Capstone Project - Netflix Challenge

Eduardo Sthory

22/11/2019

Introduction

This analysis is based on the Netflix database that was the product of a challenge that was made in order to improve the movie recommendation system. The movie rating uses a rating range of 0 to 5. In the present project, it aims to train a machine learning algorithm that will use the inputs of a data set to predict movie ratings. Said data set can be downloaded from the following link:

http://files.grouplens.org/datasets/movielens/ml-10m.zip

For movie recommendation systems to work, they need user-made ratings, in the same way, when the goal is the sale of products, customers can rate the transaction and in this way the data for user training are supplied to the algorithms.In the case of Netflix, it uses a system to predict how many stars a user will give to a specific movie. When a movie's prediction gets a high rating, then it is recommended to other users.

There are different biases in movie ratings, such as movie genres and lack of ratings. To create machine learning models, these biases must be kept in mind.

For this project, the residual mean square error (RMSE) will be used, as Netflix did in its challenge.

Data Wrangling

edx code for Cleaning Data and create Train and Validation Sets

Netflix data is not available to work with them, however the GroupLens research laboratory generated a database with approximately 20 million ratings for more than 27,000 movies and more than 138,000 users. For this project, a subset of only approximately 10 million qualifications was used.

In order to work with the data, the wrangling and the creation of a data set for training and validation were previously done, this is the code:

```
################################
# 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 1.2.1 --
```

```
## v ggplot2 3.2.1     v purrr   0.3.2
## 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 --------------------------------------------------- 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
```

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

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

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

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

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

```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 <- 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
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)
```

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

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

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

In order not to perform the download and wrangling process each time the program is run, it is stored in the following files:

```
# ------------------- Save Date -----------------------------------
# Save data frame (Not to perform all the "Data Wrangling" again)
# save(edx,file="edx.Rda")
# save(validation,file="validation.Rda")
```

In this way, every time we want to run it, we only load this way:

```
# load data frame wrangling
# load("edx.Rda")
# load("validation.Rda")
```

Required libraries

```
if(!require(ggplot2)) install.packages("ggplot2")
if(!require(tidyverse)) install.packages("tidyverse")
if(!require(tidyr)) install.packages("tidyr")
```

MovieLens Data Exploration

We can see a sample of the data with the following code:

```
edx %>%
   as_tibble()
```

```
## # A tibble: 9,000,061 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     185      5 838983525 Net, The (1995)    Action|Crime|Thriller
##  3      1     231      5 838983392 Dumb & Dumber (199~ Comedy
##  4      1     292      5 838983421 Outbreak (1995)    Action|Drama|Sci-Fi|Thri~
##  5      1     316      5 838983392 Stargate (1994)    Action|Adventure|Sci-Fi
##  6      1     329      5 838983392 Star Trek: Generat~ Action|Adventure|Drama|S~
##  7      1     355      5 838984474 Flintstones, The (~ Children|Comedy|Fantasy
##  8      1     356      5 838983653 Forrest Gump (1994) Comedy|Drama|Romance|War
##  9      1     362      5 838984885 Jungle Book, The (~ Adventure|Children|Roman~
## 10      1     364      5 838983707 Lion King, The (19~ Adventure|Animation|Chil~
## # ... with 9,000,051 more rows
```

In data, a row is a rating made by a user to a specific movie. To know how many movies and users make up this sample we can run the following code:

```
edx %>%
   summarize(users = n_distinct(userId),
             movies = n_distinct(movieId))
```

```
##   users movies
## 1 69878  10677
```

The Overall Mean Rating is calculated as follows:

