RATING PREDICTIONS: A MOVIE RECOMMENDATION SYSTEM - Capstone Project - HarvardX Data Science Professional Certificate Program Course - Harvard University

Miguel Ángel Boto Bravo, PhD in Linguistics

06/09/2020

1. INTRODUCTION

The final project Movielens is the last part of the Data Science Certificate Program Course organized by Harvardx through the Edx Platform, in which the student must create a movie recommendation system by training a machine learning algorithm and using the inputs in a subset to predict the movie ratings in a validation set.

To do this, we will use "Movielens", a movie rating database with millions of votes available at https: //grouplens.org/datasets/movielens/, although a small portion with ten million ratings will be used for the present work.

The measure used to evaluate algorithm performance is the RMSE (Root Mean Square Error), one of the most frequently used measure of the differences between values predicted by a model and the values observed.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

RMSE is an accuracy measure to compare forecasting errors throw different models. The goal of this project is to obtain a RMSE value lower than 0.86490.

2. EXPLORATORY ANALYSIS AND DATA VISUALIZATION:

2.1 Installing libraries and preparing the database:

Before starting the exploratory analysis of Movielens, we must install and run the following libraries:

```
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)
library(lubridate)
library(tinytex)
```

Next, we will download the Movielens partition with which we are going to work, format the dataset and create a training and validation sets:

```
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: 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,]

# UserId and movieId in validation set must be 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)
```

Finally, we need to include two more columns in order to use "year of released" and the "year of rating" in some graphics:

```
edx2 <- mutate(edx, year_rated = year(as_datetime(timestamp)),
                      year = as.numeric(str_sub(title,-5,-2)))

validation2 <- mutate(edx, year_rated = year(as_datetime(timestamp)),
                      year = as.numeric(str_sub(title,-5,-2)))
```

2.2 Exploratory analysis:

The resulting database contains 8 columns and 9 million rows:

```
head(edx2)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                            genres year_rated year
## 1:                Comedy|Romance       1996 1992
## 2:         Action|Crime|Thriller       1996 1995
## 3:  Action|Drama|Sci-Fi|Thriller       1996 1995
## 4:       Action|Adventure|Sci-Fi       1996 1994
## 5: Action|Adventure|Drama|Sci-Fi       1996 1994
## 6:       Children|Comedy|Fantasy       1996 1994
```

Next, we show the statistical summary of the content of main columns:

```
summary(edx2)
```

