MovieLens Project Report

Raúl Galindo Martínez

11/01/2019

Table of Contents

1. Introduction

We will create a movie recommendation system by using MovieLens dataset. It was created in 1997 by GroupLens Research. It is a web-based recommender system that recommends movies for users to watch, based on the users' preferences. It contains about 11 million ratings for about 8500 movies.

Objective

We will train a machine learning algorithm using the inputs in the test set to predict movie ratings in the validation set. The movie rating predictions will be compared to the true ratings in the validation set using RMSE.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{u,i}\left(\hat{Y}_{u,i} - Y_{u,i}\right)^2}$$

The goal is to get RMSE less or equal to 0.8649

Data description

To make the computation easier we will use the 10M version of the MovieLens dataset

This data set contains a ratings file and a movies file. All ratings are contained in the file ratings.dat. Each line of this file represents one rating of one movie by one user. There are 10,000,054 observations (rows) Each line of the movies file represents one movie, movie file has 10,681 movies (obs)

Downloading data

```
# Creating a temporary file and downloading MovieLens 10M dataset
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

# Creating a data.frame 'ratings' from ratings.dat file
ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

# Creating a matrix called 'movies' from movies.dat file
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)

# Adding column names
colnames(movies) <- c("movieId", "title", "genres")

# movies as data.frame
movies <- as.data.frame(movies) %>%
  mutate(movieId = as.numeric(levels(movieId))[movieId],
         title = as.character(title),
         genres = as.character(genres))

# Glimpsing data sets
glimpse(ratings)
```

```
## Observations: 10,000,054
## Variables: 4
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ movieId   <int> 122, 185, 231, 292, 316, 329, 355, 356, 362, 364, 37...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5...
## $ timestamp <int> 838985046, 838983525, 838983392, 838983421, 83898339...
```

```
glimpse(movies)
```

```
## Observations: 10,681
## Variables: 3
## $ movieId <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,...
## $ title   <chr> "Toy Story (1995)", "Jumanji (1995)", "Grumpier Old Me...
## $ genres  <chr> "Adventure|Animation|Children|Comedy|Fantasy", "Advent...
```

2. Methods/Analysis

We will convert data (ratings and movies) from its raw form to the tidy form called movielens, where each row represents a rating given by one user to one movie, that way movielens dataset facilitates the analysis.

Data Wrangling

We add a new variable derived from the title of the movie that represents the release year of the movie, we also add the age of the movie by subtracting the release year from the current year. Finally we create the data frame movielens by adding title and genres to ratings.

```
# defining a pattern to extract the movie year from the title
pattern <- "\\((\\d{4})\\)$"
# \\( = escape of '('
# ( = start of the group
# \\d{4} = 4 digits
# ) = end of the group
# \\) = escape of ')'
# $ = end of the string

# Adding a new column with the year of release
movies <- mutate (movies, yearRelease = as.numeric(str_match(str_trim(title), pattern)[,2]))

# Adding a new column with movie age
yearToday <- year(today())
movies <- mutate (movies, movieAge = yearToday - yearRelease)

# Adding title and genres to ratings into a data.frame called movielens
movielens <- left_join(ratings, movies, by = "movieId")

glimpse(movielens)
```

```
## Observations: 10,000,054
## Variables: 8
## $ userId      <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
## $ movieId     <dbl> 122, 185, 231, 292, 316, 329, 355, 356, 362, 364, ...
## $ rating      <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,...
## $ timestamp   <int> 838985046, 838983525, 838983392, 838983421, 838983...
## $ title       <chr> "Boomerang (1992)", "Net, The (1995)", "Dumb & Dum...
## $ genres      <chr> "Comedy|Romance", "Action|Crime|Thriller", "Comedy...
## $ yearRelease <dbl> 1992, 1995, 1994, 1995, 1994, 1994, 1994, 1994, 19...
## $ movieAge    <dbl> 27, 24, 25, 24, 25, 25, 25, 25, 25, 25, 25, 25, 25...
```

Splitting Data

Movielens dataset will be partitioned into two sets, the first one called edx will be use to train the algorithms, the second one called validation will be used as a test set. We pretend we don't know the outcome of the test

set. Validation set is 10% of the movielens dataset.

```
# Setting the seed of R's random number generator
# to be able to reproduce the random objects like test_index
set.seed(1, sample.kind="Rounding")

# createDataPartition:
# generates indexes for randomly splitting the data into training and test sets
# Validation set will be 10% of MovieLens data
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)
```

Data exploration and visualization

We know there are 10,000,054 ratings and 10,681 movies, let's see how many users there are

```
# Number of users
ratings %>% summarize(n_users = n_distinct(userId))
```

```
##   n_users
## 1   69878
```

