REPORT: MovieLens

Viriya Thach
2022-11-22

INTRODUCTION

For this project, we will be creating a movie recommendation system using the MovieLens dataset.
The version of **movielens** included in the **dslabs** package is just a small subset of a much larger dataset with millions of ratings.
We will be creating our own recommendation system using the 10M version of the MovieLens dataset to make the computation easier.
We will download the MovieLens data and run code provided to generate our datasets.
We will train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set.
We will use the following code to generate our datasets and develop our algorithm using the **edx** set. For a final test of our final algorithm, we will predict movie ratings in the **validation** set (the final hold-out test set) as if they were unknown. RMSE will be used to evaluate how close our predictions are to the true values in the **validation** set (the final hold-out test set).

MovieLens Dataset

This data set contains 10000054 ratings and 95580 tags applied to 10681 movies by 71567 users of the online movie recommender service MovieLens.

Users were selected at random for inclusion. All users selected had rated at least 20 movies. Unlike previous MovieLens data sets, no demographic information is included. Each user is represented by an id, and no other information is provided.

The data are contained in three files, movies.dat, ratings.dat and tags.dat. Also included are scripts for generating subsets of the data to support five-fold cross-validation of rating predictions. More details about the contents and use of all these files follows.

This and other GroupLens data sets are publicly available for download at GroupLens Data Sets.

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

Data Loading

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

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

# 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)

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

dl <- tempfile()
download.file("https://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")

# if using R 3.6 or earlier:
movies <- as.data.frame(movies) %>% mutate(movieId =
as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))
# if using R 4.0 or later:
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")

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

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

##

Understanding the data

The edx file have 9000055 rows and 6 columns. There are 10677 movies in the edx dataset. There are 69878 different users in the edx dataset. There are total 69,878 users, 10,677 movies, 797 genres.

```
#create summary table
edx_summary <- data.frame(number_of_rows = nrow(edx),
                          number_of_column = ncol(edx),
                          number_of_users = n_distinct(edx$userId),
                          number_of_movies = n_distinct(edx$movieId),
                          average_rating = round(mean(edx$rating),2),
                          number_of_genres = n_distinct(edx$genres),
                          the_first_rating_date =
                            as.Date(as.POSIXct(min(edx$timestamp),
                                               origin = "1970-01-01")),
                          the_last_rating_date =
                            as.Date(as.POSIXct(max(edx$timestamp),
                                               origin = "1970-01-01")))

edx_summary[,1:6] %>%
  kable(caption = "Summary of edx set (part 1)") %>%
  kable_styling(font_size = 10, position = "center",
                latex_options = c("scale_down","HOLD_position"))
```

str(edx)

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885
838983707 838984596 ...
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Adventure|Sci-Fi" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

Summary (edx)

```
##      userId        movieId         rating       timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
```

```
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##    title
##  Length:9000055
##  Class :character
##  Mode  :character
```