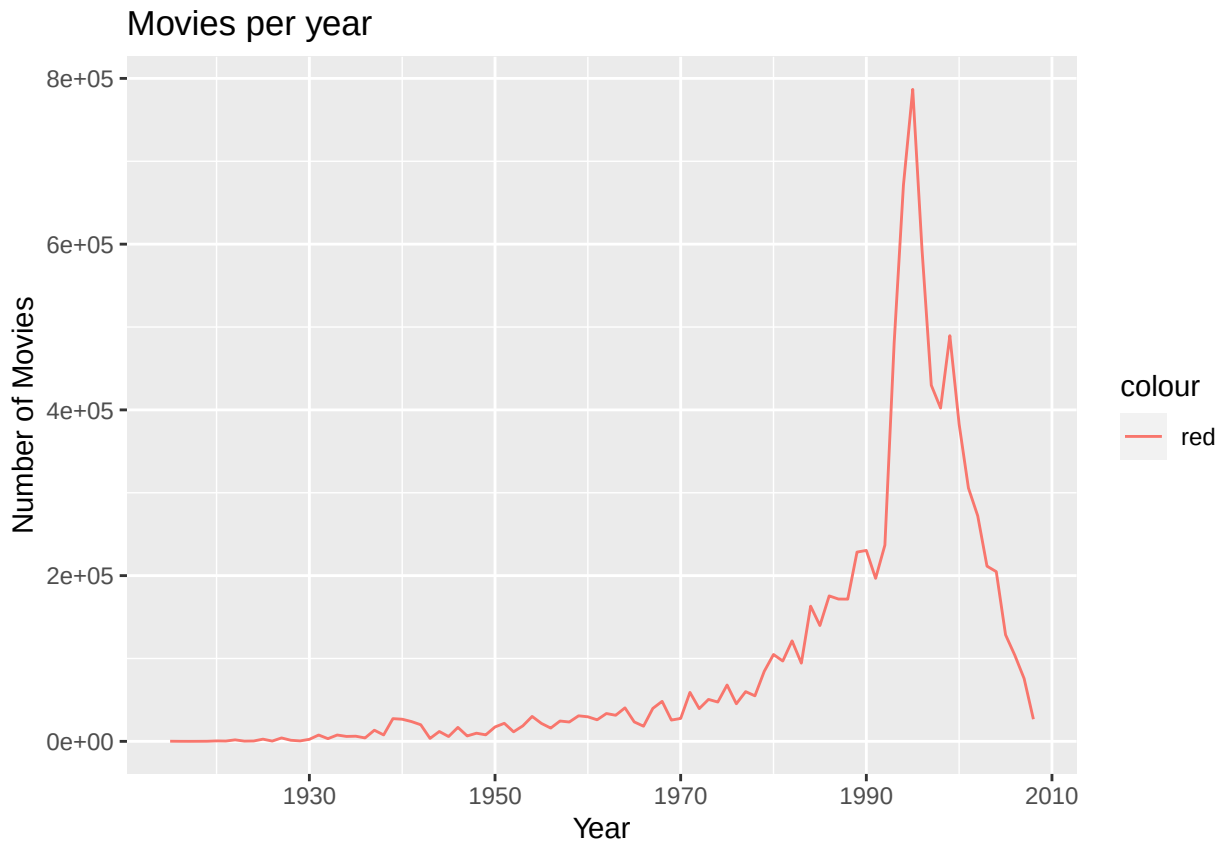
```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres            year_rated        year
##  Length:9000055     Length:9000055     Min.   :1995   Min.   :1915
##  Class :character   Class :character   1st Qu.:2000   1st Qu.:1987
##  Mode  :character   Mode  :character   Median :2002   Median :1994
##                                        Mean   :2002   Mean   :1990
##                                        3rd Qu.:2005   3rd Qu.:1998
##                                        Max.   :2009   Max.   :2008
```

```
edx2 %>% summarize(
  'Unique Users'=n_distinct(userId),
  'Movies'=n_distinct(movieId),
  'Minumum Rating'=min(rating),
  'Maximum Rating'=max(rating)

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

Unique Users	Movies	Minumum Rating	Maximum Rating
69878	10677	0.5	5

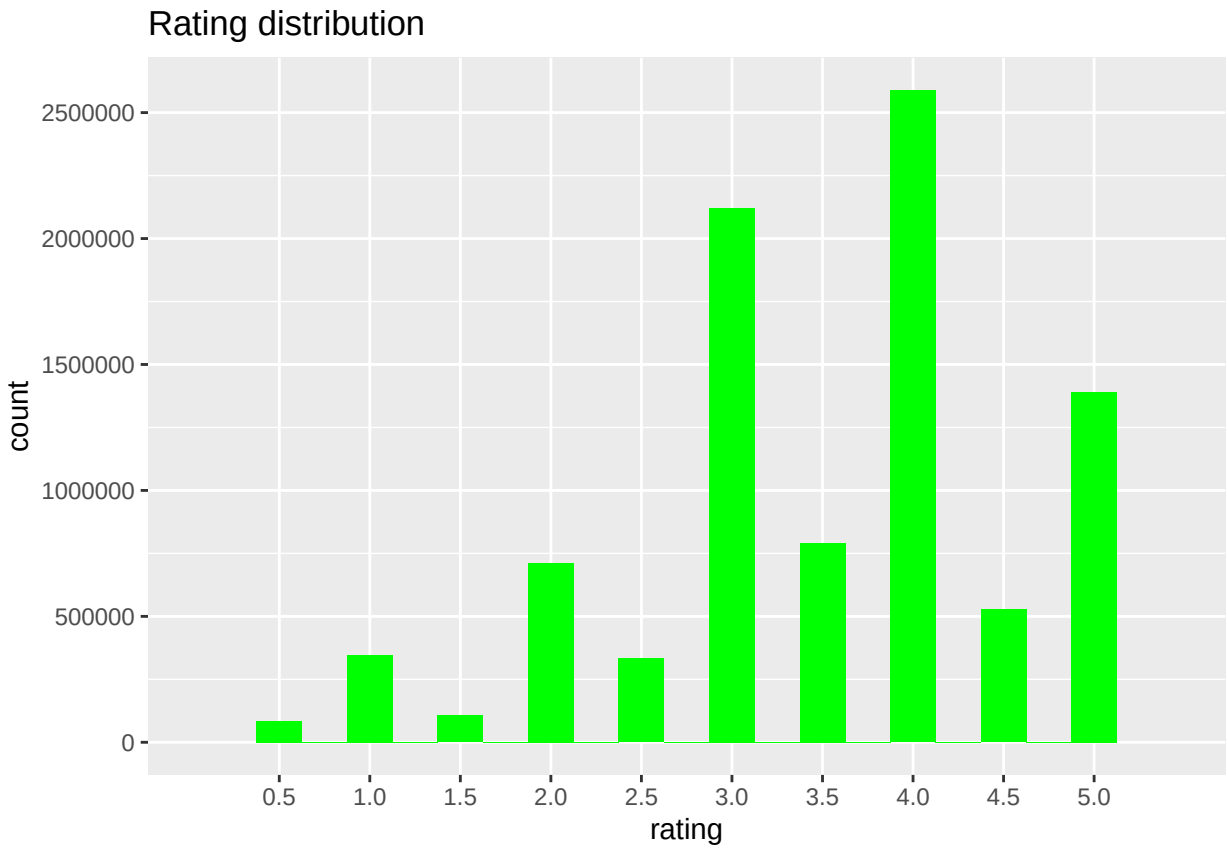
To delve into the distribution of ratings, users and movies, nothing better than to analyze the following graph. In the first case, we can observe how the majority of productions belongs to '90 decade:

```r
#Movies by year of released

edx2 %>%
  group_by(year) %>%
  count(movieId) %>%
  summarize(total = sum(n))%>%
  ggplot(aes(year, total, color = "red"))+
  geom_line()+
  xlab("Year")+
  ylab("Number of Movies")+
  ggtitle("Movies per year")
```



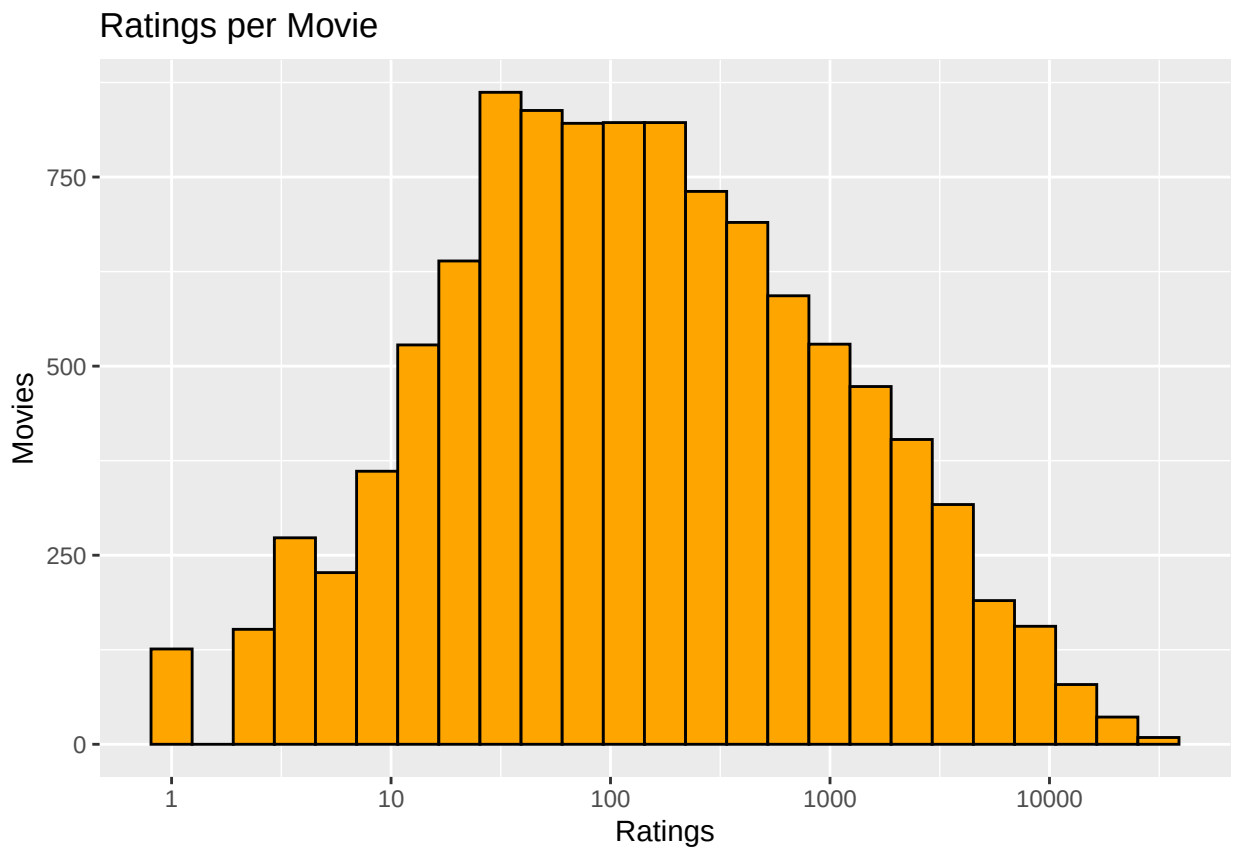
Likewise, the distribution of votes is concentrated in the range from 3.0 to 4.0 ranking

