Movie Lens ML Project

Dario Taba

01/03/2021

1 Introduction

Filtering system is the one that is responsible for cleaning redundant or unwanted information, trying to avoid the overload of irrelevant information. From here derive recommendation systems, which try to predict a rating or user preference on a content or object. These systems today are very comprehensive, ranging from the generation of music playlists, to recommendation on e-commerce platforms and search engines.

In our case, we will focus on movie recommendation systems. Recommendation systems are used by various multimedia content companies, such as Netflix, YouTube or Amazon Prime Video, to be able to decide the relevant content for their users and to be able to retain attention on their platforms for as long as possible.

1.1 Movie Recomendation

We will create a model that can predict the rating of a movie based on characteristics in the database.

The database will be MovieLens. MovieLens is a research site run by GroupLens Research at the University of Minnesota. The base has 10 million movie ratings by 72 thousand users. Users were randomly selected, having voted at least 20 times (https://grouplens.org/datasets/movielens/10m/).

1.2 Data Preparation for the model and Pre analysis

1.2.1 Getting the base database

The initial database is obtained using the following script. This code was provided at the beginning of the current course.

```
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(recosystem)) install.packages("recosystem") # For matrix factorization
if(!require(tinytex)) install.packages("tinytex")

# For making the pdf
if(!require(rmarkdown)) install.packages("rmarkdown")
#if(!require(proTeXt)) install.packages("proTeXt")
if(!require(formatR)) install.packages("formatR")
if(!require(latexpdf)) install.packages("latexpdf")
if(!require(xfun)) install.packages("xfun")
#if(!require(proTeXt)) install.packages("MikTek")
#tinytex::install_tinytex()
```

```r
options(pillar.sigfig=7)

library(tidyverse)
library(caret)
library(data.table)
library(recommenderlab)
library(recosystem)
library(ggthemes)
#library(MikTek)
library(latexpdf)
library(xfun)
library(tinytex)
library(rmarkdown)
library(formatR)
#library(proTeXt)


# 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 3.6 or earlier:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(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)
edx1 <- edx # to not use the original edx
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Before starting the analysis of the variables, we will create one more referencing the year of the movie.

```
edx_clean <-
  edx1 %>%
  mutate(title = str_trim(title)) %>% # to not have problems with white spaces
  mutate(year_title_temp = str_extract_all(title,pattern="\\((\\d{4})\\)"),year_title = as.integer(str_
  select(-year_title_temp)
```

With this code we obtain `edx_clean`, with which we will make the model, and `validation`, with which we will do the final validation.

1.2.2 First approach to the database

First we will execute a summary of the database.

```
str(edx_clean)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  7 variables:
##  $ userId    : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId   : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating    : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp : int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 8
##  $ title     : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres    : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
##  $ year_title: int  1992 1995 1995 1994 1994 1994 1994 1994 1994 1994 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

The variables are: userId (Integer), movieId (Integer), rating (Numeric), timestamp (Integer, should be date), title (character), genres (character) and year_title (integer).

We can see with this the 7 columns of the table, together with their type, where 9.000.055 is the total number of records in the `edx_clean` database.

1.2.2.1 UserId

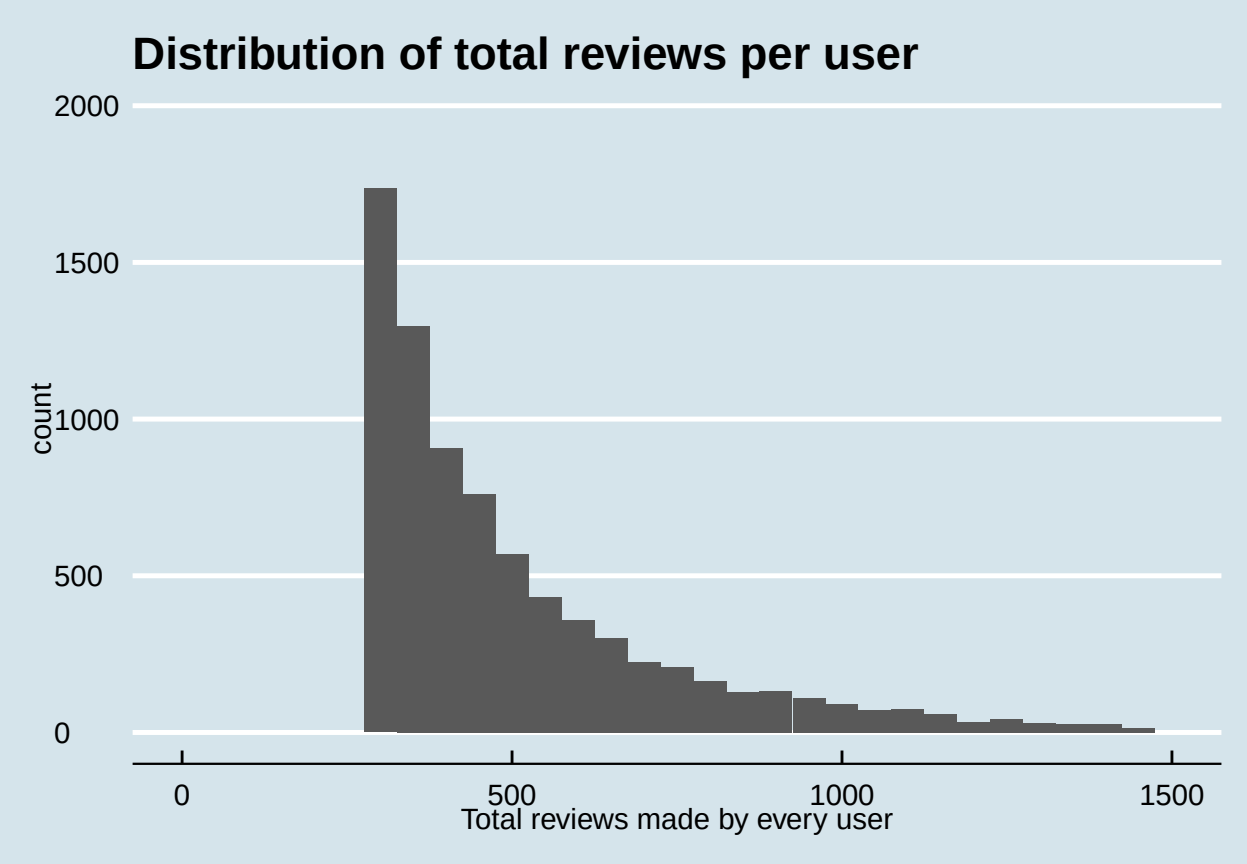
The first variable is userId. It is the identification number of the user who made the rate. In our dataset, we have 69.878 users.

```
edx_clean %>%
  group_by(userId) %>%
  summarise(n=n()) %>%
  arrange((desc(n)))
```

```
## # A tibble: 69,878 x 2
##    userId     n
##     <int> <int>
##  1  59269  6616
##  2  67385  6360
##  3  14463  4648
##  4  68259  4036
##  5  27468  4023
##  6  19635  3771
##  7   3817  3733
##  8  63134  3371
##  9  58357  3361
## 10  27584  3142
## # ... with 69,868 more rows
```

