.factor(movieId), data = train) # takes long time

# but because there are thousands of bi as each movie gets one, the lm() function will be very
# slow here # if not imposible to run.
# In this particular situation, we know that the least square estimate b^i is just the average
# of Yu,i−μ^ for each movie i. So we can compute them this way:
#############################################################################################
##

mu <- mean(train$rating)
movie_avgs <- train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"))
```




```
# Let’s see how much our prediction improves once we use y^u,i=μ^+b^i:

# create a results table with this and prior approaches
predicted_ratings <- mu + test %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i

# create a results table with this and prior approach
model_1_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie Effect Model on test set",
                                 RMSE = model_1_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0600537
Movie Effect Model on test set	0.9429615

```
#############################################################
# We already see an improvement. But can we make it better?
#
# Modeling User effects
# Let's compute the average rating for user u:
#############################################################

train %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "blue")
```



```
#############################################################################################
# Notice that there is variability across users as well: some users are very cranky and others
# love every movie. This implies that a further improvement to our model may be:
#
#   Yu,i=μ+bi+bu+εu,i
#
# where bu is a user-specific effect. Now if a cranky user (negative bu) rates a great movie
# (positive bi),the effects counter each other and we may be able to correctly predict that
# this user gave this great movie a 3 rather than a 5.
# we will compute an approximation by computing μ^ and b^i and estimating b^u as the average of
# yu,i−μ^−b^i:
#############################################################################################

# compute user effect b_u
user_avgs <- train %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# We can now construct predictors and see how much the RMSE improves:
```

```
# compute predicted values on test set
predicted_ratings <- test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

# create a results table with this and prior approaches
model_2_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie and User Effects Model on test set",
                                 RMSE = model_2_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0600537
Movie Effect Model on test set	0.9429615
Movie and User Effects Model on test set	0.8646843

```
#############################################################################################
#
# Regularization
# Regularization permits us to penalize large estimates that are formed using small sample size
s.
#
# These are noisy estimates that we should not trust, especially when it comes to prediction.
# Large errors can increase our RMSE, so we would rather be conservative when unsure.
#
# Let's look at the top 10 worst and best movies based on b^i.
# First, let's create a database that connects movieId to movie title:
#############################################################################################
#

# connect movieId to movie title
movie_titles <- train %>%
  select(movieId, title) %>%
  distinct()
```

```
# Here are the 10 best movies according to our estimate:
# top 10 best movies based on b_i
movie_avgs %>% left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	b_i
Hellhounds on My Trail (1999)	1.487544
Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)	1.487544
Shadows of Forgotten Ancestors (1964)	1.487544
Fighting Elegy (Kenka erejii) (1966)	1.487544
Sun Alley (Sonnenallee) (1999)	1.487544
Blue Light, The (Das Blaue Licht) (1932)	1.487544
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.237544
Life of Oharu, The (Saikaku ichidai onna) (1952)	1.237544
Human Condition II, The (Ningen no joken II) (1959)	1.237544
Human Condition III, The (Ningen no joken III) (1961)	1.237544

```
# And here are the 10 worst:
# top 10 worse movies based on b_i
movie_avgs %>% left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(title, b_i) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	b_i
Besotted (2001)	-3.012456
Hi-Line, The (1999)	-3.012456
Accused (Anklaget) (2005)	-3.012456
Confessions of a Superhero (2007)	-3.012456
War of the Worlds 2: The Next Wave (2008)	-3.012456
SuperBabies: Baby Geniuses 2 (2004)	-2.767775
Disaster Movie (2008)	-2.745789
From Justin to Kelly (2003)	-2.638139
Hip Hop Witch, Da (2000)	-2.603365
Criminals (1996)	-2.512456

```
# When often the best are rated:
# add number of rating of the "best" obscure movies
train %>% count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

```
## Joining, by = "movieId"
```

title	b_i	n
Hellhounds on My Trail (1999)	1.487544	1
Satan’s Tango (SÃ¡tÃ¡ntangÃ³) (1994)	1.487544	1
Shadows of Forgotten Ancestors (1964)	1.487544	1
Fighting Elegy (Kenka erejii) (1966)	1.487544	1
Sun Alley (Sonnenallee) (1999)	1.487544	1
Blue Light, The (Das Blaue Licht) (1932)	1.487544	1

