Data Science: Capstone

Horacio Báez

27/5/2020

Preface

This report is part of the project Data Science: Capstone of the HarvardX's Data Science Profesional Certificate Program.

Introduction

As an economics students, I know how important are e-commerce and streamings services in today's tech industry. Since these services rely on virtual interactions, instead of employee to client, recommendation systems play a big role in sales. A wrong recommendation has the potential to change the loyalty of a client to a given enterprise.

In 2006, Netflix started a challenge: they offered a one million dollar prize for the person that or group that could improve their recomendation system by at least 10%. Nowadays, these competitions are very common and there a lot of web pages that hold these challenges regularly.

Usually, recomendation systems are based on rating that goes from 1 to 5 grades of satisfaction (most commonly, stars). These recomendation system can be oriented to a wide variety of products: music listened, videos watcher, items purchased and so on.

The primary goal of a recommendation system is to generate better guidance so that customer can find elements based on their preferences and needs. In this project, we apply techniques learned trough the HarvardX's Data Science Professional Certificate to build a recommendation system with the Movielens dataset.

Movielens dataset

GroupLens is a research lab in the University of Minnesota that has collected and made available rating data for movies in the MovieLens web site.

The complete MovieLens dataset6 consists of 27 million ratings of 58,000 movies by 280,000 users. The research presented in this project is based on a subset of this dataset with 10 million ratings on 10,000 movies by 72,000 users.

Model Evaluation

To evaluate the performance of a machine learning algorithm, we need to compare the predicted outcome wiht the actual outcome. While there are many statistics techniques that can be used for this purpose, we will focus on the Root Mean Squared Error (RMSE).

Root Mean Squared Error (RMSE)

Is a typical metric to evaluate machine learning algorithms. It is given by this formula:

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

where N is the number of observations (ratings), $y_{u,i}$ is the rating for movie i by user u and $\hat{y}_{u,i}$ is the *prediction* of movie i by user u.

Process and workflow

The main steps in any data science project consists of:

1.Data preparation: dowloading the dataset and prepare it to be processed and analysed.

2.Data exploration and visualization: explore the data to understand the relantionship between variables.

3.Data Cleaning: the dataset may contain data that is not relevant that can be removed.

4.Data analysis and modeling: create a model using the insights obtained in the step 2. Then, test the model.

5.Communicate: reporting the results.

First, we download the dataset from Movielens and split it into two datasets for training and validation. The training subset is called `edx` and the validation one is called `validation`. Then we split again the `edx` dataset into two subsets: `train_set` and `test_set`. When the RMSE reach our target, with the model trained with the `edx` subset, with use the `validation` dataset for final validation of the model.

In the next steps we are going to analyze the data, find insights, create charts and report statistics summaries that will help later on with the train of our model.

Creating a recommendation systems involve several steps of try-and-error work. First we are going to predict just the average rating for each movie. Then we are going to add the movie effects. Later on, we are going to add a user effect as well. Finally, to reach our RMSE target, we are going to apply a regularization parameter that penalizes samples with few ratings.

Method and analysis

Data preparation

In this section, we will download the dataset and split it into two subsets: `edx` for training (with 90% of the dataset) and `validation` for final validation (with 10% of the dataset). `edx` will be split again into `train_set` (with 90% of the `edx` subset) and `test_set` (with 10% of the `edx` subset.

```
################################
# Create edx set, validation set

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")
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
if(!require(ggthemes)) install.packages("ggthemes", repos = "http://cran.us.r-project.org")
#Data dowloading################################

# Note: this process could take a couple of minutes
# MovieLens 10M dataset:
```

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

Data Exploration

Now, that we have downloaded the entire dataset an created two subsets (`edx` and `validation`) for training and testing the model, we proceed to explore the `edx` dataset before splitting it again into two subsets as previous mentioned.

```r
#First, let us explore the estructure of the dataset
glimpse(edx)
```

```
## Rows: 9,000,055
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 42...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ...
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83...
## $ title     <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
## $ genres    <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
```

We can see that we have 6 variables and over 9 million observations. We discover as well what those variables are and their class:

movieId integer

userId integer

rating numeric

timestamp numeric

title character

genres character

The function `head()` give us a better perspective of the first fiva rows in the `edx` dataset.

```
#A better look of the first lines of the dataset
head(edx)
```

```
##    userId movieId rating timestamp title genres
## 1:      1     122      5 838985046  <NA>   <NA>
## 2:      1     185      5 838983525  <NA>   <NA>
## 3:      1     292      5 838983421  <NA>   <NA>
## 4:      1     316      5 838983392  <NA>   <NA>
## 5:      1     329      5 838983392  <NA>   <NA>
## 6:      1     355      5 838984474  <NA>   <NA>
```

Note that the `rating` column is the one for which we are going to build a prediction model. The `userId` and `movieId` columns give us the code that belongs to each user and movie respectively. The `title` column give us the name of each movie and the `genres` column its genre. The `timestamp` give us the data of each movie measured in seconds since January 1st, 1970.

The next steps will give us more information about the variables.

