MovieLens Project

Alex Rivera Cruz

16/11/2020

Overview

This project is related to the MovieLens Project of the HarvardX: PH125.9x Data Science: Capstone course. The present report start with a general idea of the project and by representing its objective.

Then the given dataset will be prepared and setup. An exploratory data analysis is carried out in order to develop a machine learning algorithm that could predict movie ratings until a final model. Results will be explained. Finally the report ends with some concluding remarks.

Introduction

Statistical and knowledge discovery techniques are applied to the problem of producing product recommendations or ratings through recommmender systems and on the basis of previously recorded data. In the present report, the products are the movies.

The present report covers the 10M version of the movieLens dataset available here https://grouplens.org/datasets/movielens/10m/.

The Netflix prize (i.e. challenge to improve the predictions of Netfix's movie recommender system by above 10% in terms of the root mean square error) reflects the importance and economic impact of research in the recommendation systems field.

Aim of the project

The goal is to train a machine learning algorithm using the inputs of a provided training subset to predict movie ratings in a validation set. The predicted user ratings will be in the range from 0.5 to 5.

The value used to evaluate the algorithm performance is the Root Mean Square Error, or RMSE. (Low RMSE -> Better Performance)

The function that computes the RMSE for vectors of ratings and their corresponding predictors will be the following:

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

Finally, the best resulting model will be used to predict the movie ratings.

Dataset

Code provided by edx staff to download an create edx dataset:

```
##########################################################
# 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("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")

# 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")
```

The Movielens dataset will be splitted into 2 subset, a training subset (edx), and a testing subset (Validation):

```
# 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)
```

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

Methods and Analysis

Data Analysis

The edx subset contains six variables "userID", "movieID", "rating", "timestamp", "title", and "genres". Each row represent a single rating of a user for a single movie.

