Data Science - Capstone / MovieLens Project

Emmanuel Dupuis

August 3rd , 2022

Contents

1 Overview

Recommendation algorithm are a specific form of Machine Learning whose goal is to present items of information (movies, music, books...) that are likely to interest the user. Generally, a recommendation system allows to predict the "rating" that a user would give from his profile.

In 2006, Netflix offered a challenge to the data science community: propose an efficient recommendation algorithm.

The chapter "Data Science: Machine Learning" of the "data science Professional certificate program" proposes in fact some methods used by the participants of the Netflix challenge

In this project, we are asked to implement and improve them : we will build several algorithms which can optimally predict a user's rating. The measure of optimality of our algorithm will be based on the Root Mean Square Error (RMSE). For this project, we are only required to maintain an RMSE of less than 0.86490 (25 points out of 25 for a RMSE below 0.86490), knowing that the challenge was won the first year by the Korbell Team with an RMSE of 0.8914.

The chapter "Data Science: Machine Learning" of the "data science Professional certificate program" uses a model based on a measurement of the effects, also called biases, of different factors: it is limited to the factors "movie" and "user", we will complete it with the factors "genre" and "times stamp".

We will then improve it with a regularization to attenuate the variability of the sizes by penalizing low rated movies, low rated users and low intervening genres.

1.1 Data Sets

First, we need to download the data from which we will train and validate our different algorithms.

1.1.1 Create train and validation sets

To mimic the evaluation process, we randomly split our data into two sets : a training set and a test set. And we act that we don't know the outcome of the test set. We develop algorithms using only the training set. The test set is used only for evaluation.

To create the train and validation sets, we are asked to write the following code : the data comes from a downloaded file from which we randomly extract 10% to form the train set **edx**, the other 90% forming the training set **validation**.

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

# Note: this process could take a couple of minutes

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()

# Three alternative download links
# download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
# download.file("https://owncloud.univ-artois.fr/index.php/s/3Oj588P4tiG992t/download", 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)

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

1.1.2 Structure of train and validation sets

The train and validation sets are tables in tidy format with thousands of rows. Each row represents a rating given by one user to one movie. Let's show the matrix for five users :

edx file :

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi

validation file :

userId	movieId	rating	timestamp	title	genres
1	231	5	838983392	Dumb & Dumber (1994)	Comedy
1	480	5	838983653	Jurassic Park (1993)	Action|Adventure|Sci-Fi|Thriller
1	586	5	838984068	Home Alone (1990)	Children|Comedy
2	151	3	868246450	Rob Roy (1995)	Action|Drama|Romance|War
2	858	2	868245645	Godfather, The (1972)	Crime|Drama

We see that the tables are made up of 6 columns: "userId", "movieId", "rating", "timestamp", "title" and

"genres". And a number of lines indicated below:

edx file :

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 ...
##  $ movieId  : num  122 185 ...
##  $ rating   : num  5 5 ...
##  $ timestamp: int  838985046 838983525 ...
##  $ title    : chr  "Boomerang (1992)" ...
##  $ genres   : chr  "Comedy|Romance" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

validation file :

```
## Classes 'data.table' and 'data.frame':   999999 obs. of  6 variables:
##  $ userId   : int  1 1 ...
##  $ movieId  : num  231 480 ...
##  $ rating   : num  5 5 ...
##  $ timestamp: int  838983392 838983653 ...
##  $ title    : chr  "Dumb & Dumber (1994)" ...
##  $ genres   : chr  "Comedy" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

2 Methods

The objective of the algorithm is then to predict the rating given by one user to one movie from the other data of the same line.

2.1 Loss function

To measure the efficiency of our algorithm, we are asked to minimize 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 denote our prediction with $\hat{y}_{u,i}$. The RMSE is then defined as:

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

with N being the number of user/movie combinations and the sum occurring over all these combinations.

It is obvious that the closer the $RMSE$ is to zero, the closer all our predictions will be to the user's rating.

Let's write a function that computes the RMSE for vectors of ratings and their corresponding predictors:

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

2.2 A first simple model

In this model, we assume that the same rating μ is given for all movies and users with all the differences explained by random variation:

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

where $\epsilon_{u,i}$ are independent errors sampled from the same distribution centered at 0.

We then show mathematically that this last property implies that the best estimate of μ is the average of all ratings and that this estimate minimizes the RMSE (of course we estimate the parameter of our model on the train set **edx**):

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

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

Then, we obtain the following RMSE (of course we calculate the RMSE on the test set **validation**) with the predictor :

$$\widehat{Y}_{u,i} = \mu$$

