MovieLens Project

Data Science: Capstone

Rémi Fauve

2020-05-15

Contents

1 Introduction

This report was generated as part of the HarvardX online course "Data Science: Capstone" (PH125.9x).

The MovieLens project's goal is to predict movie ratings using data science techniques. The original dataset is the MovieLens 10M Dataset which includes 10 millions ratings on 10,000 movies, by 72,000 different users.

First, the dataset is imported and explored to gain insights. Then, different models are tested as recommandation systems, and the best one is selected. Finally, the selected model is applied on the final test set.

The following libraries are required to execute the code of this report:

```
if(!require(tidyverse))
  install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if (!require(caret))
  install.packages("caret", repos = "http://cran.us.r-project.org")
if (!require(data.table))
  install.packages("data.table", repos = "http://cran.us.r-project.org")
if (!require(ggrepel))
  install.packages("ggrepel", repos = "http://cran.us.r-project.org")
if (!require(FactoMineR))
  install.packages("FactoMineR", repos = "http://cran.us.r-project.org")
if (!require(factoextra))
  install.packages("factoextra", repos = "http://cran.us.r-project.org")
if (!require(recommenderlab))
  install.packages("recommenderlab", repos = "http://cran.us.r-project.org")
if (!require(explor))
  install.packages("explor", repos = "http://cran.us.r-project.org")
#(.packages())
```

2 Data importation

The dataset is downloaded from the GroupLens Research website.

```
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip",
              dl)
```

The data is extracted from the different files included in the zip archive into two dataframes:

- `ratings`, which contains the ratings given by different people (`userId`) for different movies (`movieId`) along with their date (`timestamp` converted to `rating_year`).

```
ratings <-
  fread(text = gsub("::",
                    "\t",
                    readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
        col.names = c("userId", "movieId", "rating", "timestamp"))

ratings <-
  as.data.frame(ratings) %>%
  mutate(rating_year = year(as.Date.POSIXct(timestamp))) %>%
  select(-timestamp)
```

- `movies`, which contains some information about the rated movies, such as their title and their genre.

```
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(movieId),
    title = as.character(title),
    genres = as.character(genres))
```

The `movies` dataframe can be enhanced by extracting the release date that is nested in the `title`.

```
# Check that every title contains a release date (with the same format)
sum(movies$title %>% str_detect("\\(\\d{4}\\)")) == nrow(movies)
```

```
## [1] TRUE
```

```
movies <- movies %>%
  mutate(release_year = as.numeric(
           str_remove_all(
             str_extract(title,"\\((\\d{4})\\)"),
             "\\(|\\)")),
         title = str_remove(title," \\((\\d{4})\\)"))
```

The `movies` dataframe can also be enhanced by refining the genre description with a boolean variable per unique genre, for each movie. Also, the genre for the movie "Pull My Daisy" was missing, and has been manually rectified ("Drama").

```
# Manual correction ("Drama" assigned to "Pull My Daisy")
movies$title[movies$genres == "(no genres listed)"]
```

```
## [1] "Pull My Daisy"
```

```
movies$genres[movies$genres == "(no genres listed)"] <- "Drama"

# Split the genre description and identify the unique genres
movies <- movies %>%
  mutate(list_genres = strsplit(genres,"\\|"))

list_unique_genre <- sort(unique(unlist(movies$list_genres)))
list_unique_genre
```

```
##  [1] "Action"      "Adventure"   "Animation"   "Children"    "Comedy"
##  [6] "Crime"       "Documentary" "Drama"       "Fantasy"     "Film-Noir"
## [11] "Horror"      "IMAX"        "Musical"     "Mystery"     "Romance"
## [16] "Sci-Fi"      "Thriller"    "War"         "Western"
```

```
# Evaluate each boolean variable for each movie
detailled_genre <-
  sapply(list_unique_genre, function(x) {
    sapply(movies$list_genres, function(y) {
      x %in% y
    })
  })

detailled_genre <- data.frame(detailled_genre)

for (i in 1:19) {
  detailled_genre[, i] <-
    factor(
      detailled_genre[, i],
      levels = c("FALSE", "TRUE"),
      labels = c(paste0(colnames(detailled_genre)[i], "_n"),
                 paste0(colnames(detailled_genre)[i], "_y")))
}

movies <- cbind(movies,detailled_genre) %>%
  select(-genres,-list_genres)

remove(list_unique_genre, detailled_genre, i)
```

Then the two dataframes can be joined.

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

The data is separated into a train set `edx` and a test set `validation`, with the test set gathering 10% of the total data set.

```
# Validation set will be 10% of MovieLens data
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, ]
```

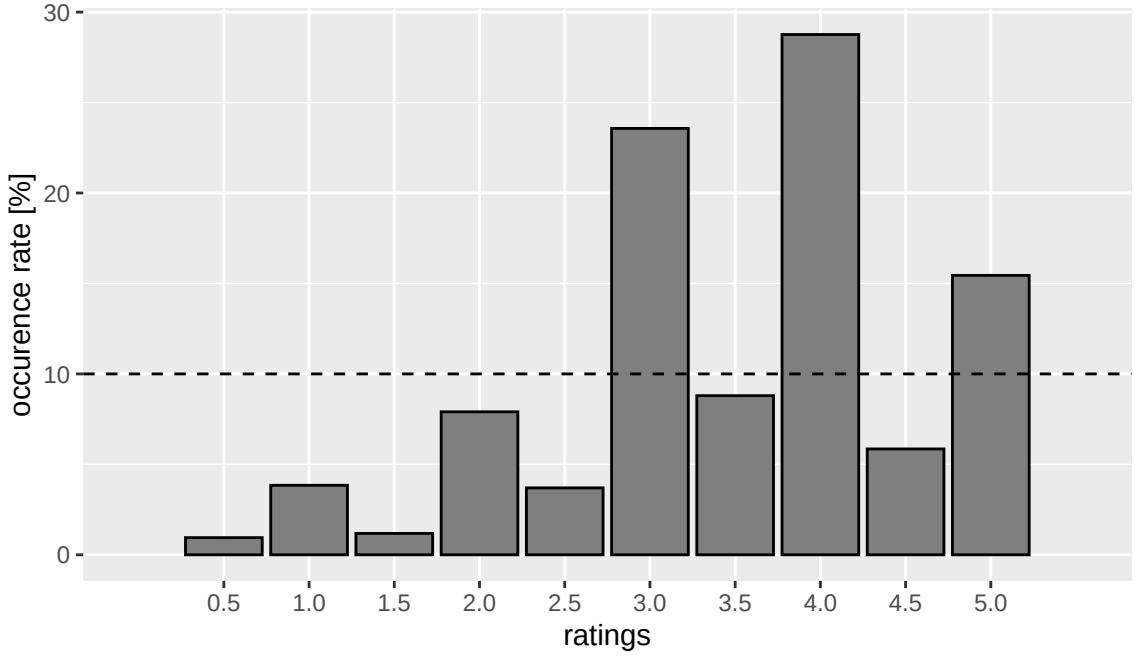
```r
# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# 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)
```

3 Data exploration

From now on, our insights have to be taken only from the `edx` data set. From those, the goal is to build a model trained on the `edx` data that can predict how a given user would rate a movie. Those predictions will be based on the different variables composing the `edx` data set.

3.1 Ratings

First, the ratings' distribution is investigated, by plotting the occurence rate of each possible rating given by a user. The horizontal dashed line represents the expected average occurence rate if they were uniform (10% each).



The occurence rates are clearly not uniform. Two additional observations can be made here:

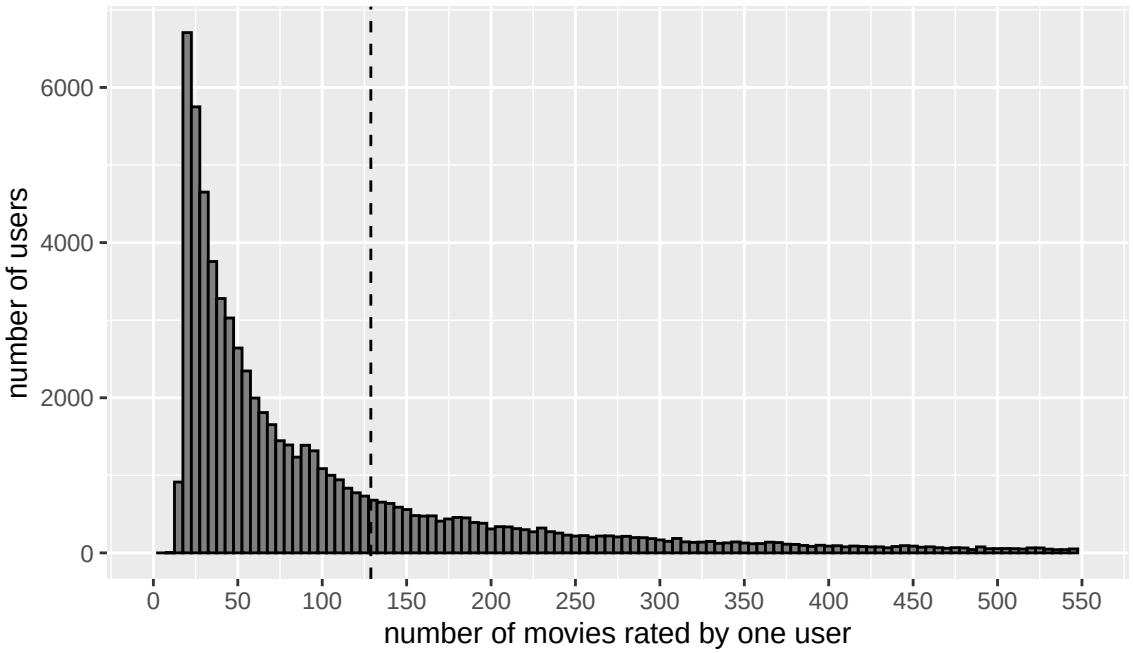
- users tend to give on average higher ratings,
- and users tend to give on average more whole ratings than halved ratings.

The non-uniform distribution of the ratings should be taken into account in our models.

3.2 Users

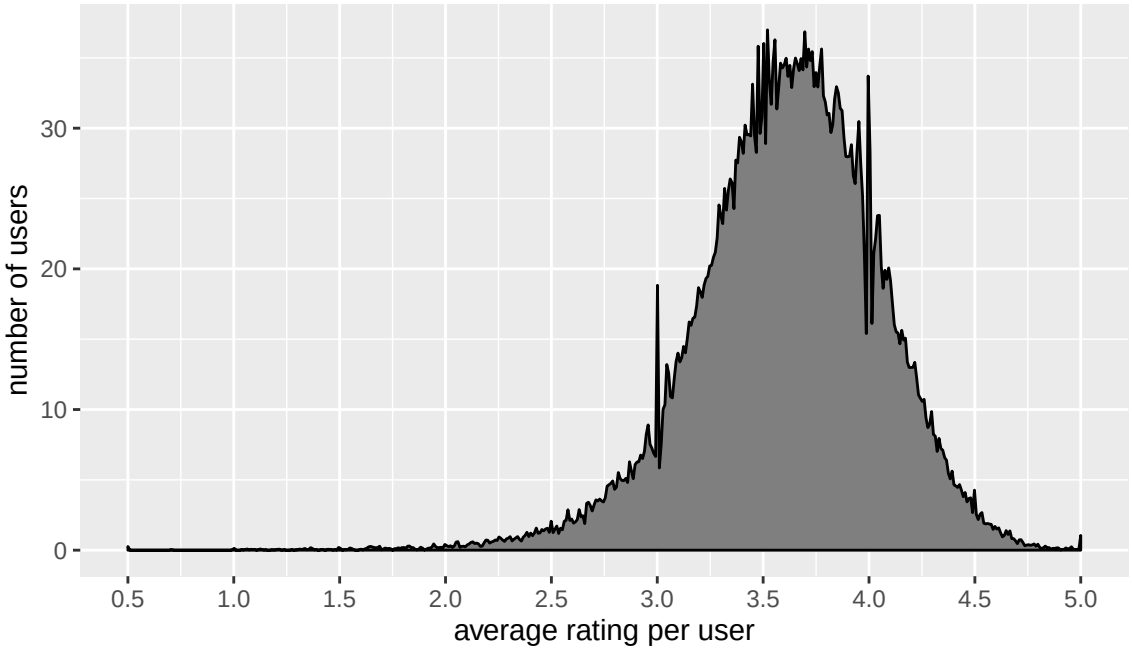
Since the goal of the project is to predict the rating given by a specific user among those listed in the data set, it is essential to investigate the users' distribution and its relationship with the ratings.

First, users don't give the same number of movies rated, as shown in the following plot. The vertical dashed line represents the expected average number of movie rated by a user if each rated the same number of movies (around 129 movies rated).

