# Harvard PH125.9x Capstone: MovieLens Analysis

Mauro Berlanda

April 2020

## Overview

The goal of this project is to develop an algorithm to predict movie ratings.

This analysis is performed on MovieLens dataset. It contains 10 millions ratings and 100 000 tags applications applied to 10 000 movies by 72 000 users.

After downloading the entire dataset, two dataframes are created:

- `edx` which is used to perform the analysis and develop the algorithm to predict ratings
- `validation` which contains the true values of the predictions

The original datasets provided by the exercise are:

Table 1: Default summary for edx dataframe

| | Length | Class | Mode |
|---|---|---|---|
| userId | 9000055 | -none- | numeric |
| movieId | 9000055 | -none- | numeric |
| rating | 9000055 | -none- | numeric |
| timestamp | 9000055 | -none- | numeric |
| title | 9000055 | -none- | character |
| genres | 9000055 | -none- | character |

Table 2: Default summary for validation dataframe

| | Length | Class | Mode |
|---|---|---|---|
| userId | 999999 | -none- | numeric |
| movieId | 999999 | -none- | numeric |
| rating | 999999 | -none- | numeric |
| timestamp | 999999 | -none- | numeric |
| title | 999999 | -none- | character |
| genres | 999999 | -none- | character |

Table 3: Head of edx dataframe

| | userId | movieId | rating | timestamp | title | genres |
|---|---|---|---|---|---|---|
| 1 | 1 | 122 | 5 | 838985046 | Boomerang (1992) | Comedy\|Romance |
| 2 | 1 | 185 | 5 | 838983525 | Net, The (1995) | Action\|Crime\|Thriller |
| 4 | 1 | 292 | 5 | 838983421 | Outbreak (1995) | Action\|Drama\|Sci-Fi\|Thriller |
| 5 | 1 | 316 | 5 | 838983392 | Stargate (1994) | Action\|Adventure\|Sci-Fi |

| | userId | movieId | rating | timestamp | title | genres |
|---|---|---|---|---|---|---|
| 6 | 1 | 329 | 5 | 838983392 | Star Trek: Generations (1994) | Action|Adventure|Drama|Sci-Fi |
| 7 | 1 | 355 | 5 | 838984474 | Flintstones, The (1994) | Children|Comedy|Fantasy |

Table 4: Summary of edx dataframe

| userId | movieId | rating | timestamp | title | genres |
|---|---|---|---|---|---|
| Min. : 1 | Min. : 1 | Min. :0.500 | Min. :7.897e+08 | Length:9000055 | Length:9000055 |
| 1st Qu.:18124 | 1st Qu.: 648 | 1st Qu.:3.000 | 1st Qu.:9.468e+08 | Class :character | Class :character |
| Median :35738 | Median : 1834 | Median :4.000 | Median :1.035e+09 | Mode :character | Mode :character |
| Mean :35870 | Mean : 4122 | Mean :3.512 | Mean :1.033e+09 | NA | NA |
| 3rd Qu.:53607 | 3rd Qu.: 3626 | 3rd Qu.:4.000 | 3rd Qu.:1.127e+09 | NA | NA |
| Max. :71567 | Max. :65133 | Max. :5.000 | Max. :1.231e+09 | NA | NA |

The greatest challenge in analysing these data is their volume. The most of the built-in models distributed via caret package will simply run out of memory.

This dataset was already used for a test regarding date parsing in the module HarvardX: PH125.6x Data Science: Wrangling. The module HarvardX: PH125.8x Data Science: Machine Learning illustrated two important concepts using `movielens` dataset:

- Recommendation systems: use ratings that users have already given to some items in order to make specific recommandations
- Regularization: add information in order to solve an ill-posed problem or to prevent overfitting

The key steps required to identify an optimal solution are:

- prepare the data for the analysis (normalize some columns, extract new features to prepare the analysis)
- explore the content of `edx` dataset to identify how every feature relates to the outcome (ratins)
- create a train and a test set partition out of `edx` dataset to start and validate the analysis
- develop a linear model on `edx` after exploring its content and validate the predictions against the actual rating
- perform cross-validation to tune the model parameters and avoid overfiiting
  - subsetting `edx` into `train_set` and `test_set` with different seeds
  - evaluate the RMSE of the different combinations of tuning and seeds
- report the RMSE against the true values of the `validation` dataset.
- to succeed the exercise, obtain RMSE $< 0.86490$ required

The quality of the prediction is evaluated using the root mean squared error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_i (\hat{y}_i - y_i)^2}$$

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

## Analysis

The process of this analysis includes data wrangling, data exploration and the description of the modeling approach.

### Data Wrangling/Cleaning

The `edx` and `validation` are the result of some manipulation on a downloaded dataset from grouplens.org.

1. download the zip file in a temporary directory and unzip it into the workspace directory
2. read the raw data from two `.dat` files creating `ratings` and `movies` dataframes
3. normalize column names and cast value as `numeric` and `character`, then join both datasets into the `movielens` dataframe
4. partition the data in `movielens` to create a train test used for developing the model (`edx`, 90%) and the true values used to calculate the RMSE of the final model (`validation`, 10%)
5. ensure that userId and movieId in validation set are also in `edx` set

```r
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")

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

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

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

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
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, sample.kind="Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

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

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

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

Before starting the analysis, we are going to prepare the data for the analysis:

```r
wrangle_data <-function(ds) {
  ds %>%
```

```r
    mutate(
      userId = factor(userId), # int to factor
      movieId = factor(movieId), # num to factor
      datetime = as_datetime(timestamp), # parse int into date
      movieYear = str_sub(title,-5,-2), # extract movieYear from title (xxxx),
      reviewYear = factor(year(datetime)),
      reviewWeek = factor(week(datetime))
    )
}
```

In order to avoid expensive repetitive task, this manipulation is cached on the filesystem unless a reload is forced. This allows to optimize the time spent performing the next steps.

#### Data Exploration and Insights

> After a first exploration exploration of the most common ML algorithms, we realized that the `train_set` was too large to be trained on my machine. The final model implementation will provide predictions on `edx` which may be compared to the actual `edx$rating`. For this reason the data exploration will be done on `edx` and the `train_set` - `test_set` partitioning will be used only for tuning purposes.

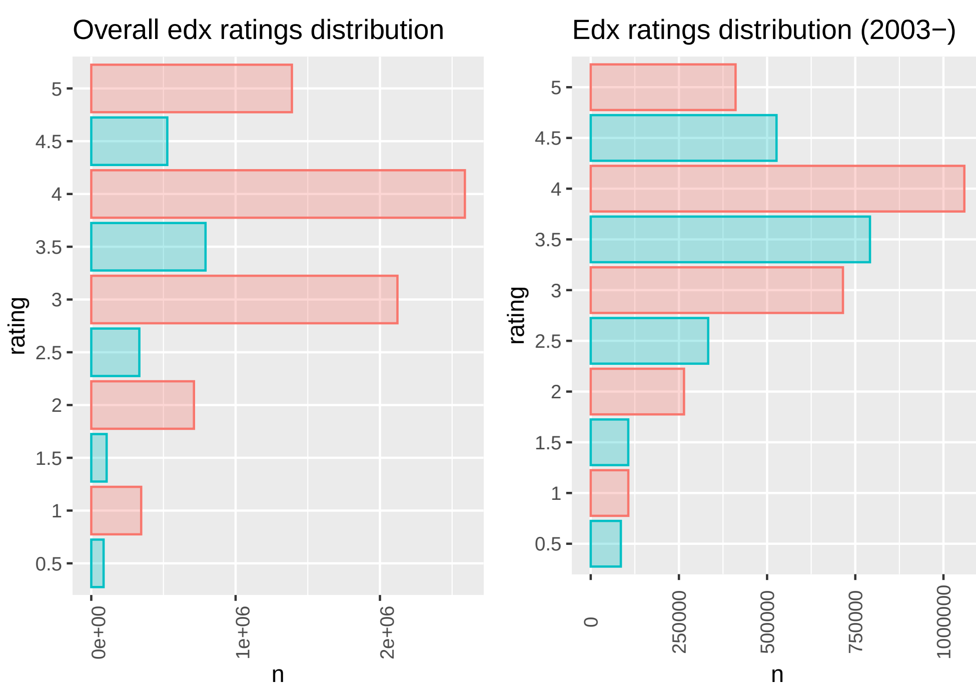
The features we may use in our analysis:

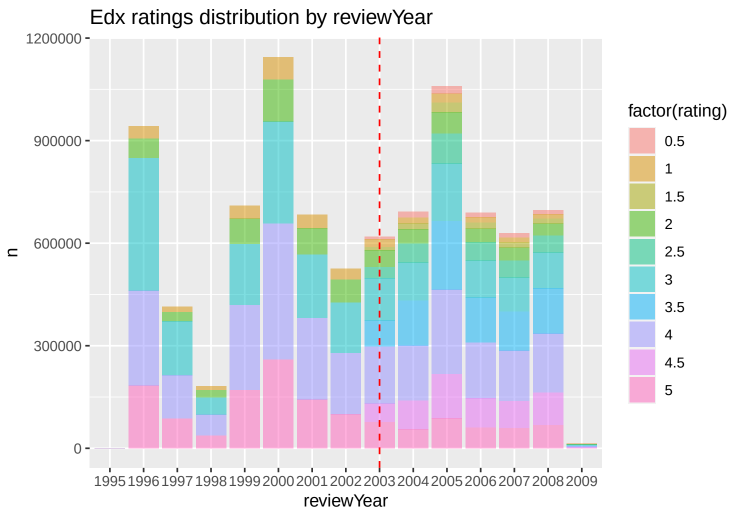
```r
setdiff(colnames(edx), c("rating", "title", "timestamp"))
```

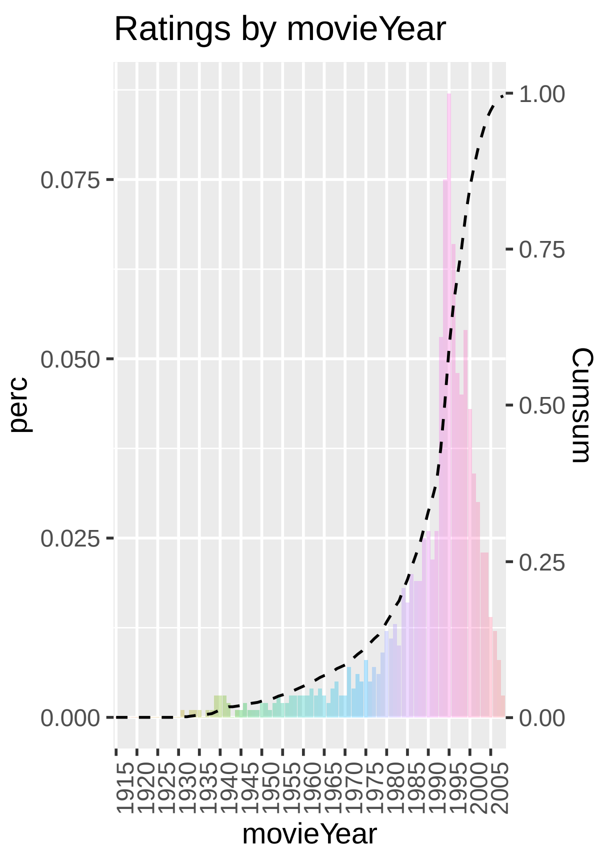
```
## [1] "userId"     "movieId"    "genres"     "datetime"   "movieYear"
## [6] "reviewYear" "reviewWeek"
```

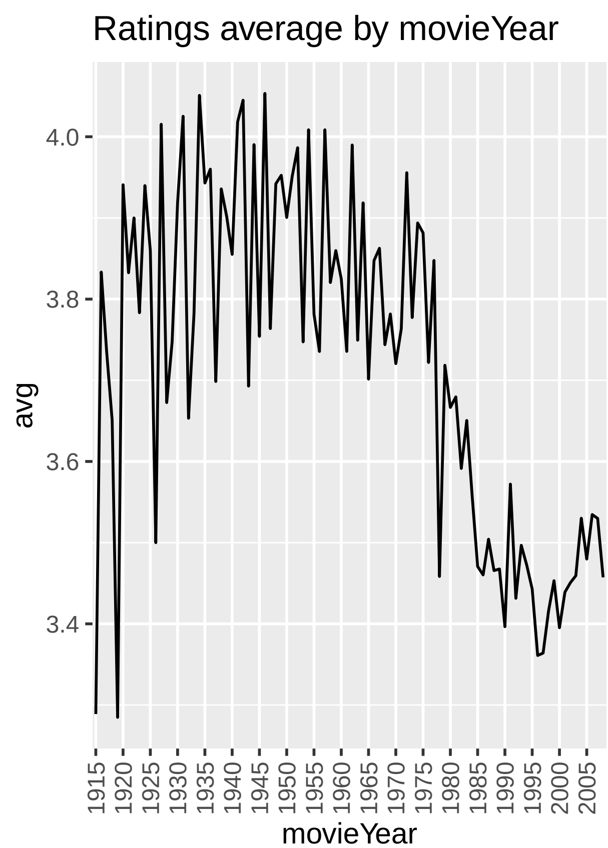
Since our goal is to predict ratings, we start looking at the overall ratings distribution in the `edx` dataset.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.000   4.000   3.512   4.000   5.000
```





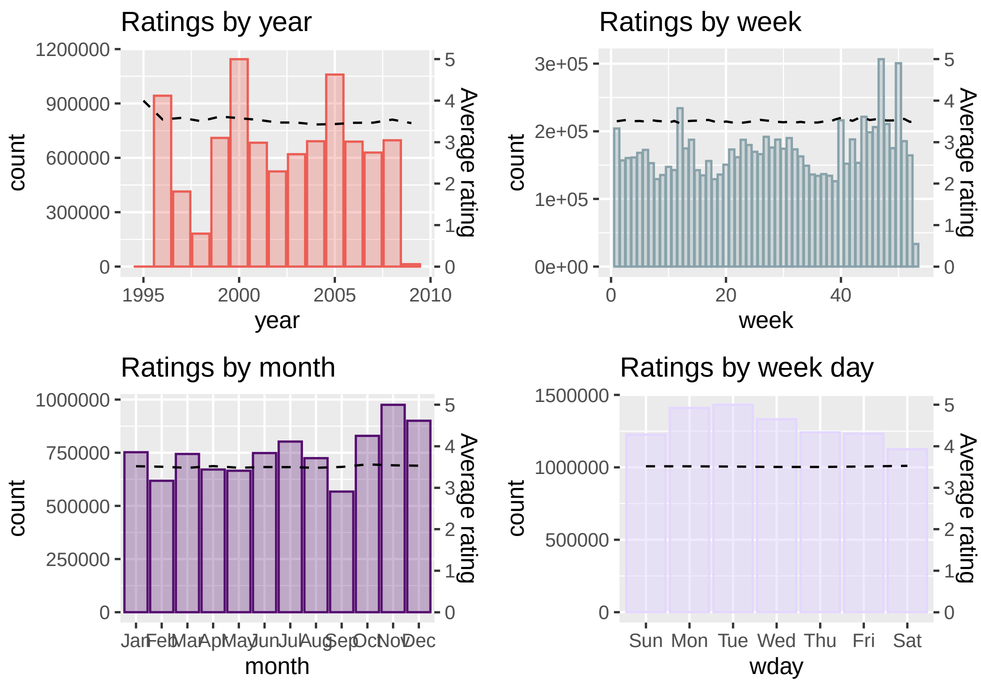




```
##   movieYear              avg
##  Length:94          Min.   :3.285
##  Class :character   1st Qu.:3.511
##  Mode  :character   Median :3.748
##                     Mean   :3.721
##                     3rd Qu.:3.900
##                     Max.   :4.053
```

We notice that whole star ratings are predominant compares to half star rating. One reason to explain the importance of this difference is that half star ratings have been introduced in 2003. We can also see that a movie is more likely to receive a rating if it was shot between the early 90s and 2000s. On average, movies before the 80s tend to have an higher rating.

