MovieLens Recommender System

Maria Belen Acosta Vera

2022

Contents

1 Summary

This machine learning project creates a prediction model based on user movie ratings from the MovieLens database. This is the first half of the Capstone course from the Data Science Professional Certificate offered by HarvardX.

It is composed of an initial data exploration and visualization of relevant variables such as the movie genre, some RMSE models and the final results obtained from the model with the lowest RMSE.

2 Introduction

Machine learning models are crucial nowadays. They bring into reality the concept of using the past to predict the future. Without it, many services as we know them wouldn't exist or would be radically different. Given this, the opportunity made available by HarvardX for everyone to learn about and create prediction models is invaluable. A great example on this is Netflix's movie recommender system, which is emulated in this project.

The MovieLens database consists of 10000054 observations of 10676 movies rated by 69878 users. This data is then processed to create prediction models, to achieve a RMSE (Root Mean Squared Method) lower than 0.86549, which is the goal of for this section of the Capstone course from the Data Science Professional Certificate from HarvardX.

Finally, as an economics undergrad, I recognize the importance of models such as these for the economics fields. Similar models could be created to model people's preferences and predict their future behavior. Also, to model shocks to the market and ways of recovering after them. As a whole, machine learning models are crucial to understand what our past is trying to tell us. Through mathematical inference and with the certainty that is provided by machine learning models, we are as close as we can be to predicting the future.

3 Data Analysis

3.1 Database preparation

3.1.1 Loading the database

First and foremost, we start the project by loading the database and the code provided on the edx course. The following code downloads, organizes and simplifies the database for the creation of the "edx" and "validation" datasets used to train and validate our model, respectively.

```r
##########################################################
# Create edx set, validation set (final hold-out test 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")

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")

# 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)

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

3.1.2 Loading extra packages

The following packages are fundamental for our data visualization and data wrangling process.

```
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
```

3.1.3 Splitting the edx database

After loading and creating the edx and validation databases, the edx database is divided again to train and test the different models before applying them to the validation database. The process will be the same as when creating the edx and validation databases.

```
#First, we divide the edx database 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,]
# Then, we assure that the variables "userId" and "movieId" are in both the test set and the train set
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
# Finally, we add the rows removed from the test set back into the train set
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)
rm(test_index, temp, removed)
```

3.1.4 Mutating the timestamp variable

To achieve this, I created a parallel database to maintain the original data. We'll use the as_datetime function of the lubridate package to transform the timestamps in a readable format.

```
#converting the time stamp on the edx database

edx <-  edx %>% mutate(timestamp = as_datetime(timestamp))

#converting the time stamp on the test_set database

test_set<- test_set %>% mutate(timestamp = as_datetime(timestamp))

#converting the time stamp on the train_set database

train_set<- train_set %>% mutate(timestamp = as_datetime(timestamp))
```

```r
#converting the time stamp on the validation database

validation <- validation %>% mutate(timestamp = as_datetime(timestamp))
```

3.1.5 Creating a column named "Release_Y"

The purpose of this column is to have an easy access to the year the movie was released.This column will be created in all the databases.

```r
#changing the edx database to add a column named Release_Y
edx <- edx %>% mutate(Release_Y = as.numeric(str_sub(title,-5,-2)))

#changing the test_set database to add a column named Release_Y
test_set <- test_set %>% mutate(Release_Y = as.numeric(str_sub(title,-5,-2)))

#changing the train_set database to add a column named Release_Y
train_set <- train_set %>% mutate(Release_Y = as.numeric(str_sub(title,-5,-2)))

#changing the validation database to add a column named Release_Y
validation <- validation %>% mutate(Release_Y = as.numeric(str_sub(title,-5,-2)))
```

3.1.6 Creating a column named "yearRated"

This column will contain the year in which the movie was rated.