Movies

We can see the number of uniques movies in the dataset with the following code:

```
#Since is a movie ratings dataset, let us explore the number of movies
length(unique(edx$movieId))
```

```
## [1] 10677
```

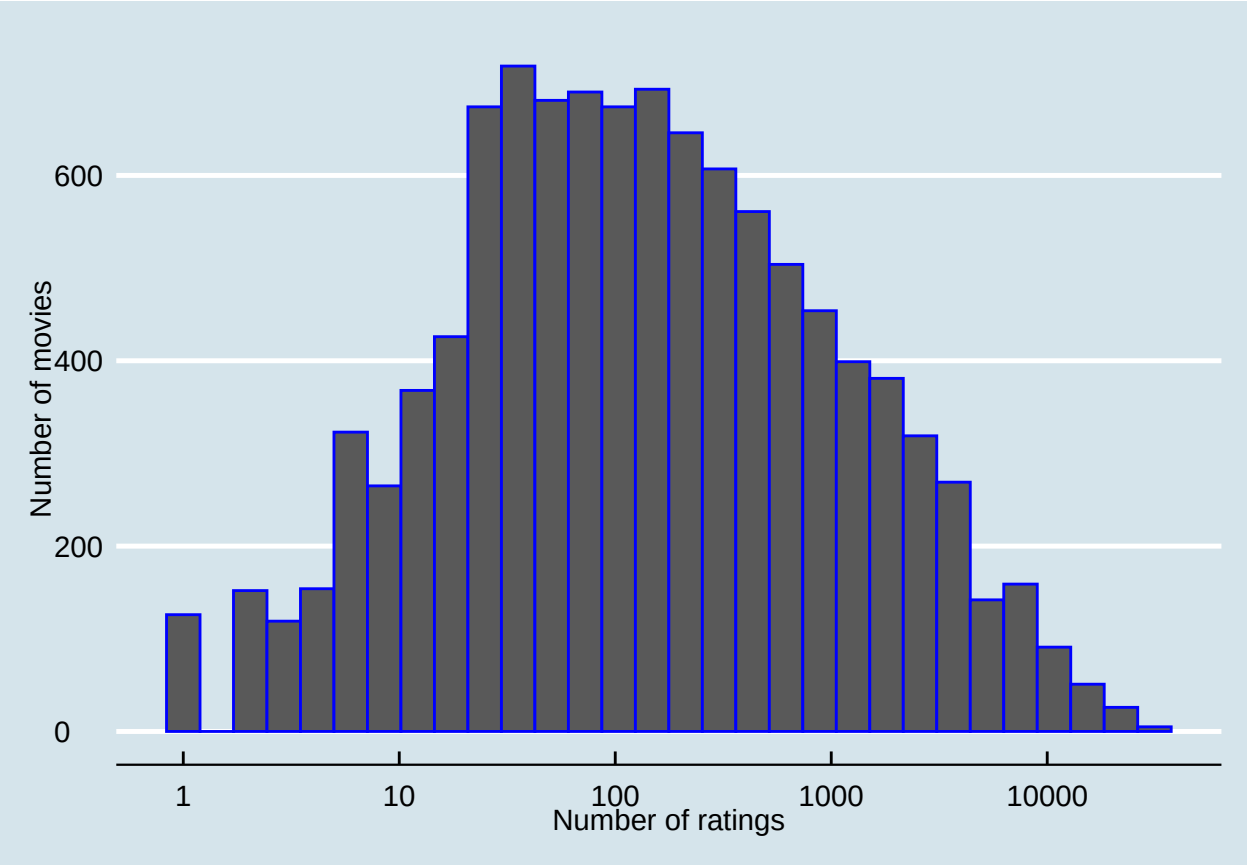
We see that there are 10,677 unique movies. With some intuition we can expect that not every movie will have the same number of ratings. Let us check that.

```
#Now, let us explore the number of ratings per movie)
edx %>%
  group_by(movieId, title) %>%
  summarize(count = n()) %>%
  arrange(desc(count)) %>%
  head()
```

```
## # A tibble: 6 x 3
## # Groups:   movieId [6]
##   movieId title count
##     <dbl> <chr> <int>
## 1     296 <NA>  31362
## 2     356 <NA>  31079
## 3     593 <NA>  30382
## 4     480 <NA>  29360
## 5     318 <NA>  28015
## 6     110 <NA>  26212
```

Just a table may not be enough to appreciate the distribution of ratings per movie.

```
#Now, we can check the distribution of movies vs the number of ratings they have
#We know from intuition that not every movie has the same number of ratings
edx %>% group_by(movieId)%>%
  summarize(count=n())%>%
  ggplot(aes(count))+
  geom_histogram(color="blue")+
  scale_x_log10()+
  xlab("Number of ratings")+
  ylab("Number of movies")+
  theme_economist()
```



The previous plot show us that the distribution of ratings is approximately normal.

Users

Now, we can do the same proccedure to explore information about users.

```
#Now, let us explore the number of users
length(unique(edx$userId))
```

```
## [1] 69878
```

