Movie Lens Project Report

By

Syed Reyadh

Introduction

A recommender system or a recommendation system is a subclass of information filtering system that seeks to predict the “rating” or “preference” a user would give to an item. Recommendation systems use ratings that users have given items to make specific recommendations. Companies that sell many products to many customers and permit these customers to rate their products, use customers rating to predict their preferences or rating for another item. Netflix uses a recommendation system to predict if user rating for specific movies. motivated by some of the approaches taken by the winners of the Netflix challenges, On October 2006, Netflix offered a challenge to the data science community: improve our recommendation algorithm by 10% and win a million dollars. In September 2009, the winners were announced. You can read a good summary of how the winning algorithm was put together here and a more detailed explanation here. We will now show you some of the data analysis strategies used by the winning team.

This assignment is to accomplish a similar goal which is to build a recommendation system that recommends movies based on a rating scale.

Data Set

for this project the MovieLens Data set collected by GroupLens Research and can be found in MovieLens web site (http://movielens.org).

Data Loading

The data set is loaded using the code provided by course instructor in this link https://bit.ly/2Ng6tVW which split the data into edx set and 10% validation set. the edx set will be split into training and test set, and validation set will be used to final evaluation.

```
#############################################################
# Create edx set, validation set, and submission file
#############################################################

# 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 1.2.1 --

## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.2     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0

## -- Conflicts ------------------------------------------ tidyverse_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

## Loading required package: lattice

##
## Attaching package: 'caret'

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

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

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

ratings <- read.table(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)
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 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")
```

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

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

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

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

before the analysis we check for any NA value

```r
anyNA(edx)
```

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

Data Summary and Exploratory Data Analysis

After loading the data set we start by looking at the data structure and type we can see that there is six variable (userId,movieID,rating,timestamp,title,genres).as shown the year need to be separated from title if needed for prediction also the genres need separation if needed.

```
str(edx)
```

```
## '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 8:
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
```

```
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              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :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.

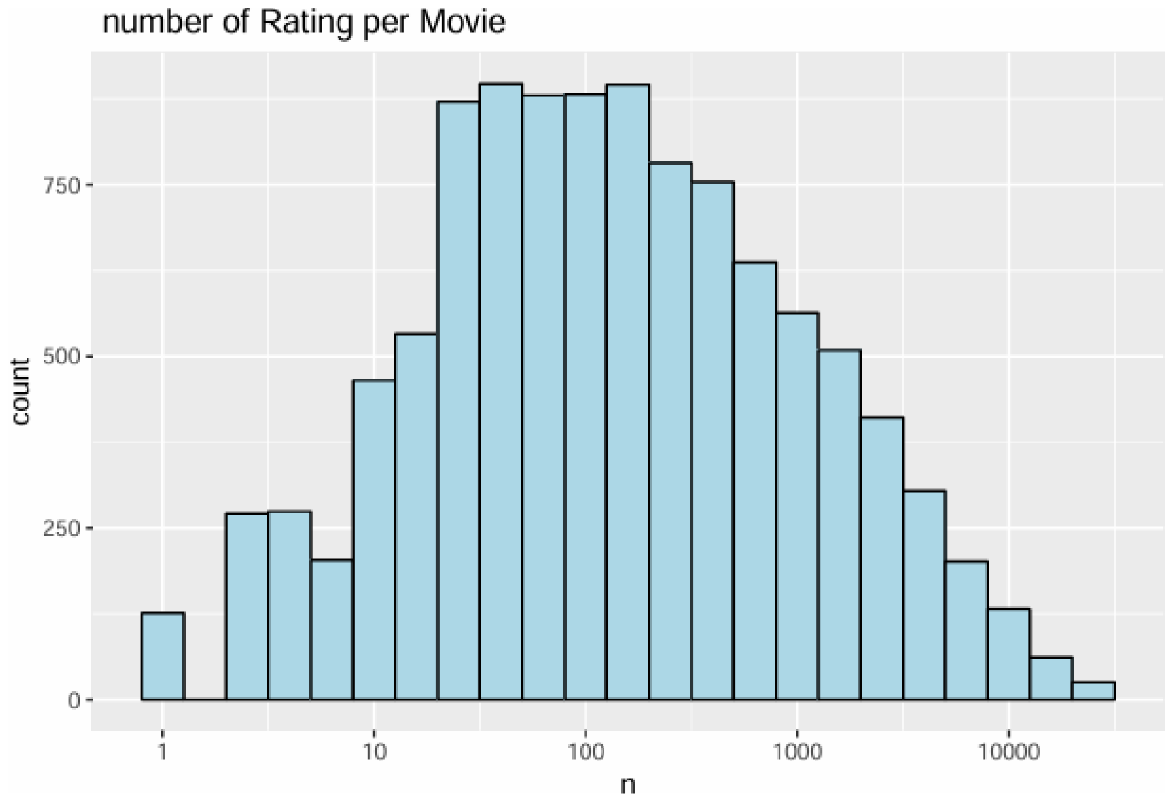
```
## Selecting by count
```