```r
#create a new column filled with the years in which the movies were rated

##on the edx database
edx$yearRated <- as.POSIXct(edx$timestamp,
origin="1970-01-01")
edx$yearRated <- format(edx$timestamp,"%Y")
edx$yearRated <- as.numeric(edx$yearRated)

##on the validation database
validation$yearRated <- as.POSIXct(validation$timestamp,
origin="1970-01-01")
validation$yearRated <- format(validation$timestamp,"%Y")
validation$yearRated <- as.numeric(validation$yearRated)

##on the train set
train_set$yearRated <- as.POSIXct(train_set$timestamp,
origin="1970-01-01")
train_set$yearRated <- format(train_set$timestamp,"%Y")
train_set$yearRated <- as.numeric(train_set$yearRated)

##on the test set
test_set$yearRated <- as.POSIXct(test_set$timestamp,
origin="1970-01-01")
test_set$yearRated  <- format(test_set$timestamp,"%Y")
test_set$yearRated <- as.numeric(test_set$yearRated)
```

3.2 Basic information on the database

The next lines of code will give us the amount of movies, genres and users on the database.

```
#join both edx and validation databases
bidtb<- rbind(edx, validation)

bidtb %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

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

```
#amount of genres

genres <- bidtb %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarise()
rm(bidtb)
rm(genres)
```

In total, there are 10677 movies from 20 different genres that were rated by 69878 users.

4 Data exploration

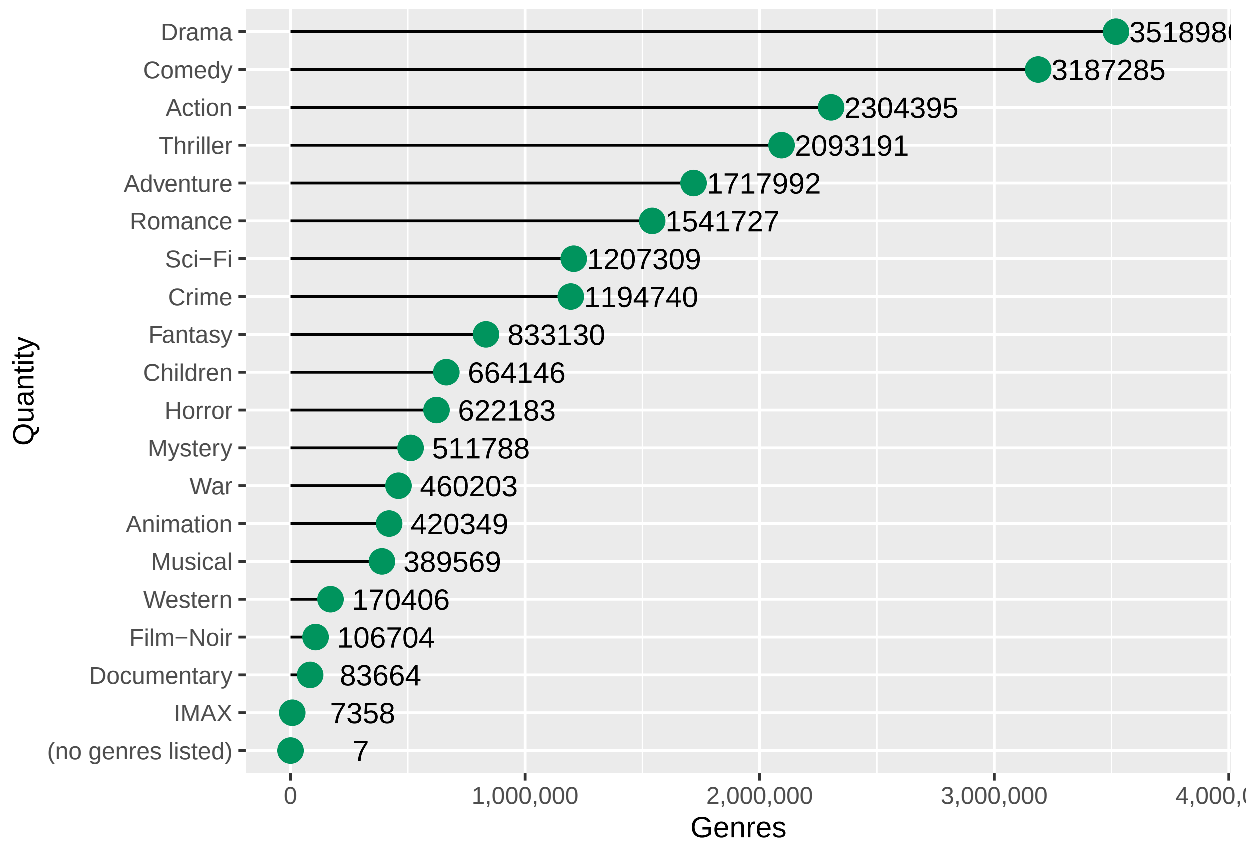
It is important to mention that the data exploration is done on the `train_set` database.

