Capstone Movielens

Lionel Fournier

17/05/2020

Contents

1 Overview

Recommendation systems are growing more and more important. It helps company to offer the most suitable product for customers without them having to look. It's especially important if they offer a large amount of choice. For streaming services, an efficient recommendation system is essential, and it will be the goal of this project.

2 Introduction

In this project, we are going to use machine learning to predict the rating of the users. We are going to use the movielens dataset (https://grouplens.org/datasets/movielens/10m/) containing 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users. This data set contains only movies released before 2009. We are going to first prepared the data and then analyse it. We then will construct our machine algorithm, starting for a very simple iteration and then adding complexity to achieve better result.

3 Methodology

First we are going to prepare the data. We are going to split it into a validation set and a training one. Then we are going to analyse them. The goal is to dive is some element that may affect our different models. We are specificaly going to look onto the effect movies have on rating. Not all movies have the same impact, while blockbusters are widely seen, some obscures movies have a very limited audience, and that may create a bias in the rating. We are also going to look into the user's rating pattern. Not all users rate the same and it can also impact our models. And finally, the "age" of a movie, more clearly the number of years between the movie release and when it was rated.

From there we are going to create for models, taking into account what we learned in our analysis. The first one will be naive and take only into account the movie average. The second one will add a movie effect b_i. The third one will add a user effect b_u. The fourth one will add a movie "age" effect b_a.Finaly, we are going to add a tuning parameter lambda to our model to further improve our predictions.

To test the accuracy of each model, we are going to use the residual mean squared error (RMSE). The RMSE as we will use it in R:

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

We can interpret the rmse as the typical error we make when predicting a movie rating. If this number is larger than 1, it means our typical error is larger than one star. The smaller the number, the better the model. Our goal will be to reach a rmse smaller than 0.8649.

4 Data preparation

4.1 Creation of the data set

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

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

#Warning: This part run well on R 3.6.2, but may encounter trouble on R 4.0

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
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)

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

4.2 Data preparation

Our first step will be to have a quick look at the data.

```
#Checking the first entries of the edx dataset
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                         genres
## 1                Comedy|Romance
## 2          Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7       Children|Comedy|Fantasy
```

From our first look at the data, we can see that edx has 6 columns. Some useful informations in the data are not useable in this current form and we are going to extract them. All those transformation must be done in the edx set and in the validation set.

First step, we want to transform the timestamp into a format that is readable for humans. We are going to extract the year of the rating from the timestamp.

```
#changing timestamp to have year of rating
edx$date<-as.POSIXct(edx$timestamp,origin="1970-01-01")
edx$rateYear<-format(edx$date,"%Y")

validation$date<-as.POSIXct(validation$timestamp,origin="1970-01-01")
validation$rateYear<-format(validation$date,"%Y")
```

In the data set, the title of the movie and its year of release are merged into one column, we want to separate them into two columns, one containing the title and the other one the year of release.

```
#extract year of the movie release
edx<-edx%>%
  extract(title, c("titletemp", "year"), regex = "^(.*) \\(([0-9 \\-]*)\\)$", remove = F)%>%
  mutate(title=titletemp)%>%
  select(-titletemp)

validation<-validation%>%
  extract(title, c("titletemp", "year"), regex = "^(.*) \\(([0-9 \\-]*)\\)$", remove = F)%>%
  mutate(title=titletemp)%>%
  select(-titletemp)
```

We also want to create a new column with the age of the movie when it was rated. We use for that the rating year and the release year that we just extracted from the data.

```
#Extract age of movie, as the diffrence between when the movie was rated and when it was released
edx<-edx%>%mutate(ageOfMovie= as.integer(rateYear)-as.integer(year))

validation<-validation%>%mutate(ageOfMovie= as.integer(rateYear)-as.integer(year))
```

Finally, we clean our data set from some temporary information we created during the process and we keep the element that we are going to use in our analysis and to create our prediction models.

```
#Removing the columns we won't use
edx<-edx%>% select(userId, movieId, rating, title, year, genres, rateYear, ageOfMovie)

validation<-validation%>% select(userId, movieId, rating, title, year, genres, rateYear, ageOfMovie)
```

5 Analysis

Before we start building our models, we need to analyse our data in order to better understand the distribution of ratings and the importance of different predictors. We will use what we learn to build better models-

```
#Computing the summary of the edx database
summary(edx)
```

```
##      userId         movieId          rating         title
##  Min.   :    1   Min.   :    1   Min.   :0.500   Length:9000055
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   Class :character
##  Median :35738   Median : 1834   Median :4.000   Mode  :character
##  Mean   :35870   Mean   : 4122   Mean   :3.512
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000
##  Max.   :71567   Max.   :65133   Max.   :5.000
##      year           genres            rateYear          ageOfMovie
##  Length:9000055     Length:9000055     Length:9000055     Min.   :-2.00
##  Class :character   Class :character   Class :character   1st Qu.: 2.00
##  Mode  :character   Mode  :character   Mode  :character   Median : 7.00
##                                                           Mean   :11.98
##                                                           3rd Qu.:16.00
##                                                           Max.   :93.00
```

Using summary to get a better understanding of the data, we can see that the average rating in edx is at 3.512.

```
#number of user and nomber of movie
edx%>%summarize(num_user=n_distinct(userId),num_movie=n_distinct(movieId))
```

```
##   num_user num_movie
## 1    69878     10677
```

The data set countains 69878 users and 10677 movies.