Breaking down the distribution by the review `timestamp`, even if the rating count varies somehow over the time, the average rating seems quite constant.

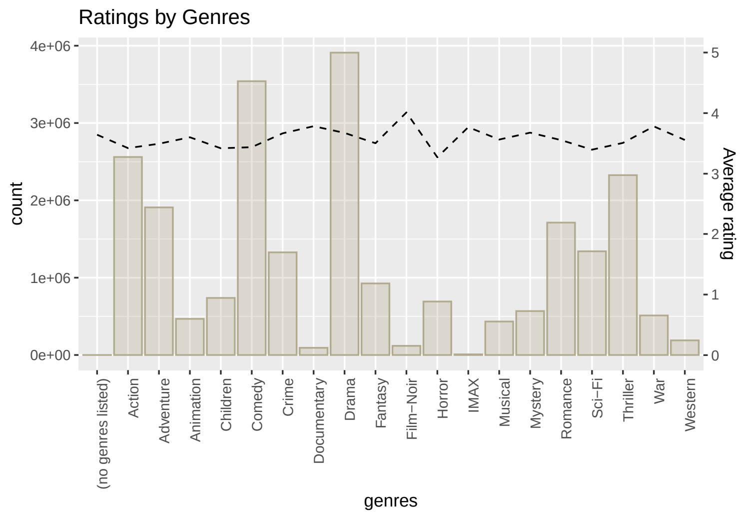



We can clearly see that on average ratings is costant across the `month` and the `wday`. There is some seasonality in the reviews distribution, with a majority of ratings given in the last quarter of the year.

The ratings breakdown by top 10 `genres` is:

Table 5: Top rated movie genres

| genres | count | avg |
|---|---|---|
| Drama | 3910127 | 3.673131 |
| Comedy | 3540930 | 3.436908 |
| Action | 2560545 | 3.421405 |
| Thriller | 2325899 | 3.507676 |
| Adventure | 1908892 | 3.493544 |
| Romance | 1712100 | 3.553813 |
| Sci-Fi | 1341183 | 3.395743 |
| Crime | 1327715 | 3.665925 |
| Fantasy | 925637 | 3.501946 |
| Children | 737994 | 3.418715 |




Comparing the distribution of rating by `movieId` and by `userId`

```
n_distinct(edx$movieId)
```

```
## [1] 10677
```

```
n_distinct(edx$userId)
```

```
## [1] 69878
```


Out of curiosity, the 10 most rated movies are:

Table 6: Top rated movies

| movieId | title | count | avg_rating |
|---|---|---|---|
| 296 | Pulp Fiction (1994) | 31362 | 4.154789 |

| movieId | title | count | avg_rating |
|---|---|---|---|
| 356 | Forrest Gump (1994) | 31079 | 4.012822 |
| 593 | Silence of the Lambs, The (1991) | 30382 | 4.204101 |
| 480 | Jurassic Park (1993) | 29360 | 3.663522 |
| 318 | Shawshank Redemption, The (1994) | 28015 | 4.455131 |
| 110 | Braveheart (1995) | 26212 | 4.081852 |
| 457 | Fugitive, The (1993) | 25998 | 4.009155 |
| 589 | Terminator 2: Judgment Day (1991) | 25984 | 3.927859 |
| 260 | Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) | 25672 | 4.221311 |
| 150 | Apollo 13 (1995) | 24284 | 3.885789 |

This data exploration has been done within the limits of my machine capacity More powerful machines can help to find out more advanced patterns.

### Modeling

In the first data exploration we could not find identify a single feature allowing to predict clearly.

As I first step we tried to solve the problem as a `classification` problem with the following outcomes:

```
seq(0.5, 5, 0.5)
```

```
##  [1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

With this result normalization, every result could have been converted to the real values levels:

```
n <- 2.7
ceiling(n*2)/2
```

In *HarvardX: PH125.8x* we have seen that if we want to handle more than two features we can use regression trees or random forrest.

Running a regression tree on all these dimensions was not really helpful:

```
# Regression tree model
rt_model <- rpart(rating ~ ., data = train_set)
plot(rt_model, margin = 0.1)
text(rt_model, cex = 0.75)

# Warning message:
# In labels.rpart(x, minlength = minlength) :
#   more than 52 levels in a predicting factor, truncated for printout
```

Other approaches like randomForrest will just run out of memory.

Models such as logistic_regression, lda, qda, loess, knn perform better with at maximum two features. We may just elect two features or use principal component analysis to define two features.

In this report we are modeling the solution taking inspiration from the approach suggested by the team winning the Netflix $1 million award in 2009.

The solution suggested was based on two models:

*a. Normalization of global effects*

- consider a baseline rating (overall mean)
- a user specific effect based on the previous ratings
- a movie specific effect based on the previous ratings
- a specific interaction (e.g. I like this movie because a given actor is playing)

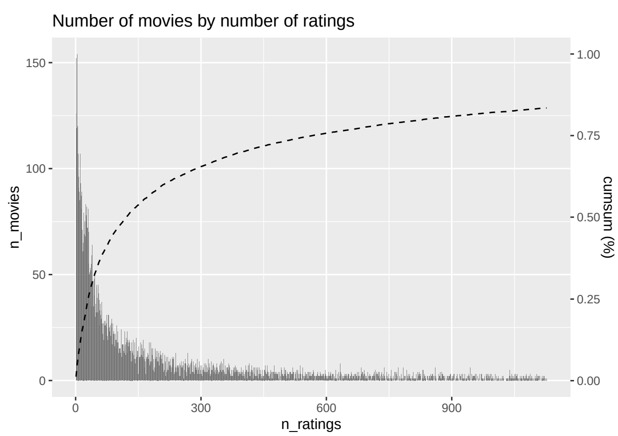
*b. Neighborhood Models*

- item-item approach: similar items may receive similar ratings
- user-user approach: similar users may give similar ratings

In our model we are going to regularize the data using *penalized least square* to constraint the total variability of the effect sizes.

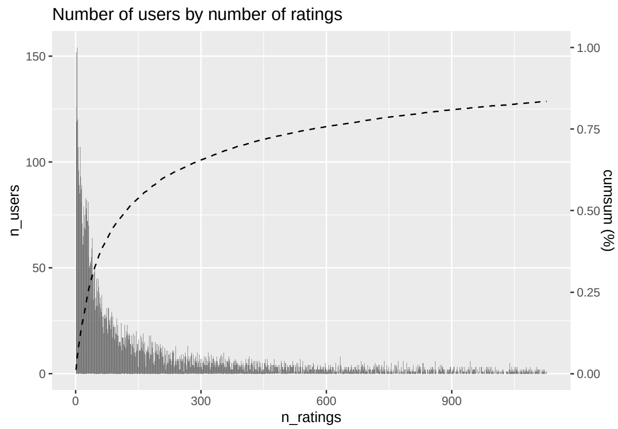
As we can see most of the items have very few ratings:

```
movies_reviews_count <- edx %>% group_by(movieId) %>% count() %>% as_tibble() %>%
  group_by(n) %>% count() %>% rename(n_ratings=n, n_movies=nn)
movies_reviews_count <- movies_reviews_count %>%
  add_column(cumsum=cumsum(movies_reviews_count$n_movies/sum(movies_reviews_count$n_movies)))

movies_reviews_count %>% head(1000) %>%
  ggplot(aes(x=n_ratings)) +
  geom_histogram(aes(y = n_movies), stat = "identity", alpha = 0.7) +
  geom_line(aes(y=cumsum * max(movies_reviews_count$n_movies)), linetype = "dashed") +
  scale_y_continuous(sec.axis = sec_axis(~./max(movies_reviews_count$n_movies), name = "cumsum (%)")) +
  ggtitle("Number of movies by number of ratings")
```