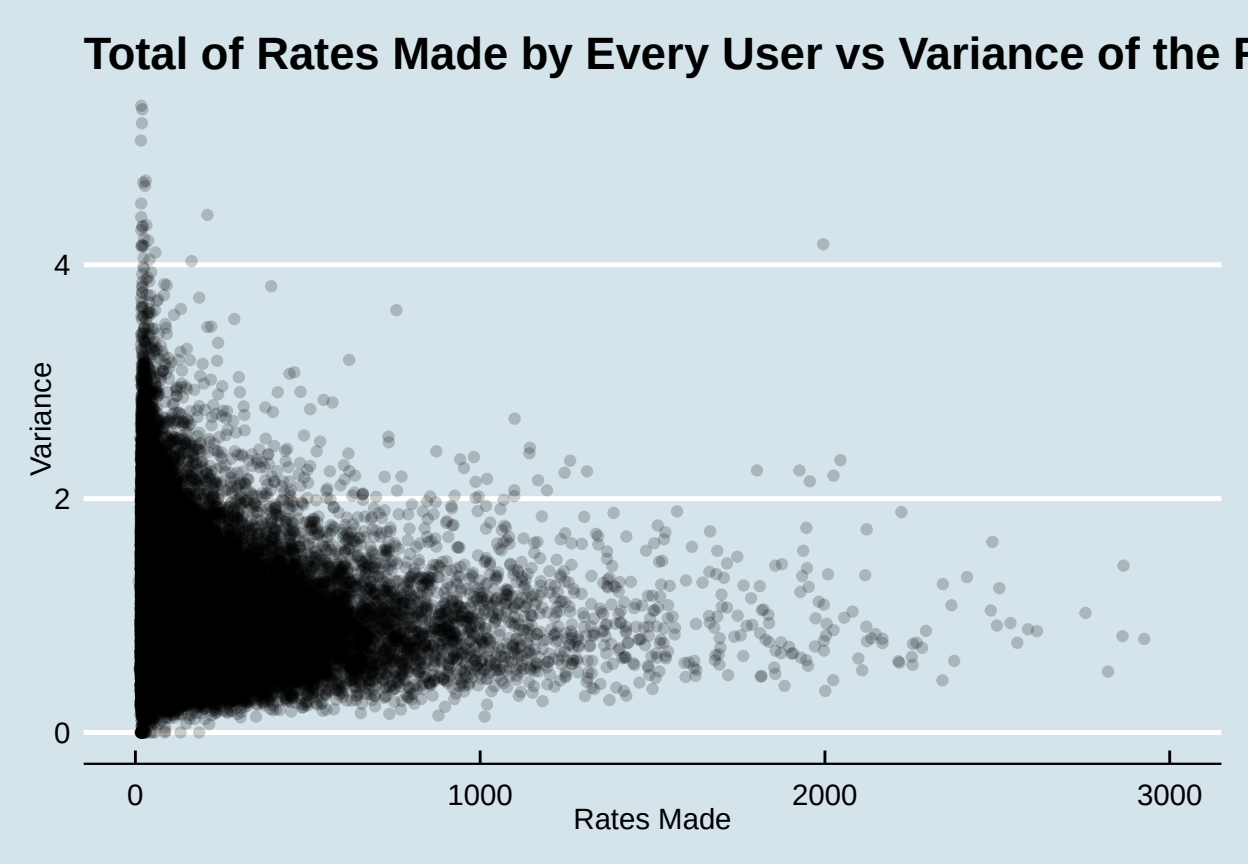
With the following graph we can see the distribution of the rates made. We can see that it is distributed to the left, showing that most users accumulate in the first bars.

```
edx_clean %>%
  group_by(userId) %>%
  summarise(n=n()) %>%
  arrange((desc(n))) %>%
  ggplot(aes(n)) +
  geom_histogram(binwidth = 50) +
  xlim(0,1500) + #Beeing 0.2% who made more than 1500 reviews
  ylim(0,2000)+
  xlab("Total reviews made by every user") +
  theme_economist() +
  ggtitle("Distribution of total reviews per user")
```



Here is a graph that represents the number of rates made by each userId, against the variance of the rates. What can be observed is that as we have more records of each userId, we can see that there are people who tend to score movies on the same values, decreasing the variance, demonstrating the existence of a user effect.

```
edx_clean %>%
  group_by(userId) %>%
  summarise(n=n(),rating_avg=mean(rating),var_ratings = var(rating)) %>%
  ggplot(aes(n,var_ratings)) +
  geom_point(alpha=0.2)+ # Making it more visual where is most of the data
  xlim(0,3000)+
  ggtitle("Total of Rates Made by Every User vs Variance of the Ratings")+
  xlab("Rates Made")+
  ylab("Variance")+
  theme_economist()
```