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 3      1     292      5 838983421               Outbreak (1995)
## 4      1     316      5 838983392               Stargate (1994)
## 5      1     329      5 838983392 Star Trek: Generations (1994)
## 6      1     355      5 838984474       Flintstones, The (1994)
##                          genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 3  Action|Drama|Sci-Fi|Thriller
## 4       Action|Adventure|Sci-Fi
## 5 Action|Adventure|Drama|Sci-Fi
## 6       Children|Comedy|Fantasy
```

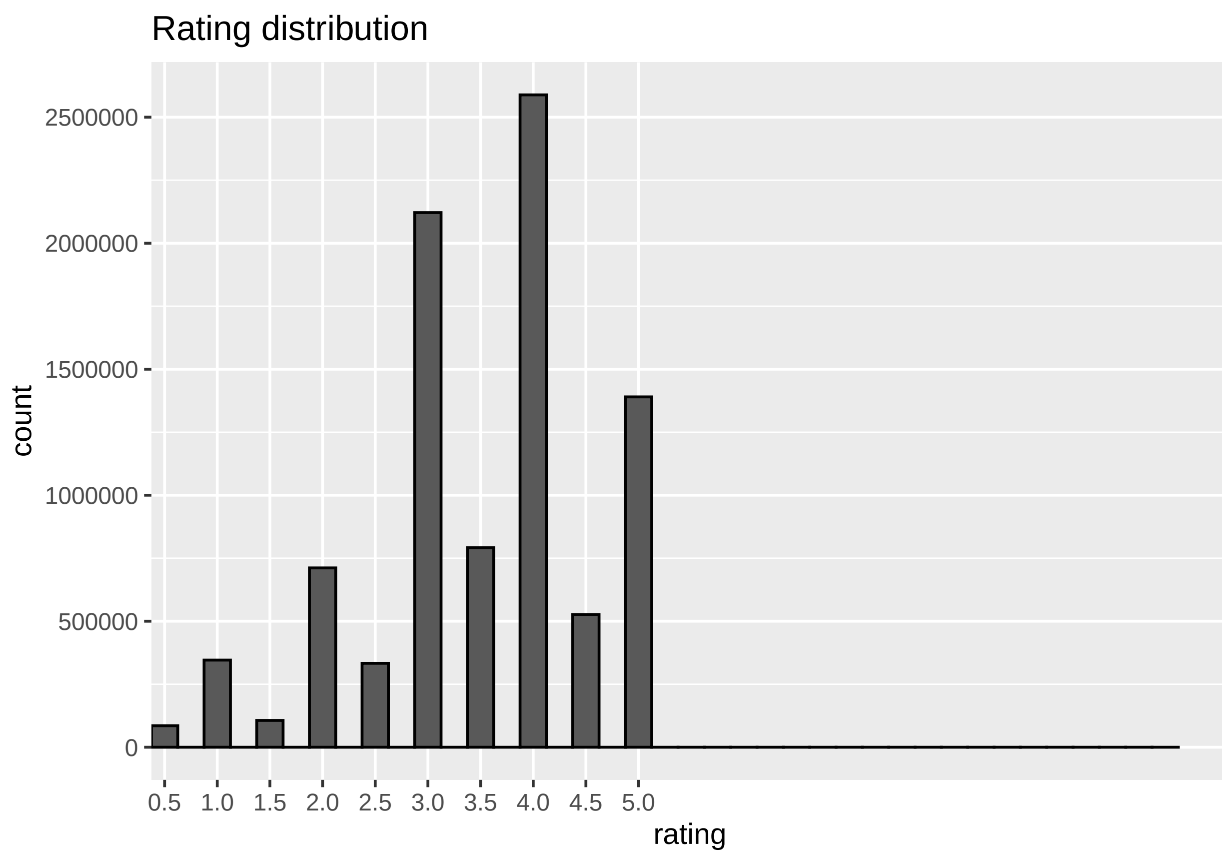
We can summarize the edx subset to confirm that there are no missing values.

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

The number of unique movies and users in the edx subset:

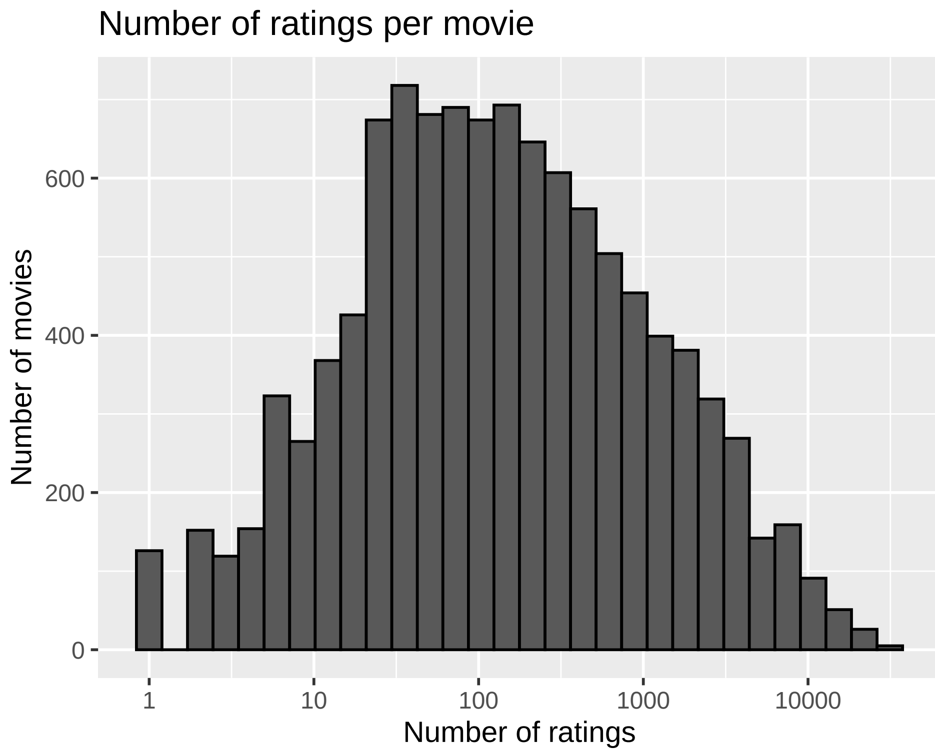
```
##   n_users n_movies
## 1   69878    10677
```

We can see, in the figure below, that the rating distribution has a left-skewed distribution. Users have a preference to rate movies rather higher than lower. The rating 4 is the most common rating, followed by 3 and 5.



We can plot the data and determine that some movies are rated more often than others, while some have very few ratings and sometimes only one rating. Thus regularization and a penalty term will be applied to the models in this report.