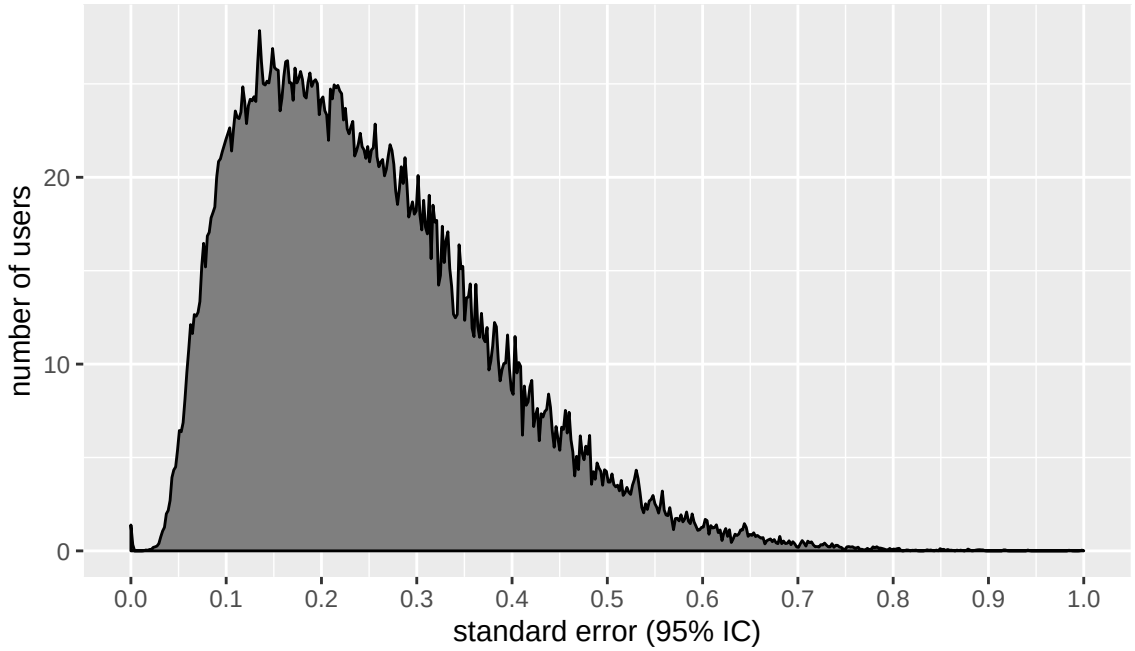


This figure shows that about 6500 users have rated around 20 movies each, and that some users rated more than 500 movies (up to 6616 movies rated by a single user !).

Then, the distribution of the average rating of each user is plotted.



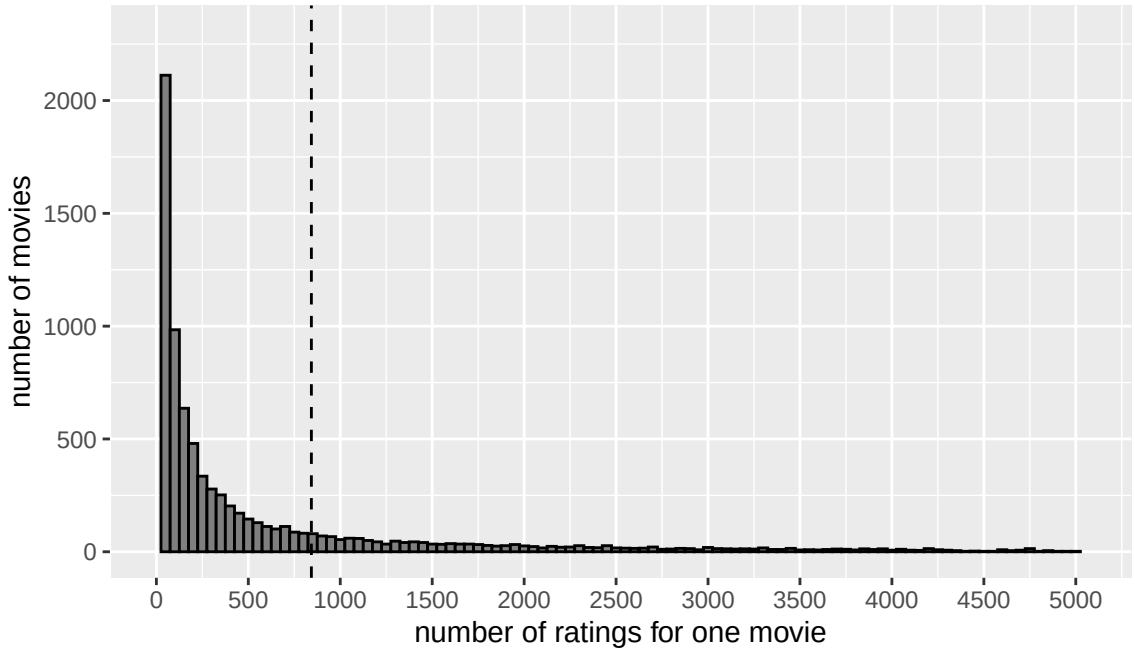
This distribution is close to the distribution of the ratings, which is expected. This figure was obtained with a very low binwidth, and despite the expected general bell shape, strong artefacts appears at ratings of 3.0 and 4.0. These can be explained by the integer, or half-integer, nature of the ratings, whose averages have higher chances to be integers to. Two weaker artefacts can be observed at ratings of 0.5 and 5.0: those out-of-trend peaks shows that some users rated every movies with the lowest (or the highest) rating. Another way to reveal such behaviour is to plot the standard error of ratings for each user.



The peak at a null standard error confirms that certain users rated their movies with a single rating. The standard errors being relatively low (around one tenth of the average rating per user) can be explained by the short range of possible ratings (ten values from 0.5 to 5) and by the high number of rating per user, being mainly over 20 movies rated.

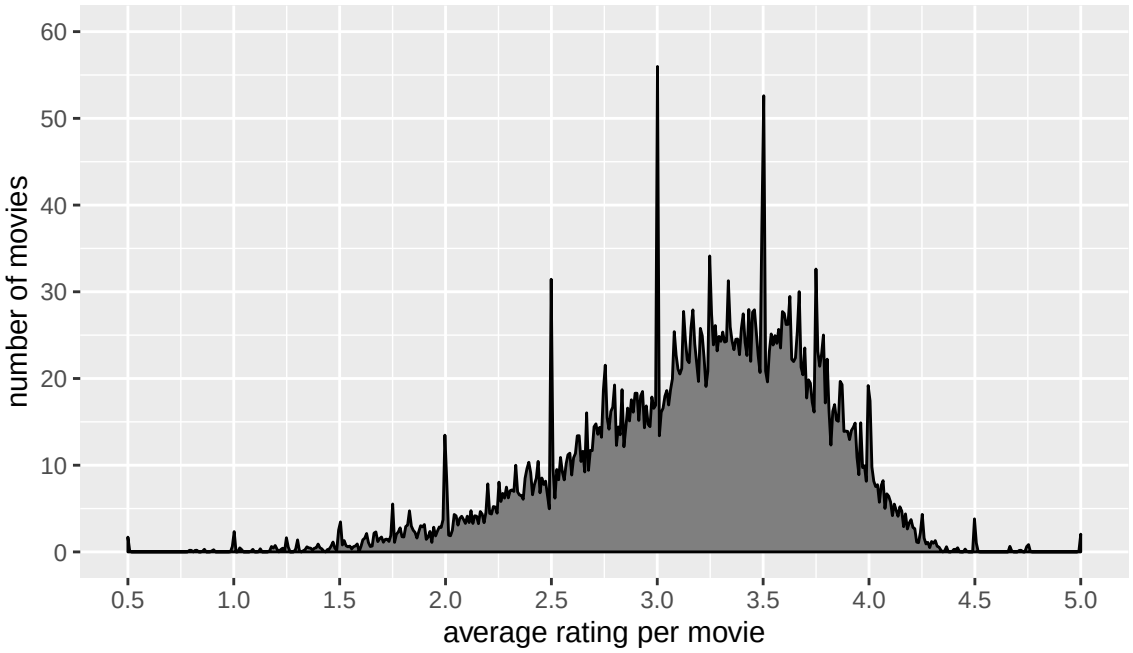
3.3 Movies

It is expected to have movies with higher numbers of ratings than others, or with higher average ratings than others, as shown in the following plot. The vertical dashed line represents the expected average number of ratings per movie if each movie received the same number of ratings (around 843 ratings per movie).



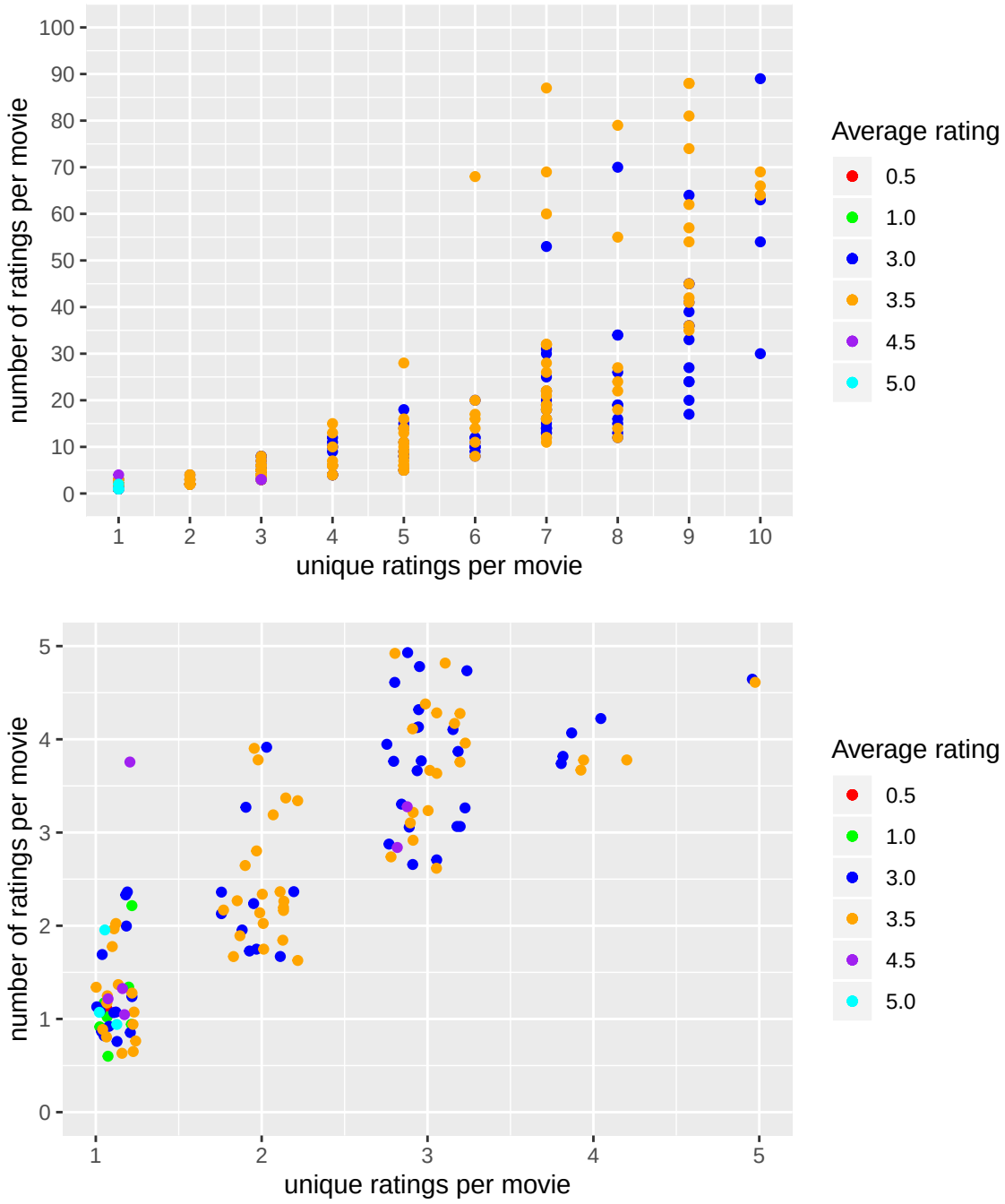
This figure shows that more than 2000 movies have been rated around 50 times, and that some movies have been rated more than 5000 times (up to 31362 ratings for a single movie: Pulp Fiction).

Then, the distribution of the average rating per movie is plotted.



