Capstone Exam: The MovieLens project

Gergely Fenyvesi

3/31/2020

Introduction

Creating recommendation systems is a typical machine learning exercice. In October 2006, Netflix offered a challenge to the data science community: improve their recommendation algorithm by 10% and win a million dollars. The Netflix data is not publicly available, but the GroupLens research lab (Grouplens) generated their own database with over 20 million ratings for over 27,000 movies by more than 138,000 users. We use a subset of this data for our Capstone Exam. The exercice is the following: split the movielens data to a training (data.frame:edx) and a validation (data.frame:validation) set. Train an algorithm on the edx dataset and achieve the smallest possible RMSE (Residual Mean Squared Error) when you predict the given rating by each user in the validation set. The validation set must not be used under any circumstance for training.

Dataset creation

The following code will create the edx and validation datasets. For data exploration I kept the whole movielens dataset as well.

```
##############################################################################
# Create edx set, validation set + keep whole movielens set for data exploration
##############################################################################

# Note: this process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")

# 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")
```

```r
# 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)` instead
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)

#we remove unnecessary data
rm(dl, ratings, movies, test_index, temp, removed)
```

Methods

To determine the proper method to use, I started to explore the whole MovieLens dataset, by:

1. Exploring the structure of the dataset
2. Transforming the structure of the dataset:

- Extract the production year of the film from the title
- Transform the timestamp of the rating given to a conventional date format

3. Exploring the ratings given by films and by users

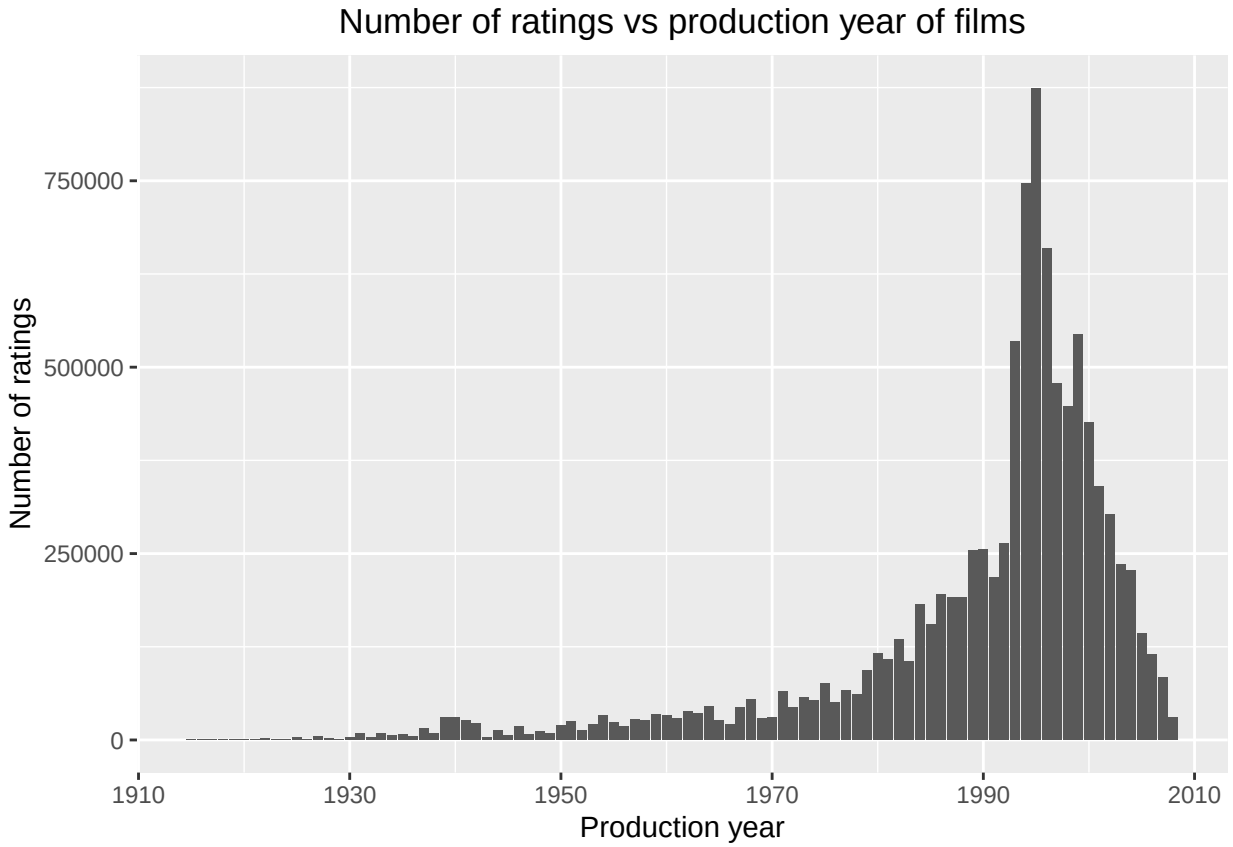
MovieLens dataset

```
## 'data.frame':    10000054 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983653 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller"
```

