ML_project_markdown

Paul Gis

11/21/2020

1. Introduction

MovieLens dataset is presented in the capstone course as a part of the certification program for data science. This dataset consists of a list of ratings of movies given by users. The main objective of the excersise is to create our own movie recomendation system.

In this document, we will present the main steps followed to achieve the primary endpoint.

1.1 Dataset description and model

The dataset is composed of a list of users (UserId numbers), movies (movieId and movie title with year), ratings, genres and a timestamp column of when the ratings were made. The original one is to large, so, for this project, a 10M version will be used for making easier the computation.

The Root Mean Squared Error (RMSE) will be used to evaluate how close our predictions are to the true values in the validation set. the goal is to obtain a RMSE < 0.8649

we used a linear model based on the following formula to add “effects” to the simplest model of averaging the ratings:

$$y = \mu + b_i + b_u + b_g + b_y + \epsilon_{i,u,g,y}$$

1.2 Workflow

After reviewing the dataset, there was found that some features could be split up in order to obtain new features that could bring valuable information to the model; such as genre and year, where all genres were contained within one single variable and gathering up to 8 for several movies and only the first one could be useful. regarding the year, is was included in the title of the movie, which could also be separated and used as another feature for modeling.

As mentioned before, based in the simplest model of averaging the ratings, we build and add up effects to increase the accuracy of the model. Then this model was regularized in order to control extreme values that could affect the results and we picked the lamda wich presented the lowest RMSE.

Finally, the completed and regularized model was used in the validation set.

1.3 results and counclusions

This MovieLens project approach based in a linear model was optimal to achieve the mail goal of RMSE < 0.8649. Starting by the simplest model of averaging ratings and then adding different effects to improve the prediction, we could move from 1.060054 to 0.8642193 in the training dataset. These results were also proved to be consistent in the final validation set, were a final 0.8644344 was achieved. This is an importat result considering that there were biased effects added to the model, but the regularization step brought a key step to minimize the error.

In general, our model showed to be optimal for predicting accurate movie ratings, however, a more in depth analysis could be performed in order to look for additional effects to improve the model, such as other genres; or maybe other use model approaches, such as MSE or Matrix factorization, however the RMSE was the requested method. In summary, for the primary goal of the excercise, this proposed model was adequate.

2. Methods and Analysis

We will need to split the main dataset in two subsets: one for trainning, and another for validation purpuses, and then train a machine learning algorithm using the inputs in the training subset to predict movie ratings in the validation set. This process of creating the **edx** (Train) and **validation** subsets is given directly in the course, so this step is directly arranged by default in the course contents according the following code:

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-proje
ct.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-pro
ject.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("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(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)`
```

```
## 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 = FA
LSE)
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)
```

2.1 Comprehensive analysis of the database

Let´s move straight forward to review what is in the edx set:

```
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 838
983653 838984885 838983707 838984596 ...
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargat
e (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|T
hriller" "Action|Adventure|Sci-Fi" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

It shows 900005 obs and 6 variables. the first thing I notice is that R reads some variables as numeric (userId, movieId) when they´re not, and also the timestamp variable can also be arrange to date; so we need to change that first:

```
edx$userId <- as.character(edx$userId)
edx$movieId <- as.character(edx$movieId)
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-proje
ct.org")
edx$timestamp <- as_datetime(edx$timestamp, origin = "1970-01-01")
```

Now, we can make some arrangements here with some variables which we an break down in some other variables, for instance, year is in the title and there are several genres joined in most of the observations. so let ´s make some feature engineering.

We will start with the genres column. let´s see how it is:

```
tibble(count = str_count(edx$genres, pattern = "|"), genres = edx$genres) %>%
  group_by(count, genres) %>%
  summarise(n = n()) %>%
  arrange(-count) %>%
  head()
```

```
## # A tibble: 6 x 3
## # Groups:   count [4]
##   count genres                                                      n
##   <int> <chr>                                                   <int>
## 1    61 Action|Adventure|Comedy|Drama|Fantasy|Horror|Sci-Fi|Thriller   256
## 2    60 Adventure|Animation|Children|Comedy|Fantasy|Musical|Romance    515
## 3    58 Adventure|Animation|Children|Comedy|Crime|Fantasy|Mystery    10975
## 4    58 Adventure|Animation|Children|Comedy|Drama|Fantasy|Mystery      355
## 5    52 Adventure|Animation|Children|Comedy|Fantasy|Musical            836
## 6    52 Adventure|Animation|Children|Comedy|Fantasy|Romance          13063
```

It shows that a great deal of movies are classiffied with several genres, and 256 up to 8 differrent genres, these clearly could be split out. We can keep the first genre as it is the most important one and not all movies have been cathegorized with two or more genres

```
edx <- edx %>% separate(genres, c("genre", "genre2"), sep = "[|]", fill = "warn") %>%
select(-genre2)
```

```
## Warning: Expected 2 pieces. Additional pieces discarded in 4621711 rows [2, 3,
## 4, 5, 6, 7, 8, 9, 11, 12, 13, 15, 16, 18, 20, 21, 23, 24, 25, 27, ...].
```

```
## Warning: Expected 2 pieces. Missing pieces filled with `NA` in 1740569 rows [14,
## 26, 39, 41, 43, 50, 58, 62, 76, 83, 89, 111, 114, 120, 121, 123, 124, 125, 126,
## 127, ...].
```

```
edx$genre <- as.factor(edx$genre)
```