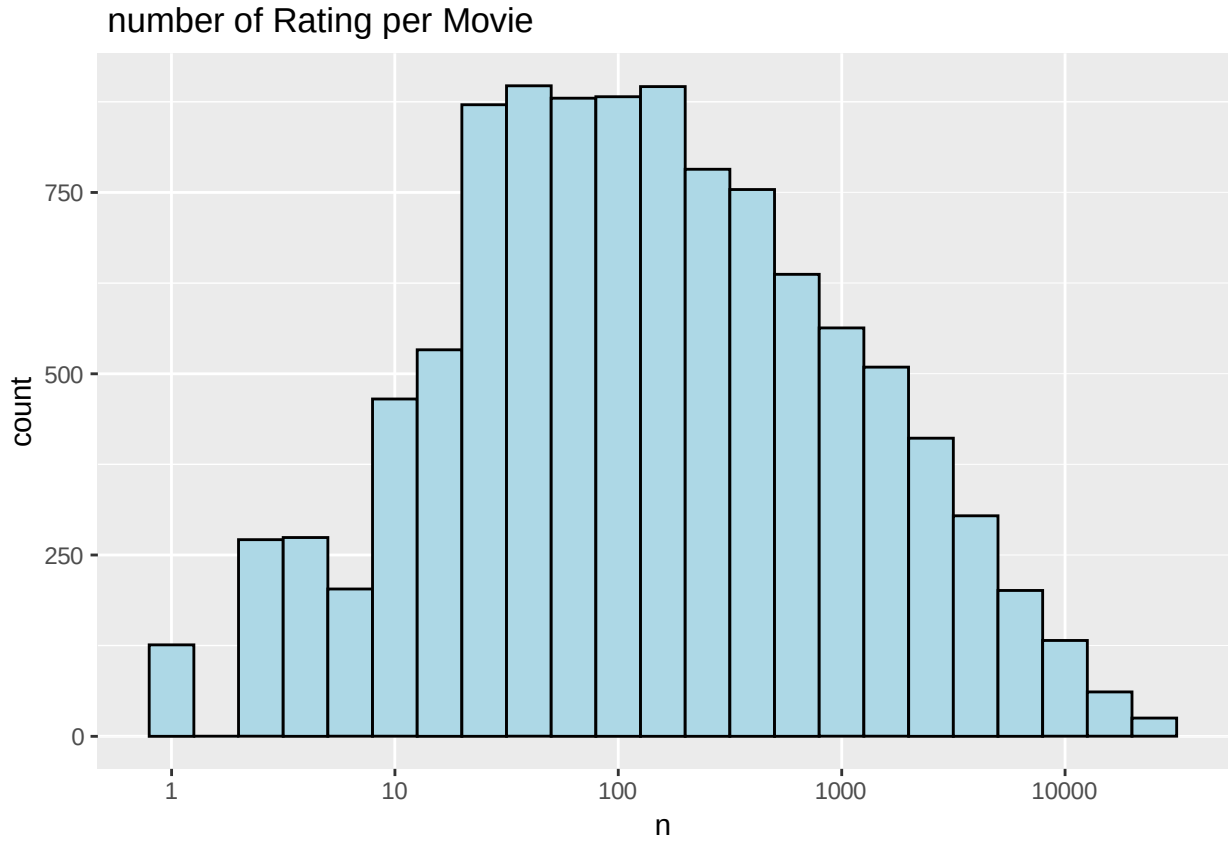
from the summary of data we see that the minimum rating is 1 and max is 5 and the mean for the rating is 3.512 and the mode is 4.0.

```
## Selecting by count
## # A tibble: 5 x 2
## rating count
## ## 1 4 2588430
## 2 3 2121240
## 3 5 1390114
## 4 3.5 791624
## 5 2 711422
```

