# Movielens_final project

Rita Sánchez

December 2021

## Capstone: Movielens Project

This report closes the MovieLens Capstone Project as part of the HarvardX PH125.9x course. The goal of this project is to study different recommendation systems, comparing them to determine which one gets the best result for our goal.

### Introduction

A recommendation system filters the data using different algorithms and recommends the most relevant items to user. It first captures the past behavior of a customer and based on that, recommends products which the users might be likely to buy. Eg: in the case of e-commerce which product to buy, in the case of kindle which book to read or, in the case of Netflix, which movie to watch.

In particular, Netflix uses a recommendation system to predict how many stars a user will give a specific movie: one start suggests "bad movie"; but five starts suggests "excellent movie". The purpose of this report is to study the basics of how this recommendation are made, by replicating some of the data analysis strategies used by the winning team of the *Netflix challenges* [1].

The Netflix data is not publicly available, but the GroupLens research lab [2] generated their own database with over 20 millions ratings for over 27.000 movies by more than 138.000 users. For this project, we will use a small subset of this data generated with the code provided by the course instructors. The new dataset is named *"movielens"*.

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

library(tidyverse)
library(caret)
library(data.table)

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:
```

[1] In October 2006, Netflix offered a challenge to the data science community: improve our recommendation algorithm by 10% and win a million dollars. In September 2009, the winners were announced. A good summary of how the winning algorithm was put together here: http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/ and a more detailed explanation here: http://www.netflixprize.com/assets/GrandPrize2009_ BPC_BellKor.pdf.

[2] https://grouplens.org/

```
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")
```

## Data exploration

We can see that *"movielens"* dataset is a table in tidy format with 10,000,054 rows and 6 columns.

```
movielens %>% as_tibble()
# A tibble: 10,000,054 x 6
   userId movieId rating timestamp title                  genres
    <int>   <dbl>  <dbl>     <int> <chr>                  <chr>
 1      1     122      5 838985046 Boomerang (1992)       Comedy|Romance
 2      1     185      5 838983525 Net, The (1995)        Action|Crime|Thriller
 3      1     231      5 838983392 Dumb & Dumber (1994) Comedy
 4      1     292      5 838983421 Outbreak (1995)        Action|Drama|Sci-Fi|Thrill...
 5      1     316      5 838983392 Stargate (1994)        Action|Adventure|Sci-Fi
```

The columns are:

*userId*: it is a unique number assigned to each user.

*movieId*: it is a unique number assigned to each movie.

*rating*: from 1 (lowest rating) to 5 (highest rating)

*timestamp*: it is the time and data in which the rating was provided. the units are seconds since January 1, 1970.

*title*: it is the movie tittle

*genres*: the genre associated to the movie.

So, each row represents a rating given by one user to one movie in a given date. We can see also the number of unique users that provided ratings and the number of unique movies that are rated: 69878 unique users and 10677 unique movies.