```
## # A tibble: 5 x 2
##   rating   count
##    <dbl>   <int>
## 1      4 2588430
## 2      3 2121240
## 3      5 1390114
## 4    3.5  791624
## 5      2  711422
```

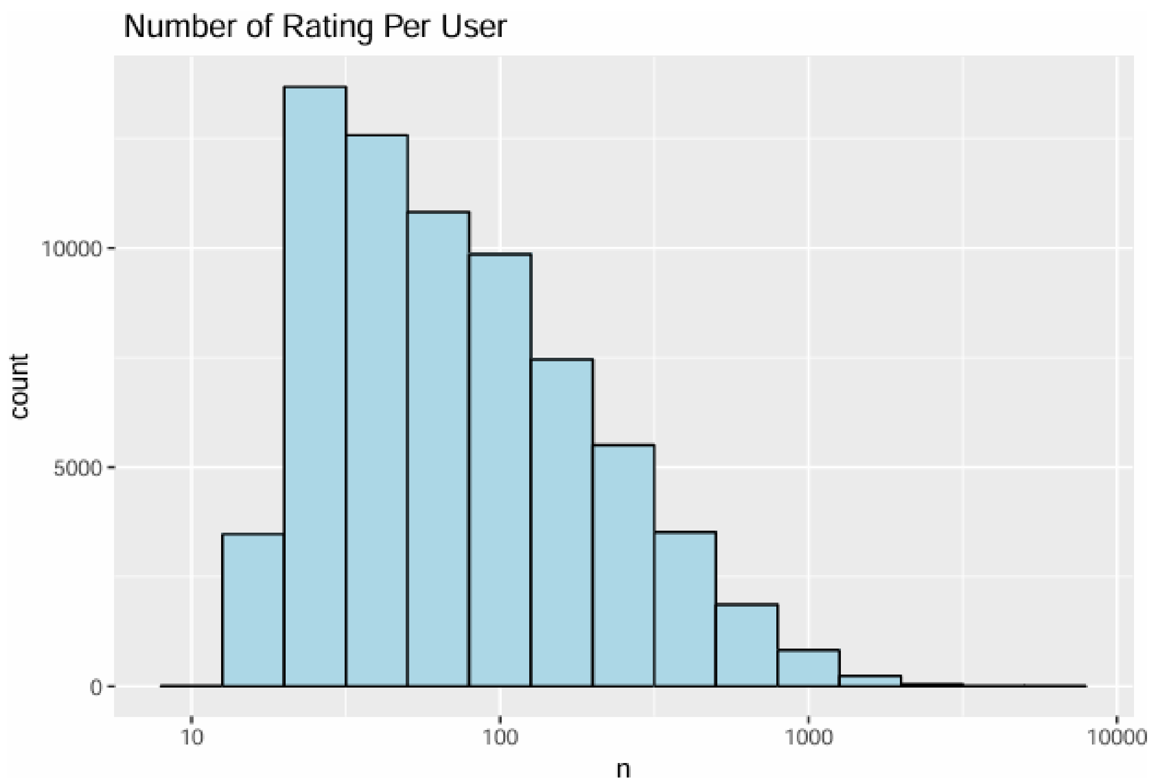
This code prints the number of unique movies and users in the data set:

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

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

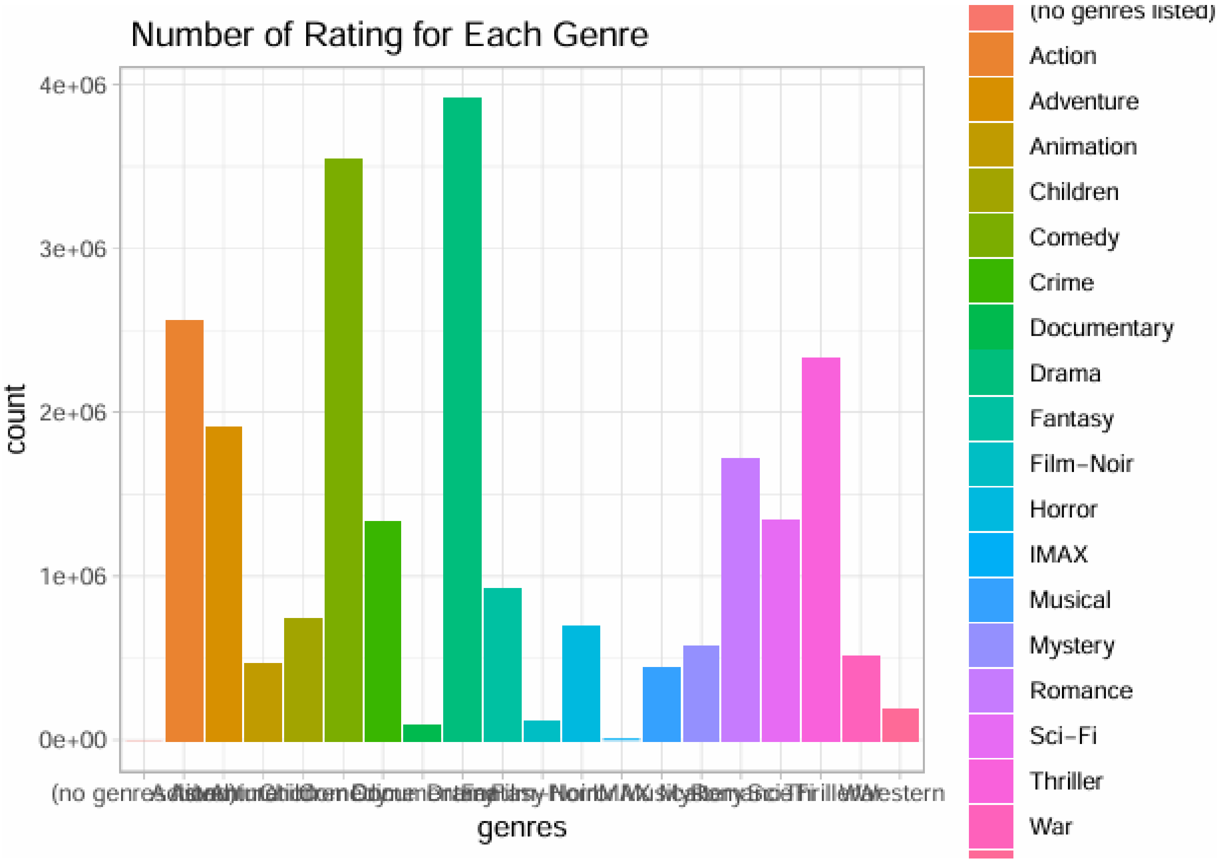


We note that some movies get more ratings it could be due to popularity. Now we visualize the number of ratings for each user.



we can see that some users 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 is 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()
```