```
rm(movies_reviews_count)
```

```
users_reviews_count <- edx %>% group_by(movieId) %>% count() %>% as_tibble() %>%
  group_by(n) %>% count() %>% rename(n_ratings=n, n_users=nn)
users_reviews_count <- users_reviews_count %>%
  add_column(cumsum=cumsum(users_reviews_count$n_users/sum(users_reviews_count$n_users)))

users_reviews_count %>% head(1000) %>%
  ggplot(aes(x=n_ratings)) +
  geom_histogram(aes(y = n_users), stat = "identity", alpha = 0.7) +
  geom_line(aes(y=cumsum * max(users_reviews_count$n_users)), linetype = "dashed") +
  scale_y_continuous(sec.axis = sec_axis(~./max(users_reviews_count$n_users), name = "cumsum (%)")) +
  ggtitle("Number of users by number of ratings")
```




```r
rm(users_reviews_count)
```

In order to identify the `lambda` parameter (penalty term) to apply to our model, we are going to use the `test_set` we extracted earlier.

> It is important to note that all users and movies in `validation` are included by design in the `edx`. This allows us to stretch the impact of the section b. of the winning algorithm introduced above.

To identify an acceptable solution (RMSE $<$ 0.86490), we first define the `RMSE` function:

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

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

#### 1. naive approach: use the average

A first naive approach could be just to take the average rating and predict all the results with it:

```r
predict_model_0 <- function(df, l=0) {
  rep(mu, nrow(df))
}

# Rudimental model execution timing for benchmarks
start_time <- Sys.time()
preds_0 <- predict_model_0(edx)
model_0_rmse <- RMSE(y, preds_0)
end_time <- Sys.time()
model_0_time <- end_time - start_time
rmse_results <- tibble(method = "Model 0: Average", RMSE = model_0_rmse, accuracy = mean(preds_0==y), la

rm(preds_0, start_time, end_time, model_0_time)
```

#### 2. Introduce the movie effect

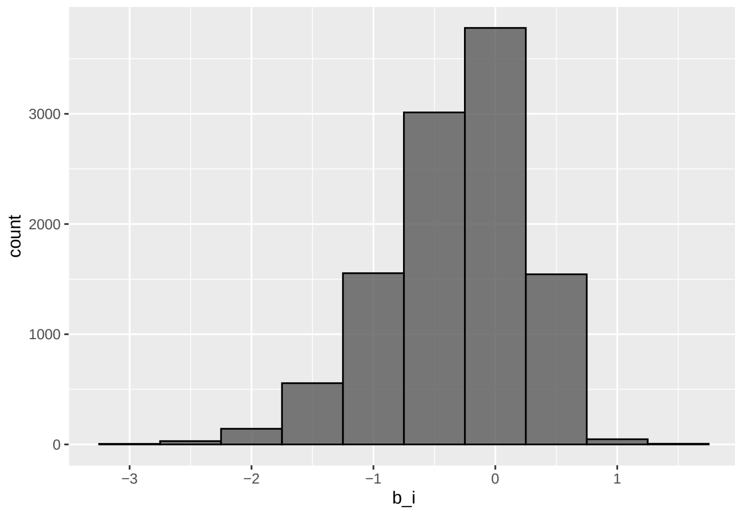
```r
# fit <- lm(rating ~ movieId, data = train_set)
# Error: vector memory exhausted (limit reached?)
```

On a regular machine is not possible to train a linear model on by `movieId`. There are two options:

- reduce the size of the `train_set`
- avoid using some vendor library and write the model by ourselves

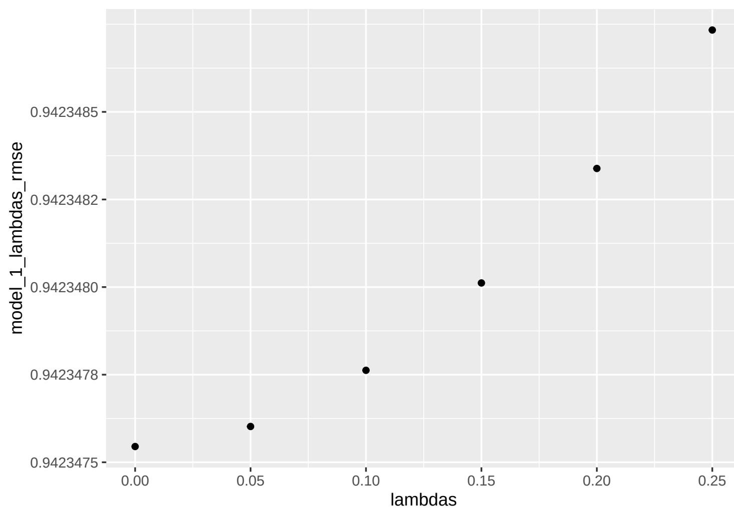
Following the second approach:

```
# regularized movie averages
l <- 0
reg_movie_avgs <- edx %>%
     group_by(movieId) %>%
     summarize(b_i = sum(rating - mu)/(n()+l))

ggplot(aes(x=b_i), data=reg_movie_avgs) + geom_histogram(bins=10, alpha=0.8, color="black")
```




```
predict_model_1 <- function(df, l=0) {
  mu <- mean(edx$rating)
  reg_movie_avgs <- edx %>%
     group_by(movieId) %>%
     summarize(b_i = sum(rating - mu)/(n()+l))

  mu + df %>% left_join(reg_movie_avgs, by='movieId') %>% .$b_i
}

# Rudimental model execution timing for benchmarks
start_time <- Sys.time()
preds_1 <-predict_model_1(edx)
model_1_rmse <- RMSE(y, preds_1)
end_time <- Sys.time()
model_1_time <- end_time - start_time

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Model 1: Movie Effect",
                                     RMSE = model_1_rmse,
                                     accuracy = mean(preds_1==y),
                                     lambda = 0,
                                     time = model_1_time
                                     ))
```

```r
# Minimize the RMSE using a penalty factor
l_min <- 0
l_max <- 5
l_step <- .25
lambdas <- seq(l_min, l_max, l_step)

model_1_lambdas_rmse <- sapply(lambdas, function(l) {
  RMSE(y, predict_model_1(edx, l))
})
l <- lambdas[which.min(model_1_lambdas_rmse)]

l_min <- max(0, l-l_step)
l_max <- l+l_step
l_step <- l_step/5
lambdas <- seq(l_min, l_max, l_step)

model_1_lambdas_rmse <- sapply(lambdas, function(l) {
  RMSE(y, predict_model_1(edx, l))
})
l <- lambdas[which.min(model_1_lambdas_rmse)]
qplot(lambdas, model_1_lambdas_rmse)
```

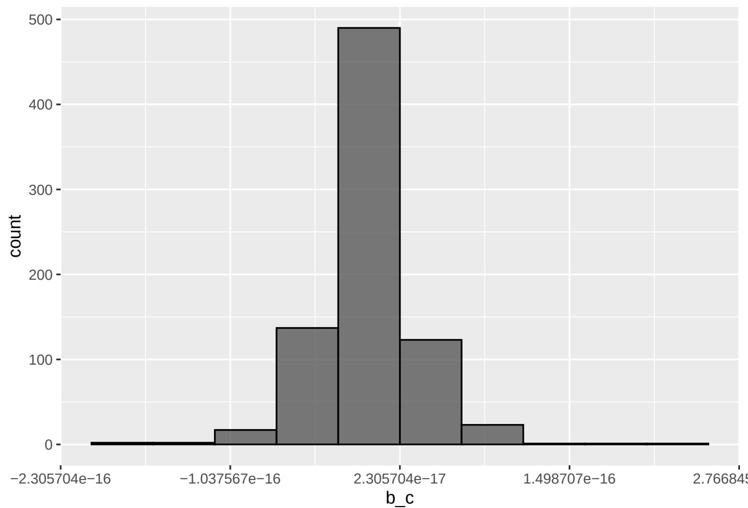



```r
if(l != 0) {
  preds_1l <- predict_model_1(edx, l)
  rmse_results <- bind_rows(rmse_results,
                            data_frame(
                              method="Model 1: Movie Effect (reg)",
                              RMSE = min(model_1_lambdas_rmse),
                              accuracy = mean(preds_1l==y),
                              lambda = l,
                              time = model_1_time
                            ))
}

rm(lambdas, l, l_min, l_max, preds_1, preds_1l, start_time, end_time, model_1_time)
```

#### 3. Introduce the genres effect -> don't

In the model b was described an item-item approach. In order to group similar movies we may used the `genres` column provided in the datasets.