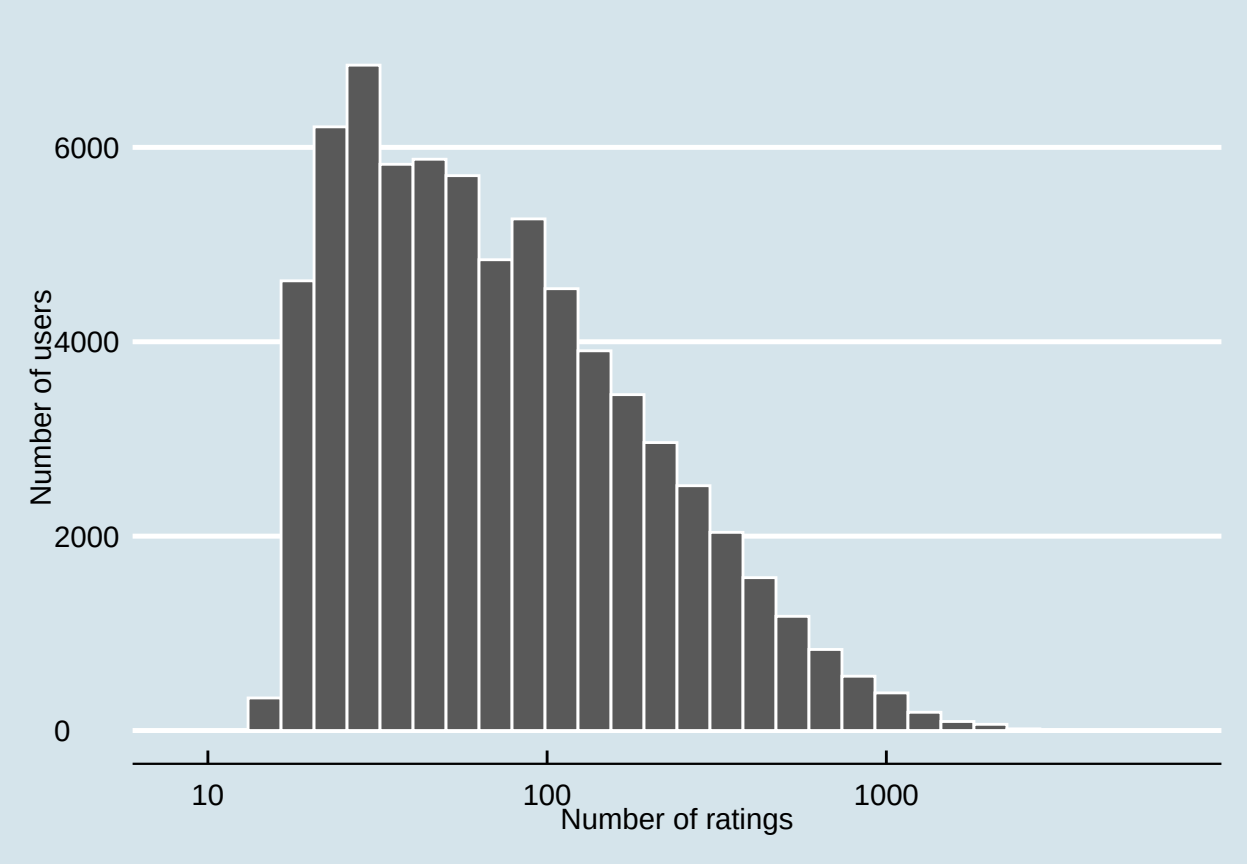
We can see that there are 69878 unique users in the dataset. Let us explore the distribution of movies rated per user.

```
##Now, let us explore the number of movies rated per user
edx %>%
  group_by(userId) %>%
  summarize(count = n()) %>%
  arrange(desc(count))%>%
  head()
```

```
## # A tibble: 6 x 2
##   userId count
##    <int> <int>
## 1  59269  6616
## 2  67385  6360
## 3  14463  4648
## 4  68259  4036
## 5  27468  4023
## 6  19635  3771
```

Again, this may not be enough, let us plot it.

```
#Now we can do the same for users, we can expect that not many users would rate a lot of movies (e.g. >
edx%>%
  group_by(userId)%>%
  summarize(count=n())%>%
  ggplot(aes(count))+
  geom_histogram(color="white")+
  scale_x_log10()+
  xlab("Number of ratings")+
  ylab("Number of users")+
  theme_economist()
```



We can appreciate that the distribution is left skewed because most users have rated less than 100 movies and very few of them have rated more than 1,000.

Genres

Now, let us explore a little the genres variable by first getting the number of categories in this variable.

```
#Now, let us check the number of genres availables
length(unique(edx$genres))
```

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

Now let us explore a little bit the way the number of movies distribute among the categories.

```
#Now, let us check the number of movies per genre
edx %>%
  group_by(genres) %>%
  summarize(count = n()) %>%
  arrange(desc(count))%>%
  head()
```

```
## # A tibble: 1 x 2
##   genres   count
##   <chr>    <int>
## 1 <NA>   9000055
```

This table show us than many movies are classified in more than one genre.