Now, we can take a quick view and see some tendences:

```
edx %>% group_by(genre) %>%
  summarize(count = n()) %>%
  top_n(10) %>%
  arrange(desc(count))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## Selecting by count
```

```
## # A tibble: 10 x 2
##    genre         count
##    <fct>         <int>
##  1 Action      2560545
##  2 Comedy      2437260
##  3 Drama       1741668
##  4 Adventure    753650
##  5 Crime        529521
##  6 Horror       233074
##  7 Animation    218123
##  8 Children     181217
##  9 Thriller      94718
## 10 Documentary   80966
```

Action appears to be the most rated genre, folowed by Comedy and Drama. Moving on, we arrange the title column and separate the year into a different column:

```
edx <- edx %>% mutate(year = as.numeric(str_sub(title,-5,-2)))
```

Now we can take a closer look to see how the year performs within the data

```
edx %>% group_by(year) %>%
  summarize(count =n()) %>%
  arrange(desc(count)) %>%
  select(year, count)
```

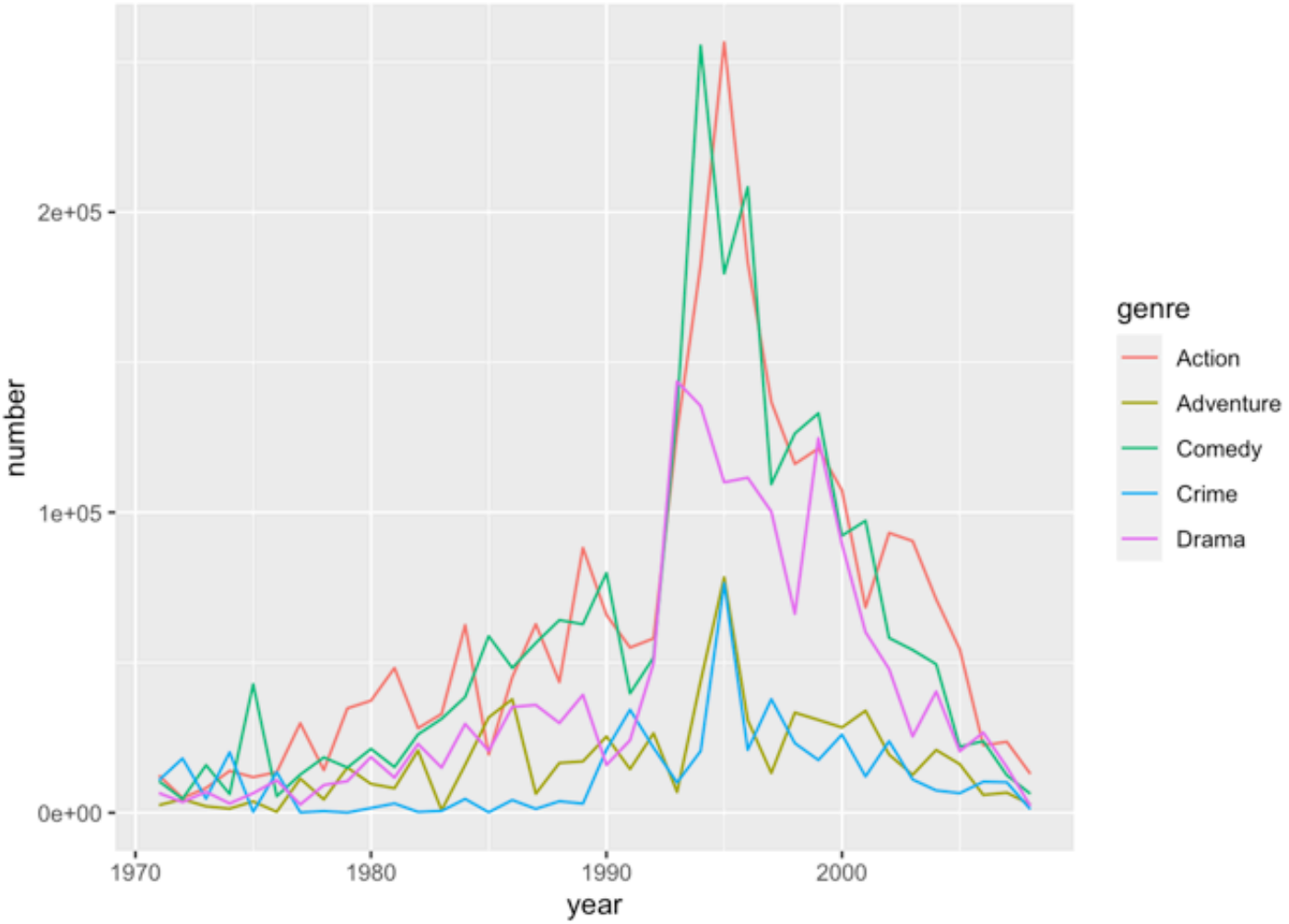
```
## # A tibble: 94 x 2
##     year  count
##    <dbl>  <int>
##  1  1995 786762
##  2  1994 671376
##  3  1996 593518
##  4  1999 489537
##  5  1993 481184
##  6  1997 429751
##  7  1998 402187
##  8  2000 382763
##  9  2001 305705
## 10  2002 272180
## # … with 84 more rows
```