Let's check the data structure of edx and validation datasets

```
#data structure
glimpse(edx)
```

```
## Observations: 9,000,055
## Variables: 8
## $ userId      <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
## $ movieId     <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, ...
## $ rating      <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,...
## $ timestamp   <int> 838985046, 838983525, 838983421, 838983392, 838983...
## $ title       <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (...
## $ genres      <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action...
## $ yearRelease <dbl> 1992, 1995, 1995, 1994, 1994, 1994, 1994, 1994, 19...
## $ movieAge    <dbl> 27, 24, 24, 25, 25, 25, 25, 25, 25, 25, 25, 25, 26...
```

edx dataset is 90% of movilens and contains samples of all the movies and users

```
# number of users, movies, genres and ratings of movielens and edx
movielens %>% summarize(n_users = n_distinct(userId),
                        n_movies = n_distinct(movieId),
                        n_genres = n_distinct(genres),
                        n_ratings = n())
```

```
##   n_users n_movies n_genres n_ratings
## 1   69878    10677      797  10000054
```

```
edx %>% summarize(n_users = n_distinct(userId),
                  n_movies = n_distinct(movieId),
                  n_genres = n_distinct(genres),
                  n_ratings = n())
```

```
##   n_users n_movies n_genres n_ratings
## 1   69878    10677      797   9000055
```

A movie can belong to more than one genre

```
#first 10 obs, there might be more than one genre per movie
movies %>% slice(1:10)
```

```
##    movieId                               title
## 1        1                    Toy Story (1995)
## 2        2                      Jumanji (1995)
## 3        3             Grumpier Old Men (1995)
## 4        4            Waiting to Exhale (1995)
## 5        5 Father of the Bride Part II (1995)
## 6        6                           Heat (1995)
## 7        7                        Sabrina (1995)
## 8        8                 Tom and Huck (1995)
## 9        9                 Sudden Death (1995)
## 10      10                    GoldenEye (1995)
##                                          genres yearRelease movieAge
## 1  Adventure|Animation|Children|Comedy|Fantasy        1995       24
## 2                   Adventure|Children|Fantasy        1995       24
## 3                              Comedy|Romance        1995       24
## 4                        Comedy|Drama|Romance        1995       24
## 5                                      Comedy        1995       24
## 6                       Action|Crime|Thriller        1995       24
## 7                              Comedy|Romance        1995       24
## 8                         Adventure|Children        1995       24
## 9                                      Action        1995       24
## 10                  Action|Adventure|Thriller        1995       24
```

Users tend to rate higher than lower, the most common rates are 4,3,5, the less common ratings are 2.5, 1.5 and 0.5.

```
# Summary
summary(edx$rating)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.000   4.000   3.512   4.000   5.000
```

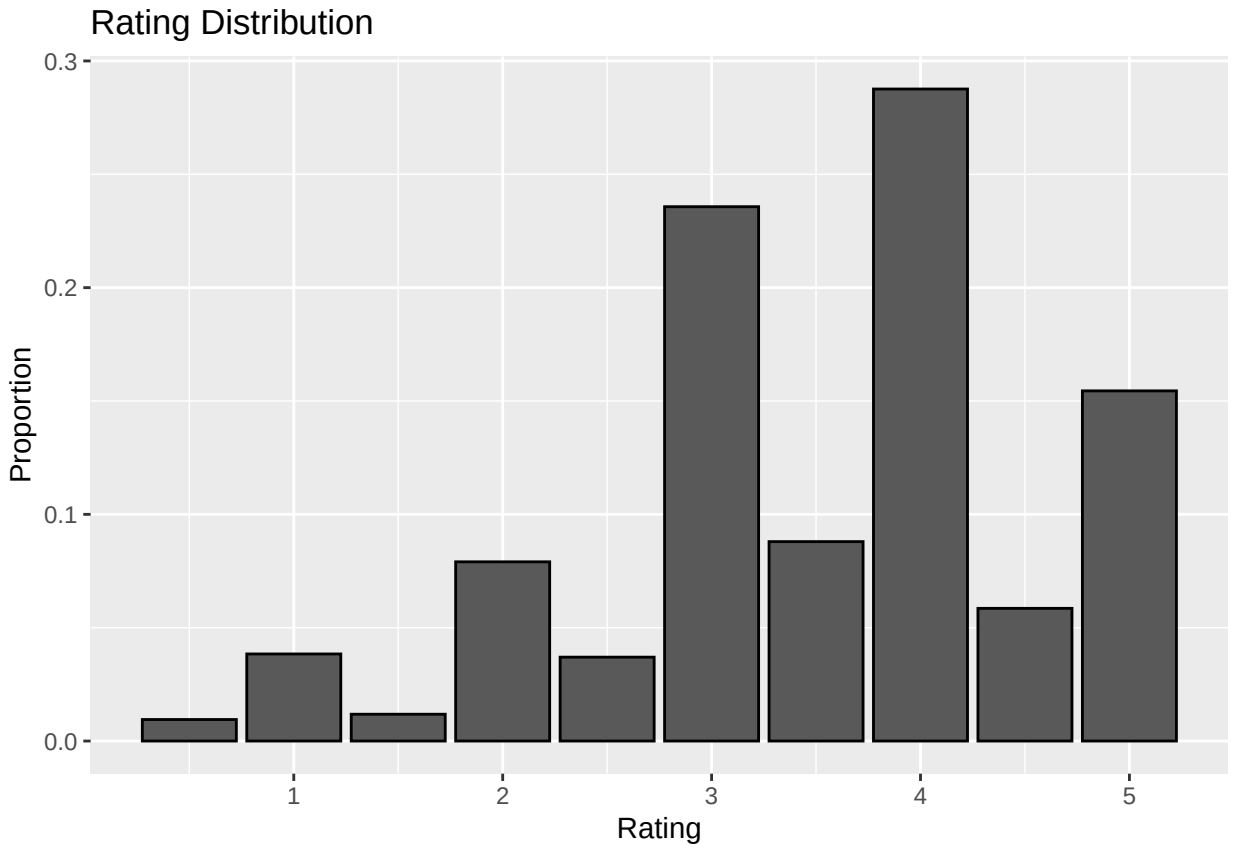
```
# Adding proportion of rating
tab <- edx %>% count(rating) %>% mutate(proportion = n/sum(n))
```

```
# Ordering by proportion
tab %>% arrange(desc(proportion)) %>%
  kable(booktabs = T) %>%
  kable_styling( "latex", latex_options = "striped", full_width = "F", position = "left")
```

