Dear reader,

Thank you for taking time to review this project. The target of this work is to design an algorithm that will forecast the rating given by a list of users to alist of movies stored in the verification set provided by edx, and this using an algorithm trained on a training set, of course without using the verification set before the very last step. The performance of the algorithm is measured by computing the mean difference between the forecast and the real rating, called RMSE and computed with the formula:

$$RMSE = \sqrt{\sum (R - Y)^2}$$

Where R is the actual rating and Y the rating computed by the algorithm.

It is commonly known that to perform such a forecast, the most efficient way is to use the matrix factorization method, directly available in some R packages. However, I personally consider that the target of this project is not to get the lowest RMSE and earn the Netflix price, but to learn R, follow my own ideas, and, at the end, get an home-made and unique algorithm that will lower the RMSE enough to get a good grade (< 0.86 as far as I remember), even if the resulting RMSE would likely have been lower by using some ready-to-use R packages.

Here is how the project is organized:

- Step 0: Download and split of the movie data frame using the code chunk provided by edx
- Step 1: Data exploration, discovery of the edx set
- Step 2: Split of the edx set into partitions for training and test
- Step 3: Design of the algorithm and its different terms
- Step 4: Test of the resulting algorithm on the test set from set 2, measure of the RMSE
- Step 5: Regularization step for the different terms using the test set of step 2, measure of the resulting RMSE
- Step 6: Training of the algorithm with the full edx set
- Step 7: Application of the algorithm from step 6 and regularization factors of step 5 on the validation set provided by edx.
- Step 8: Conclusion and perspectives

Step 0: download movies.dat and split it into edx and validation set using the code chunk provided by edx.

This first part is a simple copy / paste of the code provided by edx to get the training set (dataframe called edx) and the test set (dataframe called 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()
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)

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

Step 1: Data exploration, discovery of the edx set

First, let's browse a bit the data. Each entry is made of the following attributes:

```
head(edx)
```

	userId <int>	movieId <dbl>	rating <dbl>	timestamp <int>	title <chr>	genres <chr>
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

6 rows

Three remarks so far:

- The timestamp as an integer may not be very convenient, a date would likely be more meaningful and could give more information, such as the year or the period of the year. For example we can imagine that Christmas movies get better rating around the Christmas period, or that the ratings fluctuate

with people's seasonal mood fluctuations;

- We have no metadata of the movies (actors, producer, etc.) to make correlations, but the title includes the release year between bracket. This may be a useful information, it will have to be extracted and stored as a new column;
- The genre of the movie can be a combinations of multiple genres, the number of possible combinations will likely be huge.

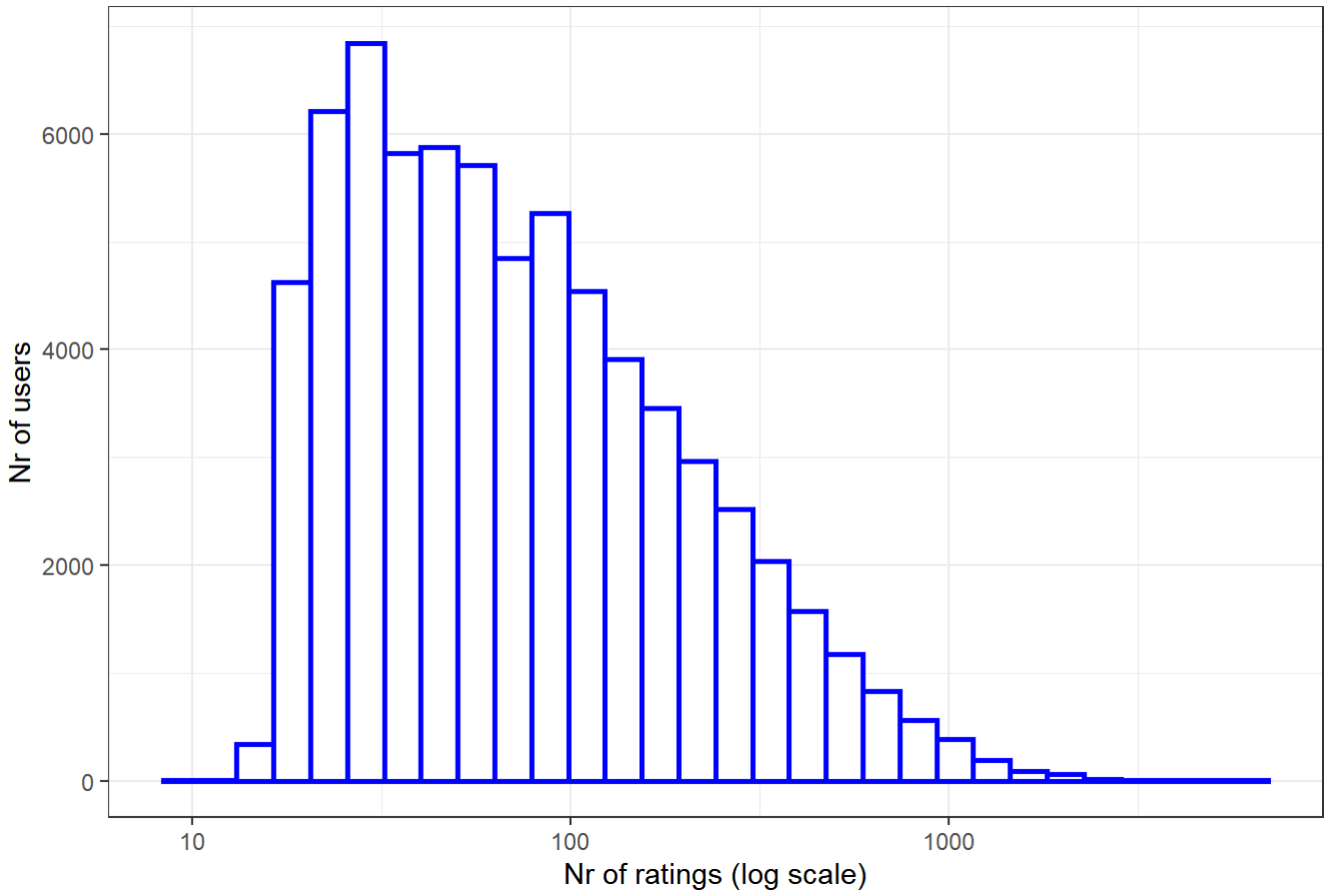
Now, let's have a look to the columns individually.

UserId:

There are 69878 different users in the training set.

Let's see how many ratings they have:

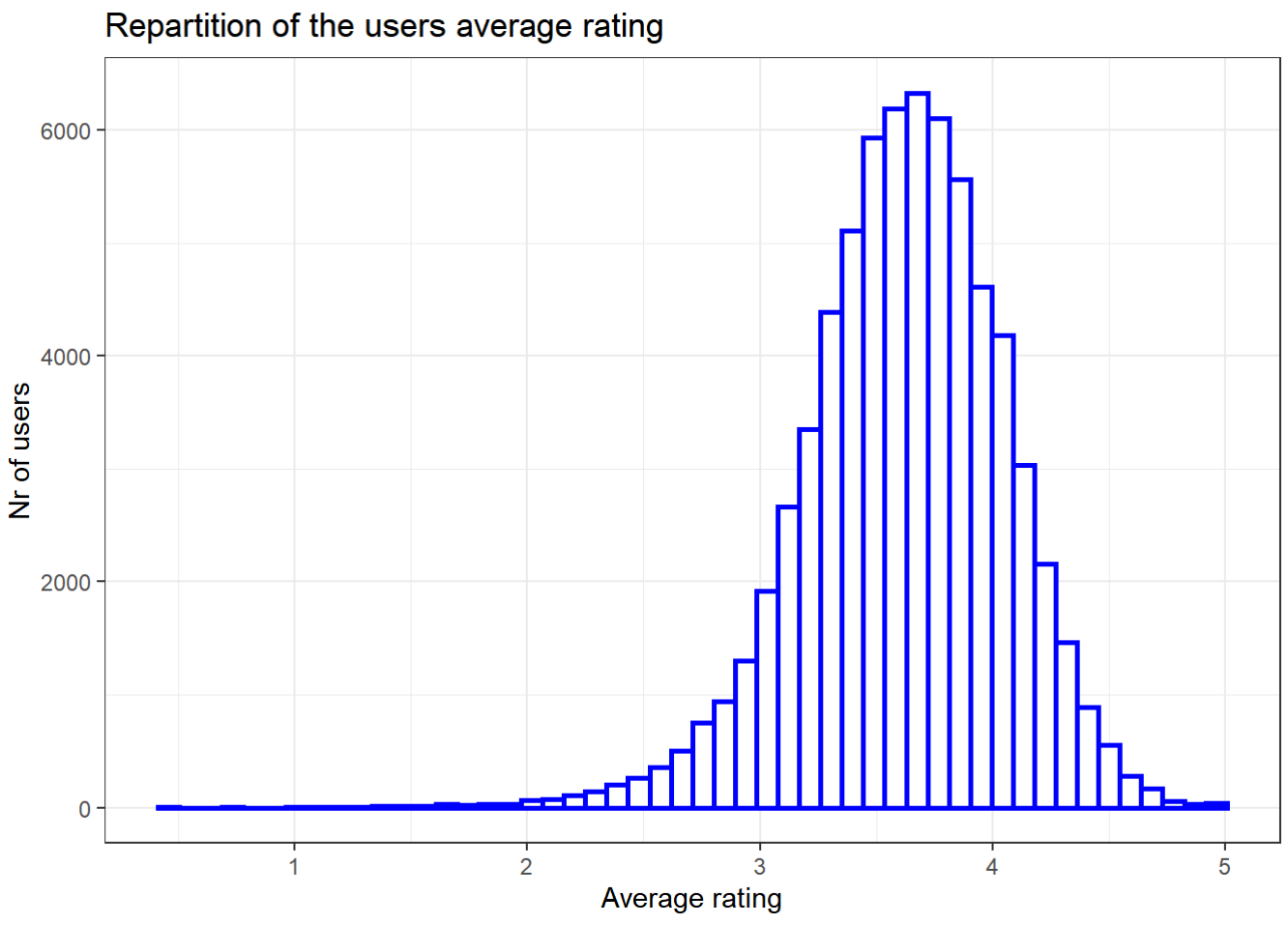
```
User <- edx %>% select(userId, rating) %>%
                group_by(userId) %>%
                summarize(n = n(), avgRating = mean(rating)) %>%
                arrange(desc(n))

User %>% ggplot(aes(n)) +
         geom_histogram(color = "blue", fill = "white", size = 1) +
         scale_x_log10() +
         theme_bw() +
         labs(x = "Nr of ratings (log scale)", y = "Nr of users", title = "Repartition of the number of ratings per user")
```



We can see that very few users have less than 20 ratings, most of the users have rated at least 20, some of them up to a couple of thousand movies. This is a good news because the more rating a user have, the most accurate a machine-learning algorithm will be.

Let's now have a look to the dispatch of average rating per user:

```
User %>% ggplot(aes(avgRating)) +
         geom_histogram(bins = 50, color = "blue", fill = "white", size = 1) +
         theme_bw() +
         labs(x = "Average rating", y = "Nr of users", title = "Repartition of the users average rating")
```

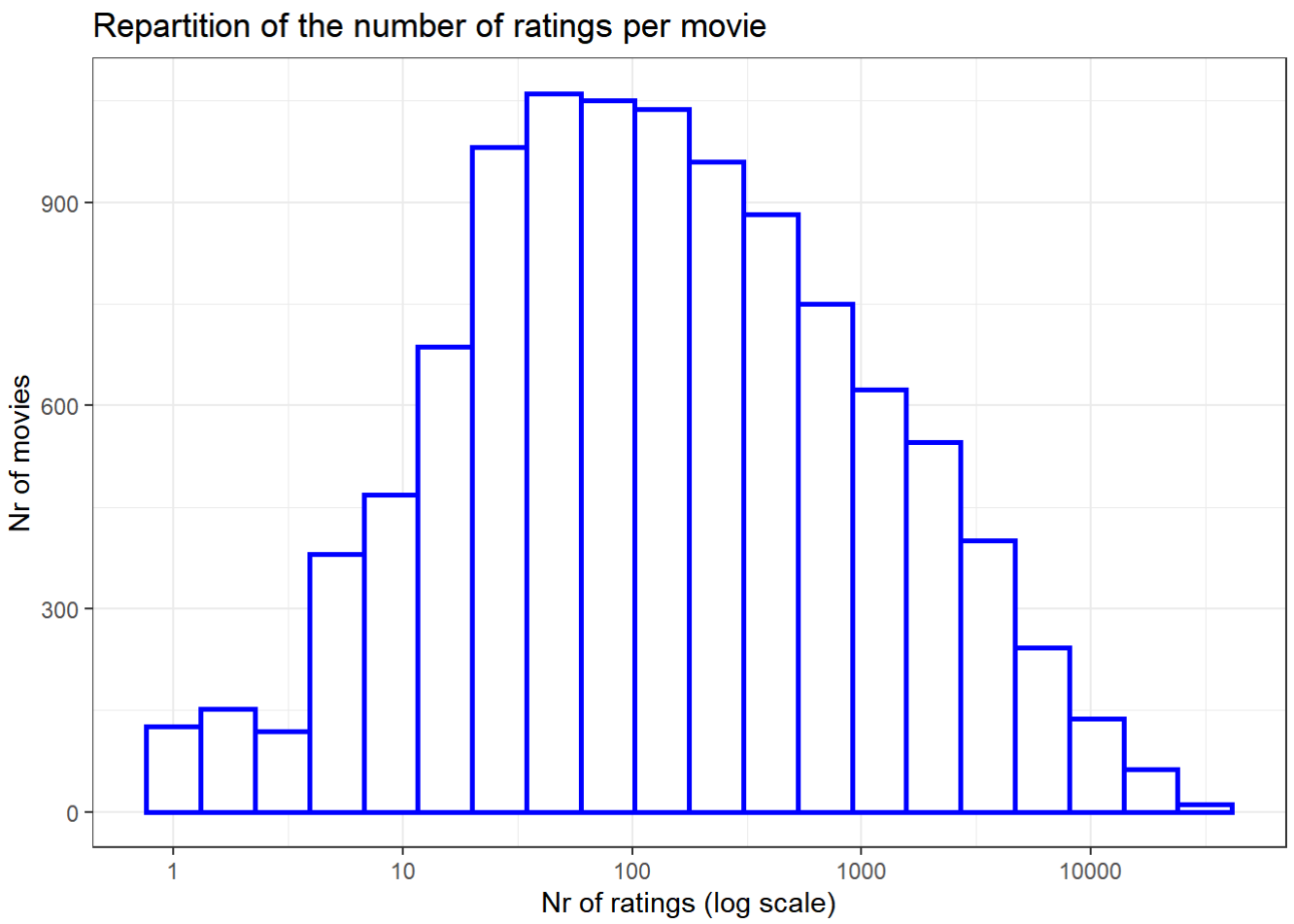


The average ratings appear to be normally distributed, with a maximum around 3.7.