```r
## regularized genres averages
reg_genres_avgs <- edx %>%
    left_join(reg_movie_avgs, by='movieId') %>%
    group_by(genres) %>%
    summarize(b_c = mean(rating - mu - b_i))

ggplot(aes(x=b_c), data=reg_genres_avgs) + geom_histogram(bins=10, alpha=0.8, color="black")
```

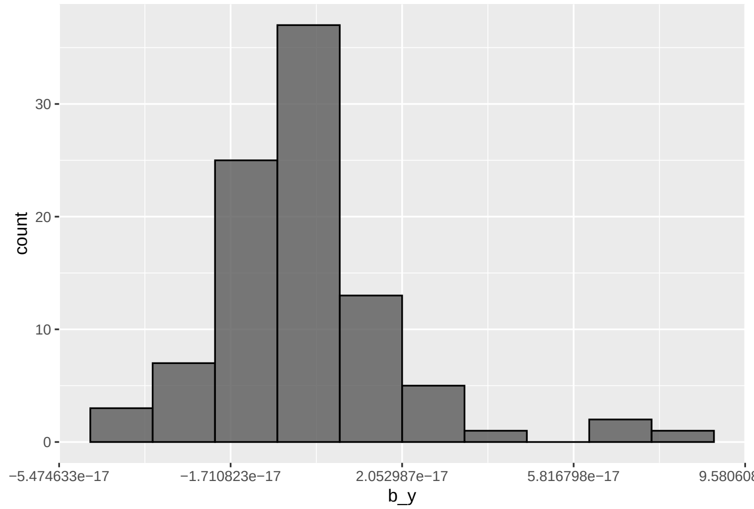



As we can see in the plot, the `genres` effect after the movie effect does not seem to be relevant (affecting the rating only by 10ˆ-17). It is not worth to include it in our model.

#### 4. Introduce the movieYear effect -> don't

Another item-item approach could be explored using `movieYear`.

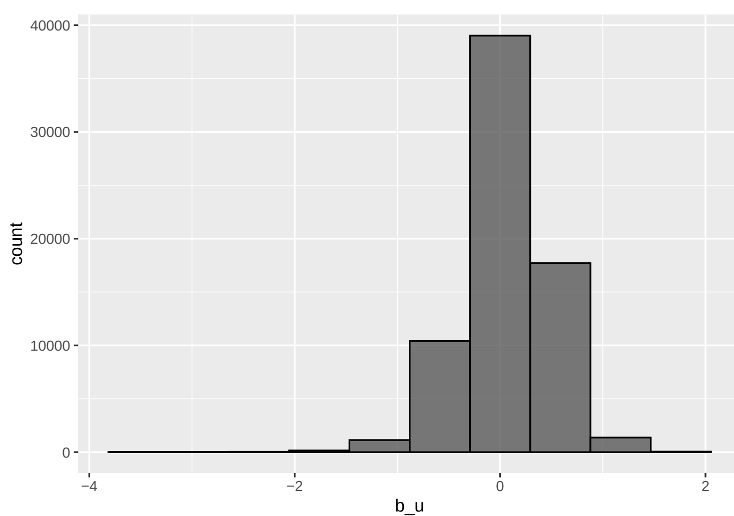
```r
## regularized movieYear averages
reg_movieYear_avgs <- edx %>%
    left_join(reg_movie_avgs, by='movieId') %>%
    group_by(movieYear) %>%
    summarize(b_y = mean(rating - mu - b_i))

ggplot(aes(x=b_y), data=reg_movieYear_avgs) + geom_histogram(bins=10, alpha=0.8, color="black")
```




As we can see in the plot, the `movieYear` effect after the movie effect does not seem to be relevant (affecting the rating only by 10^-17). It is not worth to include it in our model.

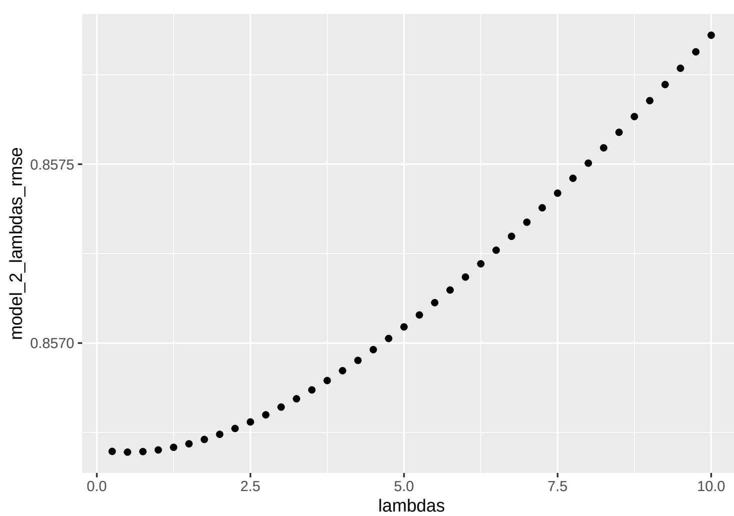
### 5. Introduce the userId effect

```
## regularized reviewYear averages
l <- 0
reg_user_avgs <- edx %>%
    left_join(reg_movie_avgs, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u = mean(rating - mu - b_i))

ggplot(aes(x=b_u), data=reg_user_avgs) + geom_histogram(bins=10, alpha=0.8, color="black")
```

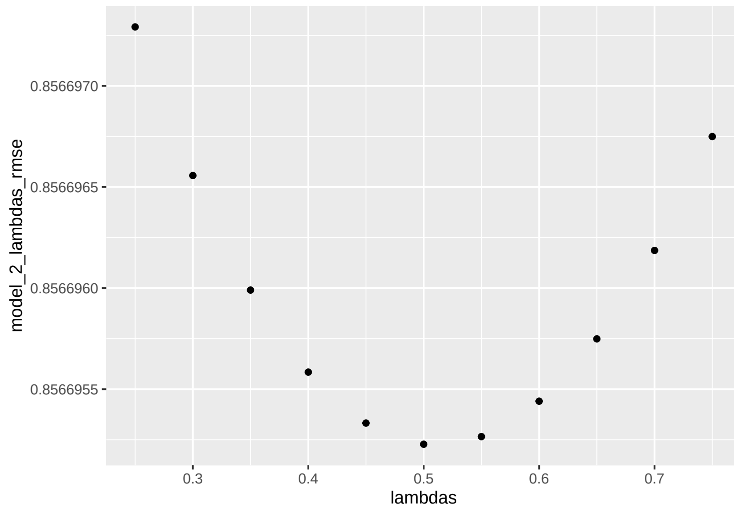



```
predict_model_2 <- function(df, l=0) {
  mu <- mean(edx$rating)

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

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

  df %>%
   left_join(reg_movie_avgs, by='movieId') %>%
   left_join(reg_user_avgs, by='userId') %>%
   mutate(pred = mu + b_i + b_u) %>% .$pred
}

# Rudimental model execution timing for benchmarks
start_time <- Sys.time()
model_2_rmse <- RMSE(y, predict_model_2(edx, 0))
end_time <- Sys.time()
model_2_time <- end_time - start_time

preds_2 <- predict_model_2(edx, 0)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Model 2: Movie + User Effect",
                                     RMSE = model_2_rmse,
                                     accuracy = mean(preds_2==y),
                                     lambda = 0,
                                     time = model_2_time
                                     ))
# Minimize the RMSE by a penalty factor
l_min <- 0.25
l_max <- 10
l_step <- .25
lambdas <- seq(l_min, l_max, l_step)

model_2_lambdas_rmse <- sapply(lambdas, function(l) {
  RMSE(y, predict_model_2(edx, l))
})
l <- lambdas[which.min(model_2_lambdas_rmse)]
qplot(lambdas, model_2_lambdas_rmse)
```