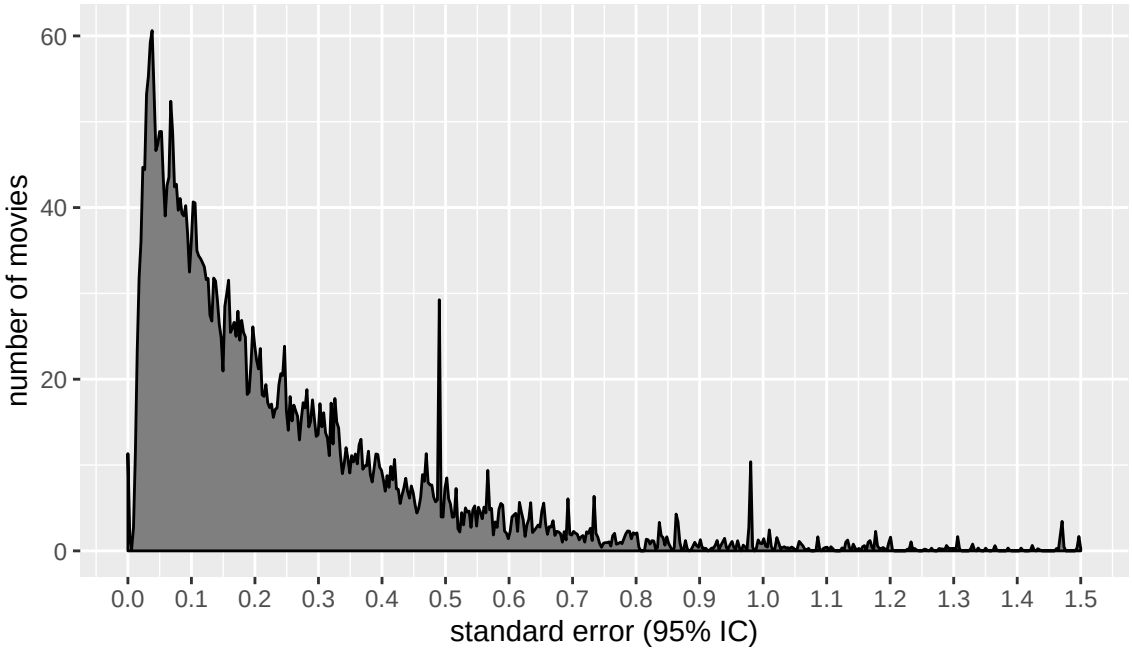
This distribution is similar to the distribution of the ratings, but it is stretched toward the medium rating. This reflects that movies are rarely unanimously acclaimed or rejected, thus their average ratings being pulled toward medium ratings in general. Also, peaks at whole or halved integer are more visible here than when considering average ratings per user, which can be explained by the number of ratings per movie being highly superior on average than the number of ratings per user.

This behavior can also be explained by plotting the number of ratings per movie against the number of different unique ratings per movie in the two following graphs.



As expected, movies can have an average rating of 3.0 or 3.5 even with a high number of different unique ratings, whereas movies with an average rating of 0.5 or 5.0 can only be resulting from a low number of ratings. To see these extreme cases on the plot, it is necessary to zoom closely and add jitter (second plot).

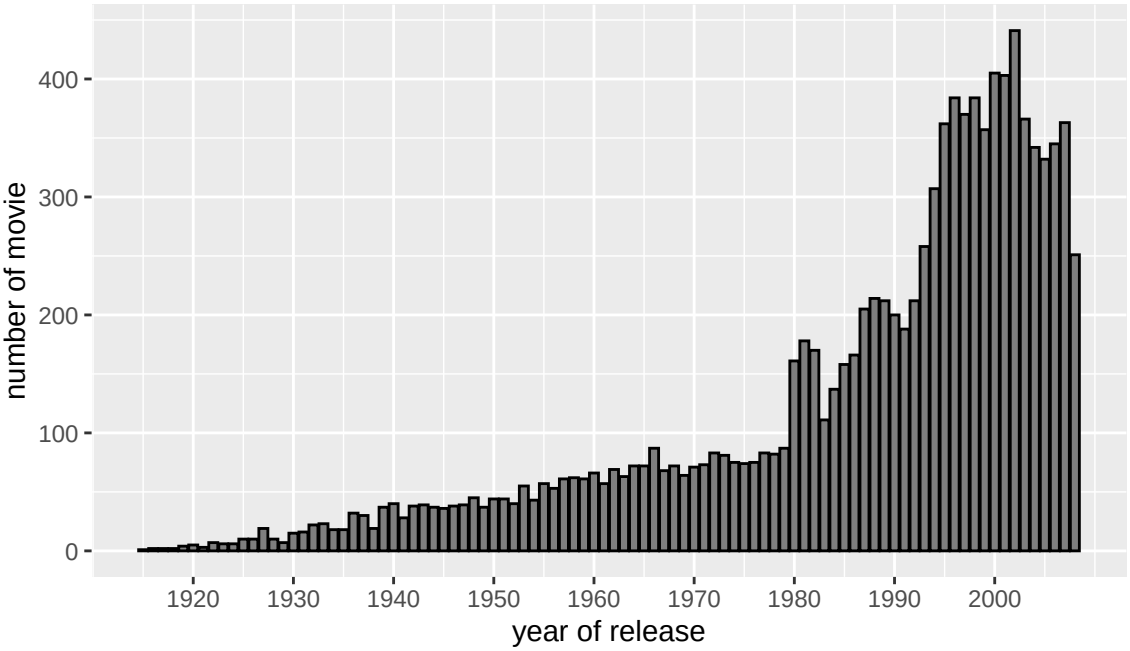
Other strange artefacts are observed when plotting the distribution of the standard error of ratings for each movie.



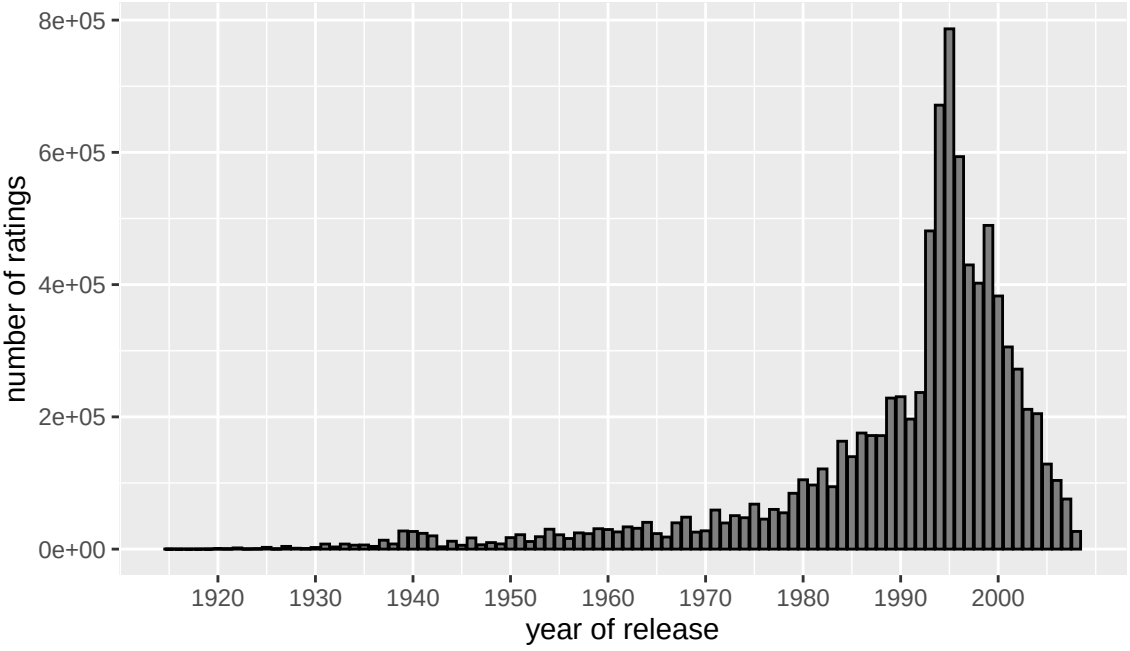
The peak at a null standard error confirms that certain movies are rated with a single rating by different users. The standard errors being relatively low (major peak around one tenth of the average rating per user) can be explained by the short range of possible ratings (ten values from 0.5 to 5) and by the high number of rating per movie, being mainly over 50 ratings, up to more than 5000. However, the tail of this distribution extends well over a standard error of 0.5, which means that for certain movies, the variability of the ratings are quite strong (movies that are, at the same time, acclaimed or despised by many different users). Also, the peaks around a standard error of 0.5, 1.0 and 1.5 remain difficult to explain.

3.4 Release year

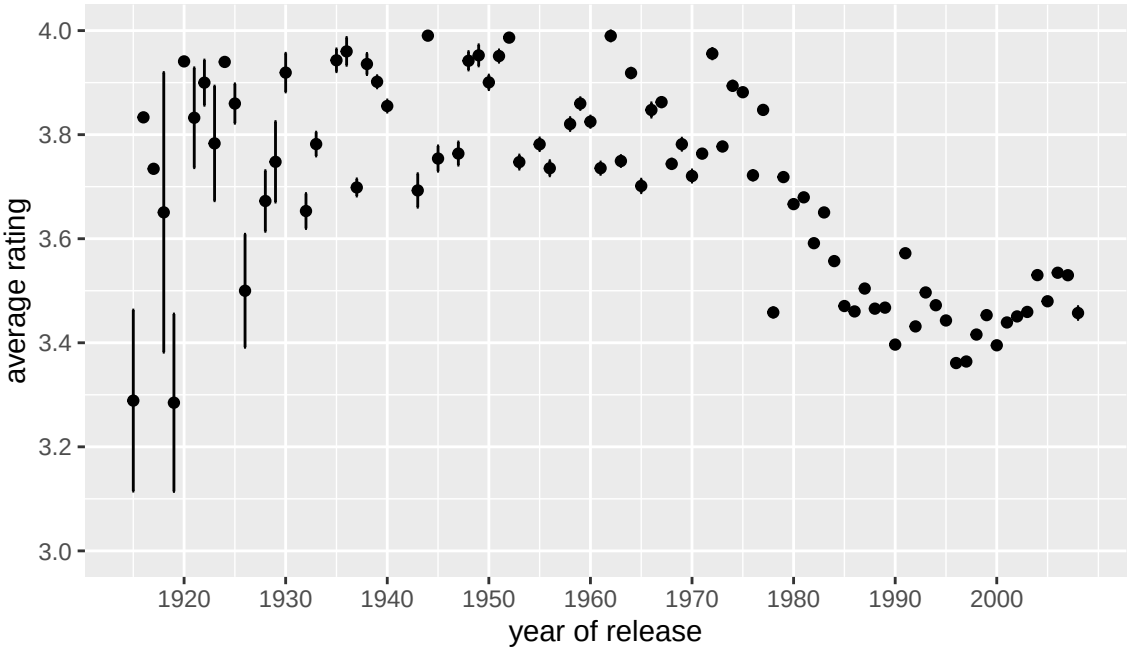
As movies are generally influenced by the trends and customs of their time, the release year of a movie may provide a useful information to predict if a given user will prefer movies from a given period. As shown in the following plot, the cinema production never ceased to increase (based on this MovieLens database), especially after the 1980s.



When looking at the ratings' distribution among the different release years, in the following plot, it is clear that the movies from the 1990s were the most watched/rated by the users. Maybe it is because this period gathers the childhood movies (like Disney's productions) for a large part of the users. Anyhow, there is a clear discrepancy between the number of ratings of old movies and recent movies.



This very non-uniform distribution of the number of ratings have an effect on the average rating per release year, with a decreasing standard error on the average rating as the year increases, since more people rated those movies.



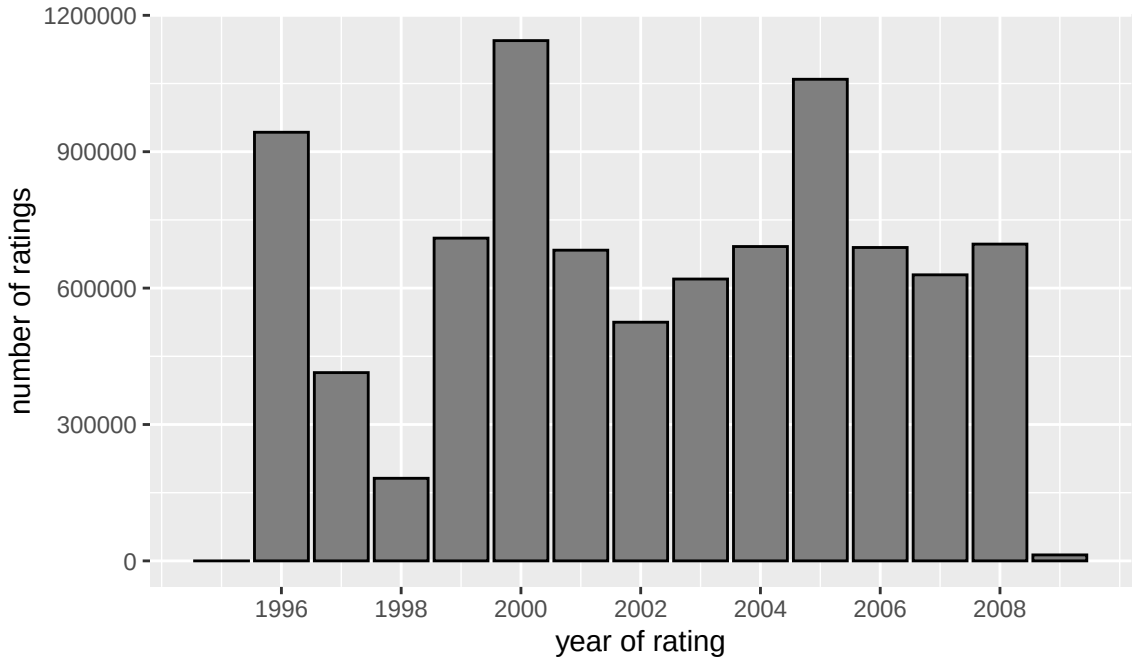
However, three different trends are noticeable :

- from 1915 to 1930, there aren't enough movies and ratings, so the average rating varies between 3.1 and 4.0,
- from 1930 to 1980, the standard error is lower, and the average rating varies between 3.6 and 4.0,
- and from 1980 to 2008, the standard error is virtually null, and the average rating varies between 3.3 and 3.6.