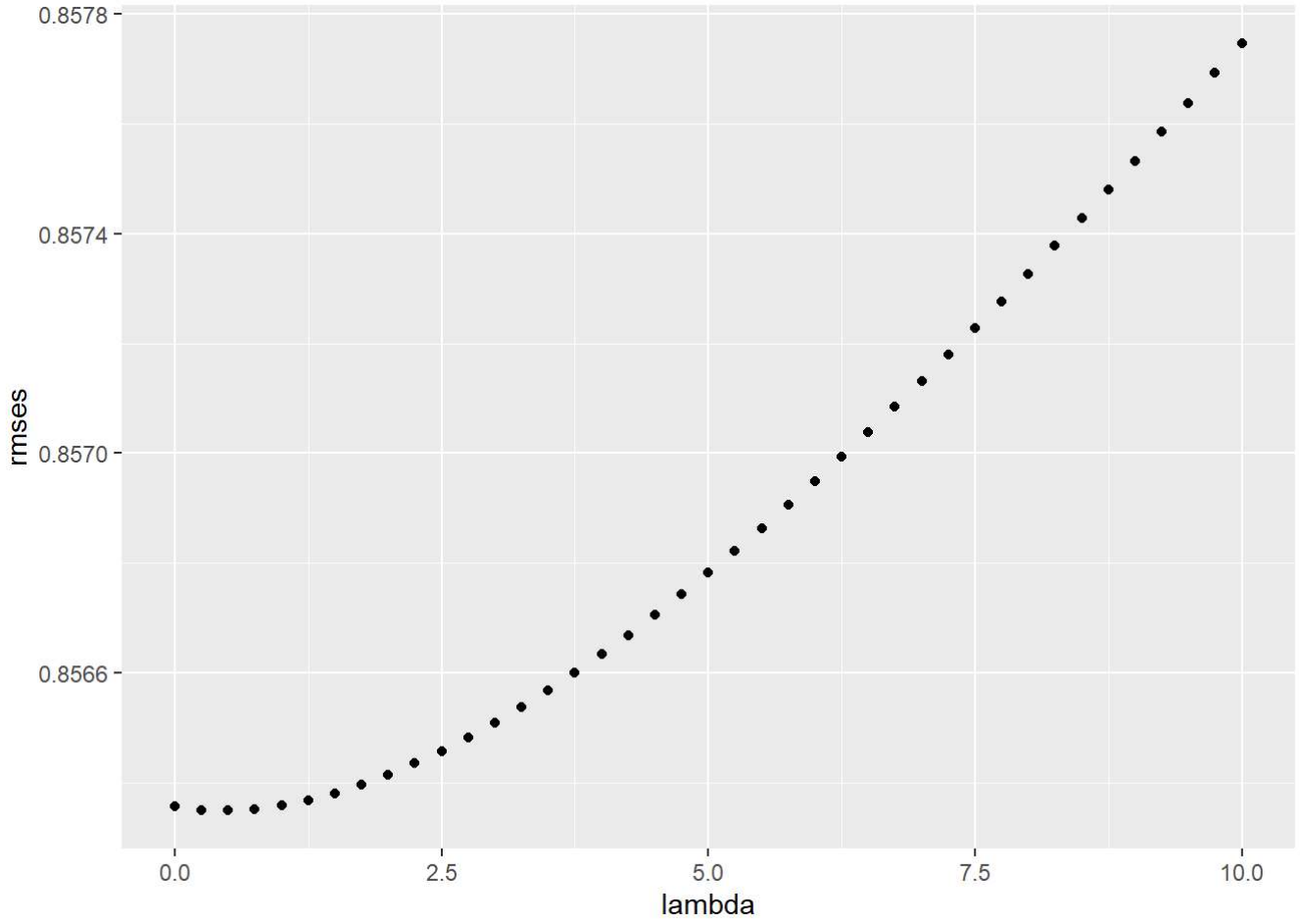
title	b_i	n
Who’s Singin’ Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.237544	4
Life of Oharu, The (Saikaku ichidai onna) (1952)	1.237544	2
Human Condition II, The (Ningen no joken II) (1959)	1.237544	4
Human Condition III, The (Ningen no joken III) (1961)	1.237544	4

```
# When often the worse are rated:
# add number of rating of the "worse" obscure movies

train %>% count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