```r
#Rating distribution
edx2 %>%
  ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.25, fill = "green") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  scale_y_continuous(breaks = c(seq(0, 3000000, 500000))) +
  ggtitle("Rating distribution")
```

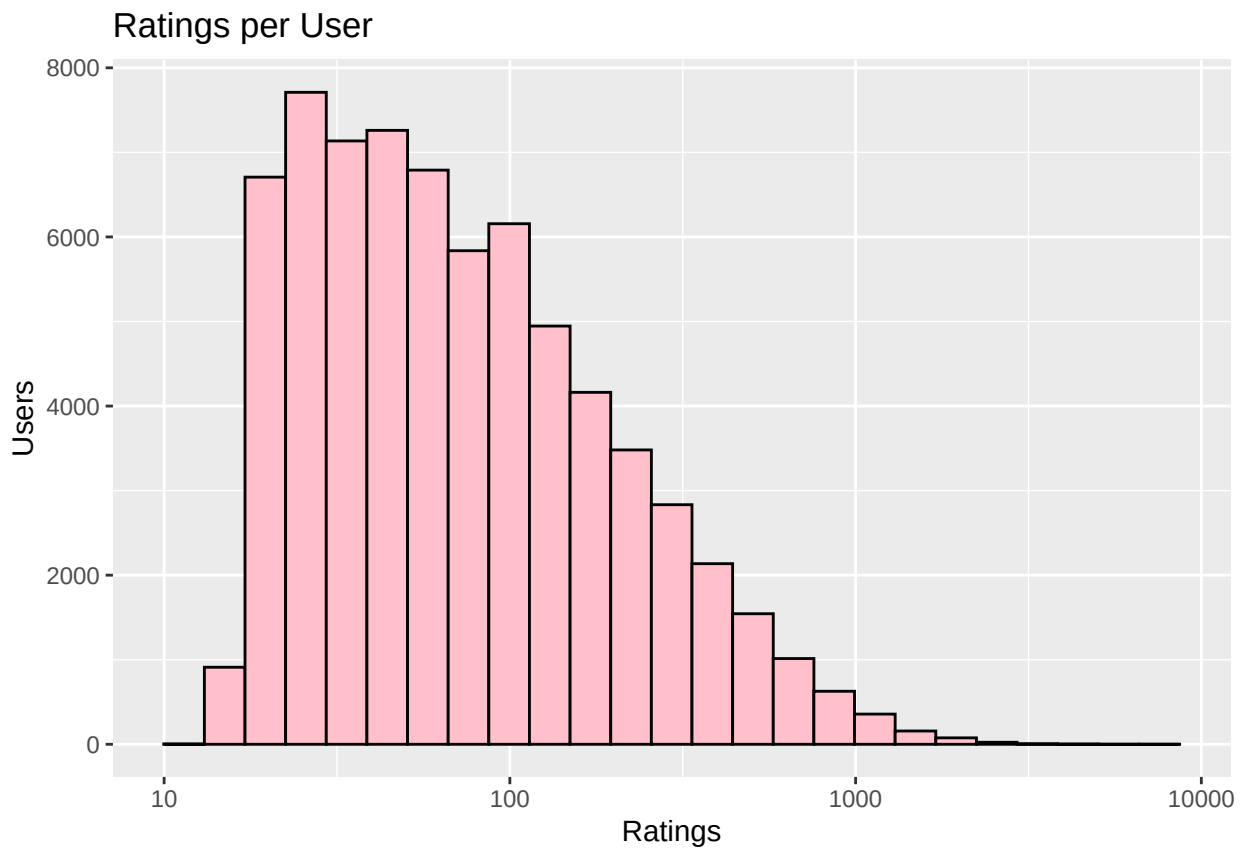


Other graphs that serve as an orientation to the algorithm model are the following.

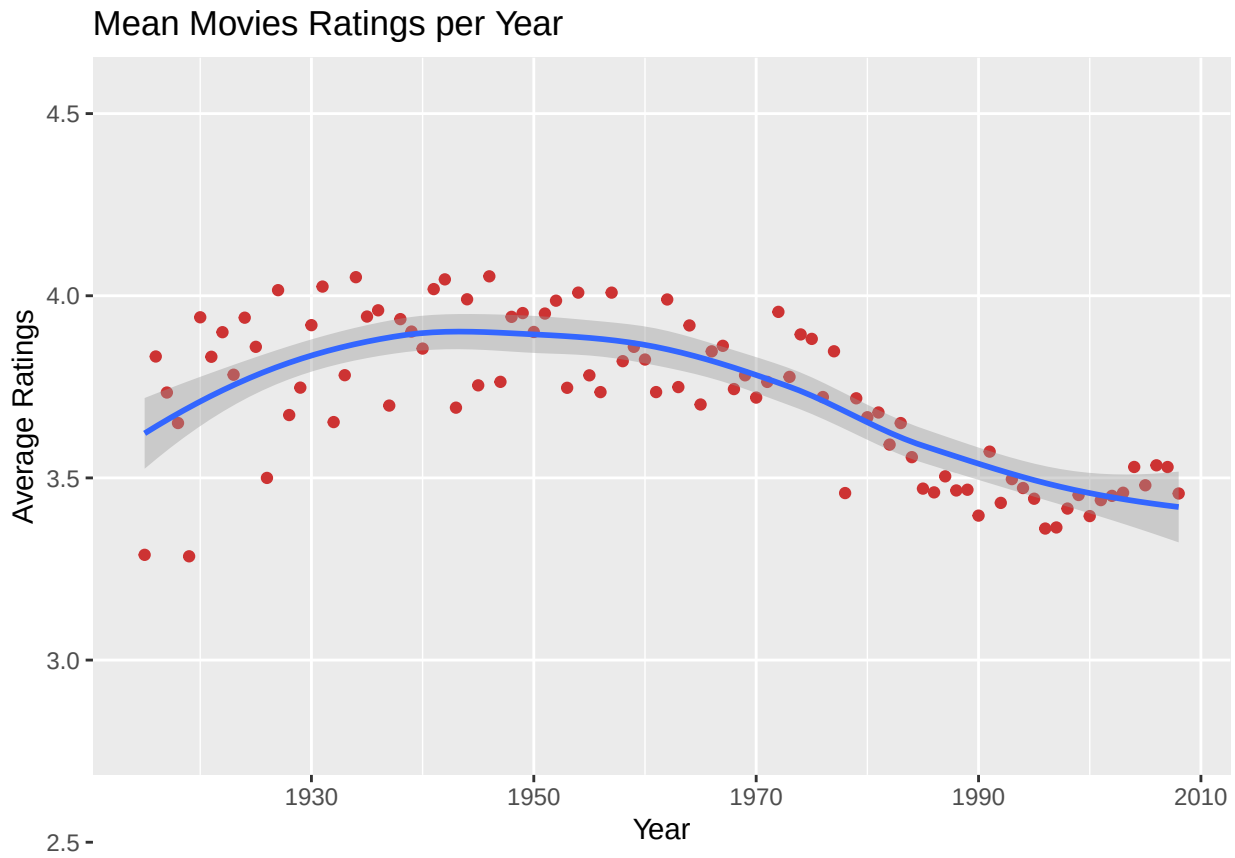
```
#Ratings per movie
edx2 %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins=25, color= "black", fill="orange")+
  scale_x_log10()+
  xlab("Ratings")+
  ylab("Movies")+
  ggtitle("Ratings per Movie")
```



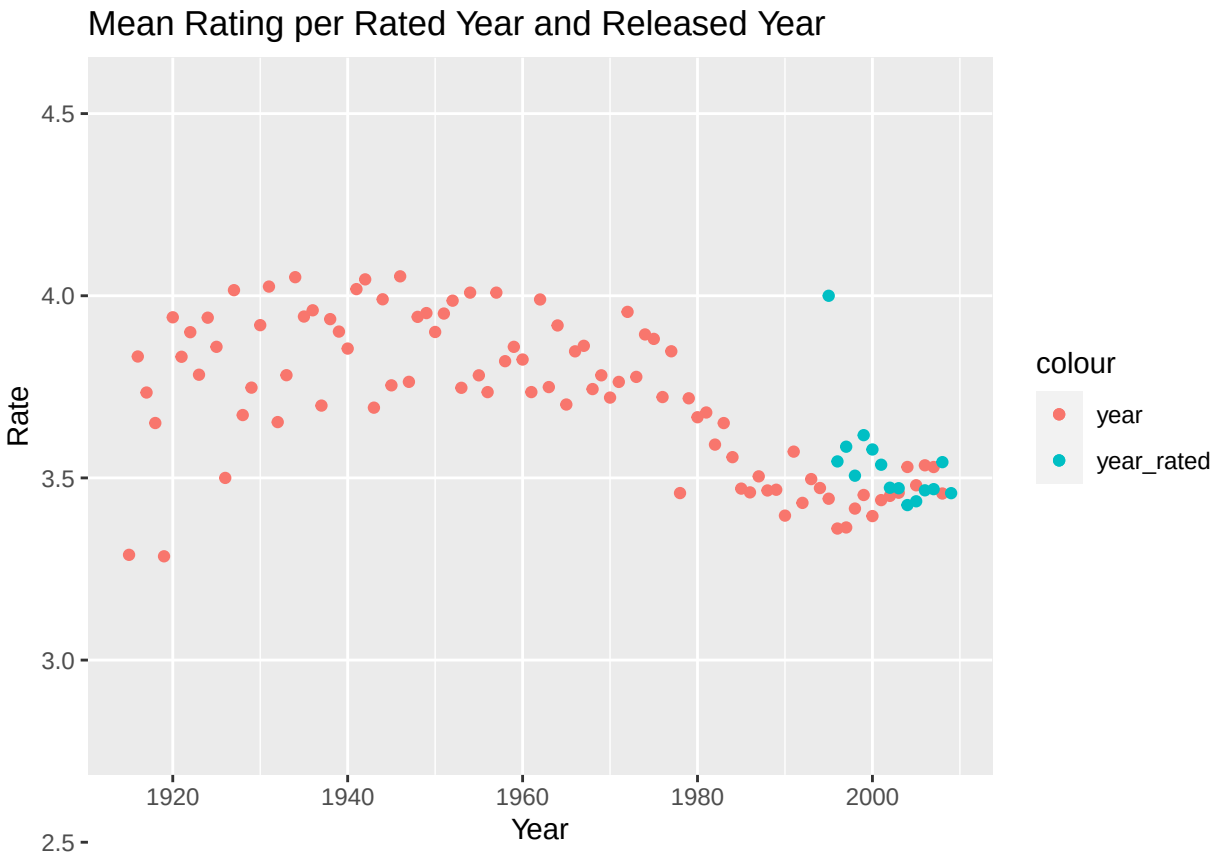