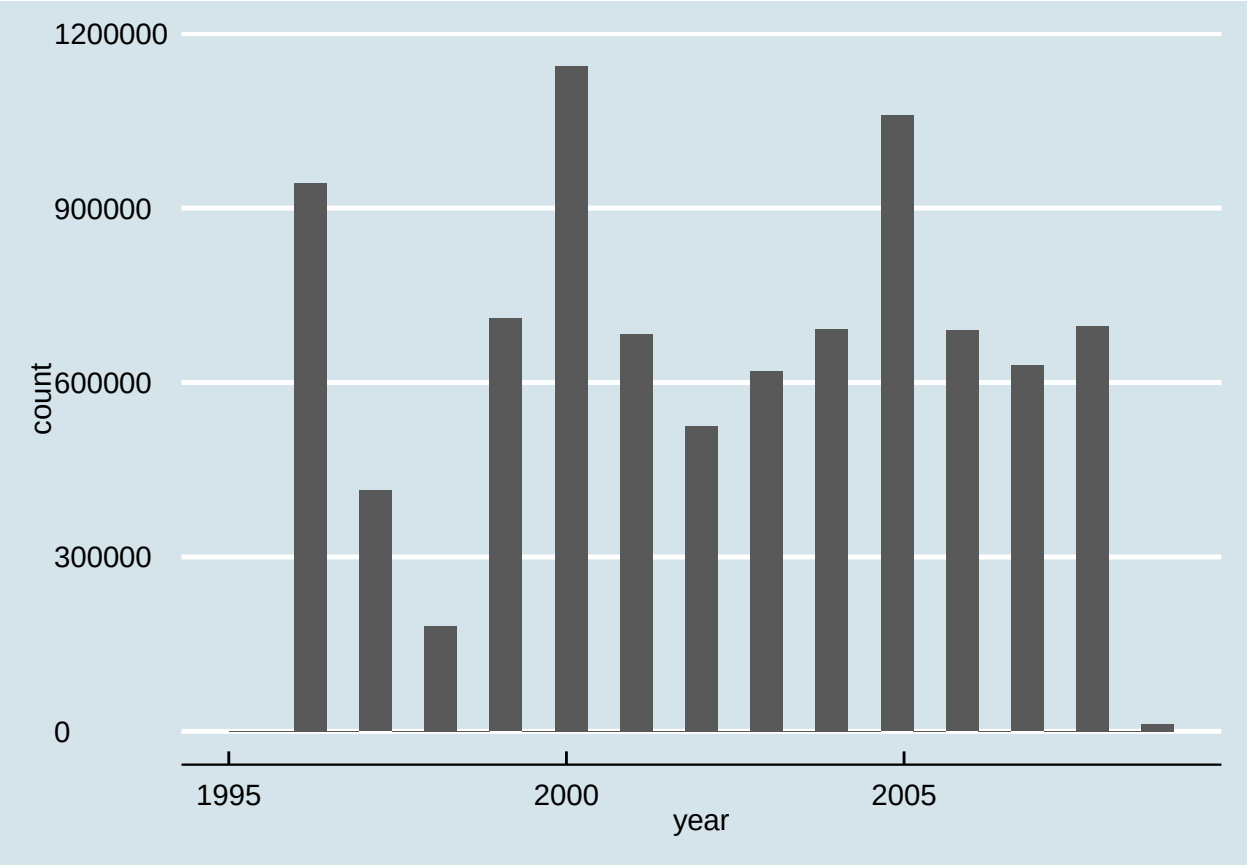
Dates

The rating period was collected over almost 14 years.

```
#Now, we are going to explore the dates of the movies to see a 14 years period approximately
tibble('Initial Date'=date(as_datetime(min(edx$timestamp),origin="1970-01-01")),
       "Final Date"=date(as_datetime(max(edx$timestamp),origin="1970-01-01")))%>%
  mutate(Period=duration(max(edx$timestamp)-min(edx$timestamp)))
```

```
## # A tibble: 1 x 3
##   `Initial Date` `Final Date` Period
##   <date>         <date>       <Duration>
## 1 1995-01-09     2009-01-05   441479727s (~13.99 years)
```

```
#Let us explore now the distribucion of ratings vs years
edx%>%
  mutate(year=year(as_datetime(timestamp,origin="1970-01-01")))%>%
  ggplot(aes(x=year))+
  geom_histogram()+
  theme_economist()
```



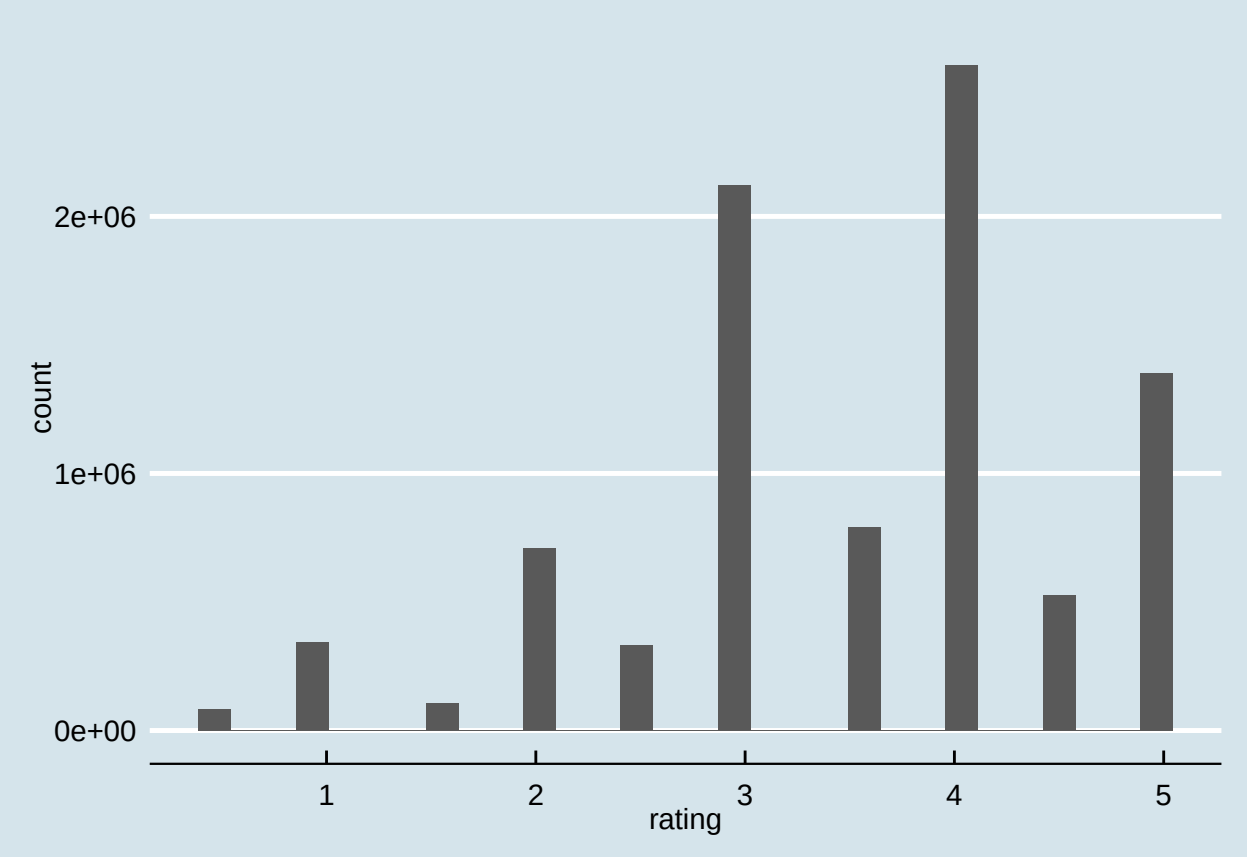
Ratings

Now at last, let us explore the rating variable.

```r
#Going on, let us explore how the rating califications is
edx %>%
  group_by(rating) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
```

```
## # A tibble: 10 x 2
##    rating   count
##     <dbl>   <int>
##  1    4   2588430
##  2    3   2121240
##  3    5   1390114
##  4    3.5  791624
##  5    2    711422
##  6    4.5  526736
##  7    1    345679
##  8    2.5  333010
##  9    1.5  106426
## 10    0.5   85374
```

```r
#Now we can plot this to see the distribution
edx %>% group_by(rating) %>%
  ggplot(aes(x=rating)) +
  geom_histogram()+
  theme_economist()
```