rating	n	proportion
4.0	2588430	0.2876016
3.0	2121240	0.2356919
5.0	1390114	0.1544562
3.5	791624	0.0879577
2.0	711422	0.0790464
4.5	526736	0.0585259
1.0	345679	0.0384085
2.5	333010	0.0370009
1.5	106426	0.0118250
0.5	85374	0.0094859

The below graph shows the rating distribution

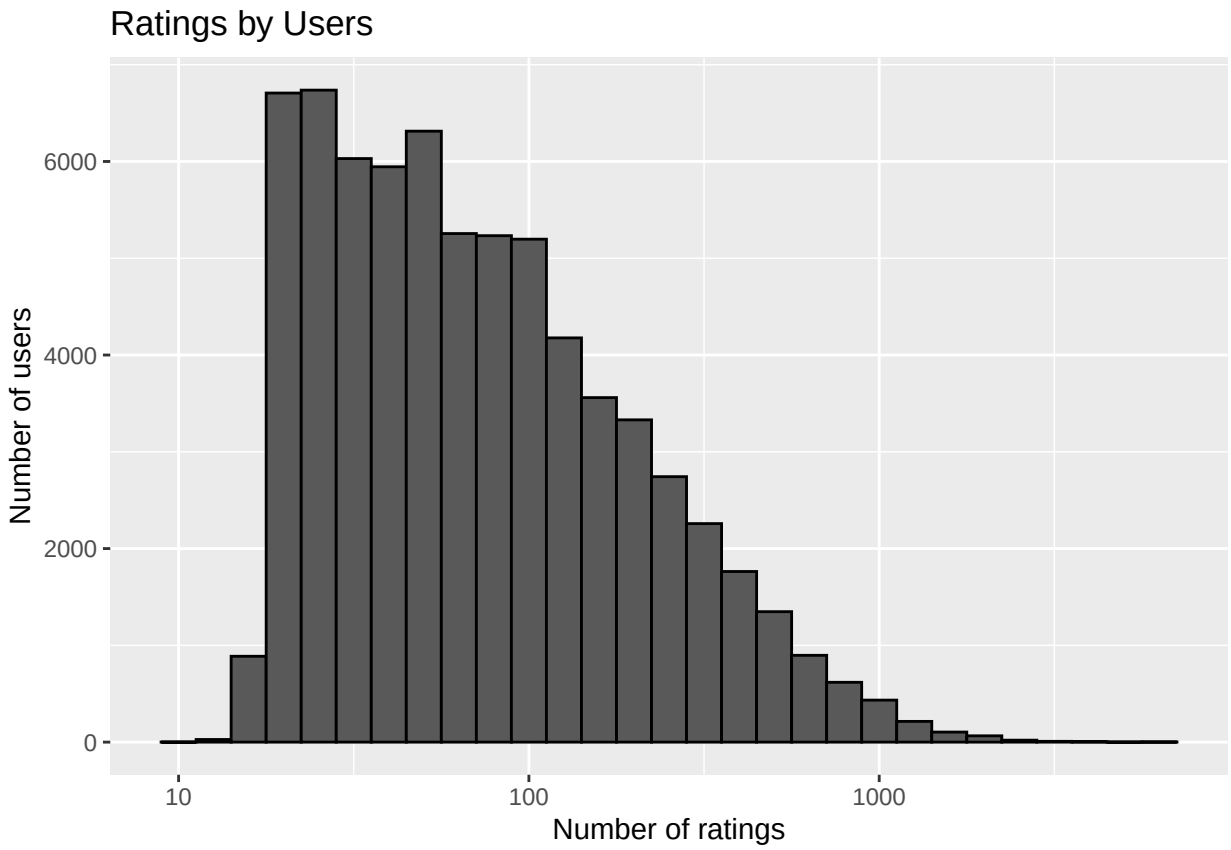
```
# Visualizing rating distribution
tab %>%
  ggplot(aes(rating,proportion)) +
  geom_bar(color = "black", stat = "identity") +
  xlab("Rating") +
  ylab("Proportion") +
  ggtitle("Rating Distribution")
```



Not every user are equally active at rating movies, we can see that there is a bias at rating users