1.2.2.2 MovieId

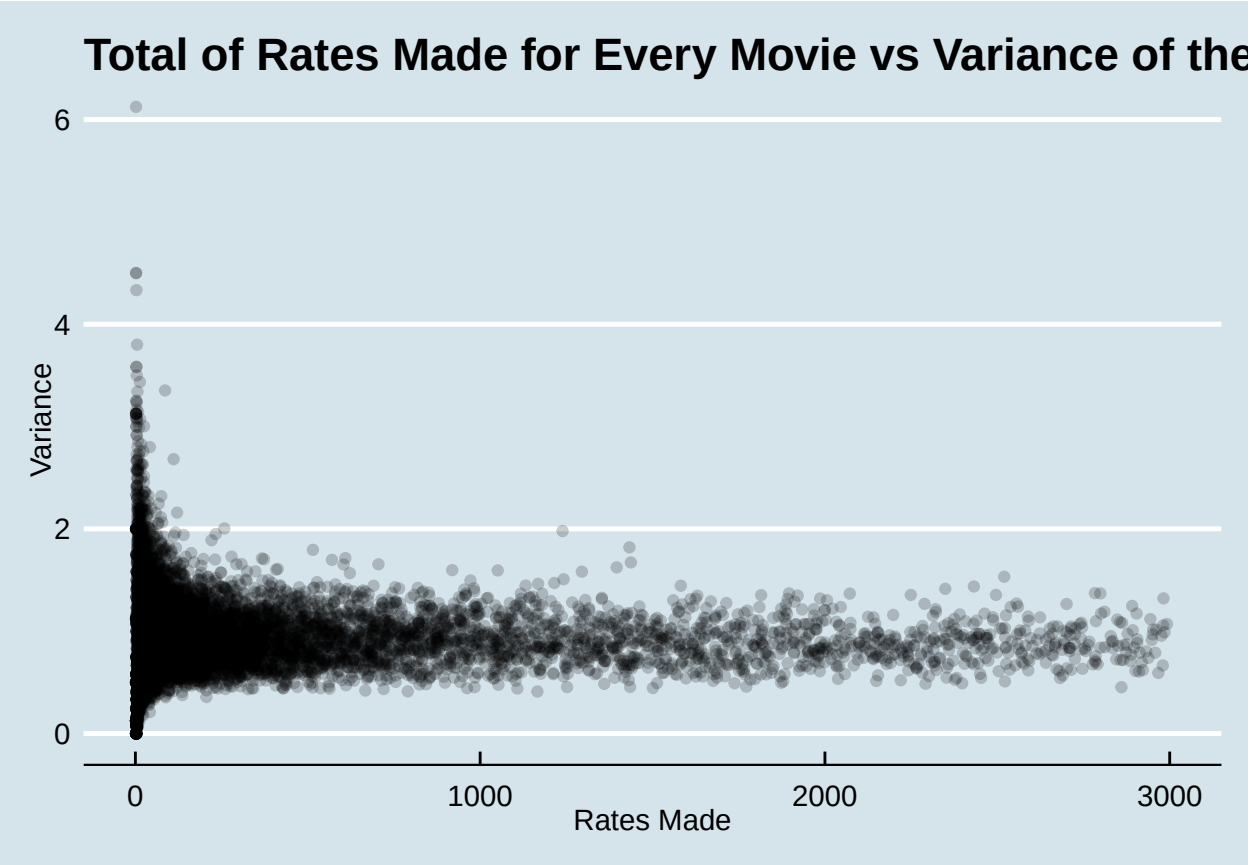
MovieId is the code of every movie rated in the dataset. Movielens provides us with ratings information for 10,677 movies.

```
edx_clean %>%
  group_by(movieId) %>%
  summarise(n=n()) %>%
  arrange((desc(n)))
```

```
## # A tibble: 10,677 x 2
##    movieId     n
##      <dbl> <int>
##  1     296 31362
##  2     356 31079
##  3     593 30382
##  4     480 29360
##  5     318 28015
##  6     110 26212
##  7     457 25998
##  8     589 25984
##  9     260 25672
## 10     150 24284
## # ... with 10,667 more rows
```

In the case of movies it is more direct to think that there is a relationship with their rating. In the following graph it can be seen that as each movie has more ratings, the variance begins to stabilize and decrease, demonstrating with data the relationship.

```
edx_clean %>%
  group_by(movieId) %>%
  summarise(n=n(),rating_avg=mean(rating),var_ratings = var(rating)) %>%
  ggplot(aes(n,var_ratings)) +
  geom_point(alpha=0.2)+ # Making it more visual where is most of the data
  xlim(0,3000)+
  ggtitle("Total of Rates Made for Every Movie vs Variance of the Ratings")+
  xlab("Rates Made")+
  ylab("Variance")+
  theme_economist()
```



1.2.2.3 Genres

There are 20 genres in the dataset, and combinations of them, making 797 possibilities.

```
edx_clean %>%
  group_by(genres) %>%
  summarise(n=n()) %>%
  arrange((desc(n)))
```

```
## # A tibble: 797 x 2
```

```
##    genres                          n
##    <chr>                       <int>
##  1 Drama                      733296
##  2 Comedy                     700889
##  3 Comedy|Romance             365468
##  4 Comedy|Drama               323637
##  5 Comedy|Drama|Romance       261425
##  6 Drama|Romance              259355
##  7 Action|Adventure|Sci-Fi    219938
##  8 Action|Adventure|Thriller 149091
##  9 Drama|Thriller             145373
## 10 Crime|Drama                137387
## # ... with 787 more rows
```

Calculating the average, we can see differences between the different genres, a characteristic that we will use later to build the model.