```
new_rmse <- RMSE(validation$rating, mu)
new_rmse
```

```
## [1] 1.061202
```

For comparing different approaches, let's start by creating a results table:

method	RMSE
Just the average	1.061202

2.3 A second model with movie effects

We can make the hypothesis that some films are better rated than others by all the spectators (this hypothesis will be confirmed by a histogram drawn next). Them, we can augment our previous model by adding the term b_i to represent average ranking for movie i:

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

where $\epsilon_{u,i}$ are always independent errors sampled from the same distribution centered at 0.

This last property, as in the previous paragraph, implies that the best estimate of b_i is the average of all: $Y_{u,i} - \mu$ (estimation of the parameters on the train set **edx**):

```
mu=mean(edx$rating)
bi <- edx %>%
      group_by(movieId) %>%
      summarize(bi = mean(rating - mu))
```

We can then now construct predictors (prediction on the test set **validation**):

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

```
prediction_test <- validation %>%
                   left_join(bi, by = "movieId") %>%
                    mutate(rating_hat = (mu + bi))
```

and calculate the new RMSE:

```
new_rmse <- RMSE(prediction_test$rating_hat,validation$rating)
new_rmse
```

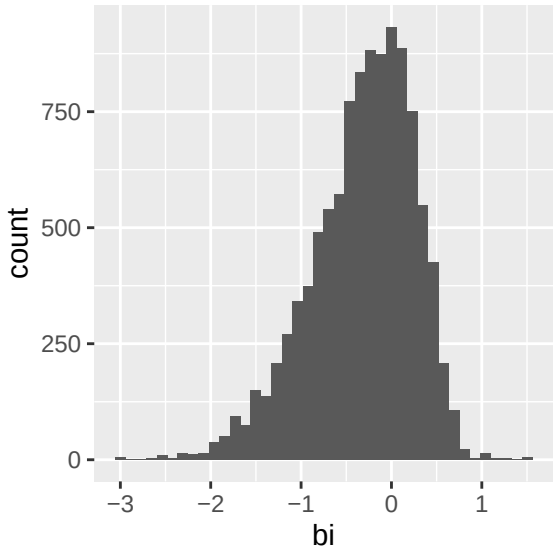
```
## [1] 0.9439087
```

Taking into account the movie effects allows us to improve the algorithm by decreasing the RMSE.

We can then complete the previous table:

method	RMSE
Just the average	1.0612018
with movie effects	0.9439087

Now let's plot the histogram representing the distribution of b_i (movie effects)



Since to calculate the b_i, we have already subtracted μ, we notice that they are centered on 0. But not all b_i are identical to zero. This confirms our hypothesis of movie effects (some films are better rated than others...)

2.4 A third model with user effects

We can also make the assumption that there is significant variability among users: some users generously rate all movies and others are less generous (this hypothesis will be also confirmed by a histogram drawn next). This implies that a further improvement to our model could be:

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

where b_u is a user-specific effect and $\epsilon_{u,i}$ are always independent errors sampled from the same distribution centered at 0.

This last property still implies that the best estimate of b_u is the average of all: $Y_{u,i} - \mu - b_i$ (estimation of the parameters on the train set **edx**):