```
#Ratings per user
edx2 %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins=25, color= "black", fill="pink")+
  scale_x_log10()+
  xlab("Ratings")+
  ylab("Users")+
  ggtitle("Ratings per User")
```




```
#Mean movies ratings per year
edx2 %>%
  group_by(year) %>%
  summarize(avg = mean(rating)) %>%
  ggplot(aes(year, avg)) +
  geom_point(color = "brown3")+
  scale_y_discrete(limits = c(seq(0.5,5,0.5)))+
  geom_smooth()+
  xlab("Year")+
  ylab("Average Ratings")+
  ggtitle("Mean Movies Ratings per Year")
```

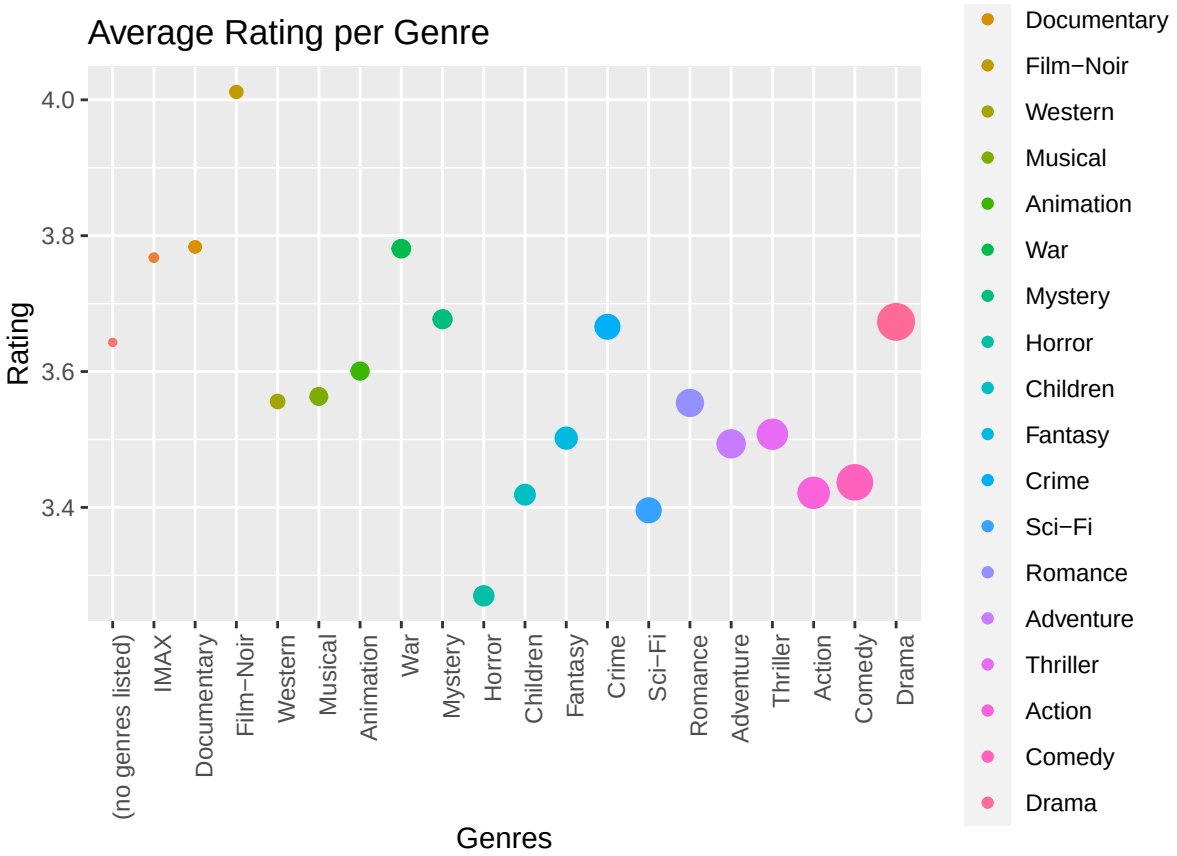