```r
l_min <- max(0, l-l_step)
l_max <- l+l_step
l_step <- l_step/5
lambdas <- seq(l_min, l_max, l_step)

model_2_lambdas_rmse <- sapply(lambdas, function(l) {
  RMSE(y, predict_model_2(edx, l))
})
l <- lambdas[which.min(model_2_lambdas_rmse)]
qplot(lambdas, model_2_lambdas_rmse)
```



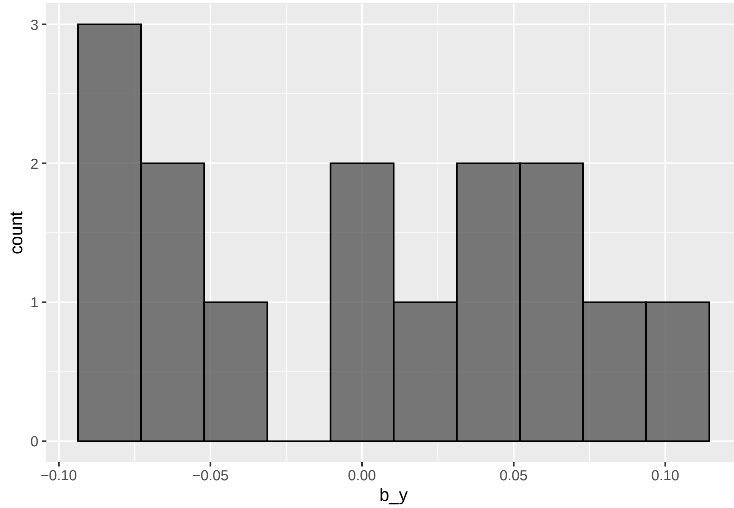

```r
if(l != 0) {
  preds_2l <- predict_model_2(edx, l)
  rmse_results <- bind_rows(rmse_results,
                            data_frame(
                              method="Model 2: Movie + User Effect (reg)",
                              RMSE = min(model_2_lambdas_rmse),
                              accuracy = mean(preds_2l==y),
                              lambda = l,
                              time = model_2_time
                            ))
}
rm(lambdas, L, l_min, l_max, preds_2, preds_2l, start_time, end_time, model_2_time)
```

### 6. Introduce the reviewYear and reviewWeek effect -> don't

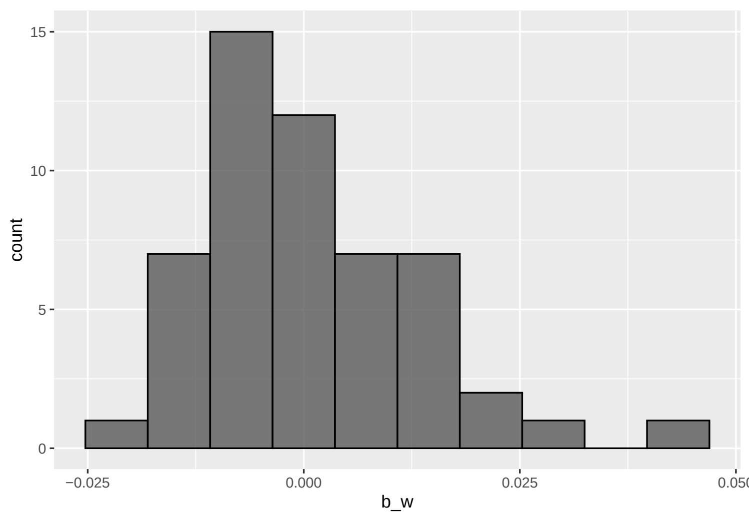
We do not have any element to evaluate a user-user approach. On of the few approaches we can imagine is that at a given moment in the time the people tend to rate in a more similar way.

```r
## regularized reviewYear averages
l <- 0
reg_reviewYear_avgs <- edx %>%
    left_join(reg_movie_avgs, by='movieId') %>%
    left_join(reg_user_avgs, by='userId') %>%
    group_by(reviewYear) %>%
    summarize(b_y = sum(rating - mu - b_i)/(n()+l))
```

```
ggplot(aes(x=b_y), data=reg_reviewYear_avgs) + geom_histogram(bins=10, alpha=0.8, color="black")
```



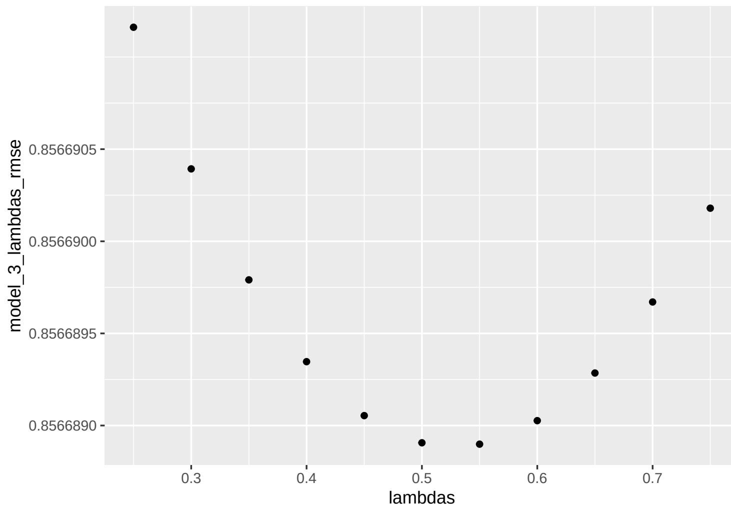

```
reg_reviewWeek_avgs <- edx %>%
    left_join(reg_movie_avgs, by='movieId') %>%
    left_join(reg_user_avgs, by='userId') %>%
    left_join(reg_reviewYear_avgs, by='reviewYear') %>%
    group_by(reviewWeek) %>%
    summarize(b_w = sum(rating - mu - b_i - b_u - b_y)/(n()+1))

ggplot(aes(x=b_w), data=reg_reviewWeek_avgs) + geom_histogram(bins=10, alpha=0.8, color="black")
```




```
predict_model_3 <- function(df, l=0) {
  mu <- mean(edx$rating)

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

  reg_user_avgs <- edx %>%
    left_join(reg_movie_avgs, by='movieId') %>%
```

```
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n()+l))

  reg_reviewYear_avgs <- edx %>%
    left_join(reg_movie_avgs, by='movieId') %>%
    left_join(reg_user_avgs, by='userId') %>%
    group_by(reviewYear) %>%
    summarize(b_y = sum(rating - mu - b_i - b_u)/(n()+l))

  reg_reviewWeek_avgs <- edx %>%
    left_join(reg_movie_avgs, by='movieId') %>%
    left_join(reg_user_avgs, by='userId') %>%
    left_join(reg_reviewYear_avgs, by='reviewYear') %>%
    group_by(reviewWeek) %>%
    summarize(b_w = sum(rating - mu - b_i - b_u - b_y)/(n()+l))

  df %>%
   left_join(reg_movie_avgs, by='movieId') %>%
   left_join(reg_user_avgs, by='userId') %>%
   left_join(reg_reviewYear_avgs, by='reviewYear') %>%
   # left_join(reg_reviewWeek_avgs, by='reviewWeek') %>%
   mutate(pred = mu + b_i + b_u + b_y) %>% .$pred
}

# Rudimental model execution timing for benchmarks
start_time <- Sys.time()
model_3_rmse <- RMSE(y, predict_model_3(edx, 0))
end_time <- Sys.time()
model_3_time <- end_time - start_time
preds_3 <- predict_model_3(edx, 0)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Model 3: Movie + User + reviewYearWeek Effect",
                                     RMSE = model_3_rmse,
                                     accuracy = mean(preds_3==y),
                                     lambda = 0,
                                     time = model_3_time
                                     ))