5.1 Genres

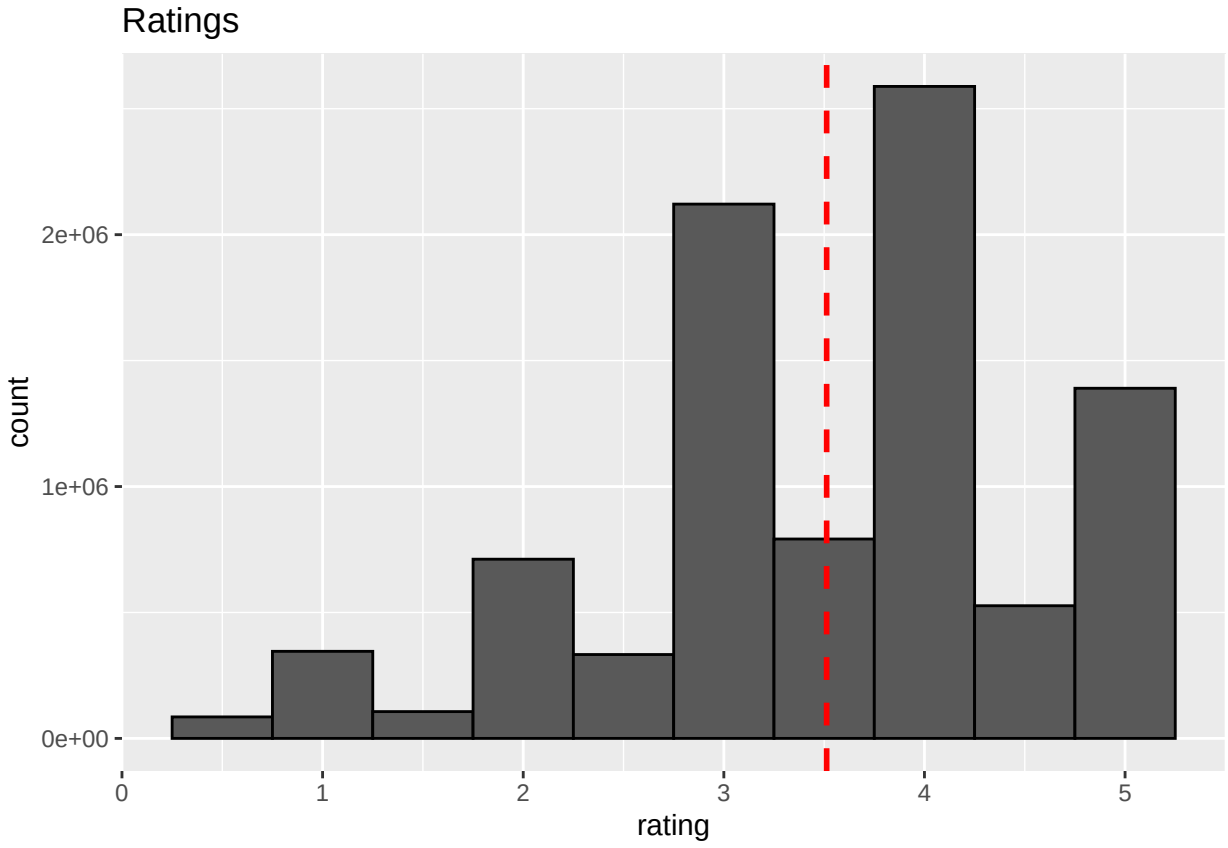
The movies are classified by genres. We are not going to use the genre in our model, but this information is still worth a look.

```
## # A tibble: 6 x 2
##   genres                                                n
##   <chr>                                             <int>
## 1 (no genres listed)                                    7
## 2 Action                                            24482
## 3 Action|Adventure                                  68688
## 4 Action|Adventure|Animation|Children|Comedy         7467
## 5 Action|Adventure|Animation|Children|Comedy|Fantasy  187
## 6 Action|Adventure|Animation|Children|Comedy|IMAX      66
```

We can see that one movie can be classified by multiple genre. Few of them, 7 to be exact, have no genres listed.

5.2 Rating

Users tend to rates more often good movie, as the rating distribution show. 4, 3, 5 and 3,5 are the 4 top ratings. We can also see in this graph that users tend to give less often half rates than full one.

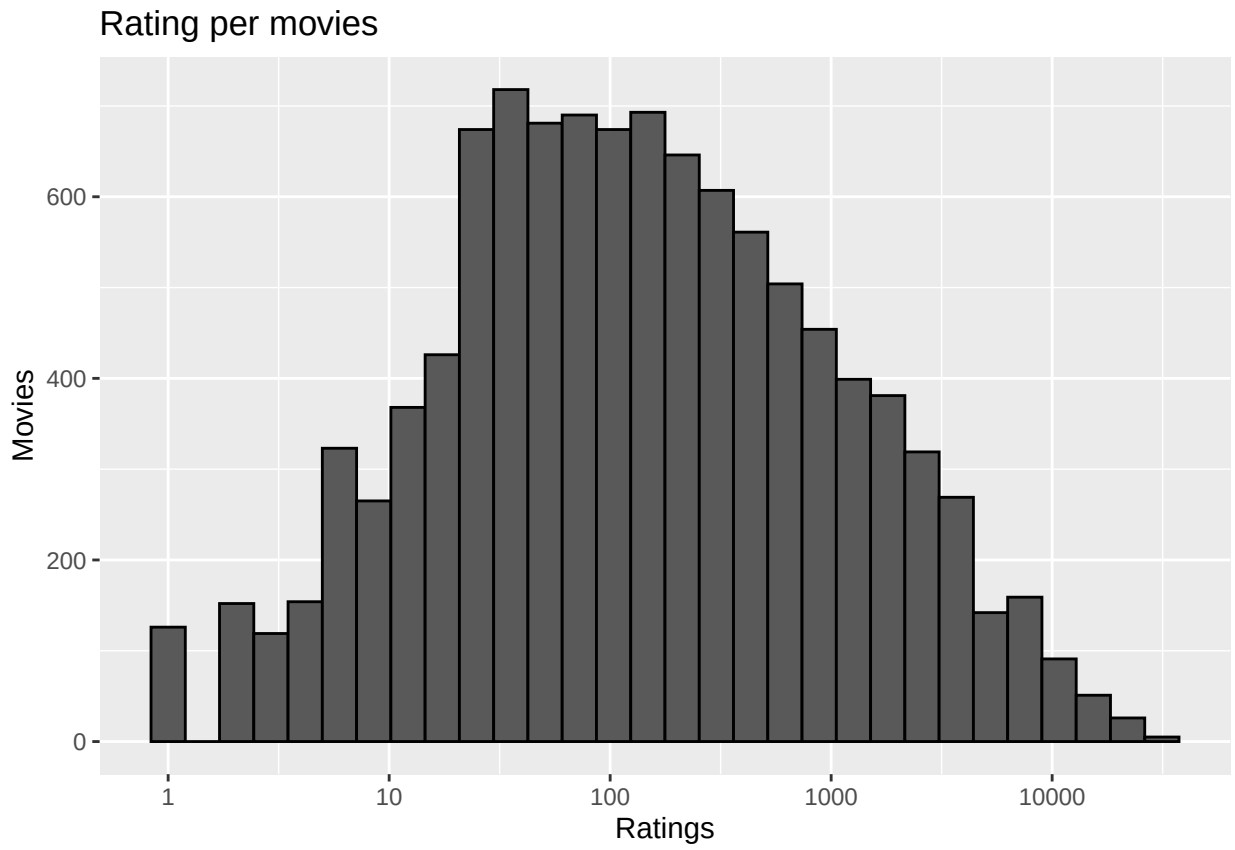


Not all movies get as much rating. Blockbuster and iconic movies get rated more often, as shown by this list of the most rated movies.