4.1 Quantity of movies per genre

For this section, we will separate the genres in movies with more than one genre so as to have the total count of movies per genre. In consequence, a part of the data will be duplicated, so, the total of movies per genre *does not represent* the total of movies in the train_set_ts database.

```
movies_per_genre <- train_set %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarise(quantity = n()) %>%
  arrange(desc(quantity))

#We can see it graphically as
movies_per_genre %>% ggplot(aes(genres, quantity)) +
  geom_segment(aes(x=reorder(genres, quantity) ,xend=genres, y=0, yend=quantity), color="black") + geom_
  geom_text(aes(label= quantity), position = position_nudge(y= 300000)) +
  xlab("Quantity") + scale_y_continuous(name="Genres", labels = scales::comma)
```



```
rm(movies_per_genre)
```

As we can observe on the graph, the genres with most movies are (in descendent order): drama, comedy, action, thriller and adventure.

4.2 Movies per year

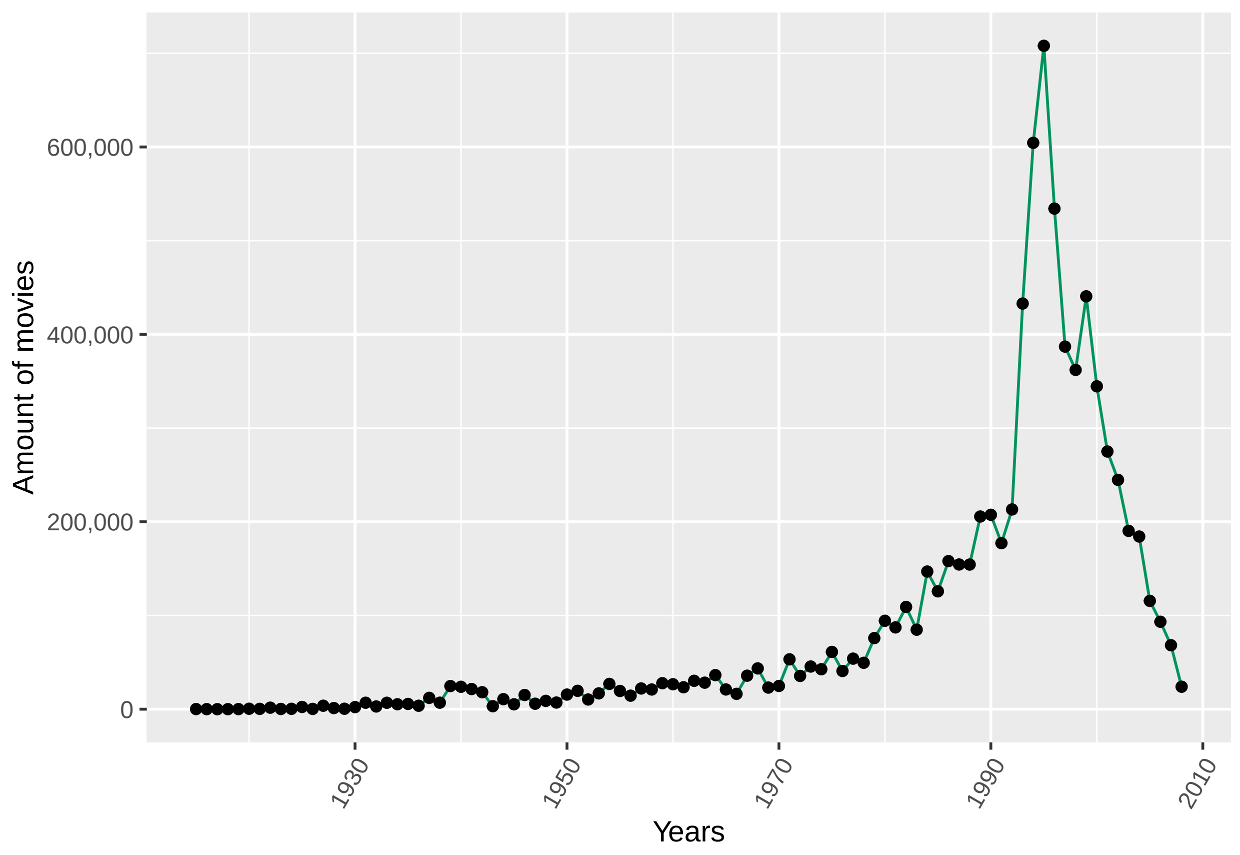
For this section, we're going to visualize the amount of movies released per year. To achieve this, we're using the Release_Y column we've created before.

```
#Making the table
movies_per_releaseY <- train_set %>%
   group_by(Release_Y) %>%
  summarise(quantity = n()) %>%
  arrange(desc(quantity))
head(movies_per_releaseY, 10)
```

```
## # A tibble: 10 x 2
##    Release_Y quantity
##        <dbl>    <int>
##  1      1995   707979
##  2      1994   604423
##  3      1996   534246
##  4      1999   440619
##  5      1993   432917
##  6      1997   386984
##  7      1998   362084
##  8      2000   344593
##  9      2001   275005
## 10      2002   244747
```

```
#Graphically,

movies_per_releaseY %>%
  ggplot(aes(x=Release_Y, y=quantity)) +
  geom_line(color="#00945d") +
  geom_point(color= "black") +
  xlab("") +
  theme(axis.text.x=element_text(angle=60, hjust=1)) + xlab("Years") +
  scale_y_continuous(name="Amount of movies", labels = scales::comma)
```



```
rm(movies_per_releaseY)
```