movieId:

There are 10677 different movies in the edx set. Let's see how many ratings they have:

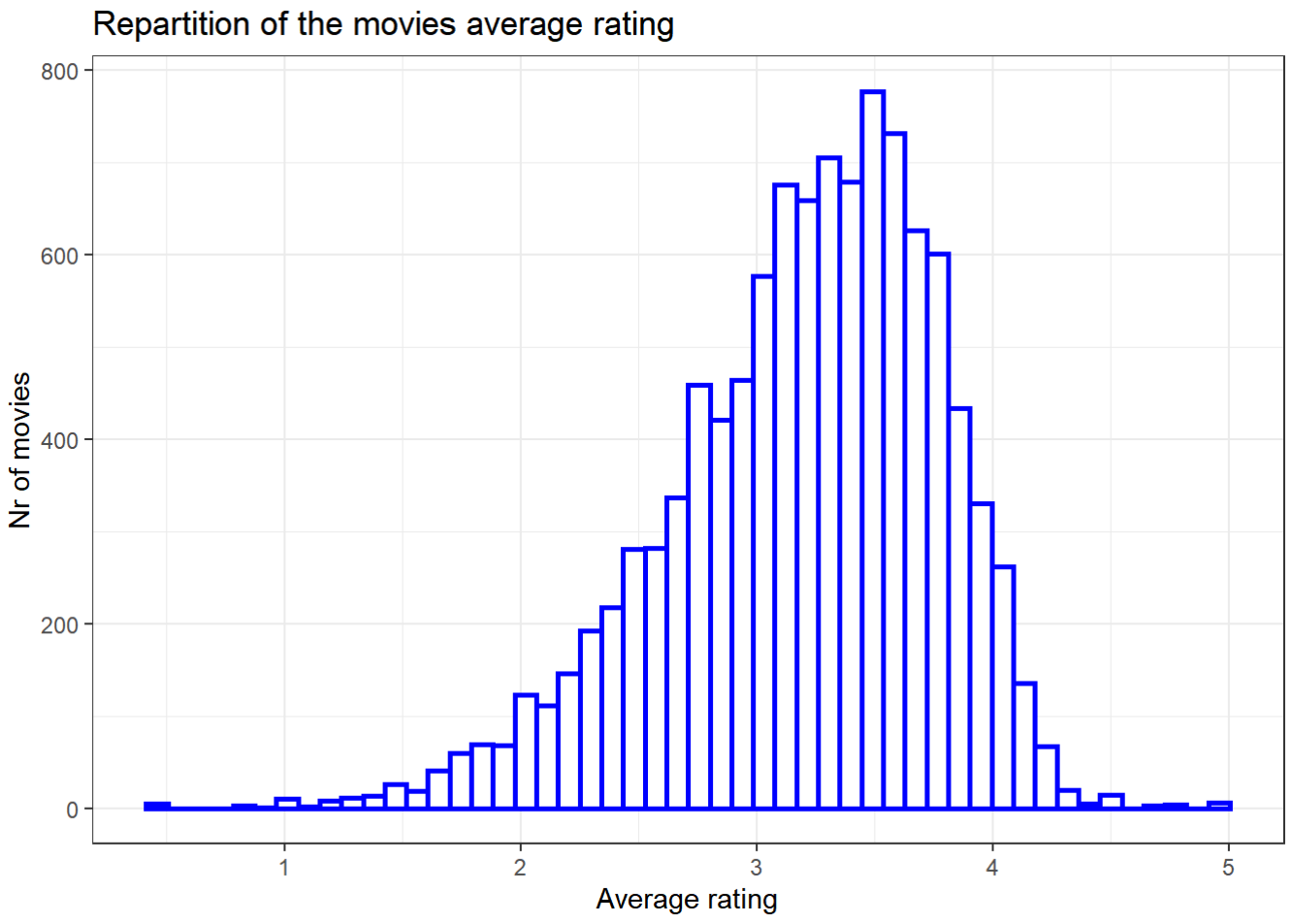
```
Mov <- edx %>%
       select(movieId, rating) %>%
       group_by(movieId) %>%
       summarize(n = n(), avgRating = mean(rating)) %>%
       arrange(desc(n))

Mov %>% ggplot(aes(n)) +
        geom_histogram(bins = 20, color = "blue", fill = "white", size = 1) +
        scale_x_log10() +
        theme_bw() +
        labs(x = "Nr of ratings (log scale)", y = "Nr of movies", title = "Repartition of the number of ratings per movie")
```



The number of ratings per movie is more widely distributed. The highest range seems to be between 30 and 150 ratings but a non-negligeable amount of movies have a low number of rating (< 10).

Let's now have a look to the dispatch of the average rating per movie:

```
Mov %>% ggplot(aes(avgRating)) +
        geom_histogram(bins = 50, color = "blue", fill = "white", size = 1) +
        theme_bw() +
        labs(x = "Average rating", y = "Nr of movies", title = "Repartition of the movies average rating")
```



The average rating of movies is close to a normal distribution, but the bell curve is a bit flattened, average ratings are quite spread across the range 1 to 4.5.

timestamp:

The two first following packages will be usefull to convert the timestamp to a date, the third one is used for graph scales:

```
# Loading libraries

if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
if(!require(anytime)) install.packages("anytime", repos = "http://cran.us.r-project.org")
if(!require(scales)) install.packages("scales", repos = "http://cran.us.r-project.org")

library(lubridate)
library(anytime)
library(scales)

options(digits = 5)
```

Ratings start in January 1995 and end in January 2009. Let's now have a look how the number of ratings are dispatched through time:

```
RatingThroughYear <- edx %>% mutate(TimeFactor = year(anytime(timestamp))) %>%
                             group_by(TimeFactor) %>%
                             summarise(n = n(), avgRating = mean(rating)) %>%
                             mutate(var = "Year")

RatingThroughMonth <- edx %>% mutate(TimeFactor = month(anytime(timestamp))) %>%
                              group_by(TimeFactor) %>%
                              summarise(n = n(), avgRating = mean(rating)) %>%
                              mutate(var = "Month")


RatingThroughTime <- rbind(RatingThroughYear, RatingThroughMonth)

RatingThroughTime %>% ggplot(aes(TimeFactor, n)) +
                      geom_line(color = "blue", size = 1) +
                      theme_bw() +
                      scale_x_continuous(breaks = pretty_breaks()) +
                      xlab("Time (month on the left graph / year on the right graph)") +
                      facet_grid(cols = vars(var), scales = "free") +
                      labs(x = "rating month (left graph) / rating year (right graph)", y = "Nr of ratings", title = "Fluctu
ation of the number of ratings across time")
```

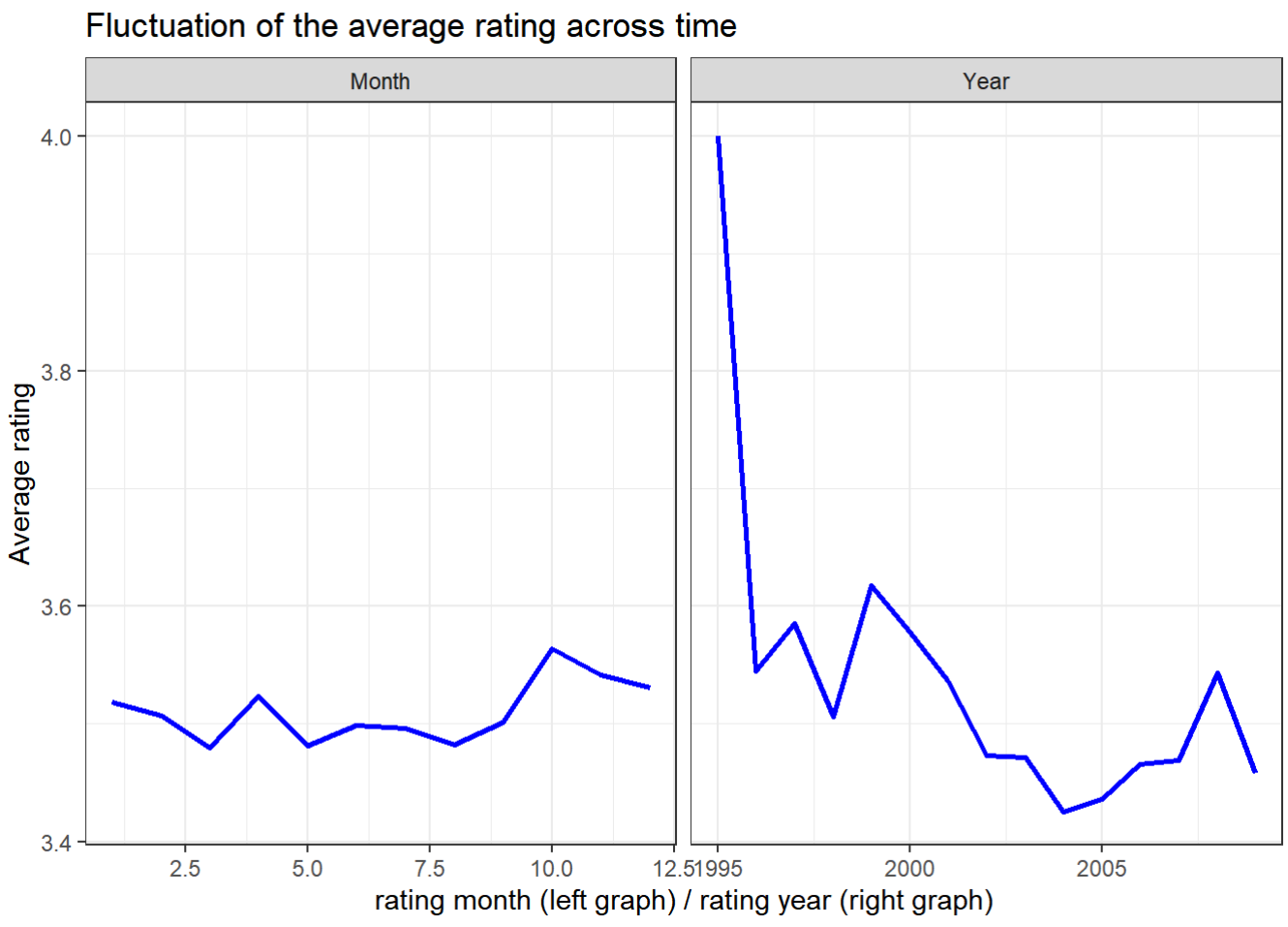
The two graphs here show:

- The fluctuation of the number of ratings across the period 1995-2009 (right graph);
- The fluctuation of ratings across the month of the year (left graph).

We see that even exclusion done of the two extremes (1995 and 2009), we have important fluctuations over the period of ratings we have. However across seasons of the year, we notice a small drop in summer and a small peak in autumn (North hemispher) but seasonal fluctuations appear to moderate.

Now let's have a look to the average ratings across time:

```
RatingThroughTime %>% ggplot(aes(TimeFactor, avgRating)) +
                      geom_line(color = "blue", size = 1) +
                      theme_bw() +
                      xlab("Time (month on the left graph / year on the right graph)") +
                      facet_grid(cols = vars(var), scales = "free") +
                      labs(x = "rating month (left graph) / rating year (right graph)", y = "Average rating", title = "Fluct
uation of the average rating across time")
```



Analog as before, we notice a fluctuation of the average rating across the period 1995 - 2009. Exclusion done of 1995 (not enough ratings this year to be relevant), we have a peak in 1999 and a drop in 2004, with an amplitude of roughly 0.2. Average rating fluctuation across the seasons (left graph) appear to be smoother. We notice a small peak in October but nothing really relevant.