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                count
##      <dbl> <chr>                                                <int>
##  1     296 Pulp Fiction                                         31362
##  2     356 Forrest Gump                                         31079
##  3     593 Silence of the Lambs, The                            30382
##  4     480 Jurassic Park                                        29360
##  5     318 Shawshank Redemption, The                            28015
##  6     110 Braveheart                                           26212
##  7     457 Fugitive, The                                        25998
##  8     589 Terminator 2: Judgment Day                           25984
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) 25672
## 10     150 Apollo 13                                            24284
## # ... with 10,667 more rows
```

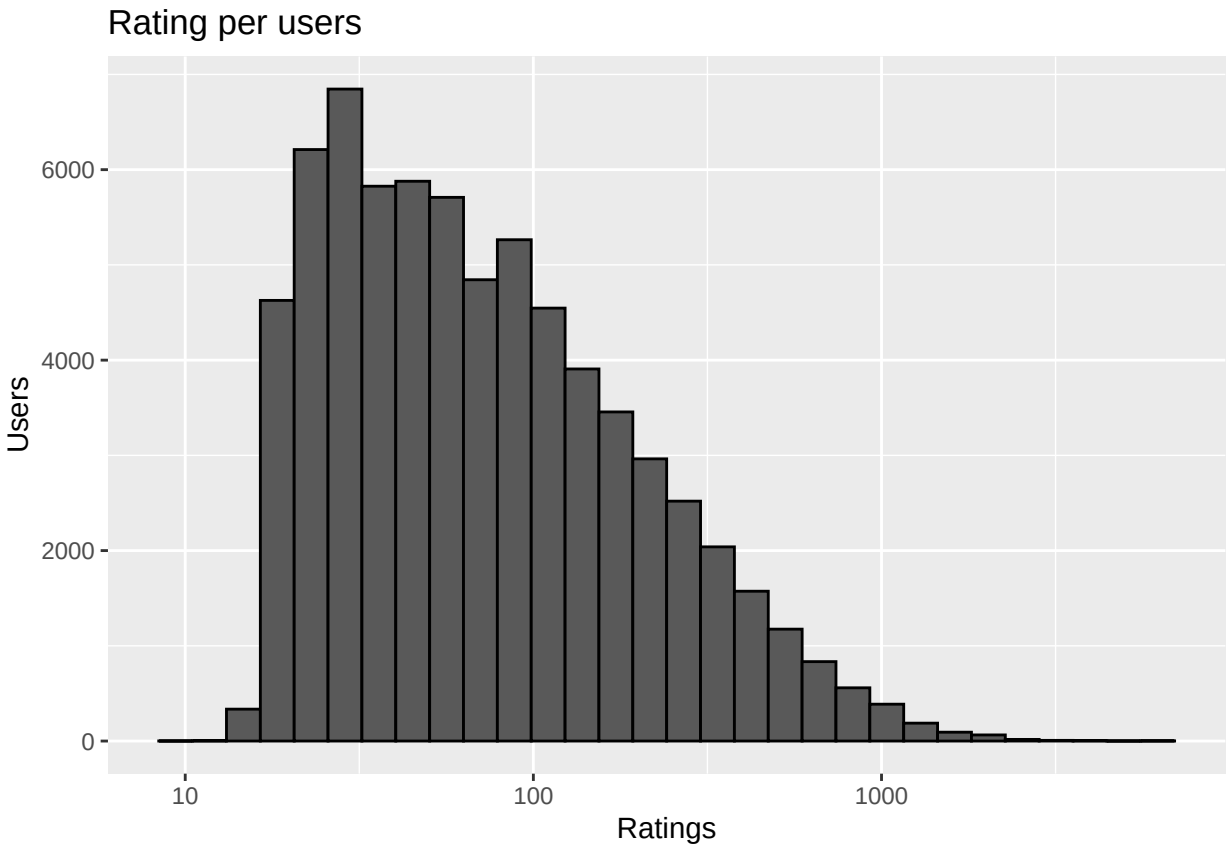
5.3 Movies

As we have seen, some movies get a lot of reviews, as shown in this graph, on the other hand, some more obscure one get very few ratings, some off them only get one rating, making it tricky for our models as it may create outliers.

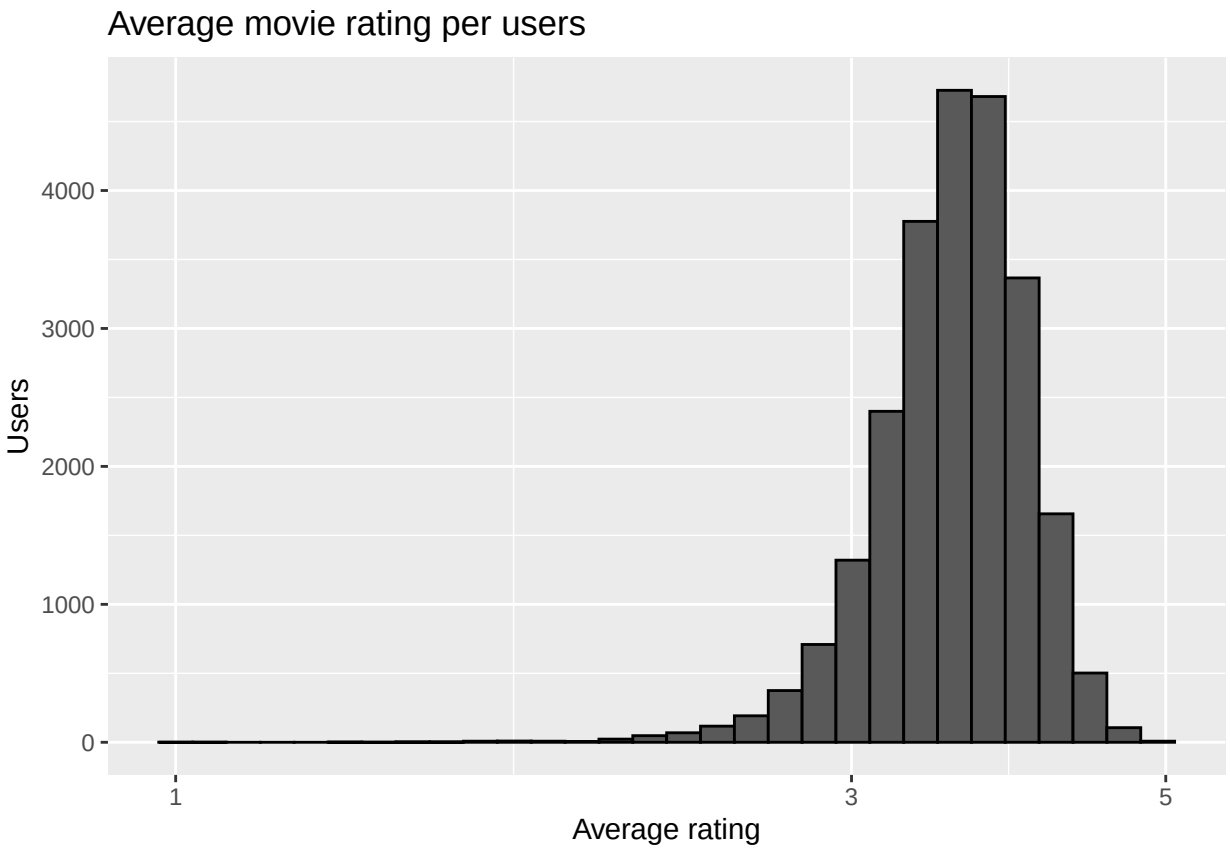


5.4 Users

If not all movies are equally rated, the users also show different patterns. Some of them will only rate 10 movies, but some do rate more than a thousand of them.



Users have also different standards of how they rate a movie. Some users are very generous and will easily give 5 stars to a movie, while other have a lower average. In this graph we can see the average rating for heavy users that rated at least 100 movies.



5.5 Age of movie

Our data set contains rating from 1995 to 2009.

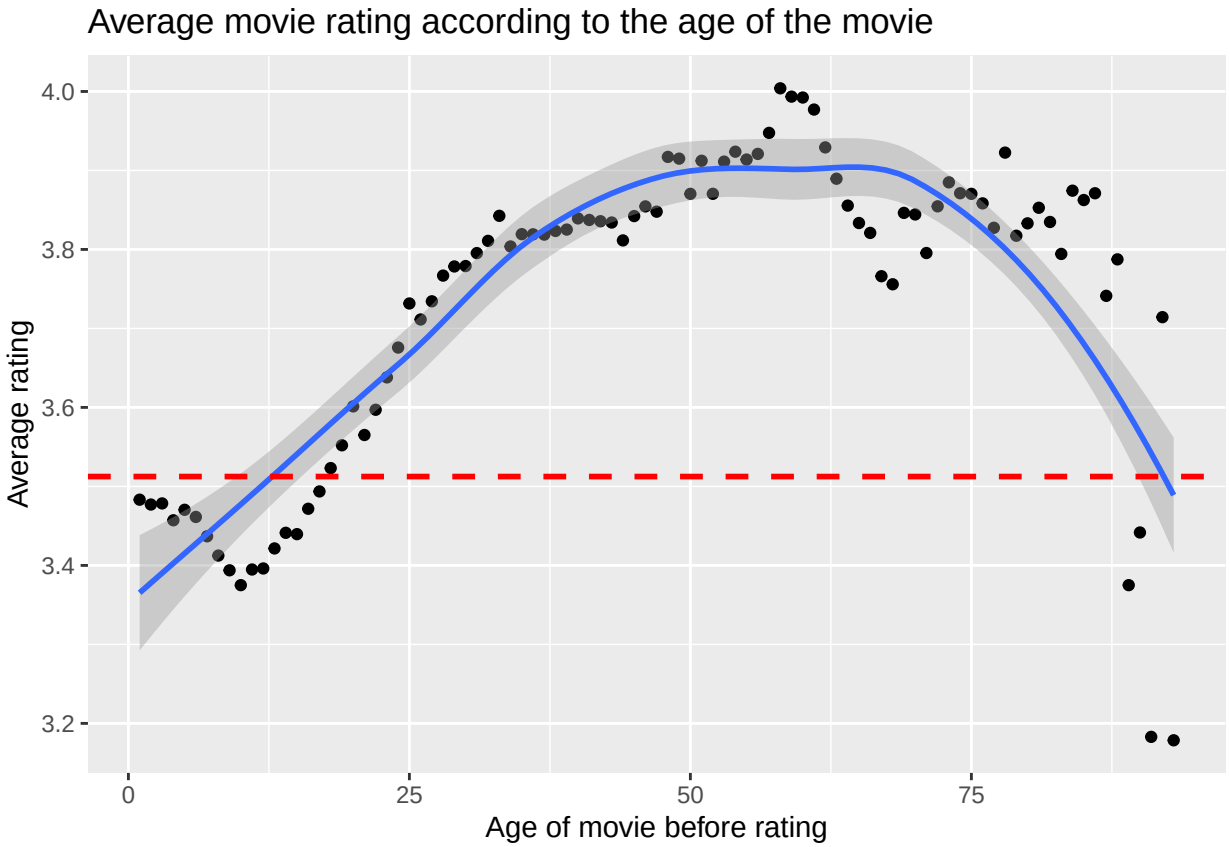
```
#Range of rating years
range(edx$rateYear)
```