```
edx %>%
count(movieId) %>%
ggplot(aes(n)) +
geom_histogram(bins = 30, color = "black") +
scale_x_log10() +
xlab("Number of ratings") +
  ylab("Number of movies") +
ggtitle("Number of ratings per movie")
```



As 20 movies that were rated only once appear to be obscure, predictions of future ratings for them will be difficult.

```r
edx %>%
  group_by(movieId) %>%
  summarize(count = n()) %>%
  filter(count == 1) %>%
  left_join(edx, by = "movieId") %>%
  group_by(title) %>%
  summarize(rating = rating, n_rating = count) %>%
  slice(1:20) %>%
  knitr::kable()
```

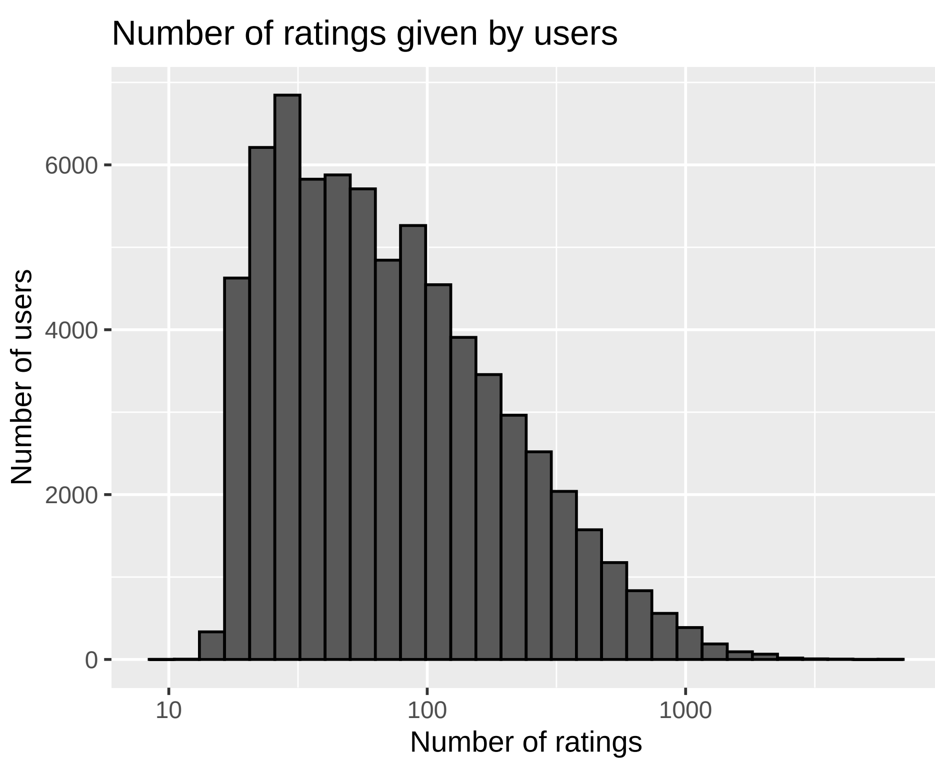
```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
```

title	rating	n_rating
1, 2, 3, Sun (Un, deuz, trois, soleil) (1993)	2.0	1
100 Feet (2008)	2.0	1
4 (2005)	2.5	1
Accused (Anklaget) (2005)	0.5	1
Ace of Hearts (2008)	2.0	1
Ace of Hearts, The (1921)	3.5	1
Adios, Sabata (Indio Black, sai che ti dico: Sei un gran figlio di…) (1971)	1.5	1
Africa addio (1966)	3.0	1
Aleksandra (2007)	3.0	1
Bad Blood (Mauvais sang) (1986)	4.5	1