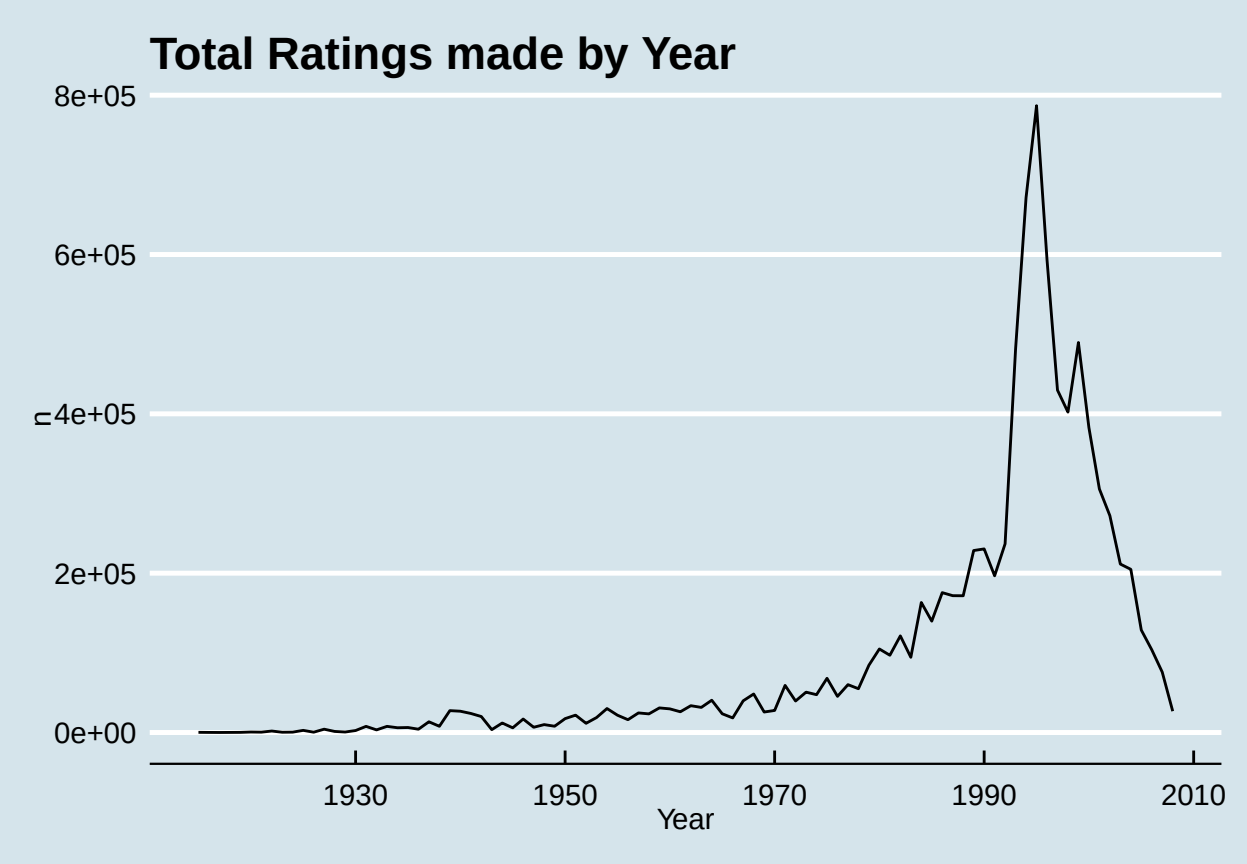
```
edx_clean %>%
  group_by(genres) %>%
  summarise(n=n(),avg_rating=mean(rating)) %>%
  arrange(desc(n))
```

```
## # A tibble: 797 x 3
##    genres                          n avg_rating
##    <chr>                       <int>      <dbl>
##  1 Drama                      733296   3.712364
##  2 Comedy                     700889   3.237858
##  3 Comedy|Romance             365468   3.414486
##  4 Comedy|Drama               323637   3.598961
##  5 Comedy|Drama|Romance       261425   3.645824
##  6 Drama|Romance              259355   3.605471
##  7 Action|Adventure|Sci-Fi    219938   3.507407
##  8 Action|Adventure|Thriller 149091   3.434101
##  9 Drama|Thriller             145373   3.446345
## 10 Crime|Drama                137387   3.947135
## # ... with 787 more rows
```

1.2.2.3 Year_title

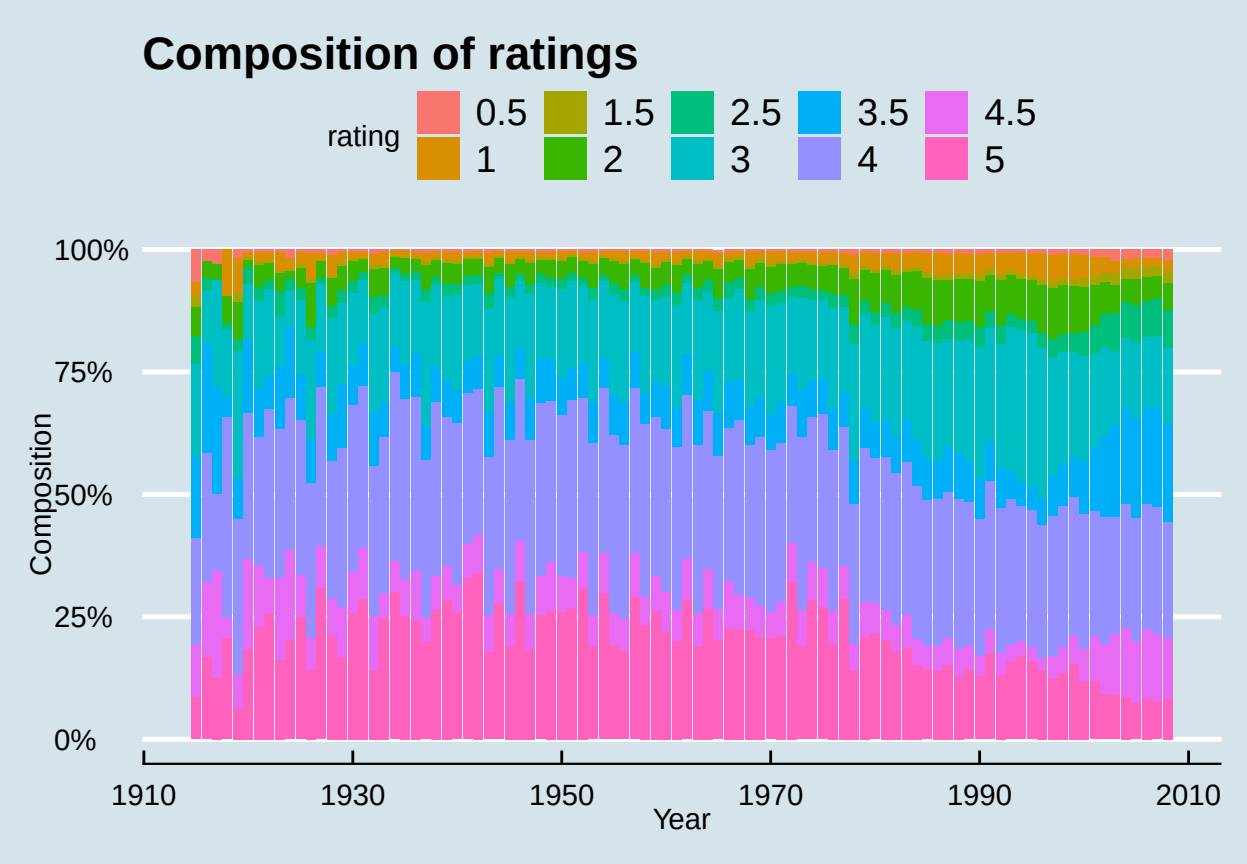
This variable indicates the year the movie was released. There are movies from 1915 to 2008.

```
edx_clean %>%
  group_by(year_title) %>%
  summarise(n=n()) %>%
  ggplot(aes(year_title,n)) +
  geom_line()+
  theme_economist()+
  xlab("Year")+
  ylab("n")+
  ggtitle("Total Ratings made by Year")
```