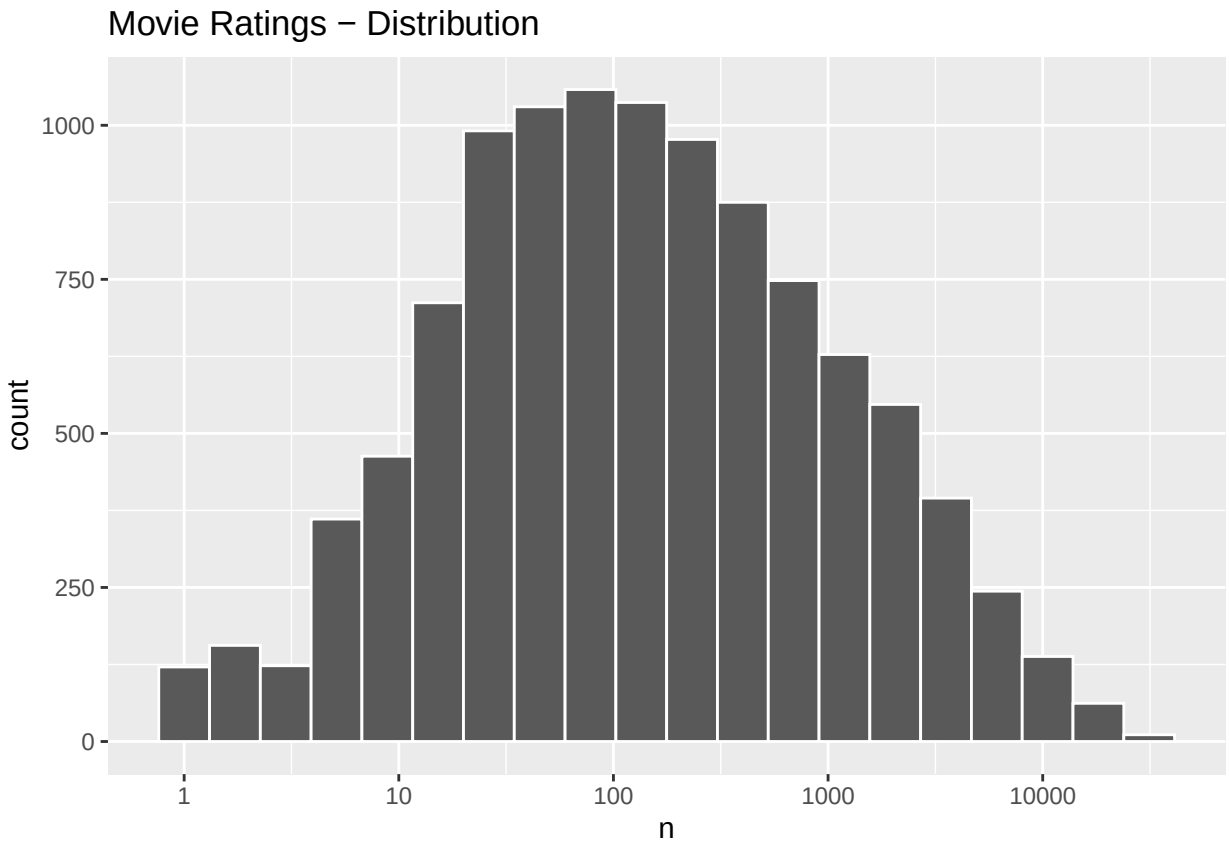
```
rating_mean <- mean(edx$rating)
rating_mean
```

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

Distributions

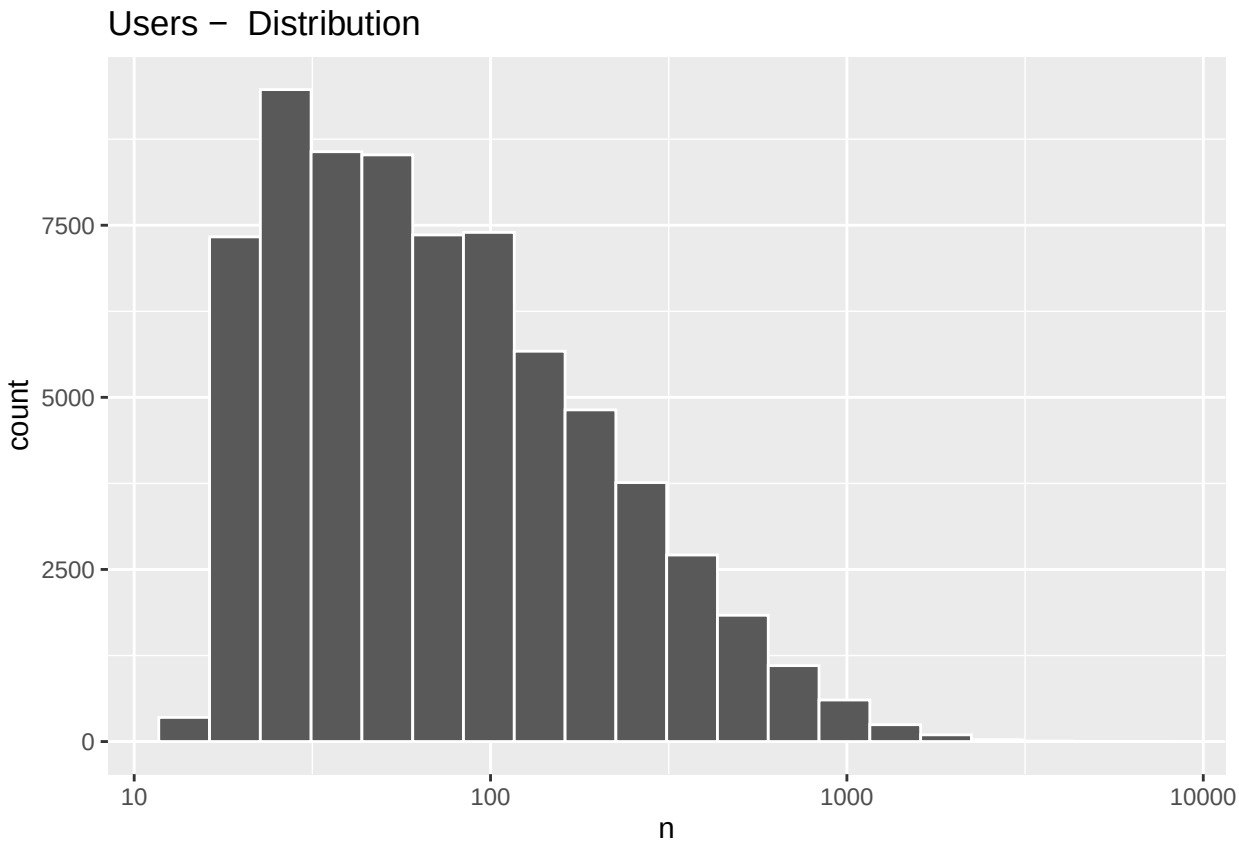
Checking the distribution of the data, we can see that some films are rated more than others, this is their distribution:

```
edx %>%
   count(movieId) %>%
   ggplot(aes(n)) +
   geom_histogram(bins = 20, color = "white") +
   scale_x_log10() +
   ggtitle("Movie Ratings - Distribution")
```