```
#Mean Rating per Rated Year and Released Year
ggplot() +
  geom_point(data = edx2 %>%
    group_by(year) %>%
    summarize(avg = mean(rating), number = n()),
    aes(year, avg, col="year"))+
  geom_point(data = edx2 %>%
    group_by(year_rated) %>%
    summarize(avg = mean(rating), number = n()),
    aes(year_rated, avg, col="year_rated")) +
  ggtitle("Mean Rating per Rated Year and Released Year")+
  scale_y_discrete(limits = c(seq(0.5,5,0.5))) +
    xlab("Year")+
    ylab("Rate")
```



We can observe that one of the main challenges is to minimize the rating of under-voted films, either because they belong to the released decades or genres with the fewest views; or because, in general, some films have received little evaluation regardless of their decade of production. In this sense, we can see how there is a correlation between a higher average of votes per movie gender, the lower the number of votes cast:

```
#Mean Rating per Genre
sep_genres <- edx2 %>% separate_rows(genres, sep ="\\|")

num_genres <- sep_genres %>% group_by(genres) %>%
  summarize(totalratings = n(), averagerating = mean(rating)) %>%
  arrange(desc(averagerating))


num_genres %>% mutate(genres = reorder(genres, totalratings)) %>%
  ggplot(aes(x= genres, y = averagerating, color = genres)) +
  geom_point(aes(size=totalratings)) +
  ggtitle("Average Rating per Genre") +
  xlab("Genres")+
  ylab("Rating") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  guides(size=FALSE)
```



3.PREDICTIONS AND RESULTS

The general approach to defining best in machine learning is to define a *loss function*, which can be applied to both categorical and continuous data.The most commonly used *loss function* is the squared loss function:

Because we have a validation set with many observations, say N, we will use the MSE. For its part, RMSE it is just the square of MSE. In binary outcomes, both RMSE and MSE are equivalent to accuracy. Our goal is to build an algorithm that minimizes the loss so it is as close to 0 as possible.

For that, we will test four generative models with RMSE as a measure to evaluate our algorithm performance until reaching a RMSE value equal to or less than 0.86490.

3.1 Simple Mean Ratings Prediction Model.

The first one it is the simplest one, because it assumes the same rating for all movies and users with all the differences explained by random variation:

Yu,i = mu + eu,i

```
#Simple Mean Ratings prediction

mu <- mean(edx2$rating)
mu
```

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

```
naive_RMSE <- RMSE(validation2$rating, mu)
naive_RMSE
```

```
## [1] 1.060331
```

```
RMSE_results = data.frame(Method = "Naive by Mean", RMSE = naive_RMSE)
RMSE_results
```

```
##          Method     RMSE
## 1 Naive by Mean 1.060331
```

3.2 Bias Effects Prediction Model

In our exploratory analysis, we confirm that some movies are just generally rated higher than others. We can augment our previous model by adding the term bi to represent average ranking for movie i.

Yu,i = mu + bi + eu,i

So, in this case we are going to apply Bias term for each movie, based on difference between movies mean and overall mean rating.

