HarvardX PH125.9x Capstone - project movielens

Rainer Baumgartner

2022-08-14

Table of content

Introduction

'PH125.9x:Data Science: Capstone' is the final course in the *'HarvardX Data Science Professional Certificate'* program.

One of the graded components of this course is the *'Movielens Project'*.

The data for this project are based on the 10M version of the movieLens dataset is available here: https://grouplens.org/datasets/movielens/10m/.

It contains 10 million ratings applied to 10,000 movies by 72,000 user and was released in 1/2009.

The data are provided by MovieLens.

The aim of the *'Movielens Project'* is to develop and train a recommendation machine learning algorithm with R to predict a rating given by an user to a movie in the dataset.

The tool for the project is R. R is a free software environment for statistical computing and graphics.

To evaluate the accuracy of the algorithm, the Residual Mean Square Error (RMSE) will be used.
The target is to achieve full points for the result with an RMSE < 0.86490.

The RMSE is then defined as:

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

$y_{u,m}$ is defined as the rating for movie m by user u and $\hat{y}_{u,m}$ is the corresponding predicted rating. N is the total number of ratings.

This report will present a data-review, the data-processing, methods, analysis, results and a conclusion.

Data import

Before any review or analysis, the data needs to be downloaded, prepared and split into training- and test-data.

HarvardX provides the R-Code for the input. As the author experienced, it is essential to run the correct code refering to R-Versions *'R 4.0 or later'* versus *'R 3.6 or earlier'*. In the code below the R3.6 code is 'out-commented' by the author, to adapt to the authors installation.

```
#--------------------------------------------------------------------------

# * Data import ----------------------------------------------------------
#--------------------------------------------------------------------------

############################################################
# Create edx set, validation set (final hold-out test set)
############################################################

# Note: this process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
## Lade nötiges Paket: tidyverse
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.8     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
## Lade nötiges Paket: caret
## Lade nötiges Paket: lattice
##
## Attache Paket: 'caret'
##
## Das folgende Objekt ist maskiert 'package:purrr':
##
##     lift
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
## Lade nötiges Paket: data.table
##
## Attache Paket: 'data.table'
##
## Die folgenden Objekte sind maskiert von 'package:dplyr':
##
##     between, first, last
##
## Das folgende Objekt ist maskiert 'package:purrr':
##
##     transpose

library(tidyverse)
library(caret)
library(data.table)
```

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

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

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

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

# if using R 3.6 or earlier:
#movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           # title = as.character(title),
                                            #genres = as.character(genres))
# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))


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

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

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

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
edx <- rbind(edx, removed)

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

Load additional packages

R has many built-in base functions. R packages provide additional functions for certain purposes, like improved graphics or algorithms. Some packages where already installed and loaded with the code provided by HarvardX for the data import.

The following code installs (if necessary) and loads other packages used in this project.

```
#------------------------------------------------------------------------------
```

```
# * Load additional packages --------------------------------------------
#_------------------------------------------------------------------------------


  # define required packages
  packs <- c("lubridate", "ggthemes", "knitr", "gridExtra", "scales", "pryr", "tinytex" )

  # install packages (if needed)  and load them
  for (package in packs) {
    if (!require(package, character.only=T, quietly=T)) {
      install.packages(package, repos = "http://cran.us.r-project.org")
      library(package, character.only=T)
    }
  }
```

A quick view on the movielens-data

After the import the data-set *'edx'* represents the training-set whereas *'validation'* serves as very final test-set. Both sets have the same structure.

With str()-function we can check the data-structure:

```
#_____________________________________________________________________________

# ** Check data structure ---------------------------------------------------
#_____________________________________________________________________________

  # check the data-structure edx
    str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
##  - attr(*, ".internal.selfref")=<externalptr>
```

Comments to the variables:

- userId is a unique identification number for each user giving ratings.
- movieId is a unique identification number for each movie.
- rating is a number for the rating of one movie by one user, ranging from 0.5 to 5.0 in steps of 0.5. It is defined as 0.5 = worst, 5.0 = best.
- timestamp indicates when the movie was rated. The figure represents seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.
- title is the name of the movie containing the year of the release. The title has a 1:1 relation to the movieId.
- genres are the piped genre categories assigned to the movie.
- `-attr(*,...` in the last row is not part of the analysed data

With the head()-function we can see some data examples and use kable to make it look nice:

```
#_____________________________________________________________________________

# ** Data examples ------------------------------------------------
#_____________________________________________________________________________

  # Data examples
    knitr::kable(head(edx))
```

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

Further data preparation

In order to optimize the data-structure and extract further information from the existing variables, this adaptions will be applied:

- add variable 'ts_date' with converted timestamp as POSIX date.time

```
#_____________________________________________________________________________

# * Further data preparation -------------------------------------------
#_____________________________________________________________________________

#_____________________________________________________________________________

# ** add variable 'ts_date' ---------------------------------------------
#_____________________________________________________________________________

    edx$ts_date <- date(as.POSIXct(edx$timestamp, origin="1970-01-01"))
```

- extract the year when the movie was released (from the digits at the end of the movie-title) as a new variable

```
#_____________________________________________________________________________

# ** add movie year -----------------------------------------------------
#_____________________________________________________________________________
      edx$year <- edx$title %>% str_sub(-5,-2) %>%  as.numeric()
```

- convert the genre from text to factor, for improved memory usage and performance

```
#_____________________________________________________________________________

# ** convert genre to factor --------------------------------------------
#_____________________________________________________________________________
      edx$genres <- as.factor(edx$genres)
```

- add a new variables 'rating-lag' and 'year.rating', indicating the lag between 'movie was released [year]' and 'movie was rated [year.rating]'. Remark: rating lags <0 will be converted to 0 in order to avoid potential problems with negative values. The investigation on the impact will be part of the data exploration.

```
#_____________________________________________________________________________

# ** add variables year.rating & rating.lag ------------------------------
#_____________________________________________________________________________

      edx <- edx %>% mutate ( year.rating = as.numeric(isoyear(ts_date)),
                              rating_lag =  year.rating - year,
                              rating_lag = ifelse (rating_lag<0,0,rating_lag))
```

Movielens data exploration and visualization

Rating

The target of the project is to predict a rating, thus let's first look at the rating itself. As already mentioned before, rating is a number ranging from 0.5 to 5.0 in steps of 0.5. It is defined as 0.5 = worst, 5.0 = best.

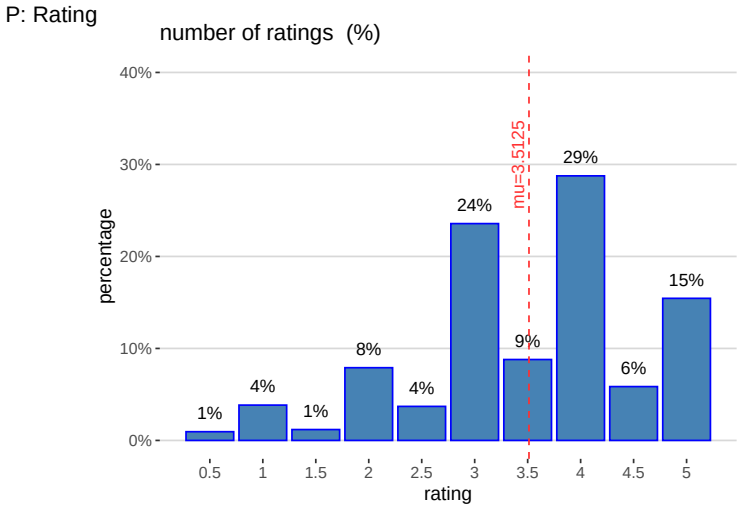
The total number of ratings is 9000055 and equals the length of the edx dataset.

Summing up all ratings (0.5...5) and divide the sum by the total number of ratings, will give us the average rating. Throughout the project we will name this variable as **'mu'** for the greek letter 'μ'.

We calculate mu for further use:

```
#_____________________________________________________________________________

# ** calculate mu ------------------------------------------------------------
#_____________________________________________________________________________

  mu <- mean(edx$rating)  #calculate the average rating
```

The bar plot shows the distribution (in %) for each rating and the calculated mu:

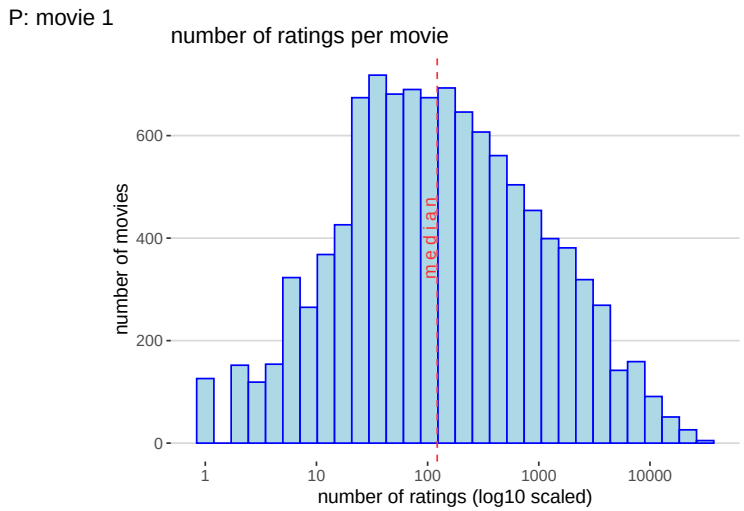


We observe a left-skewed distribution and under-represented 'half-value' ratings.

Movies

The data-set edx contains 10677 movies.

This histogram shows the number of ratings per movie:



Most movies are rated 1 - ca. 150 times (the median is 122), but some movies are extremly often rated.