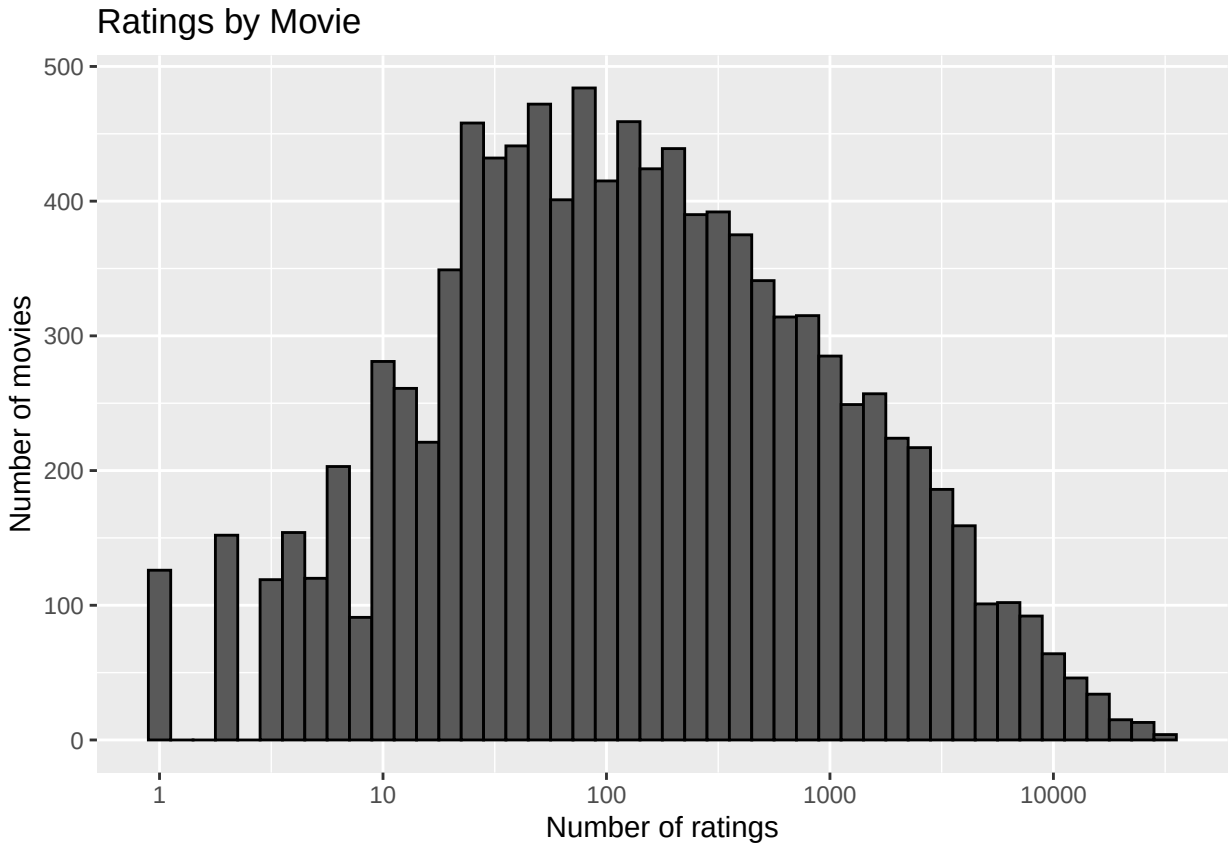
```
# Distribution of Rating by Users
edx %>% group_by(userId) %>% summarize(n = n()) %>%
  ggplot(aes(n)) +
  geom_histogram(binwidth=0.1, color = "black") +
```

```
  scale_x_log10() +
  xlab("Number of ratings") +
  ylab("Number of users") +
  ggtitle("Ratings by Users")
```



Blockbuster movies get way more rating than independent movies (movie bias)

```
# Distribution of Rating by Movie
edx %>% group_by(movieId) %>% summarize(n = n()) %>%
  ggplot(aes(n)) + geom_histogram(binwidth=0.1, color = "black") +
  scale_x_log10() +
  xlab("Number of ratings") +
  ylab("Number of movies") +
  ggtitle("Ratings by Movie")
```



Creating a model

The goal is create a model which RMSE <= 0.8649. The first step is define a function that computes RMSE

```
# Creating the function RMSE that computes the RMSE
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Model 1: Average rating

Predicting the same rating for all users and movies. A model that apply the same rating for all users and movies with all the differences explained by random variation would be like this:

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

with:

- μ the rating for all movies
- $\epsilon_{u,i}$ independent errors

We take $\hat{\mu}$ = average ratings as an estimator of μ
Let's compute RMSE

```
# calculating predicted ratings
mu_hat <- mean(edx$rating)

#RMSE of model 1
model_1_rmse <- RMSE(validation$rating, mu_hat)
model_1_rmse
```