The total variation of the average rating per year is between 3.1 and 4.0, which is 20% of the total span of the possible ratings, which is significant enough to be considered later on.

3.5 Rating year

The time period for the users to post their ratings is very narrow (around 15 years, from 1995 to 2009) and it appears, in the following plot, that the number of ratings per year does not show a clear trend, so it will not be considered as useful information to predict the rating of users.



3.6 Genres combinations

As presented in the data importation section, each possible genre is now a boolean variable. This may allow a better estimation of the preferences of each user. A Multiple Correspondance Analysis (MCA) has been conducted to illustrate how some genre, or genre associations, can get higher average rating than others.

3.6.1 Multiple Correspondance Analysis setup

First, variables with "near zero variance" should be removed from the MCA since they can distort the analysis. The threshold chosen here means that a variable for which the two most frequent values have a ratio of 99 to 1 will be excluded.

```
nzv <- nearZeroVar(edx[, 6:24],
                   freqCut = 99/1,
                   saveMetrics = TRUE)

nzv[with(nzv,order(-freqRatio)), c(1,4)]
```

```
##              freqRatio   nzv
## IMAX        1099.11673  TRUE
## Documentary   95.70615 FALSE
## Film.Noir     74.92356 FALSE
## Western       46.52028 FALSE
## Musical       19.78151 FALSE
## Animation     18.26514 FALSE
## War           16.60757 FALSE
## Mystery       14.83591 FALSE
## Horror        12.01555 FALSE
## Children      11.19530 FALSE
## Fantasy        8.72309 FALSE
## Crime          5.77860 FALSE
## Sci.Fi         5.71053 FALSE
## Romance        4.25673 FALSE
```

```
## Adventure      3.71481 FALSE
## Thriller       2.86950 FALSE
## Action         2.51490 FALSE
## Comedy         1.54172 FALSE
## Drama          1.30173 FALSE
```

```
edx %>%
  filter(Documentary == "Documentary_y") %>%
  summarise(nb = n(),
            percent = nb/nrow(edx)*100)
```

```
##      nb percent
## 1 93066 1.03406
```

```
edx %>%
  filter(IMAX == "IMAX_y") %>%
  summarise(nb = n(),
            percent = nb/nrow(edx)*100)
```

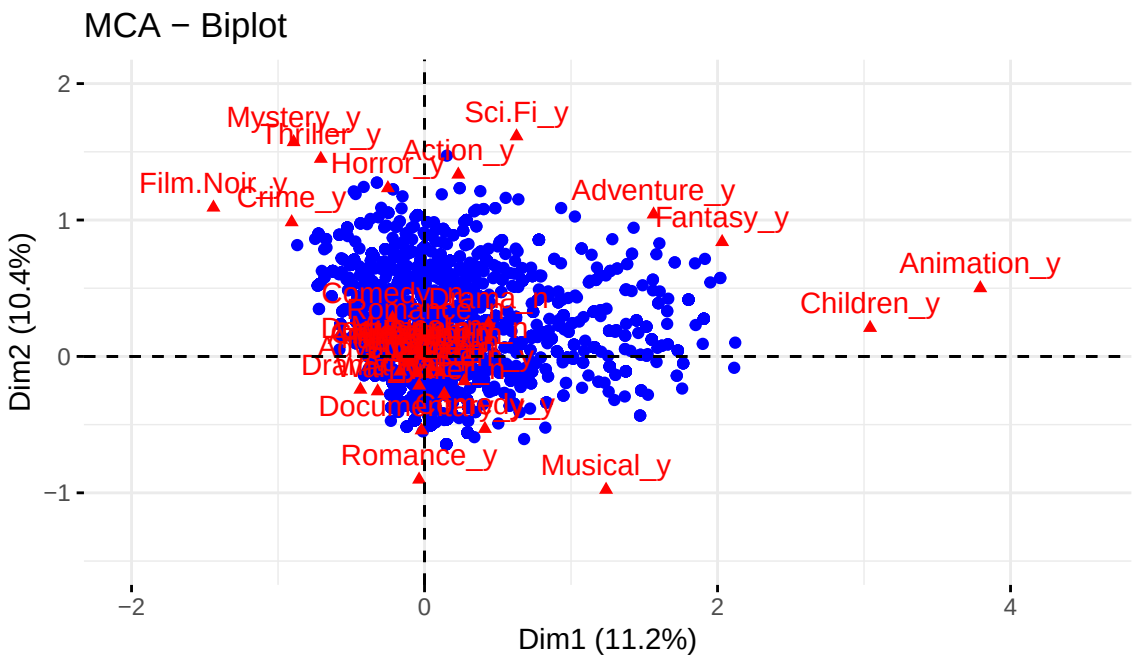
```
##     nb   percent
## 1 8181 0.0908994
```

The variable `IMAX` is removed, since more than 99.9% of the observations have the same value (`IMAX_n`).

The MCA will be performed on the genres of the movies, so observations will be movies, and since our target is the ratings, individual ratings are ignored here and each movie will have its average rating calculated.

```
edx_mca <- edx %>%
  mutate(release_year = as.numeric(release_year)) %>%
  group_by(movieId) %>%
  mutate(avg_rating = mean(rating)) %>%
  ungroup() %>%
  mutate(uniq_movie = duplicated(movieId)) %>%
  filter(uniq_movie == FALSE) %>%
  select(-userId, -movieId, -title, -rating, -uniq_movie, -release_year)

res.mca <- MCA(edx_mca[, c(-19)],
               ncp = 18,
               graph = FALSE)
```

As usual, the results are presented in a graph along the first and second principal dimensions resulting from the MCA. Here are presented, on the same graph (biplot), the coordinates of each individual (movie) and of each variables' values (genre).

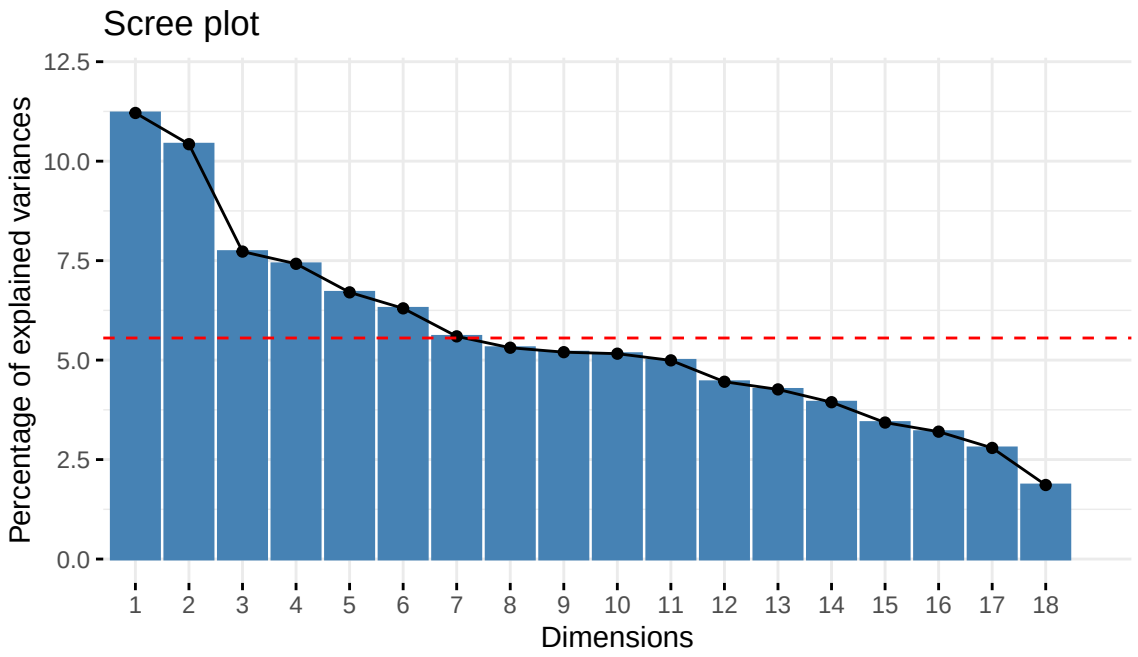


This plot shows that is essentially the "yes" values of the different genre that are responsible of the distinction between different kind of movies ("no" values are all gathered around zero). Also, on this biplot, the first bissector seems to separate two main general genres : on the upper left, more serious and dark movies, and on the lower right, more light-hearted movies.

Regarding genre associations, four different groups can be contructed from this biplot :

- movies for children (`Children_y`, `Animation_y`)
- comedies (`Comedy_y`, `Musical_y`, `Romance_y`, `Documentary_y`)
- action/adventure movies (`Action_y`, `Sci.Fi_y`, `Adventure_y`, `Fantasy_y`)
- thrillers (`Mystery_y`, `Thriller_y`, `Horror_y`, `Action_y`, `Sci.Fi_y`, `Film.Noir_y`, `Crime_y`)

This is an analysis based only the first two dimensions, out of the 18 that were calculated. A restricted number of relevant dimensions for a further analysis are selected using a screeplot. The red dashed line represents the percentage of explained variances if its distribution was uniform.



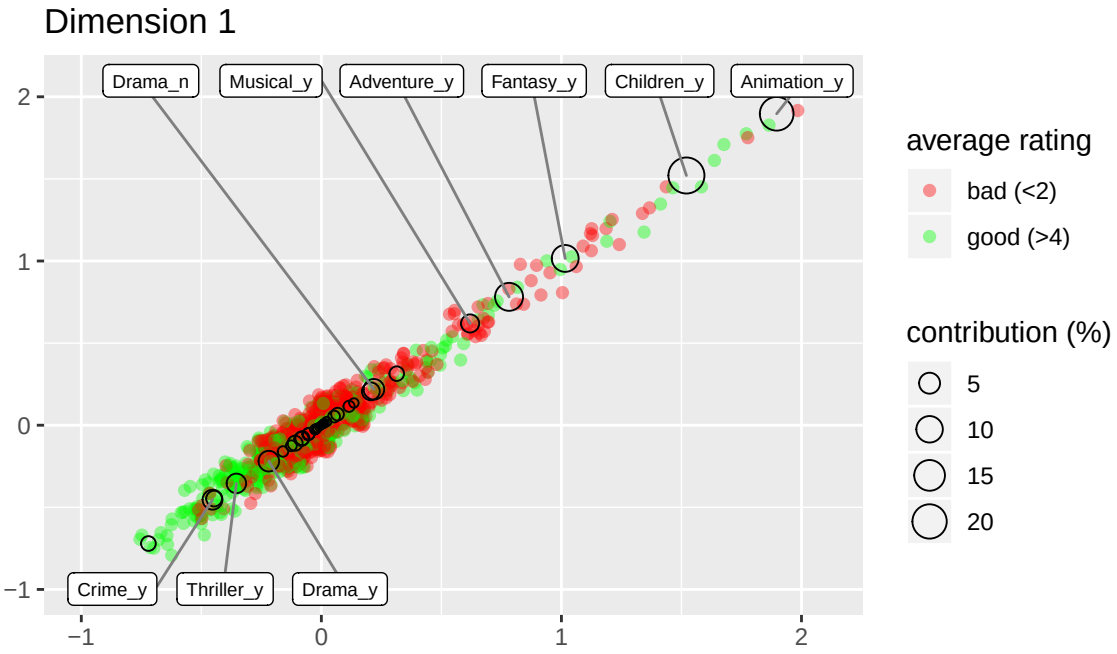
The screeplot shows that the first two dimensions explain more than 20% of the variance. The first seven dimensions will be considered for this analysis, which explain more than 55% of the variance. The rest of this analysis is an interpretation of the distribution of both movies and genre along each of the seven considered dimensions. For the sake of clarity only good (over 4.0) and bad (under 2.0) average rated movies will be displayed on the graphs.

3.6.2 Multidimensionnal analysis

The first seven dimensions produced by the MCA conducted on genres are presented here. Each graph plots individuals and variables values in one dimension, against itself. This allows a clear visualisation of the distribution of individuals and variables along a single dimension. The coordinates of the variables are divided by 2 for a better superposition with the individuals coordinates. This arbitrary homothety does not introduce any bias for further interpretation, since the order is preserved.