# Minimize the RMSE using a penalty factor
l_min <- 0
l_max <- 5
l_step <- .25
lambdas <- seq(l_min, l_max, l_step)

model_3_lambdas_rmse <- sapply(lambdas, function(l) {
  RMSE(y, predict_model_3(edx, l))
})
l <- lambdas[which.min(model_3_lambdas_rmse)]

l_min <- max(0, l-l_step)
l_max <- l+l_step
l_step <- l_step/5
```

```
lambdas <- seq(l_min, l_max, l_step)

model_3_lambdas_rmse <- sapply(lambdas, function(l) {
  RMSE(y, predict_model_3(edx, l))
})
l <- lambdas[which.min(model_3_lambdas_rmse)]
qplot(lambdas, model_3_lambdas_rmse)
```




```
if(l != 0) {
  preds_3l <- predict_model_3(edx, l)
  rmse_results <- bind_rows(rmse_results,
                            data_frame(
                              method="Model 3: Movie + User + reviewYearWeek Effect (reg)",
                              RMSE = min(model_3_lambdas_rmse),
                              accuracy = mean(preds_3l==y),
                              lambda = l,
                              time = model_3_time
                            ))
}
rm(lambdas, l, l_min, l_max, preds_3, preds_3l, start_time, end_time, model_3_time)
```

Despite the impact is moderate, taking in consideration also `reviewYear` and `reviewWeek` is slightly improving the RMSE.

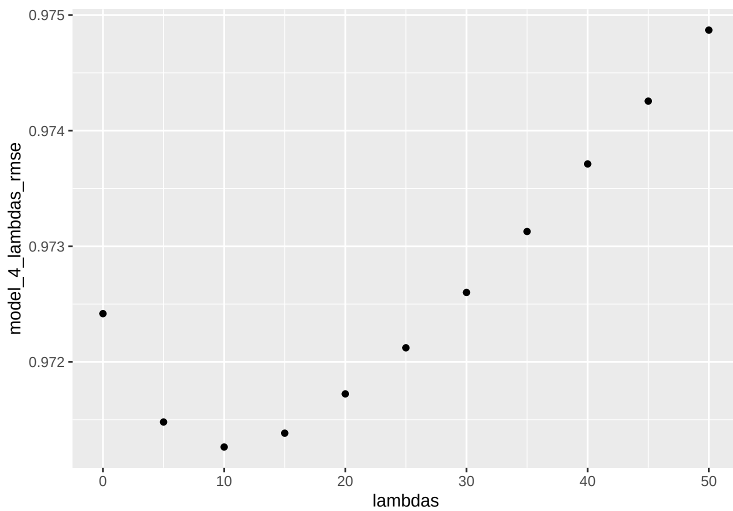
### 7. Rounding -> don't

In order to provide exact predictions to match the true values, we should round our predictions in order to match an available rating option. We have seen in the data exploration sections that half-rating ratings have been introduced in 2003.

When predicting a rating, we can use this information to make sure that the prediction will be more accurate and see the impact. Since we will already include the `reviewYear` in our mode, we may discard the manipulation at point 6.

```
## ceiling factor for rounding
uReviewYear <- sort(unique(edx$reviewYear))
ceiling_reviewYear <- tibble(
  reviewYear = uReviewYear,
```

```r
  c_i = sapply(uReviewYear, function(x) { ifelse(as.numeric(levels(uReviewYear)[x]) >= 2003, 2, 1) })
)

l <- 0
predict_model_4 <- function(df, l=0) {
  mu <- mean(edx$rating)
  reg_movie_avgs <- edx %>%
   group_by(movieId) %>%
   summarize(b_i = sum(rating - mu)/(n()+l))

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

  df %>%
   left_join(reg_movie_avgs, by='movieId') %>%
   left_join(reg_user_avgs, by='userId') %>%
   left_join(ceiling_reviewYear, by='reviewYear') %>%
   mutate(pred = ceiling((mu + b_i + b_u)*c_i)/c_i) %>% .$pred
}

# Rudimental model execution timing for benchmarks
start_time <- Sys.time()
preds_4 <- predict_model_4(edx, 0)
model_4_rmse <- RMSE(y, preds_4)
end_time <- Sys.time()
model_4_time <- end_time - start_time


rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Model 4: Movie + User Effect rounded",
                                     RMSE = model_4_rmse,
                                     accuracy = mean(preds_4==y),
                                     lambda = 0,
                                     time = model_4_time
                                     ))
# Minimize the RMSE by a penalty factor
l_min <- 0
l_max <- 50
l_step <- 5
lambdas <- seq(l_min, l_max, l_step)

model_4_lambdas_rmse <- sapply(lambdas, function(l) {
  RMSE(y, predict_model_4(edx, l))
})
l <- lambdas[which.min(model_4_lambdas_rmse)]
qplot(lambdas, model_4_lambdas_rmse)
```




```r
rm(preds_4, start_time, end_time, model_4_time)
```

As we will see in the results, the accuracy is still under 50% and the RMSE gets very bad. We may discard this option even if it was interesting to take in consideration

### 8. Cross-validation

In the first version of this report, the model was overfitting the edx set. In order to avoid this overfitting we can perform cross validation on several subset of the `edx` dataset

```r
generate_sets <- function (seed) {
  set.seed(seed, sample.kind="Rounding")
  test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
  train_set <- edx[-test_index,]
  temp <- edx[test_index,]

  test_set <- temp %>%
    semi_join(train_set, by = "movieId") %>%
    semi_join(train_set, by = "userId")

  removed <- anti_join(temp, test_set, by=c("movieId", "userId"))
  train_set <- rbind(train_set, removed)

  rm(test_index, temp, removed)
  list(train_set, test_set)
}

predict_model <- function(train, test, l=0) {
  mu <- mean(train$rating)
  reg_movie_avgs <- train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  reg_user_avgs <- train %>%
    left_join(reg_movie_avgs, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n()+l))
```

```r
  test %>%
    left_join(reg_movie_avgs, by='movieId') %>%
    left_join(reg_user_avgs, by='userId') %>%
    mutate(pred = mu + b_i + b_u) %>% .$pred
}

cross_validate <- function(seed) {
  lambdas <- seq(0,10,.25)
  sets <- generate_sets(seed)

  train_set <- sets[[1]]
  test_set <- sets[[2]]
  rm(sets)

  y <- test_set$rating

  sapply(lambdas, function(l){
    preds <- predict_model(train_set, test_set, l)
    RMSE(y, preds)
  })
}

lambdas <- seq(0,10,.25)
seeds <- seq(1, 50, 7)

rmses_matrix <- sapply(seeds, cross_validate)
colnames(rmses_matrix) <- seeds
rownames(rmses_matrix) <- lambdas
save(rmses_matrix, file = "./data/rmses_matrix.rda")
```

Let's visualize the result:

```r
load("data/rmses_matrix.rda")

colMeans(rmses_matrix)
```

```
##         1         8        15        22        29        36        43        50
## 0.8642755 0.8654374 0.8649130 0.8654868 0.8657095 0.8648669 0.8652598 0.8636925
```