```
movielens %>%
summarize(n_users = n_distinct(userId),
          n_movies = n_distinct(movieId))
 n_users n_movies
1   69878    10677
```

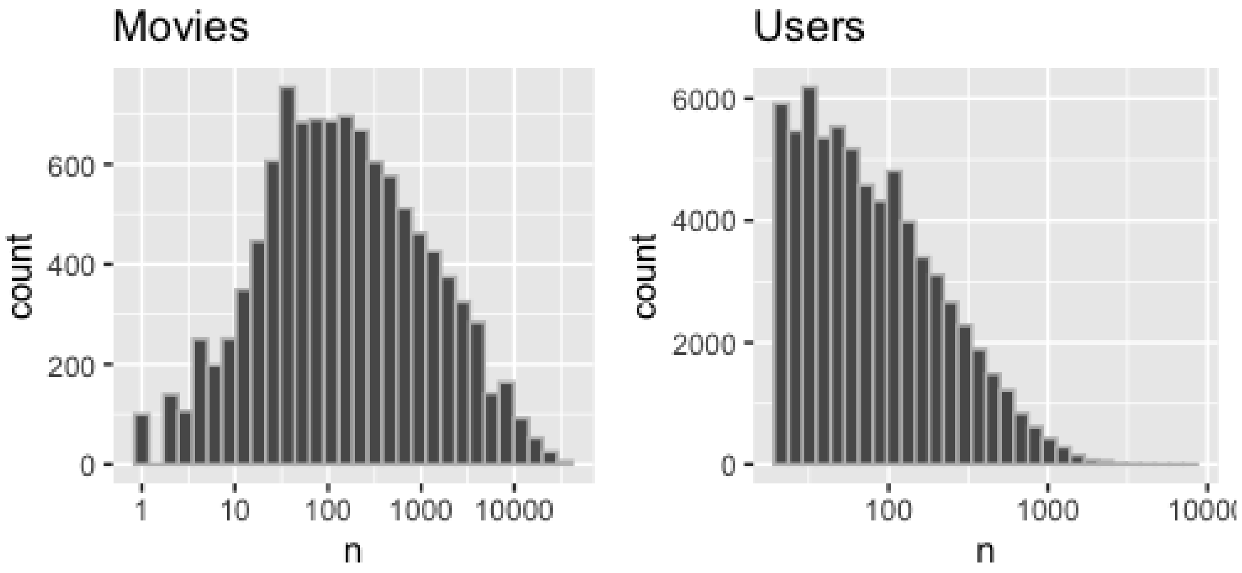
If we multiply the unique users and the unique movies, we get a number close to 750 million, yet our data tabla has about 10 million rows. That implies that not every user rated every movie.

This machine learning task is challenged because each outcome $Y$ has a different set of predictors. To see that, take into account that if we are predicting the rating for movie $i$ by user $u$, in principle, all other ratings related to movie $i$ and by user $u$ may be used as predictors, but different users rate different movies and a different number of movies. Furthermore, we may be able to use information from other movies that we have determined are similar to movie $i$ or from users determined to be similar to user $u$.

Exploring data, we can see that some movies get rated more than others. In the same way, some users are more active than others at rating movies.

```
# rating per movie
movielens %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "grey") +
  scale_x_log10() +
  ggtitle("Movies")
```

```
# rating per user
movielens %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "grey") +
  scale_x_log10() +
  ggtitle("Users")
```



## Recommendation systems

Recommendation system are a subclass of machine learning which, generally, deal with ranking or rating products/users. Generally speaking, a recommendation system is a system which predicts ratings that a user might give to a specific item. These predictions will then be ranked and returned back to the user. Recommendation systems deal with a large volume of information by filtering the important information based on the data provided by a user and other factors that address the user's preferences and interests. They compare a user's data collection with a similar collection and create a list of articles and/or topics to recommend to the user.

To develop one of these recommendation system, we need to build an algorithm with the data we have collected that will then be applied outside our control, when users look for movie recommendations. Let's create a training set (named *edx*) to develop the algorithm and a test set (named *validation*) to evaluate that algorithm. To make sure we do not include users and movies in the test set that do not appear in the training set, we have to remove these entries using the ***semi_join ()*** function. For that purpose, we use again the code provided by the course instructors.

```
# 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,]    #train set
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")  #test set