We can also see there are years with most review of others, starting with 1995 and followed by other 7 positions of years of the 90 decade. This tells us that clearly there is a trend for rating and this could depend on other variables, such blockbusters, their genres and when they were launched.

we can see this association in the following graph, taking as example the top 5 genres rated:

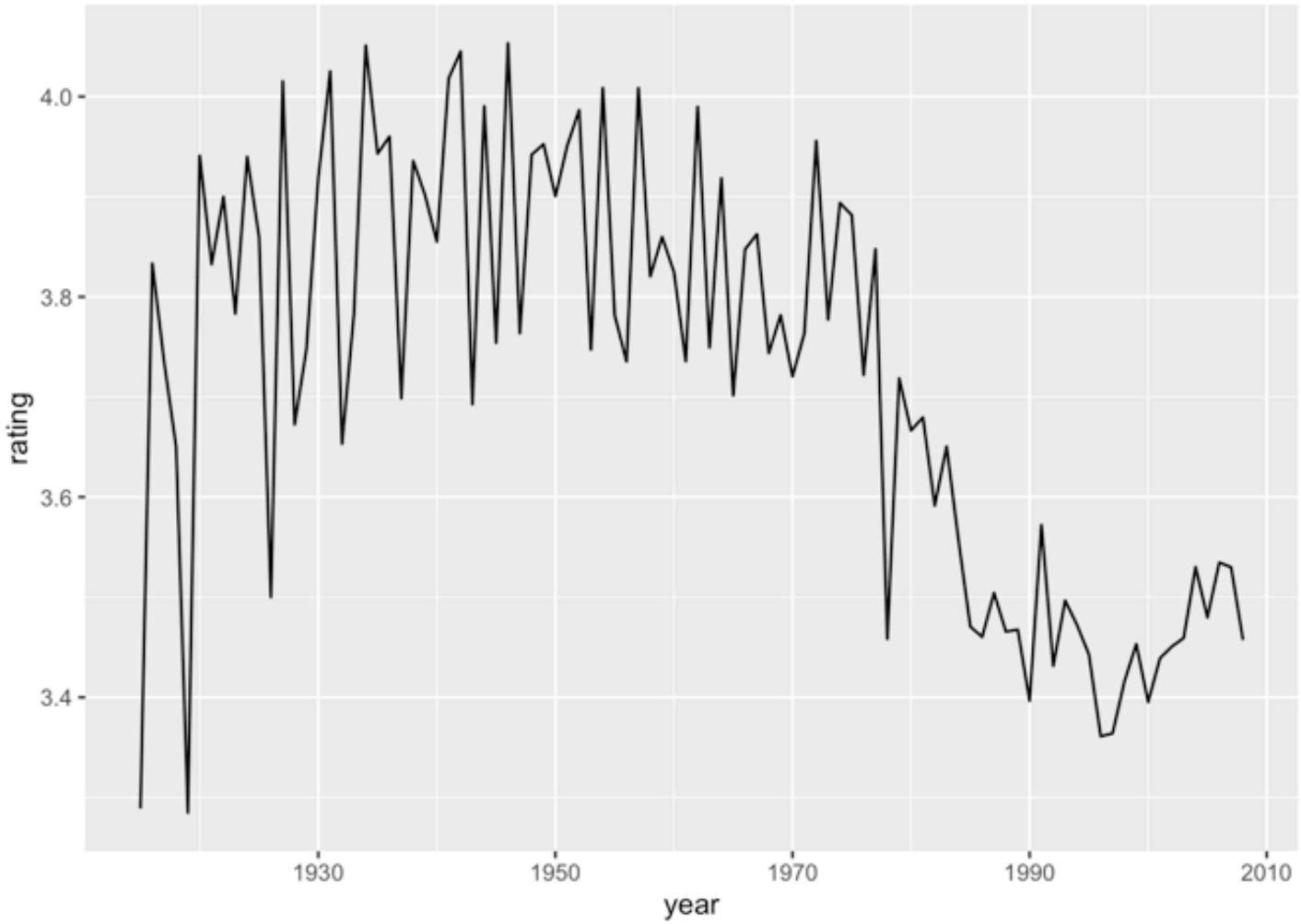
```
genre_year <- edx %>%
  select(movieId, genre, year) %>%
  group_by(year, genre) %>%
  summarise(number = n())
```

```
genre_year %>%
  filter(year > 1970) %>%
  filter(genre %in% c("Action", "Comedy", "Drama", "Adventure", "Crime")) %>%
  ggplot(aes(x = year, y = number)) +
  geom_line(aes(color=genre))
```