```
## [1] 1.061202
```

```
# Storing RMSE
rmse_results <- data_frame(method = "Model 1: Average rating", RMSE = model_1_rmse)
```

Model 2: Movie Effect

We know that there is a movie effect since different movies get different ratings, this effect is called movie bias (b_i), we add this effect to the previous model:

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

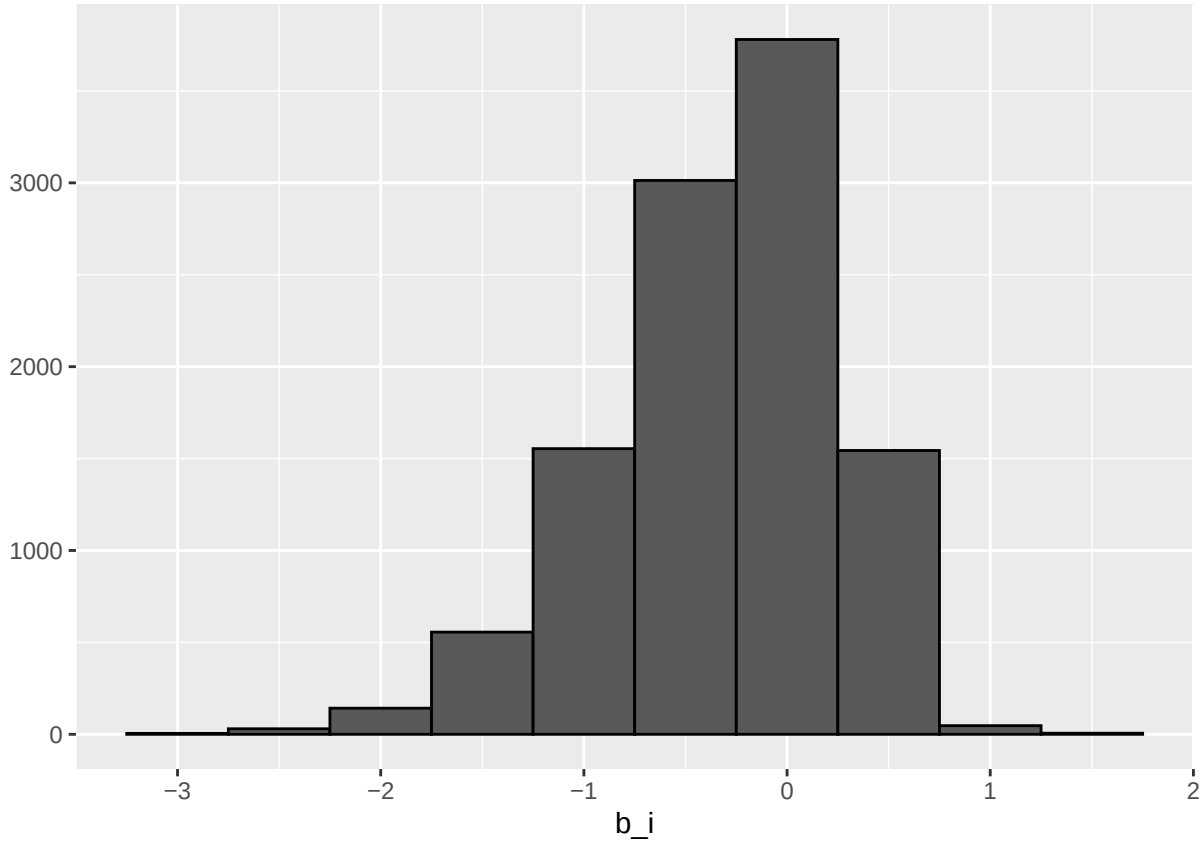
- b_i the average rating for movie i

We use the least square estimate as estimator of b_i

```
# movie_avg: estimator for b_i
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))
```

We see that the histogram of b_i is skewed to the left, that means that there are more movies with negative b_i than positive

```
# b_i histogram
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"))
```



Let's compute RMSE

```r
# calculating predicted ratings
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(pred = mu_hat + b_i) %>%
  pull(pred)