```r
rowMeans(rmses_matrix)
```

```
##         0      0.25       0.5      0.75         1      1.25       1.5      1.75
## 0.8653841 0.8653069 0.8652407 0.8651825 0.8651306 0.8650843 0.8650428 0.8650058
##         2      2.25       2.5      2.75         3      3.25       3.5      3.75
## 0.8649727 0.8649435 0.8649177 0.8648952 0.8648758 0.8648592 0.8648454 0.8648341
##         4      4.25       4.5      4.75         5      5.25       5.5      5.75
## 0.8648252 0.8648187 0.8648144 0.8648121 0.8648119 0.8648135 0.8648169 0.8648220
##         6      6.25       6.5      6.75         7      7.25       7.5      7.75
## 0.8648288 0.8648372 0.8648470 0.8648583 0.8648709 0.8648849 0.8649001 0.8649165
##         8      8.25       8.5      8.75         9      9.25       9.5      9.75
## 0.8649341 0.8649527 0.8649725 0.8649932 0.8650150 0.8650376 0.8650612 0.8650857
##        10
## 0.8651109
```

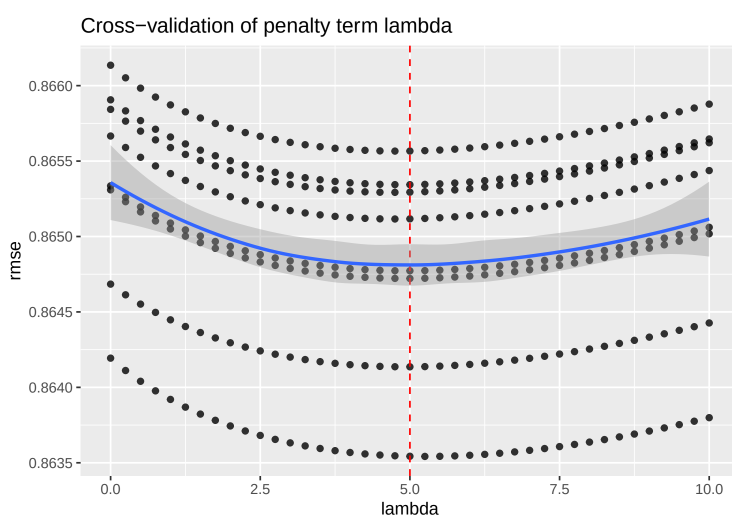
```r
lambdas <- seq(0,10,.25)

validation_lambda <- lambdas[which.min(rowMeans(rmses_matrix))]
validation_lambda
```

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

```r
library(reshape2)
```

```
##
## Attaching package: 'reshape2'

## The following objects are masked from 'package:data.table':
##
##     dcast, melt

## The following object is masked from 'package:tidyr':
##
##     smiths
```

```r
melt(rmses_matrix, value.name = "rmse") %>%
  rename(lambda = Var1, seed = Var2) %>%
  ggplot(aes(x = lambda, y=rmse)) +
  geom_point(alpha = .8) +
  geom_vline(xintercept = validation_lambda,color="red", linetype="dashed") +
  geom_smooth(formula = y ~ x, method = "loess") +
  ggtitle("Cross-validation of penalty term lambda")
```



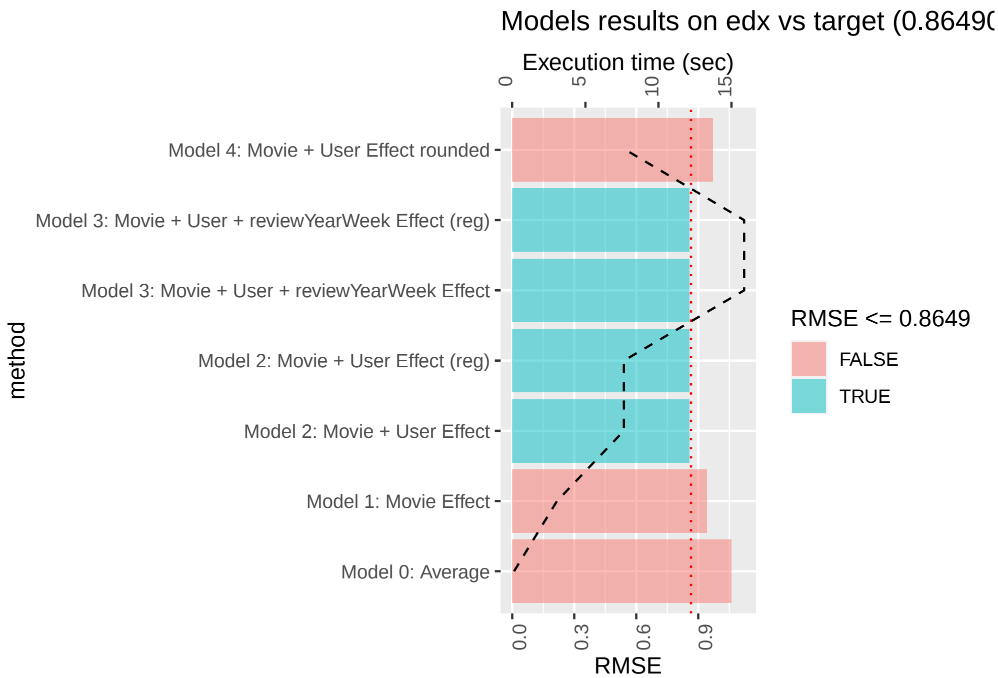

## Results

Let's see the results of the model developed on `edx`:

Table 7: RMSE on edx dataset

| method | RMSE | accuracy | lambda | time |
|---|---|---|---|---|
| Model 0: Average | 1.0603313 | 0.0000000 | 0.00 | 0.1226888 secs |
| Model 1: Movie Effect | 0.9423475 | 0.0002501 | 0.00 | 3.0636802 secs |
| Model 2: Movie + User Effect | 0.8567039 | 0.0000000 | 0.00 | 7.6394737 secs |

| method | RMSE | accuracy | lambda | time |
|---|---|---|---|---|
| Model 2: Movie + User Effect (reg) | 0.8566952 | 0.0000000 | 0.50 | 7.6394737 secs |
| Model 3: Movie + User + reviewYearWeek Effect | 0.8566979 | 0.0000000 | 0.00 | 15.8719451 secs |
| Model 3: Movie + User + reviewYearWeek Effect (reg) | 0.8566889 | 0.0000000 | 0.55 | 15.8719451 secs |
| Model 4: Movie + User Effect rounded | 0.9724173 | 0.3433422 | 0.00 | 7.7635105 secs |




We can clearly see that on the training dataset `edx` our prediction 0.8566889 was considerably lower the target *RMSE < 0.86490*.

Looking at the results, `model_3` training was very long to run compares to other models. The benefit in terms of RMSE may not justify the consumption of resources and time (10^-5 impact).

When trying to introduce the round to increase the accuracy the RMSE became insufficient and the accuracy did not exceed 34%.

Given the results obtained, we would stick on model 2 taking in consideration only movie and user effect.

Since the model tuning was overfitted for `edx` dataset, I performed some cross-validation by sampling 8 different `train_set` and `test_set`. This allowed to identify the `validation_lambda`.

The result on the `validation` dataset is

```
final_RMSE <- RMSE(validation$rating, predict_model_2(validation, validation_lambda))
final_RMSE
```

```
## [1] 0.8648177
```

> **(The final RMSE is 0.8648177.**

## Conclusion

R is a great programming language for exploring data and create model. Several tools are distributed as library and the most of machine learning algorithms are collected in the already mentionned `caret` package.

Working with Big Data can be however really challenging. The memory consumption of this dataset made the caret models impossible to train on a regular machine.

A limitation of R as programming language is the lack of flexiblity, especially when it comes to performances. For instance, I managed to reduce the time on my research by implementing some cache in `.rda` files, but I could not opt for parallelism (e.g. when performing cross-validation), use more than a CPU per process or even use my GPU.

Given these limitations, I preferred to continue building a linear model incrementally instead of training a very small percentage of the known observations dataset `edx`.

The initial results reported in the table of the previous section display some interesting result which were mainly due to a model overfitting. Choosing the penalty term after a cross validation on 8 different partitions of the dataset, removed the bias and made the model more accurate. **This allowed to obtain an optimal RMSE.**

About accuracy, an important limitation of this model is the outcome type. Since the user can only choose across a limited set of ratings (`seq(.5,5,.5)`), we can consider the problem as a categorical problem. This model is however returning a continous outcome, so even if we minimize the `RMSE`, the prediction accuracy is still close to 0. In the sub-section 7. of modeling I suggested an approach to round the prediction, but it was out of the scope of this research (and it was affecting negatively the RMSE).

As a future developement, it would be interesting if other ML frameworks allow to better distribute the computation and memory charge. If it is the case, it will be interesting to compare this result with an ensemble of the most known algorithms.