The composition by rating of the previous graph.

```
edx_clean %>%
  group_by(year_title,rating) %>%
  summarise(n=n()) %>%
  mutate(rating = as.factor(rating))%>%
  ggplot(aes(year_title,n,fill=rating))+
  geom_bar(position="fill", stat="identity")+
  scale_y_continuous(labels = scales::percent_format())+
  theme_economist()+
  xlab("Year")+
  ylab("Composition")+
  ggtitle("Composition of ratings")
```



2 Methods and Analysis

We will use different methods to generate models and they will be compared to see the predictability of each one. In order to quantify and measure predictability we will use the RMSE of each one, being a frequently used measure of the differences between the values predicted by a model or an estimator and the observed values.

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

We will generate, within the edx_clean dataset, a partition to train the model and another to test it.Thanks to the size of our dataset, the partition will be 90/10, being 90% for training and 10% for testing. This leaves us with the following 3 objects: validation_final, test_set and train_set.

```
#Creating the partition of the edx_clean dataset. We'll use 90% for the training of the model and 10% f
#This ratio is possible thanks to the large amount of data
test_index <- createDataPartition(edx_clean$rating,p=0.1,list=FALSE)


train_set <- edx_clean[-test_index,]
test_set_temp <- edx_clean[test_index,]
validation_temp <- validation
```

```
# Matching the dataset variables for the test set
test_set <- test_set_temp %>%
  semi_join(train_set,by="movieId") %>%
  semi_join(train_set,by="userId") %>%
  semi_join(train_set,by="genres") %>%
  semi_join(train_set,by="year_title")

# Matching the dataset variables for the validation set
validation_final <-
  validation_temp %>%
  # select(title) %>%
  #  head(n = 1) %>%
  mutate(title = str_trim(title)) %>%
  mutate(year_title_temp = str_extract_all(title,pattern="\\((\\d{4})\\)"),year_title = as.integer(str_
  select(-year_title_temp) %>%
  semi_join(train_set,by="movieId") %>%
  semi_join(train_set,by="userId") %>%
  semi_join(train_set,by="genres") %>%
  semi_join(train_set,by="year_title")
```

2.1 Just the average rating

What we do in this first approximation is to explain the ratings based on the general average, explaining the differences due to random variations.

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

```
mu_hat <-mean(train_set$rating) #average of all ratings

predicted_avg <- rep(mu_hat,times= nrow(test_set)) #the predictions, all the same

RMSE_avg<- RMSE(predicted_avg,test_set$rating)

RMSEs_Table <- tibble(Method = "Goal", RMSE = 0.8649)
RMSEs_Table <- bind_rows(RMSEs_Table,
                    tibble(Method = "Just the average of ratings",RMSE=RMSE_avg))

RMSEs_Table
```

```
## # A tibble: 2 x 2
##   Method                          RMSE
##   <chr>                          <dbl>
## 1 Goal                          0.8649
## 2 Just the average of ratings 1.061135
```

As we can see, we still obtain an RMSE very far from the ideal.

2.2 Adding the effect of other variables

2.2.1 Adding the effect of movieId

To reduce the participation of random variation, we are going to explain part of it by the movieId effect, leaving our model as follows.

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

Using this new addition, our model would be:

```
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_movie = mean(rating - mu_hat)) # substracting the part explained by the total average

predicted_avg_movie <- test_set %>%
  left_join(movie_avgs,by='movieId') %>%
  mutate(prediction = mu_hat+b_movie) %>%
  .$prediction


RMSE_avg_movie<- RMSE(predicted_avg_movie,test_set$rating)

RMSEs_Table <- bind_rows(RMSEs_Table,
                    tibble(Method = "Avg + movie effect",RMSE=RMSE_avg_movie))
RMSEs_Table
```

```
## # A tibble: 3 x 2
##   Method                          RMSE
##   <chr>                          <dbl>
## 1 Goal                         0.8649
## 2 Just the average of ratings 1.061135
## 3 Avg + movie effect          0.9441568
```

2.2.2 Adding the effect of userId

Following the previous logic, we will add the userId variable to explain part of the previous random variation.

