Report on MovieLens Project

Flávia Lemos Xavier

12/08/2019

Overview

This is a report designed to present the project development of a movie recommendation system based on the “MovieLens” dataset. The 10 M version of movielens dataset was recommended in the “PH125.9x Data Science: Capstone” by HarvardX course and it is just a small subset of a much larger dataset with millions of ratings.

Introduction

In 2006, Netflix offered a million US-Dollar price to whomever was able to surpass the performance of their movie recommendation algorithm by at least 10%.This goal was achieved by Team BellKor’s Pragmatic Chaos(https://www.netflixprize.com/community/topic_1537.html (https://www.netflixprize.com/community/topic_1537.html)) in 2009, after a nearly 3-year long contest.

As you may already know, Netflix offers thousands of TV shows available for streaming. It recommends titles for each user. If you use Netflix you may have noticed they create really precises ratings and recommendations, for example, they make recommendations by movie genre. How do they come up with those genres? How to they deal with giving great recommendations to their 100 million-plus subscribers who are already used to getting recommendations from pretty much every platform they use? Machine learning, algorithms and creativity.

Every time you press play and spend some time watching a TV show or a movie, Netflix is collecting data that informs the algorithm and refreshes it. The more you watch the more up to date the algorithm is.

Let’s see what Netflix said about its recommender system and this kind of challenge:

Whenever you access the Netflix service, the recommendation system tries to help you find a show or movie easily. We estimate the likelihood that you will watch a particular title in our catalog based on a number of factors, such as:

a. your interactions with the service (such as what you watched and how you rated other titles),
b. other subscribers with similar tastes and preferences about services and
c. information about titles such as genre, categories, actors, release year, etc.

In addition to knowing what you watched, to better customize recommendations, Netflix also notes:

d. the time you watch,
e. the devices on which you watch Netflix and
f. even how long you watch.

All this data is taken into account by the algorithms.

This information is very useful to develop this project. According to this information, it is possible to explore the database and improve the prediction model through these interaction effects or biases. This strategy can handle the challenge of this type of movie recommendation system that is using a different set of predictors.

Goal

The project aims to create its own recommendation system that will explore the necessary tools that have been demonstrated throughout the course of this series to achieve better accuracy of the predictive model. So, the project will show you how to implement different movie recommendation approaches and evaluate them to see which one has the best performance.

Methodology

The dataset that I'm working with is MovieLens, one of the most common datasets that is available on the internet for building a Recommender System. Following the course recommendation, I will work with the 10M version of the MovieLens dataset to enable computing.

The entire MovieLens 10M Dataset contains 10000054 ratings and 95580 tags applied to 10681 movies by 71567 users of the online movie recommender service. [Here follows the MovieLens 10M Dataset for more details]: (https://grouplens.org/datasets/movielens/10m/ (https://grouplens.org/datasets/movielens/10m/)).

Firstly, I will install some needed packages, load their libraries and download the MovieLens version 10M data. Then I will run the code provided by the course to generate the datasets.

I will also do some exploratory analysis to familiarize with the data in order to prepare better for developing the project.

Secondly, I wil creat the edx partition, spliting it into a training and test/validation set to develop the algorithm.

We start with a model that assumes the same rating for all movies and all users, with all the differences explained by random variation: If μ represents the true rating for all movies, users and genre and ϵ represents independent errors sampled from the same distribution centered at zero, then we will evaluate it gradually by inserting movie (b_i), user (b_u) and genre(b_g) biases into the following equation:

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

Later I will use the regularization to penalize large estimates that come from small sample sizes. That's why I will use cross-validation to choose the best lambda for the model.

In order to build and evaluate the recommendation system I will use the mean square error (RMSE) as the loss function.

Note that because there are thousands of effects (b 's), the lm function will be very slow or cause R to crash, so I will not use linear regression to calculate these effects.

Loading Libraries and Movielens data

```
## Loading required package: tidyverse
```

```
## -- Attaching packages ---------------------------------------- tidyverse 1.2.
1 --
```

```
## v ggplot2 3.2.0     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------- tidyverse_conflicts
() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked _by_ '.GlobalEnv':
##
##     RMSE
```

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

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

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

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

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

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

Exploratory Data Analysis

The GroupLens research lab generated their own database with this subset of date - 10M MovieLens Dataset. According to the course recommendation on the MovieLens Project, I will analyse its subset “edx data” that contains 9,000,055 ratings in the rows and 6 variables in the columns - “userId”, “movieId”, “rating”, “timestamp”, “title”, “genres”, among them there are 10,677 different movies evaluated by more than 69,878 different users.

Let´s see further details about the distribution of these observations.

```
#Firstly, let´s check the dimension (the Number of Rows and Columns) of the Dataset and its class:

dim(edx)
```

```
## [1] 9000055       6
```

```
class(edx)
```

```
## [1] "data.frame"
```

```
#We can see this table in tidy format with thousands of rows and with six observations in the columns:

edx%>% as_tibble()
```

userId <int>	movieId <dbl>	rating <dbl>	timestamp <int>	title <chr>
1	122	5.0	838985046	Boomerang (1992)
1	185	5.0	838983525	Net, The (1995)
1	292	5.0	838983421	Outbreak (1995)
1	316	5.0	838983392	Stargate (1994)
1	329	5.0	838983392	Star Trek: Generations (1994)
1	355	5.0	838984474	Flintstones, The (1994)
1	356	5.0	838983653	Forrest Gump (1994)

userId <int>	movieId <dbl>	rating <dbl>	timestamp <int>	title <chr>
1	362	5.0	838984885	Jungle Book, The (1994)
1	364	5.0	838983707	Lion King, The (1994)
1	370	5.0	838984596	Naked Gun 33 1/3: The Final Insult (1994)

