MovieLens Project

Esteban Emilio Rodriguez

1. Introduction

This project is part of the last course of the **EDX/HarvardX Data Science Specialization**. The goal is to create a **movie recommendation system in R for the MovieLens dataset**. In particular, we will be using the 10M version of the MovieLens dataset. The 3 files of this project are available at *https://github.com/esterodr/MovieLens*, and are the following:

- "movielens_esterodr.Rmd": The R markdown file that create this report. It contains the analysis, the justifications, and the tunning of the final model.
- "movielens_esterodr.pdf": The version of this report on pdf format.
- "final_model_esterodr.R": The code for the final model, that gives a **Root Mean Squared Error (RMSE) of 0.8347068** in the validation set. It consist on mostly the Section 3 of this report.

In Section 2 the *edx* and *validation* sets will be created, following by a description of the database. I will pretend to not have access to the validation set until Section 4, meaning that all the training/testing process will be made with the edx set. For this purpose, the *edx* set will be split in a *train* and in a *test* set.

Section 3 contains the construction of the model, training the model with the *train* set and tunning its paramaters according to the resulting predictions on the *test* set. The model will be constructed in a sequential way, showing at each step the progress in predicting the movie ratings according to the RMSE. After constructing the final model, in Section 4 it will be trained on the whole *edx set* and used it to predict the movie ratings of the *validation* set, reporting the RMSE obtained. The final Section of this report ressume the main conclusions of the project.

2. Creation and description of the datasets.

2.1 Creation of the datasets.

The *edx* and the *validation* sets were created with the code provided by the instructors, which is provided here to allow the reproduction of the project. However, as **the download and creation of the database takes time, the idea is to do this just once and save the sets in the local drive**. This is done with an *if* condition that checks if the database is already downloaded.

Note: If you are trying to reproduce the code, I strongly recommend you to do it in more than one session. Download the database with the code bellow, and save the *edx* and *validation* dataframes in an object named "edx.Rda".

```
if (file.exists("edx.Rda")) {
  load("edx.Rda")
} else {
  # The following code was provided by the instructors:

  ##############################################################
  # Create edx set, validation set, and submission file
  ##############################################################

  # 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")

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

  movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
  colnames(movies) <- c("movieId", "title", "genres")
  movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                             title = as.character(title),
                                             genres = as.character(genres))

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

  # Validation set will be 10% of MovieLens data

  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)

  ## The next line was introduced by me to avoid creating the datasets every time
  save(edx, validation, file = "edx.Rda")

}
```

With the excecution of the previous code, two objects have been created:

1. The "edx" dataframe, with 9000055 observations and 6 variables.
2. The "validation" dataframe, with 999999 observations and 6 variables.

I will try to predict ratings in the "validation" dataframe using the "edx" dataframe, pretending that I don't have acces the ratings of *validation* set. That means that I need to test the models with data from the *edx* set, so I need to split this dataset in a *train* and a *test* set. The *train* set will be the 90% of the *edx* set. Again, I will save the datasets in an object in the local drive named "train_test.Rda", avoiding the repetition of this process with an *if* condition. I recommend you to do the same if you are trying to reproduce the code. I will be using the *tidyverse* and the *caret* packages.

```
library(tidyverse)
library(caret)
```

```
if (file.exists("train_test.Rda")) {
  load("train_test.Rda")
} else {
  set.seed(1)
  test_index <- createDataPartition(y=edx$rating, times=1,p=0.1,list=FALSE)
  edx_train <- edx[-test_index,]
  temp <- edx[test_index,]

  # Make sure userId and movieId in test set are also in train set

  edx_test <- temp %>%
    semi_join(edx_train, by = "movieId") %>%
    semi_join(edx_train, by = "userId")

  # Add rows removed from test set back into train set

  removed <- anti_join(temp, edx_test)
  edx_train <- rbind(edx_train, removed)

  rm(temp, test_index, removed)

  # Save the datasets to avoid repeat this again
  save(edx_test, edx_train, file="train_test.Rda")
}
```

Now I have all the datasets needed to train, test and validate the model.

2.2 Despcription of the "edx" dataset.

The "edx" dataset contains 9000055 observations of 6 variables:

```
head(edx)
```

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

Each row of this dataframe consist in a unique pair of 'userId' and 'movieId' identificators, with the rating that the user gave to the movie, the date of the grade (in seconds since UTC January 1, 1970), the title and the genre of the rated movie. The number of unique movies and users is:

```
cat("Unique users:", n_distinct(edx$userId))
```

```
## Unique users: 69878
```

```
cat("Unique movies:", n_distinct(edx$movieId))
```

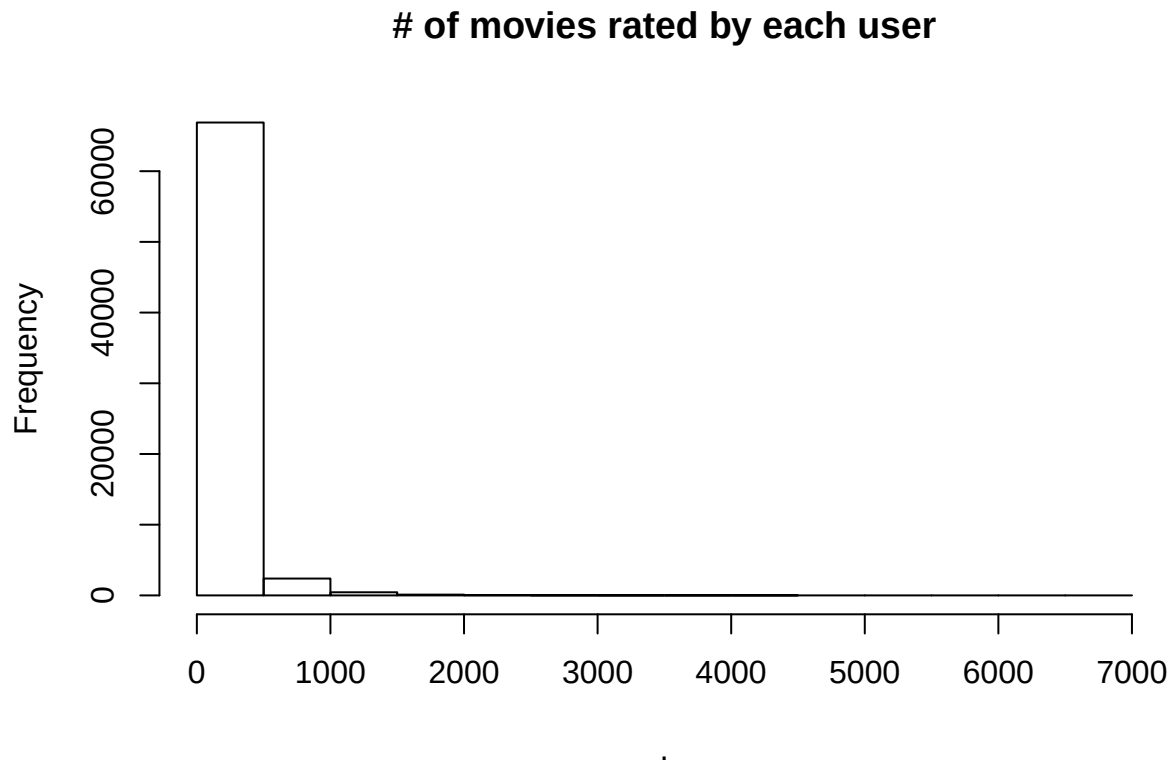
```
## Unique movies: 10677
```

Most of the users rated a few movies (25% of users rated 32 or less movies), while some of them rated a lot of movies (1% of users rated from 953 to 6616 movies). The median user rated 62 movies.

```
edx %>% group_by(userId) %>%
  summarise(n=n()) %>%
  ungroup() %>%
  .$n %>%
  quantile(c(0.25,0.5, 0.99, 1))
```

```
##  25%  50%  99% 100%
##   32   62  953 6616
```

```
edx %>% group_by(userId) %>%
  summarise(n=n()) %>%
  ungroup() %>%
  .$n %>%
  hist(main="# of movies rated by each user")
```