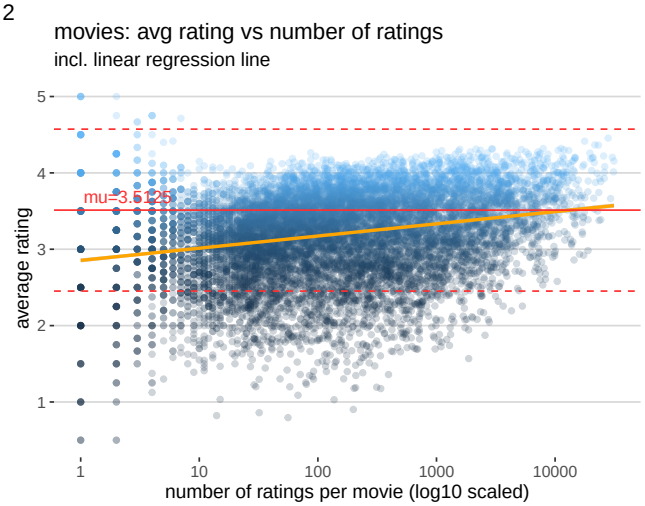
Let's have a look on the 10 most rated movies:

Table 2: movies with most ratings

movieId	title	n_ratings	avg_rating
296	Pulp Fiction (1994)	31362	4.154789
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

Is there any influence of the number of ratings on the given rating?

P: movie 2

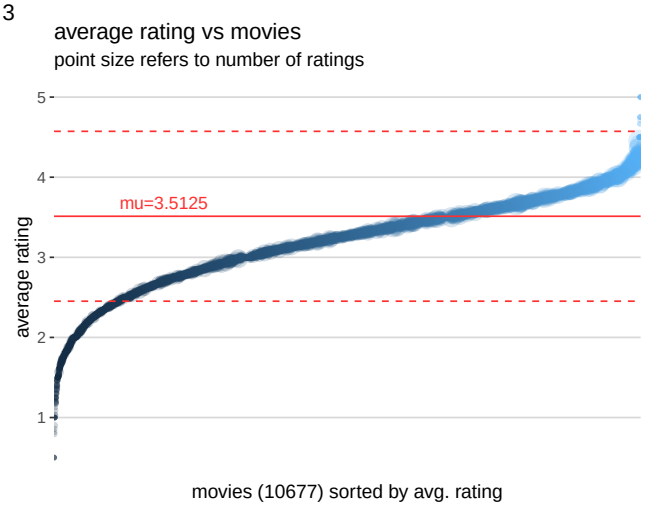


There seems to be a slight positive correlation. We can see only a few ratings <=1 from around 300 number of ratings on wards. On the other hand, all movies rated by >04.5 have less then 10 ratings.

General information: The dashed lines, 1.0603 ratings above and below 'mu', represent the 1 σ (or standard deviation) borders.

The rating is what we want to predict, thus we check the influence of the movie itself on the given ratings (movieId ordered by ascending rating):

P: movie 3



The variance of the data is clearly visible.

Here are the 10 best / worst rated movies:

Table 3: Movies with best ratings

movieId	title	n_ratings	avg_rating
3226	Hellhounds on My Trail (1999)	1	5.00
33264	Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)	2	5.00
42783	Shadows of Forgotten Ancestors (1964)	1	5.00
51209	Fighting Elegy (Kenka erejii) (1966)	1	5.00
53355	Sun Alley (Sonnenallee) (1999)	1	5.00
64275	Blue Light, The (Das Blaue Licht) (1932)	1	5.00
5194	Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	4	4.75
26048	Human Condition II, The (Ningen no joken II) (1959)	4	4.75
26073	Human Condition III, The (Ningen no joken III) (1961)	4	4.75
65001	Constantine's Sword (2007)	2	4.75

Table 4: Movies with worst ratings

movieId	title	n_ratings	avg_rating
5805	Besotted (2001)	2	0.5000000
8394	Hi-Line, The (1999)	1	0.5000000
61768	Accused (Anklaget) (2005)	1	0.5000000
63828	Confessions of a Superhero (2007)	1	0.5000000
64999	War of the Worlds 2: The Next Wave (2008)	2	0.5000000
8859	SuperBabies: Baby Geniuses 2 (2004)	56	0.7946429
7282	Hip Hop Witch, Da (2000)	14	0.8214286
61348	Disaster Movie (2008)	32	0.8593750
6483	From Justin to Kelly (2003)	199	0.9020101
604	Criminals (1996)	2	1.0000000

```
##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells   2692154 143.8    7808958  417.1  22391674 1195.9
## Vcells 106421827 812.0  320819489 2447.7 317553514 2422.8
```

Note that these best/worst ratings have very view reviews by users (n_ratings). We already noticed similar observations in the plot 'P: movie 2'. Additionally, at least from the authors pov, the movies are not very famous.

Is it correct/fair to say 'Hellhounds on My Trail' (rated by one user with 5) is a better movie than (the most rated movie with 31362 ratings) 'Pulp Fiction' with an average rating of 4.15? We can not give the answer here, but we will consider this when we apply regularization (see chapter 'Methods and analysis').

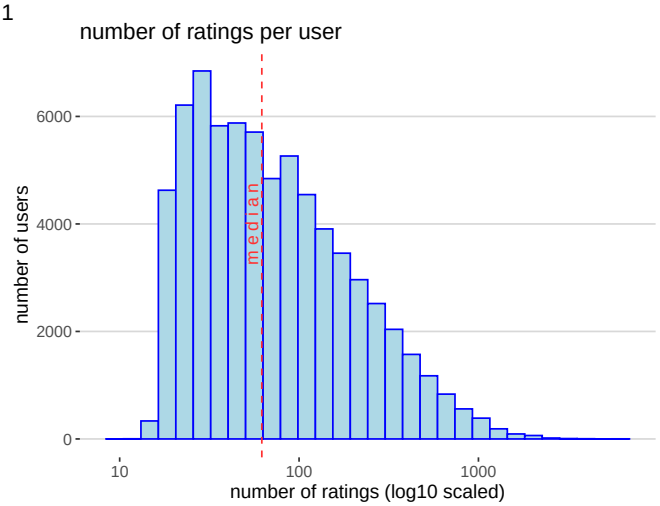
User

In edx we find 69878 users giving ratings.

The maximal number of ratings by an user is 6616.

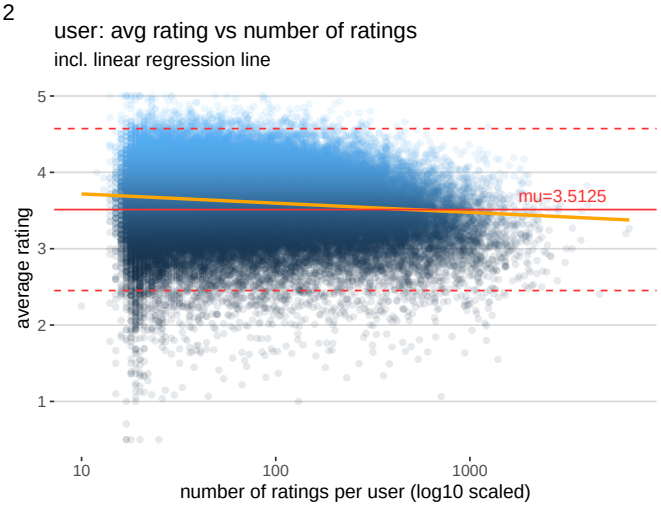
We can see the distribution of ratings in the histogram.

P: user 1



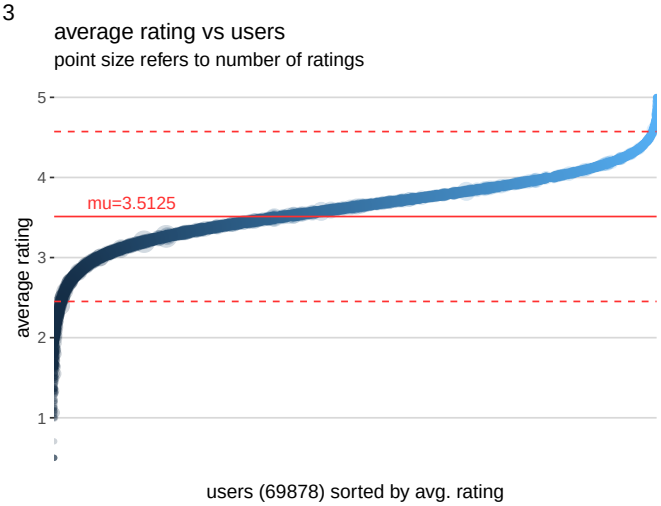
The distribution is right skewed (especially when considering the log10 scales axis).

P: user 2



Here is no huge trend visible, but the variability decreases with the number of users.

P: user 3

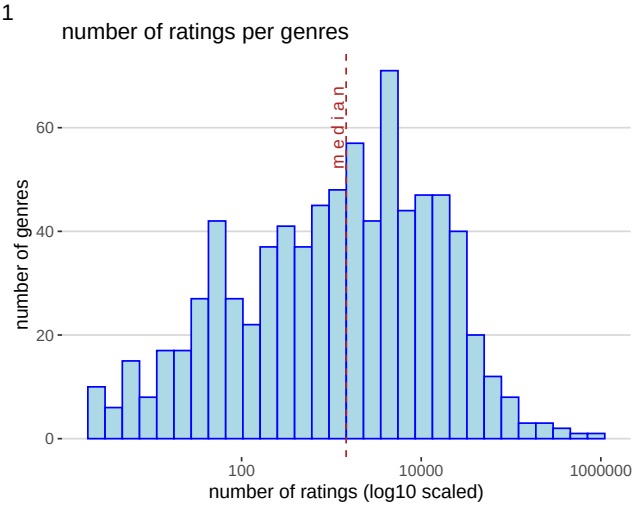


Apart of the very extremes (low/high rating), the values are quite close to mu. A bit more moderate variability compared to movies.

Genres

As already mentioned, the genres are piped: A movie can be assigned to several genres, the genres are separated by '|' . There are 797 piped genres, i.e. different combinations of individual genres.