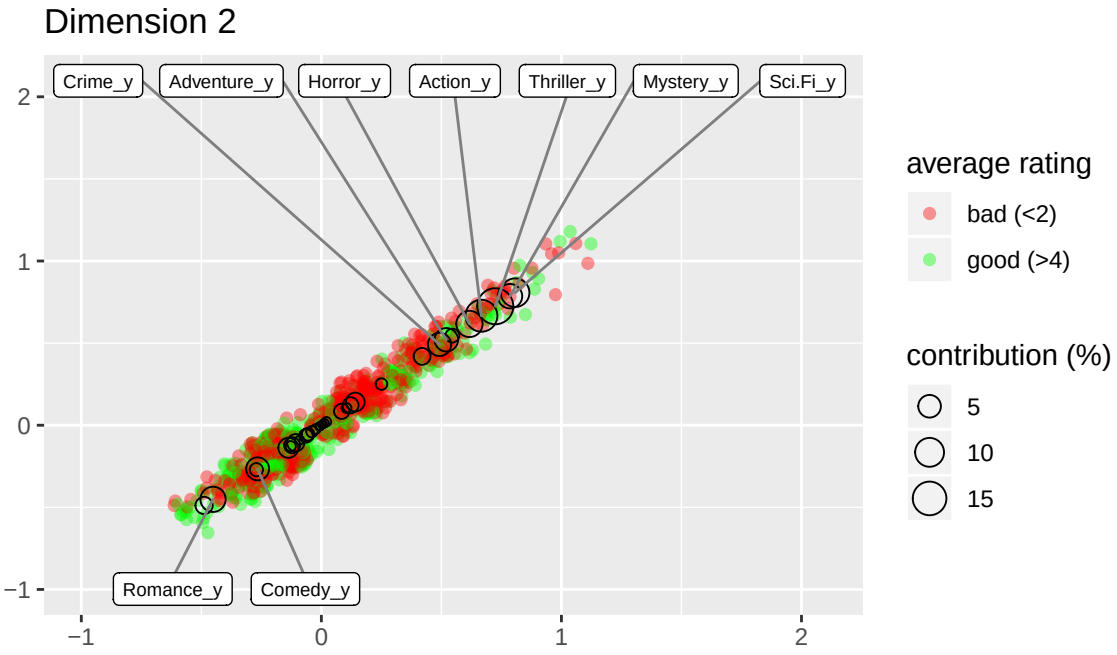
The individuals are either colored red (average rating lower than 2.0) or green (average rating higher than 4.0). The variables are represented by black circles, which size is related to its contribution to the dimension's variance. Only the variables with the 10 highest absolute contribution are labeled, for clarity's sake.

- Dimension 1



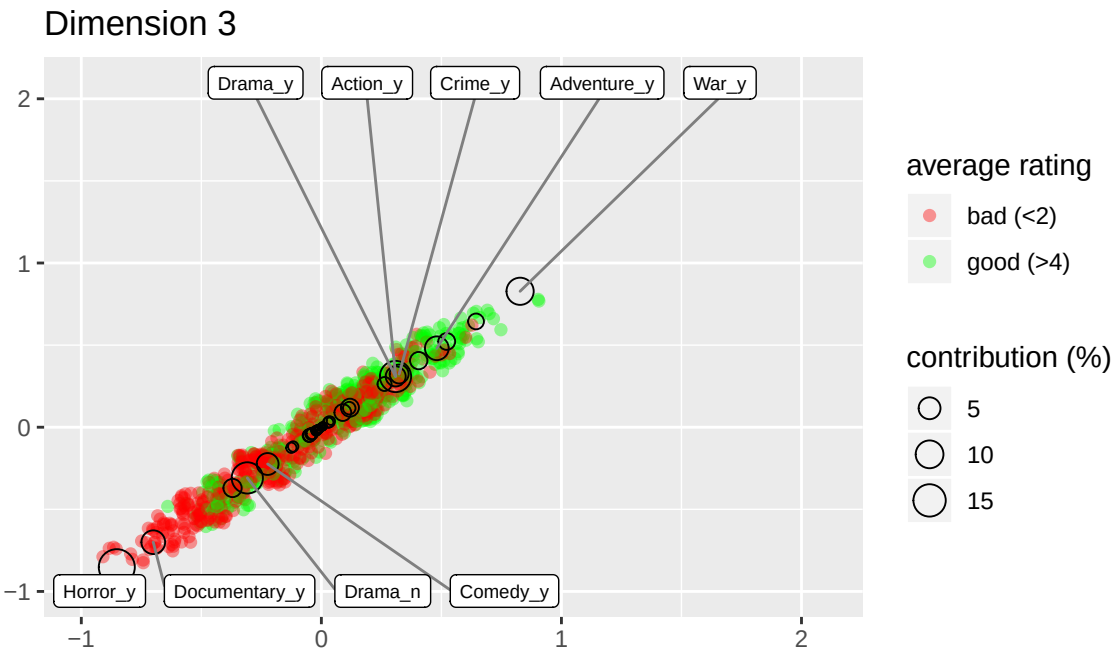
This dimension separates light-hearted movies (positive) from dark-themed movies (negative). In this dimension, a higher proportion of dark-themed movies have an average rating higher than 4.0, whereas light-hearted movies show an even mix of low and hight rated movies.

- Dimension 2



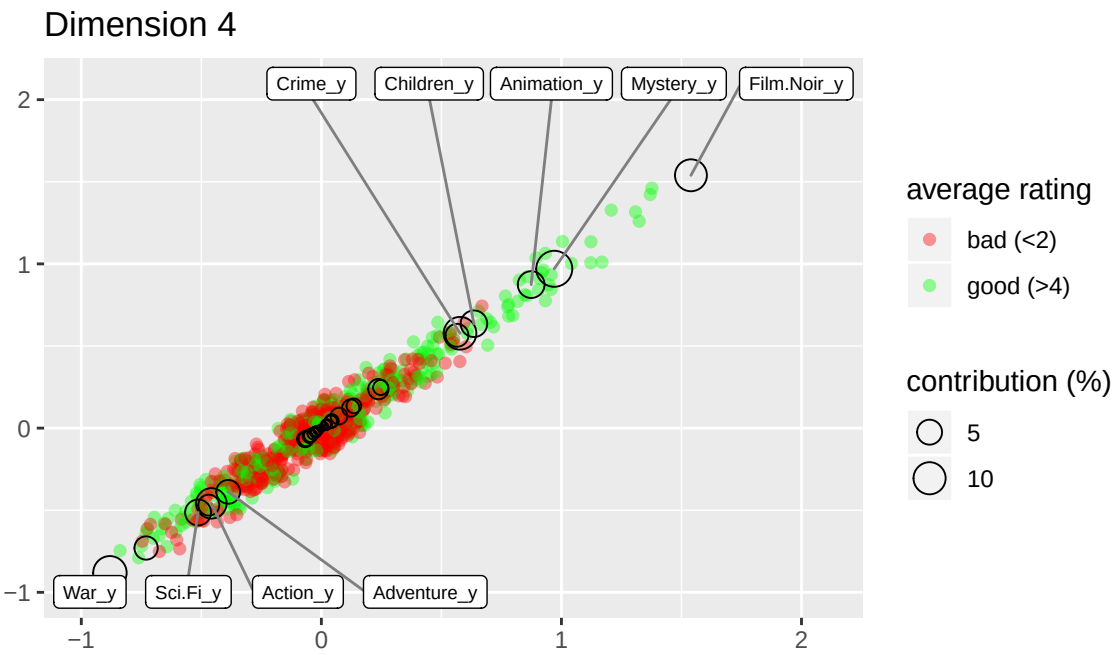
This dimension separates romantic movies (negative) from thriller or action-oriented movies (positive). No clear trend is noticeable in this dimension.

- Dimension 3



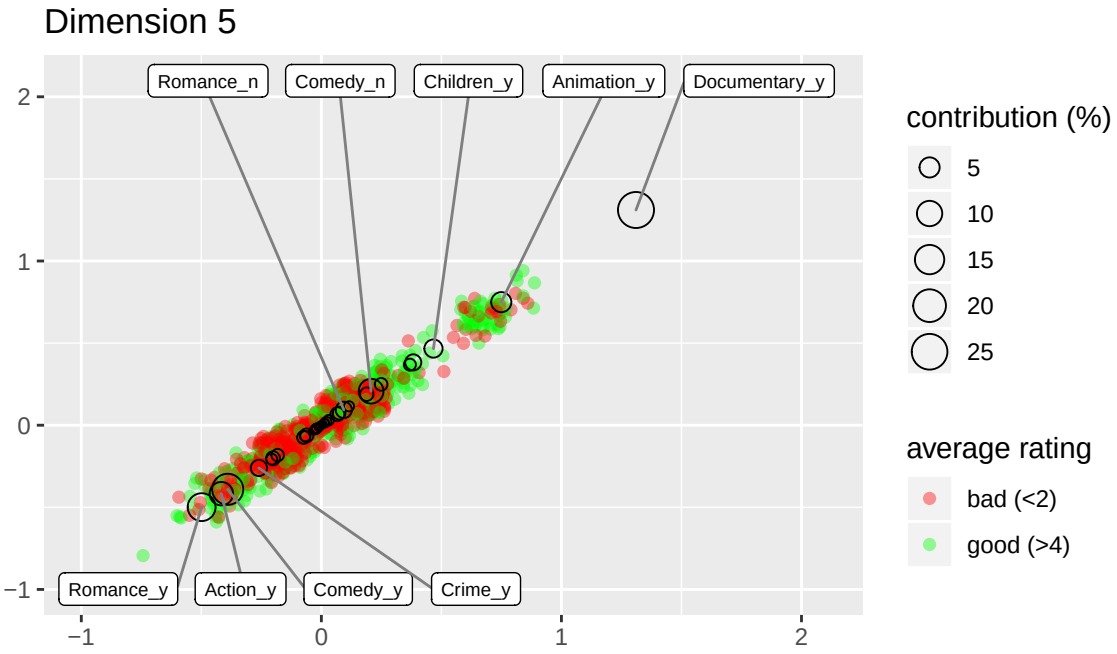
This dimension separates horror-fiction (negative) from reality based movies (positive), with the exception of `Documentary_y` being negative. In this dimension, a higher proportion of reality based movies have an average rating higher than 4.0, and a higher proportion of horror-fictional movies have an average rating lower than 2.0.

- Dimension 4



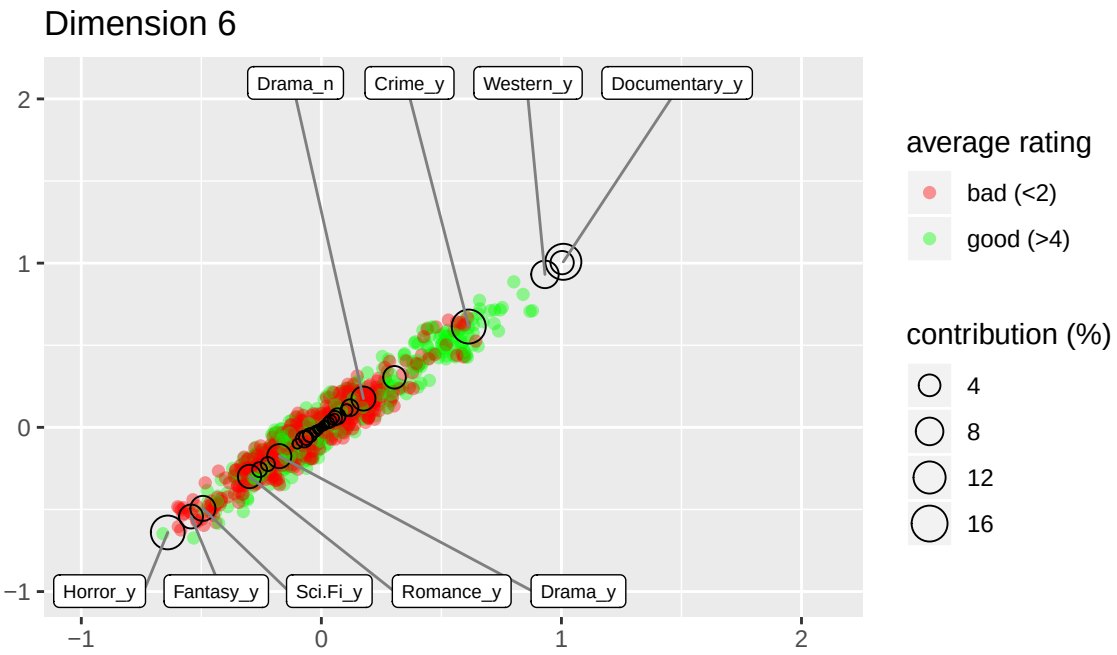
This dimension separates movies with investigations (positive) from action-oriented movies (negative). In this dimension, a higher proportion of movies with investigations have an average rating higher than 4.0, whereas action-oriented movies show an even mix of low and hight rated movies.

- Dimension 5



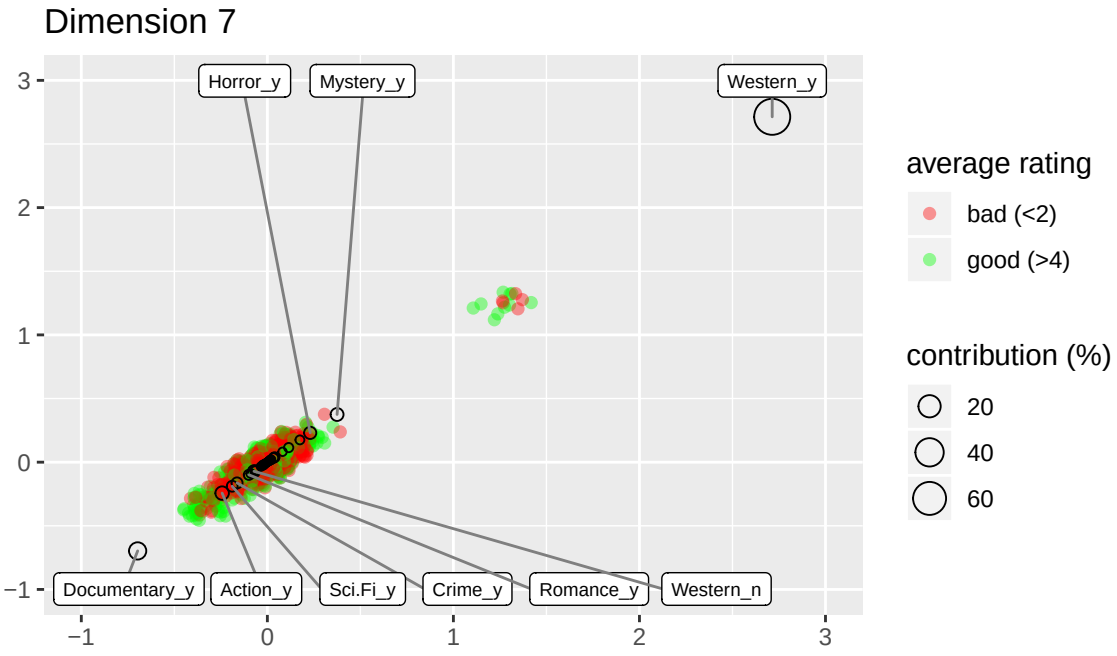
This dimension separates emotion-oriented movies (negative) from non emotion-oriented movies (?) (positive). No clear trend is noticeable in this dimension.