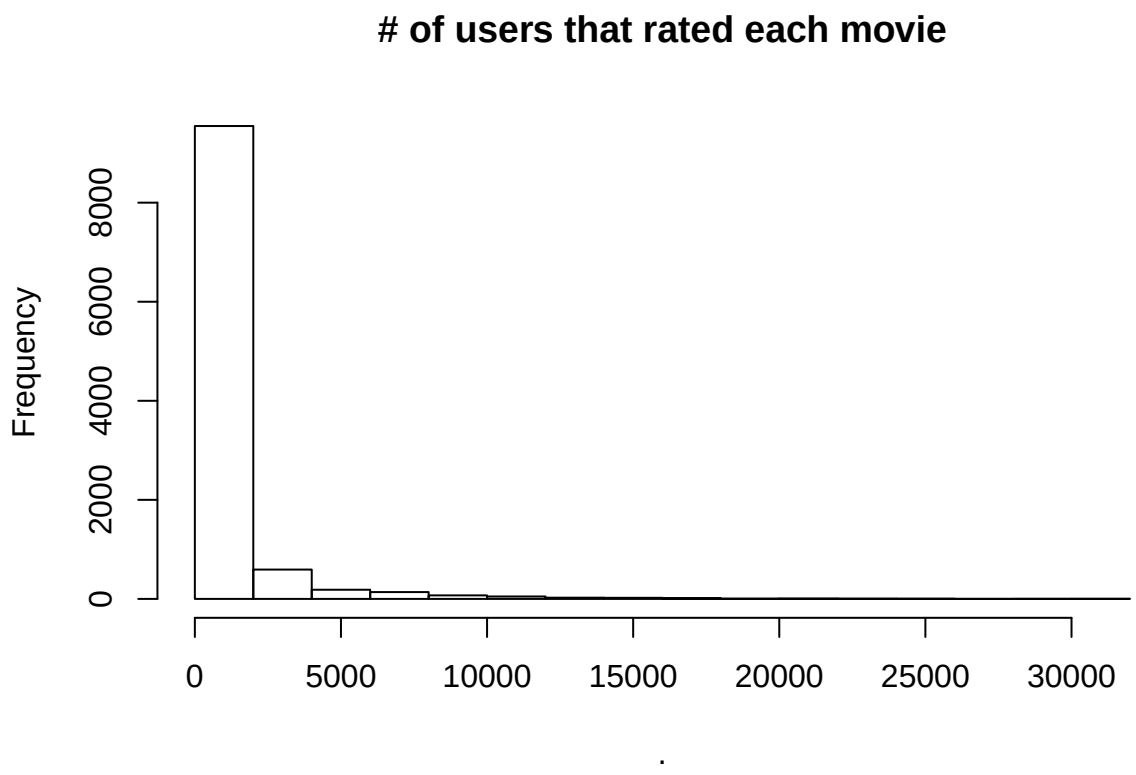
.

In a similar way, most of the movies received a few number of ratings (25% of the movies where rated 30 times or less) and a few movies received a lot of ratings (1% of the movies where rated from 11521 to 31362 times). The median movie was rated 122 times.

```
edx %>% group_by(movieId) %>%
  summarise(n=n()) %>%
  ungroup() %>%
  .$n %>%
  quantile(c(0.25,0.5, 0.99, 1))
```

```
##      25%      50%      99%     100%
##    30.00   122.00 11521.08 31362.00
```

```
edx %>% group_by(movieId) %>%
  summarise(n=n()) %>%
  ungroup() %>%
  .$n %>%
  hist(main="# of users that rated each movie")
```



.

For future use, I will save the times each movie was rated and the number of movies that each user has rate in the dataframes 'ratings_per_user' and 'ratings_per_movie':

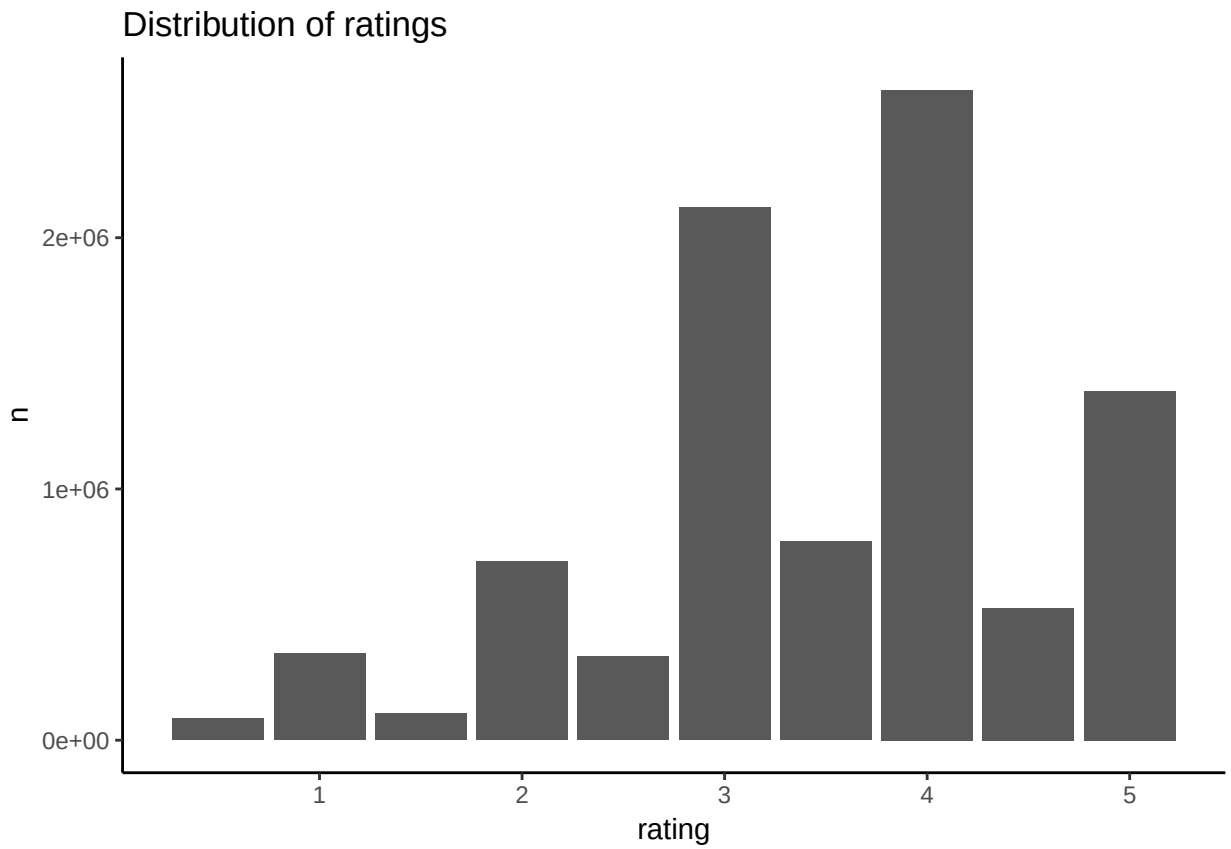
```
ratings_per_user <- edx %>%
  group_by(userId) %>%
  summarise(ratings_per_user = n())
ratings_per_movie <- edx %>%
  group_by(movieId) %>%
  summarise(ratings_per_movie = n())
```

The range of ratings is from 0.5 to 5. Around 80% of the ratings are integer numbers, with 4 being the most popular rating.

```
edx %>% group_by(rating) %>%
  summarise(n=n()) %>%
  mutate(p=n/sum(n))
```

```
## # A tibble: 10 x 3
##    rating       n       p
##     <dbl>   <int>   <dbl>
##  1    0.5   85374 0.00949
##  2    1    345679 0.0384
##  3    1.5  106426 0.0118
##  4    2    711422 0.0790
##  5    2.5  333010 0.0370
##  6    3   2121240 0.236
##  7    3.5  791624 0.0880
##  8    4   2588430 0.288
##  9    4.5  526736 0.0585
## 10    5   1390114 0.154
```

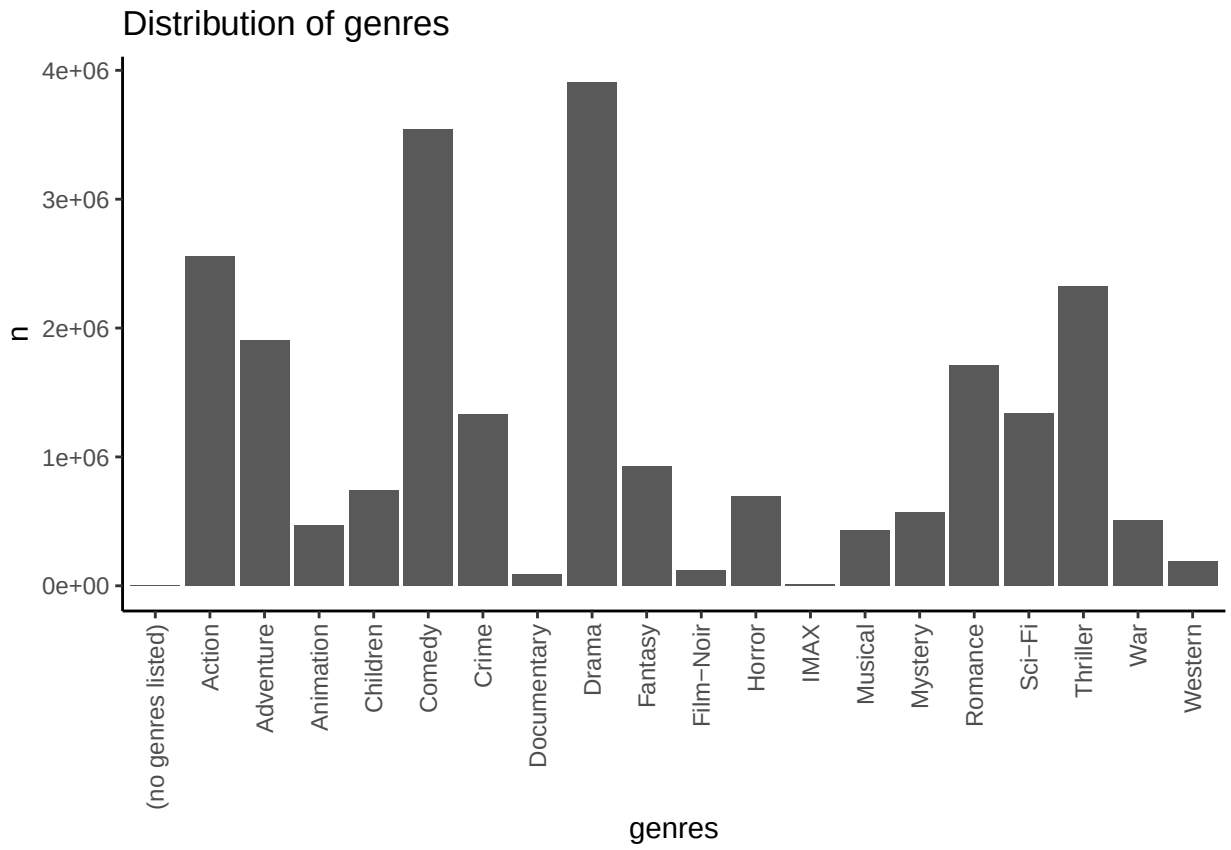
```
edx %>% group_by(rating) %>%
  summarise(n=n()) %>%
  ungroup() %>%
  ggplot(aes(x=rating, y=n)) +
  geom_col() +
  theme_classic() +
  ggtitle("Distribution of ratings")
```