As supposed, there are more blockbuster movies than others and because of that they have more ratings. Another observation is that there are users who perform more qualifications than others:

```
edx %>%
   count(userId) %>%
   ggplot(aes(n)) +
   geom_histogram(bins = 20, color = "white") +
   scale_x_log10() +
   ggtitle("Users -  Distribution")
```



Data Analysis

Loss Function

As an error metric we will use the residual mean square error (RMSE), in this way, we can determine the performance of the algorithms that we are going to test.

Definitions:

$y_{u,i}$ –> rating for movie i by user u

$\hat{y}_{u,i}$ –> prediction

N –> number of user per movie combinations

The rmse formula is:

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

The rmse is the mistake made when predicting a movie rating. If rmse is greater than 1, it is because the error is greater than a star, that would be bad. The written function that calculates the rmse will be:

```
# RMSE function
RMSE <- function(preds, values_true){
  sqrt(mean((values_true - preds) ^ 2))
}
```

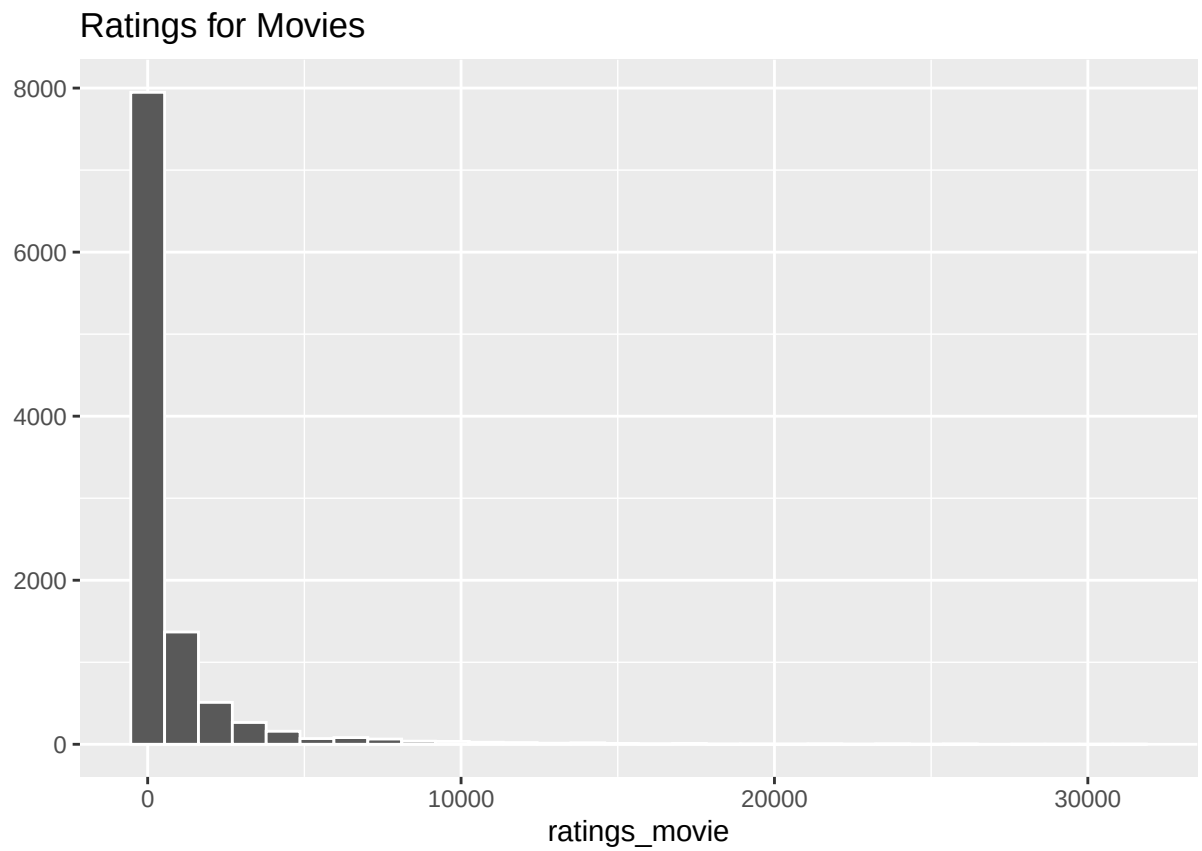
Model 1: Movie effects

As we saw earlier, there are movies that receive more ratings than others, likewise, there are also movies that receive better ratings, and this can be seen in the data. We will take the average of subtracting the "rating" minus the general average of the rating "rating_mean" (calculated above)

```
model_movie_effects <- edx %>%
   group_by(movieId) %>%
   summarize(movie_effect = mean(rating - rating_mean),
             ratings_movie = n())
```

The following graph shows that there is a lot of variability.

```
model_movie_effects %>%
   qplot(ratings_movie,
         geom ="histogram",
         bins = 30,
         data = .,
         colour = I("white")) +
   ggtitle("Ratings for Movies")
```



Now, we make the prediction, calculate the rsme and attach the results of the models in the data frame "rmse_results", let's see how it was

```
# Predictions

preds <- validation %>%
  left_join(model_movie_effects, by = "movieId") %>%
  mutate(mix_effect = movie_effect) %>%
  mutate(predictions = rating_mean + mix_effect) %>%
  .$predictions

# rmse calculation

rmse <- RMSE(preds, validation$rating)

# Print rmse

rmse
```

```
## [1] 0.9437046
```

```
# add model and rmse to table results

rmse_results <- data.frame(Method = "Model 1: Movie effects",
                           Rmse = rmse)

# Print table results
```