```
#Bias Movies & users Effects Model

avgs <- edx2 %>% group_by(movieId) %>%
  summarise(bi = mean(rating - mu))

avgs
```

```
## # A tibble: 10,677 x 2
##    movieId      bi
##      <dbl>   <dbl>
##  1       1  0.415
##  2       2 -0.307
##  3       3 -0.365
##  4       4 -0.648
##  5       5 -0.444
##  6       6  0.303
##  7       7 -0.154
##  8       8 -0.378
##  9       9 -0.515
## 10      10 -0.0866
## # ... with 10,667 more rows
```

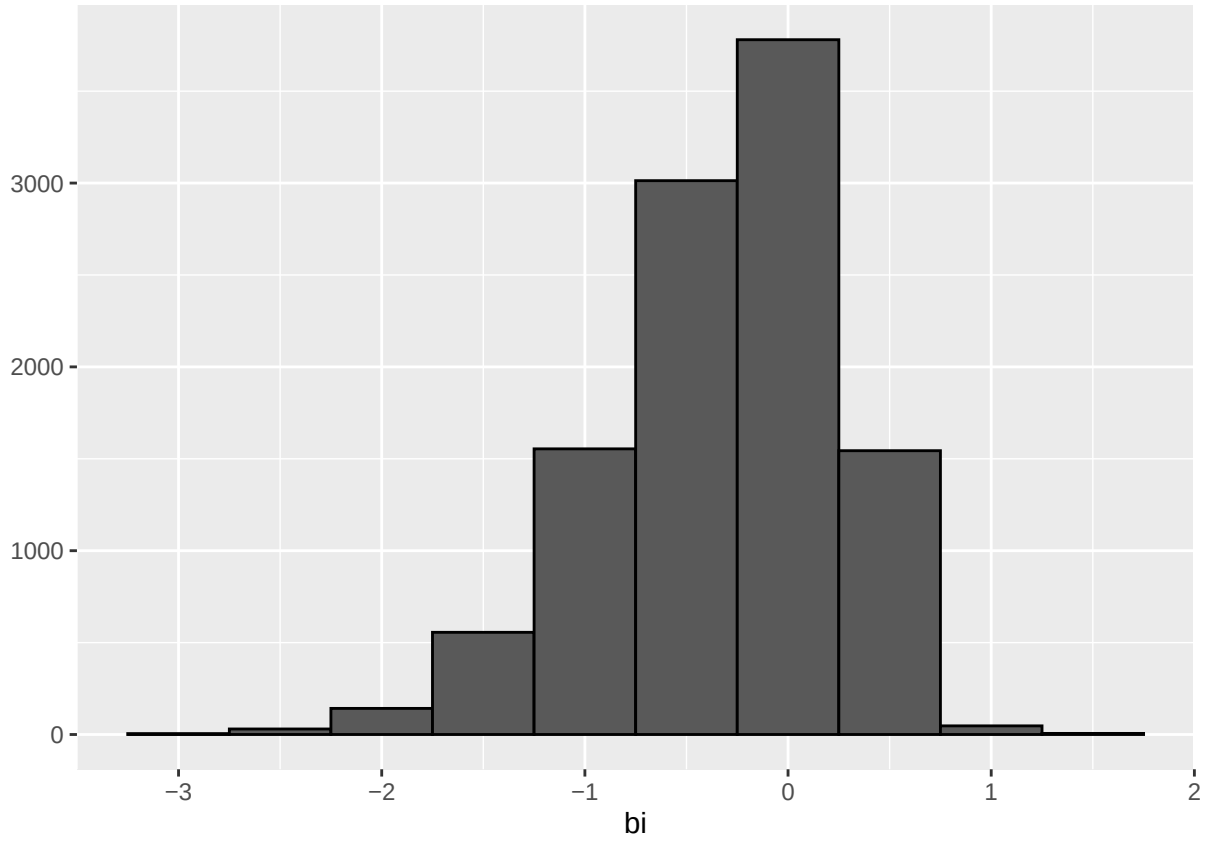
```
predictions <- mu + validation2 %>%
  left_join(avgs, by="movieId") %>%
  pull(bi)

RMSE_effect_model<- RMSE(predictions, validation2$rating)
RMSE_effect_model
```

```
## [1] 0.9423475
```

```
RMSE_results <-  bind_rows(RMSE_results,
                           data.frame(Method = "Bias Effect Model",
                                      RMSE = RMSE_effect_model))
RMSE_results %>% knitr::kable()
```

Method	RMSE
Naive by Mean	1.0603313
Bias Effect Model	0.9423475

```
qplot(bi, data = avgs, bins = 10, color = I("black"))
```



3.3 Bias with Movie & User Effects Model

Now, we are go to incorporate User Bias Effects Model to previous Bias Movie Effects Model:

```
#Bias with Movie & User Effects Model

avgs_us <- edx2 %>% group_by(userId) %>%
  left_join(avgs, by = "movieId") %>%
  summarise(bu = mean(rating - mu - bi))

avgs_us
```

```
## # A tibble: 69,878 x 2
##    userId      bu
##     <int>   <dbl>
##  1      1  1.68
##  2      2 -0.236
##  3      3  0.264
##  4      4  0.652
##  5      5  0.0853
##  6      6  0.346
##  7      7  0.0238
##  8      8  0.203
##  9      9  0.232
## 10     10  0.0833
## # ... with 69,868 more rows
```

```
predictions_2 <- validation2 %>%
  left_join(avgs, by = "movieId") %>%
  left_join(avgs_us, by = "userId") %>%
  mutate(pre_bi_bu = mu + bi + bu) %>%
  pull(pre_bi_bu)

RMSE_usermovie_effect_model <- RMSE(predictions_2, validation2$rating)

RMSE_usermovie_effect_model
```

```
## [1] 0.8567039
```

```
RMSE_results <- bind_rows(RMSE_results,
                          data.frame(Method="Bias User & Movie Effect Model",
                                     RMSE = RMSE_usermovie_effect_model))

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

Method	RMSE
Naive by Mean	1.0603313
Bias Effect Model	0.9423475
Bias User & Movie Effect Model	0.8567039

With this model we have achieved our goal, but we will apply the Regularization method to compare if we can improve the RSME result.