```
bu <- edx %>%
      left_join(bi, by = "movieId") %>%
      group_by(userId) %>%
      summarize(bu = mean(rating -bi - mu))
```

We can then now construct predictors (prediction on the test set **validation**):

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

```
prediction_test <- validation %>%
                   left_join(bi, by = "movieId") %>%
                   left_join(bu, by = "userId") %>%
                   mutate(rating_hat = (mu + bu + bi))
```

and calculate the new RMSE:

```
new_rmse <- RMSE(prediction_test$rating_hat,validation$rating)
new_rmse
```

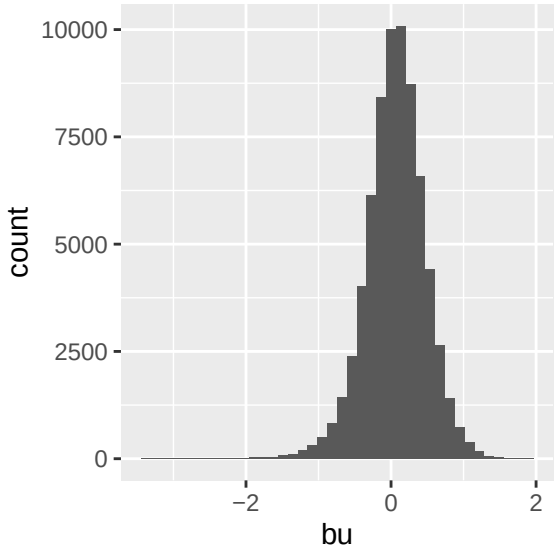
```
## [1] 0.8653488
```

Taking into account the movie effects allows us to improve the algorithm by decreasing the RMSE.

We can then complete the previous table:

method	RMSE
Just the average	1.0612018
with movie effects	0.9439087
with movie and user effects	0.8653488

Now let's plot the histogram representing the distribution of b_i (movie effects)



The distribution show that not all b_u are identical to zero. This confirms our hypothesis of user effects (some films are better rated than others...)

2.5 A fourth model with genres effects

Like the previous generalizations of our model, we will now take into account the genres of the films. This implies that a further improvement to our model could be:

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

where b_g is a genre-specific effect and $\epsilon_{u,i,g}$ are always independent errors sampled from the same distribution centered at 0.

In the manner of the previous generalizations, b_g is the average of all: $Y_{u,i} - \mu - b_i - b_u$ (estimation of the parameters on the train set **edx**):

```
bg <- edx %>%
      left_join(bi, by = "movieId") %>%
      left_join(bu, by = "userId") %>%
```

```
      group_by(genres) %>%
      summarize(bg = mean(rating -bi -bu - mu))
```

We can then now construct predictors (prediction on the test set **validation**):

$$\widehat{Y}_{u,i} = \mu + b_u + b_i + b_g$$

```
prediction_test <- validation %>%
                   left_join(bi, by = "movieId") %>%
                   left_join(bu, by = "userId") %>%
                   left_join(bg, by = "genres") %>%
                   mutate(rating_hat = (mu + bu + bi + bg))
```

and calculate the new RMSE:

```
new_rmse <- RMSE(prediction_test$rating_hat,validation$rating)
new_rmse
```

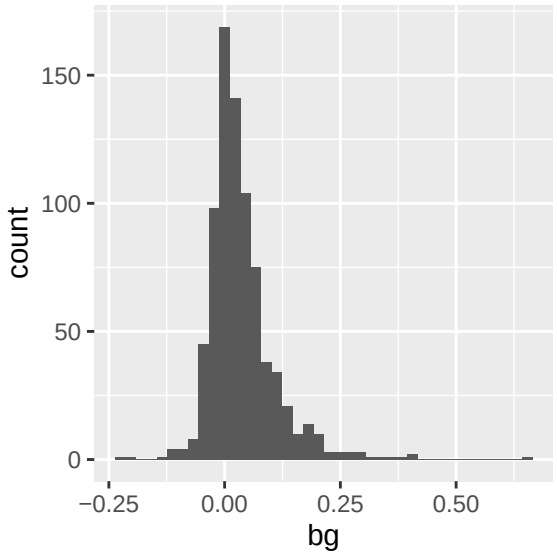
```
## [1] 0.8649469
```

Taking into account the genres effects allows us to improve the algorithm by decreasing the RMSE.

We can then complete the previous table:

method	RMSE
Just the average	1.0612018
with movie effects	0.9439087
with movie and user effects	0.8653488
with movie, user and genres effects	0.8649469

Now let's plot the histogram representing the distribution of b_g (genres effects)



The distribution show that not all b_g are identical to zero. This confirms our hypothesis of genres effects.

2.6 A fifth model with a times stamp effects

We will now take into account a times stamp effect on the films ratting. This implies that a further improvement to our model could be:

$$Y_{u,i} = \mu + b_i + b_u + b_g + b_t + \epsilon_{u,i,g,t}$$

where b_t is a times stamp effect and $\epsilon_{u,i,g,t}$ are always independent errors sampled from the same distribution centered at 0.

In the manner of the previous generalizations, b_t is the average of all: $Y_{u,i} - \mu - b_i - b_u - b_t$ (estimation of the parameters on the train set **edx**).

But the average will be done here on each of the different weeks of the times stamp. Using the **as_datetime** function that converts the data in the **timestamps** column to a date and the **round_date** function that determines the week number of this date.

```
bt <- edx %>%
  mutate(weeks = round_date(as_datetime(timestamp),"week")) %>%
  left_join(bi, by = "movieId") %>%
  left_join(bu, by = "userId") %>%
  left_join(bg, by = "genres") %>%
  group_by(weeks) %>%
  summarize(bt = mean(rating -bi -bu - bg - mu))
```

We can then now construct predictors (prediction on the test set **validation**):

$$\widehat{Y}_{u,i} = \mu + b_u + b_i + b_g + b_t$$

```
prediction_test <- validation %>%
  mutate(weeks = round_date(as_datetime(timestamp),"week")) %>%
  left_join(bi, by = "movieId") %>%
  left_join(bu, by = "userId") %>%
  left_join(bg, by = "genres") %>%
  left_join(bt, by = "weeks") %>%
  mutate(rating_hat = (mu + bu + bi + bg + bt))
```

and calculate the new RMSE:

```
new_rmse <- RMSE(prediction_test$rating_hat,validation$rating)
new_rmse
```

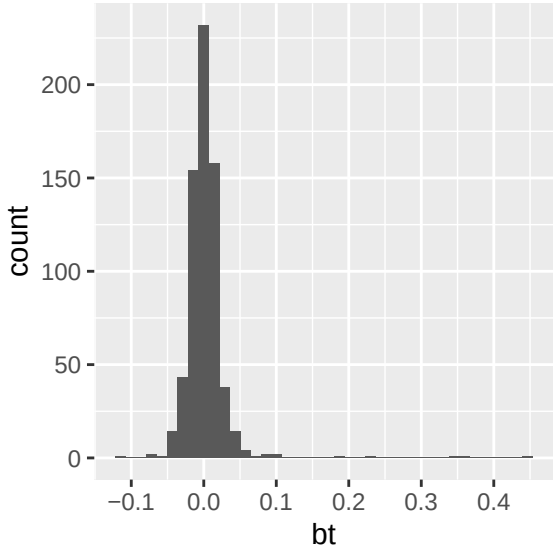
```
## [1] 0.8648392
```

Taking into account the times stamp effects allows us to improve the algorithm by decreasing the RMSE.

We can then complete the previous table:

method	RMSE
Just the average	1.0612018
with movie effects	0.9439087
with movie and user effects	0.8653488
with movie, user and genres effects	0.8649469
with movie, user, genres and times effects	0.8648392

Now let's plot the histogram representing the distribution of b_t (timestamp effects)

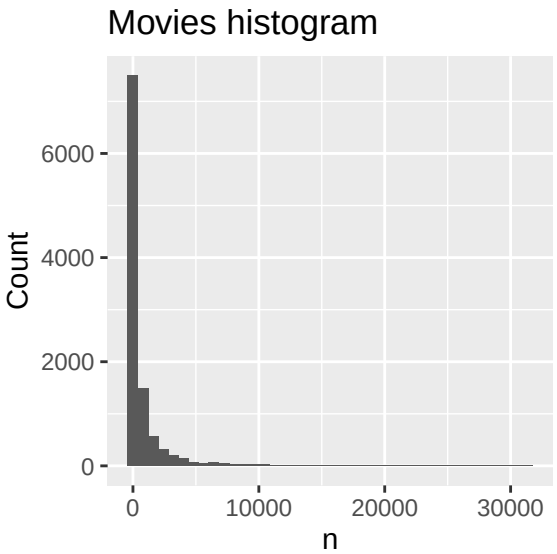


The distribution show that not all b_t are identical to zero. This confirms our hypothesis of timestamp effects.

2.7 Regularization

2.7.1 Motivation for regularization

Let's look at some of the general properties of the data to better understand the challenges. In the following histogram, we show the number of ratings per film. And we notice that few movies get a lot of ratings: blockbuster movies are watched by millions and independent movies are watched by a few.



The problem is that the b_i average of a film evaluated by a small number of viewers is not reliable: we make an average on few data.

2.7.2 Averages with penalty coefficients

The idea is, instead of using a classical average to calculate b_i as we did before, to use an average involving a penalty coefficient λ:

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

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

where n_i is the number of ratings made for movie i.

Indeed, when our sample size n_i is very large, a case which will give us a stable estimate, then the penalty λ is effectively ignored since $n_i + \lambda \approx n_i$. However, when the n_i is small, then the estimate $b_i(\lambda) \approx 0$. Thus, the films evaluated by a small number of spectators will have little impact on the calculation of our estimate.

The reasoning is the same for the other coefficients b:

$$b_u(\lambda) = \frac{1}{N + \lambda} \sum_{i} (Y_{i,u} - b_i - \mu)$$

$$b_g(\lambda) = \frac{1}{N + \lambda} \sum (Y - b_i - b_u - \mu)$$

$$b_t(\lambda) = \frac{1}{N + \lambda} \sum (Y - b_i - b_u - b_g - \mu)$$

It will then suffice to make a calculation similar to the one already made to calculate the $RMSE$ for different values of λ. And to choose the λ minimizing the $RMSE$.

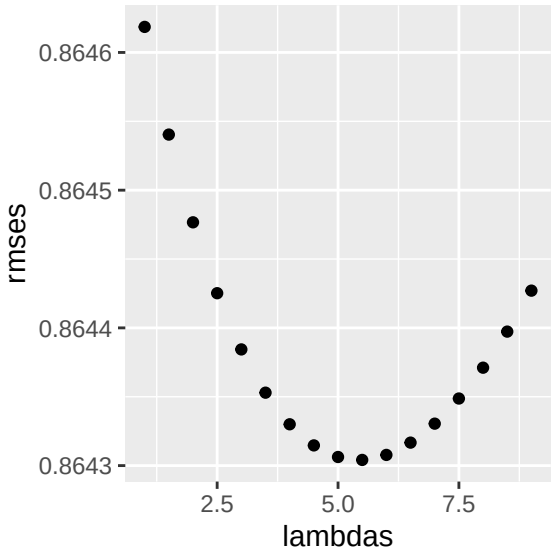
2.7.3 Choice of the best penalty coefficient

It is then enough to apply the following code which applies what we have just described on a set of λ between 1 and 9 :