Most movies on our database were released between the years 1990 and 2005. There's also a large increase in movie quantity when comparing to the previous years.

4.3 Rating distribution per decade

```
train_set %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarise(avg_rating = mean(rating))
```

```
## # A tibble: 20 x 2
##    genres             avg_rating
##    <chr>                   <dbl>
##  1 (no genres listed)       3.64
##  2 Action                   3.42
##  3 Adventure                3.49
##  4 Animation                3.60
##  5 Children                 3.42
##  6 Comedy                   3.44
##  7 Crime                    3.67
##  8 Documentary              3.78
##  9 Drama                        3.67
## 10 Fantasy                      3.50
## 11 Film-Noir                    4.01
## 12 Horror                       3.27
## 13 IMAX                         3.77
## 14 Musical                      3.56
## 15 Mystery                      3.68
## 16 Romance                      3.55
## 17 Sci-Fi                       3.40
## 18 Thriller                     3.51
## 19 War                          3.78
## 20 Western                      3.56
```

We can see that most movies have an average of three points in rating, with Noir films having the highest rating average. Horror films have the lowest average rating.

5 Models

As instructed, we will be working with RMSE. It tells us the average distance between the predicted values from the model and the actual values in the dataset.

The lower the RMSE, the better a given model is able to "fit" a dataset. Source: Statology

Our goal is to achieve an RMSE lower or equal to **0.86549**

The formula is the following:

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

Where:

- $y_{u,i}$ is the rating of movie i given by user u denoting our prediction $\hat{y}_{u,i}$
- N is the number of user/movies combinations, where the sum occurs on all these combinations.

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

5.1 Model 1

First, let's compute an RMSE based on the mean. Our formula is the following:

$$Y_{u,i} = \hat{\mu} + \varepsilon_{u,i}$$

Where $\varepsilon_{i,u}$ represent the independent errors sampled from the same distribution centered at 0 and $\hat{\mu}$ is the "true" rating for all movies.

```
mean <- mean(train_set$rating)
#we'll use our RMSE function

first_RMSE <- RMSE(test_set$rating, mean)

#Let's create a dataframe to store and compare all of our results.
results <- data.frame(
  Model="First Model", RMSE = first_RMSE)
results%>% knitr::kable()
```

Model	RMSE
First Model	1.060054

```
results
```

```
##         Model     RMSE
## 1 First Model 1.060054
```

Given that our model doesn't take into account many variables, it leads to a high RMSE, equal to **1.060054**'. This is very far from our goal so let's try other models.