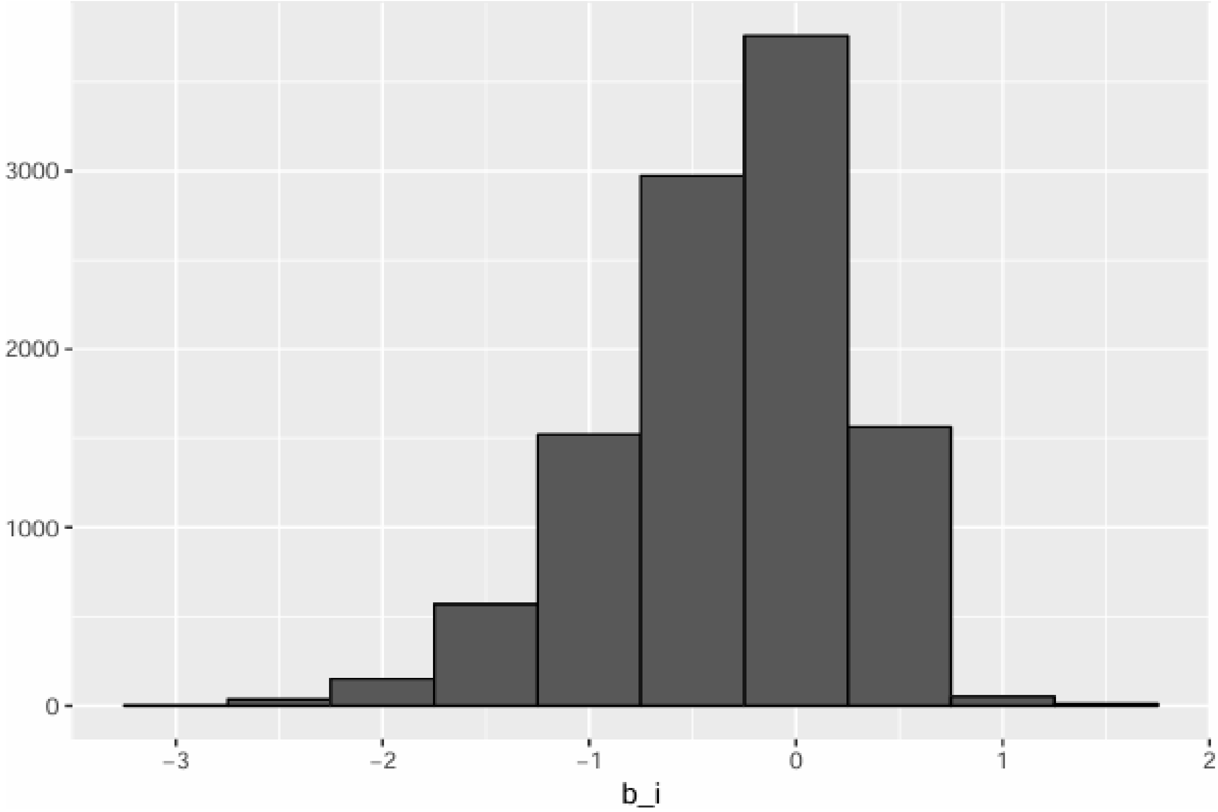
method	RMSE
Just the average	1.059909

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 Yu,i =??+bi

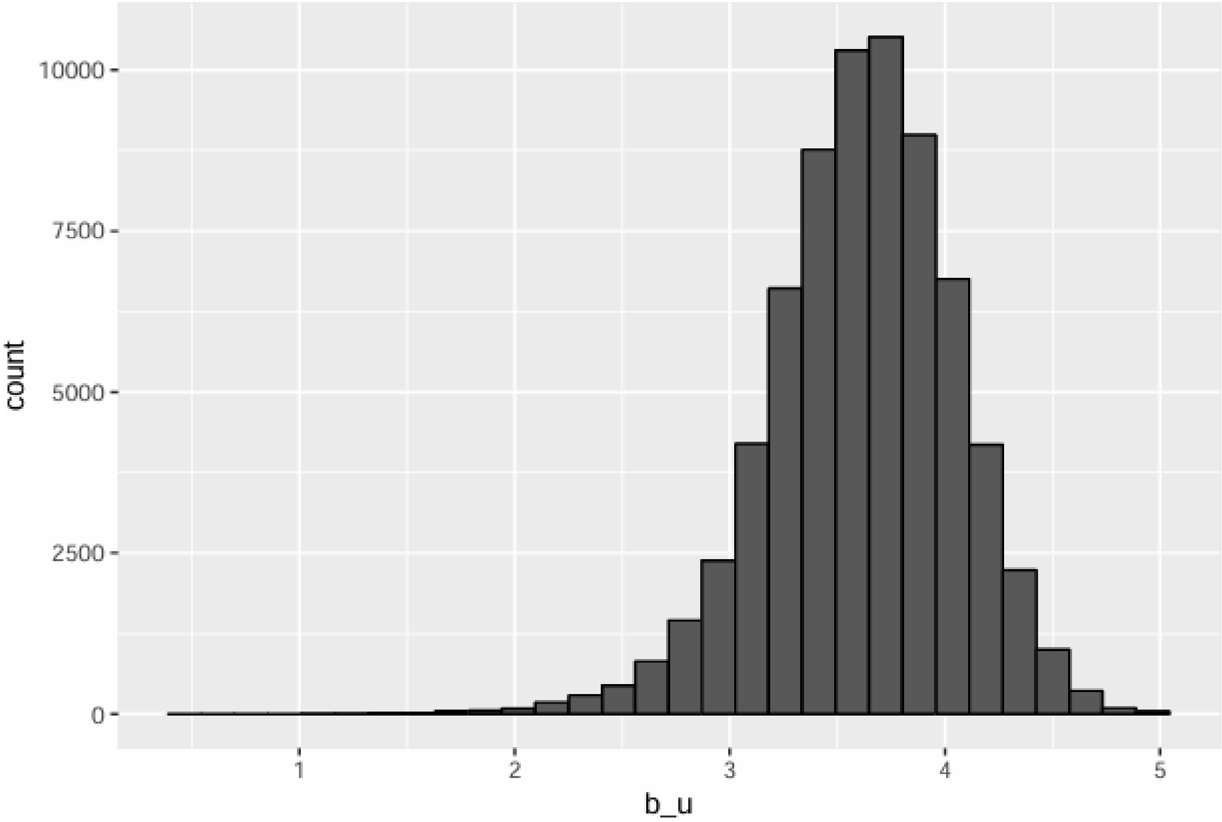
```
predicted_ratings <- Mu_2 + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

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	0.9437429

Third Model – User Effect

Let's compare 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 Yu,i =??+bi+??u,i we could fit this model by using the lm() function but as mentioned earlier it would be very slow lm(ratingas.factor(movieId)+as.factor(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
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

I have developed a naive approach, 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.