1-10 of 10,000 rows | 1-5 of 6 columns Previous **1** 2 3 4 5 6 … 1000 Next

```
names(edx)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

The edx dataset provided the following information:

- `userId` contains unique user identifier.
- `movieId` contains unique movie identifier.
- `rating` represents user's rating for a movie.
- `timestamp` shows the date and time of user's rating in timestamp-format. Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.
- `title` includes the title as well as the publishing year of the rated movie.
- `genres` includes all movie related genres, seperated with the symbol "|".

It´s important to notice that each line of this data represents one rating of one movie by one user.

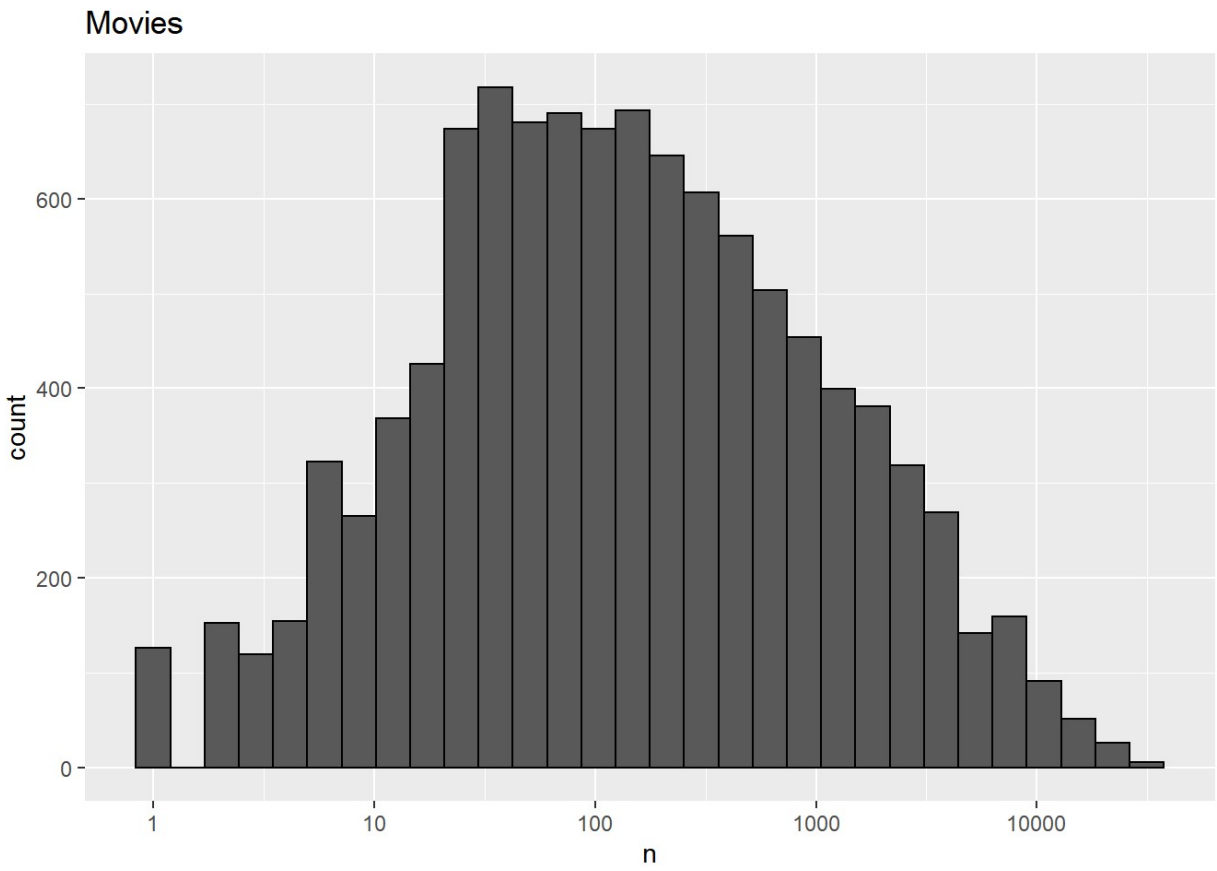
We can see the number of unique users that provided ratings and how many unique movies were rated:

n_users <int>	n_movies <int>
69878	10677

1 row

Let's look at some of the general properties of the data to better understand the challenges.

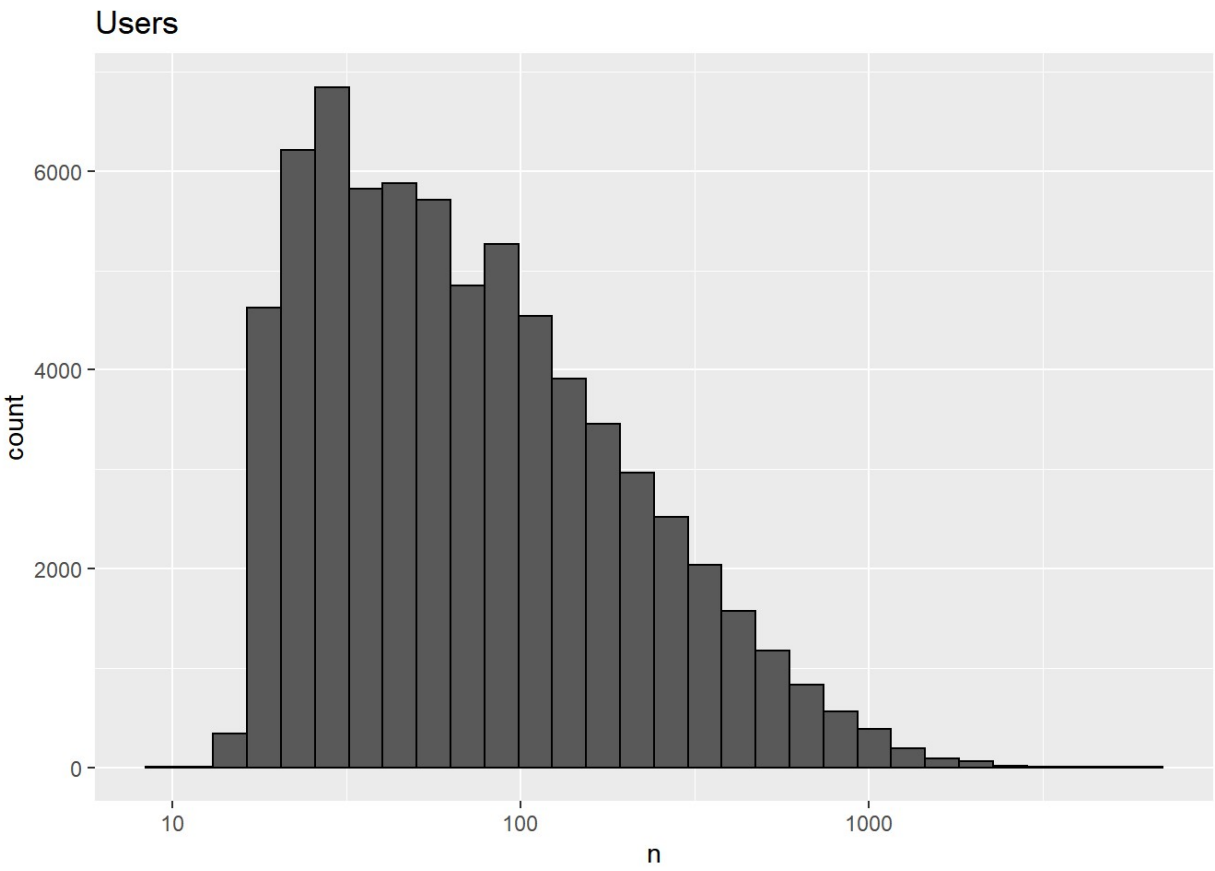
The first thing we notice is that some movies get rated more than others. Here is the distribution:



movieId	title
<dbl>	<chr>
296	Pulp Fiction (1994)
356	Forrest Gump (1994)
593	Silence of the Lambs, The (1991)
480	Jurassic Park (1993)
318	Shawshank Redemption, The (1994)
110	Braveheart (1995)
457	Fugitive, The (1993)
589	Terminator 2: Judgment Day (1991)
260	Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
150	Apollo 13 (1995)

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 ... 1000 Next

My second observation is that some users are more active than others at rating movies:



Ratings are made on a 5-star scale, with half-star increments. Note that no movies have a rating of 0 and movies are rated from 0.5 to 5.0 in 0.5 increments. Have a look at these analyzes:

```
edx %>% filter(rating == 0) %>% tally()
```

n
<int>
0

1 row

Here follows the distribution of Movie Ratings:

```
#summary(edx$rating)

summary(edx$rating)
```

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

```
edx %>% group_by(rating) %>% summarize(count = n())
```

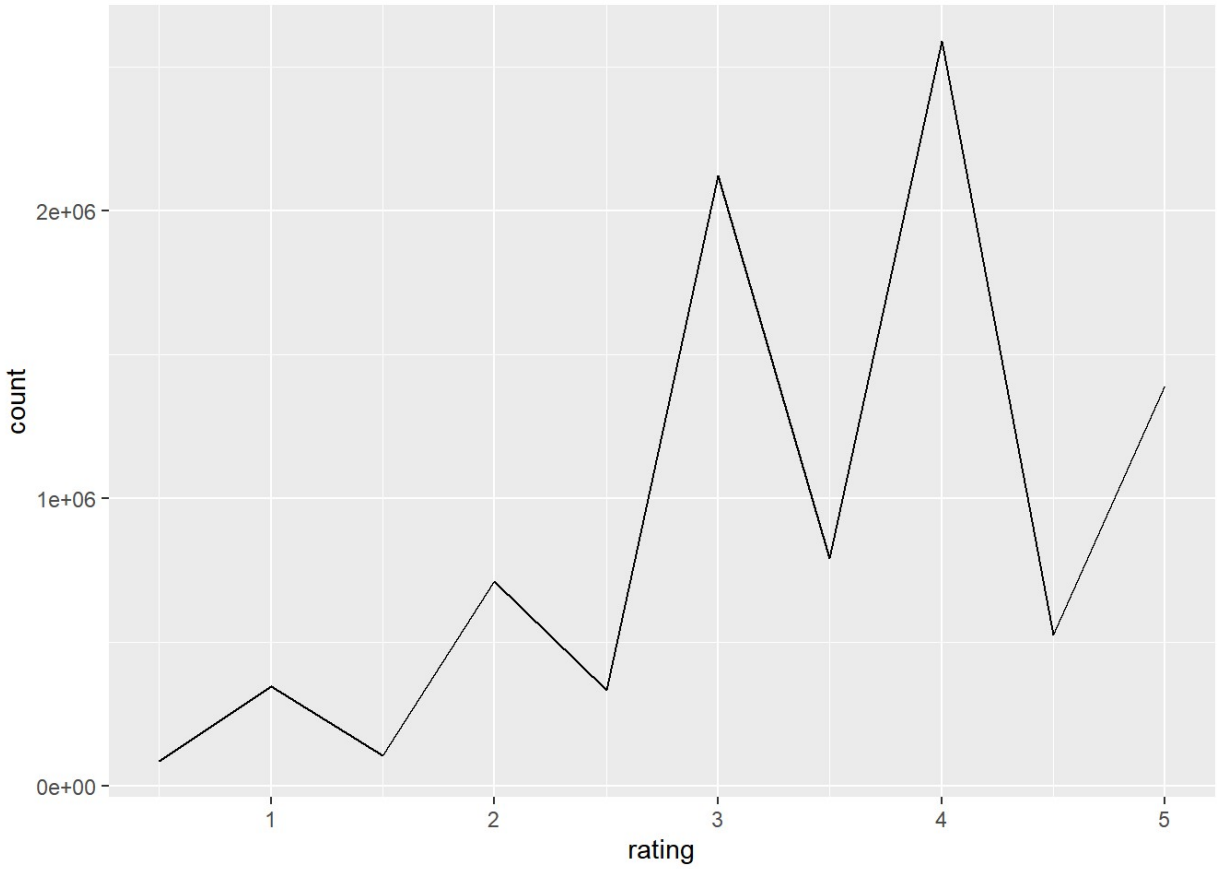
rating	count
<dbl>	<int>

rating <dbl>	count <int>
0.5	85374
1.0	345679
1.5	106426
2.0	711422
2.5	333010
3.0	2121240
3.5	791624
4.0	2588430
4.5	526736
5.0	1390114

1-10 of 10 rows

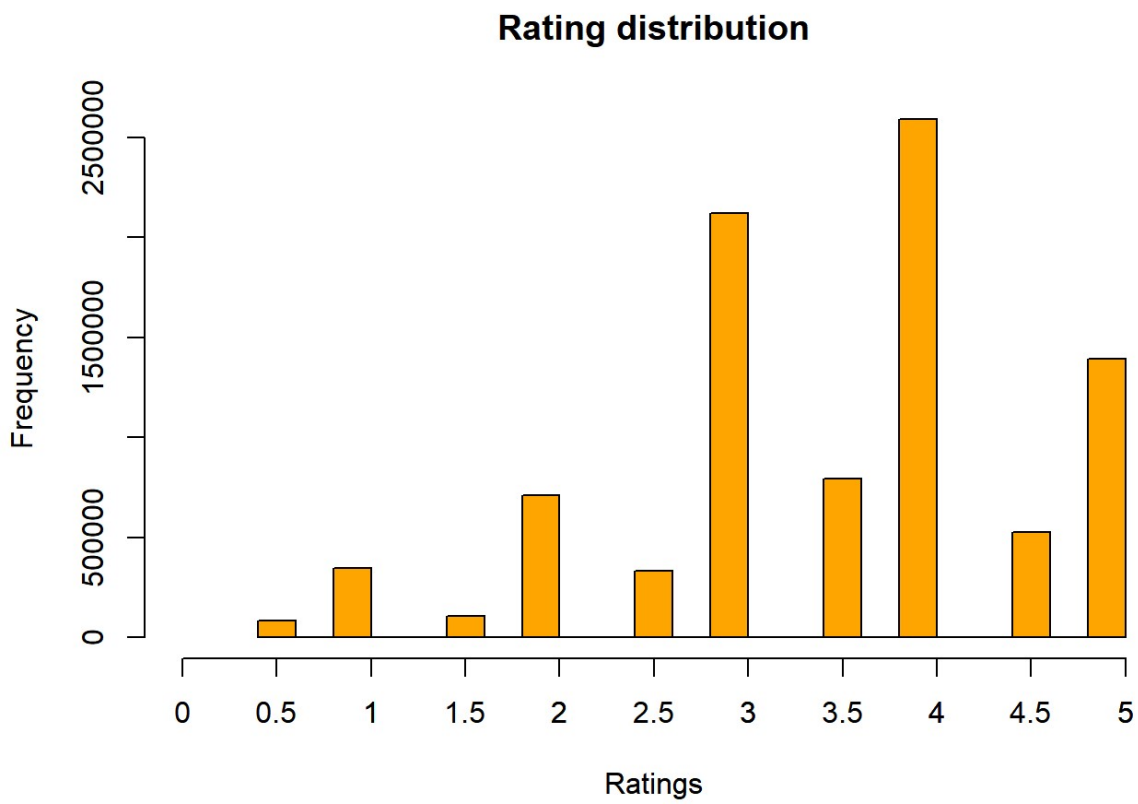
```
#Visually, the Ratings distribution can be seen here:

edx %>%
  group_by(rating) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = rating, y = count)) +
  geom_line()
```



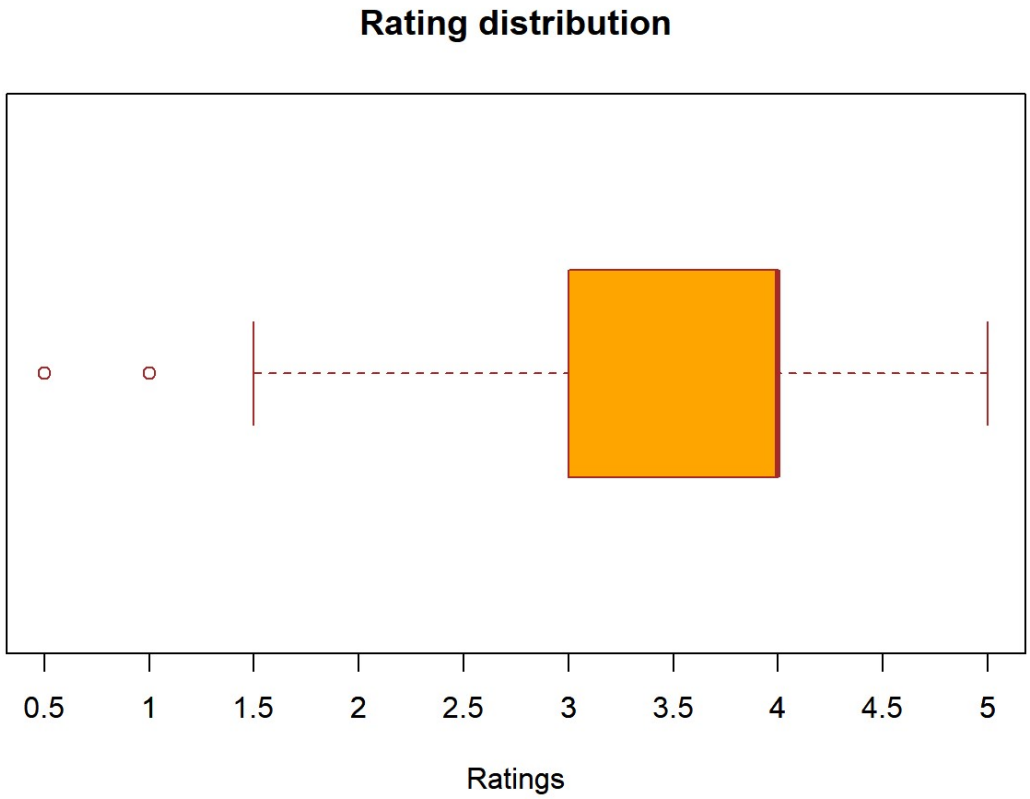

```
# This discrete rating distribution is more understandable in this histogram:

hist(edx$rating,
     main="Rating distribution",
     xlab="Ratings",
     xlim=c(0,5),
     col="orange",
     freq=TRUE
)
axis(side=1, at=sort(unique(edx$rating)), labels=sort(unique(edx$rating)))
```



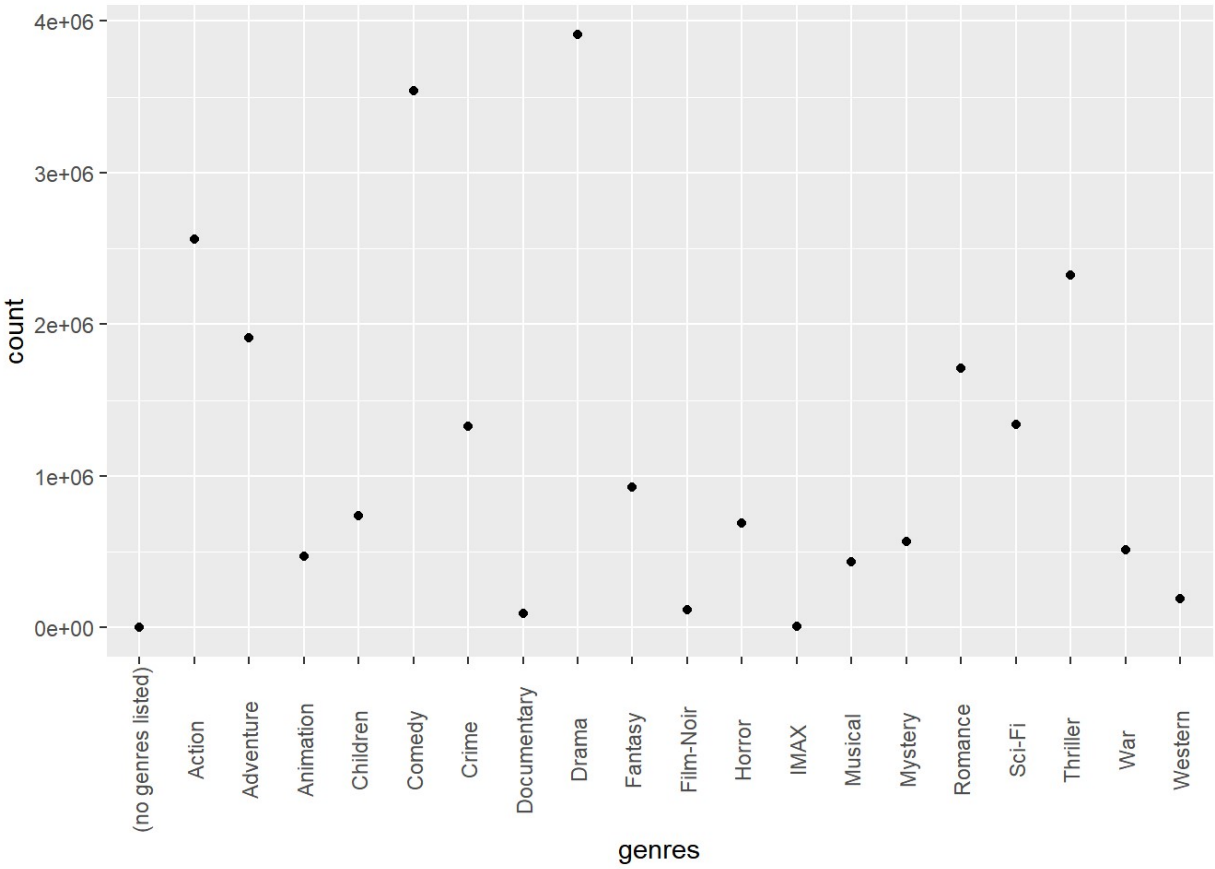