The movies are classified according to 20 genres. However, most of the movies are classified under more than

one genre. 'Drama', 'Comedy', 'Action' and 'Thriller' are, in that order, the most frequent genres.

```
edx %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarise(n=n()) %>%
  ggplot(aes(x=genres, y = n)) +
  geom_col() +
  theme_classic() +
  ggtitle("Distribution of genres") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```



The last variable to consider is the *timestamp* variable. As it was mentioned before, it contains the date of the grade in seconds since UTC January 1, 1970. As I'm not expecting *time-effects* at the level of a second, I will create new variables with the dates at a day frequency:

```
edx$time_day <- as.Date(as.POSIXct(edx$timestamp , origin="1970-01-01"))
edx_train$time_day <- as.Date(as.POSIXct(edx_train$timestamp , origin="1970-01-01"))
edx_test$time_day <- as.Date(as.POSIXct(edx_test$timestamp , origin="1970-01-01"))
```

Let's explore the distribution of dates:

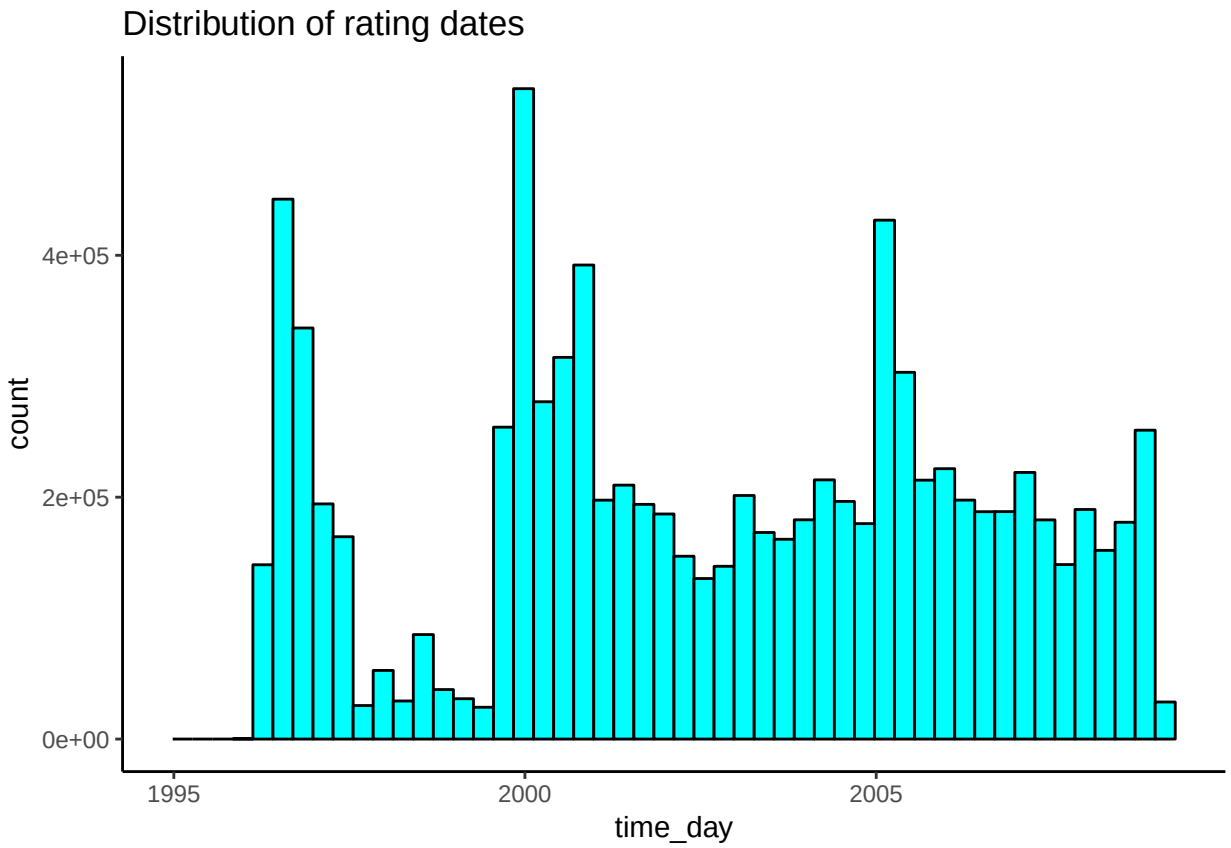
```
min(edx$time_day)
```

```
## [1] "1995-01-09"
```

```
max(edx$time_day)
```

```
## [1] "2009-01-05"
```

```r
edx %>% select(time_day) %>%
  ggplot() +
  geom_histogram(aes(x=time_day), bins=50, fill="cyan", color="black") +
  theme_classic() +
  ggtitle("Distribution of rating dates")
```



The range of dates is between "1995-01-09" and "2009-01-05", following a more or less uniform distribution, with some particulars peaks.

After this short description of the data, I will start the main task of the project: **the construction of a model to predict the rating that each user give to each movie**.

3. Analysis and develop of a model to predict movie ratings

This Section is the main one of the project. Here I will build the final model step by step, tunning some parameters and testing the results on the *edx_test* set. In concrete, I will divide the process in 3 major steps:

1. Baseline model
2. The addition of time variables to the baseline model
3. Matrix factorization.

At each step, the starting point will be the residual rating of the previous step. I will be showing the RMSE of the predictions at each step, which will be calculated with the following function:

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

3.1 Baseline model

The baseline model will predict that user u will give to movie i a rating consisting in three terms:

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

Where μ is the mean rating in the whole database, b_i is the mean rating for each movie (after considering μ), and b_u is the mean rating given by each user (after considering μ and b_i). The $\varepsilon_{u,i}$ is the error of the prediction (if the model is inbiased, the error should have a mean of 0).

The idea behind this model is straightforward: some movies are better than others, so they receive a higher rating than the average movie. On the other side, some users are more generous giving rates than others. The b_i and b_u factors should control this two effects.

Another thing to consider is that many movies where rated very few times, and many users rated very few movies. As a result, I will estimate b_i and b_u using *penalized least squares*. This means that these factors will be estimated with the following formulas:

$$\hat{b}_i(\lambda_1) = \frac{1}{\lambda_1 + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu})$$

$$\hat{b}_u(\lambda_2) = \frac{1}{\lambda_2 + n_u} \sum_{u=1}^{n_u} (Y_{u,i} - \hat{\mu} - \hat{b}_i(\lambda_1))$$

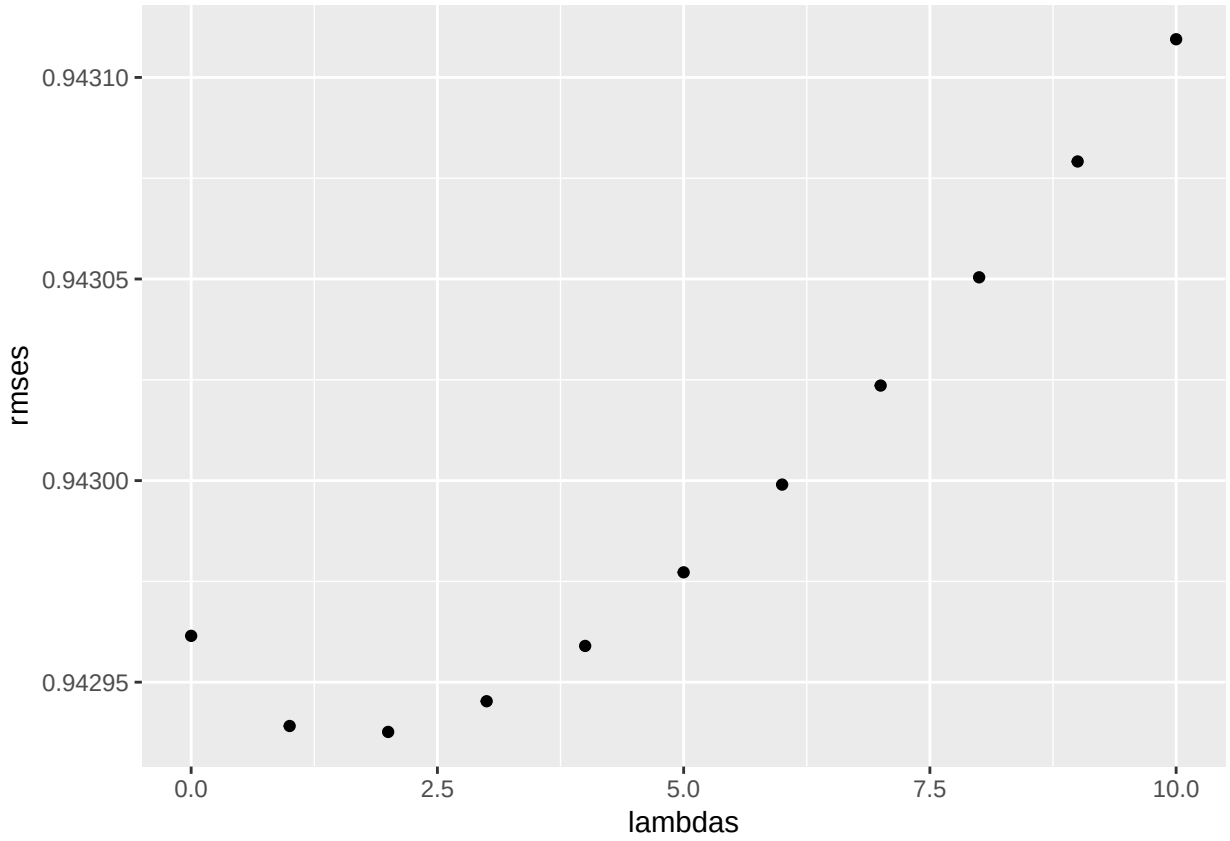
The idea of this approach is that, when n is large, $\lambda + n$ is close to n, but when n is small, the estimate $\hat{b}(\lambda)$ is shrunken towards 0. I will pick λ_1 and λ_2 according with the RMSE of the predictions. But first I need to estimate μ:

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

Now I will estimate λ_1

Note: Originally, I tested a larger set of values for λ_1. As this takes time, I present here just a few values around the best result.