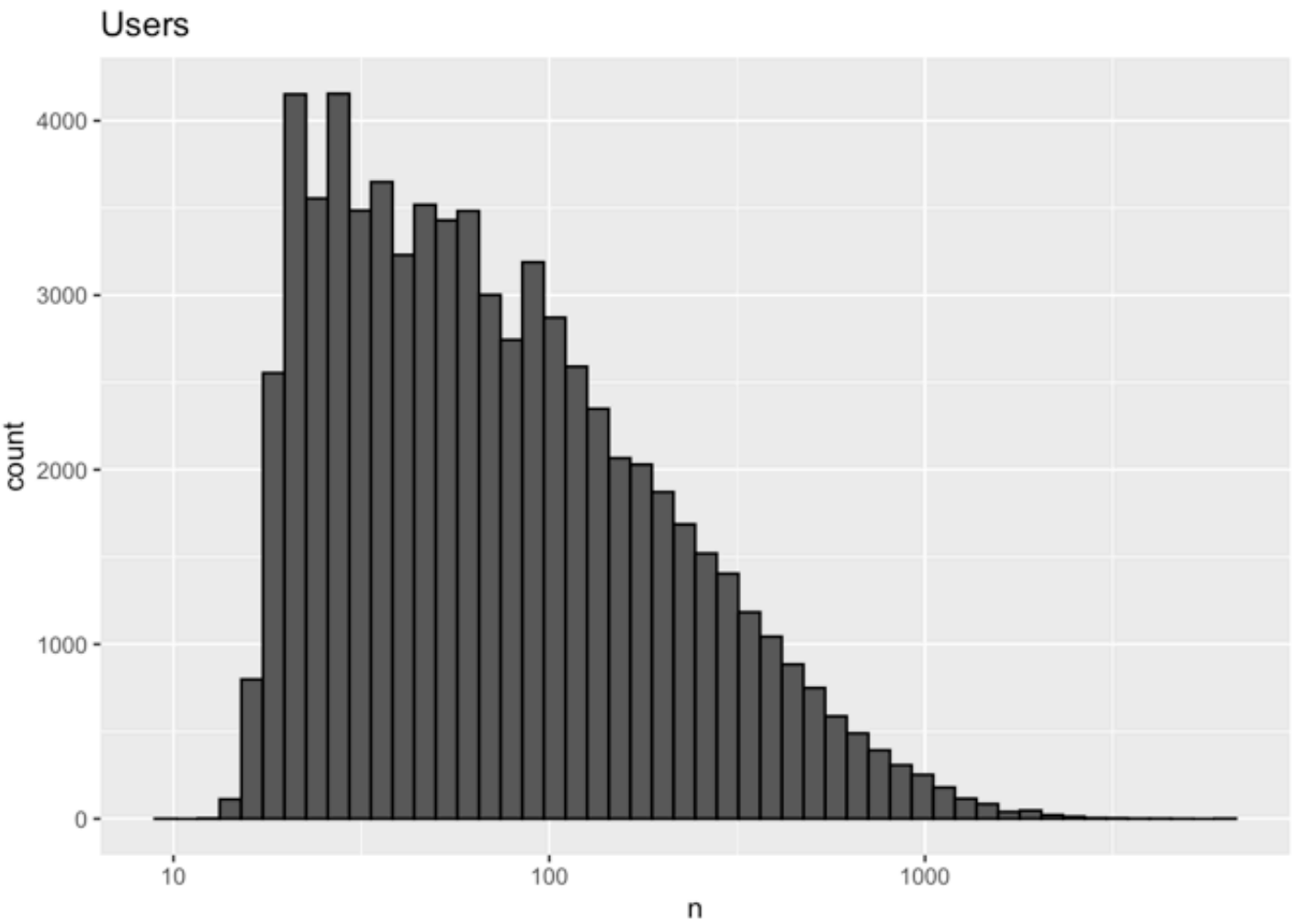
It shows as expected, the most rated genres are concentrated mostly in the 90`s, and we can explore how this trend is correlated with the ratings given by the users:

```
edx %>% group_by(year) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(year, rating)) +
  geom_line()
```