# 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)
```

Now, we are ready to elaborate and evaluate different regressions models. But, previously, we must establish a criterion to determine which model is "better" than other anticipating a user's choice, that is, we have to define our loss function.

### Loss function

The Netflix challenges used the typical error loss: they decided on a winner based on the residual mean squared error (*RMSE*) on a test set. We define $y_{u,i}$, as the rating for movie $i$ by user $u$, and $\hat{y}_{u,i}$ our predictor. Then, the *RMSE* can be defined as:

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

with $N$ as the number of all combination of user/movies.

The code used to calculate the *RMSE* for vector of ratings and their corresponding predictors is as follows:

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

*RMSE* can be interpreted similarly to a standard deviation: it is the typical error we make when predicted a movie rating. If this number is larger than 1, it means that our typical error is larger than one start… and that's not good. Take in mind that to win the prize of the *Netflix challenge* a participating team had to get a *RMSE* of about **0,857**.

With that value in mind, we will explore different models, from the simplest one to others more complex and seeing the evolution of RMSE in each one.

### The simplest model

The simplest possible recommendation system is to predict the same rating for all movies, regardless the user. A model that assumes the same rating for all movies and users with all the differences explained by random variables could be expressed as:

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

with $\epsilon_{u,i}$, independent error sampled from the same distribution centered at 0 and $\mu$ the "true" rating for all movies. We know that the estimate that minimize the *RMSE* is the *least square* estimate of $\mu$ and, in this case, is the average of all rating.

```
mu <- mean(edx$rating)
mu
[1] 3.512465
```

If we predict all the ratings with $\mu$, the *RMSE* is calculated with the following code:

```
rmse_model_mu <- RMSE(validation$rating, mu)
rmse_model_mu
[1] 1.061202
```

So, with a $RMSE = 1.0612$ our team would be very far from winning the grand prize of the *Netfilx challenge*!!. Let's improve our model.

### Modeling movie effects

We know that some movies are rated higher than others. So, we can improve our previous model adding the term $b_i$ to represent average rating for movie $i$.

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

We could estimate this equation use *least squares* with the ***lm()*** function, but running this code would be very slow, considering that our data base has thousand of movies, and each movie $i$ gets its $b_i$. Instead of that, we can use the concept that the least square estimate $\hat{b}_i$ is just the average of $Y_{u,i} - \hat{\mu}$ for each movie $i$. This is the code:

```
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating - mu))
```

How much does our prediction improve?

```
predicted_rating_b_i <- mu + validation %>%
  left_join(movie_avgs, by = 'movieId') %>%
  pull(b_i)

rmse_model_movie <- RMSE(validation$rating, predicted_rating_b_i)
rmse_model_movie
[1] 0.9439087
```

Nice improvement! Let's try to make it even better.

### Modeling movie and user effects

We are conscious that some users are more critical when it comes to rating a film than others. To include this user-specific effect we added a new variable $b_u$ to the equation. Now, our model looks like this:

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

For the same reason explained for the previous model, we won't use the ***lm()*** function. Instead, we will take an approximation by calculating $\hat{\mu}$ and $\hat{b}_i$. Then, we will estimate $\hat{b}_u$ as the average of $y_{u,i} - \hat{\mu} - \hat{b}_i$. For that, we will use the following code:

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

Now, we can see how much the *RMSE* improves:

```r
# predicted ratings
predicted_rating_b_u <- validation %>%
  left_join(movie_avgs, by = 'movieId') %>%
  left_join(user_avgs, by = 'userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

rmse_model_user <- RMSE(validation$rating,predicted_rating_b_u)
rmse_model_user
[1] 0.8653488
```

We can notice that including both movie and user effect, the *RMSE* decreases more than 8,0%, with respect to the one obtained only including the movie effect. In addition, the error is only 1.0% higher than the one in the model presented by the winning team of the *Netflix challenge*.

## Regularization

When we train our computational models with a train data set we are making the algorithm capable of generalizing a concept so that, when query it for a new unknown data set, it will be able to return a reliable result.

But it may happen that our train set includes data that introduce a lot of "noise" in the process. And, what do we mean by "noise"? By "noise" we refer to those specific data that do not represent the real characteristics of our database because they show a rather atypical behavior.

In our train dataset, named **edx**, we observed that there are certain movies qualified among the 10 "best" and the 10 "worst" that are rated by very few users, in most cases, just one user.