```
lambdas <- seq(0, 10, 1)
rmses <- sapply(lambdas, function(l){

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

  predicted_ratings <-
    edx_test %>%
    left_join(b_i, by = "movieId") %>%
    mutate(pred = mu + b_i) %>%
    .$pred

  return(RMSE(edx_test$rating, predicted_ratings))
})
qplot(lambdas, rmses)
```




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

```
## [1] 2
```

We can see that $\lambda_1 = 2$ results in the lower RMSE, so I can estimate b_i with this value.

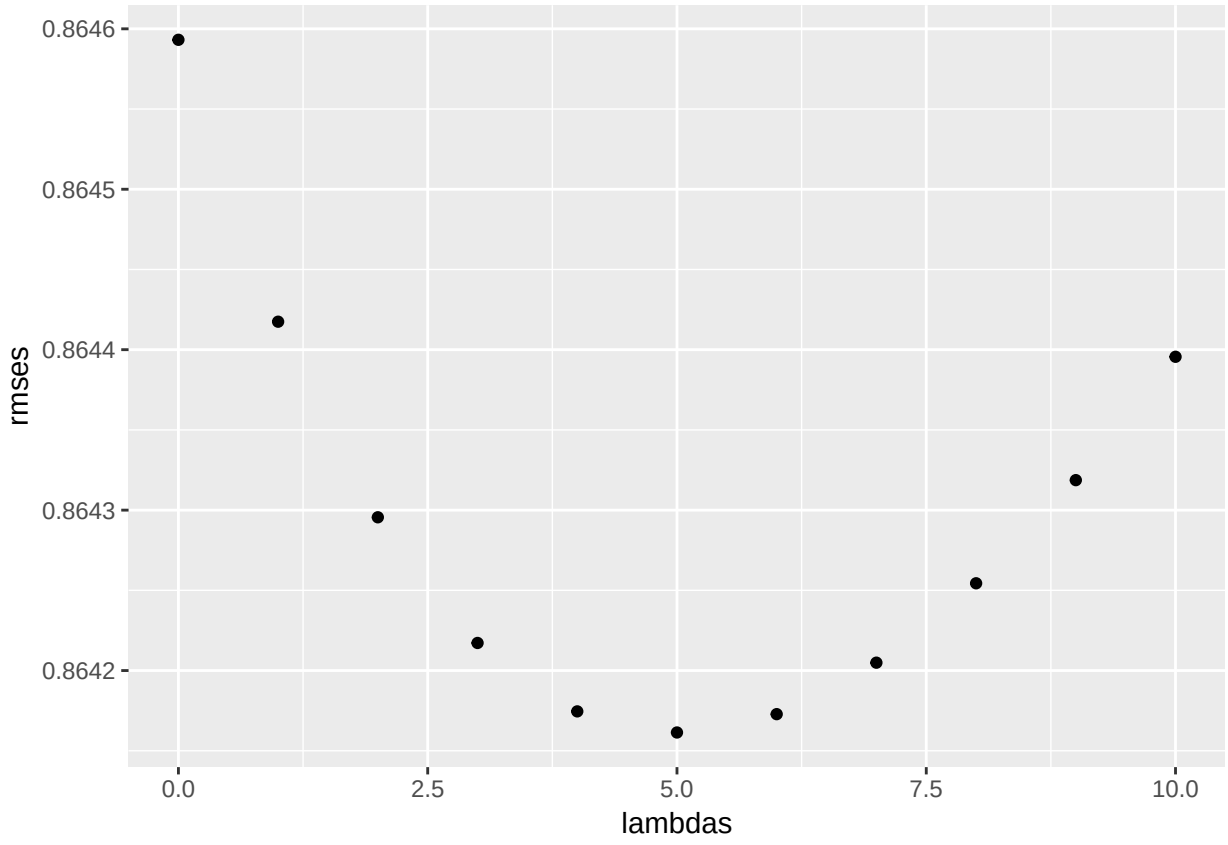
```
b_i <- edx_train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+2))
```

Now I will estimate λ_2:

Note: Originally, I tested a larger set of values for λ_2. As this takes time, I present here just a few values around the best result.

```
lambdas <- seq(0, 10, 1)
rmses <- sapply(lambdas, function(l){

  b_u <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  predicted_ratings <-
    edx_test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred
```

```
  return(RMSE(edx_test$rating, predicted_ratings))
})
qplot(lambdas, rmses)
```




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

```
## [1] 5
```

We can see that $\lambda_2 = 5$ results in the lower RMSE, so I can estimate b_u with this value.

```
b_u <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+5))
```

Now I can make the predictions for the *edx_test* set using this baseline model and check the RMSE:

```
predicted_ratings <-
  edx_test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId")  %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

baseline_rmse <- RMSE(edx_test$rating, predicted_ratings)
rmse_results <- data_frame(method = "Baseline model", RMSE = baseline_rmse)
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
```

```
## This warning is displayed once per session.
```

```
rmse_results %>% knitr::kable()
```

method	RMSE
Baseline model	0.8641614

The resulting RMSE is of 0.8641614.

3.2. Addition of time variables to the Baseline model

The way a movie is seen and the rating that a user gives can change over time, so we need to incorporate the time dimension to the analysis. I will follow a simplifyed version of the model described by Yehuda Koren in the paper "The BellKor Solution to the Netflix Grand Prize" (https://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf). Basicaly, the proposed solution consist in making b_i and b_u dependant on time:

"...an item's popularity may change over time. For example, movies can go in and out of popularity as triggered by external events such as the appearance of an actor in a new movie. This is manifested in our models by treating the item bias b_i as a function of time. The second major temporal effect allows users to change their baseline ratings over time. For example, a user who tended to rate an average movie"4 stars", may now rate such a movie"3 stars". This may reflect several factors including a natural drift in a user's rating scale, the fact that ratings are given in the context of other ratings that were given recently and also the fact that the identity of the rater within a household can change over time. Hence, in our models we take the parameter b_u as a function of time." (Koren, 2009)

However, I couldn't achieve any improvement in the RMSE with time-users effect, so I will just include the time dimension to the item biases. Definetly, this is an aspect that could be improved in the future.

For the item bias, the *BellKor Solution* consist in spliting the b_i into time-based bins, using a constant item bias for each time period:

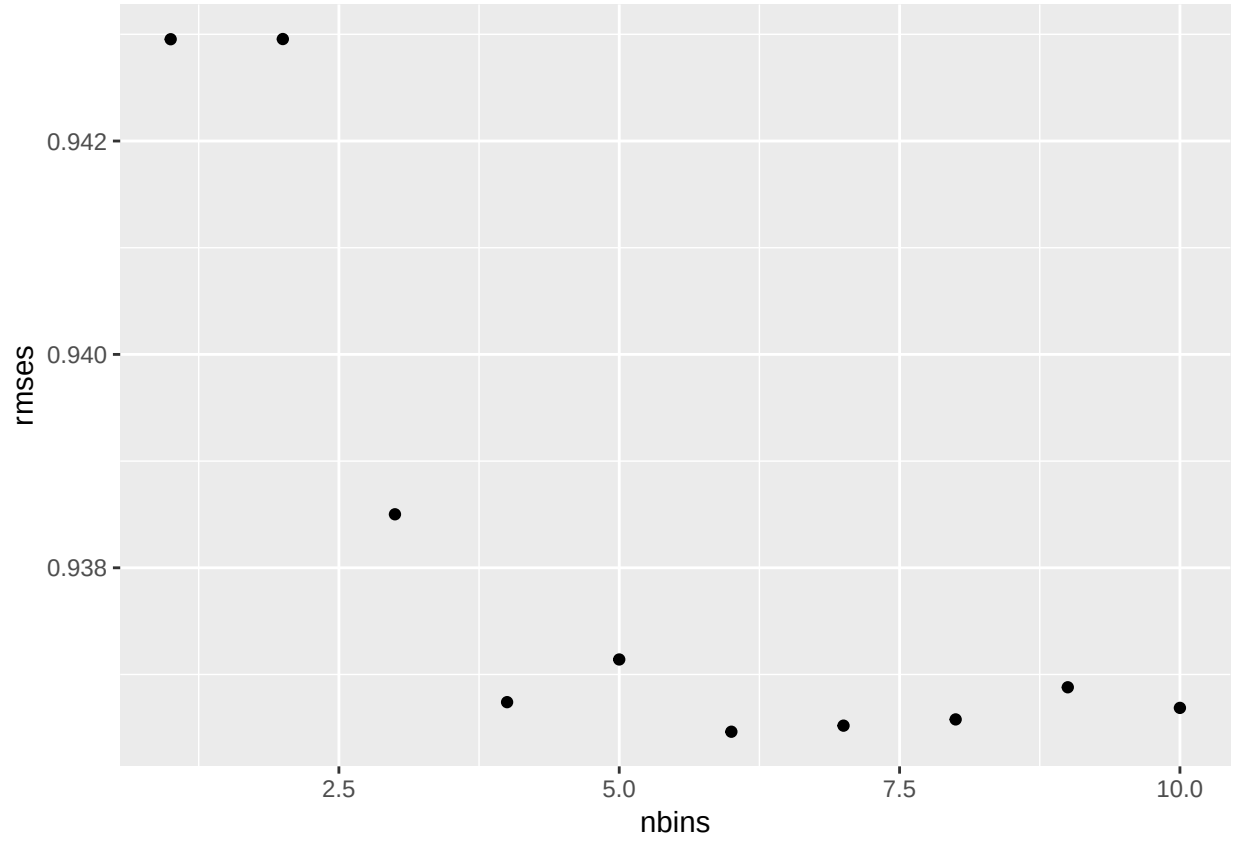
$$b_i(t) = b_i + b_{i,Bin(t)}$$

In my model, the b_i will be the one estimated in Section 3.1 and already saved in the b_i dataframe. For the other factor, after spliting the time spam of the dataset in a specific number of bins, I will estimate the mean of the residual of each movie in each bin. As some movies could have very few ratings in a bin, I will use a *lambda* factor as in Section 3.1. For simplicity, I will use the $\lambda_1 = 2$ already estimated. However, I need to decide the number of bins. I will try different numbers, and pick the one that result in a better RSME.

Note: Originally, I tested a larger number of bins. As this takes time, I present here just a few values around the best result.