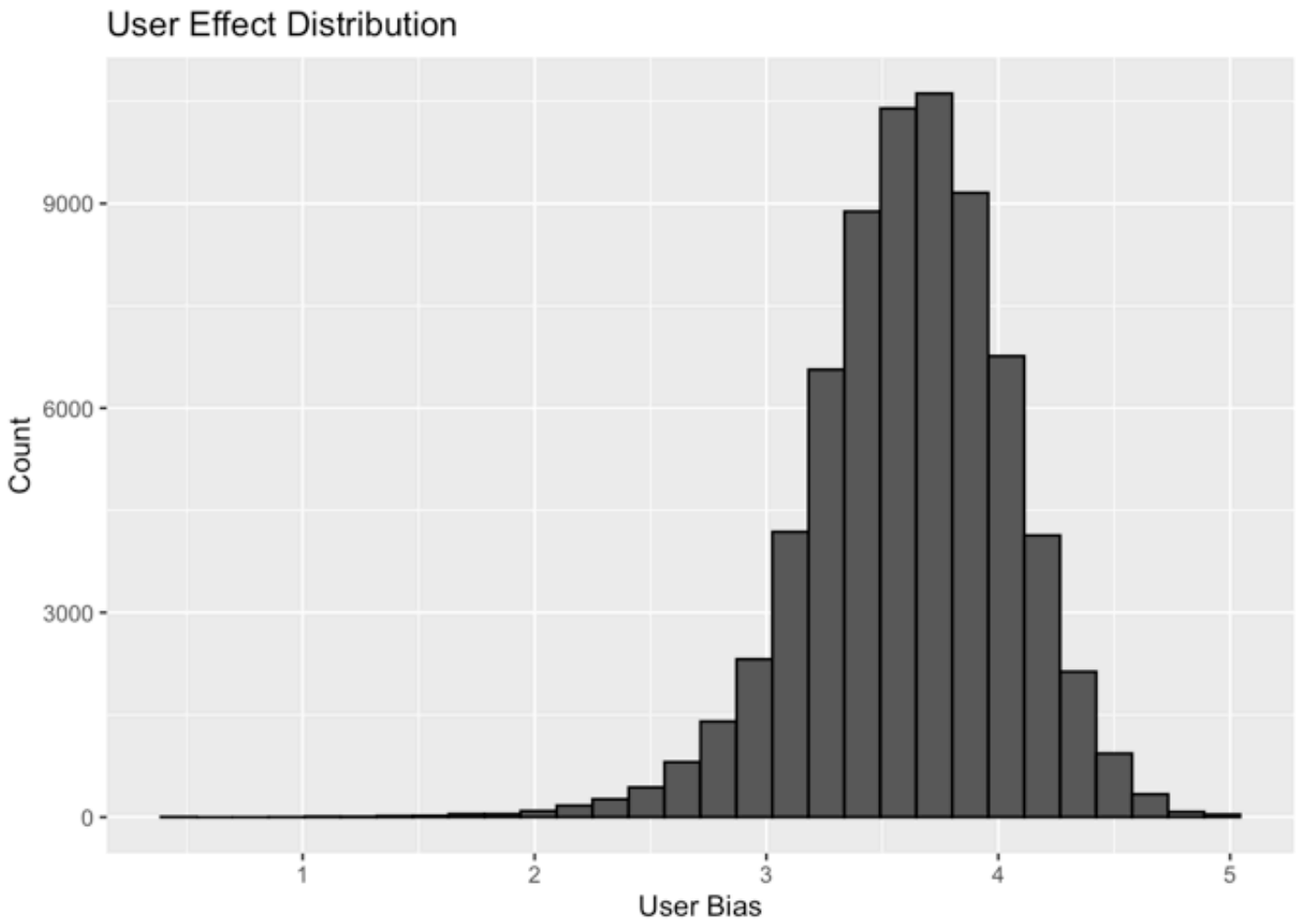
Interestly, we see that users who have more classic-taste on movies are less and trend to rate higher those movies, in contrast with a broader set of modern-taste users, who trend to give lower average ratings. This is more clear to see in the folowing graph:

```
edx %>% count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 50, color = "black") +
  scale_x_log10() +
  ggtitle("Users")
```



As it shows, the distribution is skewed and could affect our prediction; so we need to consider that all user are different (user effect) and some of them give few ratings, but in general, the ratings distribution is close to normal:

```
edx %>% group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(color = "black") +
  ggtitle("User Effect Distribution") +
  xlab("User Bias") +
  ylab("Count")
```



So in general, we see that there is 4 major effects in the data set: user, genre year and of course, the movie. All this features will be considered to build our model and explained in the model section.

2.2 Model

As it was described in the Machine Learning course, let´s create the Root Mean Squared Error (RMSE) function:

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

based in the analysis made before, we will use a linear model based on the formula

$$y = \mu + b_i + b_u + b_g + b_y + \epsilon_{i,u,g,y}$$

where starting form the simplest model: the mean μ of the ratings, which considers that all users give the same rating to every movie. This of course is not optimal, but according the statistical theory presented in the course, the average minimizes the RMSE, so any model should be based on that.

Creating test and training sets from edx

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

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

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

Make sure userId and movieId in test set are also in train set

```
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId") %>%
  semi_join(train_set, by = "genre") %>%
  semi_join(train_set, by = "year")
```

Add rows removed from test set back into train set

```
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)

rm(test_index, temp, removed)
```

build the simplest model: averaging the ratings

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

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

```
RMSE(test_set$rating, mu)
```

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

So 1.060054 will be our starting point, and we will need to move down to a number bellow 0.8649 to reach our goal.