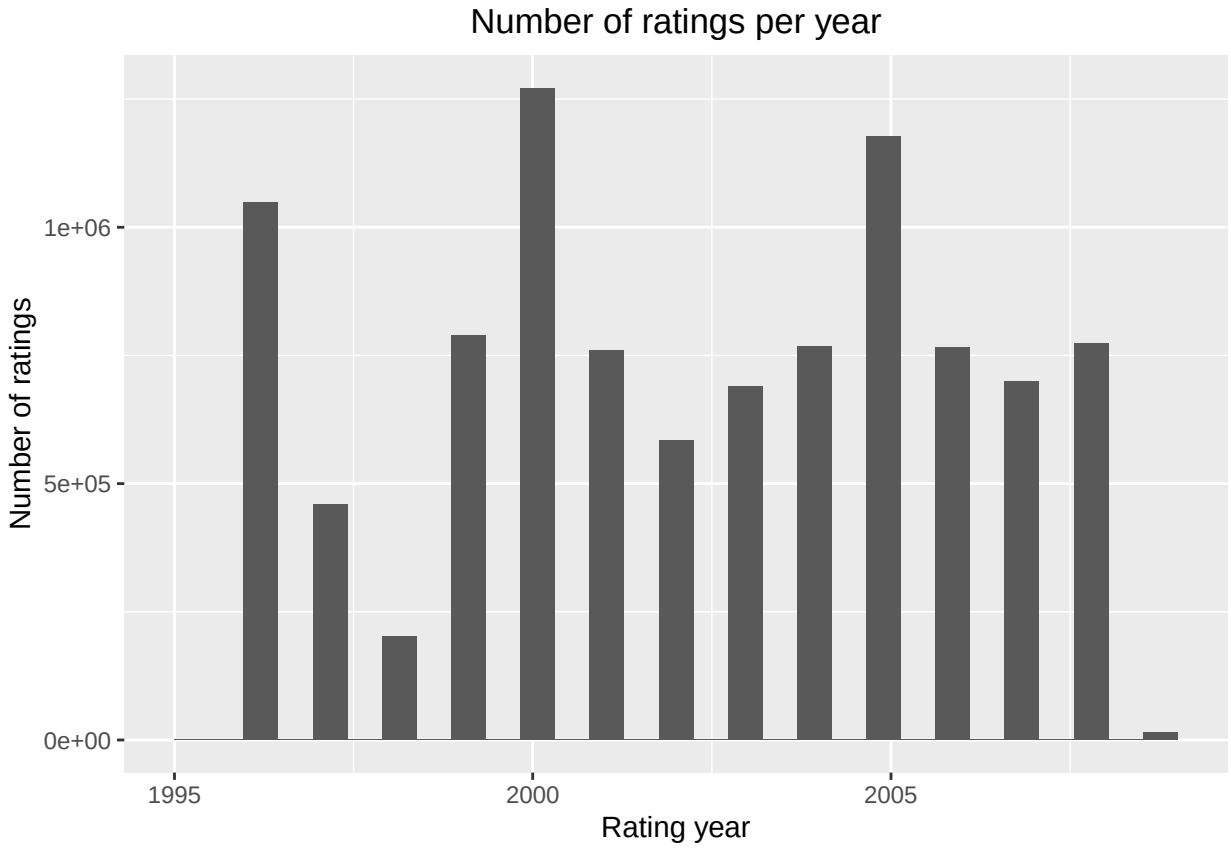
```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18123   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35740   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53608   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:10000054    Length:10000054
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

The MovieLens dataset contains 10,000,054 ratings given by 69,878 users, including 10,677 films, produced between 1915 and 2008. The most rated films are not the most recent ones, but from the mid-90s, which are recent enough to be popular and users had enough time to rate:

prodyear	n
1995	874436
1994	746042
1996	659425
1999	543990
1993	534899
1997	477463
1998	446739
2000	425218
2001	339508
2002	302452



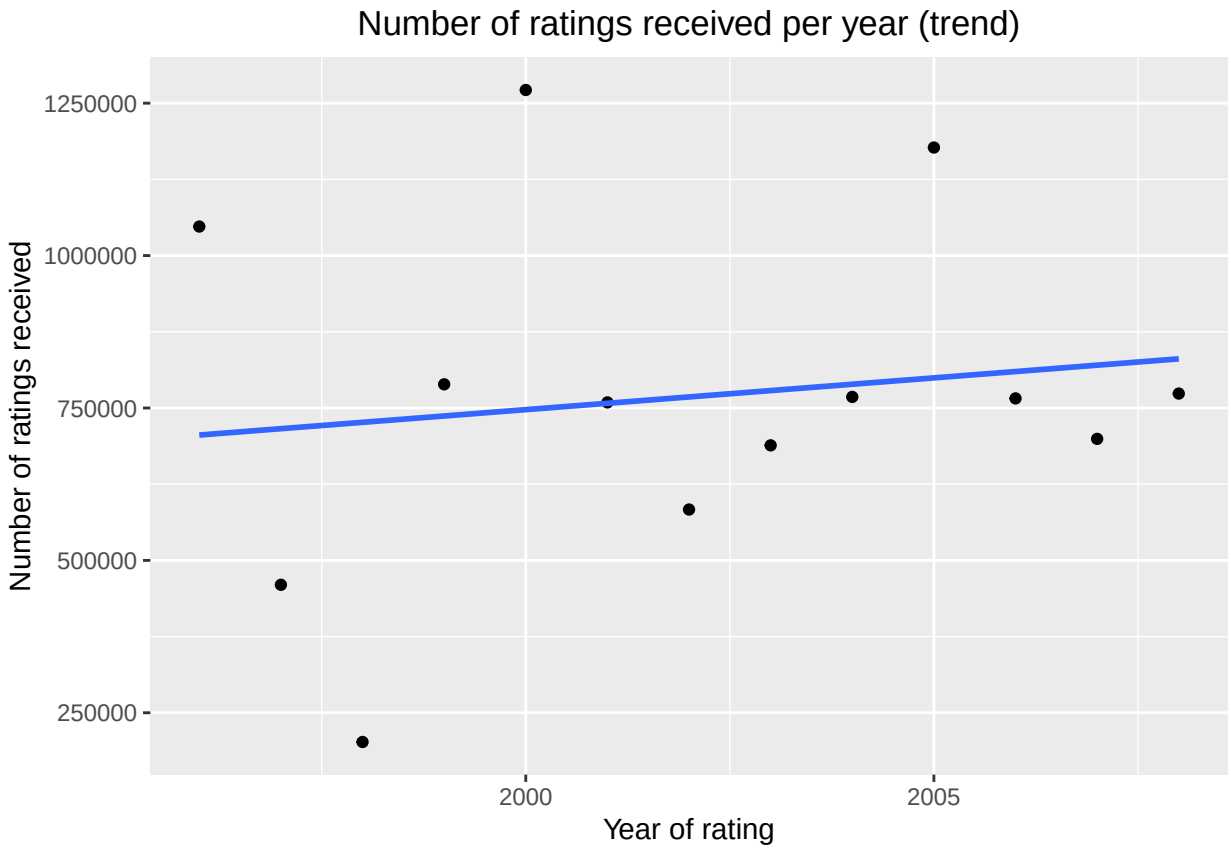
Ratings were given between 1995-01-09 11:46:49 and 2009-01-05 05:02:16. The number of rating per year is not constant:



Ratings per year:

rate_year	n
2000	1271623
2005	1177283
1996	1047618
1999	788793
2008	773617
2004	768168
2006	765733
2001	759141
2007	699325
2003	688694
2002	583409
1997	459947
1998	202092
2009	14608