```
nbins <- seq(1,10,1)
# In l1 I save the first rating in the dataset
# and in l2 I save the last rating in the dataset
l1 <- min(edx_train$time_day)
l2 <- max(edx_train$time_day)
rmses <- sapply(nbins, function(nb){

  # Calculate the bins
  bins <- seq(l1, l2, length.out = nb)

  b_it <- edx_train %>%
    # Asign each observation to a bin
    mutate(interval = findInterval(time_day, bins)) %>%
```

```
    # Incorporate the already estimated b_i
    left_join(b_i, by="movieId") %>%
    group_by(movieId, interval) %>%
    # Estimate b_it with a lambda=2
    summarise(b_it = sum(rating-mu-b_i)/(n()+2))

  predicted_ratings <-
    edx_test %>%
    left_join(b_i, by="movieId") %>%
    mutate(interval = findInterval(time_day, bins)) %>%
    left_join(b_it, by = c("movieId", "interval")) %>%
    replace_na(list(b_it=0)) %>%
    mutate(pred = mu + b_i + b_it) %>%
    .$pred

  return(RMSE(edx_test$rating, predicted_ratings))
})
qplot(nbins, rmses)
```




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

```
## [1] 6
```

The minimum RMSE is with 6 bins, so I estimate the **time dependant item effect** $b_i(t)$ with 6 bins:

```
bins <- seq(l1, l2, length.out = 6)
b_it <- edx_train %>%
```

```
  left_join(b_i, by="movieId") %>%
  mutate(interval = findInterval(time_day, bins)) %>%
  group_by(movieId, interval) %>%
  summarise(b_it = sum(rating-mu-b_i)/(n()+2), b_i=first(b_i)) %>%
  ungroup() %>%
  mutate(b_it = b_i + b_it) %>%
  select(-b_i)
```

As an example of what the $b_i(t)$ factor is trying to capture, let's see the evolution of the ratings received by Jim Carrey's movie "The Mask" through the different bins:

```
edx_train %>%
   filter(movieId==367) %>%
   mutate(interval = findInterval(time_day, bins)) %>%
   group_by(interval) %>%
   summarise(rating=mean(rating))
```

```
## # A tibble: 6 x 2
##   interval rating
##      <int>  <dbl>
## 1        1   3.34
## 2        2   3.32
## 3        3   3.22
## 4        4   2.97
## 5        5   2.92
## 6        6   2.5
```

We can see a downward tendency there, and the $b_i(t)$ factor should be able to capture it. Let's see if the addition of the $b_i(t)$ factor improves the RMSE. As the b_u factor was estimated after b_i, I will have to estimate it again considering $b_i(t)$ instead. Again, for simplicity, I will use the same $\lambda_2 = 5$ factor.

```
b_u <- edx_train %>%
   mutate(interval = findInterval(time_day, bins)) %>%
   left_join(b_it, by=c("movieId", "interval")) %>%
   group_by(userId) %>%
   summarize(b_u = sum(rating - b_it - mu)/(n()+5))

predicted_ratings <-
  edx_test %>%
  left_join(b_u, by="userId") %>%
  mutate(interval = findInterval(time_day, bins)) %>%
  left_join(b_it, by = c("movieId", "interval")) %>%
  replace_na(list(b_it=0)) %>%
  mutate(pred = mu + b_it + b_u) %>%
  .$pred

time_items_rmse <- RMSE(edx_test$rating, predicted_ratings)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Baseline Model + time-item effect",
                                     RMSE = time_items_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Baseline model	0.8641614
Baseline Model + time-item effect	0.8603403

We can see an improvement in the RMSE up to 0.8603403, that is a reduction of 0.44% from the previous estimation. It may seem a not very impressive gain, but it removes one factor that affects ratings and could be challenging to capture through the matrix factorization conducted next.

3.3. Matrix factorization

Even with the adition with temporal effects, the baseline model leaves out an important source of variation related to the fact that **groups of movies have similar rating patterns and groups of users have similar rating patterns as well**. I will try to identify some of these patterns starting from the residuals of the previous estimations:

$$r_{u,i} = Y_{u,i} - \hat{b}_{i,t} - \hat{b_u}$$

To study these residuals, I'm going to construct a matrix with each user in a row and each movie in a column. However, this matrix would be huge and mostly empty. In fact, $69,878$ rows x $10,677$ columns give us $746,087,406$ potential ratings, even though I only have $8,100,065$ ratings in the *edx_rain* set (around 1% of the matrix). These means that the estimation of the other 99% ratings wount be very precise, besides that the matrix factorization could be impossible to do on a regular computer or a limited time-spam. Considering this, I will focus in a smaller matrix with higher data density, where the identification of patterns between between movies and users would be easy (and faster) to find.

It's hard to find the optimal size of the matrix and I'm not aiming to look for it. I found that including in the matrix just the movies with at least 1500 ratings and just the users with at least 100 ratings, works fine. Of course, it's possible that another matrix size results in better predictions.

Let's filter the data and estimate the residuals:

```
resid <- edx_train %>%
  left_join(ratings_per_movie, by="movieId") %>%
  left_join(ratings_per_user, by="userId") %>%
  filter(ratings_per_movie >= 1500) %>%
  filter(ratings_per_user >= 100) %>%
  left_join(b_u, by="userId") %>%
  mutate(interval = findInterval(time_day, bins)) %>%
  left_join(b_it, by = c("movieId", "interval")) %>%
  replace_na(list(b_it=0)) %>%
  mutate(resid = rating - mu - b_it - b_u)
```

We can see that this result in $4,579,381$ observations. Let's construct the matrix and name properly its columns and rows

```
resid_matrix <- resid %>%
  select(userId, movieId, resid) %>%
  spread(movieId, resid) %>%
  as.matrix()
rownames(resid_matrix)<- resid_matrix[,1]
resid_matrix <- resid_matrix[,-1]
movie_titles <- edx %>%
  select(movieId, title) %>%
  distinct()
colnames(resid_matrix) <- with(movie_titles, title[match(colnames(resid_matrix), movieId)])
# I'm going to delete the "resid" object from memory
rm(resid)
# Let's chech the dimmensions of the matrix
dim(resid_matrix)
```

```
## [1] 24115  1448
```

We have a matrix with 24,115 rows and 1,448 columns. This means that we have more than 13% of the 34,918,520 possible ratings, not so bad to try an estimation with principal components. Let's take a look at the first 5 rows and colums of the matrix:

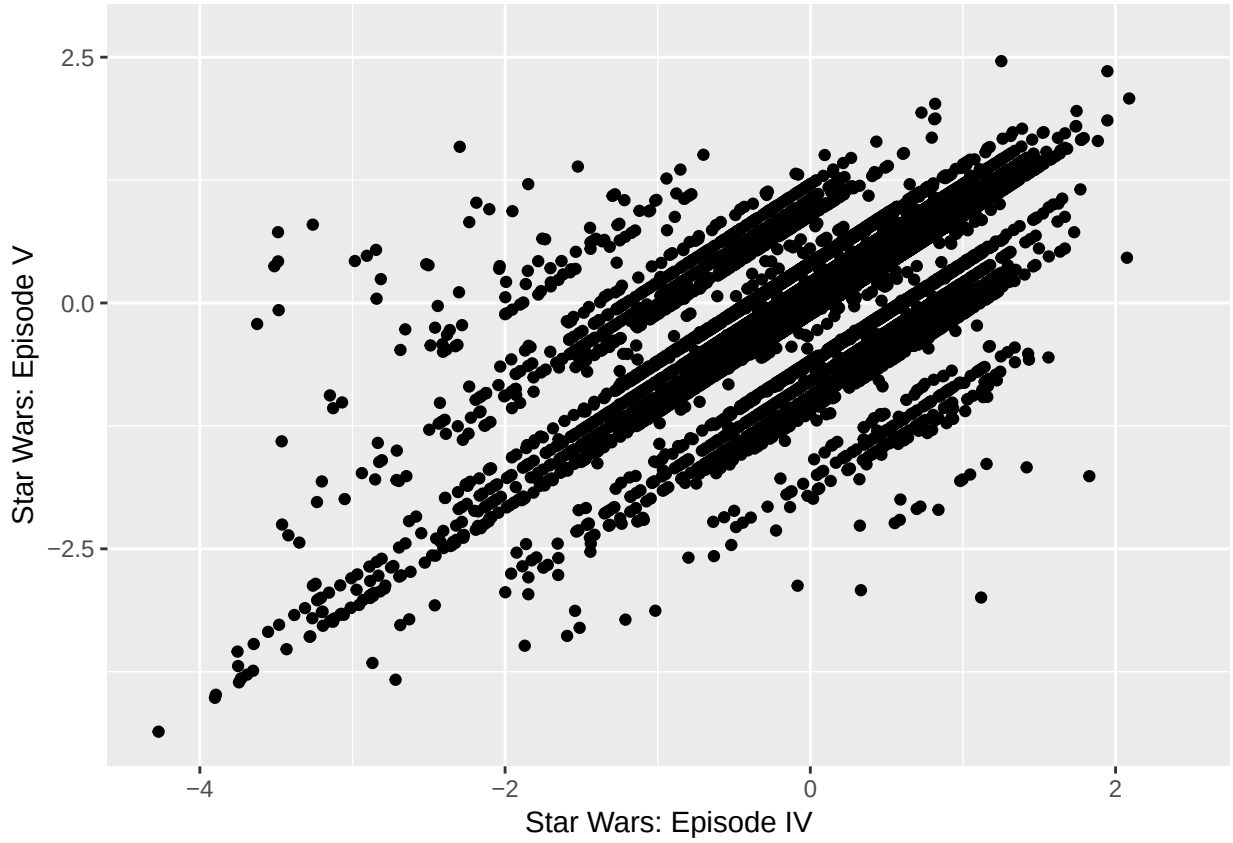
```
resid_matrix[1:4,1:4]
```

```
##    Toy Story (1995) Jumanji (1995) Grumpier Old Men (1995)
## 8                NA    -0.63117539                       NA
## 10               NA             NA                       NA
## 13               NA    -0.15378455                       NA
## 18       -0.9956545    -0.05499395                       NA
##    Waiting to Exhale (1995)
## 8                        NA
## 10                       NA
## 13                       NA
## 18                       NA
```

Let's see an example of which are the patterns that I'll try to identify. In the book of the Specialization (https://rafalab.github.io/dsbook/matrix-factorization.html) it is shown that the resids of The Godfather I and II are highly correlated. Let's see if the same happens with Episodes IV and V of Star Wars.

```
qplot(resid_matrix[,"Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)"],
      resid_matrix[,"Star Wars: Episode V - The Empire Strikes Back (1980)"],
      xlab = "Star Wars: Episode IV", ylab = "Star Wars: Episode V")
```

```
## Warning: Removed 14555 rows containing missing values (geom_point).
```



We can see that the resids are highly correlated: used that likes one movie also like the other one.

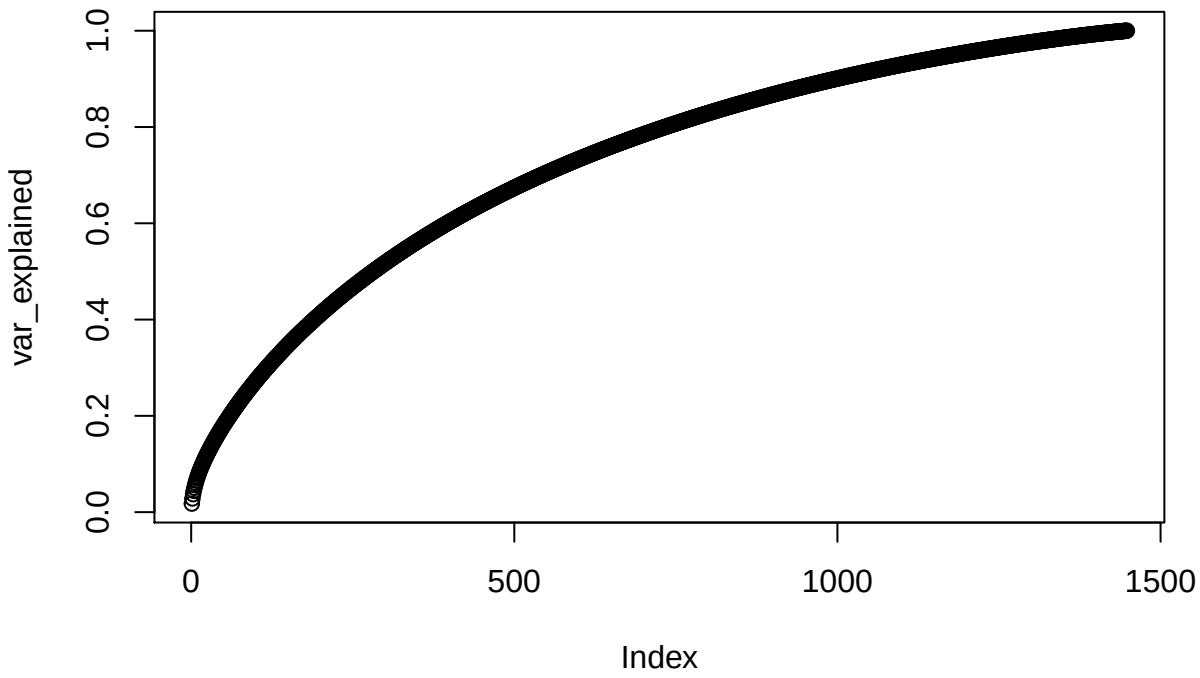
Before computing Principal Components, we need to replace the NAs with some value, 0 for simplicity. This will have consequences at the moment of decide the number of principal component to use for prediction: if we use too many principal components, we will predict a matrix of 0s.

Note: If you are going to reproduce the following code, consider that estimating the principal components could take several minutes. If you are going to work in more than one session, you can save the results in an object named "pca.Rda"

```
resid_matrix[is.na(resid_matrix)] <- 0
if(file.exists("pca.Rda")) {
  load("pca.Rda")
} else {
  pca <- prcomp(resid_matrix, center = TRUE, scale = FALSE)
}
```

Let's take a look to the cummulative variance explained by the principal components:

```
var_explained <- cumsum(pca$sdev^2/sum(pca$sdev^2))
plot(var_explained)
```



As we expected, the firsts commponents explains a higher portion of the variance than the last ones. Let's see if there is some pattern in the movies according to the first principal component:

```
pcs <- data.frame(pca$rotation, name = colnames(resid_matrix))
pcs %>% select(name, PC1) %>% arrange(PC1) %>% slice(1:10)
```

```
##                                              name         PC1
## 1              Independence Day (a.k.a. ID4) (1996) -0.18824701
## 2                                 Armageddon (1998) -0.12828642
```

```
## 3  Star Wars: Episode I - The Phantom Menace (1999) -0.11757384
## 4                                     Titanic (1997) -0.10754404
## 5                                Pretty Woman (1990) -0.10648885
## 6                                     Twister (1996) -0.10335844
## 7                                Forrest Gump (1994) -0.10038766
## 8                               Jurassic Park (1993) -0.10012242
## 9                                       Ghost (1990) -0.09628441
## 10                                    Top Gun (1986) -0.09402455
```

```
pcs %>% select(name, PC1) %>% arrange(desc(PC1)) %>% slice(1:10)
```

```
##                             name        PC1
## 1            Pulp Fiction (1994) 0.15950158
## 2                  Fargo (1996) 0.13583080
## 3    Clockwork Orange, A (1971) 0.12712566
## 4  2001: A Space Odyssey (1968) 0.12201737
## 5   Being John Malkovich (1999) 0.09880215
## 6         American Beauty (1999) 0.09774839
## 7            Blade Runner (1982) 0.09400398
## 8          Apocalypse Now (1979) 0.09343667
## 9          Godfather, The (1972) 0.09026294
## 10            Taxi Driver (1976) 0.08882995
```