```
## [1] "1995" "2009"
```

The movie in the data set are from 1915 to 2008.

```
#Range of movie release years
range(edx$year)
```

```
## [1] "1915" "2008"
```



From this graph, we can see that movies that are less than 20 years old when rated tend to be rated beyond average, while older movies get better ratings. It may be linked to the fact that old movies which are still watched today are mostly "classic" or largely appreciated movies, while new movies are not so well know and users may watch them not knowing the quality of the movies.

6 Modeling

Based on our analysis, we are going to build few machine learning algorithm through a training set and a test set from edx. Our goal is to achieve the lowest RMSE. We are going to test our best model with the edx and the validation dataset.

```
#Creating a test_set and a training set
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

6.1 Average model

For our first model, we find the mean rating (mu) from edx and consider it as the rating for all the movies. We predict all the unknown rating as mu, in order to get a first naive rmse

```r
#getting the average
mu<-mean(train_set$rating)
mu
```

```
## [1] 3.512482
```

```r
#test with simple prediction
naive_rmse<- RMSE(test_set$rating,mu)
naive_rmse
```

```
## [1] 1.059904
```

```r
#Save the result of our first model in a data frame
rmse_results<- data_frame(method="Average rating model", RMSE=naive_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Average rating model	1.059904

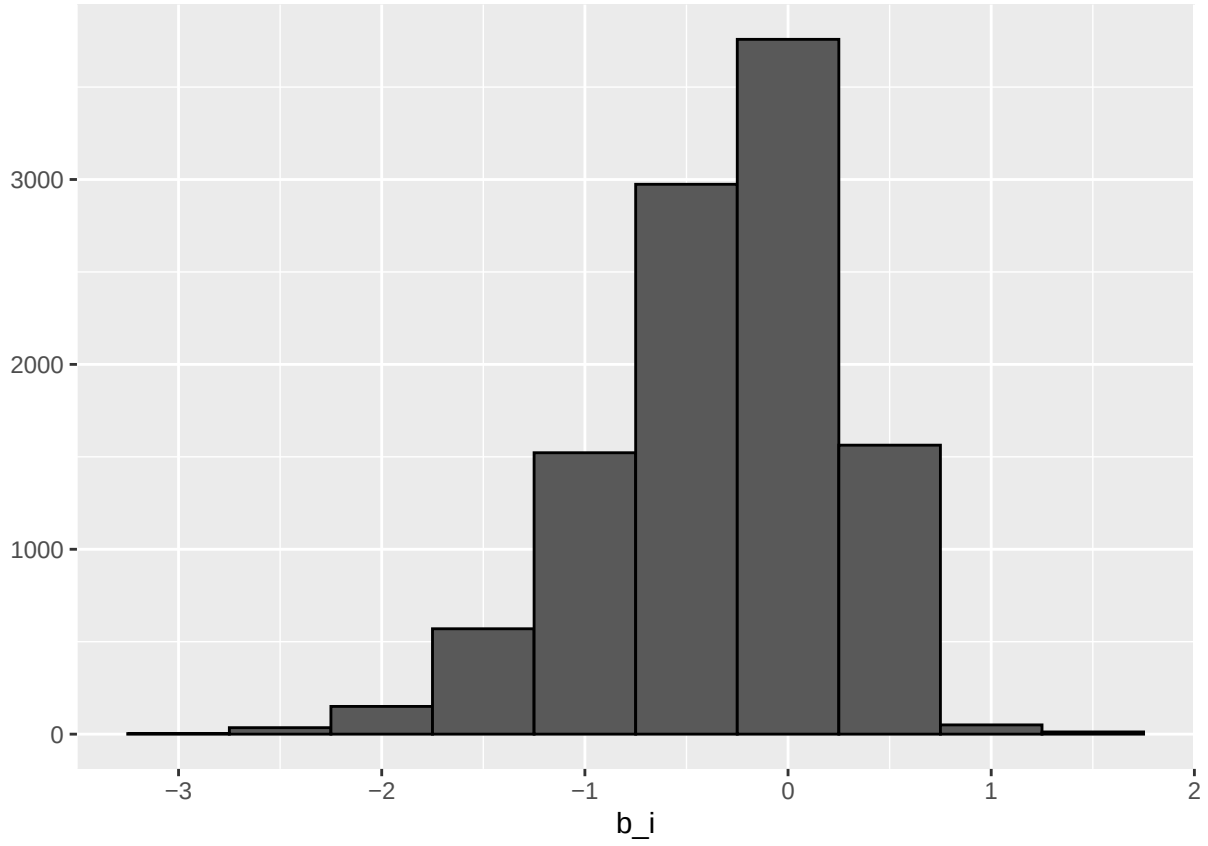
With a RMSE of 1.0599, this model is not very useful and we can improve it. This result will allow us to see if our next models can do better than simply predicting the average.

6.2 Model with movie effect

As we have seen in our analysis, all movies are not rated the same. Popular or iconic movies and blockbusters tend to have a higher rating. To take that element into consideration we compute the standard deviation of each movie from the mean and calculate the mean of those deviataions. b_i will show the "bias" of movie rating, or the movie effect.

```r
#We add a movie effect for this model, by subtracting the mean from the rating for movies
#we create a variable with the movie effect b_i
movie_avgs<- train_set%>%
  group_by(movieId)%>%
  summarize(b_i=mean(rating-mu))
```

We can look at the movie effect distribution.




```
#test and add the result to our table
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i

model_1_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = model_1_rmse ))

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

method	RMSE
Average rating model	1.0599043
Movie Effect Model	0.9437429

With a RMSE of 0.9437, this model already shows clear improvement, but leaves some room to do better.