Data Cleaning

As mentioned before, now we proceed to split the `edx` dataset into two subsets for training and testing respectively.

```
library(caret)
#First, create the data partition into train and test sets
set.seed(1, sample.kind="Rounding")
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")

# 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)
```

Note that we have several features that can be used to build the model, but since this will increase the computation power required, this project will be focus just on using the movie's and user's information.

```
#Since we are not using all variables, we are going to select only some of them
train_set <- train_set %>% select(userId, movieId, rating, title)
test_set  <- test_set  %>% select(userId, movieId, rating, title)
```

Modeling

Linear Model

Even though a linear model does not always meet the requirements for a final model, it is always a good first approach to the model.

We could start at any point with a linear model, but statiscally we know that the average is the number that minimizes the RMSE. So, at first we are going to asume that there is a random variability in the distribution of the ratings and predict just the average rating for every movie. This model will be given by this formula:

$\hat{Y}_{u,i} = \mu + \epsilon_{i,u}$

Where $\hat{Y}$ is the predicted rating μ is the observed average (or mean) $\epsilon_{i,u}$ is the error distribution. Any other value would increase RMSE.

As we saw in previous steps, each movie has a different number of ratings and we can expect as well a different average calification for them. We can add this movie effect (also known as bias) to our linear model in order to get a better guess. It is exprese by b_i and given by this formula:

$\hat{Y}_{u,i} = \mu + b_i + \epsilon_{i,u}$

This effect can be calculated by the average difference between the observed rating y and the average μ. This is the formula:

$\hat{b}_i = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{\mu})$

Similar to the movie effect, we saw that not all users have rated the same number of movies, we can expect from that each user will have they own bias when rating a movie. This is the user effect and it is expressed similarly to the movie effect by this formula:

$\hat{b}_u = \frac{1}{N}\sum_{i=1}^{N}(y_{u,i} - \hat{b}_i - \hat{\mu})$

The final model that include both effects is given by this formula:

$\hat{Y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$

It is possible to add a genre and maybe even a year effect as well, but in this project we will not cover a model with those biases.

Regularization

Although this linear model provides a good stimation, it does not take into account that some movies are not very famous and many users does not rate a lot of movies. Statistycally, a small sample size leads to large stimated error.

The stimated error can be improved by adding a factor of penalization to small sample sizes and that, at the same time, has little impact on bigger sample sizes.

This is known as regularization and is given by these formulas:

$\hat{b}_i = \frac{1}{n_i+\lambda}\sum_{u=1}^{n_i}(y_{u,i} - \hat{\mu})$

$\hat{b}_u = \frac{1}{n_u+\lambda}\sum_{i=1}^{n_u}(y_{u,i} - \hat{b}_i - \hat{\mu})$

Where n represents the sample size and λ the penalization factor. This implies that with a sample size similar to λ the value of $\hat{b}_i$ and $\hat{b}_u$ is smaller than the original one, but for values of n larger than λ the result does not change very much.

Since this is a tunning parameter, the best way to choose a value of λ is by running simulations with several values.

Results

Model Evaluation Function

For the evaluation of the model we will be using the `RMSE` function of the `caret`, that takes two arguments: `observed_ratings` and `predicted_ratings`.

Linear Model Evaluation

We want to achieve the final model which includes the average rating, the movie effect and the user effect.

$\hat{Y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$

Just the average

First, as we said before, we are going to predict just the average for each movie.

```
#Calculate the average rating for all movies
avg_rating<- mean(train_set$rating)
avg_rating
```

```
## [1] 3.5125
```

```
#The error of this first approach
naive_error<- RMSE(test_set$rating, avg_rating)
naive_error
```

```
## [1] 1.0601
```

```
#The error compared with the goal of the project
options(digits = 5)
results <- tibble(Method = "Goal of the project", RMSE = 0.86490)
results <- bind_rows(results,
                     tibble(Method = "Just the average prediction",
                            RMSE = naive_error))
results
```

```
## # A tibble: 2 x 2
##   Method                       RMSE
##   <chr>                       <dbl>
## 1 Goal of the project         0.865
## 2 Just the average prediction 1.06
```

Including movie effect (bi)

Now we include b_i, which represents the bias for movie i.