This small analysis shows that the year should be a relevant factor to take into account, but that the rating month can be ignored.

genres:

Movies in the edx set are dispatched in 797 different genres. Let's see which genres are the most and the least used:

```
MovperGenre <- edx %>% select(genres, movieId)

MovperGenre <- unique(MovperGenre) %>%
               group_by(genres) %>%
               summarize(nMovies = n())

MovperGenre %>% arrange(desc(nMovies)) %>% top_n(10)
```

genres <chr>	nMovies <int>
Drama	1815
Comedy	1047
Comedy|Drama	551
Drama|Romance	412
Comedy|Romance	379
Documentary	350
Horror	267
Comedy|Drama|Romance	255
Drama|Thriller	192
Drama|War	173

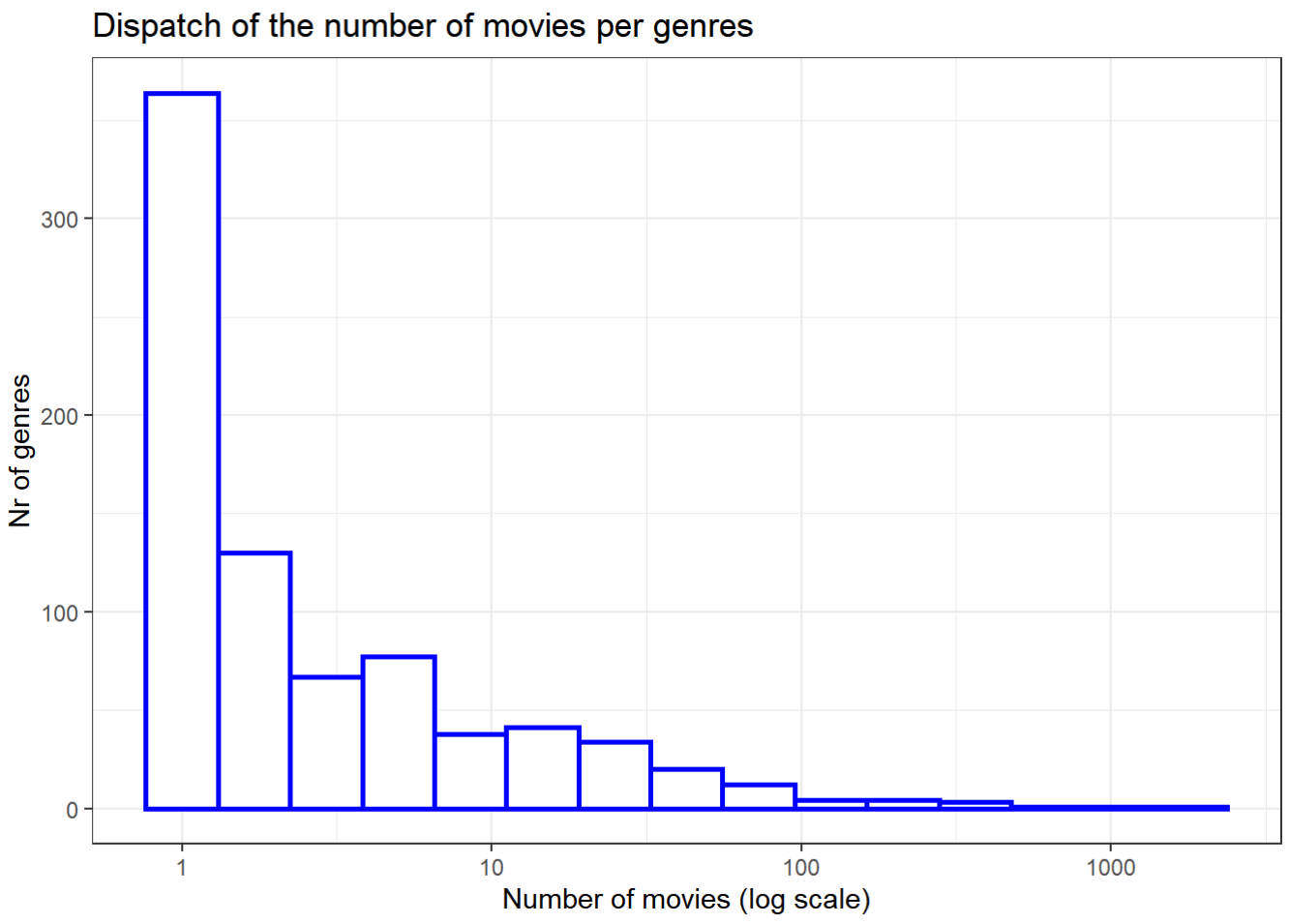
1-10 of 10 rows

```
MovperGenre %>% arrange(nMovies) %>% top_n(-10)
```

genres <chr>	nMovies <int>
(no genres listed)	1
Action|Adventure|Animation|Children|Comedy|Fantasy	1
Action|Adventure|Animation|Children|Comedy|IMAX	1
Action|Adventure|Animation|Children|Fantasy	1
Action|Adventure|Animation|Children|Sci-Fi	1
Action|Adventure|Animation|Comedy|Drama	1
Action|Adventure|Animation|Comedy|Sci-Fi	1
Action|Adventure|Animation|Drama|Fantasy|Sci-Fi	1
Action|Adventure|Animation|Fantasy|Sci-Fi	1
Action|Adventure|Animation|Horror|Sci-Fi	1

1-10 of 364 rows Previous **1** 2 3 4 5 6 … 37 Next

```
MovperGenre %>% ggplot(aes(nMovies)) +
                geom_histogram(bins = 15, color = "blue", fill = "white", size = 1) +
                scale_x_log10() +
                theme_bw() +
                labs(x = "Number of movies (log scale)", y = "Nr of genres", title = "Dispatch of the number of movies per g
enres")
```



It is not a surprise to see that genres made of a combination of 5, 6 or sometimes even 7 of the few differents individual genres, become so specific that they are assigned to a single movie. We still fortunately find some genres more widely used.

Now let's have a look to the average rating per genres, the best rated and least rated:

```
AvgRperGenre <- edx %>% select(genres, rating) %>%
                        group_by(genres) %>%
                        summarize(avgRating = mean(rating))

AvgRperGenre %>% arrange(desc(avgRating)) %>% top_n(10)
```

genres <chr>	**avgRating** <dbl>
Animation|IMAX|Sci-Fi	4.7143
Drama|Film-Noir|Romance	4.3041
Action|Crime|Drama|IMAX	4.2971
Animation|Children|Comedy|Crime	4.2754
Film-Noir|Mystery	4.2395
Crime|Film-Noir|Mystery	4.2168
Film-Noir|Romance|Thriller	4.2165
Crime|Film-Noir|Thriller	4.2102
Crime|Mystery|Thriller	4.1990
Action|Adventure|Comedy|Fantasy|Romance	4.1956

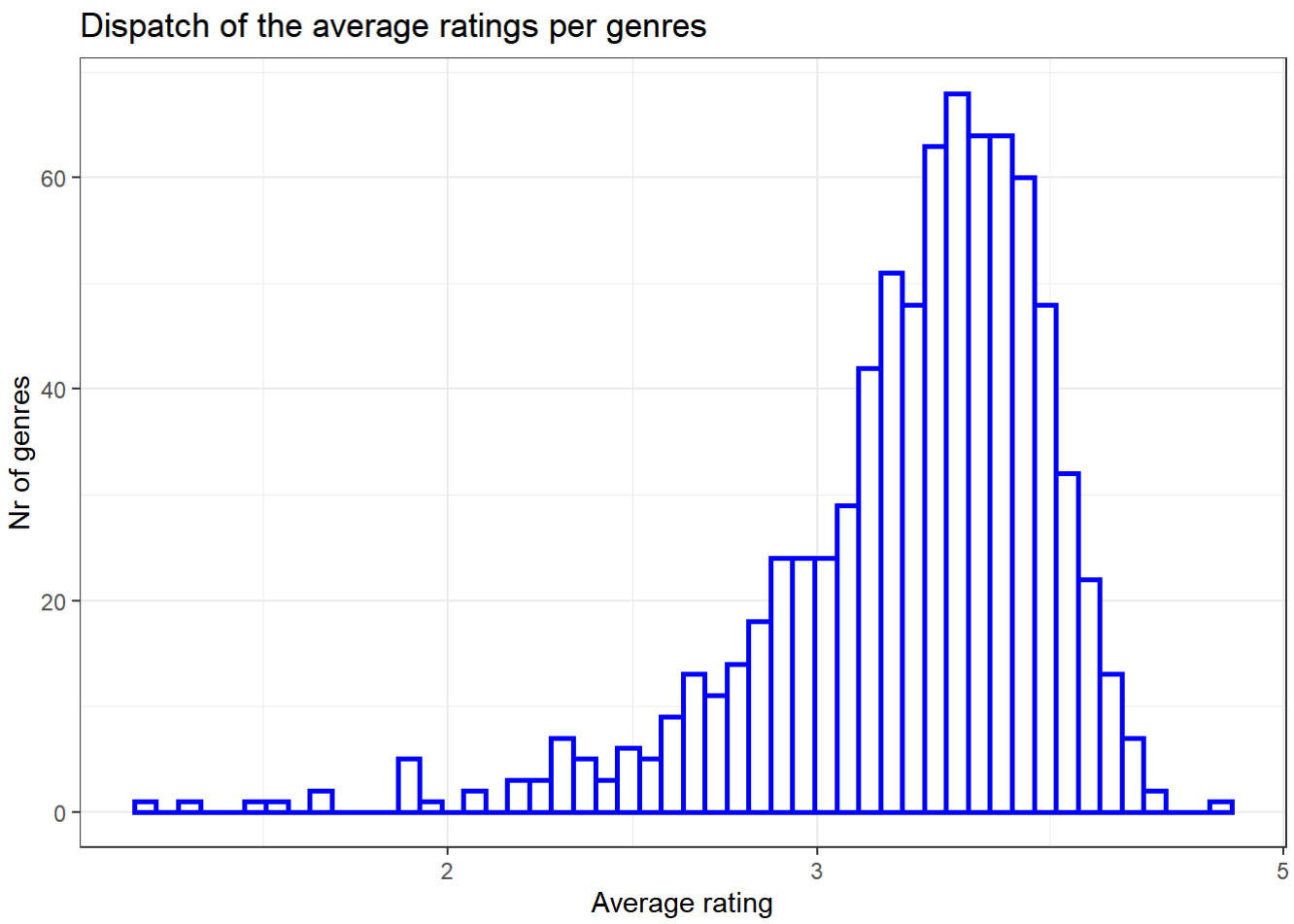
1-10 of 10 rows

```
AvgRperGenre %>% arrange(avgRating) %>% top_n(-10)
```

genres <chr>	**avgRating** <dbl>
Documentary|Horror	1.4491
Action|Animation|Comedy|Horror	1.5000
Action|Horror|Mystery|Thriller	1.6070
Comedy|Film-Noir|Thriller	1.6429
Action|Drama|Horror|Sci-Fi	1.7500
Adventure|Drama|Horror|Sci-Fi|Thriller	1.7512
Action|Adventure|Drama|Fantasy|Sci-Fi	1.9035
Action|Children|Comedy	1.9102
Action|Adventure|Children	1.9150
Adventure|Animation|Children|Fantasy|Sci-Fi	1.9247

1-10 of 10 rows

```
AvgRperGenre %>% ggplot(aes(avgRating)) +
                 geom_histogram(bins = 50, color = "blue", fill = "white", size = 1) +
                 scale_x_log10() +
                 theme_bw() +
                 labs(x = "Average rating", y = "Nr of genres", title = "Dispatch of the average ratings per genres")
```

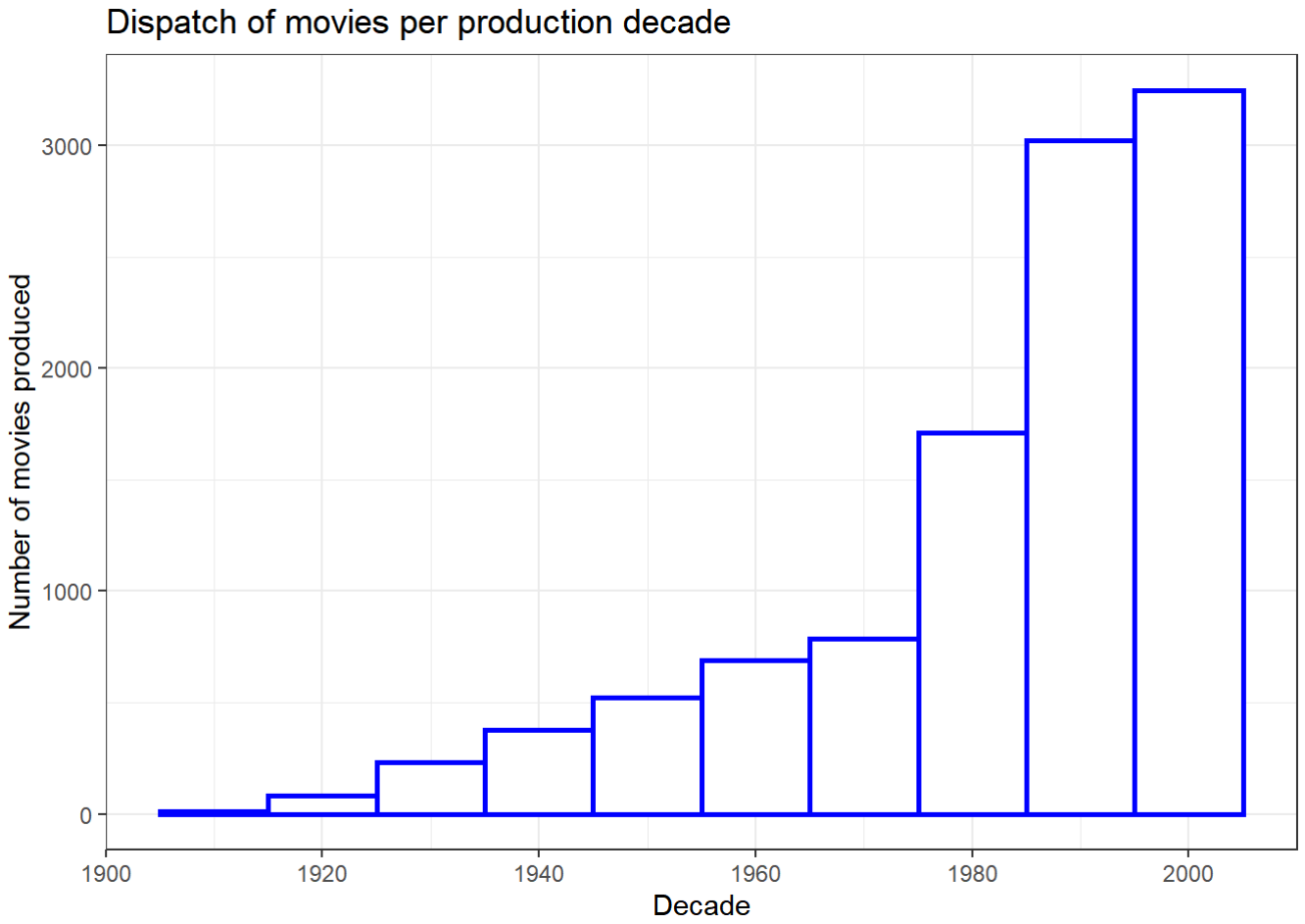


Average rating of the different genres appear to be a normal distribution with a peak around 3.5 and values spread between 2 and 4.5. This diversity of average rating across genres may be a useful information to exploit for genres assigned to different movies.