P: genres 1



Lets have a glimpse on some extreme genres:

Table 5: Genres with most movies

genres	n.movie
Drama	733296
Comedy	700889
Comedy|Romance	365468
Comedy|Drama	323637
Comedy|Drama|Romance	261425
Drama|Romance	259355

Table 6: Genres with least movies

genres	n.movie
Adventure|Fantasy|Film-Noir|Mystery|Sci-Fi	2
Adventure|Mystery	2
Crime|Drama|Horror|Sci-Fi	2
Documentary|Romance	2
Drama|Horror|Mystery|Sci-Fi|Thriller	2
Fantasy|Mystery|Sci-Fi|War	2

Table 7: Genres with best ratings

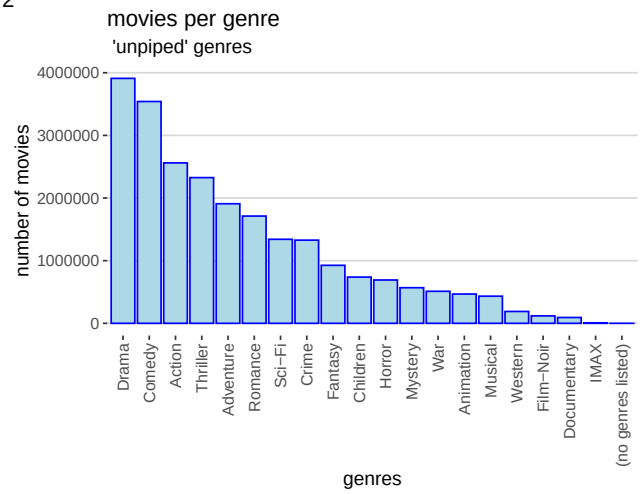
genres	avg_rating
Animation|IMAX|Sci-Fi	4.714286
Drama|Film-Noir|Romance	4.304115
Action|Crime|Drama|IMAX	4.297068
Animation|Children|Comedy|Crime	4.275429
Film-Noir|Mystery	4.239479
Crime|Film-Noir|Mystery	4.216803

Table 8: Genres with worst ratings

genres	avg_rating
Adventure|Drama|Horror|Sci-Fi|Thriller	1.751152
Action|Drama|Horror|Sci-Fi	1.750000
Comedy|Film-Noir|Thriller	1.642857
Action|Horror|Mystery|Thriller	1.607034
Action|Animation|Comedy|Horror	1.500000
Documentary|Horror	1.449112

For all the individual (not piped) genres, the plot presents the number of movies per 'unpiped' genre:

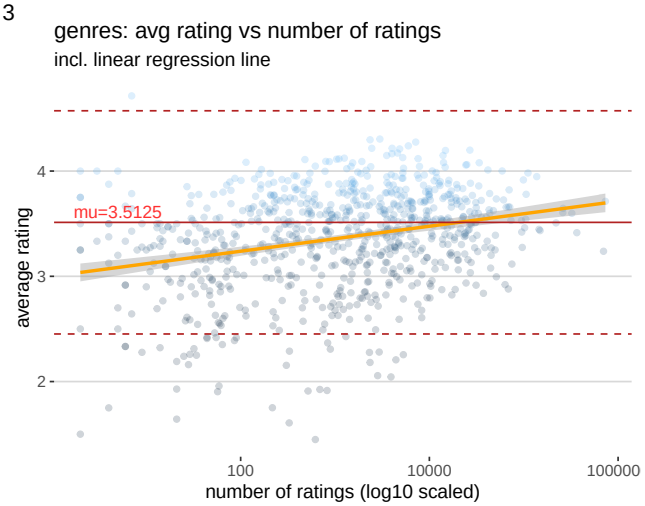
P: Genre 2




```
##             used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells   2732586 146.0   63929164 3414.2  79911454 4267.8
## Vcells 106480991 812.4  320819489 2447.7 320819480 2447.7
```

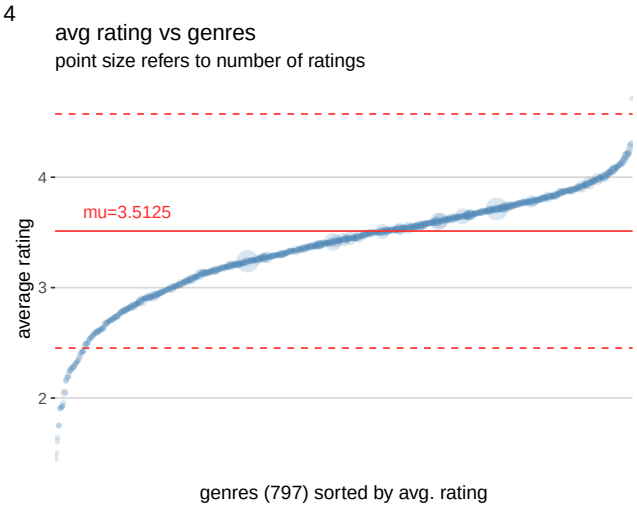
Also for genres we check the average rating against number of ratings:

P: genres 3



The data are very scattered, not showing much of pattern.

P: genres 4

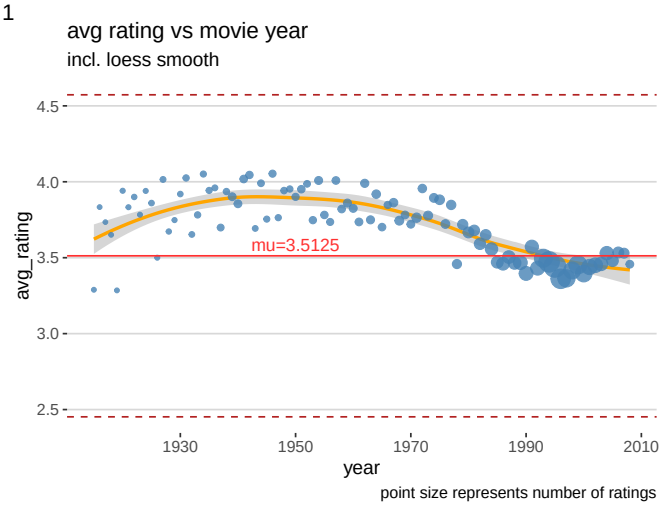


The curve looks similar to the movies, but shows slightly less variability (looking at the bends and the very left/right values).

Movie year

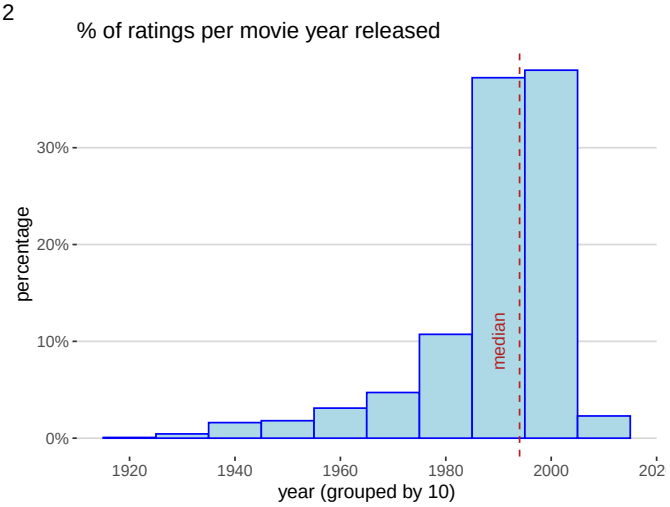
The first temporal variable we explore and visualize is the year a movie was released.

P: year 1



Movies rated before the 90's get mostly ratings quite above the average whereas later the values are slightly below or around the average.

P: year 2

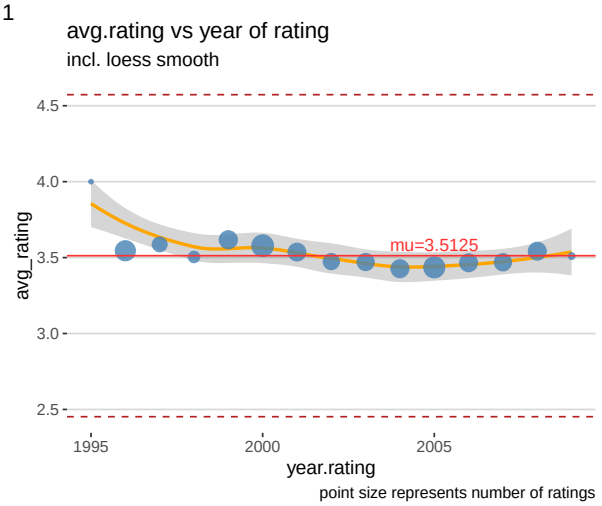


There are no movies beyond 2009 as the movielens data-set was released in 1/2009. In the data-set we find only very few movies up the 80's.

Year of rating

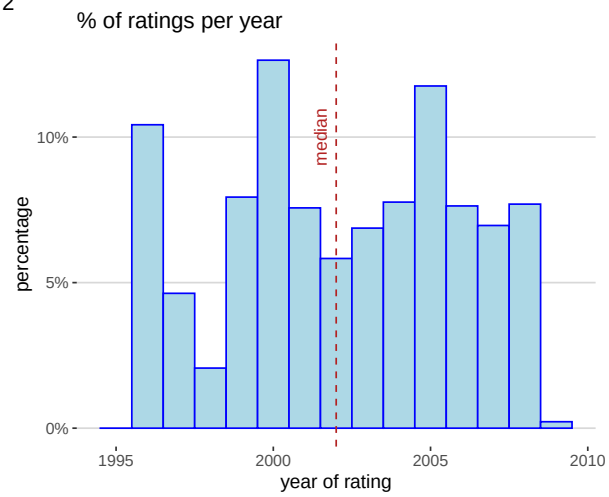
The year of rating indicates, derived from the timestamp, in which year a rating was given. Is there a trend towards higher or lower rating? Are ratings in certain years very extrem?

