MovieLens

Alexis Barrière

30/01/2020

Introduction

MovieLens data have been collected by the GroupLens Research Lab. They have collected and made available rating data sets from the MovieLens web site (http://movielens.org). The data sets were collected over various periods of time, depending on the size of the set.

In this study we will use the MovieLens10M dataset. This data set contains 10'000'054 ratings and 95'580 tags applied to 10'681 movies by 71'567 users of the online movie recommender service MovieLens. Users were selected at random for inclusion. All users selected had rated at least 20 movies. No demographic information is included. Each user is represented by an id, and no other information is provided.

The data are contained in three files, movies.dat, ratings.dat and tags.dat, but here we will use only the first two files.

The goal of this project is to find a method to predict ratings for a user and a movie based on known ratings. We won't cover the prediction of ratings for a new user or a new movie based on external data.

MovieLens10M Dataset

Once the two files have been dowloaded, the data are merged in a single dataframe which includes movie and rating information:

```
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")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")
```

The data are then splitted in a main set on which we will work and define our parameters, called **edx** and a validation dataset, called **validation** that we will only use at the end to validate our model. This validation dataset will hold only 10% of the whole MovieLens10M dataset.

We also make sure that all the movies and users from the validation dataset are also in the edx dataset to avoid having not defined predictions by removing from **validation** the *bad* lines and include them back in the **edx** dataset.

```
set.seed(1, sample.kind="Rounding")
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)
```

Data Exploration

With the seed choosed to split our MovieLens dataset we then have a working dataset **edx** of 9'000'055 rows and 6 columns :