title:

The title itself is unlikely an information usable in the algorithm, but the production year between braket at the end of each title can be a useful information. We extract it, store it in a new column and group the movies by production decade (we group because some years would not have enough movies to be relevant):

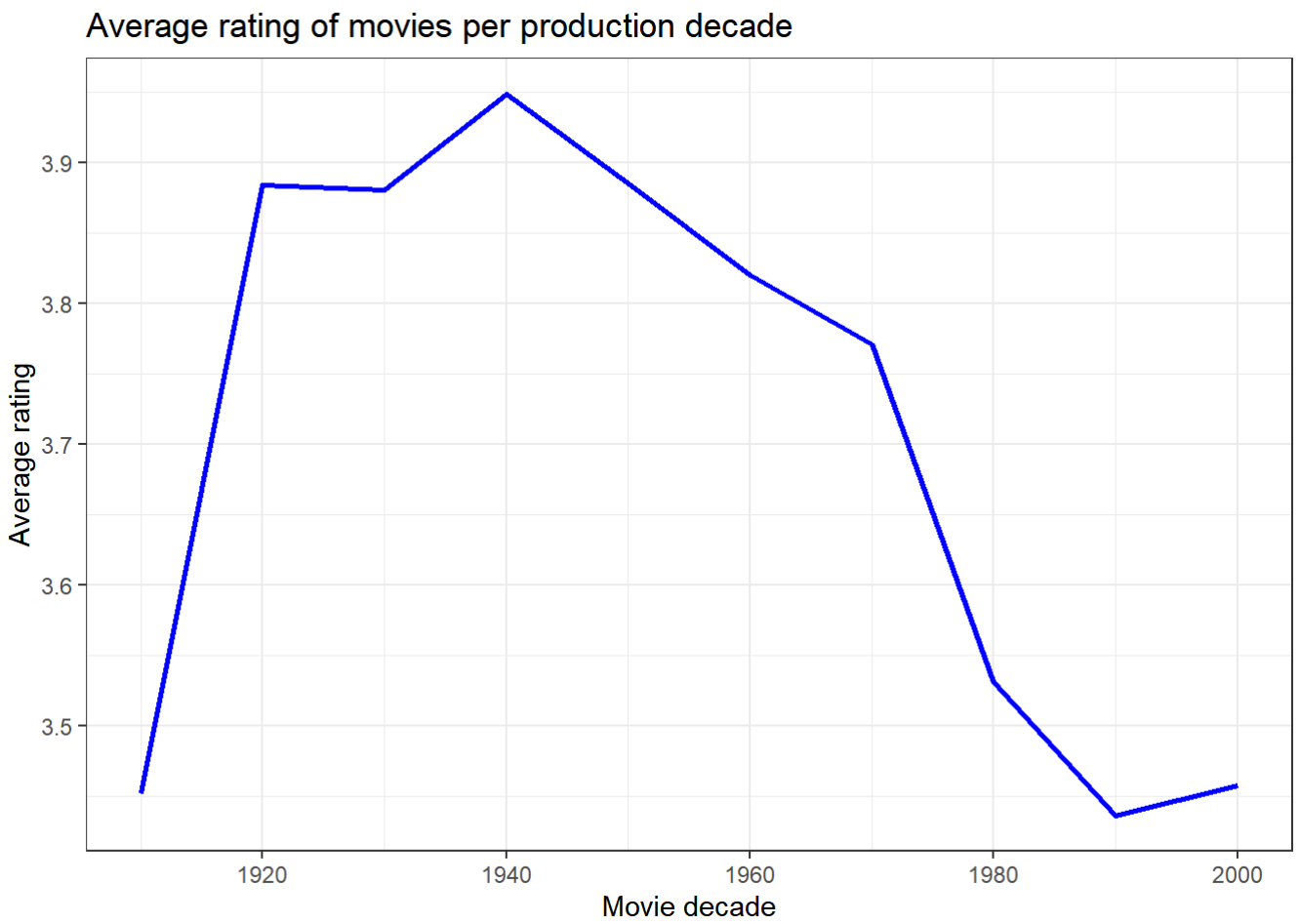
```
MovieDecades <- edx %>% mutate(movieYear = substring(title, nchar(title)-4, nchar(title)-1)) %>%
                        mutate(movieDecade = 10*floor(as.numeric(movieYear)/10)) %>% select(movieDecade, movieId)

MovieDecades <- unique(MovieDecades)

MovieDecades %>% ggplot(aes(movieDecade)) +
                 geom_histogram(bins = 10, color = "blue", fill = "white", size = 1) +
                 theme_bw() +
                 scale_x_continuous(breaks = pretty_breaks()) +
                 labs(x = "Decade", y = "Number of movies produced", title = "Dispatch of movies per production decade")
```



We have movies produced across the full century, the number increases with time, not surprisingly.

Let’s have a look to the average rating per production decade:

```
MovieDecades <- edx %>% mutate(movieYear = substring(title, nchar(title)-4, nchar(title)-1)) %>%
                        mutate(movieDecade = 10*floor(as.numeric(movieYear)/10)) %>%
                        group_by(movieDecade) %>%
                        summarize(n = n(), avgRating = mean(rating))

MovieDecades %>% ggplot(aes(movieDecade, avgRating)) +
                 geom_line(color = "blue", size = 1) +
                 theme_bw() +
                 scale_x_continuous(breaks = pretty_breaks()) +
                 labs(x = "Movie decade", y = "Average rating", title = "Average rating of movies per production decade")
```



We see that there are important fluctuations of average ratings between recent and old movies with a highest average rating for 40’s classics, and a lowest for movies of the 90’s. Amplitude between those two extremes is roughly 0.5. A deeper analysis would likely show important trends for individual users, some prefering old classics and some prefering more recent movies.

We conclude after this small analysis that the model should take into account: the user, the movie, the rating year, the genre and the movie realization decade. The rating month can be excluded from the algorithm, we have seen that it doesn’t seems to be a relevant factor.

Now, serious things can start, we can design the algorithm.

```
# A bit of cleanup first.

rm(MovieDecades, MovperGenre, AvgRperGenre, RatingThroughTime, RatingThroughYear, RatingThroughMonth, Mov, User)
```

Step 2: Split of the edx set into partitions for training and test

The edx dataframe is split in two sets, 90% going to a training set, the 10 remaining percent giving the test set. The training set will be called TrainingSet, the test set will be called TestSet:

```
GenerateSubSet <- function(DF, n) {

  set.seed(2020, sample.kind="Rounding")
  DFSample <- sample(nrow(DF), size = n, replace = FALSE)
  DF[DFSample,]
}


SubSet <- GenerateSubSet(edx, nrow(edx))
indexes <- createDataPartition(y = SubSet$userId, times = 1, p = 0.1, list = FALSE)

TrainingSet <- SubSet[-indexes,]
TestSet <- SubSet[indexes,]

rm(SubSet, indexes)
```

Step 3: Design of the algorithm and its different terms

The principle of the algorithm is to compute an extrapolated rating (Y_hat) by addition of different parameters computed step by step. The equation of the algorithm will be:

$$Y = \sum_{i=1}^{n} Ti$$

Where Ti is the algorithm term number i

Each term is computed by the difference between the average rating for one / several factor(s) (eg. the average rating for a user a particular year), and the average Y_hat for the same factors using the algorithm of the previous step. A new Y_hat is computed by adding this new term to the previous, and so on for all terms. Also at each step we isolate the number of values for each line so that we can later on use the regularization methods.

Each Step is made of a function so that it can be reused after on the full edx set at the end.

Before starting, we extract from the timestamp the rating year, and the movie realization year from the title. Those two values will be useful later.

```
# Function that will extract the year from the timestamp and the decade of the movie production and add to the data set .

ExtractMovieYearAndRatingYear <- function(DF) {

  DF %>% mutate(ratingYear = year(anytime(timestamp))) %>%
         mutate(movieYear = substring(title, nchar(title)-4, nchar(title)-1)) %>%
         mutate(movieDecade = 10*floor(as.numeric(movieYear)/10))

}

  TrainingSet <- ExtractMovieYearAndRatingYear(TrainingSet)
```

Algorithm term 1: As first element, the average value for all movies and all users is computed.

```
# Function that extract the average movie rating of all movies and all users.

GetTheBasis <- function(DF) {

  AvgOverall <- DF %>% summarize(avgOverall = mean(rating)) %>% pull(avgOverall)

}


# Compute for the Training Set the average rating for all movies and all users.

  AvgOverall <- GetTheBasis(TrainingSet)

  TrainingSet <- TrainingSet %>% mutate(Y_hat = AvgOverall)
```

Considering that some movies are better than others, the algorithm should include a term depending of each individual movie.

Algorithm term 2: The average rating of each movie is computed and the difference between the previous Y_hat and the average movie rating makes the second term.

```
# Function that get the number of ratings for each movie, the average rating for each movie for all users and the difference
between this movie and the average rating over all movies.

  GetTheMovieAverageRating <- function(DF) {

  AvgOverall <- GetTheBasis(DF)

  DF %>% group_by(movieId) %>%
         summarize(avgRatingMovie = mean(rating), nMovie = n()) %>%
         mutate(DiffRatingMovie = avgRatingMovie - AvgOverall) %>%
         select(movieId, DiffRatingMovie, nMovie)

}


# Compute for the Training Set the average rating of each movie for all users.

  Mavg <- GetTheMovieAverageRating(TrainingSet)

  TrainingSet <- TrainingSet %>%
                 left_join(Mavg, by = "movieId") %>%
                 mutate(DiffRatingMovie = ifelse(!is.na(DiffRatingMovie), DiffRatingMovie, 0), nMovie = ifelse(!is.na(nMovi
e), nMovie, 0))

  TrainingSet <- TrainingSet %>% mutate(Y_hat = AvgOverall + DiffRatingMovie)
```

Considering that some users tend to give better rating than others (either because they are less critic or because they tend to chose more carefully what they watch the Friday night), the algorithm should include a term depending of each individual user.

Algorithm term 3: The average rating of each user is computed and the difference between the previous Y_hat and the average user rating makes the third term.

```
# Function that computes the number of rating for each user, his averge rating across all movies and the difference between
 his average rating and the overall rating average.

GetTheUserAverageRating <- function(DF) {

  AvgOverall <- GetTheBasis(DF)

  DF %>% group_by(userId) %>%
         summarize(avgRatingUser = mean(rating), avgY_hat = mean(Y_hat), nUser = n()) %>%
         mutate(DiffRatingUser = avgRatingUser - avgY_hat) %>%
         select(userId, DiffRatingUser, nUser)

}


# Compute for the Training Set the average rating of each user for all movies.

  Uavg <- GetTheUserAverageRating(TrainingSet)

  TrainingSet <- TrainingSet %>%
                  left_join(Uavg, by = "userId") %>%
                  mutate(DiffRatingUser = ifelse(!is.na(DiffRatingUser), DiffRatingUser, 0), nUser = ifelse(!is.na(nUser), n
User, 0))

  TrainingSet <- TrainingSet %>% mutate(Y_hat = AvgOverall + DiffRatingMovie + DiffRatingUser)
```

Considering that some users will prefer comedies and some other will prefer action movies, we should include a term depending of the affinity for each genre for each user.

Algorithm term 4: The average rating of each couple genre / user is computed and the difference between the previous Y_hat and the average movie rating makes the fourth term.

```
# Function that computes the number of rating for each user for each genre, his averge rating across all movies of this genr
e and the difference between this average rating and the rating coming from the previous step.

GetTheGenreUserAverageRating <- function(DF) {

  DF %>% group_by(genres, userId) %>%
         summarize(avgGenresUser = mean(rating), avgY_hat = mean(Y_hat), nGenresUser = n()) %>%
         mutate(DiffGenresUser = avgGenresUser - avgY_hat) %>%
         select(genres, userId, DiffGenresUser, nGenresUser)

}


# Compute for the Training Set the average rating of each genre for each user.

  Gavg <- GetTheGenreUserAverageRating(TrainingSet)

  TrainingSet <- TrainingSet %>%
                  left_join(Gavg, by = c("userId", "genres")) %>%
                  mutate(DiffGenresUser = ifelse(!is.na(DiffGenresUser), DiffGenresUser, 0), nGenresUser = ifelse(!is.na(nGe
nresUser), nGenresUser, 0))

  TrainingSet <- TrainingSet %>%
                  mutate(Y_hat = AvgOverall + DiffRatingMovie + DiffRatingUser + DiffGenresUser)
```

Considering that users are humans with ups and downs in their life, good years and bad years, the algorithm should take into account the couple rating year / user, in order to consider the fluctuation that could appear in the average rating of a user across the rating period (1995 - 2009).