P: year.rating 1



The year of rating is quite close and stable to mu. Hence not much variability, except the outlier in 1995.

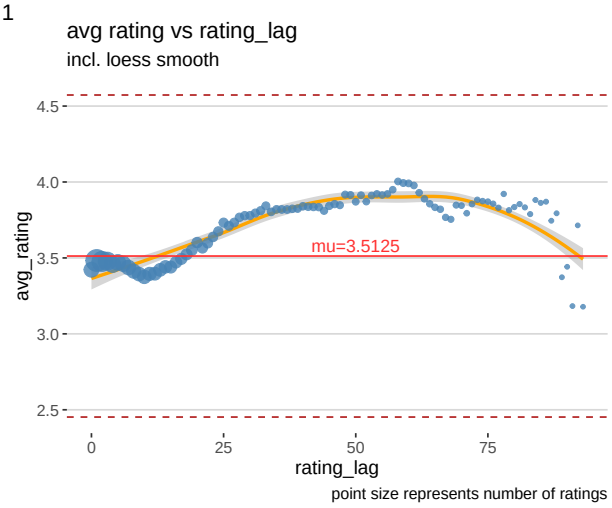
P: year.rating 2



Also the barplot reflects a low variability, despite some outliers.

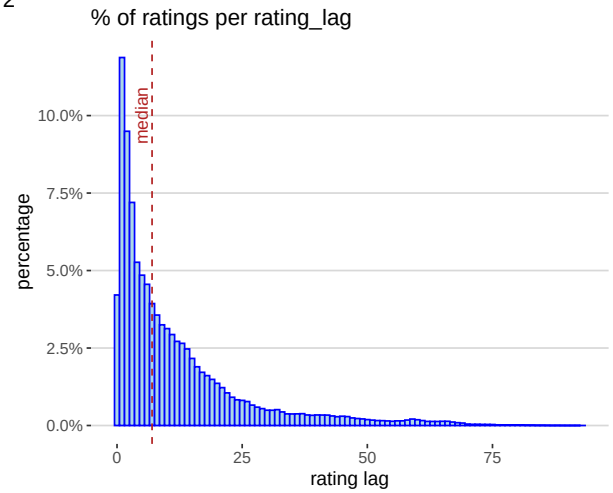
Rating lag

P: rating lag 1



The rating lag shows some variability. It seems to be similar to the movie year, but 'mirrored'.

P: rating lag 2



The most movies are rated 9 year after their release.

Methods and analysis

Regularization

Regularization allows for reduced errors caused by movies with extreme ratings and few reviews (i.e. number of ratings)

The general idea is to add a penalty for extreme values with low occurrence in the data, in order to optimize the RMSE.

With reference to table 'Movies with best ratings' and the comment afterwards: We give no evaluation on 'Hellhounds on My Trail' vs. 'Pulp Fiction', but consider the number of ratings as basis for the regularization in order to achieve the best overall RMSE.

Varying from the course (one lambda for all variables), the regularization will be first done by individual lambdas for each variable.
The idea is, to analyse if an individually optimized lambda for each variable leads to a better overall performance.
For comparison, an optimized model with lambda 'one4all' will be generated. The comparison will be done only on the models with movie/user/genres effects.

Variables to be used

Based on the data exploration, the following variables where be taken into account:

- movie
- user
- genres
- rating.lag
- year

In the final model only one temporal effect shall be used. Thus after the decision on the regularization mode, a RMSE-based decision will be taken for the better performing temporal effect.

Prediction model

The method will follow the regression model approach of 'Introduction to Data Science', chapter 34.7 by Irizarry, R. (2022-07-07).

Starting from a simple model based on mu, the model will be enhanced and optimized by combining several effects applying regularization. Each model will be compared to the others by it's RMSE.

The process towards the final prediction model is planned as follows:

1. Simple model with mu only
2. Base + movie effect
3. Base + movie effect regularized individually
4. Base + movie + user effect regularized individually
5. Base + movie + user + genre effect regularized individually
6. Base + movie + user + genre effect regularized one-for-all
7. Compare regularization individually versus one-for-all
8. Base + movie + user + genre + rating lag effect (regularization based on comparison)
9. Base + movie + user + user/genre + year effect (regularization based on comparison)
10. Review results and decide on final model
11. Final evaluation

Crossvalidation

Crossvalidation is used throughout the models, even when no parameter optimization is required. This is in order to have just one process for all. The code is inspired by the material for 'HardvardX course PH525X Biomediacal Data Science', also held by our Professor Rafael Irizarry.

System performance and memory

Already when starting to download the data and preparing the edx/validation data-sets, I encountered 'out-of-memory'-issues in RStudioCloud. Thus I switched to my local RStudio installation (16GB RAM).

Nevertheless once in a while r-functions might be performed within the code:

- rm(): Can be applied if usefull, to (large) objects that are obsolete for the further processing or analysis.
- gc(): Stands for 'garbage collection' and cleans up unused objects (also 'invisible' ones) and thus free up memory.

As additional measure (if needed): Variables in edx not needed for the models, could be removed before starting modeling from edx.

In order to keep track of runtimes within the model functions, for each iteration (CV, lambda) the runtime in seconds is stored in the result as 'duration'.

Building the prediction models

SETUP: Crossvalidation folds & RMSE function

As a first step, the folds for the crossvalidation needs to be created, using the createFolds function from the caret package. Also the KPI-measuring function for the RMSE is defined.

```
#_____________________________________________________________________________

# * Building the prediction models ------------------------------------
#_____________________________________________________________________________

#_____________________________________________________________________________

# ** setup cv  & define RMSE ------------------------------------------
#_____________________________________________________________________________


# set.seed in order to create reproducible results.
  set.seed(123, sample.kind="Rounding")

# Create 5  folds for use in crossvalidation from edx (createFolds from caret)
  folds <- createFolds(as.numeric(rownames(edx)), k = 5)

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

M1 Base model: Just the average (mu only)

Our simplest model is just to predict the overall average of the rating.

In the code we call the model function, split into train & test sets according to the 5 folds , calculate the prediction and finally the RMSE.

The called function gives back a dataframe, containing the results for each CV fold (and in later models also the parameters (lambda) to be optimized).

Based to the results from the function, a data-frame 'results' will be created. It contains a summary for the model, will be enhanced for all further models and is the basis for the performance evaluation.

Content of results:

- Method: Refers to the model and its variables (reg. ~ regularization,)
- RMSE: The RMSE for this model
- reference is some information e.g. about C.V. and the data used (e.g. edx)
- lambda contains the optimized lambda's for the model (if available)
- duration is the summed up runtime for the model in seconds (if needed)

```
#------------------------------------------------------------------------------

# ** m1 Base model: just the average ----------------------------------
#------------------------------------------------------------------------------

## model function
  m1.simply.mu <- function (x) {

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    start <- as.numeric(Sys.time())

    train <- edx[-folds[[x]],]    #define train set
    test <-  edx[folds[[x]],]     #define test set
    mu_hat <- mean(train$rating)  #mu_hat as prediction as avg. of all ratings

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    rmse <- RMSE(test$rating, mu_hat)   # calculate rmse
    end <- as.numeric(Sys.time())
    runtime <- round(end-start)

    result <- data.frame(method="simply.mu",
                         fold=x,
                         rmse=rmse,
                         lambda='na',
                         duration=runtime )  #store result in a data.frame

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>
  }

## call model function
```

```
  # apply for folds 1:5 the function 'simply_mu'
  result.m1.average <- map_df(1:5, m1.simply.mu)

## save model result to overall results dataframe
  results <- data.frame(Method="m1 simply the average",
                RMSE=mean(result.m1.average$rmse),
                reference = "edx CV 5f",
                lambda = 'na',
                duration = sum(result.m1.average$duration))
```

The result of our first model:

Method	RMSE	reference	lambda	duration
m1 simply the average	1.060331	edx CV 5f	na	1

M2 Base + movie effect

Now we add the movie effect. The effect indicates, how much the rating for a certain movie differs from the overall average. This bias we assign to the variable 'b.m'.

This scheme applies also to all further models with more effects resulting in more 'b's'.

```
#_____________________________________________________________________________

# ** m2 Movie effect ---------------------------------------------------------
#_____________________________________________________________________________

## model function
  m2.movie.effect <- function (x) {

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    start <- as.numeric(Sys.time())

    train <- edx[-folds[[x]],]
    test <-  edx[folds[[x]],]

    test <- test %>%
      semi_join(train, by = 'movieId')

    mu <-mean ( train$rating )

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    b.movie <- train %>%
      group_by(movieId) %>%
      summarize(b.m = mean ( rating - mu))  # bias b.m for movie

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    predicted <-
      mu + test %>%
      left_join(b.movie, by='movieId') %>%
      pull(b.m)

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    rmse <- RMSE (test$rating, predicted)
    end <- as.numeric(Sys.time())
    runtime <- round(end-start)

    #result.movie.effect
    result <- data.frame(method="movie.effect",
                         fold=x,
                         rmse=rmse,
                         duration = runtime )

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>
  }
```

```
## call model function
  result.m2.movie.effect <- map_df(1:5, m2.movie.effect)

## save model result to overall results dataframe
  duration.tmp = sum(result.m2.movie.effect$duration)
  results <- rbind(results, data.frame(Method="m2 movie effect",
                                RMSE=mean(result.m2.movie.effect$rmse),
                                reference = "edx CV 5f",
                                lambda = 'na',
                                duration=duration.tmp))
```

The M2 result is a good improvement:

Method	RMSE	reference	lambda	duration
m1 simply the average	1.0603314	edx CV 5f	na	1
m2 movie effect	0.9437236	edx CV 5f	na	5

SETUP: Parameters for CV folds and lambda

With the next model we start regularization. In order to get the optimized lambda, we need to simulate with a range of lambda (we use 0:10). Based on the lowest RMSE we select the optimized lambda.

The following code creates a dataframe with all combinations (via 'expand.grid') of the lambdas and folds for the use in the model function.