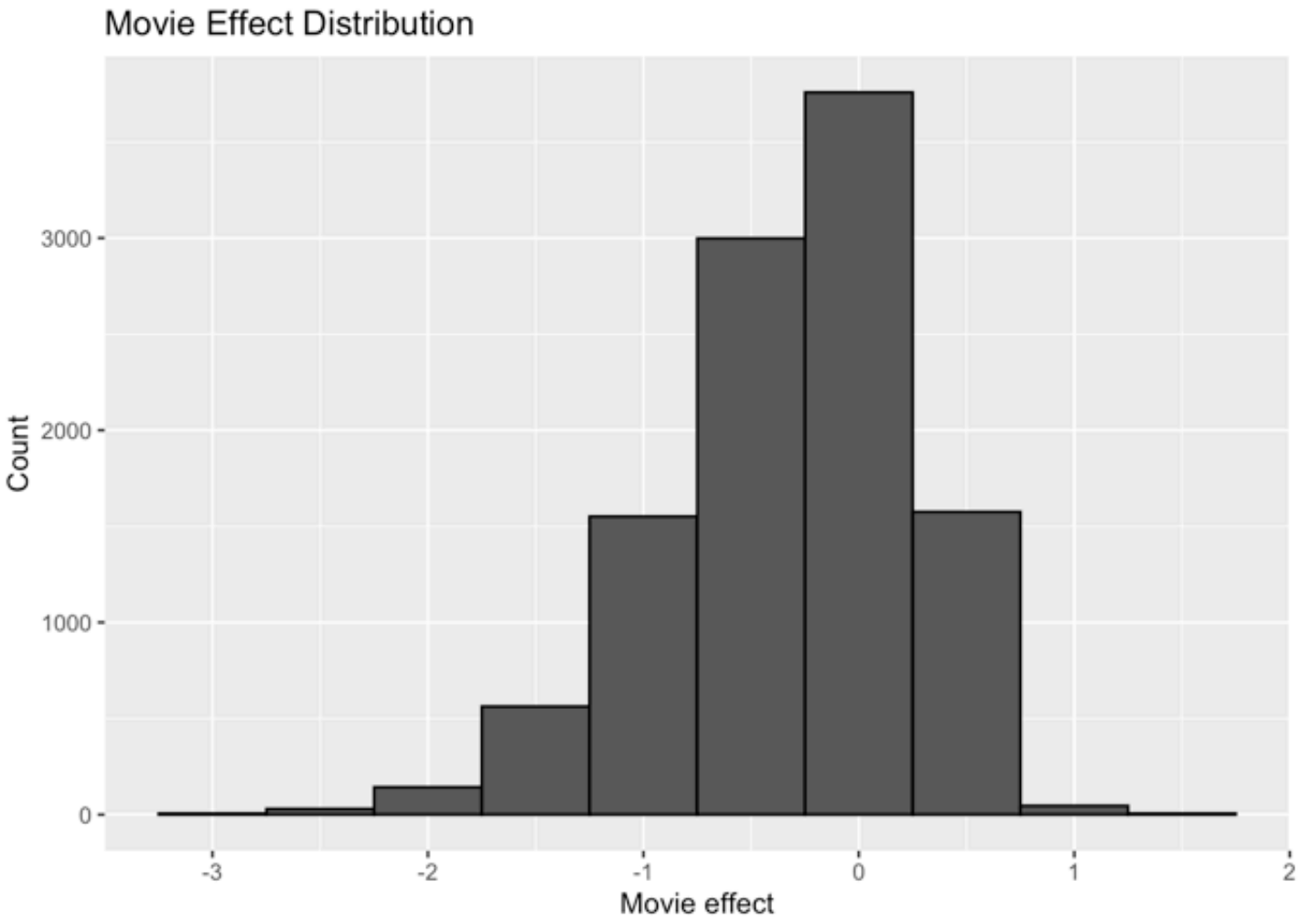
2.2.1 Movie effect (bi)

As mentioned before, we found that several features could be considered for prediction, and the movie effect is one of them. This can be explained considering that different movies have different ratings distribution as, of course, some are blockbusters and there are different user preferences. This movie “bias” or “movie effect” is expressed as b_i in the following formula (the mean of the difference between the observed rating y and the mean μ):

```
bi <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

And we can see it in the following graph

```
bi %>% ggplot(aes(x = b_i)) +
  geom_histogram(bins=10, col = I("black")) +
  ggtitle("Movie Effect Distribution") +
  xlab("Movie effect") +
  ylab("Count")
```



Now let´s see how can we predict the rating with mean + the movie effect b_i

```
y_hat_bi <- mu + test_set %>%
  left_join(bi, by = "movieId") %>%
  .$b_i
RMSE(test_set$rating, y_hat_bi)
```

```
## [1] 0.9429615
```

0.9429615 is a huge jump from our starting model with the average.

2.2.2 User effect (bu)

In the same way we did with the movie effect, we can now build a the user or user effect:

```
bu <- train_set %>%
  left_join(bi, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

Let´s see our progress predicting the rating with mean + the movie effect b_i + the user effect b_u

```
y_hat_bi_bu <- test_set %>%
  left_join(bi, by='movieId') %>%
  left_join(bu, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
RMSE(test_set$rating, y_hat_bi_bu)
```

```
## [1] 0.8646843
```

0.8646843 is beyond the cut of 0.8649 requested as goal for the excersise, however, we can move on to see how far we can get with the other variables considered in our model.

2.2.3 Gender Effect (bg)

```
bg <- train_set %>%
  left_join(bi, by ='movieId') %>%
  left_join(bu, by='userId') %>%
  group_by(genre) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))
```

and the prediction:

```
y_hat_bi_bu_bg <- test_set %>%
  left_join(bi, by='movieId') %>%
  left_join(bu, by='userId') %>%
  left_join(bg, by= "genre") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  .$pred
RMSE(test_set$rating, y_hat_bi_bu_bg)
```

```
## [1] 0.8645672
```

0.8645672 is not a big change, but it keeps helping us.