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

```
#making the averages for every user
user_avgs <- train_set %>%
  left_join(movie_avgs,by='movieId') %>%
  group_by(userId) %>%
  summarize(b_user = mean(rating - mu_hat - b_movie)) # substracting the part explained by the total av

predicted_avg_movie_user <- test_set %>%
  left_join(movie_avgs,by='movieId') %>%
  left_join(user_avgs,by='userId') %>%
  mutate(prediction = mu_hat+b_movie+b_user) %>%
  .$prediction


RMSE_avg_movie_user<- RMSE(predicted_avg_movie_user,test_set$rating)
```

```
RMSEs_Table <- bind_rows(RMSEs_Table,
                    tibble(Method = "Avg + movie and user effect",RMSE=RMSE_avg_movie_user))
RMSEs_Table
```

```
## # A tibble: 4 x 2
##   Method                            RMSE
##   <chr>                            <dbl>
## 1 Goal                           0.8649
## 2 Just the average of ratings 1.061135
## 3 Avg + movie effect           0.9441568
## 4 Avg + movie and user effect 0.8659736
```

2.2.3 Adding the effect of genres

In the case of genres, we can see that each rating can have a movie with several genres. What we will do is separate the genres in order to have complete information on each genre. In the case of compound genres, the base will not be divided.

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

```
#making the averages for every user
genre_avg_more_than_one_avg <- train_set %>%
  filter(str_detect(genres,"\\|")==TRUE) %>% #movies with more than one genre
  left_join(movie_avgs,by='movieId') %>%
  left_join(user_avgs,by='userId') %>%
  group_by(genres) %>%
  summarize(b_genre = mean(rating - mu_hat - b_user - b_movie)) # substracting the part explained by th

#making the averages for every genre
genre_avg_per_genre_avgs <- train_set %>%
  left_join(movie_avgs,by='movieId') %>%
  left_join(user_avgs,by='userId') %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(b_genre = mean(rating - mu_hat - b_user - b_movie)) # substracting the part explained by th

genre_avgs <- bind_rows(genre_avg_more_than_one_avg,genre_avg_per_genre_avgs)

predicted_avg_movie_user_genres <- test_set %>%
  left_join(movie_avgs,by='movieId') %>%
  left_join(user_avgs,by='userId') %>%
  left_join(genre_avgs,by='genres') %>%
  mutate(prediction = mu_hat+b_movie+b_user+b_genre) %>%
  .$prediction


RMSE_avg_movie_user_genres<- RMSE(predicted_avg_movie_user_genres,test_set$rating)

RMSEs_Table <- bind_rows(RMSEs_Table,
                    tibble(Method = "Avg + movie, user and genres effect",RMSE=RMSE_avg_movie_user_genr
RMSEs_Table
```

```
## # A tibble: 5 x 2
##   Method                                   RMSE
##   <chr>                                   <dbl>
## 1 Goal                                  0.8649
## 2 Just the average of ratings           1.061135
## 3 Avg + movie effect                    0.9441568
## 4 Avg + movie and user effect           0.8659736
## 5 Avg + movie, user and genres effect 0.8656152
```

2.2.4 Adding the effect of year_title

Finally, we will incorporate the last variable to the model, the year of the movie.

$$\hat{Y}_{u,i} = \mu + b_m + b_u + b_g + b_y + \epsilon_{i,u}$$

```
#making the averages for every user
year_avgs <- train_set %>%
  left_join(movie_avgs,by='movieId') %>%
  left_join(user_avgs,by='userId') %>%
  left_join(genre_avgs,by='genres') %>%
  group_by(year_title) %>%
  summarize(b_year = mean(rating - mu_hat - b_movie - b_user - b_genre)) # substracting the part explai

predicted_avg_movie_user_genres_year <- test_set %>%
  left_join(movie_avgs,by='movieId') %>%
  left_join(user_avgs,by='userId') %>%
  left_join(genre_avgs,by='genres') %>%
  left_join(year_avgs,by='year_title') %>%
  mutate(prediction = mu_hat + b_movie + b_user + b_genre + b_year) %>%
  .$prediction


RMSE_avg_movie_user_genres_year<- RMSE(predicted_avg_movie_user_genres_year,test_set$rating)

RMSEs_Table <- bind_rows(RMSEs_Table,
                    tibble(Method = "Avg + movie, user, genres and year effect",RMSE=RMSE_avg_movie_use
RMSEs_Table
```

```
## # A tibble: 6 x 2
##   Method                                      RMSE
##   <chr>                                      <dbl>
## 1 Goal                                     0.8649
## 2 Just the average of ratings              1.061135
## 3 Avg + movie effect                       0.9441568
## 4 Avg + movie and user effect              0.8659736
## 5 Avg + movie, user and genres effect      0.8656152
## 6 Avg + movie, user, genres and year effect 0.8654004
```

As we can see, with each incorporation of a variable, we are being able to reduce the RMSE, leaving less and less participation to random variations.