- Dimension 6



This dimension separates fiction (negative) from reality-based movies (positive). No clear trend is noticeable in this dimension.

- Dimension 7



This dimension essentially separates Western movies from the rest. No clear trend is noticeable in this dimension.

In conclusion, the MCA conducted here clearly shows that a movie's genres can be, in some cases, useful information to predict the rating of movies. Thus these variables should be considered when building and training our models.

4 Modeling Approach

4.1 Goal

The objective function for this project is simply the Root-Mean Square Error (RMSE) between the actual ratings y and the predicted ratings $\hat{y}$.

$$RMSE = \sqrt{\frac{1}{N}\sum(y - \hat{y})^2}$$

```
RMSE_fct <- function(test, pred) {
  sqrt(mean((test - pred) ^ 2))
}
```

The goal is to achieve a RMSE lower than 0.86490. At the end of this project, the model that will be used on the `validation` set will be the one with the lowest RMSE. Since the `validation` set should be used only after choosing the model, to avoid cherry-picking and overfitting, cross-validation will be applied on the `edx` set, to generate a training set and a testing set, on which the different models will be tested.

```
# seed for reproducible results
set.seed(2486)

# test_set contains 1/10 of the data
test_index <- createDataPartition(edx$rating,
                                  times = 1,
                                  p = 0.1,
                                  list = FALSE)
train_set <- edx[-test_index, ]
test_set <- edx[test_index, ]
remove(test_index)

# train & test should have only common variables values, to avoid NAs
test_set <- test_set %>%
  semi_join(train_set, by = "userId") %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "release_year")
```

The chosen modeling approach for this recommandation system is residuals' explanation.

4.2 Naive approach

First, the naive approach consists in predicting that any rating is just the average over all ratings μ, and errors ϵ are supposed to be residual random noise :

$$\hat{y} = \mu + \epsilon$$

```
# naive model
mod_naive <- function(train, test) {
  mu_rating <- mean(train$rating)
  test <- test %>%
    mutate(pred = mu_rating)
  test$pred
}
```

```
res <- data.frame()
```

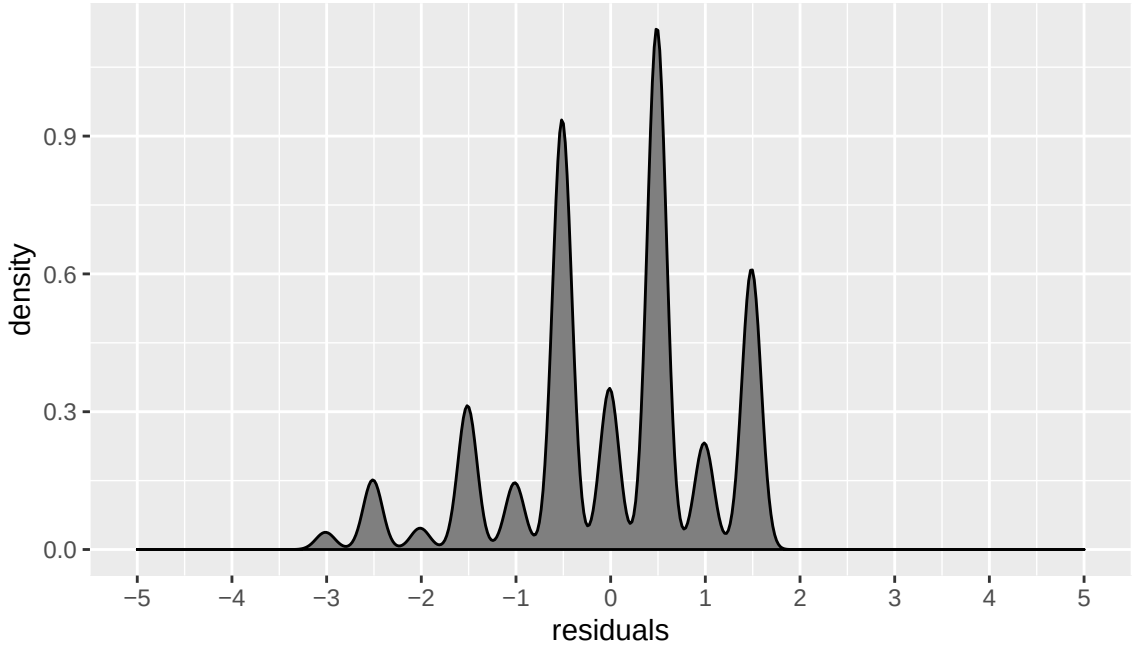
```
y_naive <- mod_naive(train_set, test_set)
res[1, 1] <- RMSE_fct(test_set$rating, y_naive)
```

If this model generates accurate predictions, errors should then be truly random, and it can be verified by checking its distribution, which should be normal, and centered at zero. If it is not the case, it means that the residuals still contain dormant information that have not been grasped by our model, and it could be improved. This test can be performed visually, by analysing the plot of the distribution of the residuals, but a metric should ease the comparison of models.

The Kolmogorov-Smirnov (KS) test has been chosen for this task. The KS test gives the maximum difference between the residuals' cumulative distribution and a normal cumulative distribution (centered at zero). A derivative metric is also evaluated: the relative error between the result of the KS test on the residuals, and the threshold under which the hypothesis of the residuals' distribution being a normal distribution cannot be rejected, with a 95% level of confidence. This second metric will be a clear indication on how much the residuals' distribution is different from a normal distribution.

```
# for alpha = 0.05 (95% confidence)
ks_crit <- 1.358 / sqrt(nrow(test_set))
```

By plotting the residuals' distribution for the naive approach, it is obvious that it is far from a normal distribution.



As expected, the naive approach is a very limited model, and a lot of information remain dormant in the residuals, which can not be considered as random noise in this case.

model	RMSE	KS	err_KS [%]
Naive	1.05998	0.177548	12303.2

As expected, the naive approach is a very limited model, and a lot of information remain dormant in the residuals, which can not be considered as random noise in this case.

4.3 Explicit biaises

One way to improve our model is to explicitely take into account the effect of certain variables. Data exploration revealed that ratings can change following the user (`userId`), the movie (`movieId`), the release year of the movie (`release_year`) and the different genres. Since the genres carry information on a multiple dimensionnal level (every genre has to be taken into account at once, or informations might be lost), it will not be directly considered in this project.

To compare the influence of each of these variables, and the different permutations, 15 different models, divided in three groups, will be tested. Each of these models will use an hyperparameter λ, that will be used to regularise the biaises.

```
# This function gives the optimal lambda (and RMSE) for given
# train/test sets and a model. The lambda associated with the lowest RMSE
# (with a given threshold) is designated as optimal.
lambda_opt <- function(train, test, fct_mod) {
  prec <- 0.1
  l_new <- 0
  i <- 2

  y <- fct_mod(train, test, l_new)
  rmse_l_new <- RMSE_fct(test$rating, y)

  out <- data.frame(l = l_new,
                    rmse = rmse_l_new)

  while (abs(i) > prec) {
    l_old <- l_new
    rmse_l_old <- rmse_l_new
    l_new <- l_old + i
    y <- fct_mod(train, test, l_new)
    rmse_l_new <- RMSE_fct(test$rating, y)
    if (rmse_l_new > rmse_l_old) {
      i <- -i / 2
    }
    out <- rbind(out,
                 c(l_new,
                   rmse_l_new))
  }
  out[nrow(out)-1,]
}
```

4.3.1 One biais

A single variable v is taken into account in these models (named `model_I`). They follow the formula :

$$\hat{y}_v = \mu + b_v + \epsilon$$

where

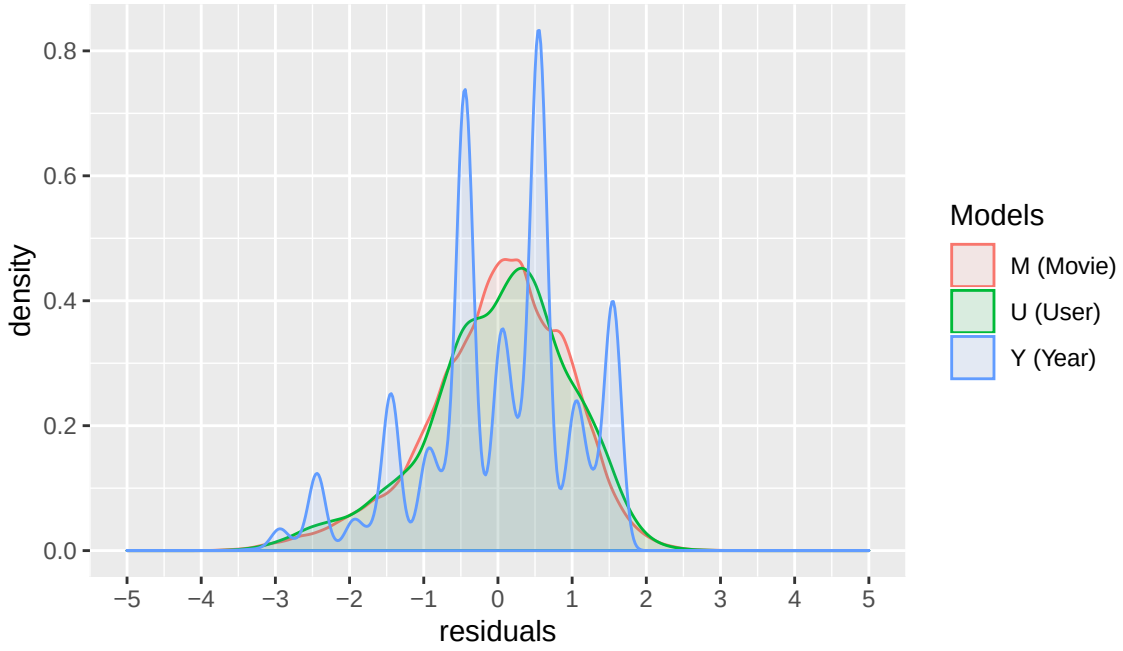
$$b_v = \frac{1}{\lambda + N} \sum^{N} (y - \mu)$$

```
# model_I - U
mod_bu <- function(train, test, lambda) {
  mu_rating <- mean(train$rating)
  user_biais <- train %>%
```

```
    group_by(userId) %>%
    summarise(b_user = sum(rating - mu_rating) / (n() + lambda))

  test <- test %>%
    left_join(user_biais, by = "userId") %>%
    mutate(pred = mu_rating + b_user)

  test$pred
}
```

These models are trained on the `train_set` and the RMSEs are calculated with the `test_set` data set.

```
l_opt <- lambda_opt(train_set, test_set, mod_bu)
res[1, 2] <- l_opt[1, 2]
y_U <- mod_bu(train_set, test_set, l_opt[1, 1])

l_opt <- lambda_opt(train_set, test_set, mod_bm)
res[1, 3] <- l_opt[1, 2]
y_M <- mod_bm(train_set, test_set, l_opt[1, 1])

l_opt <- lambda_opt(train_set, test_set, mod_by)
res[1, 4] <- l_opt[1, 2]
y_Y <- mod_by(train_set, test_set, l_opt[1, 1])
```



From only the graphical representation of the residuals' distribution, it appears that the variable `release_year` alone gives only a negligeable improvement, whereas the other two variables, `userId` and `movieId`, lead to significant improvements, as their residuals' distributions are closer to a normal distribution (approximate bell shape).