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

Method	Rmse
Model 1: Movie effects	0.9437046

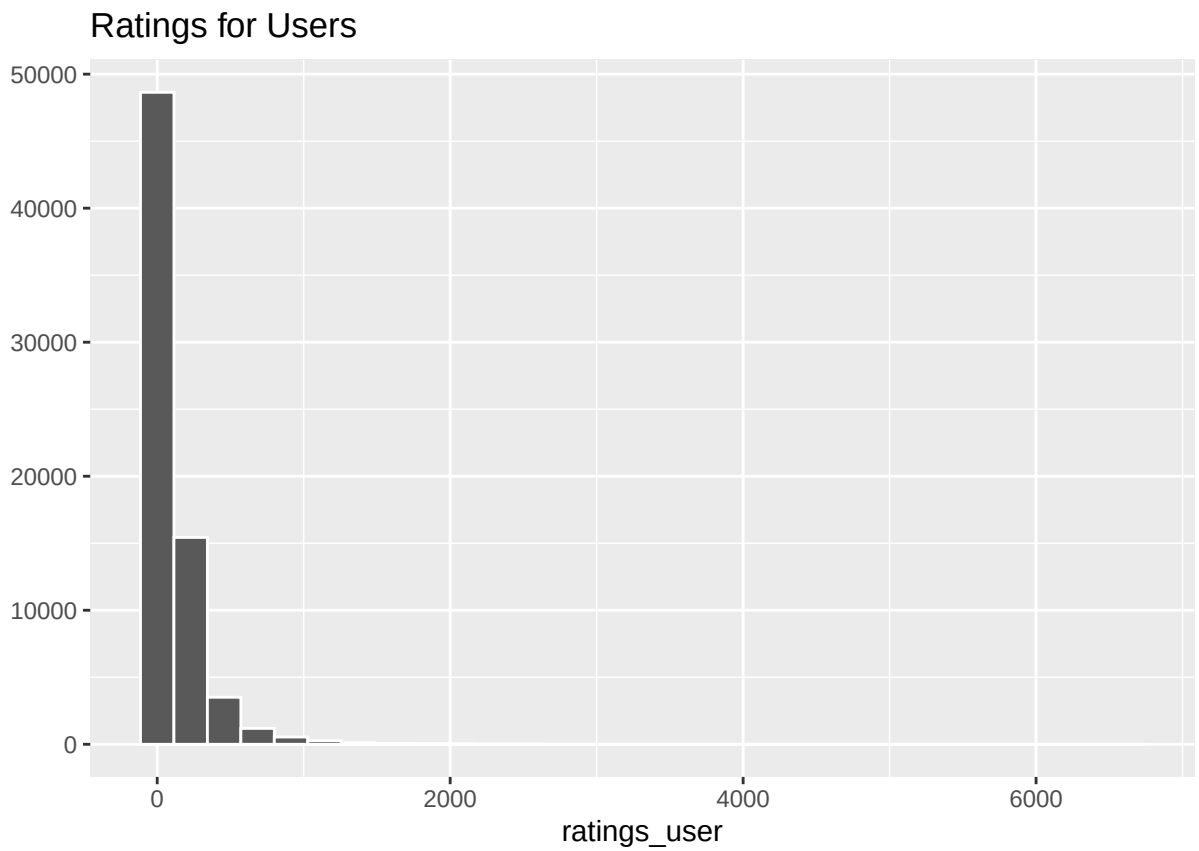
Model 2: User effects

Previously we saw, there are users who make more qualifications than others, we take the previous model and add the effect of the users to see how much our model improves.

```
# ------------- Model 2: Movie + User effects -----------------------

model_user_effects <- edx %>%
   group_by(userId) %>%