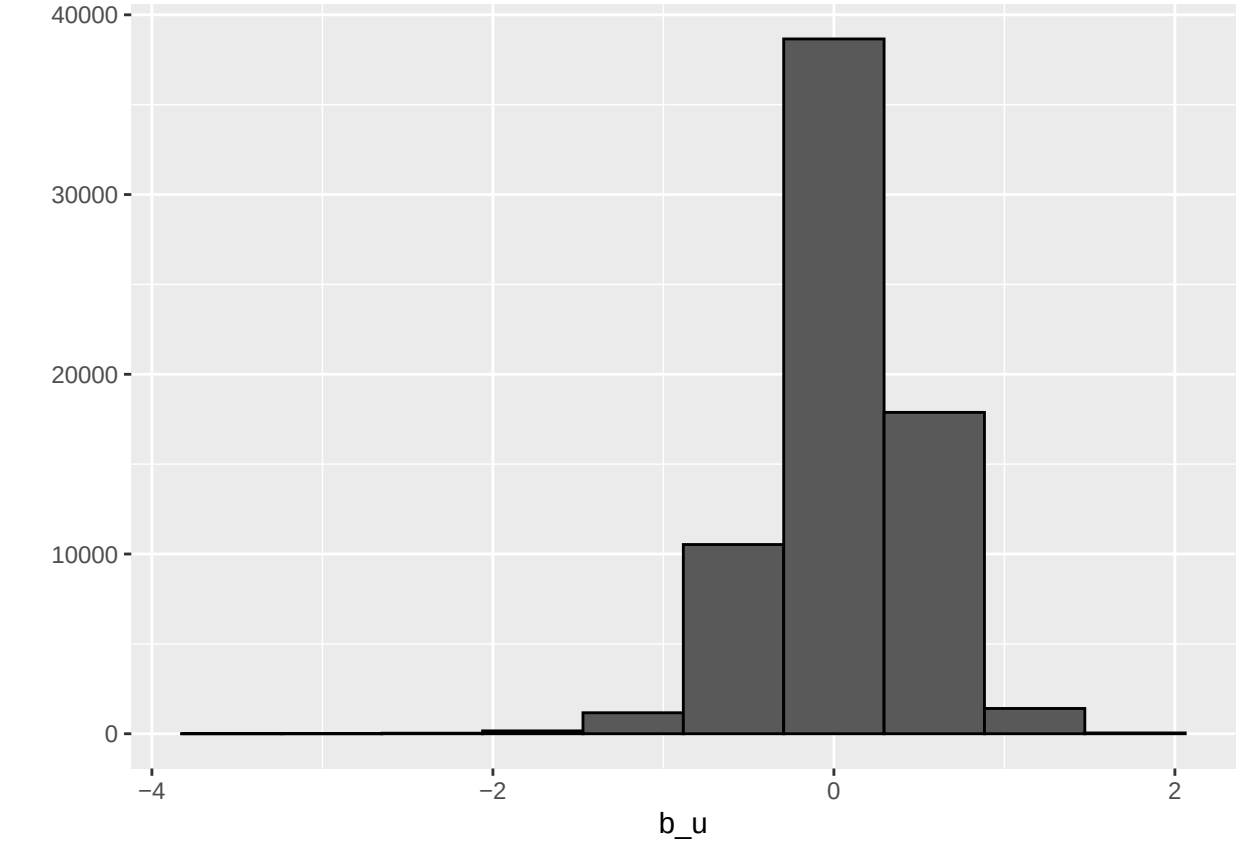
6.3 Movie and user effects

To improve our predictions, we can now take into account the user effect or "bias" (b_u) and add it to our model. As we have seen in our analysis some users are pretty generous in their rating and some are less

impressed by most movies. We make an approximation of that effect as the mean of the users rating minus the average minus b_i.

```
# as for the movie effect we create a variable users_avgs
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

We can look at the user effect distribution




```
#test and add the result to our table
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effects Model",
                                     RMSE = model_2_rmse ))
```

With a new rmse of 0.8659, we definitely improve our prediction with the user effect.

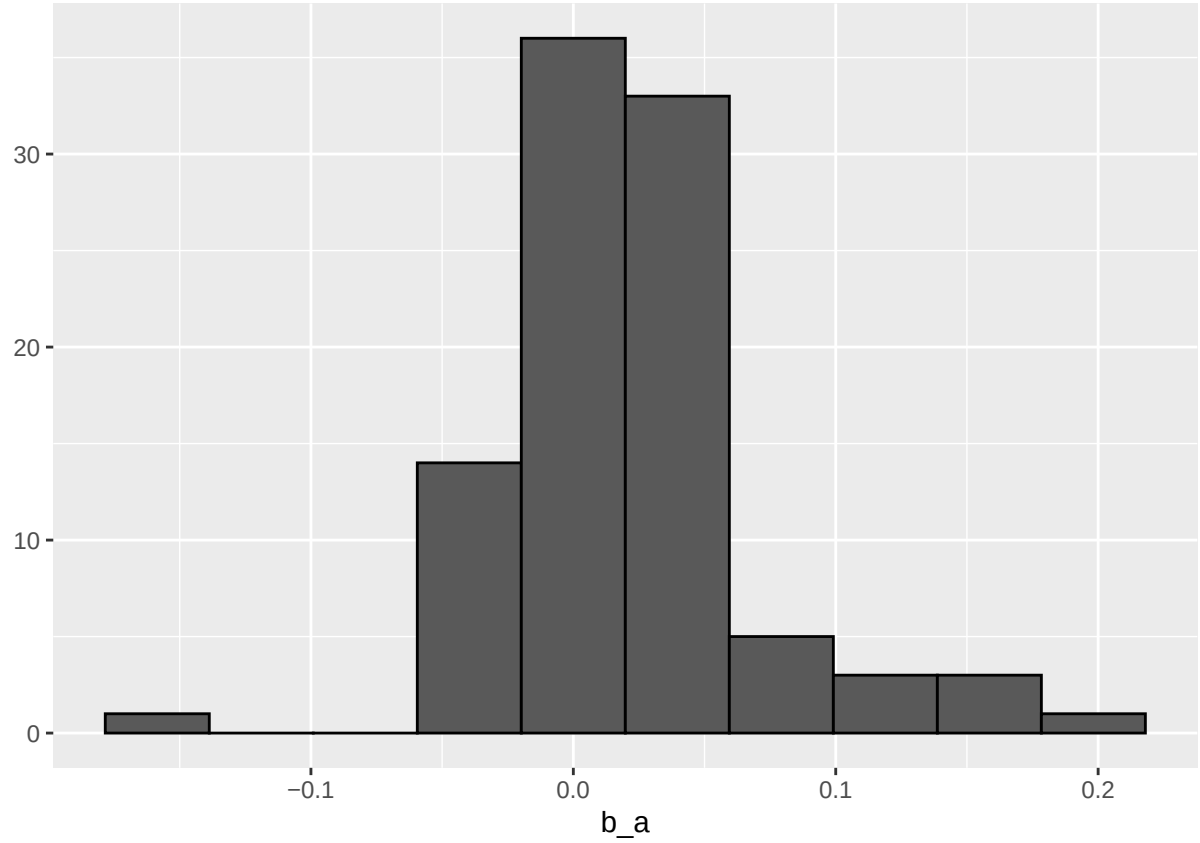
method	RMSE
Average rating model	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659320

6.4 Movie, user and age of movie effects

To improve our predictions, we can now take into account the age of movie when rated effect or "bias" (b_a) and add it to our model. As we have seen in our analysis, newer movies tend to be rated above average, while older one get better rating. We make an approximation of that effect as the mean of the users rating minus the average, b_i and b_u.