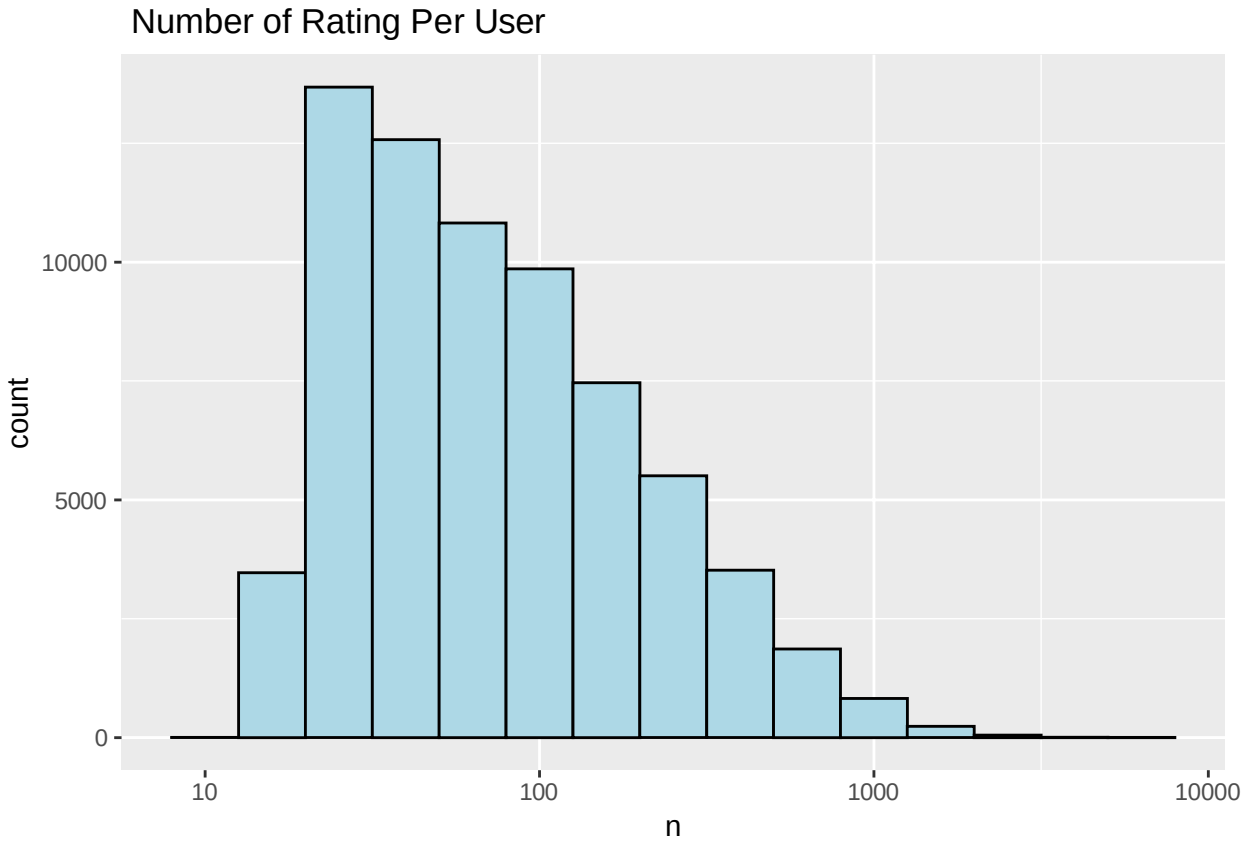
this code prints the number of unique movies and users in the data set:

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

to see how the number of ratings for every movie, we do that by plotting histogram