```
# Here it is clear where the highest ratings are concentrated, between 3 and 4 star
s.

boxplot(edx$rating,
        main = "Rating distribution",
        xlab = "Ratings",
        ylab = "",
        col = "orange",
        border = "brown",
        horizontal = TRUE,
        notch = FALSE
)
axis(side=1, at=sort(unique(edx$rating)), labels=sort(unique(edx$rating)))
```



Let’s see the gender distribution analysis. We note in the figures above that there are different distributions per movie and per user, this will be considered in the model analysis. Is there any gender effect? We will see that there is a gender effect and I will also consider this effect during the development of the model.

Let´s plot the popular genres:



Create Test and Train Data Sets

Taking into account the validation$rating can be used only to calculate RMSE at the end, I will split edx into a training and test/validation set to develop the model, including to use in the cross validation.

Following previous courses recommendation, the data set `train_set` will contain 80% of the available data. This data set will be used to train any model. To test and evaluate these models, the data set `test` with about 20% of the available data will be used.

```
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 <- edx[test_index,]
```

To make sure I will not include users and movies in the test set that do not appear in the training set, I will remove these entries using the semi_join function:

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

I will add these removed movieIds back into the train_set to predict against validation later.

```
removed <- test %>%
  anti_join(train_set, by = "movieId")

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

Baseline model

Let's start by building the simplest possible recommendation system: we predict the same rating for all movies regardless of user. What number should this prediction be? We can use a model based approach to answer this. A model that assumes the same rating for all movies and users with all the differences explained by random variation would look like this:

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

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

```
## [1] 3.512478
```

```
#If we predict all unknown ratings with mu_hat we obtain the following RMSE:
naive_rmse <- RMSE(mu_hat, test_set$rating)
naive_rmse
```

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

```
#As we go along, we will be comparing different approaches. Let's start by creating
a results table with this naive approach:

rmse_results <- data_frame(method = "Just the average", RMSE = naive_rmse)
```

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

```
rmse_results
```

method	RMSE
<chr>	<dbl>
Just the average	1.059904

1 row

From looking at the distribution of ratings, we can visualize that this is the standard deviation of that distribution. We got a RMSE over 1. It´s not good yet and it will cause a prediction error due to the low accuracy of this first model.

For instance, a participating team of the Netflix grand prize, had to get an RMSE of about 0.857 and my challenge is achieve less than 0.8649 in this course. So I can definitely do better. Let´s do it.

Modeling movie effects or movie bias (b_i)

We know from exploratory data analysis above that some movies are just generally rated higher than others, that's why I call our baseline model as a naive approach. It's recommended to improve the previous model by adding some biases as those mentioned in the introduction of this project, including in the equation of the algorithm the interaction between each movie, each user and each genre, beyond the ratings average, such as the movie bias, the user bias, the genre bias, etc.

In this line, the term "b_i" represents average ranking for movie i:

I will work with the development of biases one by one for didactic purposes to apply the knowledge of the course.

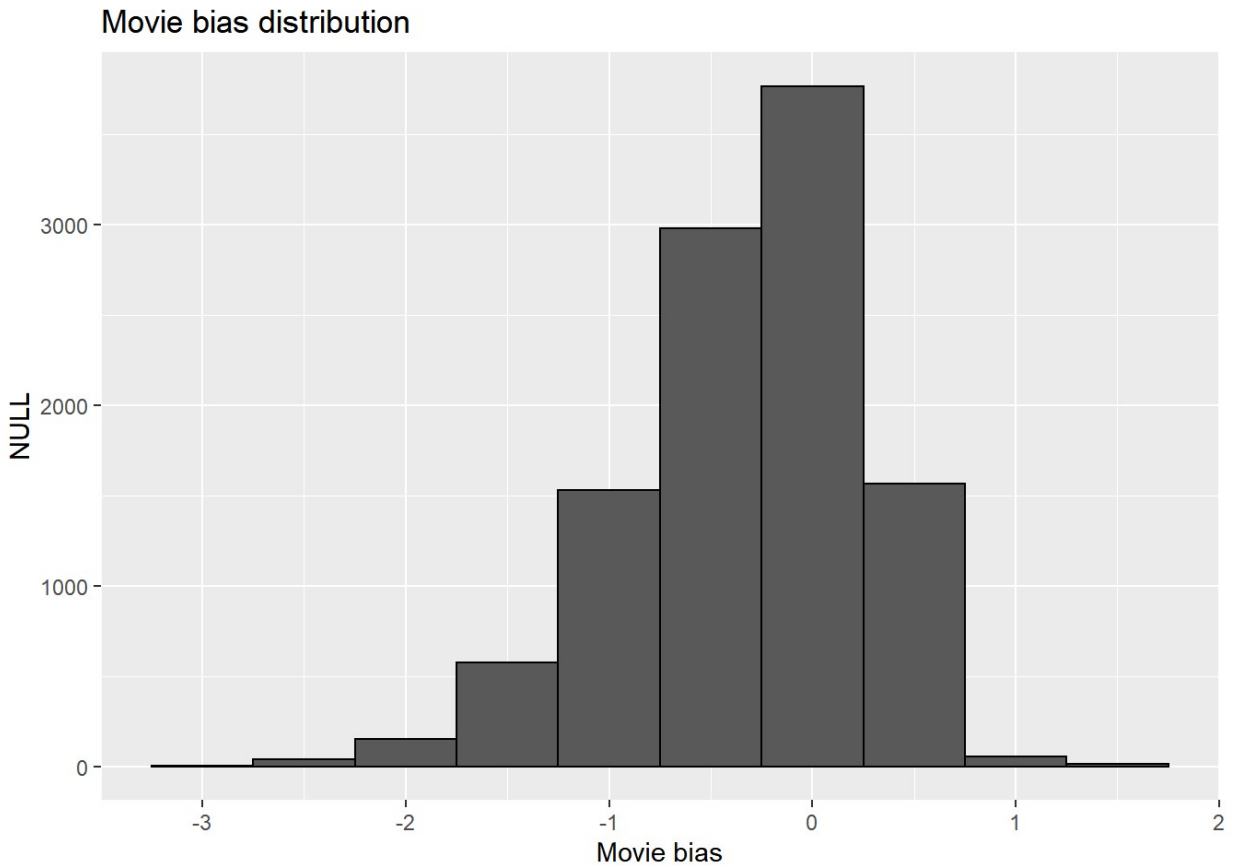
The model then becomes:

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

I will again use least squares to measure the bi_hat in the following way:

```
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i_hat = mean(rating - mu_hat))
```

Now that I have the bias of the movie, let's see better the movie ratings distribution that is much more different from the previous model where I assumed average:



It’s time to predict movie ratings based on the new model with the movie effect:

```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  mutate(pred = mu_hat + b_i_hat) %>%
  .$pred
```

Let’s see how much our prediction improves once we use the movie effect:

method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429

Reducing the RMSE, including the movie effect proved that I am on the right path of developing the predictive model for the required recommendation system. To further improve upon our model, let´s include the “User-effect”.

Modeling also user effects or user bias (b_u)

We saw there is substantial variability across users as well. Some users rate movies very well, others not so much.

Here follows a further improvement to the model:

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

where b_u is a user-specific effect.

Let’s measure the effect of this variability of ratings given by users.

Instead of fitting the model into the lm algorithm, I will continue the strategy of approximation with the average that was taught in the course, to evaluate the accuracy through the variability of the RMSE.

```
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u_hat = mean(rating - mu_hat - b_i_hat))
```

It’s time to predict movie ratings based on the model with the user effect:

```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(pred = mu_hat + b_i_hat + b_u_hat) %>%
  .$pred
```

And see how much our prediction improves once we use the movie effect:

method	RMSE

method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659319

Including the user-effect b_u in our rating predictions further reduced the RMSE.

The average star RMSE still looks high to me. Let´s develop the model with genre effect and see what happens.

Modeling also user genre effect or genre bias (b_g)

```
genre_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by = 'userId') %>%
  group_by(genres) %>%
  summarize(b_g_hat = sum(rating - mu_hat - b_i_hat - b_u_hat/n()))
```

```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = "movieId")%>%
  left_join(user_avgs, by = "userId")%>%
  left_join(genre_avgs, by = "genres")%>%
  mutate(pred = mu_hat + b_i_hat + b_u_hat+b_g_hat) %>%
  .$pred
```

method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659319
Movie + User+ Genre Effects Model	0.8655941

When considering the effects of movie, user and genre, I reduced even more the RMSE. Can i improve? I believe so.

Let’s explore the rating extremes of this database further to decide some new approach to reduce the RMSE.

Here are the top 10 best movies according to our prediction taking into account the movie bias:

title	b_i_hat	n
Hellhounds on My Trail (1999)	1.487522	1
Who’s Singin’ Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.487522	3

title	b_i_hat	n
Satan’s Tango (SÃ¡tÃ¡ntangÃ³) (1994)	1.487522	2
Shadows of Forgotten Ancestors (1964)	1.487522	1
Money (Argent, L’) (1983)	1.487522	1
Fighting Elegy (Kenka erejii) (1966)	1.487522	1
Sun Alley (Sonnenallee) (1999)	1.487522	1
Aerial, The (La Antena) (2007)	1.487522	1
Blue Light, The (Das Blaue Licht) (1932)	1.487522	1
More (1998)	1.404189	6

Here are the 10 worst movies according to our predictions:

title	b_i_hat	n
Besotted (2001)	-3.012478	1
Hi-Line, The (1999)	-3.012478	1
Accused (Anklaget) (2005)	-3.012478	1
Confessions of a Superhero (2007)	-3.012478	1
War of the Worlds 2: The Next Wave (2008)	-3.012478	2
SuperBabies: Baby Geniuses 2 (2004)	-2.749978	40
From Justin to Kelly (2003)	-2.667240	168
Legion of the Dead (2000)	-2.637478	4
Disaster Movie (2008)	-2.637478	28
Hip Hop Witch, Da (2000)	-2.603387	11

As we can see some of the best and worst movies predicted were rated sparsely. The supposed “best” and “worst” movies were rated by very few users, in most cases just 1. The same holds true for the user-effect b_u , in those cases where users only rated a very small number of movies.

These are noisy estimates that we should not trust, especially when it comes to prediction. Large errors can increase our RMSE, so I will be conservative and adjust these distortions.

Regularization permits us to penalize large estimates that are formed using small sample sizes. It has commonalities with the Bayesian approach that shrunk predictions.

Regularization

In this sense, I will determine the value of `Lambda` , that is a tuning parameter that minimizes RMSE, employing cross-validation. This shrinks the b_i , b_u and b_g in case of small number of ratings. So, let’s do it.

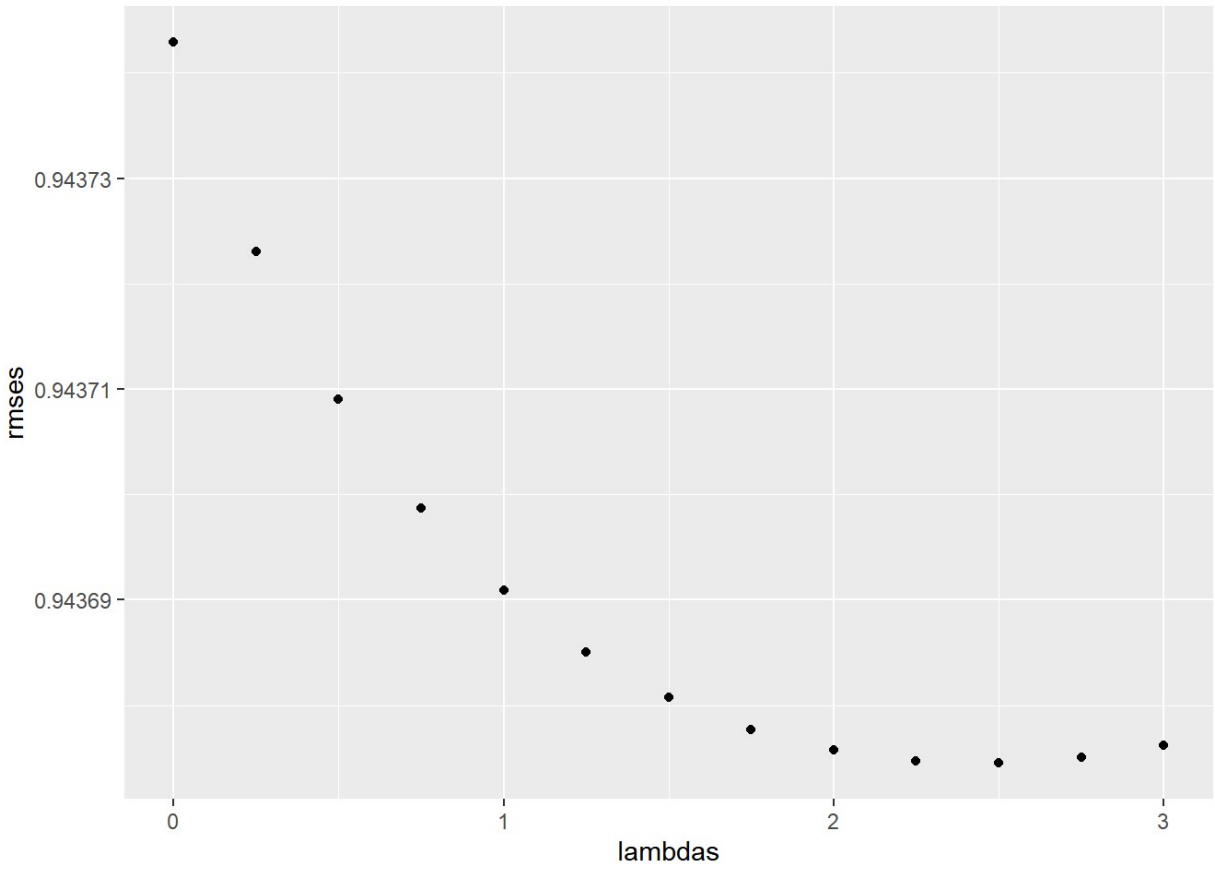
Firstly, I will chose the penalty terms.