```r
movie_titles <- movielens %>%
  select(movieId, title) %>%
  distinct()

### top 10 best
movie_avgs %>%
  left_join(movie_titles, by='movieId') %>%
  arrange(desc(b_i)) %>%
  slice(1:10) %>%
  pull(title)

###-- how often they are rated

edx %>% count(movieId) %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(movie_titles, by='movieId') %>%
  arrange(desc(b_i)) %>%
  slice(1:10) %>%
  pull(n)

[1] 1 2 1 1 1 1 4 4 4 2

### top 10 worst
movie_avgs %>%
  left_join(movie_titles, by='movieId') %>%
  arrange(b_i) %>%
  slice(1:10) %>%
  pull(title)

###-- how often they are rated
edx %>% count(movieId) %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(movie_titles, by='movieId') %>%
  arrange(b_i) %>%
  slice(1:10) %>%
  pull(n)

[1]   2   1   1   1   2  56  14  32 199   2
```

And, this is the kind of "noise" we do not want to affect our model. That's why we introduce the concept of *regularization*. *Regularization* are techniques to reduce the "noise" and to improve the performance of our model.

### Regularized movie effect

Consider a case in which we have a movie $i = 1$ with 100 user ratings and 5 movies $i = 2, 3, 4, 5, 6$ with just one user rating. We can to fit the following model:

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

If we use least squares, we know that the estimate for the first movie effect $b_1$ is the average of the 100 users rating, that is: $1/100 \sum_{i=1}^{100} (Y_{i,1} - \mu)$. However, the estimate for movies 2,3,4,5 and 6 will be the observe deviation from the average rating $\hat{b}_i = Y_{u,i} - \hat{\mu}$, which is an estimated based on just one number: not precise at all. In fact, ignoring the one user and guessing than movies 2,3,4,5 and 6 are just "average movies" ($b_i = 0$) may provide a better prediction. So, the general idea of penalized regression is to consider the total variability generated for these movies, expressed as: $\sum_{i=1}^{6} b_i^2$. That's why, instead of minimizing the least squares equation, we minimize an equation than adds a penalty term:

$$\frac{1}{N} \sum_{u,i} (y_{u,i} - \mu - b_i)^2 + \lambda \sum_i b_i^2$$

The first term is just the least squares and the second one is a penalty term. The values of $b_i$ that minimize the equation are:

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

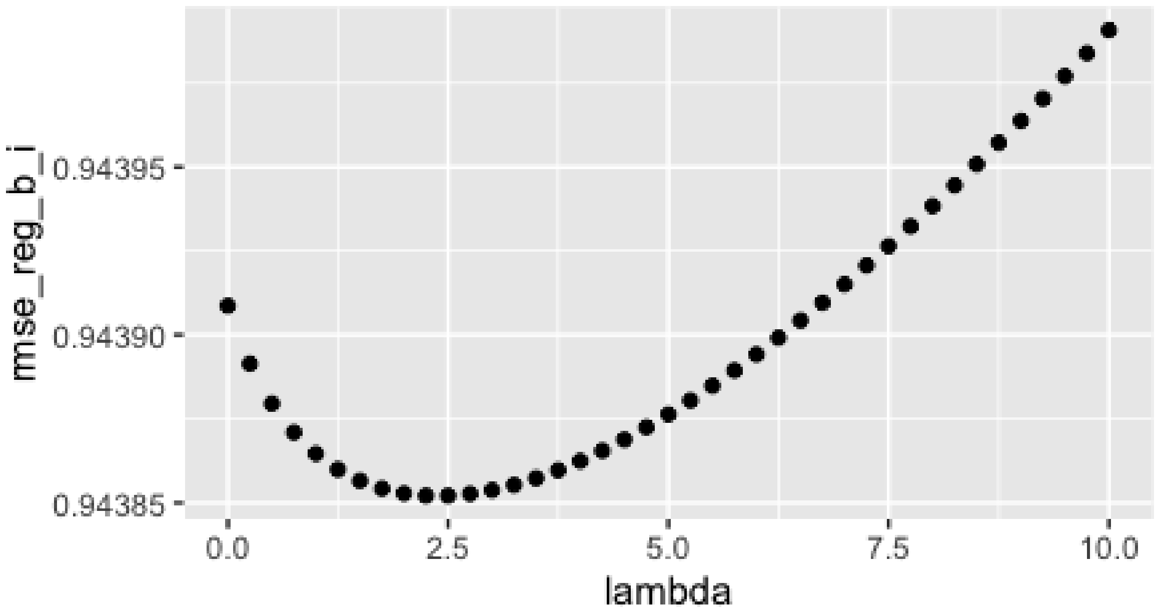
where $n_i$ is the number of ratings for movie $i$, and $\lambda$ is the tuning parameter that decides how much we want to penalize the flexibility of our model.

When our sample size $n_i$ is very large, a case in which we obtain a stable estimate, then the term $\lambda$ is ignored since $n_i + \lambda \sim n_i$. But, if $n_i$ is small, we must be very carefully chosen the value of $\lambda$. In fact, if $\lambda = 0$, the penalty term has no effect, and the estimate will be equal to just least squares. However, as $\lambda \to \infty$, the impact of the penalty term grows, and $\hat{b_i}(\lambda)$ approach to zero. That implies that selecting a good value of $\lambda$ is critical. Cross validation comes in handy for this purpose.

So, here is the code to obtain the $\lambda$ that minimize the *RMSE* of our equation

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

##--Regularized movie effect model
```