2.2.4 Year Effect (by)

And finally the year effect:

```
by <- train_set %>%
  left_join(bi, by ='movieId') %>%
  left_join(bu, by='userId') %>%
  left_join(bg, by= "genre") %>%
  group_by(year) %>%
  summarize(b_y = mean(rating - mu - b_i - b_u - b_g))
```

Final prediction with rating with all the effects: mean + bi + bu + bg + by

```
y_hat_bi_bu_bg_by <- test_set %>%
  left_join(bi, by='movieId') %>%
  left_join(bu, by='userId') %>%
  left_join(bg, by= "genre") %>%
  left_join(by, by= "year") %>%
  mutate(pred = mu + b_i + b_u + b_g + b_y) %>%
  .$pred
```

results of the model in the test set

```
results_test_set <- RMSE(test_set$rating, y_hat_bi_bu_bg_by)
results_test_set
```

```
## [1] 0.8642193
```

0.8642193 is our last result using all the effects considered for the model.

2.3 Regularization

As it was shown in the couse contents and considering that all this effects are biased, we need now to optimize the model avoiding these bias to affect the prediction by penalizing extreme values. We need then to regularize the model and pick the lamda which gives the lowest RMSE:É

```
regularization <- function(lambda, trainset, testset){
# Mean
mu <- mean(trainset$rating)
  # Movie effect (bi)
  b_i <- trainset %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))
  # User effect (bu)
  b_u <- trainset %>%
    left_join(b_i, by="movieId") %>%
    filter(!is.na(b_i)) %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n()+lambda))
  # Gender Effect (bg)
  b_g <- trainset %>%
    left_join(bi, by ='movieId') %>%
    left_join(bu, by='userId') %>%
    group_by(genre) %>%
    summarize(b_g = mean(rating - mu - b_i - b_u)/(n()+lambda))
  # Year Effect (by)
  b_y <- trainset %>%
    left_join(bi, by ='movieId') %>%
    left_join(bu, by='userId') %>%
    left_join(bg, by= "genre") %>%
    group_by(year) %>%
    summarize(b_y = mean(rating - mu - b_i - b_u - b_g)/(n()+lambda))
  # Prediction: mu + bi + bu + bg + by
  predicted_ratings <- testset %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genre") %>%
    left_join(b_y, by = "year") %>%
    filter(!is.na(b_i), !is.na(b_u), !is.na(b_g), !is.na(b_y)) %>%
    mutate(pred = mu + b_i + b_u + b_g + b_y) %>%
    pull(pred)

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

Defining a set of lambdas to tune

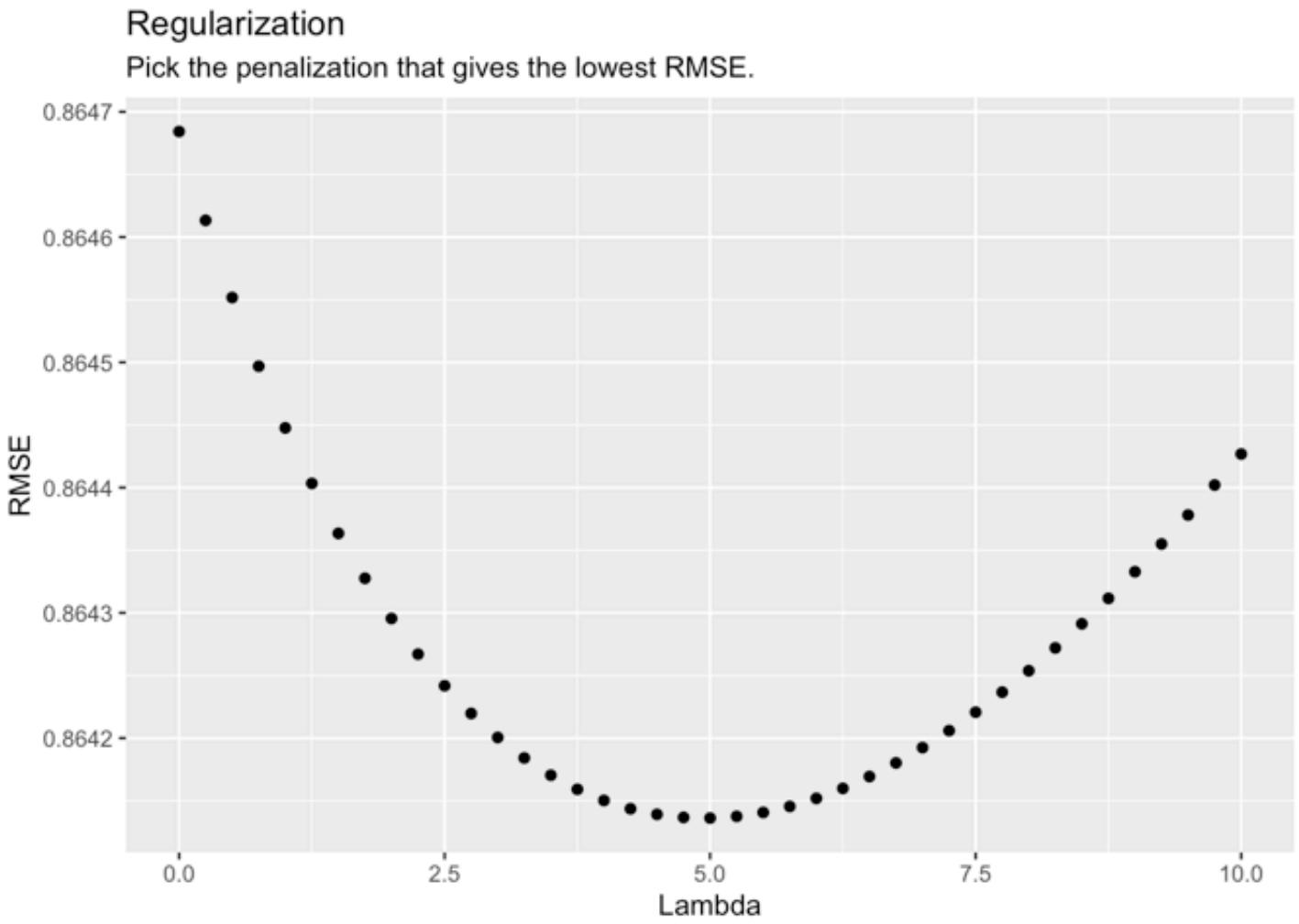
```
lambdas <- seq(0, 10, 0.25)
```

Tune lambda