```
## Joining, by = "movieId"
```

title	b_i	n
Besotted (2001)	-3.012456	1
Hi-Line, The (1999)	-3.012456	1
Accused (Anklaget) (2005)	-3.012456	1
Confessions of a Superhero (2007)	-3.012456	1
War of the Worlds 2: The Next Wave (2008)	-3.012456	2
SuperBabies: Baby Geniuses 2 (2004)	-2.767775	47
Disaster Movie (2008)	-2.745789	30
From Justin to Kelly (2003)	-2.638139	183
Hip Hop Witch, Da (2000)	-2.603365	11
Criminals (1996)	-2.512456	1

```
#############################################################################################
# The supposed “best” and “worst” movies were rated by very few users, in most cases just 1.
# These movies were mostly obscure ones. This is because with just a few users, we have more
# uncertainty. Therefore, larger estimates of bi, negative or positive, are more likely.
#
# Choosing the penalty terms
# We use regularization for the estimate both movie and user effects. We are minimizing:
#
#   1N∑u,i(yu,i−μ−bi−bu)2+λ(∑ib2i+∑ub2u)
#
# Here we use cross-validation to pick a λ:

# use cross-validation to pick a lambda:
#############################################################################################

lambda <- seq(0, 10, 0.25)

rmses <- sapply(lambda, function(l){
  mu <- mean(train$rating)

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

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

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

  return(RMSE(train$rating, predicted_ratings))
})

qplot(lambda, rmses)
```




```
# pick lambda with minimun rmse
lambda <- lambda[which.min(rmses)]
lambda  # print lambda
```

```
## [1] 0.5
```

```
# We use the test set for the final assessment of our third model
# compute movie effect with regularization on train set
b_i <- train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

# compute user effect with regularization on train set
b_u <- train %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

# # compute predicted values on test set
predicted_ratings <-
  test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# create a results table with this and prior approaches
model_3_rmse <- RMSE(test$rating, predicted_ratings)

rmse_results <- bind_rows(rmse_results,
                          tibble(method="Reg Movie and User Effect Model on test set",
                                 RMSE = model_3_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0600537
Movie Effect Model on test set	0.9429615
Movie and User Effects Model on test set	0.8646843
Reg Movie and User Effect Model on test set	0.8645518

```
##############################################################################################
##
# In this case, the long dataset prevents obscure movies with just a few users increase our RMS
E,
# so the regularization does not produce significant improvements in performance using the RMSE
 as
# a metric.
# For this reason we discard the model with Reguralization and focus on the movies and users
# effects to calculate the residuals at next section.
#
# Matrix factorization
# We have described how the model:
#
#   Yu,i=μ+bi+bu+εu,i
#
# accounts for movie to movie differences through the bi and user to user differences through th
e
# bu. But this model leaves out an important source of variation related to the fact that groups
# of movies have similar rating patterns and groups of users have similar rating patterns as wel
l.
# We will discover these patterns by studying the residuals:
#
#   ru,i=yu,i−u^−b^i−b^u
#
# We compute the residuals for train and test set:
##############################################################################################
###

# compute movie effect without regularization on train set
b_i <- train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# compute user effect without regularization on train set
b_u <- train %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - b_i - mu))

# compute residuals on train set
train <- train %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(res = rating - mu - b_i - b_u)

# compute residuals on test set
test <- test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(res = rating - mu - b_i - b_u)
```

```
##############################################################################################
# Matrix Factorization is a popular technique to solve recommender system problem.
# The main idea is to approximate the matrix Rm,n by the product of two matrixes of lower
# dimension: Pk,m and Qk,n.
#
# Matrix P represents latent factors of users. So, each k-elements column of matrix P
# represents each user. Each k-elements column of matrix Q represents each item. So, to find
# rating for item i by user u we simply
# need to compute two vectors: P[,u]' x Q[,i]. Short and full description of package is
# available here.
# This package works with data saved on disk in 3 columns with no headers.
#
# The usage of recosystem is quite simple, mainly consisting of the following steps:
#
#   Create a model object (a Reference Class object in R) by calling Reco().
# (Optionally) call the $tune() method to select best tuning parameters along a set of
# candidate values.Train the model by calling the $train() method. A number of parameters
# can be set inside the function, possibly coming from the result of $tune().
# (Optionally) export the model via $output(), i.e. write the factorization matrices P and Q
# into files or # return them as R objects.Use the $predict() method to compute predicted
# values.
##############################################################################################

# create data saved on disk in 3 columns with no headers
train_data <- data_memory(user_index = train$userId, item_index = train$movieId,
                          rating = train$res, index1 = T)

test_data <- data_memory(user_index = test$userId, item_index = test$movieId, index1 = T)

# create a model object
recommender <- Reco()
```

```
##############################################################################################
##
# The following code trains a recommender model. It will read from a training data source and
# create a model file at the specified location. The model file contains necessary information
# for prediction.
# Train the model by calling the `$train()` method
# some parameters coming from the result of `$tune()`
# This is a randomized algorithm
##############################################################################################
##

set.seed(1, sample.kind="Rounding")
```