#RMSE of model 2
model_2_rmse <- RMSE(predicted_ratings, validation$rating)
model_2_rmse
```

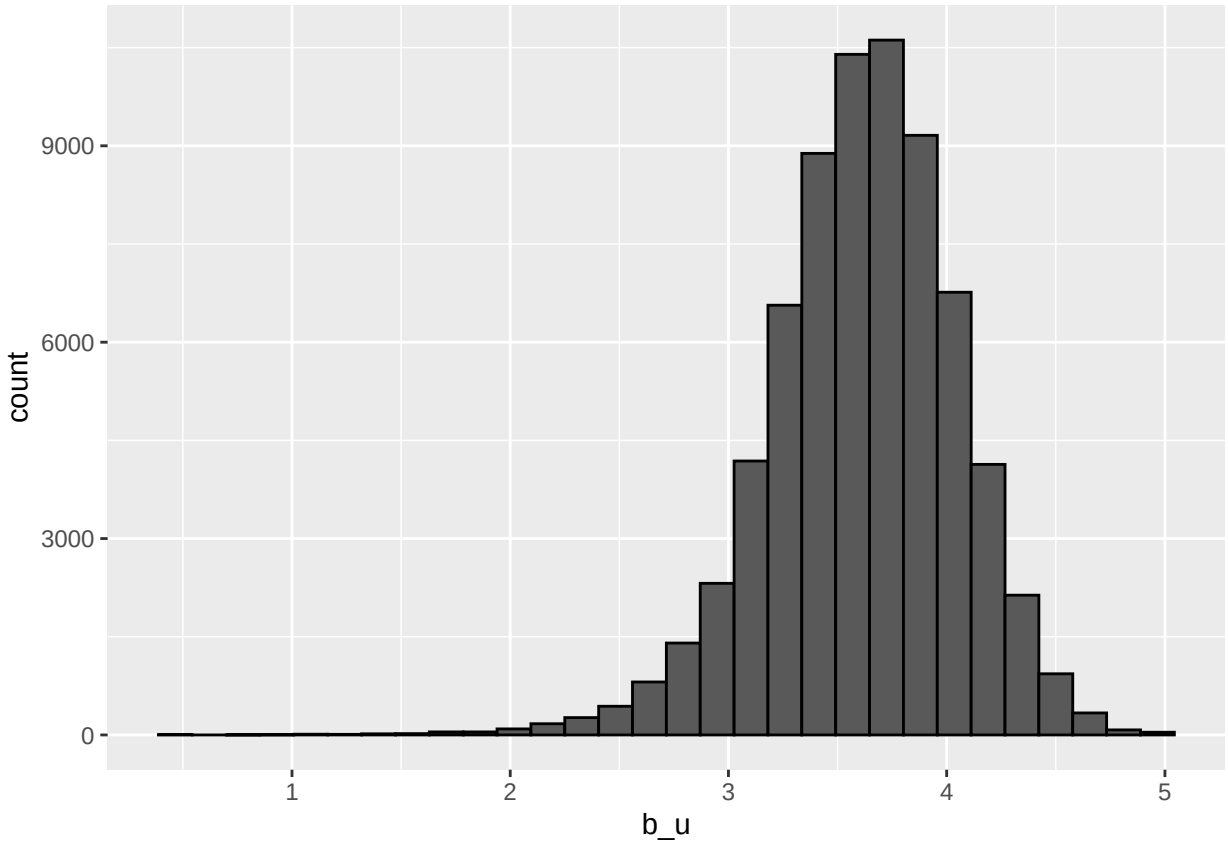
```
## [1] 0.9439087
```

```r
# Storing RMSE
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Model 2: Movie Effect",
                                     RMSE = model_2_rmse))
```

Model 3: Movie & User Effect

Let's see how users rate movies by computing the average rating per user

```r
# Average rating for user u
edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```



We can see that there is user effect due to the variability, we can add the user effect to the model 2:

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

- b_u the average rating for user u, $\hat{b}_u = Y_{u,i} - \hat{\mu} - \hat{b}_i$

```
# user_avg: estimator for b_u
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))

# Calculating predicted ratings
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  pull(pred)

# RMSE of model 3
model_3_rmse <- RMSE(predicted_ratings, validation$rating)
model_3_rmse
```

```
## [1] 0.8653488
```

```
# Storing RMSE
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Model 3: Movie & User Effect",
                                     RMSE = model_3_rmse))
```

Regularization, Penalized Least Squares

RMSE is high sensitive to large errors. Regularization permits us to penalize large estimates that are formed using small sample sizes.

Model 4: Movie & User Effect with Regularization

The general idea of penalized regression is to control the total variability of the movie and user effects. Specifically, instead of minimizing the least square equation, we minimize an equation that adds a penalty:
$\frac{1}{N}\sum_{u,i}(y_{u,i} - \mu - b_i - b_u)^2 + \lambda\Big(\sum_i b_i^2 + \sum_u b_u^2\Big)$

The first term is just least squares and the second is a penalty that gets larger when many b_i and b_u are large.