```
rmses <- sapply(lambdas,
                regularization,
                trainset = train_set,
                testset = test_set)
```

Plotting the lambda vs RMSE

```
tibble(Lambda = lambdas, RMSE = rmses) %>%
  ggplot(aes(x = Lambda, y = RMSE)) +
  geom_point() +
  ggtitle("Regularization",
          subtitle = "Pick the penalization that gives the lowest RMSE.")
```



So 5.0 is the lamda value where we minimize the most the RMSE.

3. Results and Final Validation

Finally the model is all set. We can now prove it in the validation test to see how it perfoms. As we did in the trainning set, we need to start by adding effects to the simple model of averaging the ratings:

```
lambda <- 5.0
mu_edx <- mean(edx$rating)
```

Movie effect (bi)

```
b_i_edx <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_edx)/(n()+lambda))
```

User effect (bu)

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

Genre effect (bg)

```
b_g_edx <- edx %>%
  left_join(b_i_edx, by ="movieId") %>%
  left_join(b_u_edx, by ="userId") %>%
  group_by(genre) %>%
  summarize(b_g = sum(rating - b_i - b_u - mu_edx)/(n()+lambda))
```

Year effect (by)

```
b_y_edx <- edx %>%
  left_join(b_i_edx, by ="movieId") %>%
  left_join(b_u_edx, by ="userId") %>%
  left_join(b_g_edx, by = "genre") %>%
  group_by(year) %>%
  summarize(b_y = sum(rating - b_i - b_u - b_g - mu_edx)/(n()+lambda))
```

validation set arrangements

```
validation$userId <- as.character(validation$userId)
validation$movieId <- as.character(validation$movieId)
validation$timestamp <- as_datetime(validation$timestamp, origin = "1970-01-01")
validation <- validation %>% separate(genres, c("genre", "genre2"), sep = "[|]", fill
= "warn") %>% select(-genre2)
```

```
## Warning: Expected 2 pieces. Additional pieces discarded in 512996 rows [2, 4, 6,
## 7, 8, 9, 10, 11, 14, 18, 19, 20, 21, 22, 23, 25, 27, 31, 32, 34, ...].
```

```
## Warning: Expected 2 pieces. Missing pieces filled with `NA` in 193630 rows [1,
## 16, 24, 36, 50, 60, 67, 70, 81, 87, 92, 104, 108, 113, 118, 131, 136, 142, 145,
## 149, ...].
```

```
validation$genre <- as.factor(validation$genre)
validation <- validation %>% mutate(year = as.numeric(str_sub(title,-5,-2)))
```

Final Prediction

```
y_hat_edx <- validation %>%
  left_join(b_i_edx, by = "movieId") %>%
  left_join(b_u_edx, by = "userId") %>%
  left_join(b_g_edx, by = "genre") %>%
  left_join(b_y_edx, by = "year") %>%
  mutate(pred = mu_edx + b_i + b_u + b_g + b_y) %>%
  pull(pred)
```

Final result in the validation test

```
Final_result <-  RMSE(validation$rating, y_hat_edx)
Final_result
```

```
## [1] 0.8644344
```

0.8644344 is a good result for a regularized model which included several biased effects, and is not far from the 0.8642193 obtained in our training model.

4. Conclusion

This MovieLens project approach based in a linear model was optimal to achieve the mail goal of RMSE < 0.8649. Starting by the simplest model of averaging ratings and then adding different effects to improve the prediction, we could move from 1.060054 to 0.8642193 in the training dataset. These results were also proved to be consistent in the final validation set, were a final 0.8644344 was achieved. This is an importat result considering that there were biased effects added to the model, but the regularization step brought a key step to minimize the error.

In general, our model showed to be optimal for predicting accurate movie ratings, however, a more in depth analysis could be performed in order to look for additional effects to improve the model, such as other genres; or maybe other use model approaches, such as MSE or Matrix factorization, however the RMSE was the requested method. In summary, for the primary goal of the excercise, this proposed model was adequate.