Algorithm term 5: The average rating of each user for each year is computed and the difference between the previous Y_hat and the average user / year rating makes the fifth term.

```
# Function that computes the number of rating for each user for each year, his averge rating across all movies he watched th
at year and the difference between this average rating and the rating coming from the previous step.

GetTheUserYearAverageRating <- function(DF) {

  DF %>% group_by(userId, ratingYear) %>%
         summarize(avgUserYear = mean(rating), avgY_hat = mean(Y_hat), nUserYear = n()) %>%
         mutate(DiffUserYear = avgUserYear - avgY_hat) %>%
         select(userId, ratingYear, DiffUserYear, nUserYear)

}


# Compute for the Training Set for each user (and all movies) the average rating per year.

  UYavg <- GetTheUserYearAverageRating(TrainingSet)

  TrainingSet <- TrainingSet %>%
                  left_join(UYavg, by = c("userId", "ratingYear")) %>%
                  mutate(DiffUserYear = ifelse(!is.na(DiffUserYear), DiffUserYear, 0), nUserYear = ifelse(!is.na(nUserYear),
nUserYear, 0))

  TrainingSet <- TrainingSet %>%
                  mutate(Y_hat = AvgOverall + DiffRatingMovie + DiffRatingUser + DiffGenresUser + DiffUserYear)
```

Considering that movies have fluctuation of ratings with time, some may get lower rating with time (some don't age very well, especially special effects don't always age very well), some other may get higher rating with times (nostalgy effect). So the fluctuation of rating with time for a movie should be considered.

Algorithm term 6: The average rating of each movie for each year is computed and the difference between the previous Y_hat and the average movie / year rating makes the sixth term.

```
# Function that computes the number of rating for each movie for each year, his averge rating across all users having watche
d it that year and the difference between this average rating and the rating coming from the previous step.

GetTheMovieYearAverageRating <- function(DF) {

  DF %>% group_by(movieId, ratingYear) %>%
         summarize(avgMovieYear = mean(rating), avgY_hat = mean(Y_hat), nMovieYear = n()) %>%
         mutate(DiffMovieYear = avgMovieYear - avgY_hat) %>%
         select(movieId, ratingYear, DiffMovieYear, nMovieYear)

}


# Compute for the Training Set for each movie (across all users) the average rating per year.

  MYavg <- GetTheMovieYearAverageRating(TrainingSet)

  TrainingSet <- TrainingSet %>%
                  left_join(MYavg, by = c("movieId", "ratingYear")) %>%
                  mutate(DiffMovieYear = ifelse(!is.na(DiffMovieYear), DiffMovieYear, 0), nMovieYear = ifelse(!is.na(nMovieY
ear), nMovieYear, 0))

  TrainingSet <- TrainingSet %>%
                  mutate(Y_hat = AvgOverall + DiffRatingMovie + DiffRatingUser + DiffGenresUser + DiffUserYear + DiffMovieYe
ar)
```

Considering that many users have a favorite period for movies (mute movies from the 20's, classics from the 50's, recent blockbusters, etc.), we can also include a term in the algorithm that takes into account the affinity of a user for a particular decade.

Algorithm term 7: The average rating of each user for each movie decade is computed and the difference between the previous Y_hat and the average user / movie decade makes the seventh term.

```
# Function that computes the number of rating for each user across all movies realized a certain decade, his averge rating f
or those and the difference between this average rating and the rating coming from the previous step.

GetTheUserDecadeAverageRating <- function(DF) {

  DF %>% group_by(userId, movieDecade) %>%
         summarize(avgUserDecade = mean(rating), avgY_hat = mean(Y_hat), nUserDecade = n()) %>%
         mutate(DiffUserDecade = avgUserDecade - avgY_hat) %>%
         select(userId, movieDecade, DiffUserDecade, nUserDecade)

}


# Compute for the Training Set for each user the average rating given for movies of a particular decade.

  DUavg <- GetTheUserDecadeAverageRating(TrainingSet)

  TrainingSet <- TrainingSet %>%
                  left_join(DUavg, by = c("userId", "movieDecade")) %>%
                  mutate(DiffUserDecade = ifelse(!is.na(DiffUserDecade), DiffUserDecade, 0), nUserDecade = ifelse(!is.na(nUs
erDecade), nUserDecade, 0))

  TrainingSet <- TrainingSet %>%
                  mutate(Y_hat = AvgOverall + DiffRatingMovie + DiffRatingUser + DiffGenresUser + DiffUserYear + DiffMovieYe
ar + DiffUserDecade)
```

Step 4: Test of the resulting algorithm on the test set from set 2, measure of the RMSE

Now that we have all terms of the algorithm, we can apply them one by one to the Test set and check, for each step, the resulting RMSE.

First, the term 1 (overall average rating across all movies and all users)

```
# Apply the data of the training to the Test Set.

  TestSet <- ExtractMovieYearAndRatingYear(TestSet)

  TestSet <- TestSet %>% mutate(Y_hat = AvgOverall)

  Results <- data.frame(Step = "Average overall", RMSE = sqrt(mean((TestSet$Y_hat - TestSet$rating)^2)))
  Results
```

Step <fctr>	RMSE <dbl>
Average overall	1.0606

1 row

Secondly, the term 2 (average rating per movie):

```
  TestSet <- TestSet %>%
              left_join(Mavg, by = "movieId") %>%
               mutate(DiffRatingMovie = ifelse(!is.na(DiffRatingMovie), DiffRatingMovie, 0), nMovie = ifelse(!is.na(nMovie),
nMovie, 0)) %>%
              mutate(Y_hat = AvgOverall + DiffRatingMovie)


  Results <- rbind(Results, data.frame(Step = "Average per movie", RMSE = sqrt(mean((TestSet$Y_hat - TestSet$rating)^2))))
  Results
```

Step <fctr>	RMSE <dbl>
Average overall	1.06059
Average per movie	0.94324

2 rows

Thirdly, the term 3 (average rating per user):

```
  TestSet <- TestSet %>%
              left_join(Uavg, by = "userId") %>%
               mutate(DiffRatingUser = ifelse(!is.na(DiffRatingUser), DiffRatingUser, 0), nUser = ifelse(!is.na(nUser), nUse
r, 0)) %>%
              mutate(Y_hat = AvgOverall + DiffRatingMovie + DiffRatingUser)

  Results <- rbind(Results, data.frame(Step = "Average per user", RMSE = sqrt(mean((TestSet$Y_hat - TestSet$rating)^2))))
  Results
```

Step <fctr>	RMSE <dbl>

Step <fctr>	RMSE <dbl>
Average overall	1.06059
Average per movie	0.94324
Average per user	0.86543

3 rows

Fourthly, the term 4 (average rating per user and per genre):

```
  TestSet <- TestSet %>%
              left_join(Gavg, by = c("userId", "genres")) %>%
              mutate(DiffGenresUser = ifelse(!is.na(DiffGenresUser), DiffGenresUser, 0), nGenresUser = ifelse(!is.na(nGenres
User), nGenresUser, 0)) %>%
              mutate(Y_hat = AvgOverall + DiffRatingMovie + DiffRatingUser + DiffGenresUser)

  Results <- rbind(Results, data.frame(Step = "Average per genre and per user", RMSE = sqrt(mean((TestSet$Y_hat - TestSet$ra
ting)^2))))
  Results
```

Step <fctr>	RMSE <dbl>
Average overall	1.06059
Average per movie	0.94324
Average per user	0.86543
Average per genre and per user	0.91901

4 rows

Surprisingly, taking into account the genre increase the RMSE. We have noticed earlier the high number of genres having a low number of movies, so the regularization will likely help. Let's keep this term and see the result after regularization.

Fifthly, the term 5 (average rating per user and per year):

```
  TestSet <- TestSet %>%
              left_join(UYavg, by = c("userId", "ratingYear")) %>%
              mutate(DiffUserYear = ifelse(!is.na(DiffUserYear), DiffUserYear, 0), nUserYear = ifelse(!is.na(nUserYear), nUs
erYear, 0)) %>%
              mutate(Y_hat = AvgOverall + DiffRatingMovie + DiffRatingUser + DiffGenresUser + DiffUserYear)

  Results <- rbind(Results, data.frame(Step = "Average per user per year", RMSE = sqrt(mean((TestSet$Y_hat - TestSet$rating)
^2))))
  Results
```

Step <fctr>	RMSE <dbl>
Average overall	1.06059
Average per movie	0.94324
Average per user	0.86543
Average per genre and per user	0.91901
Average per user per year	0.91721

5 rows

Sixthly, the term 6 (average rating per movie and per year):

```
  TestSet <- TestSet %>%
              left_join(MYavg, by = c("movieId", "ratingYear")) %>%
               mutate(DiffMovieYear = ifelse(!is.na(DiffMovieYear), DiffMovieYear, 0), nMovieYear = ifelse(!is.na(nMovieYea
r), nMovieYear, 0)) %>%
              mutate(Y_hat = AvgOverall + DiffRatingMovie + DiffRatingUser + DiffGenresUser + DiffUserYear + DiffMovieYear)

  Results <- rbind(Results, data.frame(Step = "Average per movie per year", RMSE = sqrt(mean((TestSet$Y_hat - TestSet$ratin
g)^2))))
  Results
```

Step <fctr>	RMSE <dbl>
Average overall	1.06059
Average per movie	0.94324
Average per user	0.86543

Step <fctr>	RMSE <dbl>
Average per genre and per user	0.91901
Average per user per year	0.91721
Average per movie per year	0.91554

6 rows

Seventhly, the term 7 (average rating per user and per movie decade):

```
  TestSet <- TestSet %>%
              left_join(DUavg, by = c("userId", "movieDecade")) %>%
              mutate(DiffUserDecade = ifelse(!is.na(DiffUserDecade), DiffUserDecade, 0), nUserDecade = ifelse(!is.na(nUserDe
cade), nUserDecade, 0)) %>%
               mutate(Y_hat = AvgOverall + DiffRatingMovie + DiffRatingUser + DiffGenresUser + DiffUserYear + DiffMovieYear+
 DiffUserDecade)

Results <- rbind(Results, data.frame(Step = "Average per user per movie decade", RMSE = sqrt(mean((TestSet$Y_hat - TestSet$r
ating)^2))))
Results
```

Step <fctr>	RMSE <dbl>
Average overall	1.06059
Average per movie	0.94324
Average per user	0.86543
Average per genre and per user	0.91901
Average per user per year	0.91721
Average per movie per year	0.91554
Average per user per movie decade	0.91346

7 rows

Step 5: Regularization step for the different terms using the test set of step 2, measure of the resulting RMSE

For each of the 6 last terms of the algorithm, we need to add a regularization factor to give a different weight to users / movies / genres / years with a low number of values and with a high number of values. A regularization factor is a number added to the denominator as following:

$$Y = T1 + \sum_{i=2}^{7} Ti * \frac{n_{Ti}}{n_{Ti}+k_{Ti}}$$

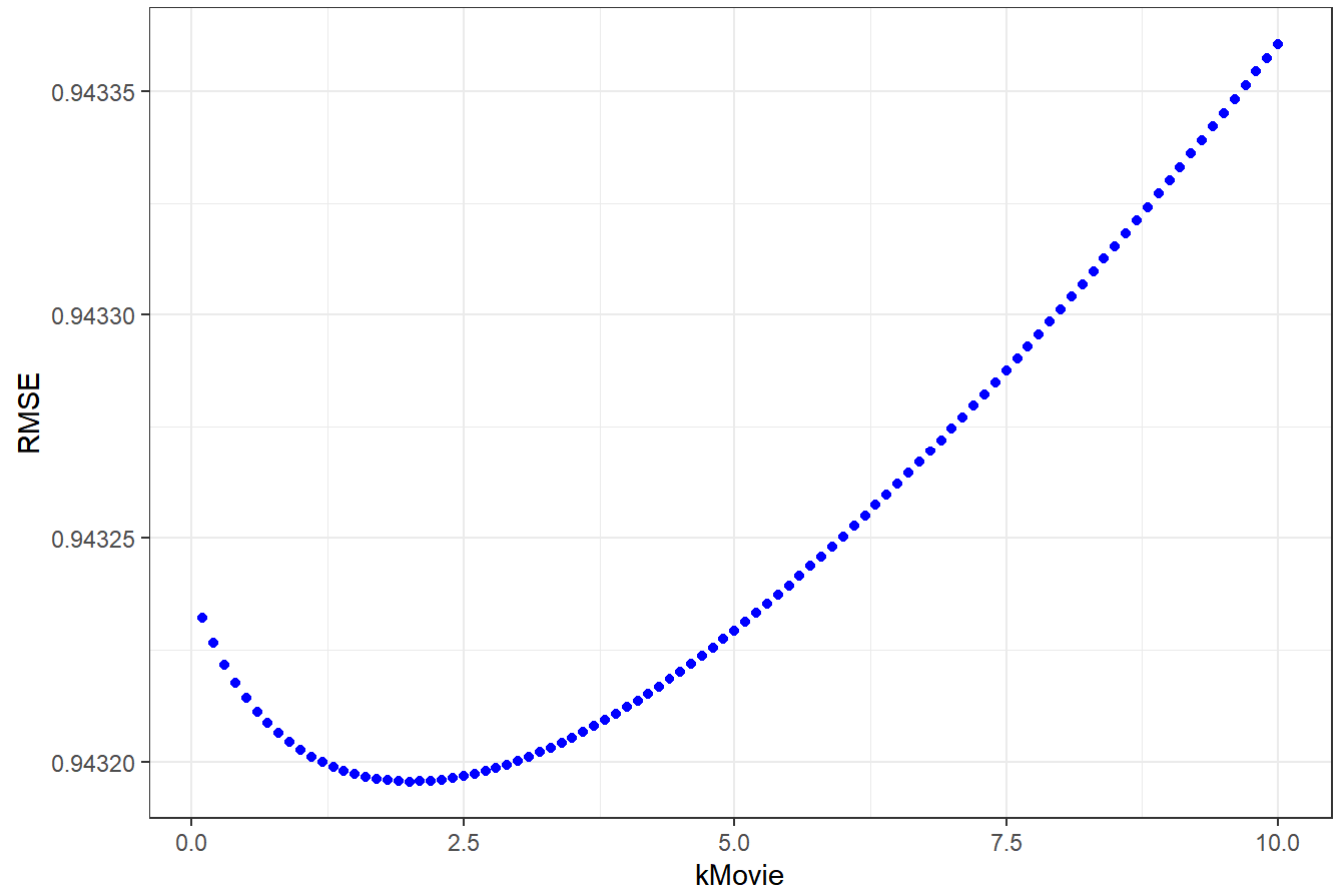
where Ti is the term i (i between 2 to 7, the term 1 has only 1 value and cannot be regularized), n_{Ti} is the number of values for this terms and k_{Ti} the regularization factor. For this purpose, for each of of the 6 last terms, we always have isolated the number of values for each line.