```
#------------------------------------------------------------------------

# ** Setup parameters for crossvalidation and optimizing lambda ------------
#------------------------------------------------------------------------

# With regularization we need a range of lambda values
  #in order to find the lambda resulting in the lowest rmse

  #create parameters for folds and lambdas
  params <- expand.grid(1:5, seq(0,10,1))
  names(params) <- c("fold","lambda")
```

M3.1 Base + movie effect regularized individually

For the following regularized mode, we need to hand over to two parameters to our function: cv folds & lambdas. We use a mapping function, that can handle to parameters and returns a dataframe: map2_dfr.

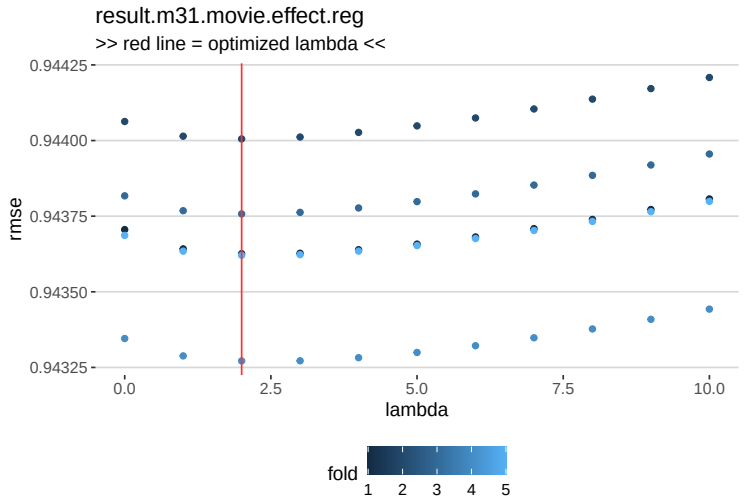
```
#_____________________________________________________________________________

# ** m3.1 Movie effect regularized ------------------------------------
#_____________________________________________________________________________

# model function
  m31.movie.effect.reg <- function (x,l) {

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    start <- as.numeric(Sys.time())

    train <- edx[-folds[[x]],]
    test <-  edx[folds[[x]],]

    test <- test %>%
      semi_join(train, by = 'movieId')

    mu <-mean ( train$rating )

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    b.movie <- train %>%
      group_by(movieId) %>%
      summarize(s.m = sum ( rating - mu), n.m =n())  %>%
      mutate ( b.m = s.m/(n.m + l))    # l  = lambda from params

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    predicted <-
      test %>%
      left_join( b.movie, by='movieId') %>%
      mutate (pred = mu + b.m) %>%
      pull(pred)

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    rmse <- RMSE (test$rating, predicted)
    end <- as.numeric(Sys.time())
    runtime <- round(end-start)

    #result.movie.effect
    result <- data.frame(method="movie.effect.reg",
                         fold=x, lambda= l,
                         rmse=rmse,
                         duration = runtime  )

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>
  }
```

```
## call model function
  # we handover 2 parameters (fold&lambda) to the function, so we use map2_dfr
  result.m31.movie.effect.reg <-
    map2_dfr(params$fold, params$lambda, m31.movie.effect.reg)

# calculate the optimized lambda
  lambda.m <- result.m31.movie.effect.reg %>%
    group_by(lambda) %>%
    summarize(rmse=mean(rmse))%>%
    filter(rmse==min(rmse)) %>% .$lambda

## calculate optimized rmse
  rmse.result.m31.movie.effect.reg <- result.m31.movie.effect.reg %>%
    group_by(lambda) %>% summarize(rmse=mean(rmse)) %>%
    filter(rmse==min(rmse)) %>% .$rmse

## save model result to overall results dataframe
  duration.tmp = sum(result.m31.movie.effect.reg$duration)
  results <- rbind(results, data.frame(Method="m3.1 movie effect reg.",
                                RMSE=rmse.result.m31.movie.effect.reg,
                                reference= "edx CV 5f",
                                lambda=lambda.m,
                                duration=duration.tmp))

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

The result dataframe for M3.1 contains the RMSE per fold and lambda. Here we see only the first 7 rows:

method	fold	lambda	rmse	duration
movie.effect.reg	1	0	0.9437056	1
movie.effect.reg	2	0	0.9440629	1
movie.effect.reg	3	0	0.9438172	1
movie.effect.reg	4	0	0.9433456	1
movie.effect.reg	5	0	0.9436864	1
movie.effect.reg	1	1	0.9436419	1
movie.effect.reg	2	1	0.9440144	1

In the plot we see the RMSE over lambda for the different CV folds:



We observe only a small improvement on the RMSE:

Method	RMSE	reference	lambda	duration
m1 simply the average	1.0603314	edx CV 5f	na	1
m2 movie effect	0.9437236	edx CV 5f	na	5
m3.1 movie effect reg.	0.9436562	edx CV 5f	2	55

Investigating the regualization impact

In order to see the regularization impact, let's compare 'Hellhounds on My Trail (1999)/movieId 3226' versus 'Pulp Fiction (1994)/movieId 296'.

In the model function we created a data.frame 'b.movie' for each training set per cv fold. For the investigation on the regularization impact, we create it on edx and include the prediction we would have done on edx. Off course, prediction wise not really correct, but it illustrates what we aim to investigate.

FYI with reference to the table: mu = 3.5124652, lambda movie = 2.

Table 13: Investigate regularization

title	n.m	mu.m	b.m.unreg	b.m.reg	pred.m.unreg	pred.m.reg
Hellhounds on My Trail (1999)	1	5.00000	1.48753	0.49584	5.00000	4.00831
Pulp Fiction (1994)	31362	4.15479	0.64232	0.64228	4.15479	4.15475

'mu.m' reflects the prediction we would have given with the unregularized Model M2 and equals 'pred.m.unreg'.
'b.m.unreg' is the bias considering the movie effect and equals the difference between the average rating for the movie (mu.m) and 'mu' (average of all ratings).
'b.m.reg' is the penalized bias considering the movie effect and lambda.
'pre.m.unreg' & 'pre.m.reg' are predictions considering the movie effect unregularized & regularized respectively.

Thus 'Hellhounds on My Trail''s unregularized prediction equals the only one rating given, whereas the regularized prediction is penalized down, even below 'Pulp Fiction'.
Nevertheless, this penalty improves the RMSE and thus the overall prediction performance.

With the many ratings given for 'Pulp Fiction', the difference (penalty) between the regularized and unregularized prediction is only marginal.

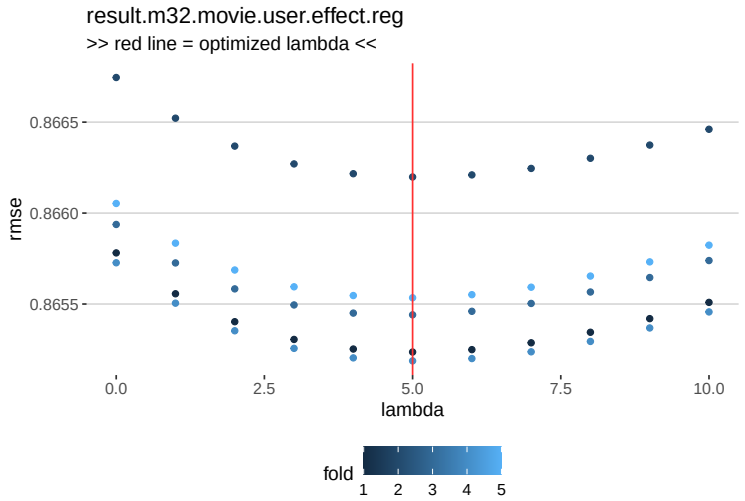
M3.2 Base + movie + user effect regularized individually