3.4 Regularization and Cross Validation

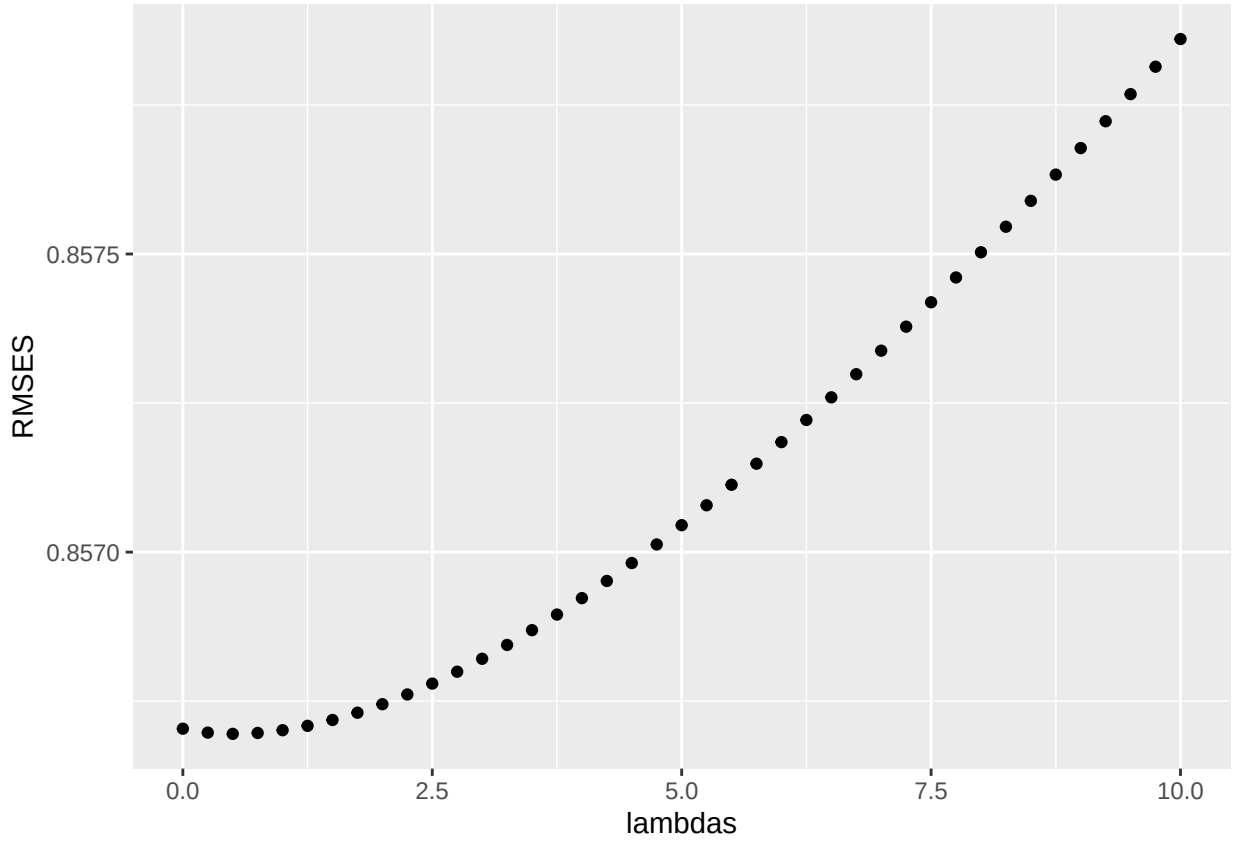
Regularization penalizes noise and outliers in data. Taking into account that, as we have been able to verify in the visualization and exploration of the data, there are movies viewed by few users and, therefore, voted by few of them, we can apply *lambda* as a parameter tuner in order to minimize the RSME value.

```
lambdas <- seq(0, 10, 0.25)

RMSES <- sapply(lambdas, function(l){
```

```
  mu <- mean(edx2$rating)
  bi <- edx2 %>%
    group_by(movieId) %>%
    summarize(bi = sum(rating - mu)/(n()+l))
  bu <- edx2 %>%
    left_join(bi, by="movieId") %>%
    group_by(userId) %>%
    summarize(bu = sum(rating - bi - mu)/(n()+l))
  predicted_ratings <-
    validation2 %>%
    left_join(bi, by = "movieId") %>%
    left_join(bu, by = "userId") %>%
    mutate(pred = mu + bi + bu) %>%
    pull(pred)
  return(RMSE(predicted_ratings, validation2$rating))
})
```

Here we can view RMSE vs lambdas in order to select the optimal lambdas value:

```
qplot(lambdas, RMSES)
```




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

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

So, here we have the complete table with all the RMSE results per method:

```
RMSE_results <- bind_rows(RMSE_results,
                          data.frame(Method="Regularized Model",
                                     RMSE = min(RMSES)))

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

Method	RMSE
Naive by Mean	1.0603313
Bias Effect Model	0.9423475
Bias User & Movie Effect Model	0.8567039
Regularized Model	0.8566952

As we can see in the summary table, *Regularization Model* has not substantially improved the results obtained with the *Bias with Movie & User Effects Model*.

4. CONCLUSION

We have reached the objective established in the project bases: We have constructed a machine learning algorithm to predict the ratings from the Movielens dataset with a RSME value lower than 0.87750. The two optimal models with RSME values lower than a goal were *Bias User & Movie Effects* and *Regularized Model*.