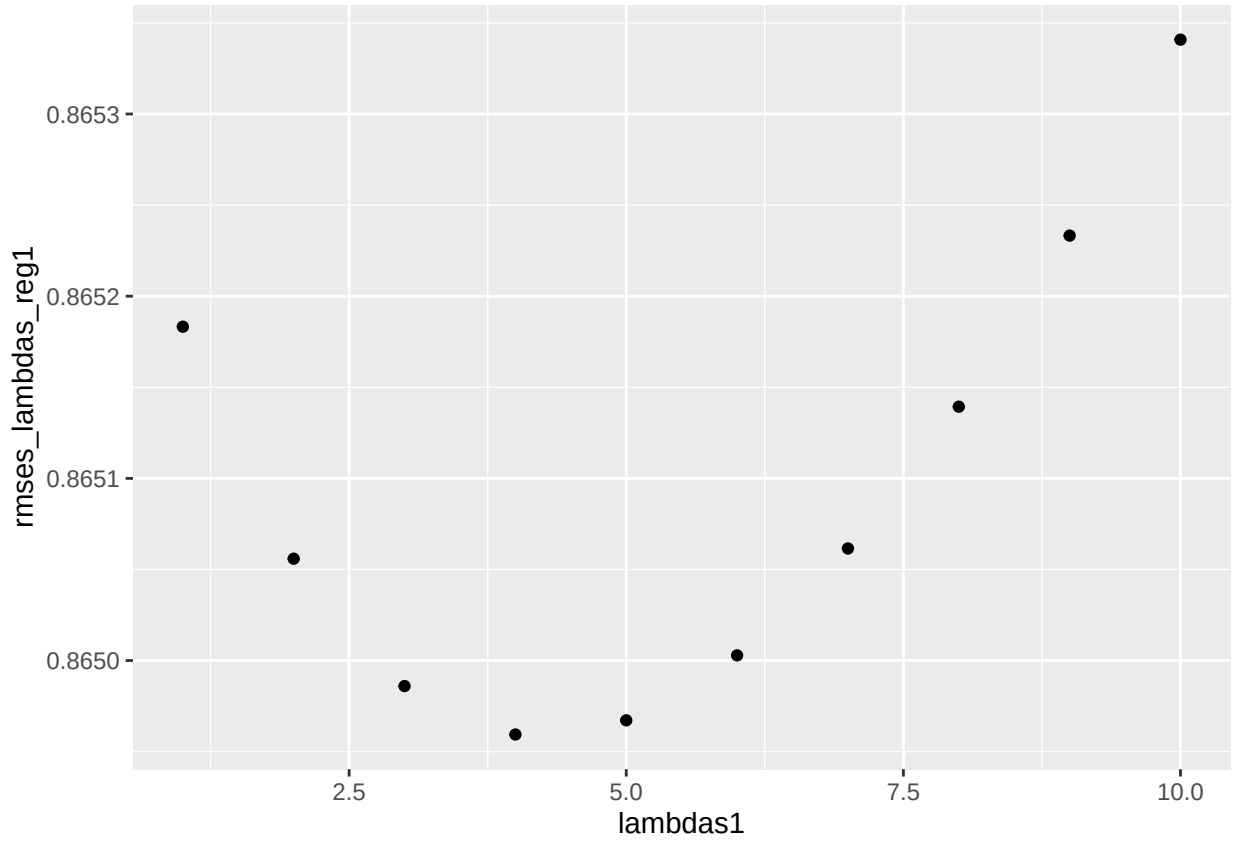
2.3 Regularization

What regularization does is penalize the information based on small samples by incorporating the lambda variable. By iterating through different lambda scenarios, the lambda that minimizes the RMSE is selected.

The structure in which the lambda is incorporated is by increasing the denominator in the averages, the higher the lambda, the more it is penalized, trying to reach an equilibrium by minimizing the RMSE.

```
lambdas1 <- seq(1,10,1) #Iteration of lambdas

# Iterating the lambdas
rmses_lambdas_reg1 <- sapply(lambdas1,function(x){
  #Using the terms that previously were done, but with some changes doing the regularization
  mu <- mean(train_set$rating)

  movie_avgs_reg1 <- train_set %>%
    group_by(movieId) %>%
    summarize(b_movie = sum(rating - mu)/(n()+x))

  user_avgs_reg1 <- train_set %>%
    left_join(movie_avgs_reg1,by='movieId') %>%
    group_by(userId) %>%
    summarize(b_user = sum(rating - mu - b_movie)/(n()+x))

  b_g_1_reg1 <- train_set %>%
    filter(str_detect(genres,"\\|")==TRUE) %>% # Knowing the separator for every genre is "|", just fil
    left_join(movie_avgs_reg1,by='movieId') %>%
    left_join(user_avgs_reg1,by='userId') %>%
    group_by(genres) %>%
    summarize(b_genres = sum(rating - mu - b_user - b_movie)/(n()+x))

  # Since separating the different genres of each film changes the original base, we will not calculate
  b_g_2_reg1 <- train_set %>%
    left_join(movie_avgs_reg1,by='movieId') %>%
    left_join(user_avgs_reg1,by='userId') %>%
    separate_rows(genres, sep = "\\|") %>%   # The separator for every genre is "|"
    group_by(genres) %>%
    summarize(b_genres = sum(rating - mu - b_user - b_movie)/(n()))

  b_g_reg1 <- bind_rows(b_g_1_reg1,b_g_2_reg1)

  b_y_reg1 <- train_set %>%
    left_join(movie_avgs_reg1,by='movieId') %>%
    left_join(user_avgs_reg1,by='userId') %>%
    left_join(b_g_reg1,by='genres') %>%
    group_by(year_title) %>%
    summarize(b_years = sum(rating - mu - b_user - b_movie-b_genres)/(n()+x))

  predicted_ratings <- test_set %>%
    left_join(movie_avgs_reg1,by='movieId') %>%
    left_join(user_avgs_reg1,by='userId') %>%
    left_join(b_g_reg1,by='genres') %>%
    left_join(b_y_reg1,by='year_title') %>%
    mutate(pred = mu + b_movie + b_user + b_genres + b_years) %>% # Making the predictions
```

```
    .$pred

  return(RMSE(predicted_ratings,test_set$rating)) # Returning just the RMSE

}
)

qplot(lambdas1,rmses_lambdas_reg1)
```




```
lambda1 <- lambdas1[which.min(rmses_lambdas_reg1)] # This lambda minimizes RMSE
```

The lambda that minimizes the RMSE is 4, with a RMSE of 0.8649594.

```
RMSEs_Table <- bind_rows(RMSEs_Table,
                         tibble(Method = "Regularization",RMSE=min(rmses_lambdas_reg1)))

RMSEs_Table
```

```
## # A tibble: 7 x 2
##   Method                                    RMSE
##   <chr>                                    <dbl>
## 1 Goal                                    0.8649
## 2 Just the average of ratings             1.061135
## 3 Avg + movie effect                      0.9441568
```

```
## 4 Avg + movie and user effect                0.8659736
## 5 Avg + movie, user and genres effect        0.8656152
## 6 Avg + movie, user, genres and year effect 0.8654004
## 7 Regularization                             0.8649594
```

As we can see, it has a negative effect to assume the information is equally true regardless of the sample in which it is held. We can improve RMSE by applying regularization.