   left_join(model_movie_effects, by = "movieId") %>%
   summarize(user_effect = mean(rating - movie_effect - rating_mean),
             ratings_user = n())
```

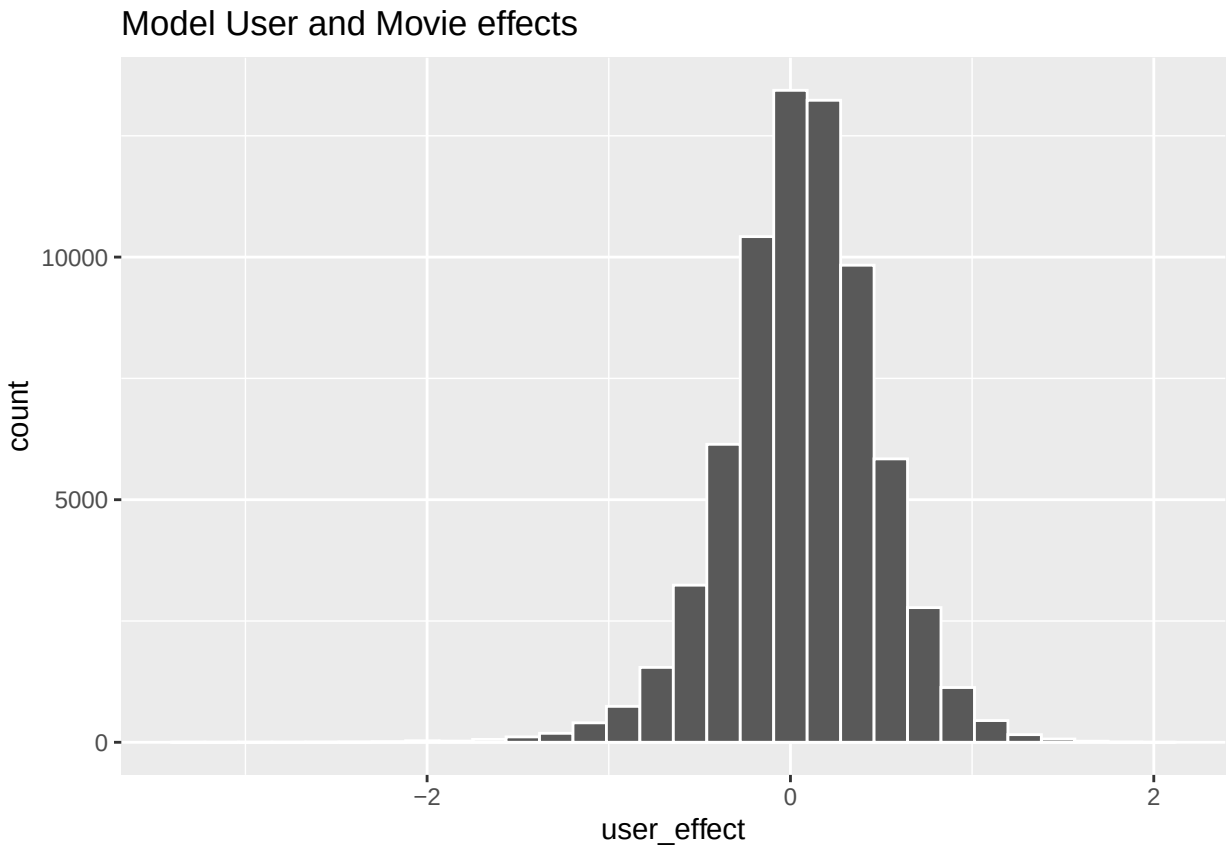
The following graph shows that there is a lot of variability in users effects.

```
# Ratings for User + Movie Model Plot

model_user_effects %>%
   qplot(ratings_user,
         geom ="histogram",
         bins = 30,
         data = .,
         colour = I("white")) +
   ggtitle("Ratings for Users")
```