5.2 Model 2: Movie Effect

Let's take into account the movie effect. This is due to the fact that some movies are often rated higher than others. For that, let's add b_i to the previous formula in Model 1. Now, our formula looks like this:

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

Where:

- $\varepsilon_{i,u}$ is the independent errors sampled from the same distribution centered at 0.
- $\hat{\mu}$ is the mean.
- And b_i is the average of $Y_{u,i} - \hat{\mu}$ for each movie i

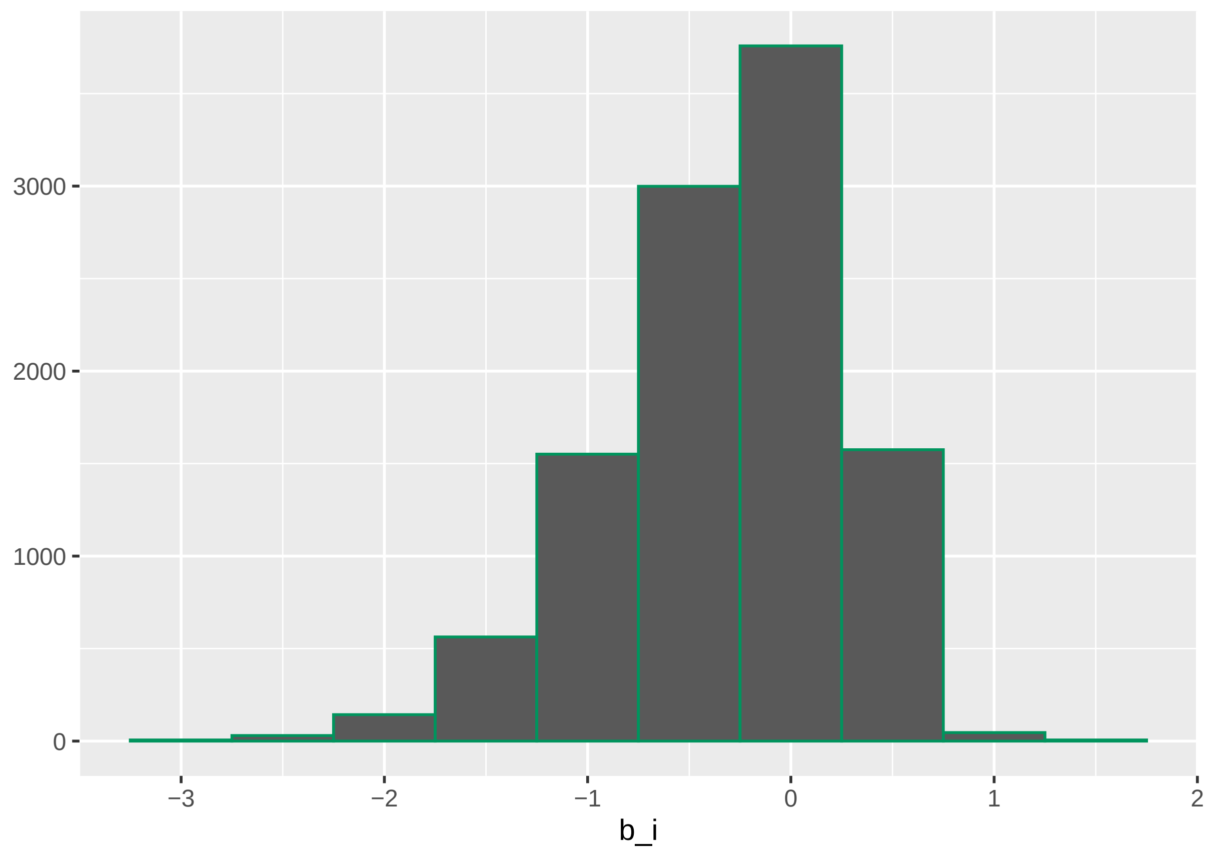
To estimate b_i we'll the following code:

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

We change the name of the mean from `mean` to `mu` to differentiate between the first and second model.

With the next plot, we can visualize these estimates:

```r
qplot(b_i, data = movie_avgs, bins = 10, color = I("#00945d"))
```



Now, let's see the improvement on our model:

```
movie_effect <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
snd_RMSE <- RMSE(movie_effect, test_set$rating)

results<- bind_rows(
  results,
  data.frame(Model="Model 2",
             RMSE = snd_RMSE))
results%>% knitr::kable()
```