For each of the 6 steps, we compute the optimal regularization factors, the one giving the lowest RMSE. The higher number of values we have per line for a term, the lower we expect the regularization factor to be. For example the optimal regularization factor for term 2 (average movies rating) or term 3 (average user rating) is expected to be lower than the term 5 (average user rating per year) or term 6 (average movie rating per year). This is to be taken into account to determine the range of values where the optimal value is to be searched.

The optimal regularization factor of each term is computed using a loop where the RMSE is computed with the fluctuation of k. The optimal k value being the one giving the lowest RMSE.

Let’s start by term 2:

```
## Step of determination of regularization factors.

FineTune <- data.frame(fact = "0", k = 0, RMSE = 0, stringsAsFactors = FALSE)

# Determination of the optimal regularization for movie average rating.

for(kMovie in seq(0.1, 10, 0.1)) {

    TestSet <- TestSet %>% mutate(Y_hat = AvgOverall +
                                    (nMovie*DiffRatingMovie)/(nMovie + kMovie))

    RMSE <- sqrt(mean((TestSet$rating - TestSet$Y_hat)^2))

    FineTune <- rbind(FineTune, data.frame(fact = "kMovie", k = kMovie, RMSE = RMSE))

  }

FineTune %>% filter(fact == "kMovie") %>%
             ggplot(aes(k, RMSE)) +
             geom_point(color = "blue") +
             geom_point(color = "blue") +
             theme_bw() +
             labs(x = "kMovie", y = "RMSE", title = "Determination of the optimal regularization factor for movies average r
ating")
```



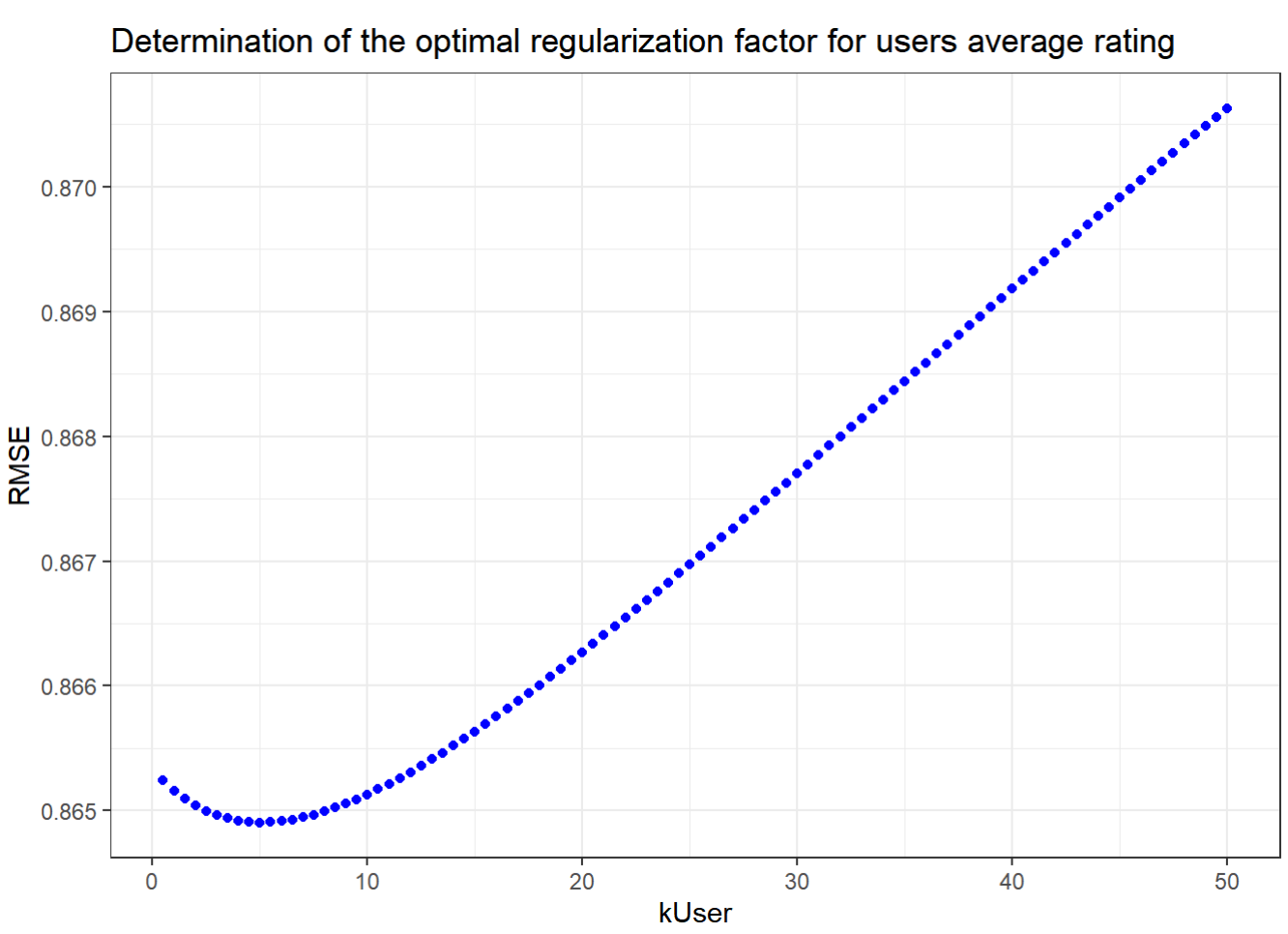

```
M <- FineTune %>% filter(fact == "kMovie") %>%
                  summarize(min(RMSE)) %>%
                  filter(row_number()==1) %>% pull()

kMovieOpt <- FineTune %>% filter(fact == "kMovie" & RMSE == M) %>% pull(k)

SummaryOptimalk <- data.frame(Factor = "kMovie", Optimalk = kMovieOpt, RMSE = M)

SummaryOptimalk
```

Factor <fctr>	Optimalk <dbl>	RMSE <dbl>
kMovie	2	0.9432

1 row

Now for term 3:

```
# Determination of the optimal regularization for user average rating.

for(kUser in seq(0.5, 50, 0.5)) {

  TestSet <- TestSet %>% mutate(Y_hat = AvgOverall +
                                  (nMovie*DiffRatingMovie)/(nMovie + kMovieOpt) +
                                  (nUser*DiffRatingUser)/(nUser + kUser))

  RMSE <- sqrt(mean((TestSet$rating - TestSet$Y_hat)^2))

  FineTune <- rbind(FineTune, data.frame(fact = "kUser", k = kUser, RMSE = RMSE))

}

FineTune %>% filter(fact == "kUser") %>%
             ggplot(aes(k, RMSE)) +
             geom_point(color = "blue") +
             geom_point(color = "blue") +
             theme_bw() +
             labs(x = "kUser", y = "RMSE", title = "Determination of the optimal regularization factor for users average rat
ing")
```



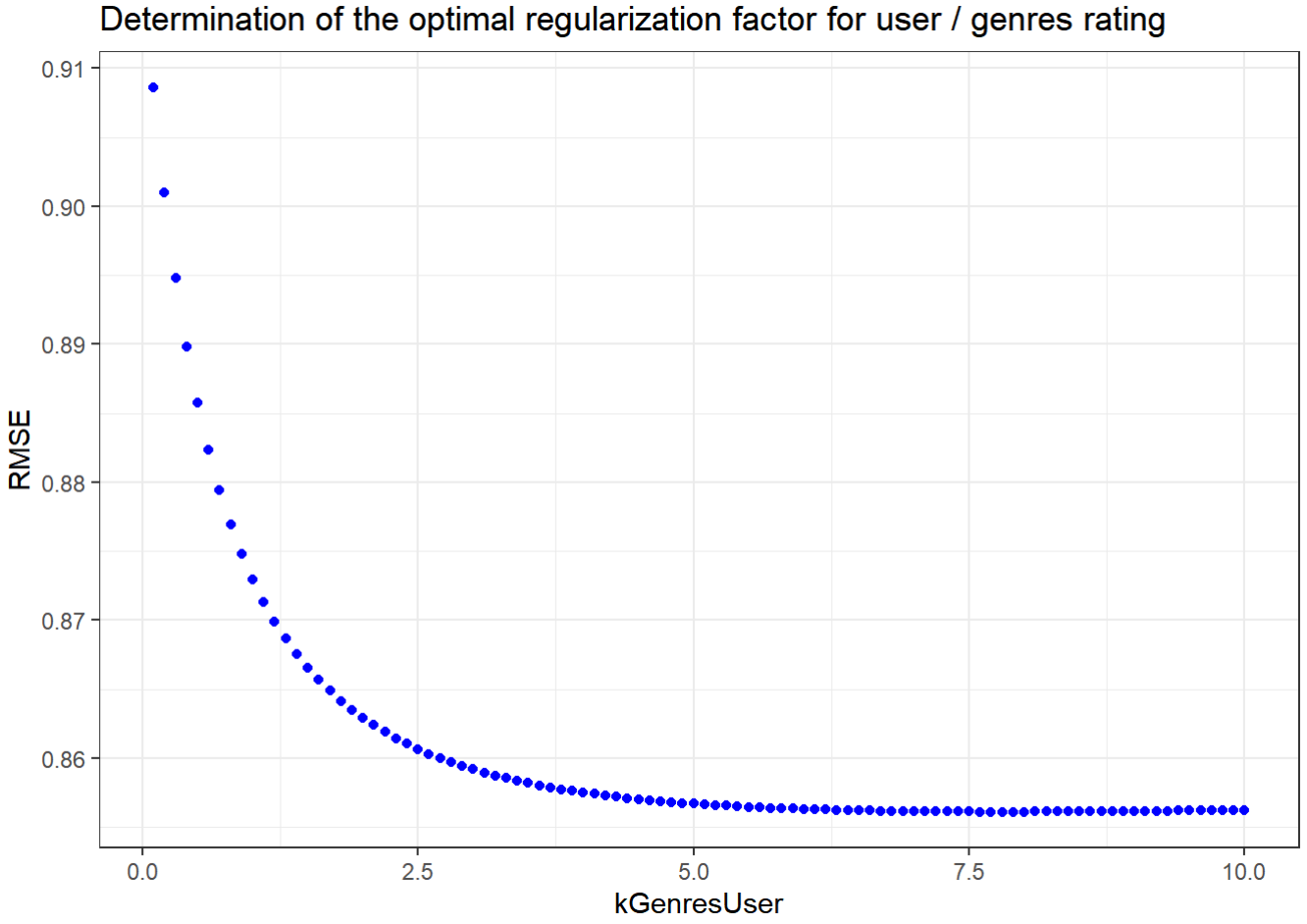

```
M <- FineTune %>% filter(fact == "kUser") %>%
                  summarize(min(RMSE)) %>%
                  filter(row_number()==1) %>% pull()

kUserOpt <- FineTune %>% filter(fact == "kUser" & RMSE == M) %>% pull(k)

SummaryOptimalk <- rbind(SummaryOptimalk, data.frame(Factor = "kUser", Optimalk = kUserOpt, RMSE = M))

SummaryOptimalk
```

Factor	Optimalk	RMSE
<fctr>	<dbl>	<dbl>
kMovie	2	0.94320
kUser	5	0.86491

2 rows

For term 4:

```
# Determination of the optimal regularization for couple user / genre average rating.

for(kGenresUser in seq(0.1, 10, 0.1)) {

  TestSet <- TestSet %>%
              mutate(Y_hat = AvgOverall +
                     (nMovie*DiffRatingMovie)/(nMovie + kMovieOpt) +
                     (nUser*DiffRatingUser)/(nUser + kUserOpt) +
                     (nGenresUser*DiffGenresUser)/(nGenresUser + kGenresUser))

  RMSE <- sqrt(mean((TestSet$rating - TestSet$Y_hat)^2))

  FineTune <- rbind(FineTune, data.frame(fact = "kGenresUser", k = kGenresUser, RMSE = RMSE))

}

FineTune %>% filter(fact == "kGenresUser") %>%
             ggplot(aes(k, RMSE)) +
             geom_point(color = "blue") +
                        theme_bw() +
                         labs(x = "kGenresUser", y = "RMSE", title = "Determination of the optimal regularization factor for
 user / genres rating")
```




```
M <- FineTune %>% filter(fact == "kGenresUser") %>%
                  summarize(min(RMSE)) %>% filter(row_number()==1) %>% pull()
kGenresUserOpt <- FineTune %>% filter(fact == "kGenresUser" & RMSE == M) %>% pull(k)

SummaryOptimalk <- rbind(SummaryOptimalk, data.frame(Factor = "kGenresUser", Optimalk = kGenresUserOpt, RMSE = M))


SummaryOptimalk
```