```
lambdas <- seq(1, 9, 0.5)
rmses <- sapply(lambdas, function(l){

  mu=mean(edx$rating)

  bi <- edx %>%
    group_by(movieId) %>%
    summarize(bi = sum(rating - mu)/(n()+l))

  bu <- edx %>%
    left_join(bi, by = "movieId") %>%
    group_by(userId) %>%
    summarize(bu = sum(rating -bi - mu)/(n()+l))

  bg <- edx %>%
    left_join(bi, by = "movieId") %>%
    left_join(bu, by = "userId") %>%
    group_by(genres) %>%
    summarize(bg = sum(rating -bi -bu - mu)/(n()+l))

  bt <- edx %>%
    mutate(weeks = round_date(as_datetime(timestamp),"week")) %>%
    left_join(bi, by = "movieId") %>%
    left_join(bu, by = "userId") %>%
    left_join(bg, by = "genres") %>%
    group_by(weeks) %>%
    summarize(bt = sum(rating -bi -bu - bg - mu)/(n()+l))

  predicted_ratings <- validation %>%
    mutate(weeks = round_date(as_datetime(timestamp),"week")) %>%
```

```
    left_join(bi, by = "movieId") %>%
    left_join(bu, by = "userId") %>%
    left_join(bg, by = "genres") %>%
    left_join(bt, by = "weeks") %>%
    mutate(rating_hat = (mu + bu + bi + bg + bt))

  return(RMSE(predicted_ratings$rating_hat, validation$rating))
})
```

We can then plot the $RMSE$ as a function of λ



The λ that minimizes the RMSE is then

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

```
## [1] 5.5
```

The associated RMSE is

```
new_rmse <- min(rmses)
new_rmse
```

```
## [1] 0.8643041
```

Taking into account the movie effects allows us to improve the algorithm by decreasing the $RMSE$.

We can then complete the previous table:

method	RMSE
Just the average	1.0612018
with movie effects	0.9439087
with movie and user effects	0.8653488
with movie, user and genres effects	0.8649469
with movie, user, genres and times effects	0.8648392
with all effects and penalty coefficient	0.8643041

3 Results

Here are the results obtained with the different models we have implemented:

method	RMSE
Just the average	1.0612018
with movie effects	0.9439087
with movie and user effects	0.8653488
with movie, user and genres effects	0.8649469
with movie, user, genres and times effects	0.8648392
with all effects and penalty coefficient	0.8643041

We see that the results kept improving as we took into account new effects. With a last improvement with a calculation of the averages with penalty coefficients.

We found the lowest value of $RMSE$ that is :

```
## [1] 0.8643041
```

4 Discussion

So we can confirm that the final model for our project is the best:

$$Y_{u,i} = \mu + b_i + b_u + b_g + b_t + \epsilon_{u,i,g,t}$$

With biases calculated using averages with penalty coefficients:

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

$$b_u(\lambda) = \frac{1}{N + \lambda} \sum_{i} (Y_{i,u} - b_i - \mu)$$

$$b_g(\lambda) = \frac{1}{N + \lambda} \sum (Y - b_i - b_u - \mu)$$

$$b_t(\lambda) = \frac{1}{N + \lambda} \sum (Y - b_i - b_u - b_g - \mu)$$

The predictors is then:

$$\widehat{Y}_{u,i} = \mu + b_u(\lambda) + b_i(\lambda) + b_g(\lambda) + b_t(\lambda)$$

5 Conclusion

Then, we have built a machine learning algorithm to predict movie ratings with MovieLens dataset using a optimal model characterised by a RMSE value lower than the initial goal of the present project: 25 points out of 25 for a RMSE below 0.86490.

Here, we have seen that our rmse was :

```
## [1] 0.8643041
```

It is much smaller than : 0.86490.

We could also improve this model by doing, for example, a principal component analysis (PCA) on the residue:

$$\epsilon_{u,i,g,t} = Y_{u,i} - \mu - b_i - b_u - b_g - b_t$$

to decompose it into a matrix product. We tried but the limitations of our machine mean that after 15 hours of calculation the result is still not complete.