2.3 Matrix Factorization

Matrix factoring algorithms work by decomposing the user-element interaction matrix into the product of two rectangular matrices of lesser dimension. The idea of this approach is to discover relationships between the different subgroups within the entire population.

```
#Making the train dataset
train_data_mf <-  with(train_set, data_memory(user_index = userId,item_index = movieId, rating     = ra

#Making the test dataset
test_data_mf  <-  with(test_set,data_memory(user_index = userId, item_index =movieId,rating= rating))

# object of class "RecoSys" that can be used to construct recommender model and conduct prediction
r <-  recosystem::Reco()

# Training the data
r$train(train_data_mf)
```

```
## iter      tr_rmse          obj
##    0       0.9630  1.3284e+007
##    1       0.8815  1.2014e+007
##    2       0.8595  1.1818e+007
##    3       0.8460  1.1657e+007
##    4       0.8394  1.1602e+007
##    5       0.8345  1.1561e+007
##    6       0.8306  1.1526e+007
##    7       0.8280  1.1504e+007
##    8       0.8261  1.1490e+007
##    9       0.8247  1.1479e+007
##   10       0.8237  1.1467e+007
##   11       0.8229  1.1460e+007
##   12       0.8222  1.1455e+007
##   13       0.8217  1.1449e+007
##   14       0.8212  1.1447e+007
##   15       0.8208  1.1441e+007
##   16       0.8205  1.1440e+007
##   17       0.8202  1.1435e+007
##   18       0.8199  1.1434e+007
##   19       0.8197  1.1429e+007
```

```
# Making predictions
y_hat_reco <-  r$predict(test_data_mf, out_memory())
```

```
RMSEs_Table <- bind_rows(RMSEs_Table,
                         tibble(Method = "Matrix Factorization",RMSE=RMSE(test_set$rating, y_hat_reco)))
RMSEs_Table
```

```
## # A tibble: 8 x 2
##   Method                                    RMSE
##   <chr>                                    <dbl>
## 1 Goal                                   0.8649
## 2 Just the average of ratings            1.061135
## 3 Avg + movie effect                     0.9441568
## 4 Avg + movie and user effect            0.8659736
## 5 Avg + movie, user and genres effect    0.8656152
## 6 Avg + movie, user, genres and year effect 0.8654004
## 7 Regularization                         0.8649594
## 8 Matrix Factorization                   0.8334137
```

We can see that of all, this model is the one that best RMSE brings us, being able to meet the objective and exceed it.

3 Results

Having already finished the models, in the following table we can see the result:

```
RMSEs_Table
```

```
## # A tibble: 8 x 2
##   Method                                    RMSE
##   <chr>                                    <dbl>
## 1 Goal                                   0.8649
## 2 Just the average of ratings            1.061135
## 3 Avg + movie effect                     0.9441568
## 4 Avg + movie and user effect            0.8659736
## 5 Avg + movie, user and genres effect    0.8656152
## 6 Avg + movie, user, genres and year effect 0.8654004
## 7 Regularization                         0.8649594
## 8 Matrix Factorization                   0.8334137
```

The model with matrix_factorization being the one with the best predictability, we test it with the dataset validation_final.

```
#Making the validation dataset
validation_data_mf  <-  with(validation_final,  data_memory(user_index = userId,
                                                item_index = movieId,
                                                rating     = rating))

r <-  recosystem::Reco()

r$train(train_data_mf)
```

```
## iter      tr_rmse          obj
##    0       0.9624  1.3286e+007
##    1       0.8823  1.1987e+007
##    2       0.8657  1.1883e+007
##    3       0.8476  1.1680e+007
##    4       0.8400  1.1607e+007
##    5       0.8346  1.1562e+007
##    6       0.8307  1.1530e+007
##    7       0.8279  1.1506e+007
##    8       0.8261  1.1489e+007
##    9       0.8248  1.1477e+007
##   10       0.8237  1.1472e+007
##   11       0.8229  1.1461e+007
##   12       0.8222  1.1455e+007
##   13       0.8217  1.1450e+007
##   14       0.8213  1.1448e+007
##   15       0.8208  1.1442e+007
##   16       0.8205  1.1441e+007
##   17       0.8201  1.1436e+007
##   18       0.8199  1.1433e+007
##   19       0.8197  1.1430e+007
```

```
y_hat_reco_validation <-  r$predict(validation_data_mf, out_memory())

RMSEs_Table <- bind_rows(RMSEs_Table,
                         tibble(Method = "Final Validation",RMSE=RMSE(validation_final$rating, y_hat_re

RMSEs_Table
```

```
## # A tibble: 9 x 2
##   Method                                    RMSE
##   <chr>                                    <dbl>
## 1 Goal                                    0.8649
## 2 Just the average of ratings             1.061135
## 3 Avg + movie effect                      0.9441568
## 4 Avg + movie and user effect             0.8659736
## 5 Avg + movie, user and genres effect     0.8656152
## 6 Avg + movie, user, genres and year effect 0.8654004
## 7 Regularization                          0.8649594
## 8 Matrix Factorization                    0.8334137
## 9 Final Validation                        0.8331758
```

The model, as in the training and test process, with the `validation_final` base also obtains a very good RMSE, 0.8331758, complying with the requirements.

4 Conclusion

We started with a very simple and basic model, assuming that all ratings are the same, their differences only explained by random variations. This first approximation left us with an RMSE of 1.061135. Being this very far from the ideal, we begin to explain this random variation with different characteristics, reaching an RMSE of 0.8654004. As this is still insufficient, we tried regularization, stripping ourselves of information that does not have enough information to be relevant like others. This led us to reach a 0.8649594 of RMSE.

Finally, with matrix factorization, we use a more advanced model, not with linear approximations as we tried previously, being able to reach an RMSE of 0.8334137

Having our model, we did tests with the validation_final base, in order to measure our model, being an RMSE similar to the previous one, with 0.8331758 fo RMSE.

Even with its limitations, such as not being designed to perform in the same way with new records of the other variables, for example new movies, new users, etc., this last model is by far the best approximation to the reality of all that we saw, being the one chosen to be the final model