4.3.2 Two biaises

Two variables v_i are taken into account in these models (named `model_II`). They follow the formula :

$$\hat{y}_{v_1,v_2} = \mu + b_{v_1} + b_{v_2} + \epsilon$$

where

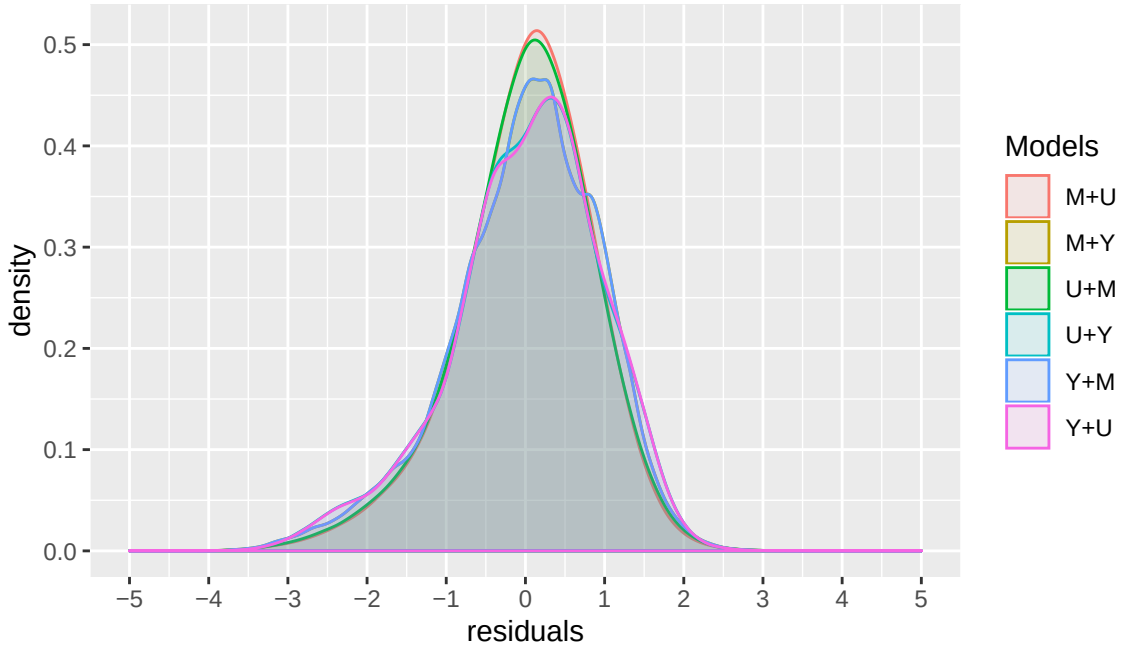
$$b_{v_1} = \frac{1}{\lambda + N} \sum^{N} (y - \mu)$$

$$b_{v_2} = \frac{1}{\lambda + N} \sum^{N} (y - \mu - b_{v_1})$$

Since b_{v_2} depends on b_{v_1}, one can wonder if the order matters. To answer this question, all the possible permutations will be attempted and compared.

```
# model_II - U + M
mod_bu_bm <- function(train, test, lambda) {
  mu_rating <- mean(train$rating)
  user_biais <- train %>%
    group_by(userId) %>%
    summarise(b_user = sum(rating - mu_rating) / (n() + lambda))

  movie_biais <- train %>%
    left_join(user_biais, by = "userId") %>%
    group_by(movieId) %>%
    summarise(b_movie = sum(rating - mu_rating - b_user) / (n() + lambda))

  test <- test %>%
    left_join(user_biais, by = "userId") %>%
    left_join(movie_biais, by = "movieId") %>%
    mutate(pred = mu_rating + b_user + b_movie)

  test$pred
}

# model_II - M + U
mod_bm_bu <- function(train, test, lambda) {
  mu_rating <- mean(train$rating)
  movie_biais <- train %>%
    group_by(movieId) %>%
    summarise(b_movie = sum(rating - mu_rating) / (n() + lambda))

  user_biais <- train %>%
    left_join(movie_biais, by = "movieId") %>%
    group_by(userId) %>%
    summarise(b_user = sum(rating - mu_rating - b_movie) / (n() + lambda))

  test <- test %>%
    left_join(movie_biais, by = "movieId") %>%
    left_join(user_biais, by = "userId") %>%
    mutate(pred = mu_rating + b_movie + b_user)

  test$pred
}
```

These models are trained on the `train_set` and the RMSEs are calculated with the `test_set` data set.



From only the graphical representation of the residuals' distribution, it appears that the combination of the variables `userId` and `movieId` are the models with the "most normal" residuals' distribution. Combinations with `release_year` display more irregularities. Also, the hierarchical order in the biaises seems to have only a negligeable impact on the residuals' distribution.

4.3.3 Three biaises

Three variables v_i are taken into account in these models (named `model_III`). They follow the formula :

$$\hat{y}_{v_1,v_2,v_3} = \mu + b_{v_1} + b_{v_2} + b_{v_3} + \epsilon$$

where

$$b_{v_1} = \frac{1}{\lambda + N} \sum^{N} (y - \mu)$$

$$b_{v_2} = \frac{1}{\lambda + N} \sum^{N} (y - \mu - b_{v_1})$$

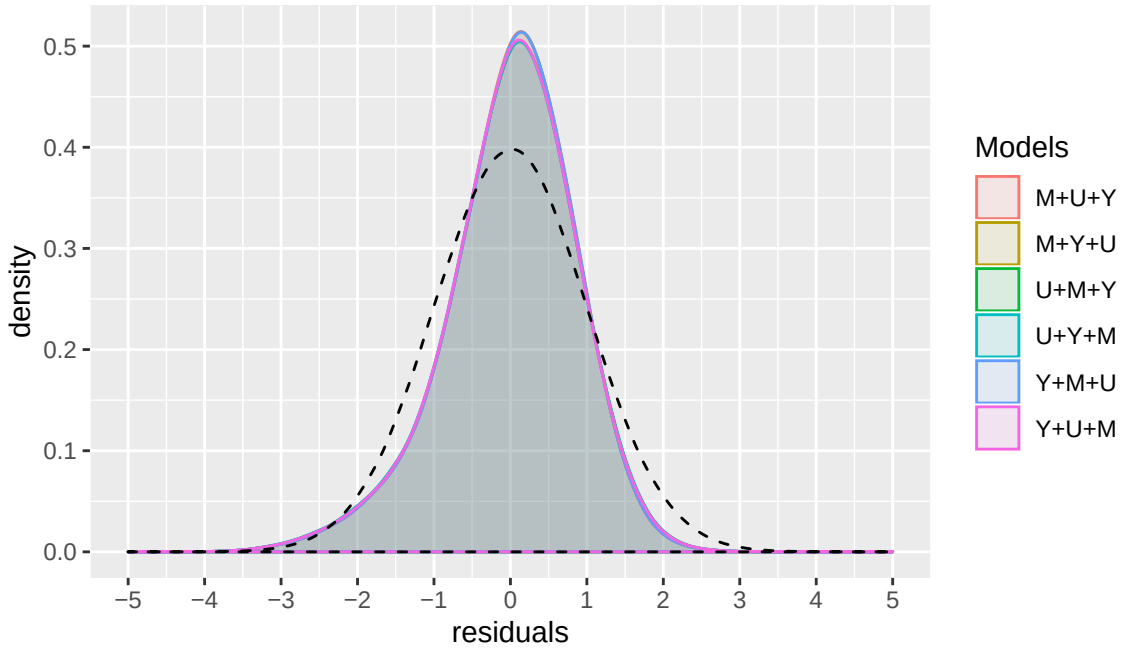
$$b_{v_3} = \frac{1}{\lambda + N} \sum^{N} (y - \mu - b_{v_1} - b_{v_2})$$

All the possible permutations will be attempted and compared.

```
# model_III - U + M + Y
mod_bu_bm_by <- function(train, test, lambda) {
  mu_rating <- mean(train$rating)
  user_biais <- train %>%
    group_by(userId) %>%
    summarise(b_user = sum(rating - mu_rating) / (n() + lambda))

  movie_biais <- train %>%
    left_join(user_biais, by = "userId") %>%
    group_by(movieId) %>%
    summarise(b_movie = sum(rating - mu_rating - b_user) / (n() + lambda))
```

```r
  year_biais <- train %>%
    left_join(user_biais, by = "userId") %>%
    left_join(movie_biais, by = "movieId") %>%
    group_by(release_year) %>%
    summarise(b_year = sum(rating - mu_rating - b_user - b_movie) / (n() + lambda))

  test <- test %>%
    left_join(user_biais, by = "userId") %>%
    left_join(movie_biais, by = "movieId") %>%
    left_join(year_biais, by = "release_year") %>%
    mutate(pred = mu_rating + b_user + b_movie + b_year)

  test$pred
}
```

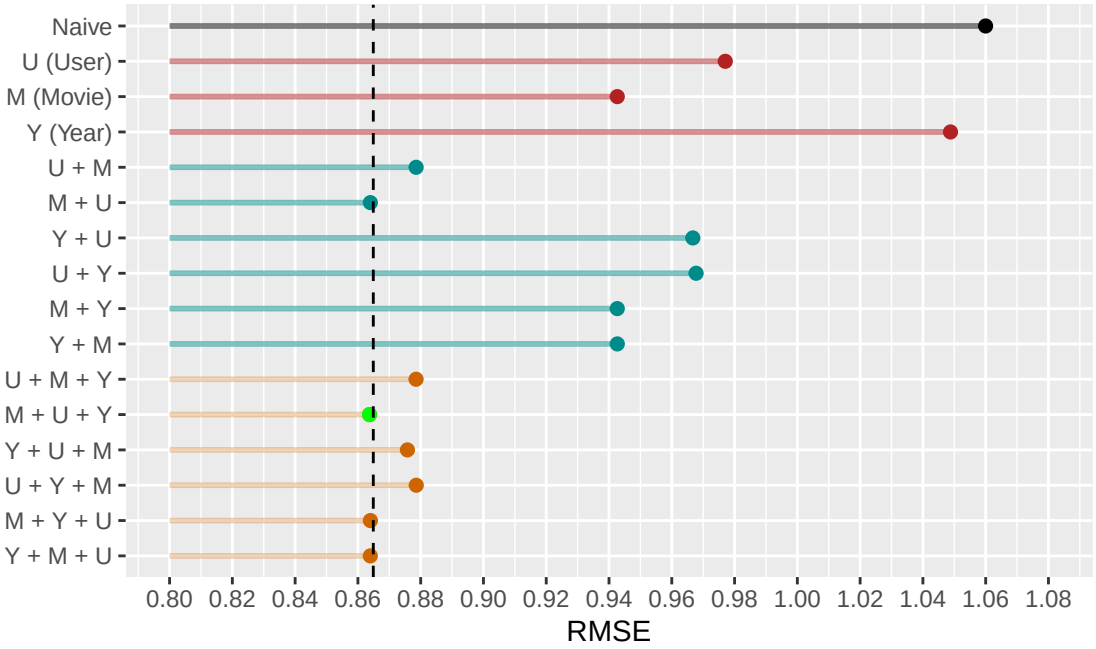
These models are trained on the `train_set` and the RMSEs are calculated with the `test_set` data set.



From only the graphical representation of the residuals' distribution, it appears that the combination of the three variables `userId`, `movieId` and `release_year` are equally good, for any hierarchical order. The normal distribution is plotted here (black dashed line) for comparison. It seems that the residuals' distributions are still a little bit shifted to positive values, with a longer tail toward negative values.

model	RMSE	KS	err_KS [%]
Naive	1.059978	0.177548	12303.16
U (User)	0.977039	0.041346	2788.34
M (Movie)	0.942635	0.044694	3022.21
Y (Year)	1.048791	0.099310	6837.59
U + M	0.878555	0.037389	2511.90
M + U	0.863972	0.041035	2766.62
Y + U	0.966708	0.038905	2617.80
U + Y	0.967743	0.036430	2444.92
M + Y	0.942633	0.044925	3038.38

model	RMSE	KS	err_KS [%]
Y + M	0.942639	0.044613	3016.56
U + M + Y	0.878540	0.038610	2597.26
M + U + Y	0.863702	0.040052	2697.98
Y + U + M	0.875778	0.037171	2496.67
U + Y + M	0.878597	0.037029	2486.81
M + Y + U	0.863985	0.041064	2768.68
Y + M + U	0.863957	0.041052	2767.83



By comparing the RMSEs from the different attempted models, it appears that the `movieId` biais has the highest influence on the quality of the model, and the `release_year` biais the lowest. It seems that the hierarchical order of the biaises have a small, but existent, influence on the RMSE.

From the results on this graph and this table, the best model is the M+U+Y model, achieving a RMSE of 0.863702, which is already below the goal of 0.86490 !

But, for practice sake, this model can be improved even further, as the relative errors `err_KS` show that our best model's KS value is still around 2500%, which entices us to think their is room for improvement. As shown in the data exploration section, the different genres of the movies may carry additional informations that could refine the predictions of this model. Instead of trying to compute explicit biaises for each genre encountered, matrix factorisation will be used to try to grasp some implicit insight from the data set.