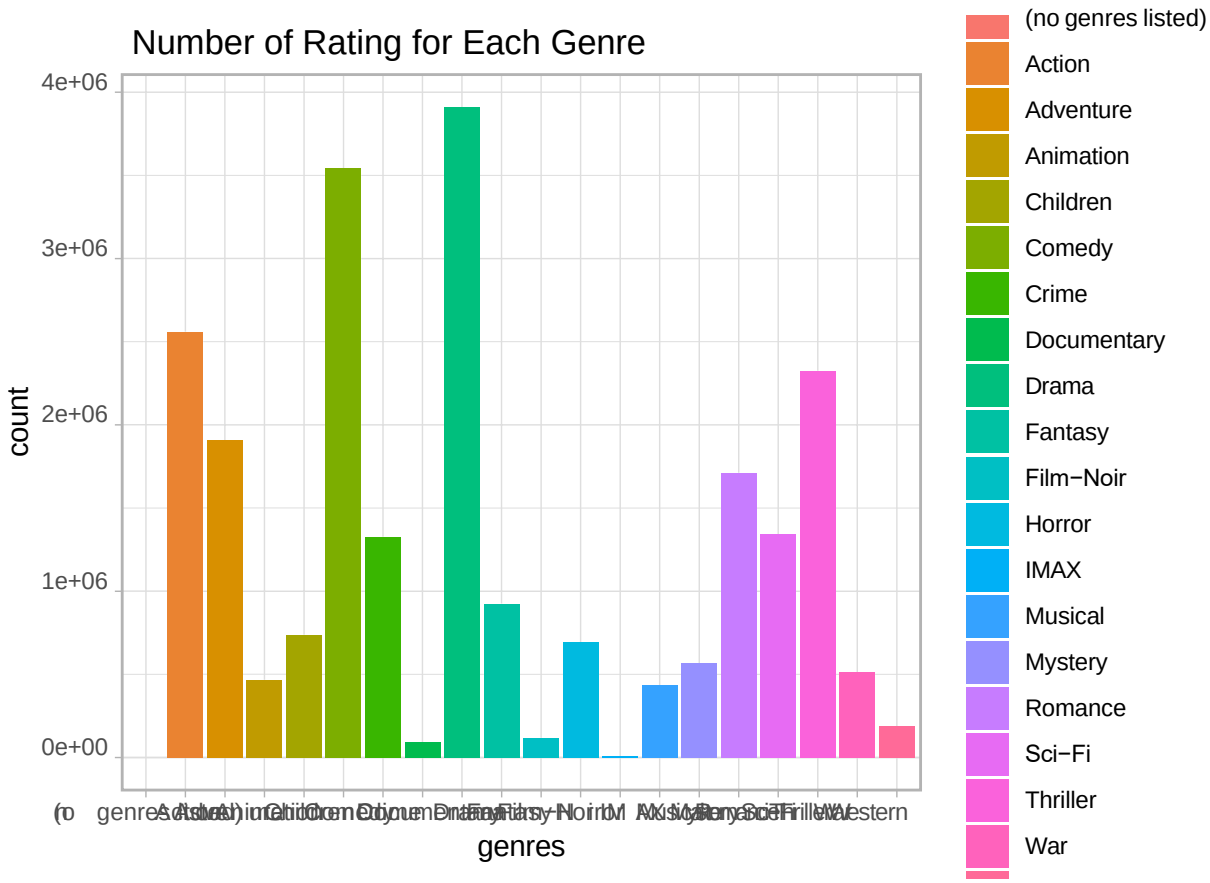


We note that some movies get more ratings it could be due to popularity. Now we visualize the number of ratings for each user



we see that some user are active more than others at rating movies.

Now let’s plot the rating for each movie genre



let's see the top 10 most popular genre

```
## # A tibble: 20 x 2
##    genres                count
##    <chr>                 <int>
##  1 Drama                 3910127
##  2 Comedy                3540930
##  3 Action                2560545
##  4 Thriller              2325899
##  5 Adventure             1908892
##  6 Romance               1712100
##  7 Sci-Fi                1341183
##  8 Crime                 1327715
##  9 Fantasy               925637
## 10 Children              737994
## 11 Horror                691485
## 12 Mystery               568332
## 13 War                   511147
## 14 Animation             467168
## 15 Musical               433080
## 16 Western               189394
## 17 Film-Noir             118541
## 18 Documentary           93066
## 19 IMAX                  8181
## 20 (no genres listed)    7
```

Data Partitioning

before building the model we partition the edx data set into 20% for test set and 80% for the training set.

```
set.seed(1)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
```

Model building and RMSE calculation

The Netflix challenge used typical error loss. They decided on a winner based on the residual mean squared error (RMSE) on a test set. The RMSE will be the measure of accuracy.

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2, na.rm = TRUE))
}
```

First Model

In the first model, we predict the same rating for all movies regardless of the user. a model that assumes the same rating for all movies and users. no bias are considered here. this method assumes the following linear equation is true: *Y u, i* =??+?? *u, i*

```
Mu_1 <- mean(train_set$rating)
Mu_1
```

```
## [1] 3.512482
```

```
naive_rmse <- RMSE(test_set$rating,Mu_1)
naive_rmse
```

```
## [1] 1.059909
```

this code creates a table for the RMSE result to store all the result from each method to compare.