For the user effect we create an individual user-lambda.

```
#_____________________________________________________________________________

# ** m3.2 Movie+User effect regularized-------------------------------------
#_____________________________________________________________________________

## model function
m32.movie.user.effect.reg <- function (x,l) {

  # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

  start <- as.numeric(Sys.time())

  train <- edx[-folds[[x]],]
  test <-  edx[folds[[x]],]

  test <- test %>%
    semi_join(train, by = 'movieId') %>%
    semi_join(train, by = 'userId')

  mu <-mean ( train$rating )

  # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

  b.movie <- train %>%
    group_by(movieId) %>%
    summarize(s.m = sum ( rating - mu), n.m =n())  %>%
    mutate ( b.m = s.m/(n.m + lambda.m))  # opt.lambda movie-effect

  # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

  b.user <- train %>%
    left_join(b.movie, by='movieId') %>%
    group_by(userId) %>%
    summarize(s.u = sum ( rating - mu - b.m), n.u =n()) %>%
    mutate ( b.u = s.u/(n.u + l))   # l  = lambda from params

  # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

  predicted <-
    test %>%
    left_join(b.movie, by='movieId') %>%
    left_join(b.user, by='userId') %>%
    mutate (pred = mu + b.m + b.u) %>%
    pull(pred)

  # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

  rmse <- RMSE (test$rating, predicted)
  end <- as.numeric(Sys.time())
  runtime <- round(end-start)
```

```
  #result.movie.effect
  result <- data.frame(method="user.effect.reg",
                       fold=x,
                       lambda=l,
                       rmse=rmse,
                       duration=runtime )

  # >>>>>>>>>>>>>>>>>>>>>>>>>>>>
}

## call model function
    result.m32.movie.user.effect.reg <-
      map2_dfr(params$fold, params$lambda, m32.movie.user.effect.reg)

# calculate the optimized lambda
    lambda.u <- result.m32.movie.user.effect.reg %>%
      group_by(lambda) %>%
      summarize(rmse=mean(rmse))  %>%
      filter(rmse==min(rmse)) %>% .$lambda

## calculate optimized rmse
  rmse.result.m32.movie.user.effect.reg <-
    result.m32.movie.user.effect.reg %>%
      group_by(lambda) %>% summarize(rmse=mean(rmse))  %>%
      filter(rmse==min(rmse)) %>% .$rmse

## save model result to overall results dataframe
  duration.tmp = sum(result.m32.movie.user.effect.reg$duration)
  results <- rbind(results, data.frame(Method="m3.2 movie+user effect reg.",
                                RMSE=rmse.result.m32.movie.user.effect.reg,
                                reference = "edx CV 5f",
                                lambda=paste('l.m=',lambda.m,'/','l.u=',
                                     lambda.u, sep=''),
                                duration=duration.tmp))

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

Plot RMSE versus lambda by CV folds:



The RMSE decreased significantly by the user-effect, but the runtime increased:

Method	RMSE	reference	lambda	duration
m1 simply the average	1.0603314	edx CV 5f	na	1
m2 movie effect	0.9437236	edx CV 5f	na	5
m3.1 movie effect reg.	0.9436562	edx CV 5f	2	55
m3.2 movie+user effect reg.	0.8655195	edx CV 5f	l.m=2/l.u=5	185

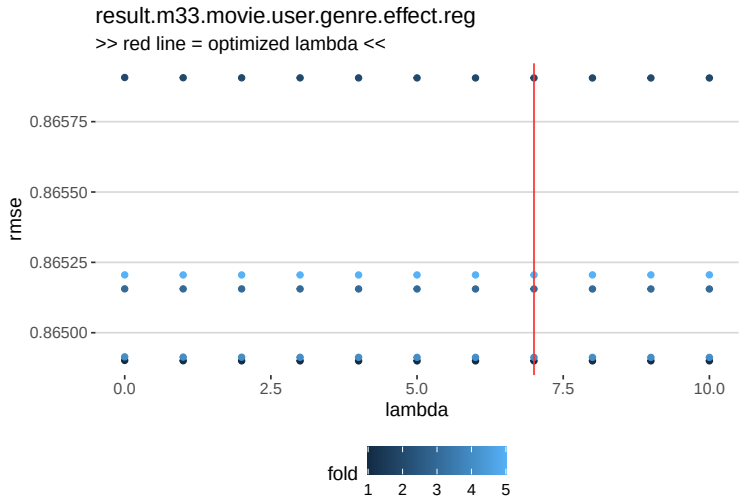
M3.3 Base + movie + user + genre effect regularized individually

We enhance the model by the genres effect and create an individual genre-lambda.

```
#_____________________________________________________________________________

# ** m3.3 Movie+User+Genres effect regularized ----------------------------
#_____________________________________________________________________________

## model function
    m33.movie.user.genre.effect.reg <- function (x,l) {

      # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

      start <- as.numeric(Sys.time())

      train <- edx[-folds[[x]],]
      test <-  edx[folds[[x]],]

      test <- test %>%
        semi_join(train, by = 'movieId') %>%
        semi_join(train, by = 'userId')

      mu <-mean ( train$rating )

      # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

      b.movie <- train %>%
        group_by(movieId) %>%
        summarize(s.m = sum ( rating - mu), n.m =n())  %>%
        mutate ( b.m = s.m/(n.m + lambda.m))   #  opt. lambda movie-effect

      # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

      b.user <- train %>%
        left_join(b.movie, by='movieId') %>%
        group_by(userId) %>%
        summarize(s.u = sum ( rating - mu - b.m), n.u =n()) %>%
        mutate ( b.u = s.u/(n.u + lambda.u))   # opt, lambda  user-effect

       # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

      b.genre <- train %>%
        left_join(b.movie, by='movieId') %>%
        left_join(b.user, by='userId') %>%
        group_by(genres) %>%
        summarize(s.g = sum ( rating - mu - b.m - b.u ), n.g = n()) %>%
        mutate ( b.g = s.g/(n.g + l))   # l  = lambda from params

      # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

      predicted <-
        test %>%
        left_join(b.movie, by='movieId') %>%
        left_join(b.user, by='userId') %>%
```

```
        left_join(b.genre, by='genres') %>%
        mutate (pred = mu + b.m + b.u + b.g) %>%
        pull(pred)

      # >>>>>>>>>>>>>>>>>>>>>>>>>>>>
      rmse <- RMSE (test$rating, predicted)
      end <- as.numeric(Sys.time())
      runtime <- round(end-start)

      #result.movie.effect
      result <- data.frame(method="movie+user+genre.effect.reg",
                           fold=x,
                           lambda= l,
                           rmse=rmse,
                           duration=runtime )

      # >>>>>>>>>>>>>>>>>>>>>>>>>>>>
    }

## call model function
    result.m33.movie.user.genre.effect.reg <-
      map2_dfr(params$fold, params$lambda, m33.movie.user.genre.effect.reg)

# calculate the optimized lambda
    lambda.g <- result.m33.movie.user.genre.effect.reg %>%
      group_by(lambda) %>%
      summarize(rmse=mean(rmse))%>%
      filter(rmse==min(rmse)) %>% .$lambda

## calculate optimized rmse
    rmse.result.m33.movie.user.genre.effect.reg <-
      result.m33.movie.user.genre.effect.reg %>%
      group_by(lambda) %>% summarize(rmse=mean(rmse)) %>%
      filter(rmse==min(rmse)) %>% .$rmse

## save model result to overall results dataframe
  duration.tmp = sum(result.m33.movie.user.genre.effect.reg$duration)
  results <- rbind(results, data.frame(
                                Method="m3.3 movie+user+genre effect reg.",
                                RMSE=rmse.result.m33.movie.user.genre.effect.reg,
                                reference = "edx CV 5f",
                                lambda=
                                  paste('l.m=',lambda.m,'/','l.u=',
                                     lambda.u,'/','l.g=',lambda.g, sep=''),
                                duration=duration.tmp))

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

Plot RMSE versus lambda by CV folds:



We observe only little reduction of the RMSE with the genre effect, but an further increase in runtime:

Method	RMSE	reference	lambda	duration
m1 simply the average	1.0603314	edx CV 5f	na	1
m2 movie effect	0.9437236	edx CV 5f	na	5
m3.1 movie effect reg.	0.9436562	edx CV 5f	2	55
m3.2 movie+user effect reg.	0.8655195	edx CV 5f	l.m=2/l.u=5	185
m3.3 movie+user+genre effect reg.	0.8652157	edx CV 5f	l.m=2/l.u=5/l.g=7	242

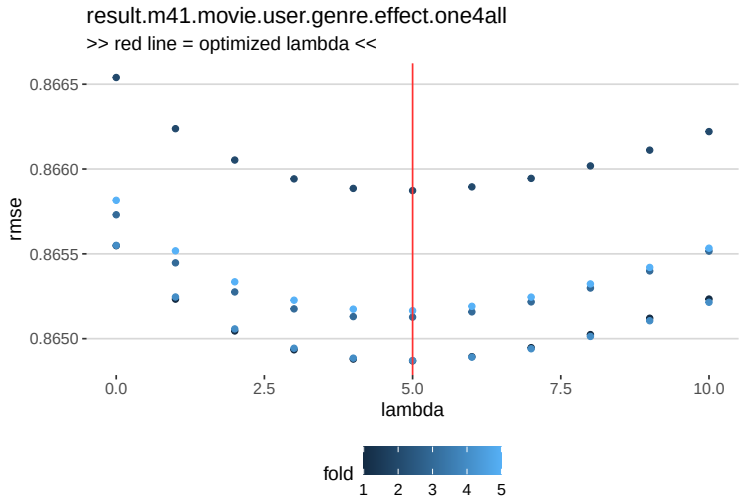
M4.1 Base + movie + user + genre effect regularized one-for-all

With the next model we keep the effects, but switch to one lambda for all effects, as applied also in course. Thus we will see if the idea of indiviual lambdas pays off.

```
#_____________________________________________________________________________

# ** m4.1 Movie+User+Genres effect regularized one-for-all --------------------
#_____________________________________________________________________________

## model function
  m41.movie.user.genre.effect.one4all <- function (x,l) {

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    start <- as.numeric(Sys.time())

    train <- edx[-folds[[x]],]
    test <-  edx[folds[[x]],]

    test <- test %>%
      semi_join(train, by = 'movieId') %>%
      semi_join(train, by = 'userId')

    mu <-mean ( train$rating )

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    b.movie <- train %>%
      group_by(movieId) %>%
      summarize(s.m = sum ( rating - mu), n.m =n())  %>%
      mutate ( b.m = s.m/(n.m + l))    #  l lambda l

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    b.user <- train %>%
      left_join(b.movie, by='movieId') %>%
      group_by(userId) %>%
      summarize(s.u = sum ( rating - mu - b.m), n.u =n()) %>%
      mutate ( b.u = s.u/(n.u + l))    # l lambda l

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    b.genre <- train %>%
      left_join(b.movie, by='movieId') %>%
      left_join(b.user, by='userId') %>%
      group_by(genres) %>%
      summarize(s.g = sum ( rating - mu - b.m - b.u ), n.g = n()) %>%
      mutate ( b.g = s.g/(n.g + l))    # lambda

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    predicted <-
      test %>%
      left_join(b.movie, by='movieId') %>%
```

```
      left_join(b.user, by='userId') %>%
      left_join(b.genre, by='genres') %>%
      mutate (pred = mu + b.m + b.u + b.g) %>%
      pull(pred)

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>
    rmse <- RMSE (test$rating, predicted)
    end <- as.numeric(Sys.time())
    runtime <- round(end-start)

    #result.movie.effect
    result <- data.frame(method="movie+user+genre.effect.reg",
                         fold=x,
                         lambda= l,
                         rmse=rmse, duration=runtime )

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>
  }

## call model function
result.m41.movie.user.genre.effect.one4all <-
  map2_dfr(params$fold, params$lambda, m41.movie.user.genre.effect.one4all)

# calculate the optimized lambda
  lambda.mug <- result.m41.movie.user.genre.effect.one4all %>%
    group_by(lambda) %>% summarize(rmse=mean(rmse))  %>%
    filter(rmse==min(rmse)) %>% .$lambda

## calculate optimized rmse
  rmse.result.m41.movie.user.genre.effect.one4all <-
    result.m41.movie.user.genre.effect.one4all %>%
    group_by(lambda) %>% summarize(rmse=mean(rmse))  %>%
    filter(rmse==min(rmse)) %>% .$rmse

## save model result to overall results dataframe
  duration.tmp = sum(result.m41.movie.user.genre.effect.one4all$duration)
  results <- rbind(results, data.frame(
                         Method="m4.1 movie+user+genre.effect reg. one4all",
                         RMSE=rmse.result.m41.movie.user.genre.effect.one4all,
                         reference = "edx CV 5f",
                         lambda=lambda.mug,
                         duration=duration.tmp))

# >>>>>>>>>>>>>>>>>>>>>>>>>>>
```