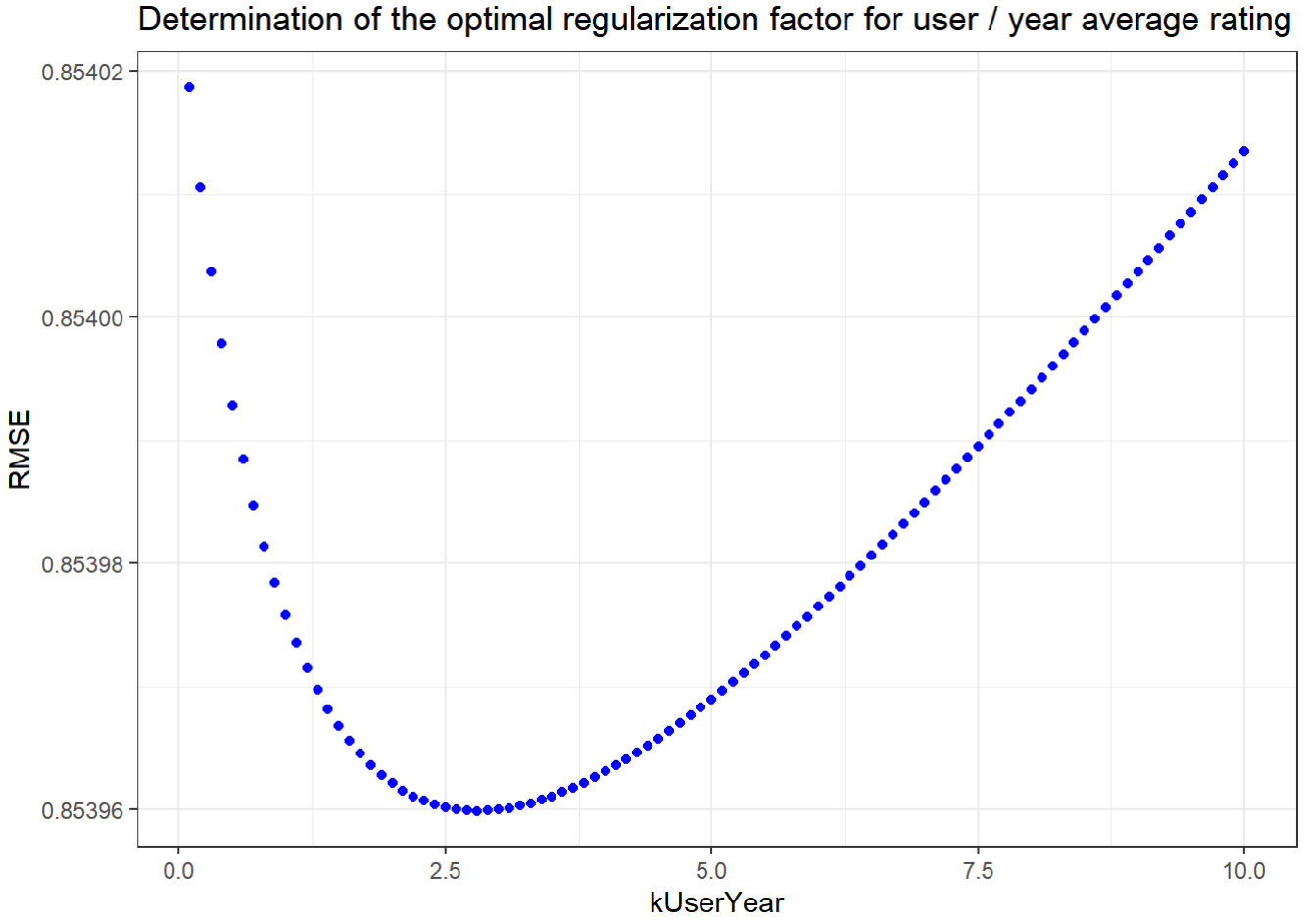
Factor <fctr>	**Optimalk** <dbl>	**RMSE** <dbl>
kMovie	2.0	0.94320
kUser	5.0	0.86491
kGenresUser	7.8	0.85611

3 rows

We notice here that even if the genre was earlier increasing the RMSE, when adding the regularization term, it finally lower the RMSE and should be kept.

For term 5:

```
# Determination of the optimal regularization for couple user / rating year average rating.

for(kUserYear in seq(0.1, 10, 0.1)) {

  TestSet <- TestSet %>%
              mutate(Y_hat = AvgOverall +
                     (nMovie*DiffRatingMovie)/(nMovie + kMovieOpt) +
                     (nUser*DiffRatingUser)/(nUser + kUserOpt) +
                     (nGenresUser*DiffGenresUser)/(nGenresUser + kGenresUserOpt) +
                     (nUserYear*DiffUserYear)/(nUserYear + kUserYear))

  RMSE <- sqrt(mean((TestSet$rating - TestSet$Y_hat)^2))

  FineTune <- rbind(FineTune, data.frame(fact = "kUserYear", k = kUserYear, RMSE = RMSE))

}

FineTune %>% filter(fact == "kUserYear") %>%
             ggplot(aes(k, RMSE)) +
             geom_point(color = "blue") +
             theme_bw() +
             labs(x = "kUserYear", y = "RMSE", title = "Determination of the optimal regularization factor for user / year a
verage rating")
```



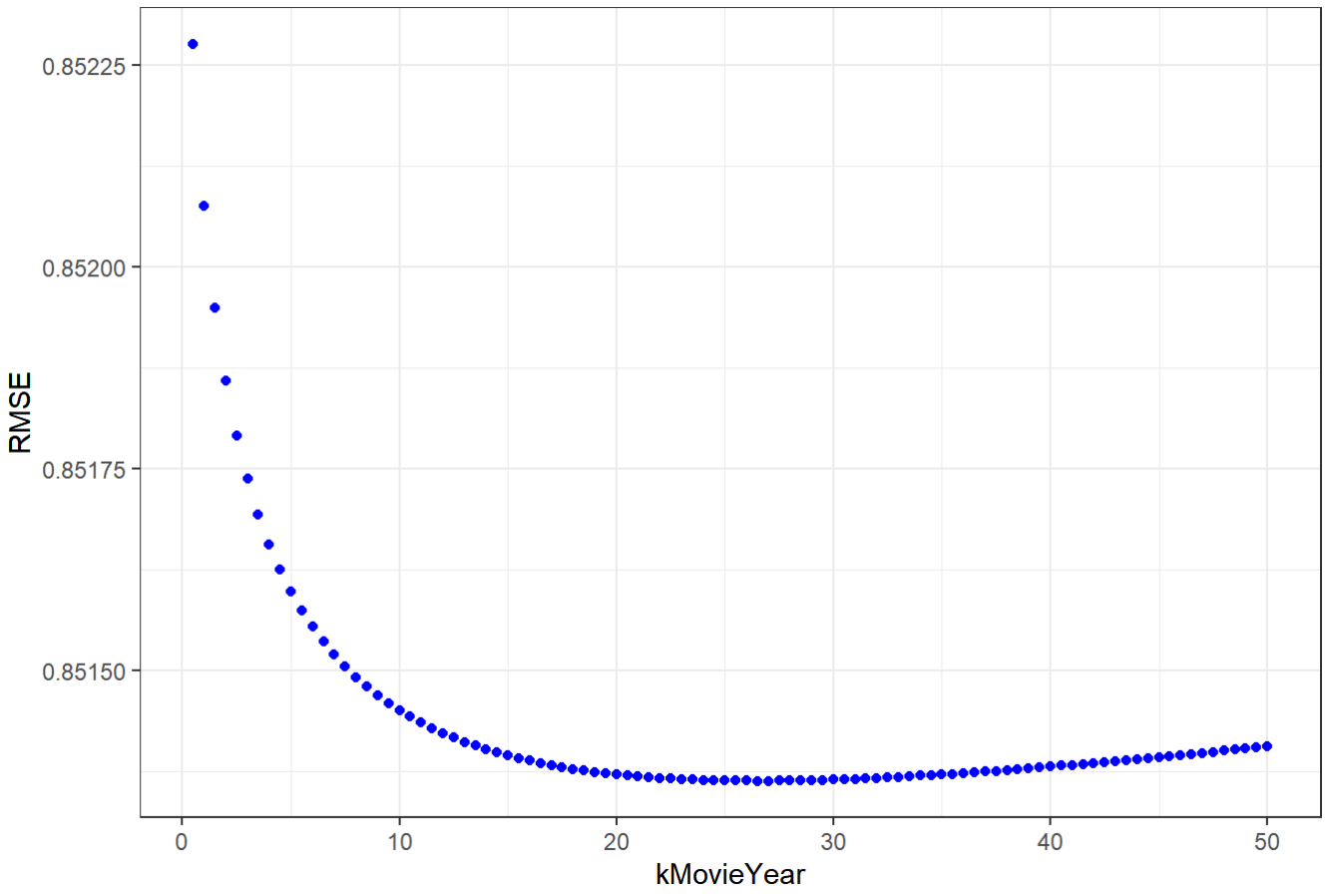

```
M <- FineTune %>% filter(fact == "kUserYear") %>%
                  summarize(min(RMSE)) %>% filter(row_number()==1) %>% pull()

kUserYearOpt <- FineTune %>% filter(fact == "kUserYear" & RMSE == M) %>% pull(k)

SummaryOptimalk <- rbind(SummaryOptimalk, data.frame(Factor = "kUserYear", Optimalk = kUserYearOpt, RMSE = M))

SummaryOptimalk
```

Factor <fctr>	Optimalk <dbl>	RMSE <dbl>
kMovie	2.0	0.94320
kUser	5.0	0.86491
kGenresUser	7.8	0.85611
kUserYear	2.8	0.85396

4 rows

For term 6:

```
# Determination of the optimal regularization for couple movie / rating year average rating.

for(kMovieYear in seq(0.5, 50, 0.5)) {

  TestSet <- TestSet %>%
              mutate(Y_hat = AvgOverall +
                     (nMovie*DiffRatingMovie)/(nMovie + kMovieOpt) +
                     (nUser*DiffRatingUser)/(nUser + kUserOpt) +
                     (nGenresUser*DiffGenresUser)/(nGenresUser + kGenresUserOpt) +
                     (nUserYear*DiffUserYear)/(nUserYear + kUserYearOpt) +
                     (nMovieYear*DiffMovieYear)/(nMovieYear + kMovieYear))

  RMSE <- sqrt(mean((TestSet$rating - TestSet$Y_hat)^2))

  FineTune <- rbind(FineTune, data.frame(fact = "kMovieYear", k = kMovieYear, RMSE = RMSE))

}

FineTune %>% filter(fact == "kMovieYear") %>%
             ggplot(aes(k, RMSE)) +
             geom_point(color = "blue") +
             theme_bw() +
             labs(x = "kMovieYear", y = "RMSE", title = "Determination of the optimal regularization factor for movie / year
average rating")
```



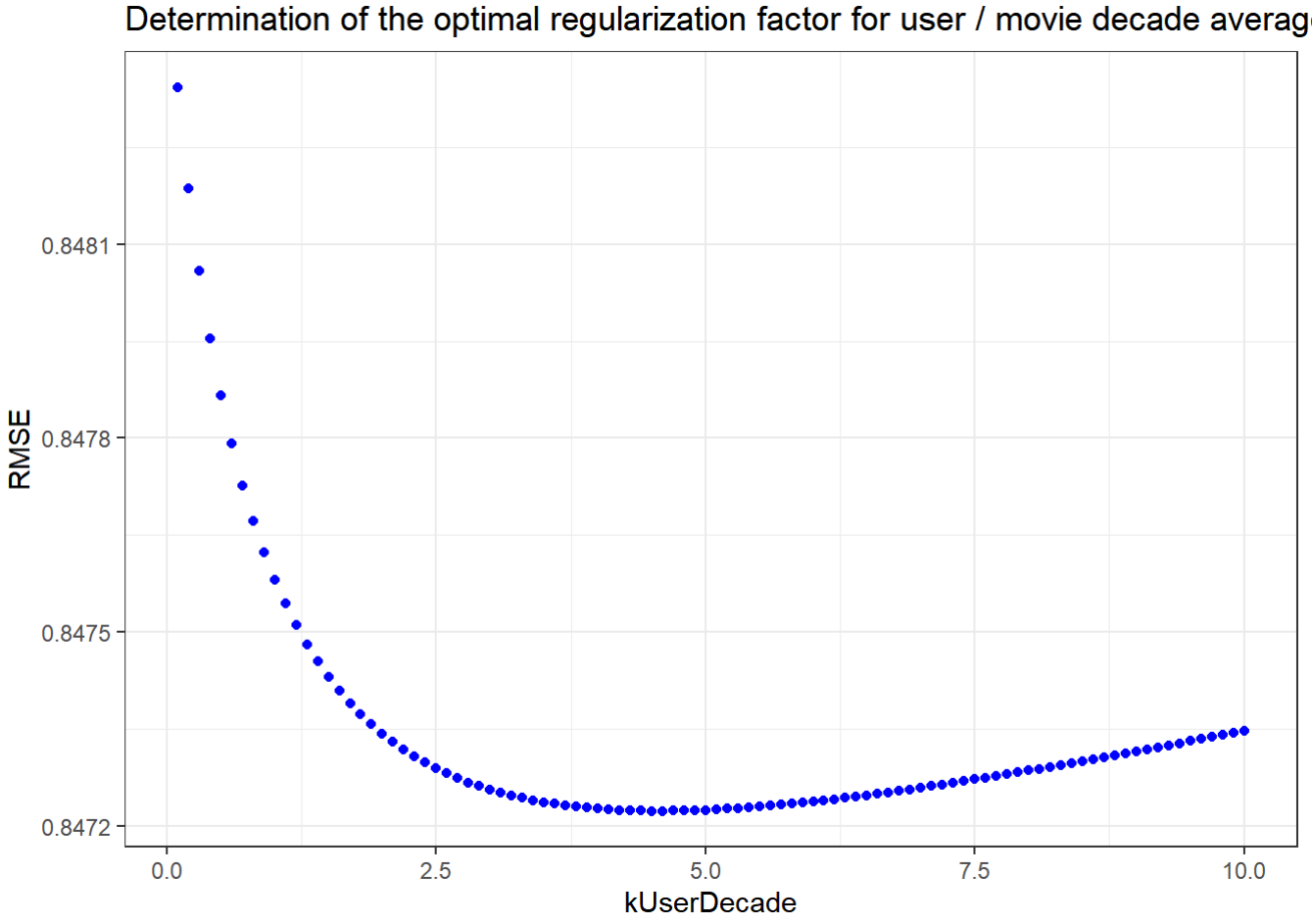

```
M <- FineTune %>% filter(fact == "kMovieYear") %>%
                  summarize(min(RMSE)) %>% filter(row_number()==1) %>% pull()

kMovieYearOpt <- FineTune %>% filter(fact == "kMovieYear" & RMSE == M) %>% pull(k)

SummaryOptimalk <- rbind(SummaryOptimalk, data.frame(Factor = "kMovieYear", Optimalk = kMovieYearOpt, RMSE = M))

SummaryOptimalk
```