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

summation_term <- edx %>%
  group_by(movieId) %>%
  summarise(s = sum(rating - mu), n_i = n())

rmse_reg_b_i <-  sapply(lambda, function(l){
  predicted_rating_reg_b_i  <- validation %>%
    left_join(summation_term, by = 'movieId') %>%
    mutate(b_i_lambda = s/(n_i + l)) %>%
    mutate(pred = mu + b_i_lambda) %>%
    pull(pred)
  return(RMSE(predicted_rating_reg_b_i, validation$rating))
})

qplot(lambda, rmse_reg_b_i)
```



As we can see in the plot, $\lambda = 2,5$ minimize the $RMSE$ (0,9438)

```
#for the full model, the optimal lambda is
lambda[which.min(rmse_reg_b_i)]
[1] 2.5
#the minimun error is
rmse_model_reg_b_i <- min(rmse_reg_b_i)
rmse_model_reg_b_i
[1] 0.9438521
```

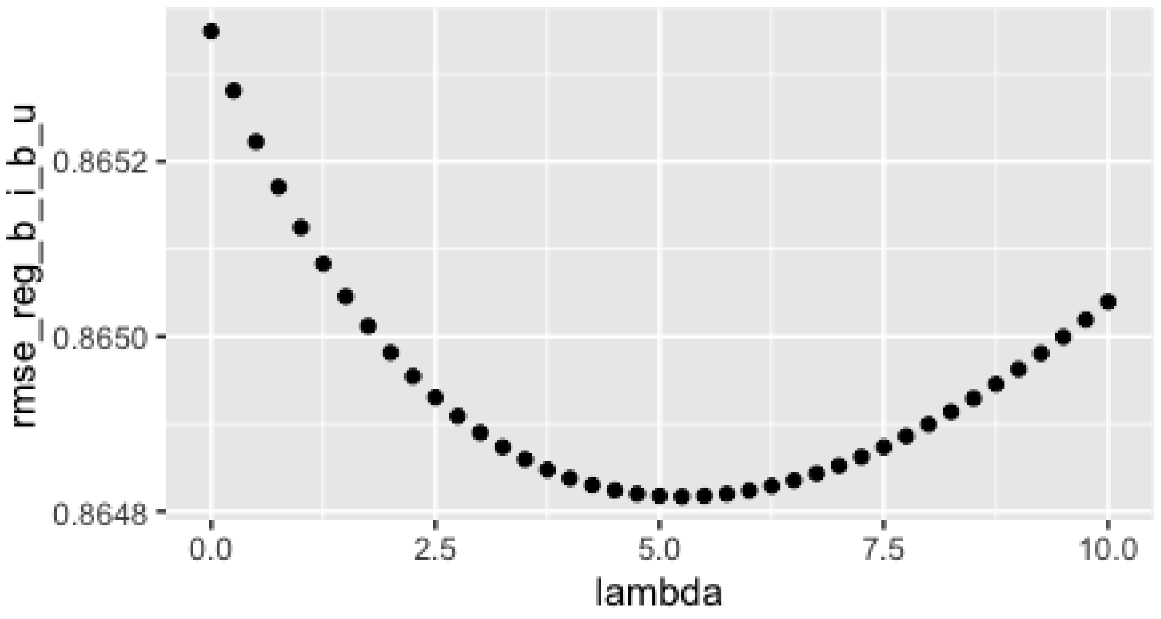
### Regularized movie and user effect

We can use regularization for the estimate user effects as well. Our model now will be:

$$\frac{1}{N}\sum_{u,i}(y_{u,i} - \mu - b_i)^2 + \lambda(\sum_i b_i^2 + \sum_u b_u^2)$$

The estimates than minimize this equation is similar to what did in previous case. The code used is as follows:

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

##--Regularized movie + user effect model

rmse_reg_b_i_b_u <- sapply(lambda, function(l){
  b_i_lambda <- edx %>%
    group_by(movieId) %>%
    summarise(b_i_lambda = sum(rating - mu)/(n()+l))


  b_u_lambda <- edx %>%
    left_join(b_i_lambda, by='movieId') %>%
    group_by(userId) %>%
    summarise(b_u_lambda = sum(rating - mu - b_i_lambda)/(n()+l))

  predicted_rating_reg_b_i_b_u <- validation %>%
    left_join(b_i_lambda, by='movieId') %>%
    left_join(b_u_lambda, by='userId') %>%
    mutate(pred = mu + b_i_lambda + b_u_lambda) %>%