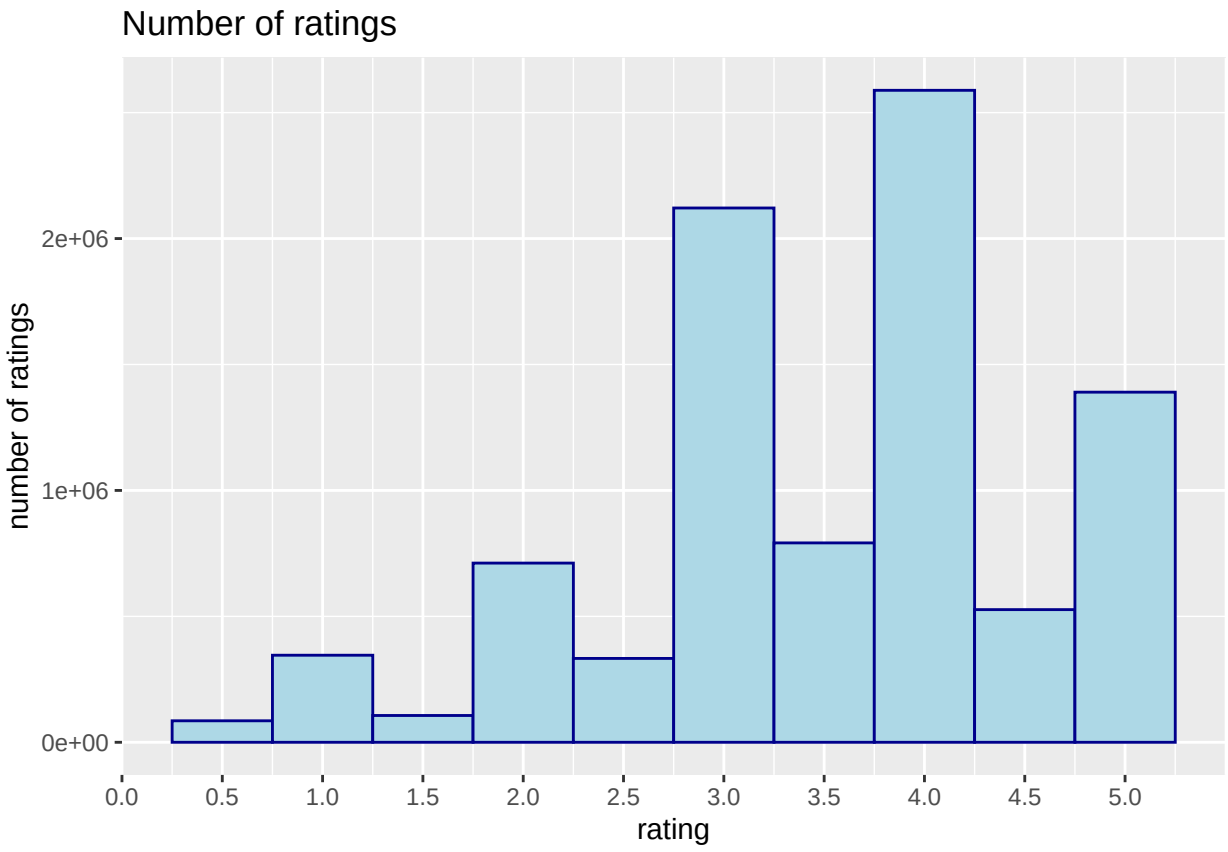
Table 1: edx dataset

	userId	movieId	rating	timestamp	title	genres
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance
2	1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

and a **validation** dataser of 999'999 rows and 6 columns.

We will now focus on the data in our **edx** dataset. We have NULL distinct users and NULL distinct movies.

The ratings are from 0.5 to 5 following this distribution‘:



We notice that :

- There is no rating 0
- The distribution is skewed to the high ratings (mean of 3.5 and median of 4)
- The half rating (.5) is less common than the whole number rating

Analysis and Results

The idea is to work with the data to see what factor is impacting the rating and use it to predict the future ratings.

The metric that we will use to validate our rating is the Root Mean Square Error (RMSE) defined mathematically by:

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

and in R by:

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

First of all, we will split our edx dataset in a train set (80%) and a test set (20%) to be able to tune some parameters before applying them to our validation dataset:

```
set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(y =edx$rating, times = 1,
                                  p = 0.2, list = FALSE)
```

```
train_set <- edx[-test_index,]
temp <- edx[test_index,]

# Make sure userId and movieId in validation set are also in edx set
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)
```

We will introduce in our analysis different factors and measure each time the prediction on the Test set based on Train set and on the Validation set based on the edx dataset. The Validation set will only be used to compute the final prediction based on the parameters found within the edx dataset.

The easiest start is to try predict the rating just based on the mean:

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

This gives us the following RMSEs :

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

method	RMSE_Test	RMSE_Valid
Just the average	1.059904	1.061202

Obviously this is not a good method to predict ratings.

So the next step is to be able to include factors that will be added or substracted to the mean value to better fit the reality.

First of all, everybody knows that there are good and bad movies, or saying it in another way some movies that are more most of the time rated below the average and movies that are rated above the average. This is what we will call the movie effect. To comupte it we will compute for each movie the mean gap between the real rating in the train set and the mean.

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

and then construct our predictions based on this movie effect:

```
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i
```

The RMSE for the validation dataset will be done using the full edx dataset and predict the ratings on the validation dataset:

```
movie_avgs_v <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

predicted_ratings_v <- mu + validation %>%
```

```
  left_join(movie_avgs_v, by='movieId') %>%
  .$b_i
```

This gives us these RMSEs :

method	RMSE_Test	RMSE_Valid
Just the average	1.0599043	1.0612018
Movie Effect Model	0.9437429	0.9439087

The result is already better but not that good.

So as we have a movie effect, there may be also a user effect. Some people tend to have a more "critical" view on movies than others and some people tend to like all movies they are watching. The way to compute goes the same way as for movies. But to be able to include it in our existing model, we will compute the mean gap between the real rating and the mean minus the movie effect:

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

and then construct our predictions with these two effects:

```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
```

Same goes for the validation dataset:

```
user_avgs_v <- edx %>%
  left_join(movie_avgs_v, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

predicted_ratings_v <- validation %>%
  left_join(movie_avgs_v, by='movieId') %>%
  left_join(user_avgs_v, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
```

This gives us these RMSEs:

method	RMSE_Test	RMSE_Valid
Just the average	1.0599043	1.0612018
Movie Effect Model	0.9437429	0.9439087
Movie + User Effect Model	0.8659319	0.8653488

The result is now much better.