Regularization of the Movie Effect Model

Here I use cross-validation to pick a λ (lambda) which minimum value of RMSE. After a few tries, we reduced the evaluation time by selecting a sequence closer to the best lambda found by the function.

```
lambdas <- seq(0, 3, 0.25)
mu <- mean(train_set$rating)
just_the_sum <- train_set %>%
group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())

rmses <- sapply(lambdas, function(l){
  predicted_ratings <- test_set %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
```




```
## [1] 2.5
```

```
mu <- mean(train_set$rating)
movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+optm_lambda))
```

```
predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  .$pred
```

I will now compare the RMSE of this model with the Movie Effect Model, as we are reconsidering the regularization of each of the effects in order of presentation in this project.

method	**RMSE**
Just the average	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659319
Movie + User+ Genre Effects Model	0.8655941
Regularized Movie Effect Model	0.9436745

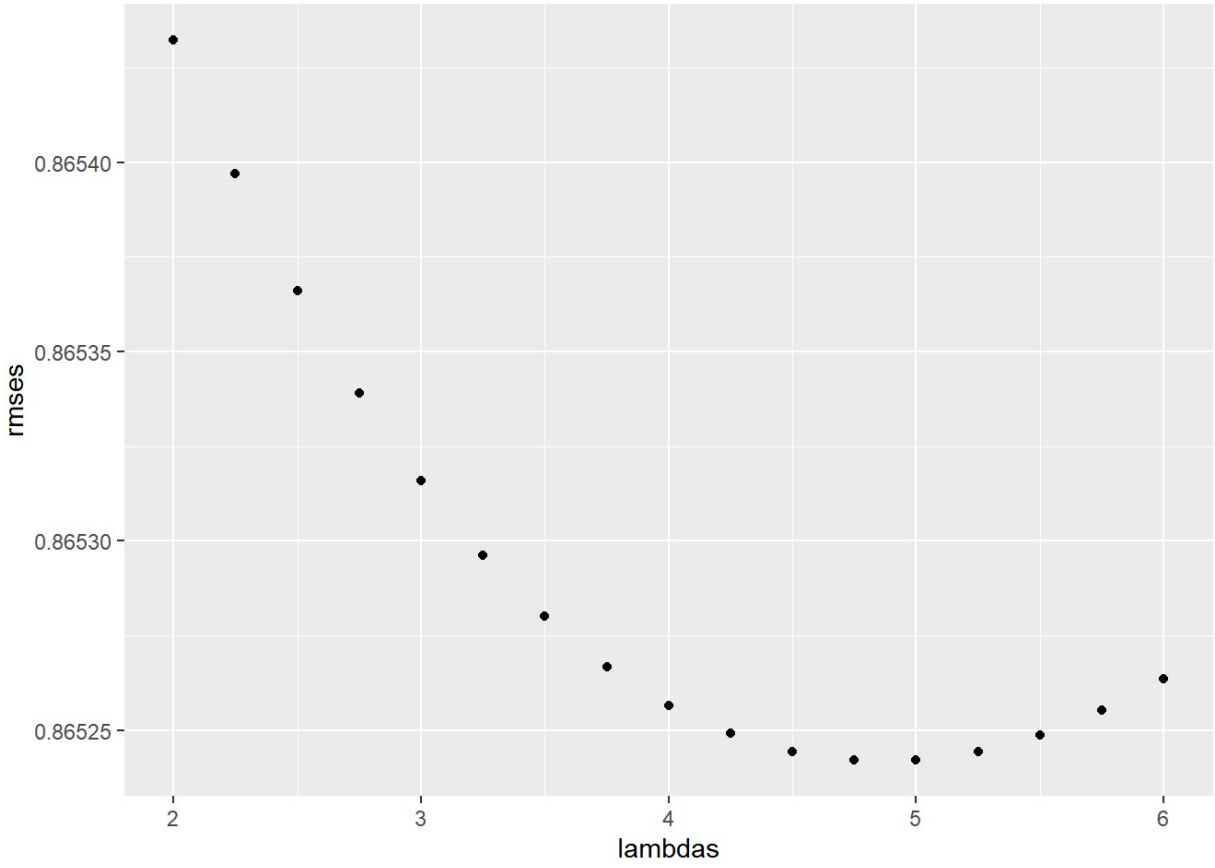
We reduced a little, right? So I will continue with the regularization of the next models. The cross validation method is the same.