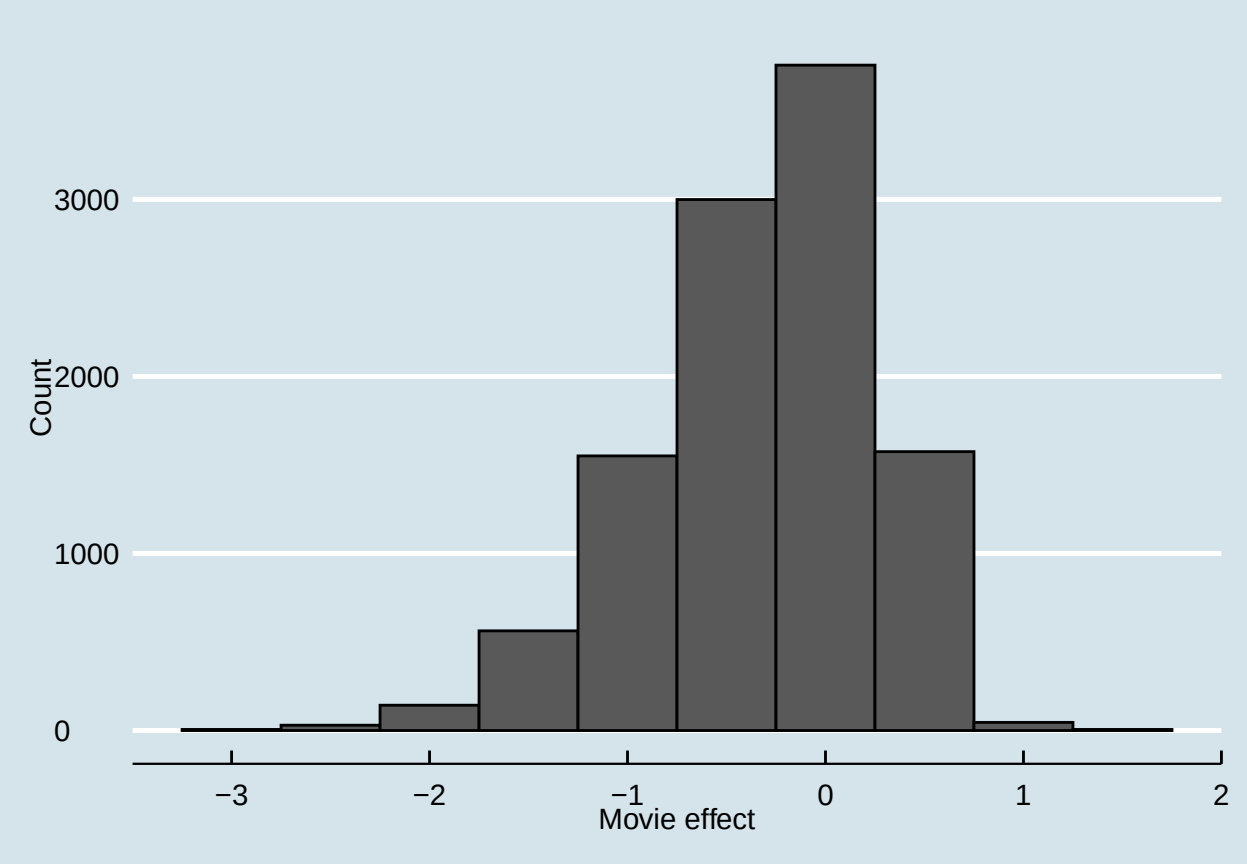
$\hat{y} = \mu + b_i + \epsilon_{u,i}$

```
#The effect (or bias) of each movie
movie_effect <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - avg_rating))
head(movie_effect)
```

```
## # A tibble: 6 x 2
##   movieId    b_i
##     <dbl>  <dbl>
## 1       1  0.415
## 2       2 -0.306
## 3       3 -0.361
## 4       4 -0.637
## 5       5 -0.442
## 6       6  0.302
```

Let us explore the distribution of this bias:

```
#Distribution of the movie effect
movie_effect %>% qplot(b_i, geom ="histogram",
                       bins = 10, data = .,
                       color = I("black"))+
  ylab("Count")+
  xlab("Movie effect")+
  theme_economist()
```




```
#Predict the rating with the movie effect
y_hat_bi <- avg_rating + test_set %>%
  left_join(movie_effect, by = "movieId") %>%
  .$b_i
head(y_hat_bi)
```

```
## [1] 3.1308 4.2210 3.7427 3.4295 3.9996 4.2789
```

```
#Compare the result with the goal of the project
results<- bind_rows(results,
                    tibble(Method = "Avg + Movie Effect (bi)",
                           RMSE= RMSE(test_set$rating, y_hat_bi)))
#RMSE when using avg as well as movie effect
results
```

```
## # A tibble: 3 x 2
##   Method                        RMSE
##   <chr>                        <dbl>
## 1 Goal of the project          0.865
## 2 Just the average prediction 1.06
## 3 Avg + Movie Effect (bi)     0.943
```

Including user effect

Now we include b_u, which represents the bias for user u.