Plot RMSE versus lambda by CV folds:



The results with 'one4all' lambda:

Method	RMSE	reference	lambda	duration
m1 simply the average	1.0603314	edx CV 5f	na	1
m2 movie effect	0.9437236	edx CV 5f	na	5
m3.1 movie effect reg.	0.9436562	edx CV 5f	2	55
m3.2 movie+user effect reg.	0.8655195	edx CV 5f	l.m=2/l.u=5	185
m3.3 movie+user+genre effect reg.	0.8652157	edx CV 5f	l.m=2/l.u=5/l.g=7	242
m4.1 movie+user+genre.effect reg. one4all	0.8651809	edx CV 5f	5	229

REVIEW: Regularization individually versus one-for-all

The individual lambdas do not pay off, although the difference to one-4-all is modest.

The decision based on this result: Continue with one-for-all lambda for the next two models with the temporal effect.

M5.1 Base + movie + user + user + lag effect regularized one-for-all

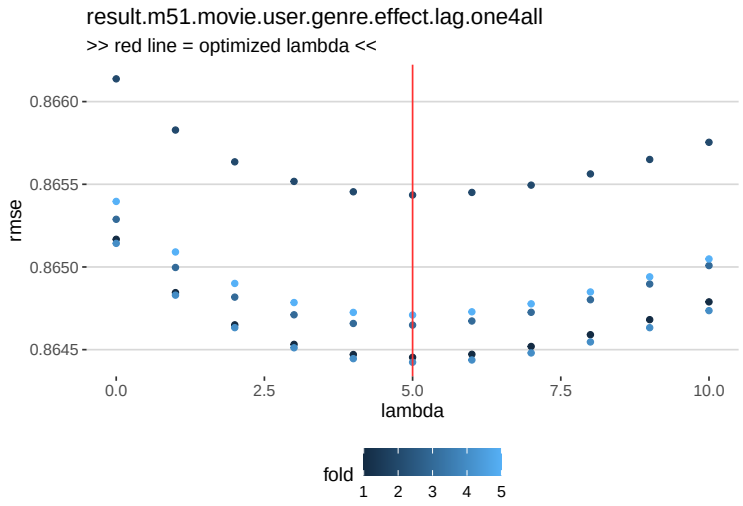
The next model includes the first temporal effect: rating lag.

```
#_____________________________________________________________________________

# ** m5.1 Movie+User+Genres+lag effect regularized one-for-all -------------
#_____________________________________________________________________________

## model function
  m51.movie.user.genre.effect.lag.one4all <- function (x,l) {

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    start <- as.numeric(Sys.time())

    train <- edx[-folds[[x]],]
    test <-  edx[folds[[x]],]

    test <- test %>%
      semi_join(train, by = 'movieId') %>%
      semi_join(train, by = 'userId')

    mu <-mean ( train$rating )

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    b.movie <- train %>%
      group_by(movieId) %>%
      summarize(s.m = sum ( rating - mu), n.m =n())  %>%
      mutate ( b.m = s.m/(n.m + l))    #  l=lambda

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    b.user <- train %>%
      left_join(b.movie, by='movieId') %>%
      group_by(userId) %>%
      summarize(s.u = sum ( rating - mu - b.m), n.u =n()) %>%
      mutate ( b.u = s.u/(n.u + l))    #  l=lambda

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    b.genre <- train %>%
      left_join(b.movie, by='movieId') %>%
      left_join(b.user, by='userId') %>%
      group_by(genres) %>%
      summarize(s.g = sum ( rating - mu - b.m - b.u ), n.g = n()) %>%
      mutate ( b.g = s.g/(n.g + l))    # l=lambda

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    b.lag <- train %>%
      left_join(b.movie, by='movieId') %>%
      left_join(b.user, by='userId') %>%
      left_join(b.genre, by='genres') %>%
```

```
      group_by(rating_lag) %>%
      summarize(s.l = sum ( rating - mu - b.m - b.u - b.g), n.l = n()) %>%
      mutate(b.l = s.l/(n.l + l))     # l=lambda

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    predicted <-
      test %>%
      left_join(b.movie, by='movieId') %>%
      left_join(b.user, by='userId') %>%
      left_join(b.genre, by='genres') %>%
      left_join(b.lag, by='rating_lag') %>%
      mutate (pred = mu + b.m + b.u + b.g + b.l) %>%
      pull(pred)

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>
    rmse <- RMSE (test$rating, predicted)
    end <- as.numeric(Sys.time())
    runtime <- round(end-start)

    #result.movie.effect
    result <- data.frame(method="movie+user+genre+lag.effect.one4all",
                         fold=x,
                         lambda= l,
                         rmse=rmse,
                         duration=runtime )

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>
  }

## call model function
  result.m51.movie.user.genre.effect.lag.one4all <-
    map2_dfr(params$fold, params$lambda,
             m51.movie.user.genre.effect.lag.one4all)

# calculate the optimized lambda
  lambda.mugl <- result.m51.movie.user.genre.effect.lag.one4all %>%
    group_by(lambda) %>% summarize(rmse=mean(rmse))  %>%
    filter(rmse==min(rmse)) %>% .$lambda

## calculate optimized rmse
  rmse.result.m51.movie.user.genre.effect.lag.one4all <-
    result.m51.movie.user.genre.effect.lag.one4all %>%
    group_by(lambda) %>% summarize(rmse=mean(rmse))  %>%
    filter(rmse==min(rmse)) %>% .$rmse

## save model result to overall results dataframe
  duration.tmp = sum(result.m51.movie.user.genre.effect.lag.one4all$duration)
  results <- rbind(results, data.frame(
                    Method="m5.1 movie+user+genre+lag effect reg. one4all",
                    RMSE=rmse.result.m51.movie.user.genre.effect.lag.one4all,
                    reference = "edx CV 5f",
                    lambda=lambda.mugl,
```

```
                    duration=duration.tmp))

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

Plot RMSE versus lambda by CV folds:



The improvement by the first temporal effect is moderate:

Method	RMSE	reference	lambda	duration
m1 simply the average	1.0603314	edx CV 5f	na	1
m2 movie effect	0.9437236	edx CV 5f	na	5
m3.1 movie effect reg.	0.9436562	edx CV 5f	2	55
m3.2 movie+user effect reg.	0.8655195	edx CV 5f	l.m=2/l.u=5	185
m3.3 movie+user+genre effect reg.	0.8652157	edx CV 5f	l.m=2/l.u=5/l.g=7	242
m4.1 movie+user+genre.effect reg. one4all	0.8651809	edx CV 5f	5	229
m5.1 movie+user+genre+lag effect reg. one4all	0.8647341	edx CV 5f	5	363

M5.2 Base + movie + user + user + year effect regularized one-for-all

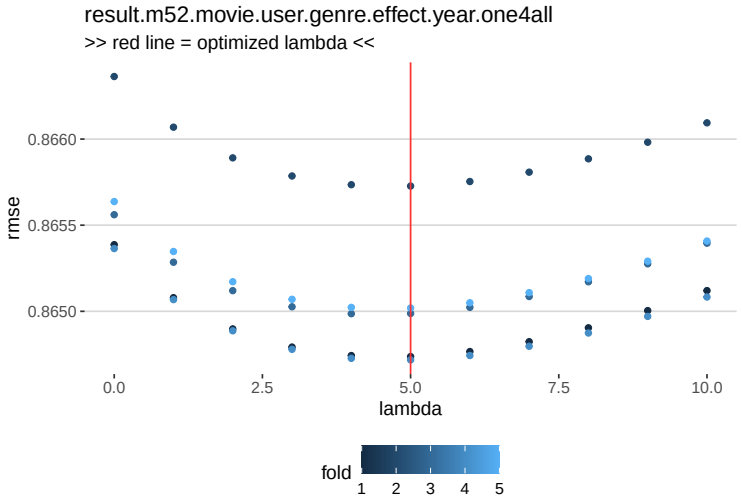
The model with year as temporal effect.