Movies of great popularity on one side, and movies very estimated by the critics on the other. Let's explore the second principal component:

```
pcs %>% select(name, PC2) %>% arrange(PC2) %>% slice(1:10)
```

```
##                                      name         PC2
## 1                 Johnny Mnemonic (1995) -0.04267280
## 2                  Batman & Robin (1997) -0.03950637
## 3                        Net, The (1995) -0.03887042
## 4                  Wild Wild West (1999) -0.03816506
## 5                           Congo (1995) -0.03727570
## 6               Gone in 60 Seconds (2000) -0.03623406
## 7                     Judge Dredd (1995) -0.03450636
## 8                       Tank Girl (1995) -0.03328574
## 9                    Pearl Harbor (2001) -0.03205831
## 10 Rocky Horror Picture Show, The (1975) -0.03184704
```

```
pcs %>% select(name, PC2) %>% arrange(desc(PC2)) %>% slice(1:10)
```

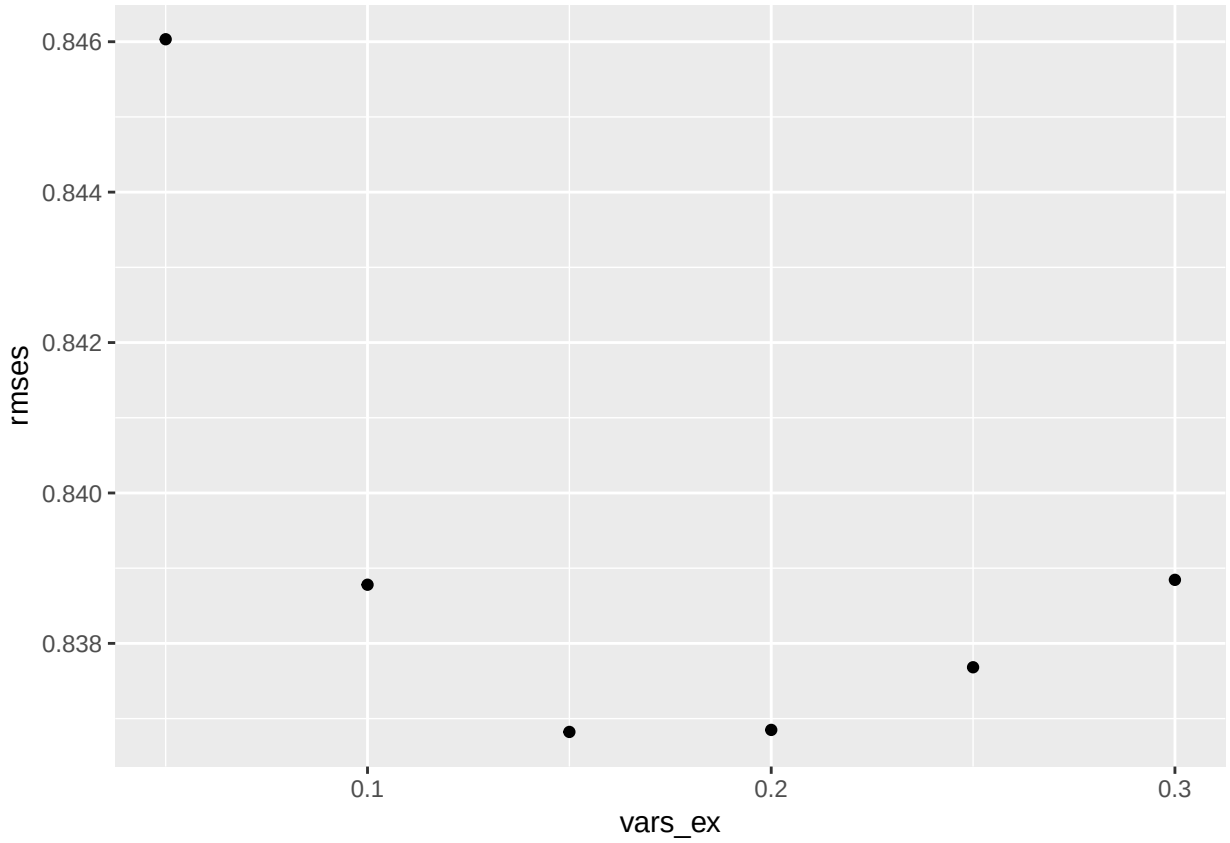
```
##                                                                                name
## 1                            Star Wars: Episode V - The Empire Strikes Back (1980)
## 2                     Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
## 3                                                                  Jurassic Park (1993)
## 4                               Star Wars: Episode VI - Return of the Jedi (1983)
## 5  Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)
## 6                                                 Terminator 2: Judgment Day (1991)
## 7                                                              Forrest Gump (1994)
## 8                                                                Braveheart (1995)
## 9                                                      Back to the Future (1985)
## 10                                                        Terminator, The (1984)
##          PC2
## 1  0.2360098
## 2  0.2312347
## 3  0.1945619
```

```
## 4  0.1889271
## 5  0.1774903
## 6  0.1766963
## 7  0.1647345
## 8  0.1522118
## 9  0.1516915
## 10 0.1494615
```

Again, we can see different types of moves at one extreme and at the other of the principal component. At one moment, the pattern in the principal components starts to become aleatory, so I need to decided how many principal component I'm going to use to estimate the resids of the matrix. I will try with different values of the variance explained, checking the RMSE of the predictions.

Note: Originally, I tested a larger number for the variance explained. As this takes time, I present here just a few values around the best result.

```
vars_ex <- seq(0.05,0.30,0.05)
rmses <- sapply(vars_ex, function(x){

  # Pick the number of PCs according to the variance explained
  pc.used <- which(var_explained>=x)[1]

  # Calculate the resids
  resid_pc <- pca$x[,1:pc.used] %*% t(pca$rotation[,1:pc.used])

  # Add the center back to data
  resid_pc <- scale(resid_pc, center = -1 * pca$center, scale=FALSE)

  # Convert the estimated resids to a dataframe
  pc_df <- as.data.frame(resid_pc)
  pc_df$userId <- as.numeric(row.names(pc_df))
  pc_df <- pc_df %>%
    gather(title, b_pc, -userId)

  # Use the results for predict ratings
  predicted_ratings <-
    edx_test %>%
    left_join(b_u, by="userId") %>%
    mutate(interval = findInterval(time_day, bins)) %>%
    left_join(b_it, by = c("movieId", "interval")) %>%
    left_join(pc_df, by=c("userId", "title")) %>%
    replace_na(list(b_it=0, b_pc=0)) %>%
    mutate(pred = mu + b_it + b_u + b_pc) %>%
    .$pred

  return(RMSE(edx_test$rating, predicted_ratings))
})
qplot(vars_ex, rmses)
```




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

```
## [1] 0.15
```

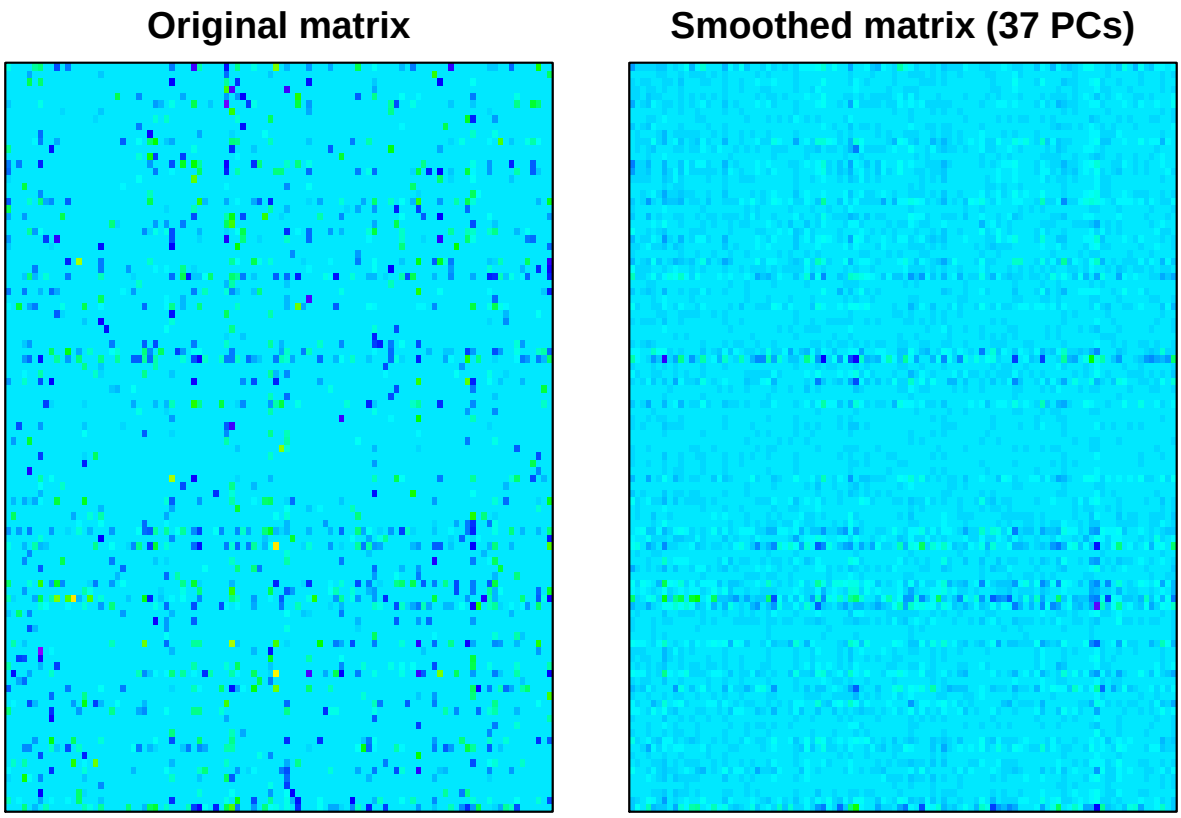
We can see that the best fit is the principal components that explains the 15% of the variance of the matrix. This corresponds to the first...

```
pc.used <- which(var_explained>=0.15)[1]
pc.used
```

```
## [1] 37
```