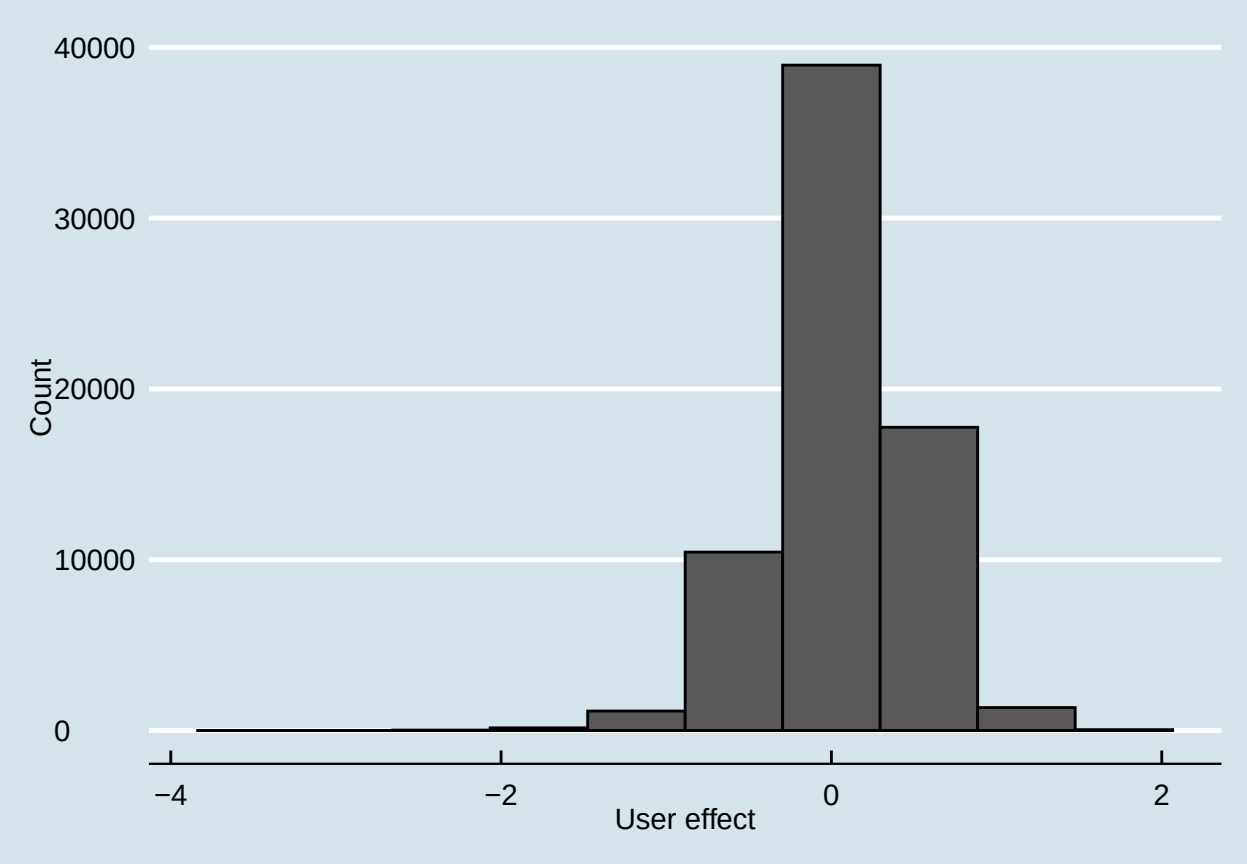
$\hat{y} = \mu + b_i + b_u + \epsilon_{u,i}$

```
#User effect (bu)
user_effect <- train_set %>%
  left_join(movie_effect, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - avg_rating - b_i))
head(user_effect)
```

```
## # A tibble: 6 x 2
##   userId    b_u
##    <int>  <dbl>
## 1      1  1.67
## 2      2 -0.434
## 3      3  0.268
## 4      4  0.698
## 5      5  0.100
## 6      6  0.336
```

Let us now explore the distribution of the user effect:

```
#Distribution of the user effect
user_effect %>% qplot(b_u, geom ="histogram",
                      bins = 10, data = .,
                      color = I("black"))+
  ylab("Count")+
  xlab("User effect")+
  theme_economist()
```




```
#Prediction including user effect as well
y_hat_bi_bu <- test_set %>%
  left_join(movie_effect, by='movieId') %>%
  left_join(user_effect, by='userId') %>%
  mutate(pred = avg_rating + b_i + b_u) %>%
  .$pred
RMSE(test_set$rating, y_hat_bi_bu)
```

```
## [1] 0.86468
```

```
#The result compared with the goal of the project
results<- bind_rows(results,
                    tibble(Method = "Avg + Movie Effect (bi) + User Effect (bu)",
                           RMSE = RMSE(test_set$rating, y_hat_bi_bu)))
#RMSE when using avg as well as movie effect and user effect
results
```

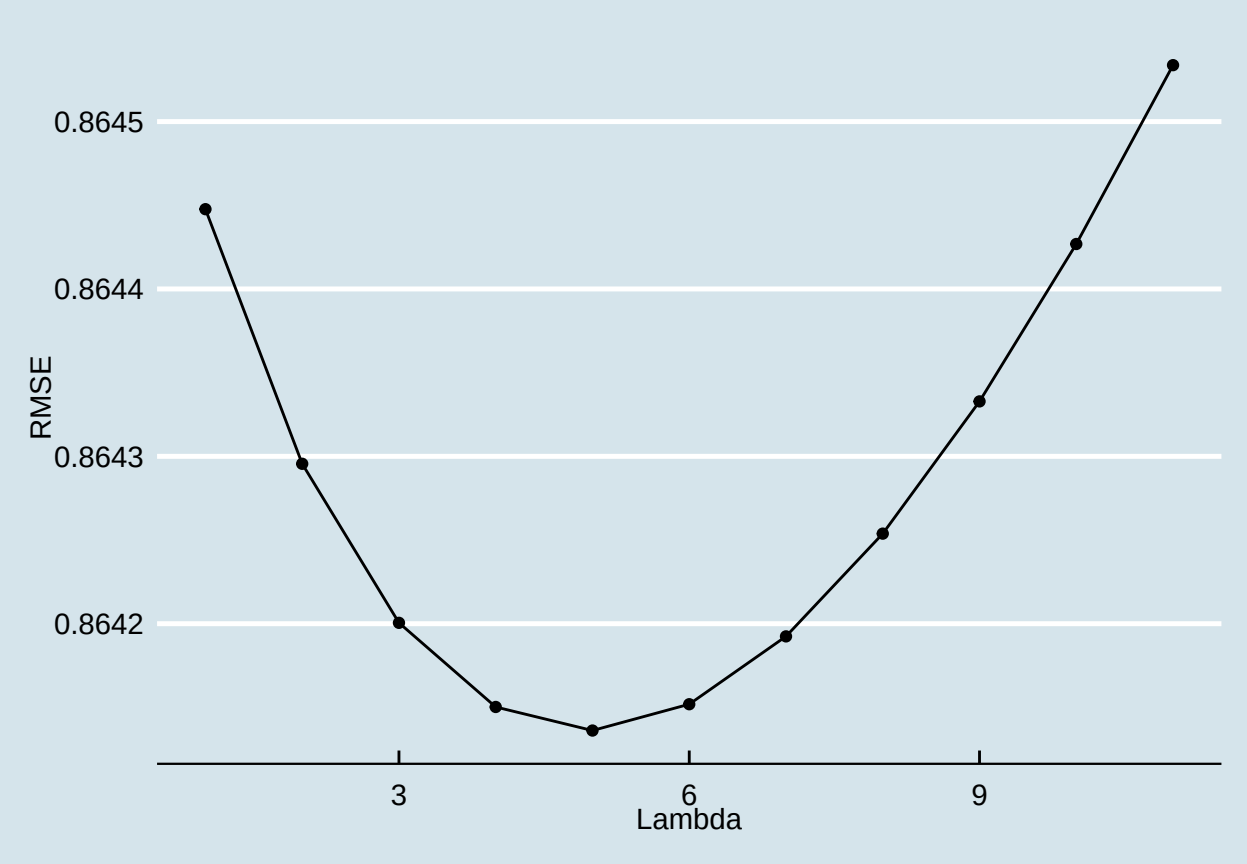
```
## # A tibble: 4 x 2
##   Method                                    RMSE
##   <chr>                                    <dbl>
## 1 Goal of the project                      0.865
## 2 Just the average prediction              1.06
## 3 Avg + Movie Effect (bi)                  0.943
## 4 Avg + Movie Effect (bi) + User Effect (bu) 0.865
```