Now what we can observe in the data is that the extreme value of our "effects", movie and user are on movies and users that have not been rated or rated a lot of time. For Movie for example:

title	b_i	n
Besotted (2001)	-3.012465	1
Hi-Line, The (1999)	-3.012465	1
Accused (Anklaget) (2005)	-3.012465	1
Confessions of a Superhero (2007)	-3.012465	1
War of the Worlds 2: The Next Wave (2008)	-3.012465	2

title	b_i	n
Hellhounds on My Trail (1999)	1.487535	1
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.487535	3
Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)	1.487535	2
Shadows of Forgotten Ancestors (1964)	1.487535	1
Money (Argent, L') (1983)	1.487535	1

and for users:

userId	b_u	n
13496	-3.420602	15
48146	-3.239966	20
49862	-3.237758	15
6322	-3.110302	13
63381	-2.975351	13

userId	b_u	n
13524	1.877483	19
18591	1.851187	15
1943	1.815695	15
56965	1.780003	22
7999	1.771917	28

Keeping in mind that the average number of rating by film is 843 and by user is 129, we see that we are really with extreme data and therefore need to do something because large errors are really penalizing our RMSE.

So to take these extreme value, the idea is to lower them by what is called regularization. it means we will introduce a parameter to balance these high values. This parameter can be viewed as a tuning parameter and then we will use cross-validation to validate it but only with the test set (we will not use the validation dataset). The first step to do this is then to find the parameter λ that minimizes the RMSE:

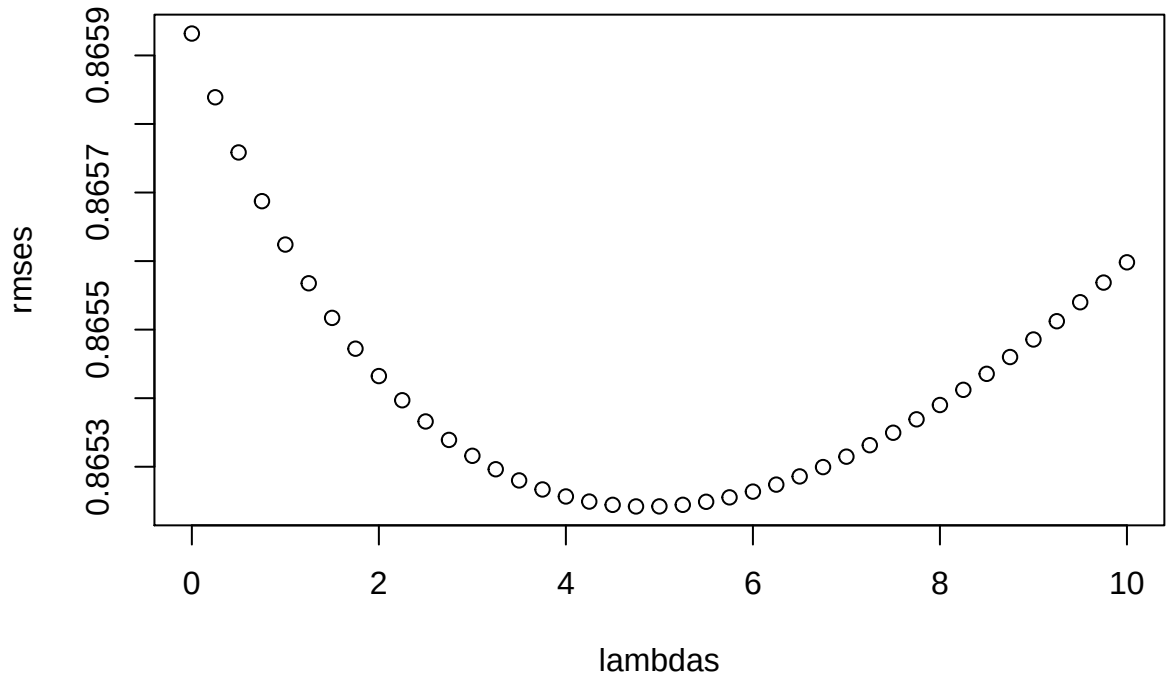
```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <-
```

```
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
```

and then our optimized parameter can be found :

```
lambda <- lambdas[which.min(rmses)]
```