Factor <fctr>	**Optimalk** <dbl>	**RMSE** <dbl>
kMovie	2.0	0.94320
kUser	5.0	0.86491
kGenresUser	7.8	0.85611
kUserYear	2.8	0.85396
kMovieYear	26.5	0.85136

5 rows

And term 7:

```
# Determination of the optimal regularization for couple user / movie decade average rating.

for(kUserDecade in seq(0.1, 10, 0.1)) {

  TestSet <- TestSet %>%
              mutate(Y_hat = AvgOverall +
                     (nMovie*DiffRatingMovie)/(nMovie + kMovieOpt) +
                     (nUser*DiffRatingUser)/(nUser + kUserOpt) +
                     (nGenresUser*DiffGenresUser)/(nGenresUser + kGenresUserOpt) +
                     (nUserYear*DiffUserYear)/(nUserYear + kUserYearOpt) +
                     (nMovieYear*DiffMovieYear)/(nMovieYear + kMovieYearOpt) +
                     (nUserDecade*DiffUserDecade)/(nUserDecade + kUserDecade))

  RMSE <- sqrt(mean((TestSet$rating - TestSet$Y_hat)^2))

  FineTune <- rbind(FineTune, data.frame(fact = "kUserDecade", k = kUserDecade, RMSE = RMSE))

}

FineTune %>% filter(fact == "kUserDecade") %>%
             ggplot(aes(k, RMSE)) +
             geom_point(color = "blue") +
             theme_bw() +
             labs(x = "kUserDecade", y = "RMSE", title = "Determination of the optimal regularization factor for user / movi
e decade average rating")
```




```
M <- FineTune %>% filter(fact == "kUserDecade") %>%
                  summarize(min(RMSE)) %>% filter(row_number()==1) %>% pull()

kUserDecadeOpt <- FineTune %>% filter(fact == "kUserDecade" & RMSE == M) %>% pull(k)

SummaryOptimalk <- rbind(SummaryOptimalk, data.frame(Factor = "kUserMovieDecade", Optimalk = kUserDecadeOpt, RMSE = M))

SummaryOptimalk
```

Factor <fctr>	Optimalk <dbl>	RMSE <dbl>
kMovie	2.0	0.94320
kUser	5.0	0.86491
kGenresUser	7.8	0.85611
kUserYear	2.8	0.85396
kMovieYear	26.5	0.85136
kUserMovieDecade	4.6	0.84722

6 rows

We now have the 6 optimal regularization factors, we can compute the final RMSE on the Test Set:

```
# Computation of the final RMSE on the Test Set using all optimal regularization factors.

TestSet <- TestSet %>%
            mutate(Y_hat = AvgOverall +
                   (nMovie*DiffRatingMovie)/(nMovie + kMovieOpt) +
                   (nUser*DiffRatingUser)/(nUser + kUserOpt) +
                   (nGenresUser*DiffGenresUser)/(nGenresUser + kGenresUserOpt) +
                   (nMovieYear*DiffMovieYear)/(nMovieYear + kMovieYearOpt) +
                   (nUserYear*DiffUserYear)/(nUserYear + kUserYearOpt) +
                   (nUserDecade*DiffUserDecade)/(nUserDecade + kUserDecadeOpt))

sqrt(mean((TestSet$Y_hat - TestSet$rating)^2))
```

```
## [1] 0.84722
```

```
# A bit of cleanup.

rm(AvgOverall, kMovie, kUser, kMovieYear, kUserYear, kGenresUser, kUserDecade, M, RMSE)
rm(Gavg, Mavg, MYavg, Uavg, UYavg, DUavg, TrainingSet, TestSet)
```

Step 6: Training of the algorithm with the full edx set

Now, we can retrain the model using the full edx data set. The same operations as before are performed for the 7 terms:

```
# First, a bit of cleanup.

rm(AvgOverall, kMovie, kUser, kMovieYear, kUserYear, kGenresUser, kUserDecade, M, RMSE)
rm(Gavg, Mavg, MYavg, Uavg, UYavg, DUavg, TrainingSet, TrainingSet, TestSet, TestSet)


# Add to the Training Set two new columns: the rating year and the movie release date.

  EedxSet <- ExtractMovieYearAndRatingYear(edx)


# Compute for the Training Set the average rating for all movies and all users.

  AvgOverall <- GetTheBasis(EedxSet)

  EedxSet <- EedxSet %>%
             mutate(Y_hat = AvgOverall)


# Compute for the Training Set the average rating of each movie for all users.

  Mavg <- GetTheMovieAverageRating(EedxSet)

  EedxSet <- EedxSet %>%
             left_join(Mavg, by = "movieId") %>%
             mutate(DiffRatingMovie = ifelse(!is.na(DiffRatingMovie), DiffRatingMovie, 0), nMovie = ifelse(!is.na(nMovie), n
Movie, 0))

  EedxSet <- EedxSet %>%
             mutate(Y_hat = AvgOverall + DiffRatingMovie)


# Compute for the Training Set the average rating of each user for all movies.

  Uavg <- GetTheUserAverageRating(EedxSet)

  EedxSet <- EedxSet %>%
             left_join(Uavg, by = "userId") %>%
             mutate(DiffRatingUser = ifelse(!is.na(DiffRatingUser), DiffRatingUser, 0), nUser = ifelse(!is.na(nUser), nUser,
0))

  EedxSet <- EedxSet %>%
             mutate(Y_hat = AvgOverall + DiffRatingMovie + DiffRatingUser)


# Compute for the Training Set for each movie (across all users) the average rating per year.

  Gavg <- GetTheGenreUserAverageRating(EedxSet)

  EedxSet <- EedxSet %>%
             left_join(Gavg, by = c("userId", "genres")) %>%
             mutate(DiffGenresUser = ifelse(!is.na(DiffGenresUser), DiffGenresUser, 0), nGenresUser = ifelse(!is.na(nGenresU
ser), nGenresUser, 0)) %>%
             mutate(Y_hat = AvgOverall + DiffRatingMovie + DiffRatingUser + DiffGenresUser)

  EedxSet <- EedxSet %>%
             mutate(Y_hat = AvgOverall + DiffRatingMovie + DiffRatingUser + DiffGenresUser)


# Compute for the Training Set for each user (and all movies) the average rating per year.

  UYavg <- GetTheUserYearAverageRating(EedxSet)

  EedxSet <- EedxSet %>%
             left_join(UYavg, by = c("userId", "ratingYear")) %>%
             mutate(DiffUserYear = ifelse(!is.na(DiffUserYear), DiffUserYear, 0), nUserYear = ifelse(!is.na(nUserYear), nUse
rYear, 0))

  EedxSet <- EedxSet %>%
             mutate(Y_hat = AvgOverall + DiffRatingMovie + DiffRatingUser + DiffGenresUser + DiffUserYear)


# Compute for the Training Set for each movie (across all users) the average rating per year.

  MYavg <- GetTheMovieYearAverageRating(EedxSet)

  EedxSet <- EedxSet %>%
             left_join(MYavg, by = c("movieId", "ratingYear")) %>%
             mutate(DiffMovieYear = ifelse(!is.na(DiffMovieYear), DiffMovieYear, 0), nMovieYear = ifelse(!is.na(nMovieYear),
nMovieYear, 0))

  EedxSet <- EedxSet %>%
             mutate(Y_hat = AvgOverall + DiffRatingMovie + DiffRatingUser + DiffGenresUser + DiffUserYear + DiffMovieYear)
```

```
# Compute for the Training Set for each user the average rating given for movies of a particular decade.

  DUavg <- GetTheUserDecadeAverageRating(EedxSet)

  EedxSet <- EedxSet %>%
             left_join(DUavg, by = c("userId", "movieDecade")) %>%
             mutate(DiffUserDecade = ifelse(!is.na(DiffUserDecade), DiffUserDecade, 0), nUserDecade = ifelse(!is.na(nUserDec
ade), nUserDecade, 0))

  EedxSet <- EedxSet %>%
              mutate(Y_hat = AvgOverall + DiffRatingMovie + DiffRatingUser + DiffGenresUser + DiffUserYear + DiffMovieYear +
 DiffUserDecade)
```

We now have a complete set that we can use on the final verification set.

Step 7: Application of the algorithm from step 6 and regularization factors of step 5 on the validation set provided by edx.

```
# Apply the algorithm on the validation set.

  validation <- ExtractMovieYearAndRatingYear(validation)

  validation <- validation %>%
                left_join(Mavg, by = "movieId") %>%
                  mutate(DiffRatingMovie = ifelse(!is.na(DiffRatingMovie), DiffRatingMovie, 0), nMovie = ifelse(!is.na(nMovi
e), nMovie, 0))

  validation <- validation %>%
                left_join(Uavg, by = "userId") %>%
                mutate(DiffRatingUser = ifelse(!is.na(DiffRatingUser), DiffRatingUser, 0), nUser = ifelse(!is.na(nUser), nUs
er, 0))

  validation <- validation %>%
                left_join(Gavg, by = c("userId", "genres")) %>%
                 mutate(DiffGenresUser = ifelse(!is.na(DiffGenresUser), DiffGenresUser, 0), nGenresUser = ifelse(!is.na(nGenr
esUser), nGenresUser, 0))

  validation <- validation %>%
                left_join(UYavg, by = c("userId", "ratingYear")) %>%
                 mutate(DiffUserYear = ifelse(!is.na(DiffUserYear), DiffUserYear, 0), nUserYear = ifelse(!is.na(nUserYear), n
UserYear, 0))

  validation <- validation %>%
                left_join(MYavg, by = c("movieId", "ratingYear")) %>%
                mutate(DiffMovieYear = ifelse(!is.na(DiffMovieYear), DiffMovieYear, 0), nMovieYear = ifelse(!is.na(nMovieYea
r), nMovieYear, 0))

  validation <- validation %>%
                left_join(DUavg, by = c("userId", "movieDecade")) %>%
                 mutate(DiffUserDecade = ifelse(!is.na(DiffUserDecade), DiffUserDecade, 0), nUserDecade = ifelse(!is.na(nUser
Decade), nUserDecade, 0))

  validation <- validation %>%
                mutate(Y_hat = AvgOverall +
                (nMovie*DiffRatingMovie)/(nMovie + kMovieOpt) +
                (nUser*DiffRatingUser)/(nUser + kUserOpt) +
                (nGenresUser*DiffGenresUser)/(nGenresUser + kGenresUserOpt) +
                (nMovieYear*DiffMovieYear)/(nMovieYear + kMovieYearOpt) +
                (nUserYear*DiffUserYear)/(nUserYear + kUserYearOpt) +
                (nUserDecade*DiffUserDecade)/(nUserDecade + kUserDecadeOpt))
```

To be sure, let's check if all entries are there and have an estimated rating:

```
# Apply the algorithm on the validation set.

sum(!is.na(validation$rating))
```

```
## [1] 999999
```

```
sum(!is.na(validation$Y_hat))
```

```
## [1] 999999
```

All entries are there and have a rating, so the algorithm is reliable and provide an estimated rating for each entry.

Now, the moment of truth, the RMSE:

```
  sqrt(mean((validation$Y_hat - validation$rating)^2))
```

```
## [1] 0.84643
```

Mission complete!

Step 8: Conclusion and perspectives

The RMSE we obtain here is somwhat acceptable (still far from the matrix factorization I agree) but I am working now ideas of improvements:

- The genres are currently not enough valuable. Each genre combination could be divided by individual genres (e.g. divide Action|Drama in Action + Drama) and design an algorithm that would compute a rating per user and per individual genre. This would increase the number of movies per couple user / genre and likely make the genre more valuable;
- Other terms could be added based upon clusters of users or clusters of movies. Clusters would group users and movies with similar ratings and the same method applied before could be applied on those to get new algorithm terms;
- Other algorithm terms could be developped based upon correlations between users and movies, this would lead closer to Matrix facorization methods.

Thank you for reading!