Battle of Russia, The (Why We Fight, 5) (1943)	3.5	1

title	rating	n_rating
Bellissima (1951)	4.0	1
Big Fella (1937)	3.0	1
Black Tights (1-2-3-4 ou Les Collants noirs) (1960)	3.0	1
Blind Shaft (Mang jing) (2003)	2.5	1
Blue Light, The (Das Blaue Licht) (1932)	5.0	1
Borderline (1950)	3.0	1
Brothers of the Head (2005)	2.5	1
Chapayev (1934)	1.5	1
Cold Sweat (De la part des copains) (1970)	2.5	1

The majority of users have rated below 100 movies, but also above 30 movies (a user penalty term will be included in the models).

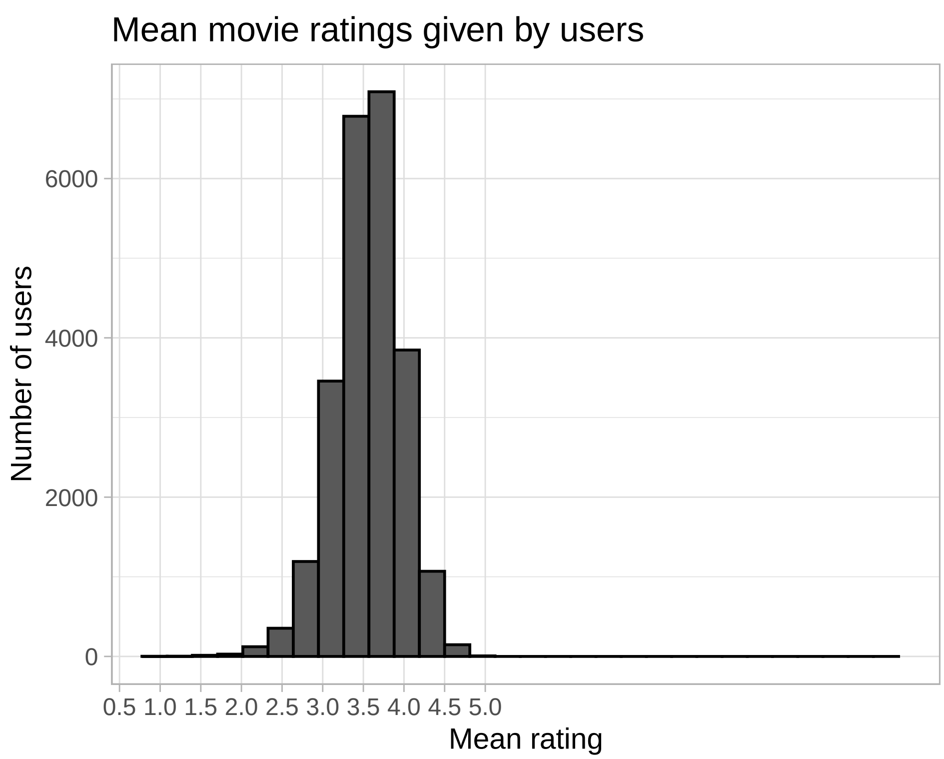
```
edx %>%
count(userId) %>%
ggplot(aes(n)) +
geom_histogram(bins = 30, color = "black") +
scale_x_log10() +
xlab("Number of ratings") +
ylab("Number of users") +
ggtitle("Number of ratings given by users")
```



Furthermore, users differ vastly in how critical they are with their ratings. Some users tend to give much lower star ratings and some users tend to give higher star ratings than average. The visualization below includes only users that have rated at least 100 movies.