```
# we create a variable age_avgs to quantify the age of movie effect
age_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by= 'userId')%>%
  group_by(ageOfMovie) %>%
  summarize(b_a = mean(rating - mu - b_i - b_u))
```

We can look at the age of movie effect distribution




```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
```

```
  left_join(age_avgs, by='ageOfMovie') %>%
  mutate(pred = mu + b_i + b_u + b_a) %>%
  .$pred

model_3_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User + Age of movie Effects Model",
                                     RMSE = model_3_rmse ))
```

With a new rmse of 0.8654, the age of movie effect brings a little improvement to our model, but not enough to reach our goal.

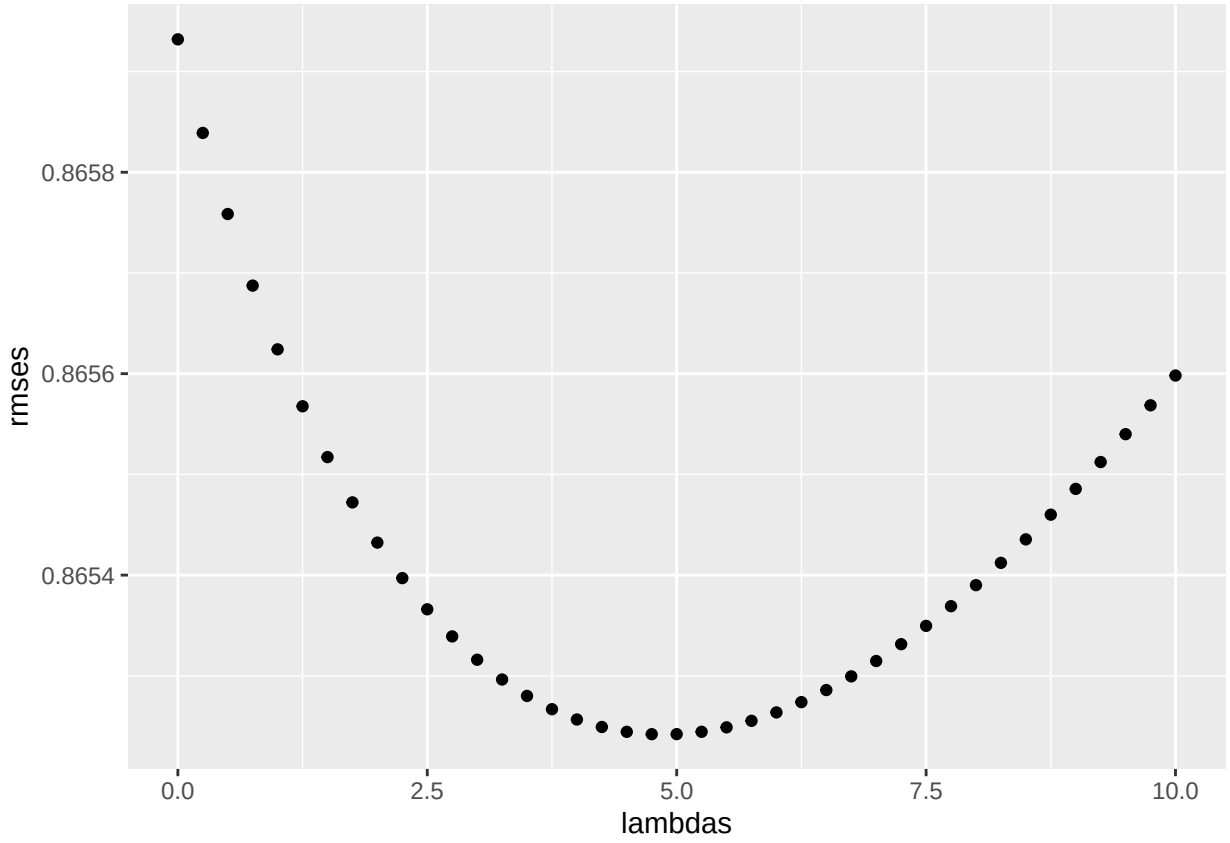
method	RMSE
Average rating model	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659320
Movie + User + Age of movie Effects Model	0.8654978

6.5 Model Movie and user effects regularized

To improve our model we are going to regularized the movie and user effects. For that we create a parameter lambda that we will defined by cross validation with the test set to optimize this parameter. It allows us to add a penalty on the different parameters of the model to make it less likely to fit the noise of the training data.

```
#we add a tuning parameter lambda, choosed by cross-validation
lambdas <- seq(0, 10, 0.25)


#Find b_i and b_u for each lambda as well as rating prediction and test add year, have to figure out va
#Note: this part may take few minutes when runing
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
```




```
#Choose the optimal lambda
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 4.75
```

Our optimal lambda is 4.75. With that we reach a rmse of 0.8652.