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

```
 # if using R 3.5 or earlier, use `set.seed(1)` instead
suppressWarnings(recommender$train(train_data, opts = c(dim = 30, costp_l1 = 0,
                                                        costp_l2 = 0.01, costq_l1 = 0,
                                                        costq_l2 = 0.1, lrate = 0.05,
                                                        verbose = FALSE)))

# We predict unknown entries in the rating matrix on test set.
# use the `$predict()` method to compute predicted values
# return predicted values in memory
predicted_ratings <- recommender$predict(test_data, out_memory()) + mu + test$b_i + test$b_u

# We use the test set for the final assessment
# ceiling rating at 5
ind <- which(predicted_ratings > 5)
predicted_ratings[ind] <- 5

# floor rating at 0.50
ind <- which(predicted_ratings < 0.5)
predicted_ratings[ind] <- 0.5

# create a results table with this approach
model_4_rmse <- RMSE(test$rating, predicted_ratings)

rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie and User + Matrix Fact. on test set",
                                 RMSE = model_4_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0600537
Movie Effect Model on test set	0.9429615
Movie and User Effects Model on test set	0.8646843
Reg Movie and User Effect Model on test set	0.8645518
Movie and User + Matrix Fact. on test set	0.7965118

```
##########################################################################################################
# Lastly, to meet the requirements of the project we calculate the final RMSE on the validation
# dataset.
# We will use the edx dataset provided that ensures that it contains the users who are in
# validation set.
##########################################################################################################

# compute movies effect without regularization on edx set
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# compute users effect without regularization on edx set
b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - b_i - mu))

# compute residuals on edx set
edx <- edx %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(res = rating - mu - b_i - b_u)

# compute residuals on validation set
validation <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(res = rating - mu - b_i - b_u)

# Using recosystem
# create data saved on disk in 3 columns with no headers

edx_data <- data_memory(user_index = edx$userId, item_index = edx$movieId,
                        rating = edx$res, index1 = T)

validation_data <- data_memory(user_index = validation$userId, item_index = validation$movieId,
 index1 = T)

# create a model object
recommender <- Reco()

# Train the model by calling the `$train()` method
# some parameters coming from the result of `$tune()`
# This is a randomized algorithm
set.seed(1, sample.kind="Rounding")
```

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

```
# if using R 3.5 or earlier, use `set.seed(1)` instead

suppressWarnings(recommender$train(edx_data, opts = c(dim = 30, costp_l1 = 0,
                                                      costp_l2 = 0.01, costq_l1 = 0,
                                                      costq_l2 = 0.1, lrate = 0.05,
                                                      verbose = FALSE)))

# use the `$predict()` method to compute predicted values
# return predicted values in memory

predicted_ratings <- recommender$predict(validation_data, out_memory()) + mu +
  validation$b_i + validation$b_u

# ceiling rating at 5
ind <- which(predicted_ratings > 5)
predicted_ratings[ind] <- 5

# floor rating at 5
ind <- which(predicted_ratings < 0.5)
predicted_ratings[ind] <- 0.5

# create a results table with this and prior approaches
model_5_rmse <- RMSE(validation$rating, predicted_ratings)

rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie and User effects and Matrix Fact. on validation s
et",
                                 RMSE = model_5_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0600537
Movie Effect Model on test set	0.9429615
Movie and User Effects Model on test set	0.8646843
Reg Movie and User Effect Model on test set	0.8645518
Movie and User + Matrix Fact. on test set	0.7965118
Movie and User effects and Matrix Fact. on validation set	0.7939559

Summary The data set is very long, so we train models that allow the data set can be adjusted and processed within the available RAM in personal machine.

The long dataset prevents obscure movies with just a few users increase our RMSE, so the regularization does not produce significant improvements in performance using the RMSE as a metric.

We discard the model with Reguralization and focus on the movies and users effects to calculate the residuals.

This residuals were modeled using Matrix Factorization.

We reach a RMSE of 0.794 on validation set using the full edx set for training. The model Movie and User effects and Matrix Factorization achieved this performance using the methodology presented in the course's book without regularization and the R package recosystem.