Model	RMSE
First Model	1.0600537
Model 2	0.9429615

```
results
```

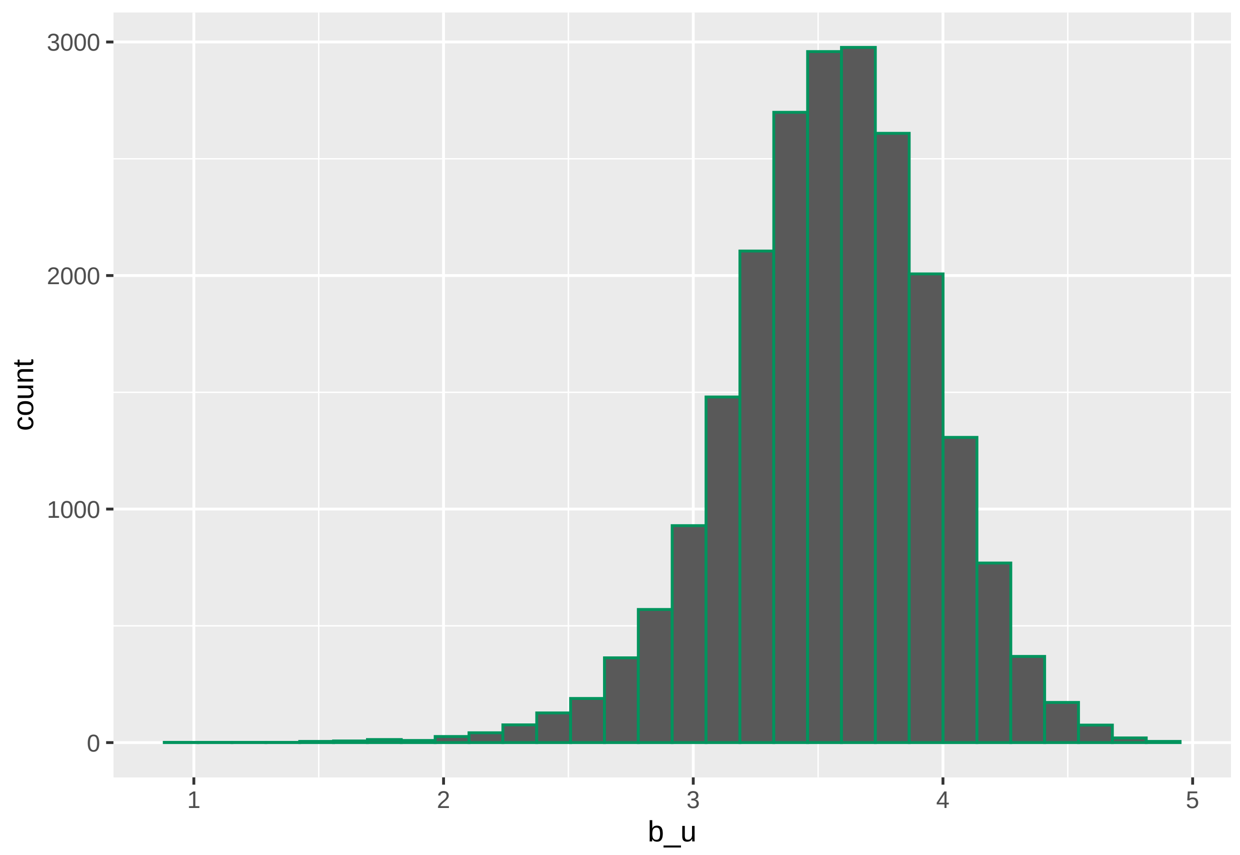
```
##         Model      RMSE
## 1 First Model 1.0600537
## 2     Model 2 0.9429615
```

There's definitely an improvement, but it can still get better. Let's not forget that our goal is an RMSE lower than **0.86549**

5.3 Model 3: Movie + User Effect.

We've seen the effect of movies, now let's see the human factor: the users. Let's see the average rating of user u who has rated 100 or more movies.

```
train_set %>%
  group_by(userId) %>%
  filter(n()>=100) %>%
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "#00945d")
```



We see a lot of extremes between users: some love movies and some hate them. We could improve our model adding this to our:

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

Where:

- $\varepsilon_{i,u}$ is the independent errors sampled from the same distribution centered at 0.
- $\hat{\mu}$ is the mean.
- b_i is the average of $Y_{u,i} - \hat{\mu}$ for each movie i.
- b_u is the average of $Y_{u,i} - \hat{\mu} - \hat{b}_i$; is a user-specific effect.