```
#test and save the result to our table
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie + User Effect Model",
                                     RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

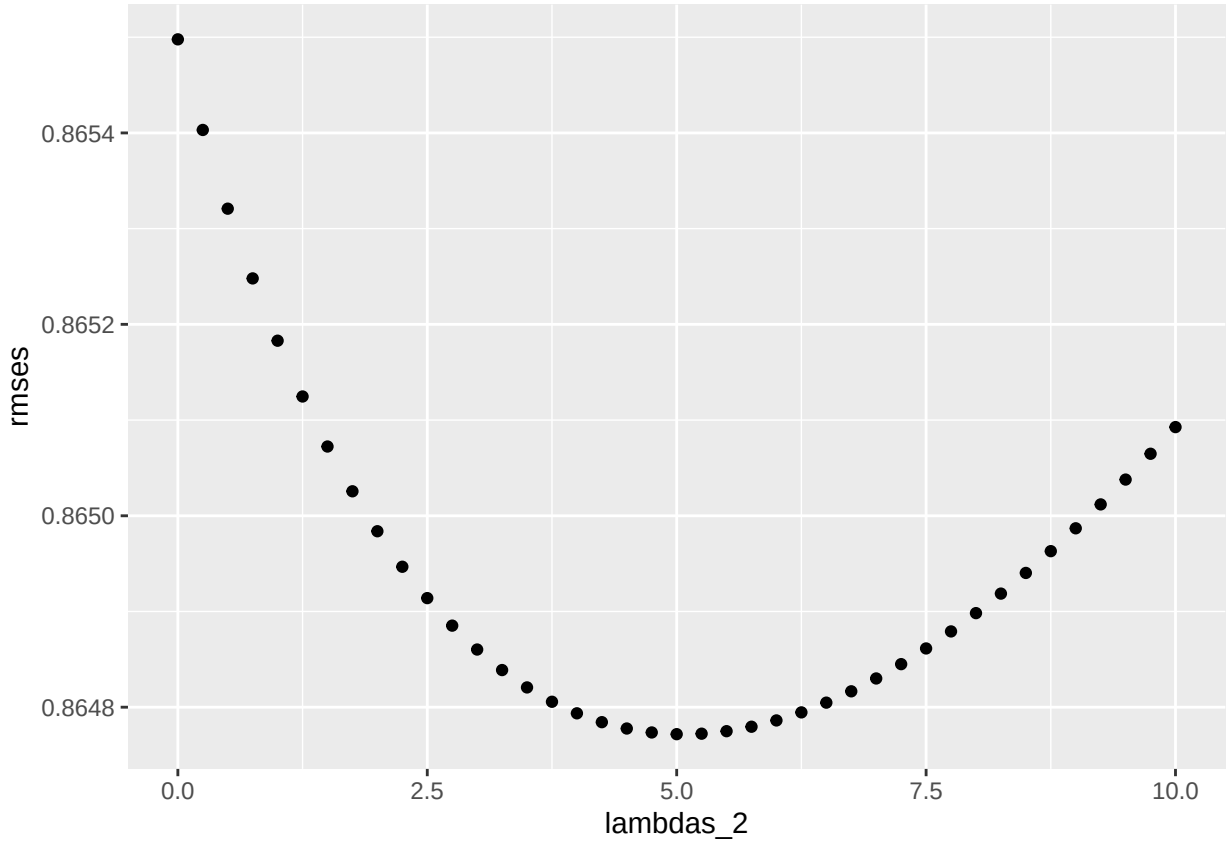
method	RMSE
Average rating model	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659320
Movie + User + Age of movie Effects Model	0.8654978
Regularized Movie + User Effect Model	0.8652421

6.6 Model Movie, user and age of movie effects regularized

We are now going to expand the previous model. Our regularized model will now also take account the age of movie effect.

```
#we add a tuning parameter lambda, choosed by cross-validation
lambdas_2 <- seq(0, 10, 0.25)


#Find b_i, b_u and b_a for each lambda as well as rating prediction and test
#Note: this part may take few minutes when runing
rmses <- sapply(lambdas_2, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  b_a <-train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(ageOfMovie) %>%
    summarize(b_a = sum(rating - b_i - b_u - mu)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_a, by = "ageOfMovie") %>%
    mutate(pred = mu + b_i + b_u + b_a) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
```




```
#Choose the optimal lambda
lambda_2 <- lambdas_2[which.min(rmses)]
lambda_2
```

```
## [1] 5
```

Our new lambda is 5. With that we reach a rmse of 0.86477, and we are now, under our target goal.

```
#test and save the result to our table
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie + User + Age of Movie Effect Model",
                                     RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Average rating model	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659320
Movie + User + Age of movie Effects Model	0.8654978
Regularized Movie + User Effect Model	0.8652421
Regularized Movie + User + Age of Movie Effect Model	0.8647718

6.7 Model Movie, user and age of movie effects regularized test with validation

Our last model is within our target goal. We can now try it with the edx data set and put it through the test of the validation. We will use lambda_2 as previously define and define all our "bias" variable with edx. Then we will be able to test our model with the validation set.

```
#Final test using the edx set and validation
#We repeat the previous steps using lambda_2 in our model
mu_hat <- mean(edx$rating)
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_hat)/(n()+lambda_2))
b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu_hat)/(n()+lambda_2))
b_a <-edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(ageOfMovie) %>%
  summarize(b_a= sum(rating - b_i - b_u - mu_hat)/(n()+lambda_2))
predicted_ratings <-
  validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_a, by = "ageOfMovie") %>%
  mutate(pred = mu_hat + b_i + b_u + b_a) %>%
  pull(pred)
```

Our final rmse is 0.86477.

```
rmse_results <- bind_rows(rmse_results, data_frame(method="Final model with validation", RMSE = min(rms
rmse_results %>% knitr::kable()
```

method	RMSE
Average rating model	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659320
Movie + User + Age of movie Effects Model	0.8654978
Regularized Movie + User Effect Model	0.8652421
Regularized Movie + User + Age of Movie Effect Model	0.8647718
Final model with validation	0.8647718

7 Results

As the result of our different model, we can see a clear improvement from our first model to the final one, taking into account user, movie and age of movie effect.

method	RMSE
Average rating model	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659320
Movie + User + Age of movie Effects Model	0.8654978
Regularized Movie + User Effect Model	0.8652421
Regularized Movie + User + Age of Movie Effect Model	0.8647718
Final model with validation	0.8647718

Our lowest value of RMSE is 0.86477

8 Conclusion

We have bulit a machine algorithm able to predict movie ratings from the movielens dataset. Our regularized model, with movie, user and age of the movie effect, is the most efficient to the task. Although, it should be possible to improve further the model, adding other effects that may influence the rating, such as the genre. It will also be possible to add tuning parameters for each of the bias effect, in order to fine tune better the model. We may also use different machine learning models, as matrix factorization to improve the rmse, although it may be heavy on computers that don't have the necessary hardware.