Regularization of the Movie and User Effects Model

After some time trying to find the best lambda between 0 and 10, we selected the test between parameters 2 and 6 because it identified the function and its best lambda in the following graph. This will reduce code analysis time.

```
lambdas <- seq(2, 6, 0.25)

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))
})
```




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

And what is the smallest RMSE for this model?

method	RMSE
Just the average	1.0599043

method	RMSE
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659319
Movie + User+ Genre Effects Model	0.8655941
Regularized Movie Effect Model	0.9436745
Regularized Movie + User Effect Model	0.8652421

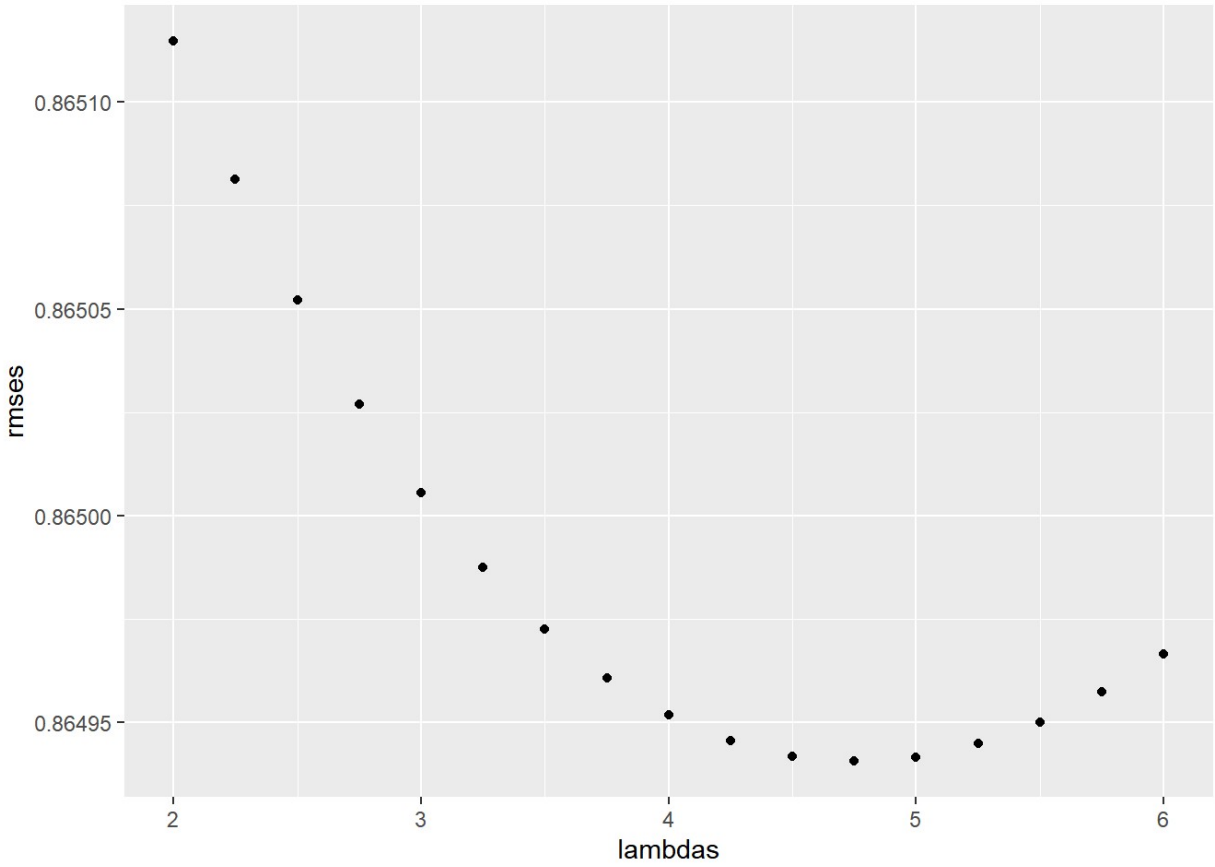
I will now compare the RMSE of this model with the Movie +User Effect Model. We keep reducing, right? So I will continue with the regularization of the Movie + User + Genre Effect Model. The cross validation method is the same.

Regularization of the Movie + User + Genre Effect Model

After some time trying I got the best lambda between 2 and 6, as you can see below.

```
lambdas <- seq(2, 6, 0.25)

rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  b_g <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u- mu)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
```



And what is the smallest RMSE for this model?

```
## [1] 0.8649407
```

method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659319
Movie + User+ Genre Effects Model	0.8655941
Regularized Movie Effect Model	0.9436745
Regularized Movie + User Effect Model	0.8652421
Regularized Movie + User + Genre Effect Model	0.8649407

Finally, I will compare the RMSE of this regularized model with Movie + User + Genre Effect Model with no regularization to see what happens.

This certainly improved our predictions.

Modeling results: Final RMSE = 0.8644

Let’s see what the final value of RMSE is by applying the Regularized Movie + User + Genre Effect Model.

Notice that now we can use the validation set to predict movie ratings and evaluated our final Model.

See what happens:

```
mu <- mean(edx$rating)
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+l))
b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+l))
b_g <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(genres) %>%
  summarize(b_g = 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_g, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  .$pred

# Calculate the predicted values for the validation data set

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

# Final RMSE

Final_RMSE <- RMSE(predicted_ratings, validation$rating)
Final_RMSE
```

```
## [1] 0.8644514
```

As we can see above, the final RMSE is better than the target of 0.8649 and lead to the best classification in course's results.

The used method is more feasible for a small processor and RAM even with a big amount of available data lines, although it is soo laborious. In a smaller database the analysis would probably be different and the use of the complex machine learning algorhythm such as Nearest Neighbor algorithm would favor the building of the model.

Conclusion

Faced with the HarvardX: PH125.9x Data Science: Capstone Course challenge presented, I used the 10M version of the MovieLens dataset to enable computing.

After preparing the database, I runned the code provided by the course to generate the datasets and did some exploratory analysis for developing the project.

Then I created the edx partition, spliting it into a training and test set to develop the algorithm.

I started with a model that assumes the same rating for all movies, all users and all genres, with all the differences explained by random variation. Then I evaluated the model building gradually by inserting movie (b_i), user (b_u) and genre(b_g) biases into the following equation:

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

Finally I used the regularization to penalize large estimates that came from small sample sizes. That's why I used cross-validation to choose the best lambda for the final model, regularizing effect by effect.

In order to build and evaluate the recommendation system I used the mean square error (RMSE) as the loss function and with a best lambda, I achieved a project goal with a final RMSE of 0.8644.

References

HarvardX Professional Certificate Program in Data Science, taught by Rafael Irizarry.

Emmanuel Paradi, **R for Beginners**. Institut des Sciences de l'Evolution ´ Universit´e Montpellier II F-34095 Montpellier c´edex 05 , France, 2002.

John M. Chambers, **Programming with Data: A Guide to the S Language**. Springer-Verlag New York, 1998.

Netflix, available in Aug 12, 2019 (https://help.netflix.com/en/node/100639 (https://help.netflix.com/en/node/100639))

James Le, **The 4 Recommendation Engines That Can Predict Your Movie Taste**, Medium, Apr 22, 2018, available in: (https://medium.com/@james_aka_yale/the-4-recommendation-engines-that-can-predict-your-movie-tastes-bbec857b8223 (https://medium.com/@james_aka_yale/the-4-recommendation-engines-that-can-predict-your-movie-tastes-bbec857b8223))