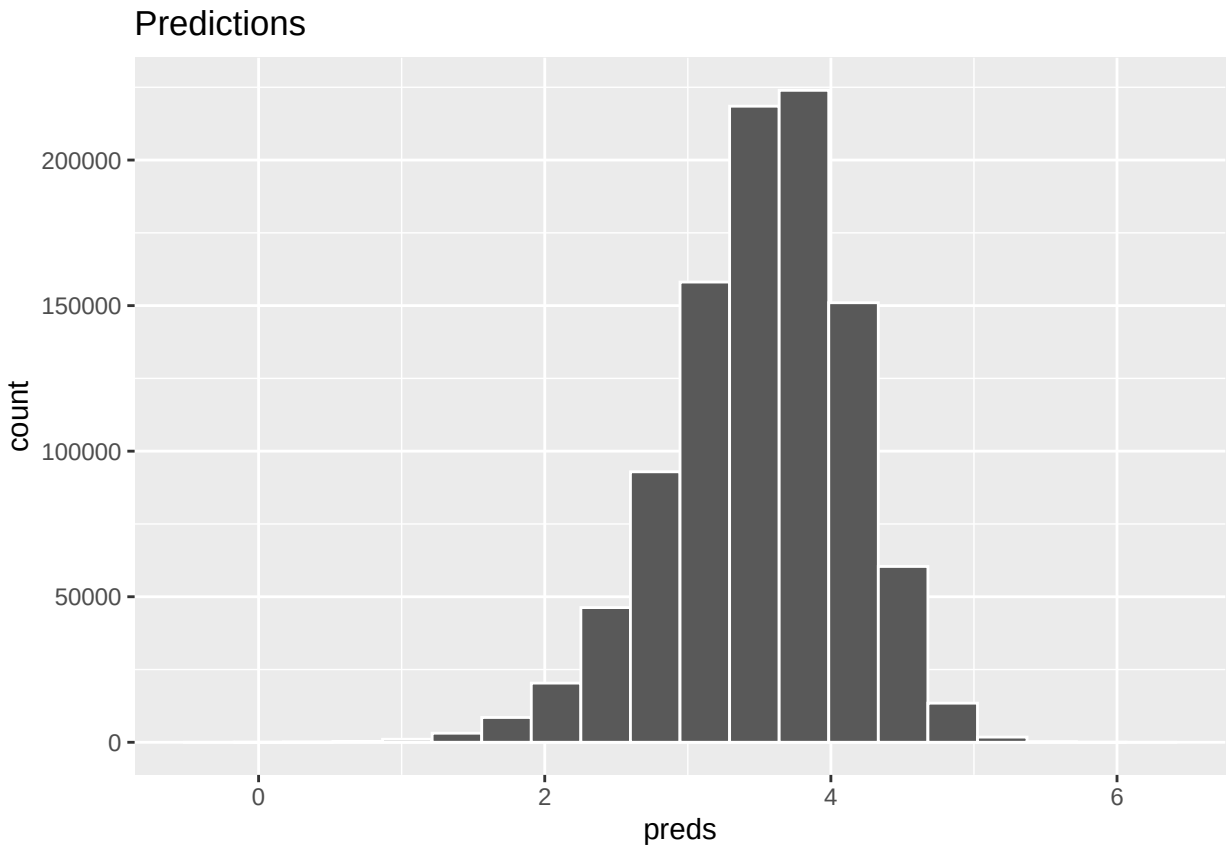
Here the Users and Movie effects

```
model_user_effects %>%
   ggplot(aes(user_effect)) +
   geom_histogram(bins = 30,
                  color = "white") +
   ggtitle("Model User and Movie effects")
```