As it was said above, b_u is the average of $Y_{u,i} - \hat{\mu} - \hat{b}_i$, we calculate it like this for our database:

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

Let's see how much our RMSE improves.

```
#First, create the predictors
movieuser_effect <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
thrd_RMSE <- RMSE(movieuser_effect, test_set$rating)

results<- bind_rows(
  results,
  data.frame(Model="Model 3",
             RMSE = thrd_RMSE))
results%>% knitr::kable()
```

Model	RMSE
First Model	1.0600537
Model 2	0.9429615
Model 3	0.8646843

```
results
```

```
##          Model      RMSE
## 1 First Model 1.0600537
## 2     Model 2 0.9429615
## 3     Model 3 0.8646843
```

We've reached an RMSE of 0.8646843, which is lower than the goal: **RMSE <= 0.86549**!!

5.4 Regularization

Let's try regularizing our models to see if it improves our RMSE. The general idea behind regularization is to constrain the total variability of the effect sizes. This could help us if we have many extreme values which could hinder our prediction. With the following formula, we're adding a penalty:

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

Where:

- n_i is the number of ratings made for movie i.
- λ is the penalty.
- $\hat{\mu}$ is the mean.
- b_i is the average of $Y_{u,i} - \hat{\mu}$ for each movie i.

In our results, we can see that `Model 3`, the one that computes movie and user effect is the best model so far, so we'll use that one for our regularization.

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

Model	RMSE
First Model	1.0600537
Model 2	0.9429615
Model 3	0.8646843

5.4.1 Penalized least squares

```r
# Let's create our lambdas
lambdas <- seq(0, 10, 0.25)

#Let's add the regularization to the third model: Movie and User effect.
rmses_movieuser <- sapply(lambdas, function(l){

  mu <- mean(train_set$rating)

  b_i <- train_set %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - mu)/(n() +l))

  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarise(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) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_set$rating))

})
reg_model_3 <- rmse_regularisation <- min(rmses_movieuser)
results<- bind_rows(
  results,
  data.frame(Model="Reg_Model 3",
             RMSE = reg_model_3))
results%>% knitr::kable()
```

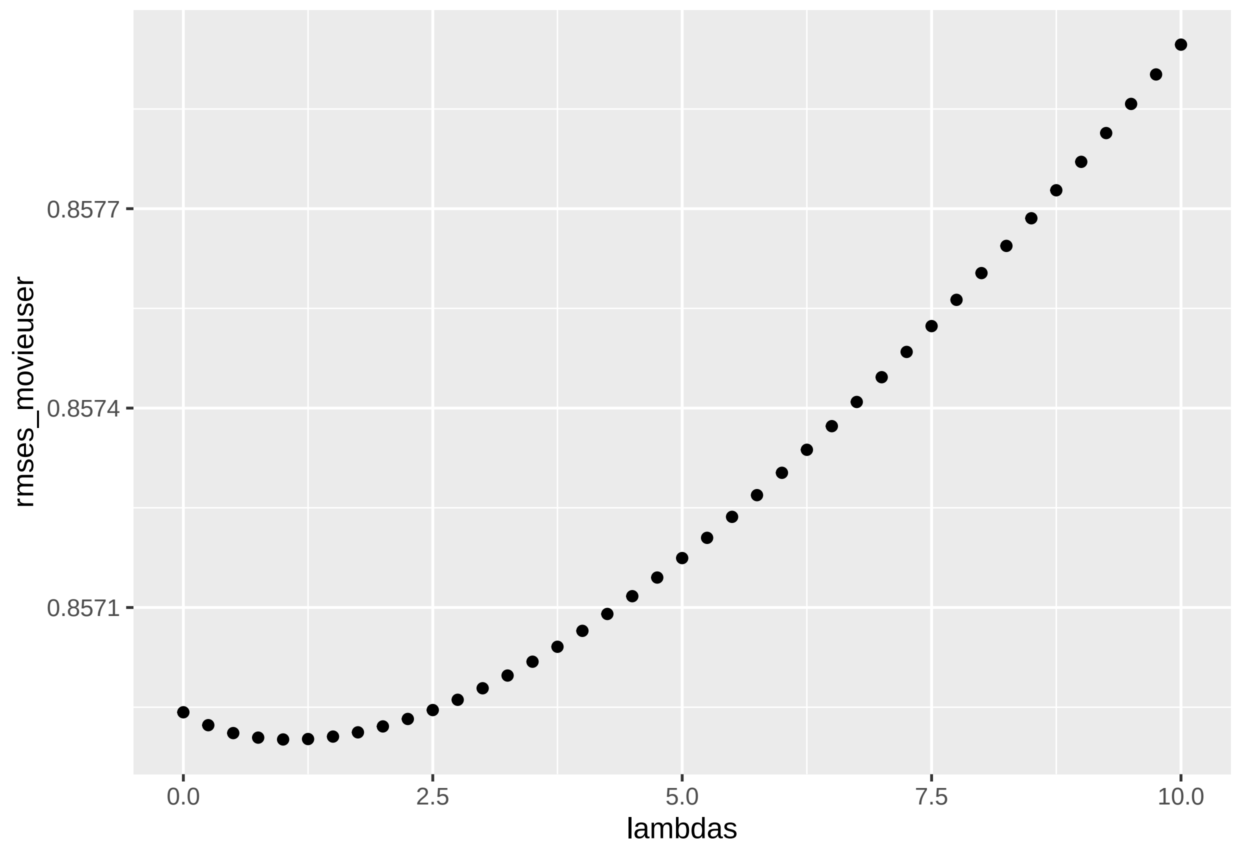
Model	RMSE
First Model	1.0600537
Model 2	0.9429615
Model 3	0.8646843
Reg_Model 3	0.8569015

```r
results
```

```
##          Model      RMSE
## 1 First Model 1.0600537
## 2     Model 2 0.9429615
## 3     Model 3 0.8646843
## 4 Reg_Model 3 0.8569015
```