    pull(pred)

  return(RMSE(predicted_rating_reg_b_i_b_u, validation$rating))
})


qplot(lambda, rmse_reg_b_i_b_u)
```



And, again thanks to the plot, we see that $\lambda$ around 5 minimize the *RMSE* (0,8648)

```
#for the full model, the optimal lambda is
lambda[which.min(rmse_reg_b_i_b_u)]
[1] 5.25
```

```
#the minimun error is
rmse_model_reg_b_i_b_u <- min(rmse_reg_b_i_b_u)
rmse_model_reg_b_i_b_u
[1] 0.864817
```

## Compared results and conclusion

In the following table we show all the *RMSE* obtained in each model,

| Method | RMSE |
|---|---|
| Average | 1.0612 |
| Movie Effect | 0.9439 |
| Movie + User Effect | 0.8653 |
| Regularized Movie Effect | 0.9439 |
| Regularized Movie + User Effect | 0.8648 |

As we can see, the lowest *RMSE* is reached with the *Regularized movie + user effect* model, although the gain respect to the same model with no regularization is negligible (less than 0,06%). Nevertheless, we can consider the *Regularized movie + user effect* model is the most suitable recommendation system among all the models analyzed in this project to predict ratings. This could be the model selected to participate in the *"Netflix challenges"*.

Futures works could include time effect, estimating the following equation:

$$Y_{u,i} = \mu + b_i + b_u + f(d_{u,i}) + \epsilon_{u,i}$$

with $d_{u,i}$ as the the day for user's $u$ rating movie $i$; and $f$, as a smooth function of $d_{u,i}$

It would also be interesting to include genre effect, estimating this equation:

$$Y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^{K}(x_{u,i}\beta_k) + \epsilon_{u,i}$$

with $g_{u,i}$ as the genre for user's $u$ rating of movie $i$; and $x_{u,i}^k = 1$ if $g_{u,i}$ is genre $k$.