To be sure that we are at a minimum, we can plot it:



We then use this parameter to compute our RMSE on the validation test:

```
movie_reg_avgs_v <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

user_reg_avgs_v <- edx %>%
  left_join(movie_reg_avgs_v, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

predicted_ratings_v <- validation %>%
  left_join(movie_reg_avgs_v, by='movieId') %>%
  left_join(user_reg_avgs_v, by='userId') %>%
```

```
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
```

This gives us these RMSEs:

method	RMSE_Test	RMSE_Valid
Just the average	1.0599043	1.0612018
Movie Effect Model	0.9437429	0.9439087
Movie + User Effect Model	0.8659319	0.8653488
Regularized Movie + User Effect Model	0.8652421	0.8648201

So we already have a good RMSE with this regularization.

We can add another effect based on our data, which is a genre effect. The genre effect is a bit complicated because it can be composed of on or multiple genres as can be seen in Table 1. But here we will keep it like that to see if there is still an effect on our RMSE. So like before we will use the test set to find the optimal lambda and then use it on our validation dataset to compute our RMSEs :

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  b_g <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})

lambda <- lambdas[which.min(rmses)]

movie_reg_avgs_v <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

user_reg_avgs_v <- edx %>%
  left_join(movie_reg_avgs_v, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
```

```r
genres_reg_avgs_v <- edx %>%
  left_join(movie_reg_avgs_v, by='movieId') %>%
  left_join(user_reg_avgs_v, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+lambda))

predicted_ratings_v <- validation %>%
  left_join(movie_reg_avgs_v, by='movieId') %>%
  left_join(user_reg_avgs_v, by='userId') %>%
  left_join(genres_reg_avgs_v, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  .$pred
```

All this gives us a final RMSEs of:

method	RMSE_Test	RMSE_Valid
Just the average	1.0599043	1.0612018
Movie Effect Model	0.9437429	0.9439087
Movie + User Effect Model	0.8659319	0.8653488
Regularized Movie + User Effect Model	0.8652421	0.8648201
Regularized Movie + User + Genres Effect Model	0.8649407	0.8644514

Alternative Method

This second method was the first one I tried but it takes me a really long time to make it work and get an interesting result. I have started from the last part of the Machine Laeraning course and wanted to use SVD method. But the problem with SVD is that it works only on complete matrix. However our rating matrix (with user in rows and movies in columns) is completely sparse. In fact that's the goal of the prediction, fill the gaps.

So I've searched how it could be done and based on the different papers found (mainly https://datajobs.com/data-science-repo/Recommender-Systems-%5BNetflix%5D.pdf, http://sifter.org/~simon/journal/20061211.html and http://nicolas-hug.com/blog/matrix_facto_1) I found that this concept of collaborative filtering (user-item interaction) can be solved by methods like Stochastic Gradient Descent that use only the rating we have to minimize the error between our rating and our prediction.

So I tried to implement a matrix factorisation model on my training data. The idea is to create two matrices (one for the users p and one for the movies q) composed of *latent factors* (could be view as components like in the PCA method) representing the different aspects of each user/movie. And the user-movie effect can then be computed by the product of the corresponding lines of the matrices. The estimated rating can then be written like:

$$\hat{y}_{u,i} = \mu + b_i + b_u + q_i^T p_u$$

To find the minimum we then need to minimize the following equation:

$$\sum_{u,i \in k} (y_{u,i} - \mu - b_i - b_u - q_i^T p_u) + \lambda(||p_u||^2 + ||q_i||^2 + b_u^2 + b_i^2)$$

And as said before this equation can be minimized by using method such as Stochastic Gradient Descent. For each rating in our training set we compute the error between the real rating and the prediction and then update our values b_u, b_i, q_i and p_u in the opposite gradient direction using a learning rate that avoids going to fast and miss the minimum.

So our parameters will be:

```
factors <- 40 # number of latent factors in p and q
lrate <- 0.01 # learning rate for SGD algorithm
reg <- 0.02 # regularization term for SGD algorithm
iter <- 10 # number of SGD iterations
mu <- mean(edx$rating)
```

We then need to initialize some objects to be able to loop correctly on our training set:

```
user <- as.list(sort(unique(train_set$userId)))
movie <- as.list(sort(unique(train_set$movieId)))
nu <- length(user)
nm <- length(movie)

userdf <- data.frame(userId=as.numeric(user)) %>%
  mutate(u=which(user==userId))

moviedf <- data.frame(movieId=as.numeric(movie)) %>%
  mutate(m=which(movie==movieId))

train <- train_set %>% select(userId, movieId, rating)
train <- inner_join(train,userdf)
train <- inner_join(train,moviedf)
test <- test_set %>% select(userId, movieId, rating)
test <- inner_join(test,userdf)
test <- inner_join(test,moviedf)
valid <- validation %>% select(userId, movieId, rating)
valid <- inner_join(valid,userdf)
valid <- inner_join(valid,moviedf)

nrow <- nrow(train)

# initialize user and movie parameters
set.seed(1, sample.kind="Rounding")
p <- replicate(factors,rnorm(nu,0,0.1))
q <- replicate(factors,rnorm(nm,0,0.1))
bu <- rep(0,nu)
bm <- rep(0,nm)
```

We can then run our SGD algorithm:

```
for(n in 1:iter){
#  cat("\niter :",n, " - time : ", format(Sys.time(),"%H:%M:%S"))
  for(row in 1:nrow){
    u <- train[row,"u"]
    m <- train[row,"m"]
    err <- train[row,"rating"] - ( mu + bu[u] + bm[m] + as.numeric(p[u,]%*%q[m,]))
    bu[u] <- bu[u]+lrate*(err-reg*bu[u])
    bm[m] <- bm[m]+lrate*(err-reg*bm[m])
    p[u,] <- p[u,]+lrate*(err*q[m,]-reg*p[u,])
    q[m,] <- q[m,]+lrate*(err*p[u,]-reg*q[m,])
    rm(u,m,err)
  }
}
```

At the end we can compute our prediction using the following function:

```
pred <- function(u,m){
  pred=as.numeric(p[u,]%*%q[m,]+mu+bu[u]+bm[m])
}
```

Applied on our test set and validation set:

```
prediction <- test %>%
  rowwise() %>%
  mutate(pred = pred(u,m))

prediction_v <- valid %>%
  rowwise() %>%
  mutate(pred = pred(u,m))
```

This gives us RMSEs of:

method	RMSE_Test	RMSE_Valid
Just the average	1.0599043	1.0612018
Movie Effect Model	0.9437429	0.9439087
Movie + User Effect Model	0.8659319	0.8653488
Regularized Movie + User Effect Model	0.8652421	0.8648201
Regularized Movie + User + Genres Effect Model	0.8649407	0.8644514
SGD Algorithm	0.8053333	0.8052537

The results is really impressive compared to the method used before. But I am not quite sure of my result as if I run more iterations (each iteration is taking approximately 10-15minutes) the values of my parameters b_u, b_i, q_i and p_u are diverging. But as I have obtained this prediction following the instructions of not using the validation dataset, it seems important to me to include it in this project report.

Conclusion

So our analysis based only on the biases allowed us to obtain a quite good RMSE of 0.8644514 which is already sufficient to obtain good predictions. The second method has overperformed with a RMSE of 0.8052537. So this is the best model I have tested to predict the ratings of our movies.

The next steps could be to include a time effect, considering that each of our parameter can have an evolution. Some people can have a period where they like Sci-Fi movies and some where they don't like.

Considering the SGD algorithm it could also be a good idea to optimize the code to find the best number of iteration that minimizes our error below a certain threshold.

For both method, it could be also a good point to exclude extreme points (movies rating only once or user having rated only one movie) even if with regularization the effect is tempered.