Now, we make the prediction, calculate the rsme and attach the results of model 2 in the "rmse_results" data frame, let's see if the result is improved.

```
# Predictions using Movie + user Model

preds <- validation %>%
   left_join(model_movie_effects, by = "movieId") %>%
   left_join(model_user_effects, by = "userId") %>%
   mutate(mix_effect = user_effect + movie_effect) %>%
   mutate(predictions = rating_mean + mix_effect) %>%
   .$predictions

# Plot Predictions

preds %>%
  data.frame() %>%
  ggplot(aes(preds)) +
  geom_histogram(bins = 20, color = "white") +
  ggtitle("Predictions")
```




```
# rmse calculation

rmse <- RMSE(preds, validation$rating)

# Print rmse

rmse
```

```
## [1] 0.8655329
```

```
# add model and rmse to table results

rmse_results <- bind_rows(rmse_results,
                          data.frame(Method = "Model 2: Movie + user effects",
                          Rmse = rmse))
```

```
## Warning in bind_rows_(x, .id): Unequal factor levels: coercing to character
```

```
## Warning in bind_rows_(x, .id): binding character and factor vector, coercing
## into character vector

## Warning in bind_rows_(x, .id): binding character and factor vector, coercing
## into character vector
```

```
# Print table results

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

Method	Rmse
Model 1: Movie effects	0.9437046
Model 2: Movie + user effects	0.8655329

Model 3: Regularization movie and user effects

The regularization tries to limit the total variability of the effect, for this a term is added that softens the results to avoid overfitting.

This is because when you train a model, it will try to adapt as much as possible to the training data, since it wants to avoid the error as much as possible. But in machine learning this is not optimal since an over-adjustment in training data can produce a low predictability in real data.

The formula we will use will be the following:

$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu})$$

Definitions:

n_i –> Will be the number of ratings made for the movie i.

λ –> Is a tuning parameter.

b_i –> Estimate movie bias 'b_i' with regualrization

μ –> Overall Mean Rating

Thus, when n_i is very large and the option is stable, it turns out that the λ penalty in practice is ignored, given that $n_i + \lambda \approx n_i$.

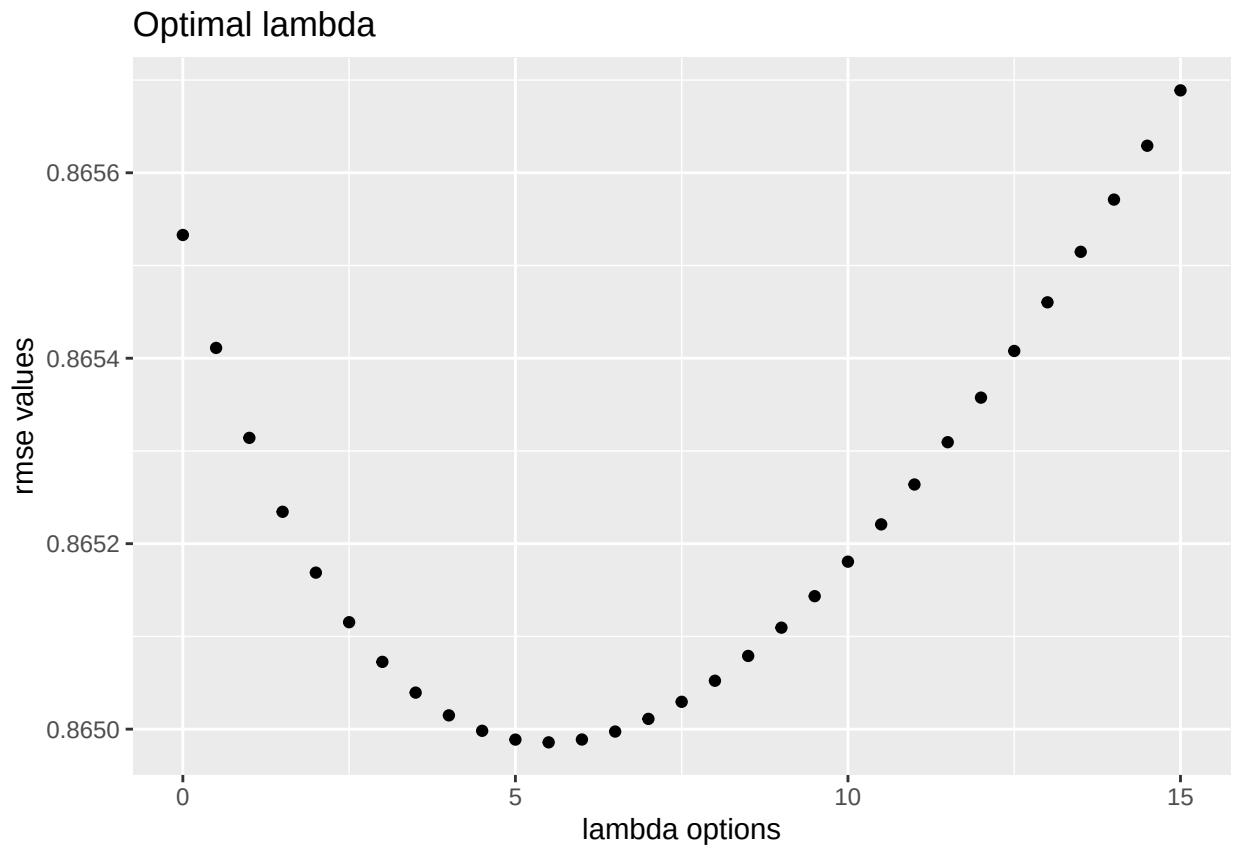
But if n_i is small, then $\hat{b}_i(\lambda)$ is reduced to 0. In this way if λ is larger, then the result shrinks.