```r
edx %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black") +
  xlab("Mean rating") +
  ylab("Number of users") +
  ggtitle("Mean movie ratings given by users") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  theme_light()
```

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



Modelling Approach

We write function, previously anticipated, that compute the RMSE, defined as follows:

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

With N being the number of user/movie combinations and the sum occurring over all these combinations. The RMSE is our measure of model accuracy.

The written function to compute the RMSE for vectors of ratings and their corresponding predictions is:

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

The quality of the model will be assessed by the RMSE (the lower the better).

I. Naive model

Creating a prediction system that only utilizes the sample mean. This implies that every prediction is the sample average.

This is a model based approach which assumes the same rating for all movie with all differences explained by random variation :

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

with $\epsilon_{u,i}$ independent error sample from the same distribution centered at 0 and μ the "true" rating for all movies. This very simple model makes the assumption that all differences in movie ratings are explained by random variation alone. We know that the estimate that minimize the RMSE is the least square estimate of $Y_{u,i}$, in this case, is the average of all ratings:

```r
mu <- mean(edx$rating)
mu
```

```
## [1] 3.512465
```

Getting the first naive RMSE: The resulting RMSE using this approach is quite high.

```r
naive_rmse <- RMSE(validation$rating, mu)
naive_rmse
```

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

Here, we represent results table with the first RMSE:

```r
rmse_results <- data_frame(method = "Naive model", RMSE = naive_rmse)
```

```
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

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

method	RMSE
Naive model	1.061202

This give us our baseline RMSE to compare with next modelling approaches.

II. Movie effect model

Higher ratings are mostly linked to popular movies among users and the opposite is true for unpopular movies.

We compute the estimated deviation of each movies' mean rating from the total mean of all movies μ. The resulting variable is called "b" (as bias) for each movie "i" b_i, that represents average ranking for movie i:

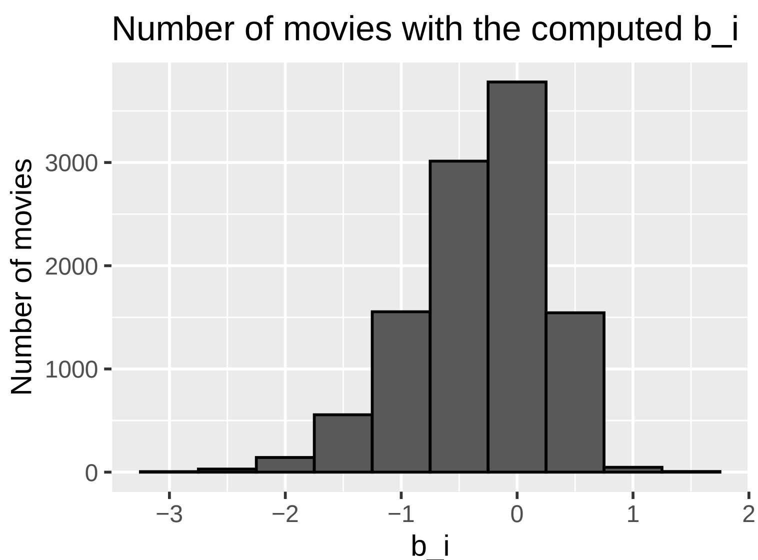
$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

The histogram is left skewed, implying that more movies have negative effects

```
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

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

```
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"),
ylab = "Number of movies", main = "Number of movies with the computed b_i")
```



This is called the penalty term movie effect.

Our prediction improve once we predict using this model.

```
predicted_ratings <- mu +  validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
model_1_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie effect model",
                                     RMSE = model_1_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Naive model	1.0612018
Movie effect model	0.9439087

So we have predicted movie rating based on the fact that movies are rated differently by adding the computed b_i to μ. If an individual movie is on average rated worse that the average rating of all movies μ , we predict that it will rated lower that μ by b_i, the difference of the individual movie average from the total average.

III. Movie and user effect model

It is understood that users may have a tendency to rate movies higher or lower than the overall mean. Let's add this into the model. First we'll calculate the bias for each user:

$$b_u = Mean_{user} - \mu$$

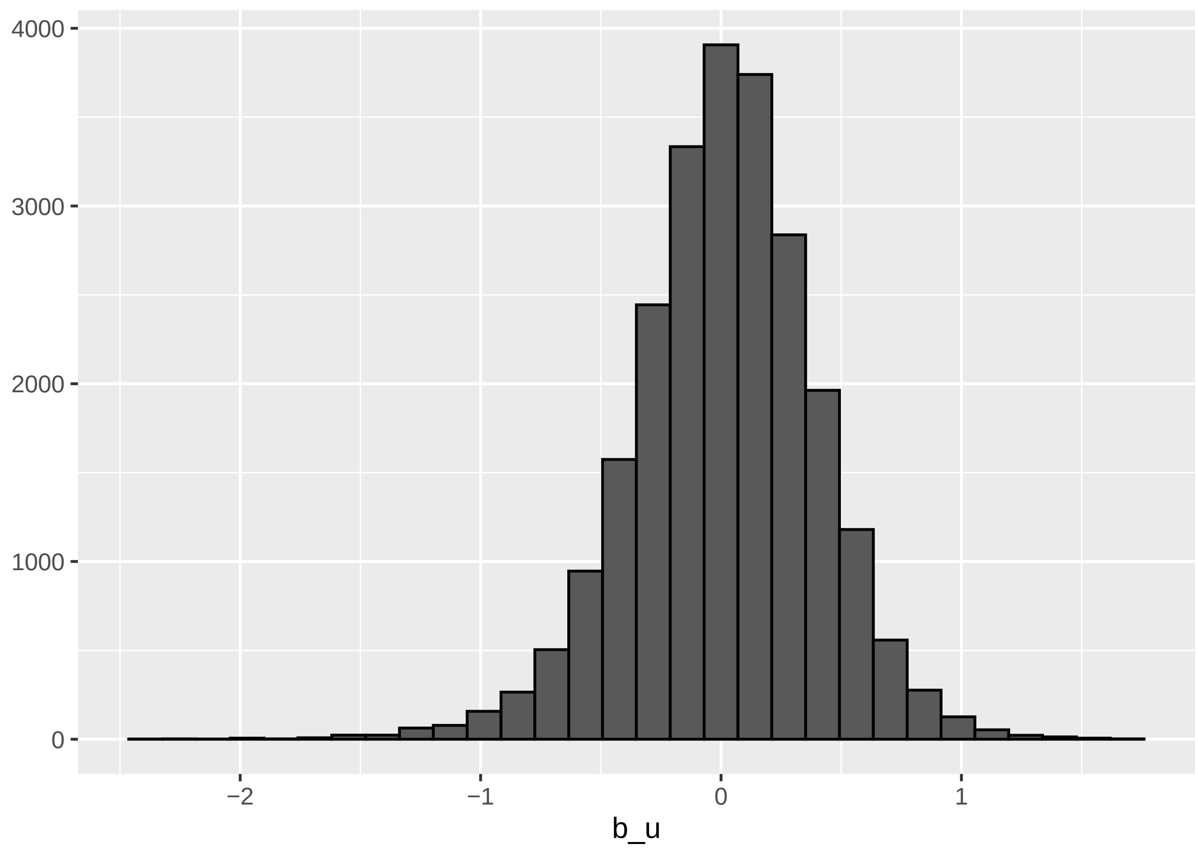
Then we'll combine the bias of a user, with the bias of a film and add both to the overall mean for a combined bias rating for each unique combination of a user rating for a given film.

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

```
user_avgs<- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

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

```
user_avgs%>% qplot(b_u, geom ="histogram", bins = 30, data = ., color = I("black"))
```



We compute an approximation by computing μ and b_i, and estimating b_u, as the average of

$$Y_{u,i} - \mu - b_i$$

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

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

We can now construct predictors and see RMSE improves:

```
predicted_ratings <- validation%>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
model_2_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie and user effect model",
                                     RMSE = model_2_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Naive model	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488

Our rating predictions further reduced the RMSE. But we made stil mistakes on our first model (using only movies). The supposes “best “ and “worst “movie were rated by few users, in most cases just one user. These movies were mostly obscure ones. This is because with a few users, we have more uncertainty. Therefore larger estimates of b_i, negative or positive, are more likely. Large errors can increase our RMSE.

IV. Regularized movie and user effect model

This model implements the concept of regularization to account for the effect of low ratings’ numbers for movies and users. In previous sections we demonstrated that few movies were rated only once and that some users only rated few movies.Hence this can strongly influence the prediction. Regularization is a method used to reduce the effect of overfitting.