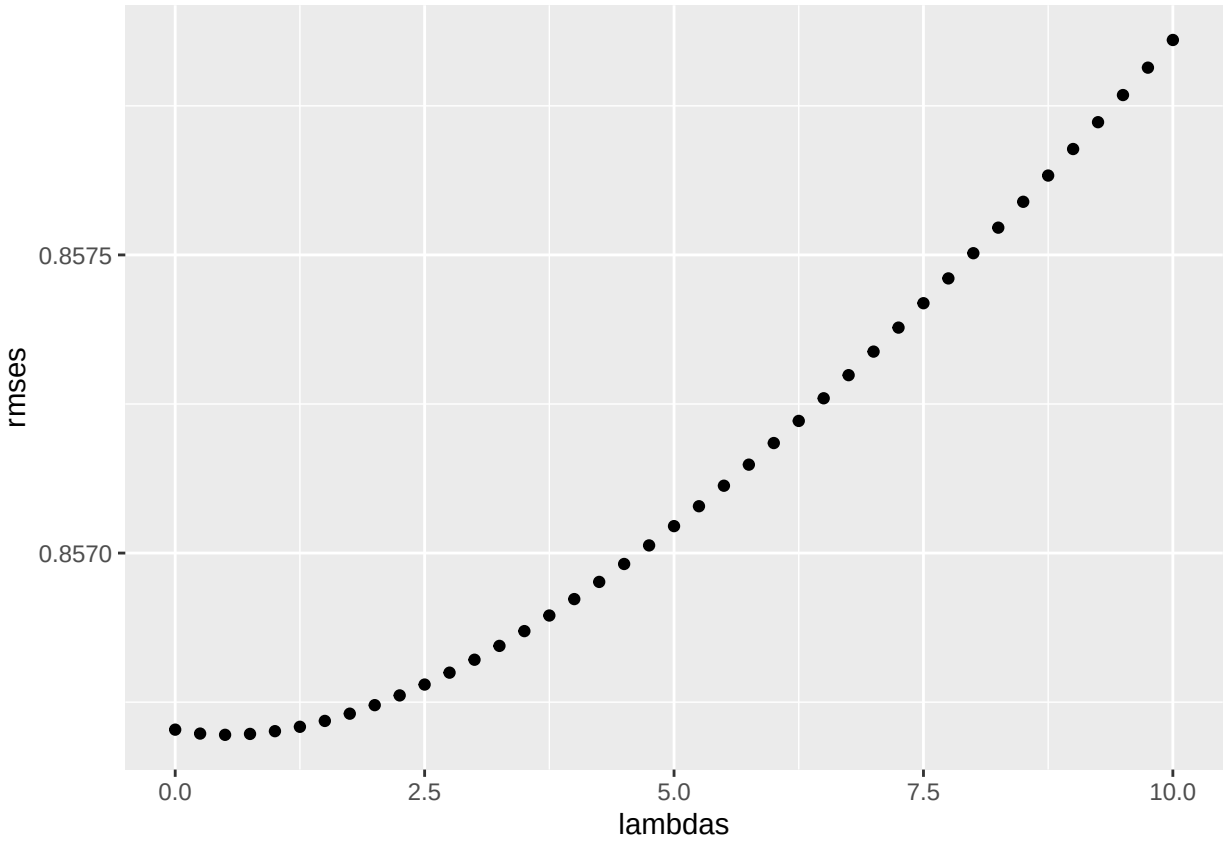
Where:

- $\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i}\sum_{n_i}\Big(y_{u,i} - \hat{\mu}\Big)$, $i \in \{movies\}$
- $\hat{b}_u(\lambda) = \frac{1}{\lambda + n_u}\sum_{n_u}\Big(y_{u,i} - \hat{b}_i(\lambda) - \hat{\mu}\Big)$, $u \in \{users\}$

As λ is a parameter, we will use cross-validation to choose the best lambda on the test set, remember that we can't use validation set for tuning parameters

```
# We use cross-validaton to choose the parameter lambda
lambdas <- seq(0, 10, 0.25)
mu_hat <- mean(edx$rating)
```

```r
#for each lambda of lambdas we get the RMSE
rmses <- sapply(lambdas, function(l){

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

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

# Predicted ratings (lambda) on test set
  predicted_ratings <- edx %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu_hat + b_i + b_u) %>%
    pull(pred)

# RMSE (lambda)
  return(RMSE(predicted_ratings, edx$rating))
})

# Choosing lambda which minimize RMSES
qplot(lambdas, rmses)
```



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

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

```
rmses[which.min(rmses)]
```

```
## [1] 0.8566952
```

Now, we apply the λ which minimize RMSES on validation set

```
# Applying lambda on validation set
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_hat)/(n()+lambda))

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

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

# RMSE of model 4
model_4_rmse <- RMSE(predicted_ratings, validation$rating)
model_4_rmse
```

```
## [1] 0.8652226
```

```
# Storing RMSE
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Model 4: Movie + User Effect + Reg",
                                     RMSE = model_4_rmse))
```

Model 5: Movie, User Effect, Year Release and Genres with Regularization

Let's see if by adding the release year of the movie and the the movie genres we improve the model In this case we are minimizing:

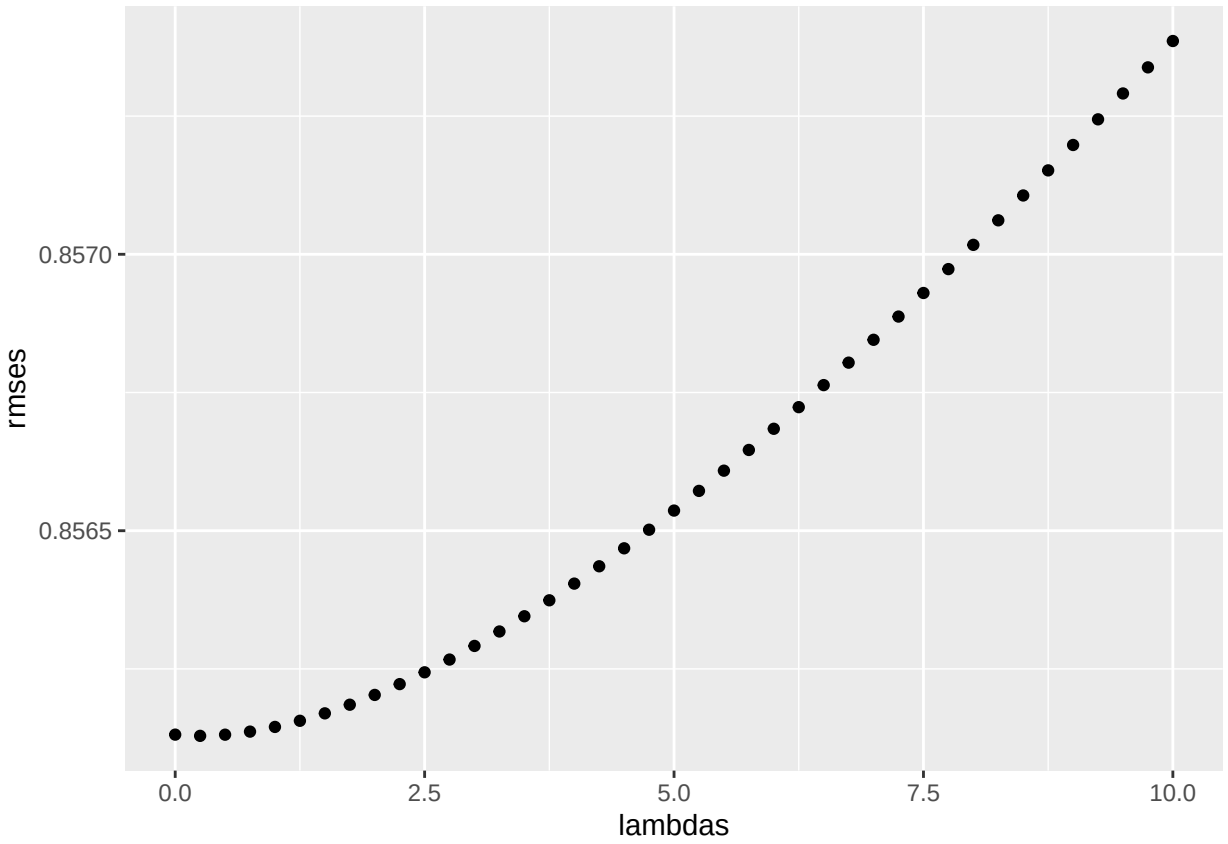
$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i-b_u)^2+\lambda\Big(\sum_i b_i^2+\sum_u b_u^2+\sum_j b_j^2+\sum_k b_k^2\Big)$$

Where:

- $\hat{b}_i(\lambda)=\frac{1}{\lambda+n_i}\sum_{n_i}\Big(y_{u,i}-\hat{\mu}\Big),\ i\in\{movies\}$
- $\hat{b}_u(\lambda)=\frac{1}{\lambda+n_u}\sum_{n_u}\Big(y_{u,i}-\hat{b}_i(\lambda)-\hat{\mu}\Big),\ u\in\{users\}$
- $\hat{b}_j(\lambda)=\frac{1}{\lambda+n_u}\sum_{n_u}\Big(y_{u,i}-\hat{b}_i(\lambda)-\hat{b}_u(\lambda)-\hat{\mu}\Big),\ j\in\{releaseYears\}$
- $\hat{b}_k(\lambda)=\frac{1}{\lambda+n_u}\sum_{n_u}\Big(y_{u,i}-\hat{b}_i(\lambda)\hat{b}_u(\lambda)-\hat{b}_k(\lambda)-\hat{\mu}\Big),\ k\in\{genres\}$

Cross-validation to pick a λ

```r
lambdas <- seq(0, 10, 0.25)
mu_hat <- mean(edx$rating)

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

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

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

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

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

 predicted_ratings <- edx %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_j, by = "yearRelease") %>%
    left_join(b_k, by = "genres") %>%
    mutate(pred = mu_hat + b_i + b_u + b_j + b_k) %>%
    pull(pred)

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

# Choosing lambda which minimize RMSES
qplot(lambdas, rmses)
```




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

```
## [1] 0.8561289
```

Applying the λ on validation set

```
# Applying lambda on validation set
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_hat)/(n()+lambda))

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

b_j <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(yearRelease) %>%
  summarize(b_j = sum(rating - b_i - b_u - mu_hat)/(n()+lambda))

b_k <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_j, by="yearRelease") %>%
  group_by(genres) %>%
```

```
  summarize(b_k = sum(rating - b_i - b_u - b_j - mu_hat)/(n()+lambda))

predicted_ratings <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_j, by = "yearRelease") %>%
  left_join(b_k, by = "genres") %>%
  mutate(pred = mu_hat + b_i + b_u + b_j + b_k) %>%
  pull(pred)

# RMSE of model 5
model_5_rmse <- RMSE(predicted_ratings, validation$rating)
model_5_rmse
```

```
## [1] 0.8646505
```

```
# Storing RMSE
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Model 5: Movie + User Effect + year release + Reg",
                                     RMSE = model_5_rmse))
```

3. Results

Below we can see RMSE values of the five models, we have achieved the best result with four features (movie, user, year release and genres) using regularization.

method	RMSE
Model 1: Average rating	1.0612018
Model 2: Movie Effect	0.9439087
Model 3: Movie & User Effect	0.8653488
Model 4: Movie + User Effect + Reg	0.8652226
Model 5: Movie + User Effect + year release + Reg	0.8646505

4. Conclusion

We have train a machine learning algorithm that can predict the movie rating of MovieLens dataset (10M version) with RMSE = 0.8646505. The RMSE table shows the improvement of each model. The difference between the first model and the last is 18.52%. The goal has been achieved as we managed to get RMSE less to 0.8649 (The given goal).