```
# lambda options
l <- seq(0, 15, 0.5)

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

   # bi calculation

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

   # bu calculation

   bu <- edx %>%
      left_join(bi, by="movieId") %>%
      group_by(userId) %>%
      summarize(bu = sum(rating - bi - rating_mean)/(n()+l))

   # Predict

   predicted_ratings <-
      validation %>%
      left_join(bi, by = "movieId") %>%
      left_join(bu, by = "userId") %>%
```

```
      mutate(pred = rating_mean + bi + bu) %>%
      pull(pred)

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

Here we can see the plot for lambda options and rmses:

```
qplot(l, rmses,
      xlab = "lambda options",
      ylab = "rmse values") +
      ggtitle("Optimal lambda")
```



Optimal λ of the final model is:

```
# lambda value that minimize rmse

lambda_optimal <- l[which.min(rmses)]
lambda_optimal
```

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

Finally, we add the model to the general results table and see the comparisons.

```
# add model and rmse to table results

rmse_results <-
   bind_rows(rmse_results,
             data_frame(Method = "Model 3: Regularized Movie + User effect",
```

```
                                    Rmse = min(rmses)))
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

```
# Print table results

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

Method	Rmse
Model 1: Movie effects	0.9437046
Model 2: Movie + user effects	0.8655329
Model 3: Regularized Movie + User effect	0.8649857

```
# print minimum rmse (final result)

min(rmses)
```

```
## [1] 0.8649857
```

```
# MovieLens Grading Rubric
# RMSE (25 points)
# 0 points: No RMSE reported AND/OR code used to generate the RMSE appears
# to violate the edX Honor Code.
# 5 points: RMSE >= 0.90000 AND/OR the reported RMSE is the result of overtraining
#                                  (validation set used for anything except
#                                  reporting
#                                  the final RMSE value)
#
# 10 points: 0.86550 <= RMSE <= 0.89999
# 15 points: 0.86500 <= RMSE <= 0.86549
# 20 points: 0.86490 <= RMSE <= 0.86499
# 25 points: RMSE <= 0.8649
```

Comparison of the final result with the objective

```
min(rmses) <= 0.8649
```

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

```
# or

0.8648177 <= 0.8649
```

```
## [1] TRUE
```

That is, the goal was achieved

Results

Now we can review results, and we see that the second model is an improvement of the first and the last with an RMSE of less than 0.8649 is the best. Our last one was able to predict the movie ratings for the films in the validation set with an RMSE of 0.8648177.

Conclusion

Our first model that took into consideration the effect of the film, the RMSE was 0.9439087, which is not so bad. In the second model, we took the effect of the user and the model gave an RMSE of 0.8653488, which is an improvement to the previous model. Finally, our regularized model gave us the lowest RMSE value: 0.8648177, better than the previous two models, this means that the film and the effect of the user can be regularized, because the highest ratings of a Small sample size of users.

It is possible that regularizing other effects such as year and genres can also improve the results, but we have already achieved the proposed goal that was an RMSE <= 0.8649 and is computationally expensive.