```
#_____________________________________________________________________________

# ** m5.2 Movie+User+Genres+year effect regularized one4all ----------------
#_____________________________________________________________________________

## model function
  m52.movie.user.genre.effect.year.one4all <- function (x,l) {

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    start <- as.numeric(Sys.time())

    train <- edx[-folds[[x]],]
    test <-  edx[folds[[x]],]

    test <- test %>%
      semi_join(train, by = 'movieId') %>%
      semi_join(train, by = 'userId')

    mu <-mean ( train$rating )

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    b.movie <- train %>%
      group_by(movieId) %>%
      summarize(s.m = sum ( rating - mu), n.m =n())  %>%
      mutate ( b.m = s.m/(n.m + l))    #  l=lambda

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    b.user <- train %>%
      left_join(b.movie, by='movieId') %>%
      group_by(userId) %>%
      summarize(s.u = sum ( rating - mu - b.m), n.u =n()) %>%
      mutate ( b.u = s.u/(n.u + l))    #  l=lambda

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    b.genre <- train %>%
      left_join(b.movie, by='movieId') %>%
      left_join(b.user, by='userId') %>%
      group_by(genres) %>%
      summarize(s.g = sum ( rating - mu - b.m - b.u ), n.g = n()) %>%
      mutate ( b.g = s.g/(n.g + l))    # l=lambda

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    b.year <- train %>%
      left_join(b.movie, by='movieId') %>%
      left_join(b.user, by='userId') %>%
      left_join(b.genre, by='genres') %>%
```

```
      group_by(year) %>%
      summarize(s.y = sum ( rating - mu - b.m - b.u - b.g), n.y = n()) %>%
      mutate ( b.y = s.y/(n.y + l))   # l=lambda

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>

    predicted <-
      test %>%
      left_join(b.movie, by='movieId') %>%
      left_join(b.user, by='userId') %>%
      left_join(b.genre, by='genres') %>%
      left_join(b.year, by='year') %>%
      mutate (pred = mu + b.m + b.u + b.g + b.y) %>%
      pull(pred)

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>
    rmse <- RMSE (test$rating, predicted)
    end <- as.numeric(Sys.time())
    runtime <- round(end-start)

    #result.movie.effect
    result <- data.frame(method="movie+user+genre+year.effect.one4all",
                         fold=x,
                         lambda= l,
                         rmse=rmse,
                         duration=runtime )

    # >>>>>>>>>>>>>>>>>>>>>>>>>>>>
  }

## call model function
  result.m52.movie.user.genre.effect.year.one4all <-
    map2_dfr(params$fold, params$lambda,
             m52.movie.user.genre.effect.year.one4all)

# calculate the optimized lambda
  lambda.mugy <- result.m52.movie.user.genre.effect.year.one4all %>%
    group_by(lambda) %>% summarize(rmse=mean(rmse))  %>%
    filter(rmse==min(rmse)) %>% .$lambda

## calculate optimized rmse
  rmse.result.m52.movie.user.genre.effect.year.one4all <-
    result.m52.movie.user.genre.effect.year.one4all %>%
    group_by(lambda) %>% summarize(rmse=mean(rmse))  %>%
    filter(rmse==min(rmse)) %>% .$rmse

## save model result to overall results dataframe
  duration.tmp = sum(result.m52.movie.user.genre.effect.year.one4all$duration)
  results <- rbind(results, data.frame(
                  Method="m5.2 movie+user+genre+year effect reg. one4all",
                  RMSE=rmse.result.m52.movie.user.genre.effect.year.one4all,
                  reference = "edx CV 5f",
                  lambda=lambda.mugy,
```

```
                  duration=duration.tmp))

# >>>>>>>>>>>>>>>>>>>>>>>>>>>

gc()
```

Plot RMSE versus lambda by CV folds:



The decision on the final model will be based on this results:

Method	RMSE	reference	lambda	duration
m1 simply the average	1.0603314	edx CV 5f	na	1
m2 movie effect	0.9437236	edx CV 5f	na	5
m3.1 movie effect reg.	0.9436562	edx CV 5f	2	55
m3.2 movie+user effect reg.	0.8655195	edx CV 5f	l.m=2/l.u=5	185
m3.3 movie+user+genre effect reg.	0.8652157	edx CV 5f	l.m=2/l.u=5/l.g=7	242
m4.1 movie+user+genre.effect reg. one4all	0.8651809	edx CV 5f	5	229
m5.1 movie+user+genre+lag effect reg. one4all	0.8647341	edx CV 5f	5	363
m5.2 movie+user+genre+year effect reg. one4all	0.8650378	edx CV 5f	5	357

REVIEW: Decide on final model

The temporal effects add some improvement on the RMSE, thus we keep the better one for the final model. In case of overfitting, they might have caused even an increased RMSE and hence we would not have included them at all to the final model.

Decision: Rating lag shows the lower RMSE, hence M5.1 is the base for the final model applied for validation.

Final evaluation

M6 Final model based on M5.1

For the final model we create the 'b's' using the optimized lambda from model M5.1.

```
#_____________________________________________________________________________

# * Final evaluation-----------------------------------------------------------

#_____________________________________________________________________________


#_____________________________________________________________________________

# ** M6 Final Model ------------------------------------------------------------

#_____________________________________________________________________________

# lambda for final model
  lambda.opt <- lambda.mugl

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>

  edx <- edx %>%
    semi_join(edx, by = 'movieId') %>%
    semi_join(edx, by = 'userId')

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>

  b.movie.f <- edx %>%
    group_by(movieId) %>%
    summarize(s.m = sum ( rating - mu), n.m =n())  %>%
    mutate ( b.m = s.m/(n.m + lambda.opt))

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>

  b.user.f <- edx %>%
    left_join(b.movie.f, by='movieId') %>%
    group_by(userId) %>%
    summarize(s.u = sum ( rating - mu - b.m), n.u =n()) %>%
    mutate ( b.u = s.u/(n.u + lambda.opt))

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>

  b.genre.f <- edx %>%
    left_join(b.movie.f, by='movieId') %>%
    left_join(b.user.f, by='userId') %>%
    group_by(genres) %>%
    summarize(s.g = sum ( rating - mu - b.m - b.u ), n.g = n()) %>%
    mutate ( b.g = s.g/(n.g + lambda.opt))

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>

    b.lag.f <- edx %>%
```

```
    left_join(b.movie.f, by='movieId') %>%
    left_join(b.user.f, by='userId') %>%
    left_join(b.genre.f, by='genres') %>%
    group_by(rating_lag) %>%
    summarize(s.l = sum ( rating - mu - b.m - b.u - b.g), n.l = n()) %>%
    mutate(b.l = s.l/(n.l+lambda.opt))

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>

gc()
```

Enhance validation set

Before we can apply the model to the validation set, we need to have the same structure as in edx. Therefor we apply the same transformations and enhancements as we did for edx.

Notice: We do not draw conclusions for the final model from validation nor we enhance information from outside.
The variables we add are just derived from intrinsic information within the validation set.

```
#_____________________________________________________________________________

# ** Enhance validation ------------------------------------------------
#_____________________________________________________________________________

# Prepare validation-set for final model

  # genres as factor
      validation$genres <- as.factor(validation$genres)

  # convert timestamp to POSIX date.time
      validation$ts_date <-
        as.POSIXct(validation$timestamp, origin="1970-01-01")

  # extract movie year from title
      validation$year <- validation$title %>% str_sub(-5,-2) %>% as.numeric()

  # create 'year of rating'   &   'rating-lag'
      validation <- validation %>%
        mutate ( year.rating = as.numeric(isoyear(ts_date)),
                 rating_lag =  year.rating - year,
                 rating_lag = ifelse (rating_lag<0,0,rating_lag))
```

Results for the final evaluation

```
#_____________________________________________________________________________

# Final prediction ----------------------------------------------------------
#_____________________________________________________________________________

# Final prediction
  predicted <-
    validation %>%
    left_join(b.movie.f, by='movieId') %>%
    left_join(b.user.f, by='userId') %>%
    left_join(b.genre.f, by='genres') %>%
    left_join(b.lag.f, by='rating_lag') %>%
    mutate (pred = mu + b.m + b.u + b.g + b.l) %>%
    pull(pred)

  rmse.final <- RMSE (validation$rating, predicted)

## save model result to overall results dataframe

  results <- rbind(results, data.frame(
                       Method  ="m6 movie+user+genre+lag effect reg. one4all",
                       RMSE =  rmse.final,
                       reference = "validation CV 5f",
                       lambda = lambda.opt,
                       duration = 'na'))
```

The table with the final results:

Method	RMSE	reference	lambda	duration
m1 simply the average	1.0603314	edx CV 5f	na	1
m2 movie effect	0.9437236	edx CV 5f	na	5
m3.1 movie effect reg.	0.9436562	edx CV 5f	2	55
m3.2 movie+user effect reg.	0.8655195	edx CV 5f	l.m=2/l.u=5	185
m3.3 movie+user+genre effect reg.	0.8652157	edx CV 5f	l.m=2/l.u=5/l.g=7	242
m4.1 movie+user+genre.effect reg. one4all	0.8651809	edx CV 5f	5	229
m5.1 movie+user+genre+lag effect reg. one4all	0.8647341	edx CV 5f	5	363
m5.2 movie+user+genre+year effect reg. one4all	0.8650378	edx CV 5f	5	357
m6 movie+user+genre+lag effect reg. one4all	0.8640163	validation CV 5f	5	na

The final RMSE on validation is 0.8640163.

Summary

The project's target was to predict movie ratings, based on a historical dataset from movielens (10M version). After downloading and preparing the data, a quick view on the data gave a first impression of the content. Enhancing the data set enabled an advanced view on the data from different angles. As a kind of data-journey, the data exploration and visualization gave a deeper insight and inspired ideas for possible prediction parameters.
Based on the gained insight, the methods were defined and, out of curiosity, the strategy to check the impact regularization with lambda per effect versus one-lambda-for all. Also the challenge between two temporal effects was included in the procedure. Enhancing the models followed the plan and showed different levels of improvement. After taking the decisions on 'what lambda' & 'what temporal effect' the final model met the target of a RMSE < 0.86490.

As the project focused on regression, other effects could have been used. E.g. to break the temporal effects further down to find seasonal effects: Do users rate different on certain weekdays? Is there a seasonal mood (christmas, spring, summerholiday,...) influencing the users ratings? Other optimization options for the future would be to apply special recommender algorithms.

From a personal point of view as quite a newbie in R, I can say I learned a lot about coding, R Markdown, memory issues and much more. I look forward to enhancing my knowledge and experience to more advanced ML methods.