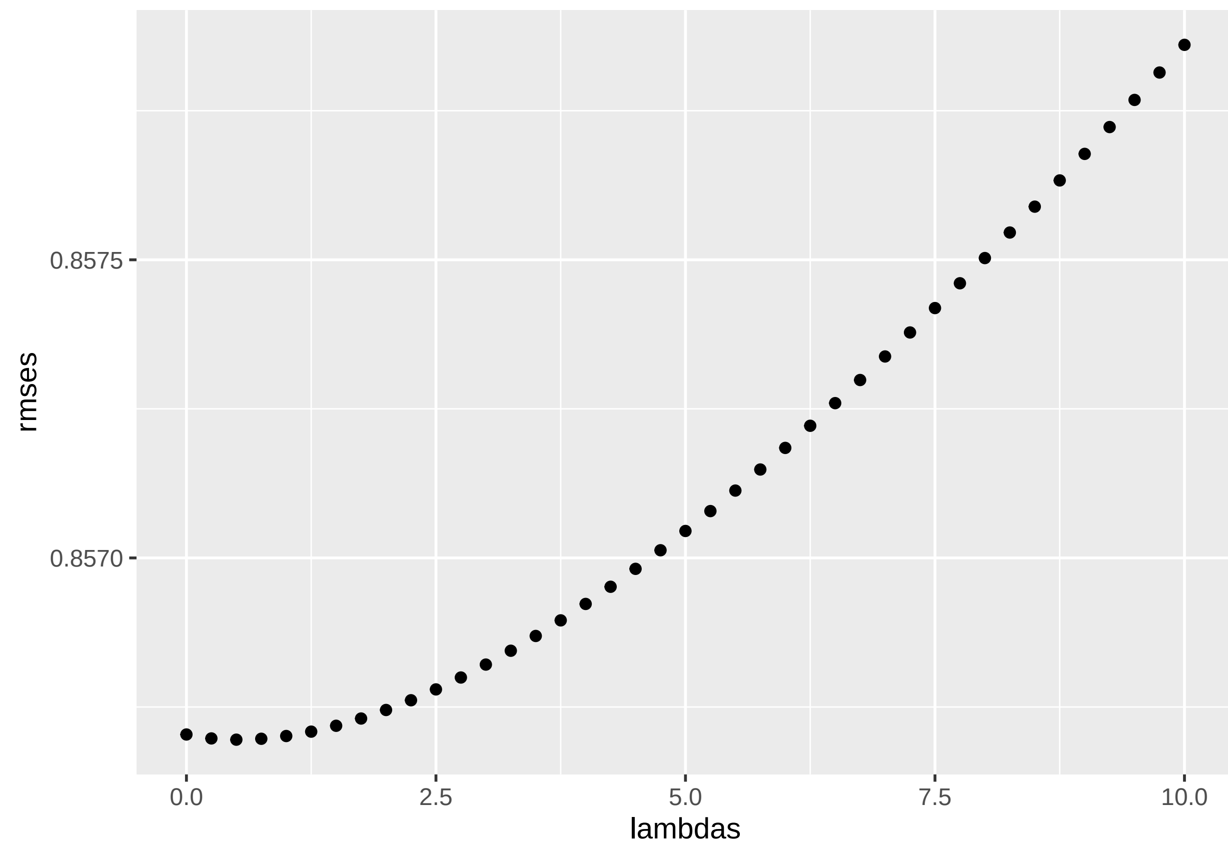
We should find the value of lambda (that is a tuning parameter) that will minimize the RMSE. This shrinks the b_i and b_u in case of small number of ratings.

Consider that we have to choose the best lambda based on the training data (edx) , not based in validation date because it would produce overtraining. (Rmemeber, you don’t know the ratings of validation data, it’s only used for testing)

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){

  mu <- mean(edx$rating)

  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l),.groups = 'drop')

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

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

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

We plot RMSE vs lambdas to select the optimal lambda

```
qplot(lambdas, rmses)
```



The optimal lambda is:

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

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

For the full model, the optimal lambda is: 0.5

The new results based on validation data will be:

```
b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda),.groups = 'drop')

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

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

```r
model_3_rmse <- RMSE(predicted_ratings, validation$rating)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized movie and user effect model",
                                     RMSE = model_3_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Naive model	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488
Regularized movie and user effect model	0.8652226

Results

The RMSE values of all the represented models are the following:

method	RMSE
Naive model	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488
Regularized movie and user effect model	0.8652226

We therefore found the lowest value of RMSE that is 0.8652226.

Discussion

So we can confirm that the final model for our project is the Regularized movie and user effect model:

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

As expected, the RMSE decreased while the model increases complexity. We can even discuss if a good alternative is to increase the parameters of the models like inlude genres, however to split the genres in the data we need more advanced hardware.

Conclusion

Machine learning algorithm was built to predict movie ratings with MovieLens dataset.

The Regularized movie and user effect model is characterized by the lower RMSE value and is hence the optimal model to use for the present project.

We could also say that some improvements in the RMSE could be achieved by adding other parameters (genre, year, age). Other different machine learning models could also improve the results further, but hardware limitations, as the RAM, are a constraint.