```
rmse_results <- data_frame(method = "Just the average", RMSE = naive_rmse)
rmse_results%>% knitr::kable()
```

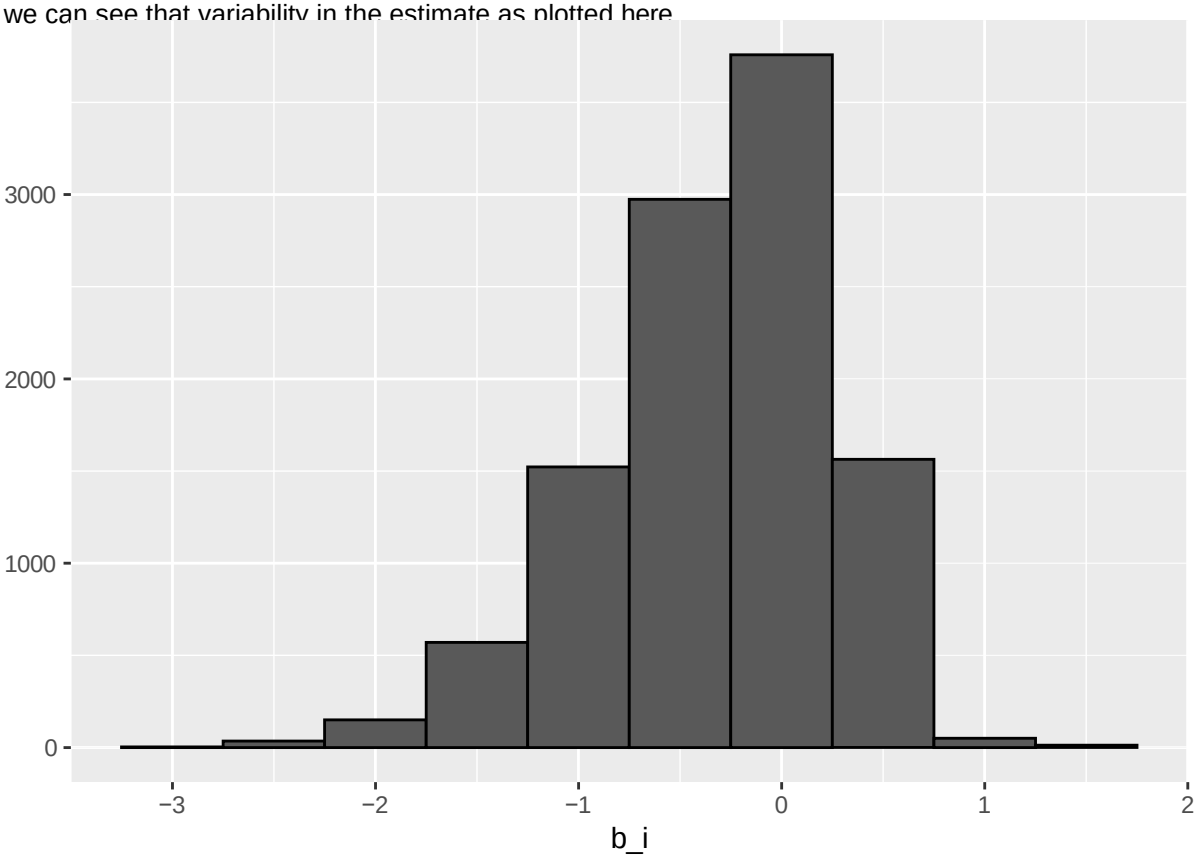
method	RMSE
Just the average	1.059909

Second Model| Movie Effect

As we saw on the exploratory analysis some movies are rated more than other we can augment our previous model by adding the term *b i* to represent the average ranking for movie *i* We can again use least squared to estimate considering the movie bias, in statics they refer to *b* as effect but in the Netflix paper referred them as “Bias” *Y u, i* =?? + *b i* +?? *u, i* Because there are thousands *b i*, each movie gets one, the lm() function will be very slow here. so we compute it using the average this way :