37 principal components. Let's take a graphical interpretation of what these 37 principal components represents for the resid matrix. Before, we need to construct the matrix from the resids:

```
# Matrix from the resids
resid_pc <- pca$x[,1:pc.used] %*% t(pca$rotation[,1:pc.used])
resid_pc <- scale(resid_pc, center = -1 * pca$center, scale=FALSE)

# Image of the first 100 rows and columns of the matrix
range <- range(cbind(resid_matrix, resid_pc))
breaks <- seq(range[1], range[2],,100)
colors <- rainbow(length(breaks)-1)
par(mfcol=c(1,2), mar=c(1,1,2,1))
image(resid_matrix[1:100,1:100], main="Original matrix", xlab="", ylab="", xaxt="n", yaxt="n", breaks=br
box()
image(resid_pc[1:100,1:100], main="Smoothed matrix (37 PCs)", xlab="", ylab="", xaxt="n", yaxt="n", brea
box()
```



We can see that some patterns of the original matrix of resids are present in the resids matrix estimated from the 37 principal component. The rightside matrix is mucho noisier thand the original one, but this is for a good reason: it is not full of 0s!! Remember that if we take much more principal component we will predict a 0 for every unknown pair of user-movie. Let's estimate the final predictions for the *edx_test* dataset:

```
# Convert the estimated resids to a dataframe
  pc_df <- as.data.frame(resid_pc)
  pc_df$userId <- as.numeric(row.names(pc_df))
  pc_df <- pc_df %>%
    gather(title, b_pc, -userId)

  # Use the results for predict ratings
  predicted_ratings <-
    edx_test %>%
    left_join(b_u, by="userId") %>%
    mutate(interval = findInterval(time_day, bins)) %>%
    left_join(b_it, by = c("movieId", "interval")) %>%
    left_join(pc_df, by=c("userId", "title")) %>%
    replace_na(list(b_it=0, b_pc=0)) %>%
    mutate(pred = mu + b_it + b_u + b_pc) %>%
    .$pred

  pc_rmse <- RMSE(edx_test$rating, predicted_ratings)
  rmse_results <- bind_rows(rmse_results,
                            data_frame(method="Matrix Factorization",
                                       RMSE = pc_rmse))
  rmse_results %>% knitr::kable()
```

method	RMSE
Baseline model	0.8641614
Baseline Model + time-item effect	0.8603403
Matrix Factorization	0.8368228

The RMSE has improve to **0.8368228**. This is a reduction of 2.7% from the previous estimation and a 3.2% reduction from the baseline model.

Is it possible to do something more? Surelly, but it is not so easy. With the matrix factorization I only considered movies with at least 1500 ratings and users with at least 100 ratings. The problem with the rest of movies and users is the lack of data. Consider that the resid matrix had $34,918,520$ spots of the $746,087,406$ of the matrix of all the combinations of users and movies. And that for estimate that matrix we use $4,579,381$ of the *edx_train* dataset. That means that we have $8,100,065 - 4,579,381 = 3,520,684$ observations of the remaining $746,087,406 - 34,918,520 = 711,168,886$ potential ratings. That is, we have less than the 0,5% of the data.

Considering this, I decide to stop here, and the final model that will be used in the next section is the following:

Final Model:

$$Y_{u,i} = \mu + b_{i,t}(\lambda_1) + b_u(\lambda_2) + b_{pc} + \varepsilon_{u,i}$$

where:

$$b_i(t) = b_i + b_{i,Bin(t)} = \frac{1}{\lambda_1 + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu}) + b_{i,Bin(t)}$$

$$b_u(\lambda_2) = \frac{1}{\lambda_2 + n_u} \sum_{u=1}^{n_u} (Y_{u,i} - \hat{\mu} - b_i(\lambda_1))$$

$\lambda_1 = 2$, $\lambda_5 = 5$, the number of Bins is 6 and the number of principal components to estimate b_{pc} is the one that explained at least the 15% of the variance of the resid matrix.

4. Results

In this Section I will train the final model in the whole *edx* dataset and evaluate it's predictions on the *validation* dataset. The following code is included in a separate file "final_model_esterodr.R".

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

# First term of the item-effect with a lambda factor of 2
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+2))

# Convert timestamp in a daily variable
edx$time_day <- as.Date(as.POSIXct(edx$timestamp , origin="1970-01-01"))
validation$time_day <- as.Date(as.POSIXct(validation$timestamp , origin="1970-01-01"))

# 6 bins for the time dependant item-bias
```

```r
l1 <- min(edx$time_day)
l2 <- max(edx$time_day)
bins <- seq(l1, l2, length.out = 6)
b_it <- edx %>%
  left_join(b_i, by="movieId") %>%
  mutate(interval = findInterval(time_day, bins)) %>%
  group_by(movieId, interval) %>%
  summarise(b_it = sum(rating-mu-b_i)/(n()+2), b_i=first(b_i)) %>%
  ungroup() %>%
  mutate(b_it = b_i + b_it) %>%
  select(-b_i)
rm(b_i, l1, l2)

# User effect with a lambda factor of 5
b_u <- edx %>%
  mutate(interval = findInterval(time_day, bins)) %>%
  left_join(b_it, by=c("movieId", "interval")) %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_it - mu)/(n()+5))

# Resid Matrix for Matrix Factorization
resid <- edx %>%
  left_join(ratings_per_movie, by="movieId") %>%
  left_join(ratings_per_user, by="userId") %>%
  filter(ratings_per_movie >= 1500) %>%
  filter(ratings_per_user >= 100) %>%
  left_join(b_u, by="userId") %>%
  mutate(interval = findInterval(time_day, bins)) %>%
  left_join(b_it, by = c("movieId", "interval")) %>%
  replace_na(list(b_it=0)) %>%
  mutate(resid = rating - mu - b_it - b_u)
resid_matrix <- resid %>%
  select(userId, movieId, resid) %>%
  spread(movieId, resid) %>%
  as.matrix()
rm(resid)
rownames(resid_matrix)<- resid_matrix[,1]
resid_matrix <- resid_matrix[,-1]
movie_titles <- edx %>%
  select(movieId, title) %>%
  distinct()
colnames(resid_matrix) <- with(movie_titles, title[match(colnames(resid_matrix), movieId)])
resid_matrix[is.na(resid_matrix)] <- 0

# Estimation of the principal components
# This will take time
pca <- prcomp(resid_matrix, center = TRUE, scale = FALSE)

# Variance explained
var_explained <- cumsum(pca$sdev^2/sum(pca$sdev^2))

# Estimating the resid with the principal components
# that explains the 15% of the variance
```

```r
pc.used <- which(var_explained>=0.15)[1]
resid_pc <- pca$x[,1:pc.used] %*% t(pca$rotation[,1:pc.used])
resid_pc <- scale(resid_pc, center = -1 * pca$center, scale=FALSE)

# Convert to dataframe
pc_df <- as.data.frame(resid_pc)
pc_df$userId <- as.numeric(row.names(pc_df))
pc_df <- pc_df %>%
  gather(title, b_pc, -userId)

# Make predictions on the validation set
predicted_ratings <-
  validation %>%
  left_join(b_u, by="userId") %>%
  mutate(interval = findInterval(time_day, bins)) %>%
  left_join(b_it, by = c("movieId", "interval")) %>%
  left_join(pc_df, by=c("userId", "title")) %>%
  replace_na(list(b_it=0, b_pc=0)) %>%
  mutate(pred = mu + b_it + b_u + b_pc) %>%
  .$pred

print("The RMSE of the predictions on the validation set is:")
```

```
## [1] "The RMSE of the predictions on the validation set is:"
```

```r
RMSE(validation$rating, predicted_ratings)
```

```
## [1] 0.8347068
```

5. Conclusion

In the previous sections, I developed a model to predict movie ratings on the MovieLens dataset. The final model consists in five terms:

1. The mean rating of all the database
2. A movie-effect for each movie.
3. A time-effect for each of the former movie-effect.
4. A user-effect for each user.
5. A factor based in the matrix factorization of the portion of the user-movie matrix with higher density of data.

Using the model to predict the ratings of the *validation* set, it gives a RMSE of 0.8347068. I consider this an acceptable result. However, I know that there is room for improvement. In particular, I suggest to try the following in order to improve the predictions:

- The *lambda* factors of the item and user effect where estimated regardless the estimations of the time-bins for temporal effect in the item bias. I think that a better combinations of lambdas-bins could be find.
- I didn't use temporal factor for user bias, even thought it is reasonable to think that user may change the way they rate over time.
- The matrix factorization could be improved at least in two ways:
- With a better pick of the number of movies and users to include.
- With a more intense use of computational power, allowing the construction of a higher dimension matrix.
- I didn't use the genre of the movie. I think they could be useful to identify clusters or other patterns not covered by the matrix factorization.

And I guess there should be several additional ways to improve the model.