4.4 Matrix factorisation

Matrix facotrisation, at the cost of high computer ressources, can explain residuals by decomposing them in the best possible way. The algorithm used here is the `funkSVD` function, that is very efficient for large sparse matrices.

$$\hat{y}_{SVD} = \mu + b_M + b_U + b_Y + \sum_{k}^{7} p_{U,k} q_{M,k} + \epsilon$$

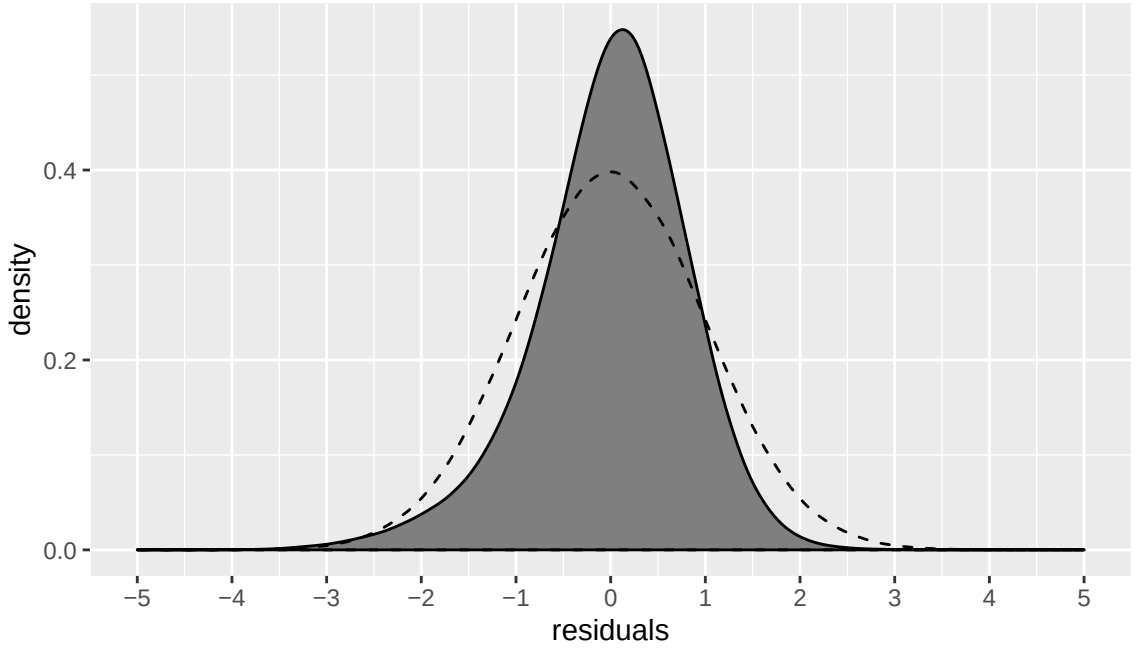
```
mod_bm_bu_by_SVD <- function(train, test, lambda) {
  mu_rating <- mean(train$rating)
```

```
movie_biais <- train %>%
  group_by(movieId) %>%
  summarise(b_movie = sum(rating - mu_rating) / (n() + lambda))

user_biais <- train %>%
  left_join(movie_biais, by = "movieId") %>%
  group_by(userId) %>%
  summarise(b_user = sum(rating - mu_rating - b_movie) / (n() + lambda))

year_biais <- train %>%
  left_join(user_biais, by = "userId") %>%
  left_join(movie_biais, by = "movieId") %>%
  group_by(release_year) %>%
  summarise(b_year = sum(rating - mu_rating - b_user - b_movie) / (n() + lambda))

resids <- train %>%
  left_join(movie_biais, by = "movieId") %>%
  left_join(user_biais, by = "userId") %>%
  left_join(year_biais, by = "release_year") %>%
  mutate(res = rating - (mu_rating + b_movie + b_user + b_year)) %>%
  select(userId, movieId, res) %>%
  spread(movieId, res)

rownames(resids) <- resids[, 1]
resids <- resids[, -1]

res_SVD <- funkSVD(resids,
                   k = 7,
                   gamma = 0.015,
                   lambda = 0.001,
                   min_improvement = 1e-06,
                   min_epochs = 30,
                   max_epochs = 50,
                   verbose = TRUE)

r <- tcrossprod(res_SVD$U, res_SVD$V)

rownames(r) <- rownames(resids)
colnames(r) <- colnames(resids)
remove(resids)

test$res_corr <- 0
test$res_corr <- sapply(1:nrow(test), function(x) {
  r[which(test$userId[x] == rownames(r)),
    which(test$movieId[x] == colnames(r))]
})
remove(r)

test <- test %>%
  left_join(movie_biais, by = "movieId") %>%
  left_join(user_biais, by = "userId") %>%
  left_join(year_biais, by = "release_year") %>%
  mutate(pred = mu_rating + b_movie + b_user + b_year + res_corr)
```

```
  test$pred
}
```

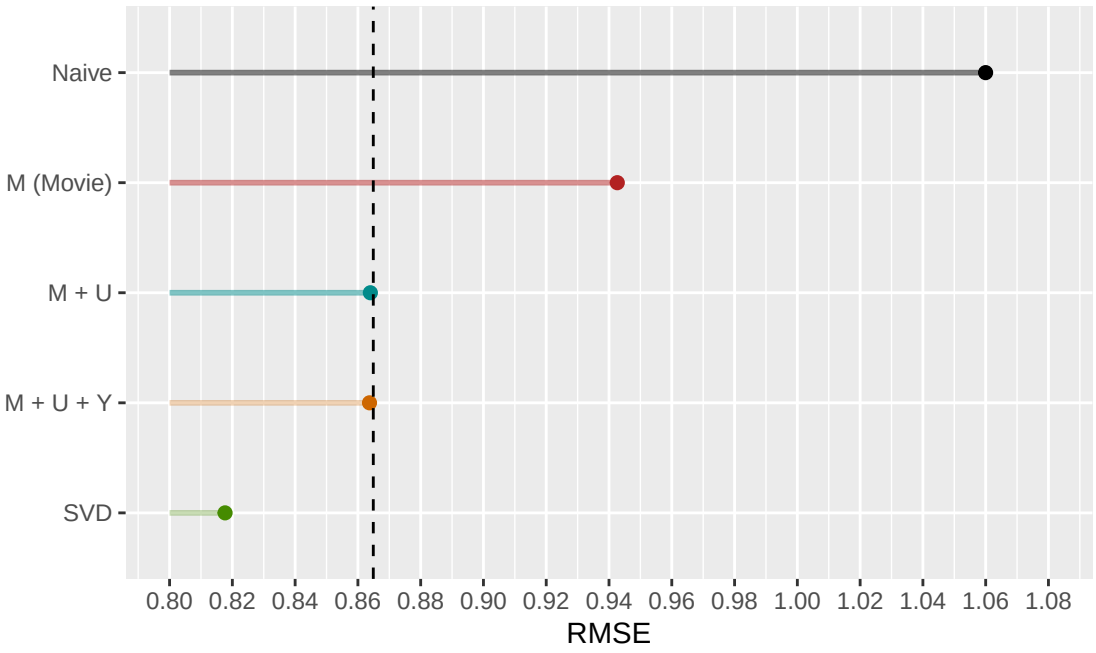
This algorithm, even with the present parameters (low number of explanatory features `k = 7`, low number of maximum iteration `max_epochs = 50`), has a high cost of computing ressources, and it took around **3 hours and a half**, with 30 GB of RAM (the residuals' matrix alone weights 5.6 GB) and an Intel i7-4790K (4 GHz) CPU.

```
# # /!\ 3h30 /!\
# l_opt <- lambda_opt(train_set, test_set, mod_bm_bu_by) #1min
# y_M_U_Y_SVD <- mod_bm_bu_by_SVD(train_set, test_set, l_opt[1, 1])
# save(y_M_U_Y_SVD, file = "./pred_SVD.Rda")

# the results has to be precalculated, as the Rmd compilation could not go through
load(file = "./pred_SVD.Rda")
res[1, 17] <- RMSE_fct(test_set$rating, y_M_U_Y_SVD) # 0.817679
```



From only the graphical representation of the residuals' distribution, no clear improvement is noticeable, in comparison with the M + U + Y model without SVD. The normal distribution is plotted here (black dashed line) for comparison. The residuals' distributions are still a little bit shifted to positive values, with a longer tail toward negative values.

model	RMSE	KS	err_KS [%]
Naive	1.059978	0.177548	12303.16
M (Movie)	0.942635	0.044694	3022.21
M + U	0.863972	0.041035	2766.62
M + U + Y	0.863702	0.040052	2697.98
SVD	0.817679	0.038664	2600.99



Despite its high computational cost, matrix factorisation, through the `funkSVD` function, manages to improve significantly the RMSE.

This is the best model attempted so far, and even if it could still be optimized even more, for example with different parameters, this model is thus designated as our definitive model for this project.

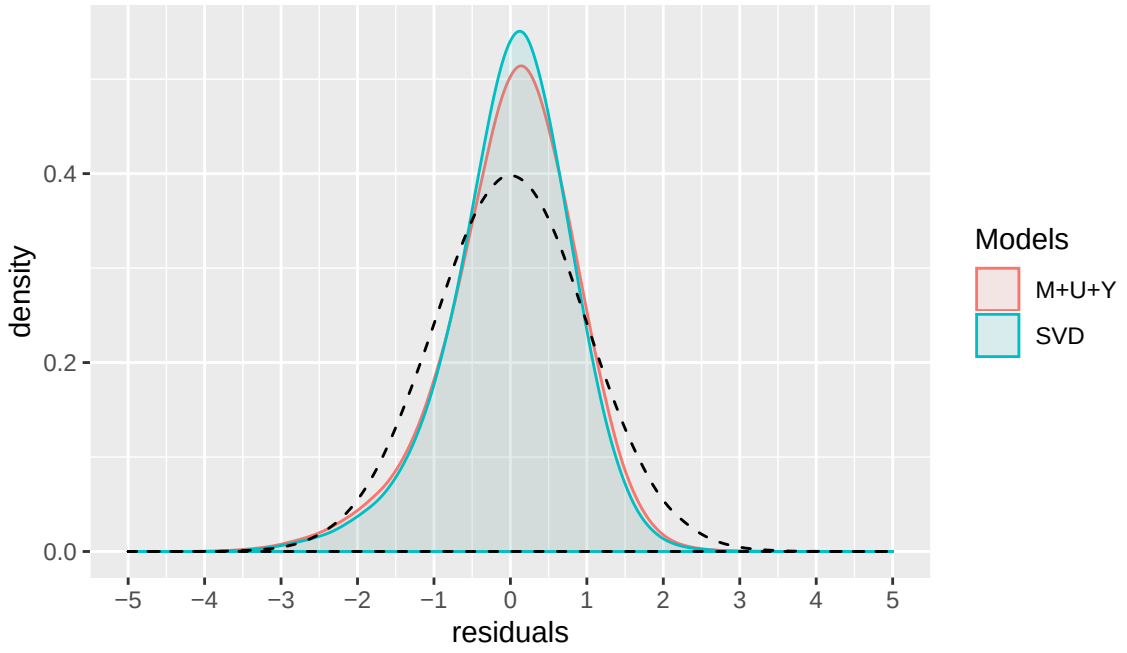
For practical reasons, the non-SVD M + U + Y model will also be considered as an acceptable model, since it is not as costly, and will provide a RMSE (hopefully under the fixed goal of 0.86490) in less than 5 min.

5 Results

For reasons presented in the previous section, two models have been selected : M + U + Y and M + U + Y + SVD.

The selected models are now trained on the full `edx` data set, and the final RMSEs are calculated with the `validation` data set.

```
res <- data.frame()

# M + U + Y model
l_opt <- lambda_opt(edx, validation, mod_bm_bu_by) #1min
res[1, 1] <- l_opt[1, 2] # 0.864522
y_M_U_Y <- mod_bm_bu_by(edx, validation, l_opt[1, 1])

# M + U + Y + SVD model   /!\ 3h30 /!\
# y_M_U_Y_SVD <- mod_bm_bu_by_SVD(edx, validation, l_opt[1, 1])
# save(y_M_U_Y_SVD, file = "./pred_SVD_final.Rda")

# the results has to be precalculated, as the Rmd compilation could not go through
load(file = "./pred_SVD_final.Rda")
res[1, 2] <- RMSE_fct(validation$rating, y_M_U_Y_SVD) # 0.815741
```



The graphical representation of the residuals' distributions are very similar to those obtained with the cross-validation on the `edx` data set alone. The normal distribution is plotted here (black dashed line) for comparison. The residuals' distributions are still a little bit shifted to positive values, with a longer tail toward negative values.

model	RMSE	KS	err_norm [%]
U + M + Y	0.864522	0.040908	2912.39
U + M + Y + SVD	0.815741	0.039179	2785.08

Both RMSEs are below the 0.86490 threshold.

6 Conclusion

The goal of this project is to build a recommandation system that can predict ratings from the MovieLens 10M data set, with a RMSE under 0.86490.

Once the data was retrieved from the Web, the training data set `edx` and the testing set `validation` were created. Nested information have been extracted (release year, different genres) to maximise the accessible data for training the models.

Data exploration of the different variables lead to the selection of relevant variables to feed the models with. MCA was performed on the genres of the movies to reveal potential further nested information.

The prediction attempt was focused on matrix facorisation, by adding biaises to a naive mean of all ratings. For multiple biaises, hierarchical order of the different biaises did not show any siginificant effect. The lowest RMSE was obtained with a model that added biaises for the user effect, the movie effect and the release year of the movie effect. An even lower RMSE was reached using SVD on the top of the previous model.

Then, the two models (with and without SVD) were trained on the `edx` and tested on the `validation`, and both RMSEs are under the 0.86490 threshold.

Regardless of the threshold, the models tested here can be further improved. Among the possible leads, the genre classification was not used in these models, and as the MCA shown, it could be used to generate either one bias by genre, or a few biais for frequently associated genres. Another lead is a SVD with a higher number of implicit features, and/or a higher number of allowed iterations for its optimization process. However, those leads may require a large quantity of RAM, and may be computationally expensive...