```
Mu_2 <- mean(train_set$rating)
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - Mu_2))
```

we can see that variability in the estimate as plotted here



let's see how the prediction improves after altering the equation $Y_{u,i} = ?? + b_i$

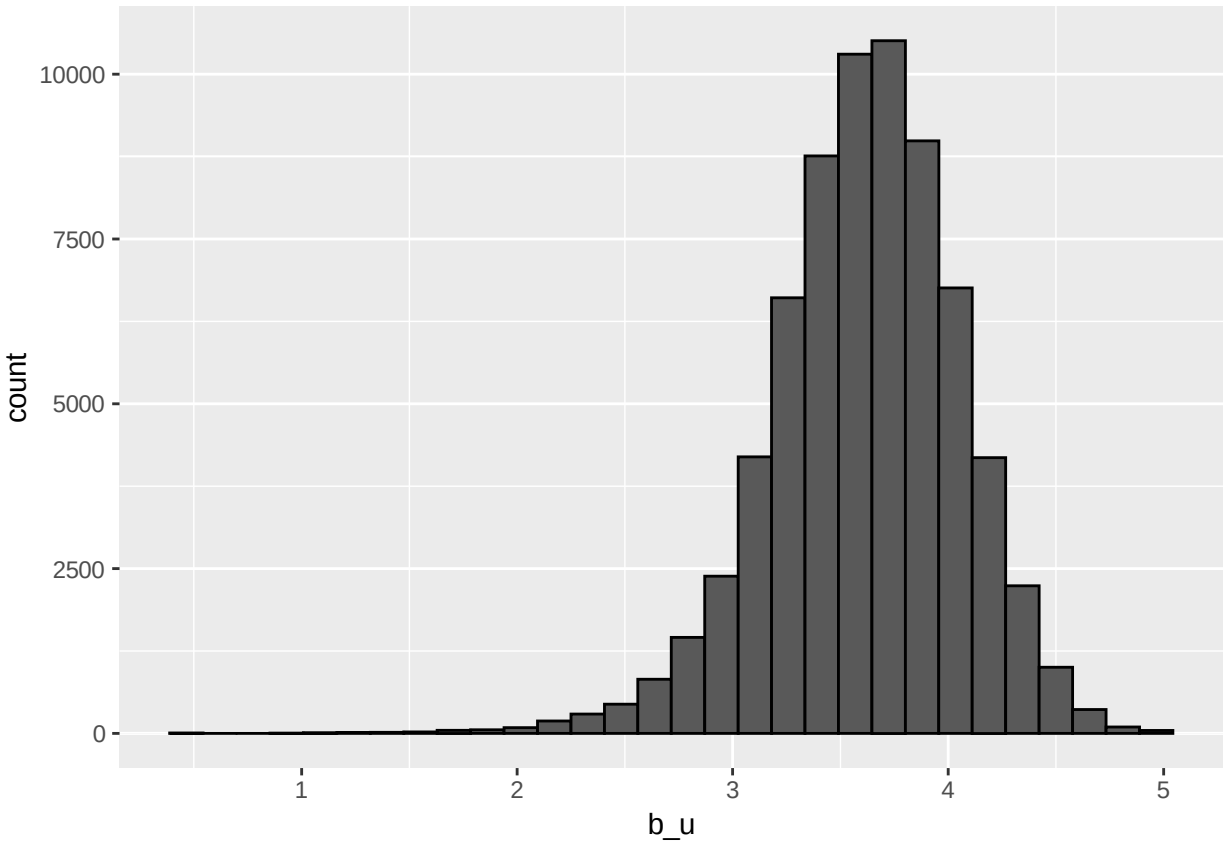
```
predicted_ratings <- Mu_2 + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = model_2_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0599094
Movie Effect Model	0.943742

9

Third Model| User Effect

let's compure the user u for , for those who rated over 100 movies.



Notice that there is substantial variability across users ratings as well. This implies that a further improvement to our model may be *Y u, i* =?? + *b i* +?? *u, i* we could fit this model by using use the lm() function but as mentioned earlier it would be very slow *lm*(*rating as.f actor*(*movieId*) + *as.f actor*(*userId*)) so here is the code

```
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - Mu_2 - b_i))
```

now let's see how RMSE improved this time

```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = Mu_2 + b_i + b_u) %>%
  pull(pred)


model_3_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effects Model",
                                     RMSE = model_3_rmse))
rmse_results%>% knitr::kable()
```

method	RMSE
Just the average	1.0599094
Movie Effect Model	0.9437429

method	RMSE
Movie + User Effects Model	0.8659320

RMSE of the validation set

```
valid_pred_rating <- validation %>%
  left_join(movie_avgs , by = "movieId" ) %>%
  left_join(user_avgs , by = "userId") %>%
  mutate(pred = Mu_2 + b_i + b_u ) %>%
  pull(pred)

model_3_valid <- RMSE(validation$rating, valid_pred_rating)
rmse_results <- bind_rows( rmse_results, data_frame(Method = "Validation Results" , RMSE = model_3_val
rmse_results%>% knitr::kable()
```

method	RMSE	Method
Just the average	1.0599094	NA
Movie Effect Model	0.9437429	NA
Movie + User Effects Model	0.8659320	NA
NA	0.8664515	Validation Results

Conclusion

The movie effect and user+movie effect the best RMSE given by the third model. for further analysis more complicated prediction using the release year of the movie as a bias considering old movies such as the 60 or 80 periods as another genre for a better predicting model. a linear model for precision is recommended.