Great news! we've achieved our RMSE goal: the RMSE (**0.8648170**) is indeed lower than 0.86549. Now, let's check which lambda value minimizes our RMSE

```
qplot(lambdas, rmses_movieuser)
```



```
lambdas[which.min(rmses_movieuser)]
```

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

To minimize the RMSE, lambda must be **4.5**

#Validation Finally, we're going to test our model in the validation database.

```
#We calculate the mean for the edx database
muhat <- mean(edx$rating)
muhat
```

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

Let's validate our model using the `validation` database.

```r
lambdas <- seq(0, 10, 0.1)
# Modeling with Regularized Movie + User Effect Model
reg_movieuser_val <- sapply(lambdas, function(l){
# Average rating by movie
b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - muhat)/(n()+l))
# Average rating by user
b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - muhat)/(n()+l))
# RMSE prediction on the validation database
predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = muhat + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predicted_ratings, validation$rating))
})
rmse_reg_movieuser_val <- min(reg_movieuser_val)

results<- bind_rows(
  results,
  data.frame(Model="Validation - Reg_Model 3",
             RMSE = rmse_reg_movieuser_val))
results%>% knitr::kable()
```

Model	RMSE
First Model	1.0600537
Model 2	0.9429615
Model 3	0.8646843
Reg_Model 3	0.8569015
Validation - Reg_Model 3	0.8648170

```r
results
```

```
##                      Model      RMSE
## 1              First Model 1.0600537
## 2                  Model 2 0.9429615
## 3                  Model 3 0.8646843
## 4              Reg_Model 3 0.8569015
## 5 Validation - Reg_Model 3 0.8648170
```

This new result more than reaches our goal since, **0.864817** is smaller than **0.8654673** (our previous RMSE) and both are lower than the goal of an **RMSE <= 0.86549**

6 Results

As it was mentioned before, we've reached our goal of an **RMSE <= 0.86549** with our last model `Validation - Reg_Model 3`. All of the results obtained can be seen on the table below.

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

Model	RMSE
First Model	1.0600537
Model 2	0.9429615
Model 3	0.8646843
Reg_Model 3	0.8569015
Validation - Reg_Model 3	0.8648170

7 Conclusion

Most definitely, in databases as large as this one, it's necessary to regularize. This is proven by the fact that to reach our goal, we had to regularize our third model, that used movie and user effects. Regularizing was very important for this project, since it allowed us to reach an RMSE of **0.864817** and achieve the project goal: an **RMSE <= 0.86549**.

7.1 Limitations and Future Work

Only two variables were used to model and there's many more that can influence and help achieve a lower RMSE. It'd be interesting to see the effect of genre or the release year for future works, also, to try and use other machine learning models.