We have already seen an improvement. However, we still need to improve our model.

Regularization

Now, in order to regularized, we need to add a penalization factor λ. We define a `regularization` function that returns the RMSE result. As we said before, since it is a tunning parameter, we try with several values and then pick the one that minimize the RMSE.

```
#Regularization function for the linear model that includes average rating, movie effect and user effec
regularization <- function(lambda, train_set, test_set){

  # Avg Rating (mu)
  avg_rating <- mean(train_set$rating)

  # Movie effect (bi)
  movie_effect <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - avg_rating)/(n()+lambda))

  # User effect (bu)
  user_effect <- train_set %>%
    left_join(movie_effect, by="movieId") %>%
    filter(!is.na(b_i)) %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - avg_rating)/(n()+lambda))

  # Prediction: mu + bi + bu
  predicted_ratings <- test_set %>%
    left_join(movie_effect, by = "movieId") %>%
    left_join(user_effect, by = "userId") %>%
    filter(!is.na(b_i), !is.na(b_u)) %>%
    mutate(pred = avg_rating + b_i + b_u) %>%
    pull(pred)

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

#Since lambda is a tunning parameter, we are going to try with several values
lambdas<-seq(0:10)

#Testing lambda with the 'edx' dataset
rmses<-sapply(lambdas,
              regularization,
              train_set=train_set,
              test_set=test_set)

#Visualization of which value of lambda give us the minimun RMSE
tibble(Lambda = lambdas, RMSE = rmses) %>%
  ggplot(aes(x = Lambda, y = RMSE))+
  geom_point()+
  geom_line()+
  theme_economist()
```



Now, we are going to apply the best value of λ in the linear model.

```
#Selecting the value of lambda that give us the minimun RMSE
lambda<- lambdas[which.min(rmses)]

#Compare the result of the regularization with the goal of the project
results <- bind_rows(results,
                     tibble(Method = "Regularized bi and bu",
                            RMSE = regularization(lambda,train_set, test_set)))
results
```

```
## # A tibble: 5 x 2
##   Method                                    RMSE
##   <chr>                                    <dbl>
## 1 Goal of the project                      0.865
## 2 Just the average prediction              1.06
## 3 Avg + Movie Effect (bi)                  0.943
## 4 Avg + Movie Effect (bi) + User Effect (bu) 0.865
## 5 Regularized bi and bu                    0.864
```

Final Validation

Now, that we have achieved the RMSE result we wanted with the `train_set` and the `test_set`, we can train the model with the entire `edx` dataset and then test the model with the `validation` dataset.

```r
#Final evaluation of Linear Model regularized with the 'validation' dataset
avg_edx<- mean(edx$rating)

#Movie effect (b_i)
movie_effect_edx<- edx%>%
  group_by(movieId)%>%
  summarize(b_i=sum(rating-avg_edx)/(n()+lambda))

#User effect(b_u)
user_effect_edx<- edx%>%
  left_join(movie_effect_edx, by="movieId")%>%
  group_by(userId)%>%
  summarize(b_u=sum(rating-avg_edx-b_i)/(n()+lambda))

#Prediction
y_hat_edx<- validation%>%
  left_join(movie_effect_edx, by="movieId")%>%
  left_join(user_effect_edx, by="userId")%>%
  mutate(pred=avg_edx + b_i + b_u)%>%
  pull(pred)

#Final result
options(digits = 5)
final_result<- as.data.frame(bind_rows(results[1,],
                    tibble(Method= "Final Validation",
                           RMSE=RMSE(y_hat_edx, validation$rating))))
final_result
```

```
##                Method    RMSE
## 1 Goal of the project 0.86490
## 2    Final Validation 0.86482
```

As expected, the RMSE with the `validation` dataset is lower than the target and a little bit higher than the RMSE of the `test_set`.

Conclusion

At first, we downloaded the dataset with which we were going to work, prepare it and found insights trough data exploration.

Then, we started scratching a model predicting just the average rating for every movie and saw that this generated a very large RMSE. After that, we added a movie and a user effect, which improved our model. After that, by adding a penalization factor with regularization we ended up we the model that generated the RMSE we were trying to achieve.

Limitation

Nowadays, most machine learning algorithms use a lot of features to improve the predictions as much as possible. Due to computation power requirements, in this project we just covered a linear model that take the average rating for all movies, the movie bias of each movie and the user bias for each user.

While we stick to the linear model method, this model can still be improved by adding a bias by genre or year. A more complicated algorithm can be generated by implementing matrix factorization, but it is not our goal to cover that here.

Future Work

There are several R packages that can be used to build a recomendation system, one of them is `recommenderlab`, that cover all the features that we covered troughout this project with different approaches that we did not covered in this project.

Other packages for recommendation systems can be found in The Comprehensive R Archive Network (CRAN) website.