1995	3

If we exclude the beginning and ending year, the two outliers, we still see more than 600% variation between the two extremes. An upward trend is observable:

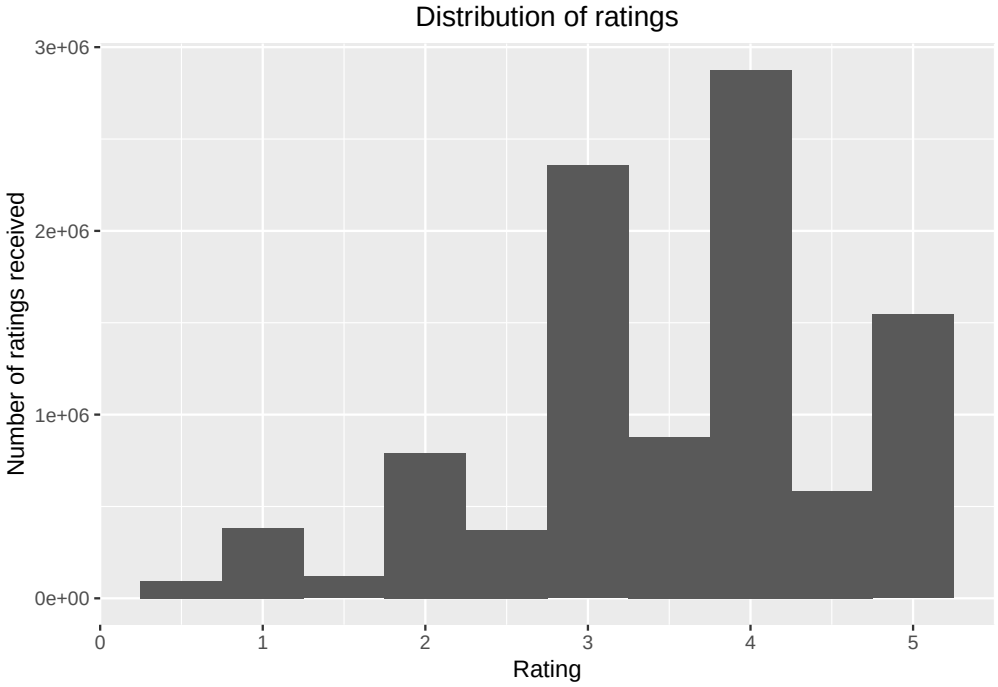


Ratings distribution

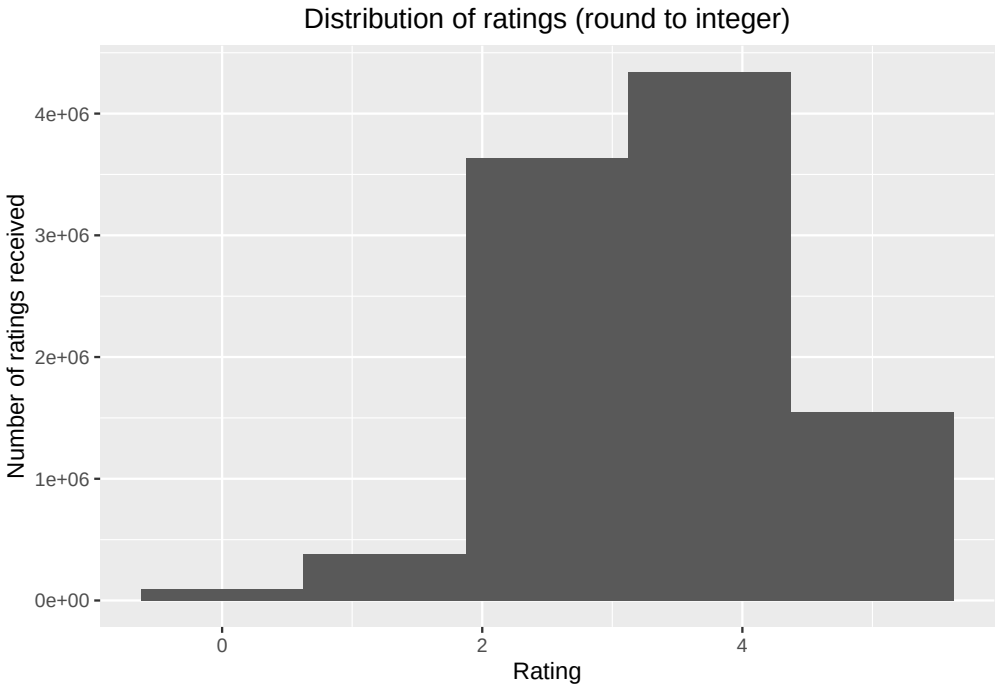
The median rating of 4 is higher than the average rating of 3.51. It means a negative skew, that we can verify by computing the 2nd, 3rd and 4th moments (stdev, skewnes and kurtosis) of the distribution (for a standard normal distriubution the average is 0, the standard deviation equals to 1, with a skewness of 0 and kurtosis of 3):

- stdev = 1.0604185
- skewness = -0.5959733
- kurtosis = 3.0083531

The empirical distribution is multimodal as half ratings are rare:



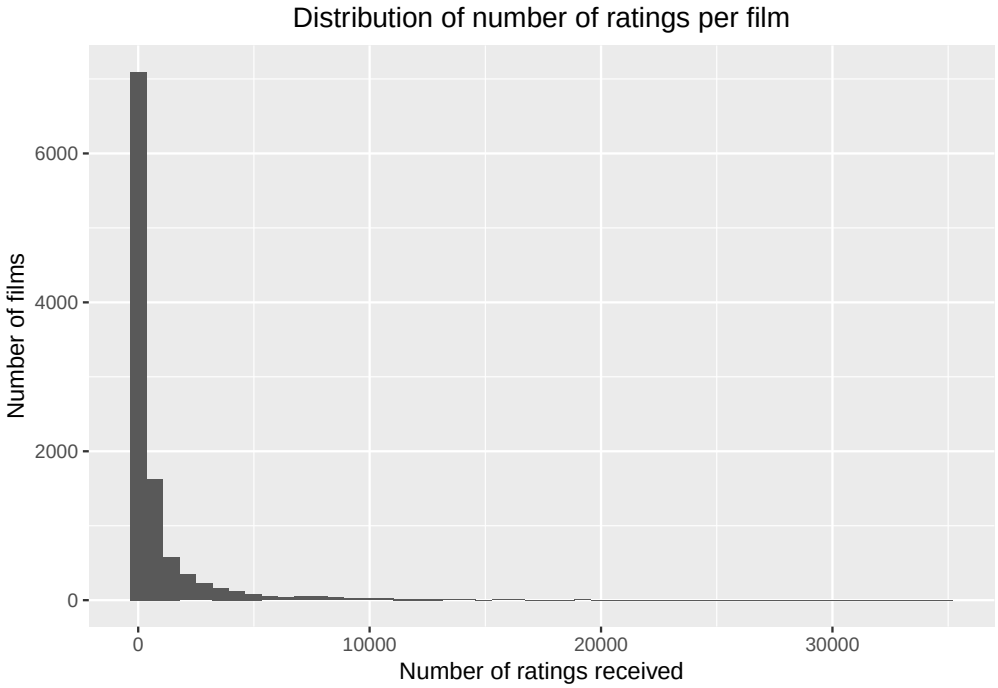
If we round ratings we can see that the distribution is skewed and unimodal, with a mode of 4:



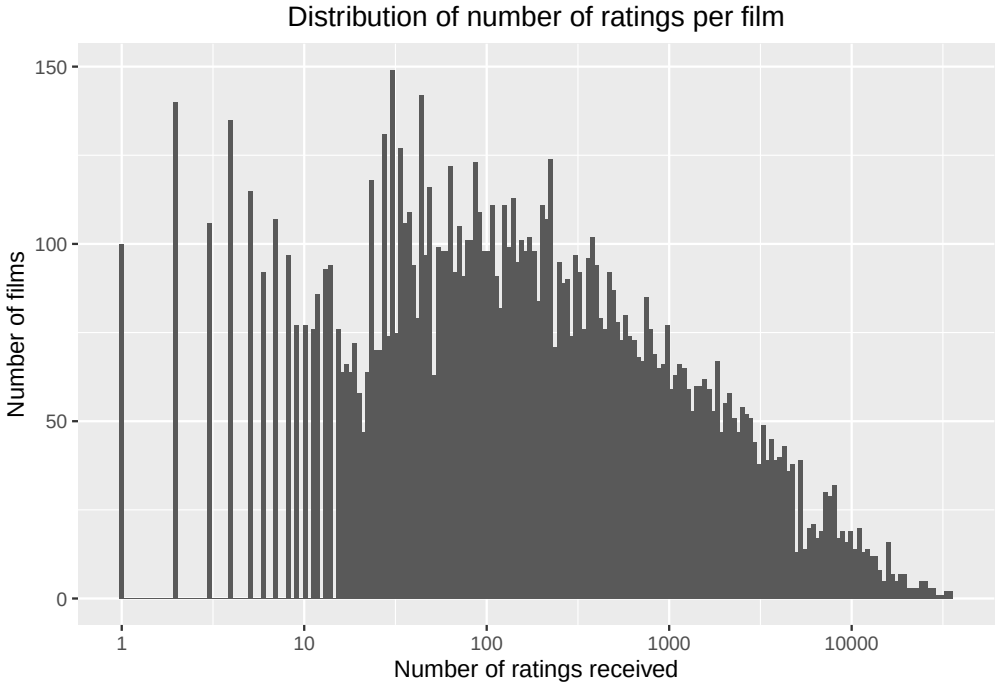
Ratings per film (the film effect)

Number of ratings

Most films received a small number of ratings:



It is easier to visualise the distribution on a log scale:



A couple of blockbusters received a really large number of ratings:

title	n

title	n
Pulp Fiction (1994)	34864
Forrest Gump (1994)	34457
Silence of the Lambs, The (1991)	33668
Jurassic Park (1993)	32631
Shawshank Redemption, The (1994)	31126
Braveheart (1995)	29154
Fugitive, The (1993)	28951
Terminator 2: Judgment Day (1991)	28948
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	28566
Apollo 13 (1995)	27035

While most films received a number of ratings of only one digits:

number_of_ratings	number_of_films
2	140
4	135
5	115
7	107
3	106
1	100
8	97
14	94
13	93
6	92

Average ratings per film

When we try to search for best and worst rated films, it is important to look at the number of ratings as well. The best rated films are the following ones:

title	mean_rating	number_rating
Satan's Tango (Sátántangó) (1994)	5.00	2
Shadows of Forgotten Ancestors (1964)	5.00	1
Fighting Elegy (Kenka erejii) (1966)	5.00	1
Sun Alley (Sonnenallee) (1999)	5.00	1
Blue Light, The (Das Blaue Licht) (1932)	5.00	1
More (1998)	4.75	8
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	4.75	4
Human Condition II, The (Ningen no joken II) (1959)	4.75	4
Human Condition III, The (Ningen no joken III) (1961)	4.75	4
Constantine's Sword (2007)	4.75	2

We can doubt well that the Hungarian Satantango is the best rated film. These films are art movies rated by just a couple of users, the rating is strongly biaised. We should select a minimum threshold. Let's try with n=20, the best films in this case are:

title	mean_rating	number_rating
Shawshank Redemption, The (1994)	4.457238	31126
Godfather, The (1972)	4.415085	19814
Usual Suspects, The (1995)	4.367142	24037
Schindler's List (1993)	4.363483	25777
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	4.321966	3255
Casablanca (1942)	4.319741	12507
Rear Window (1954)	4.316544	8825
Double Indemnity (1944)	4.315439	2403
Seven Samurai (Shichinin no samurai) (1954)	4.314119	5751
Third Man, The (1949)	4.313629	3265

Now this is much more realistic. We can observe the same effect for worst film ratings. The lowest averages are given to not well known films by just one or two users:

title	mean_rating	number_rating
Besotted (2001)	0.5000000	2
Hi-Line, The (1999)	0.5000000	1
Accused (Anklaget) (2005)	0.5000000	1
War of the Worlds 2: The Next Wave (2008)	0.6666667	3
SuperBabies: Baby Geniuses 2 (2004)	0.8032787	61
Hip Hop Witch, Da (2000)	0.8214286	14
Disaster Movie (2008)	0.9125000	40
From Justin to Kelly (2003)	0.9236111	216
Stacy's Knights (1982)	1.0000000	1
Dog Run (1996)	1.0000000	1

Applying the same threshold of n=20 we exclude the outliers for the worst ratinngs as well:

title	mean_rating	number_rating
SuperBabies: Baby Geniuses 2 (2004)	0.8032787	61
Disaster Movie (2008)	0.9125000	40
From Justin to Kelly (2003)	0.9236111	216
Pokémon Heroes (2003)	1.0096154	156
Carnosaur 3: Primal Species (1996)	1.1202532	79
Slaughterhouse 2 (1988)	1.1388889	36
Glitter (2001)	1.1671053	380
Gigli (2003)	1.1747159	352
Barney's Great Adventure (1998)	1.1752137	234
Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie) (2002)	1.1759657	233

When we create our model, we should take the number of ratings into account, using a **degressive weight for smaller number of ratings. This should normalize the film effect**. We can observe however that under average rated films received a smaller number of ratings: this could be taken to account. For negative film effects, the wheight applied should change more rapidly.

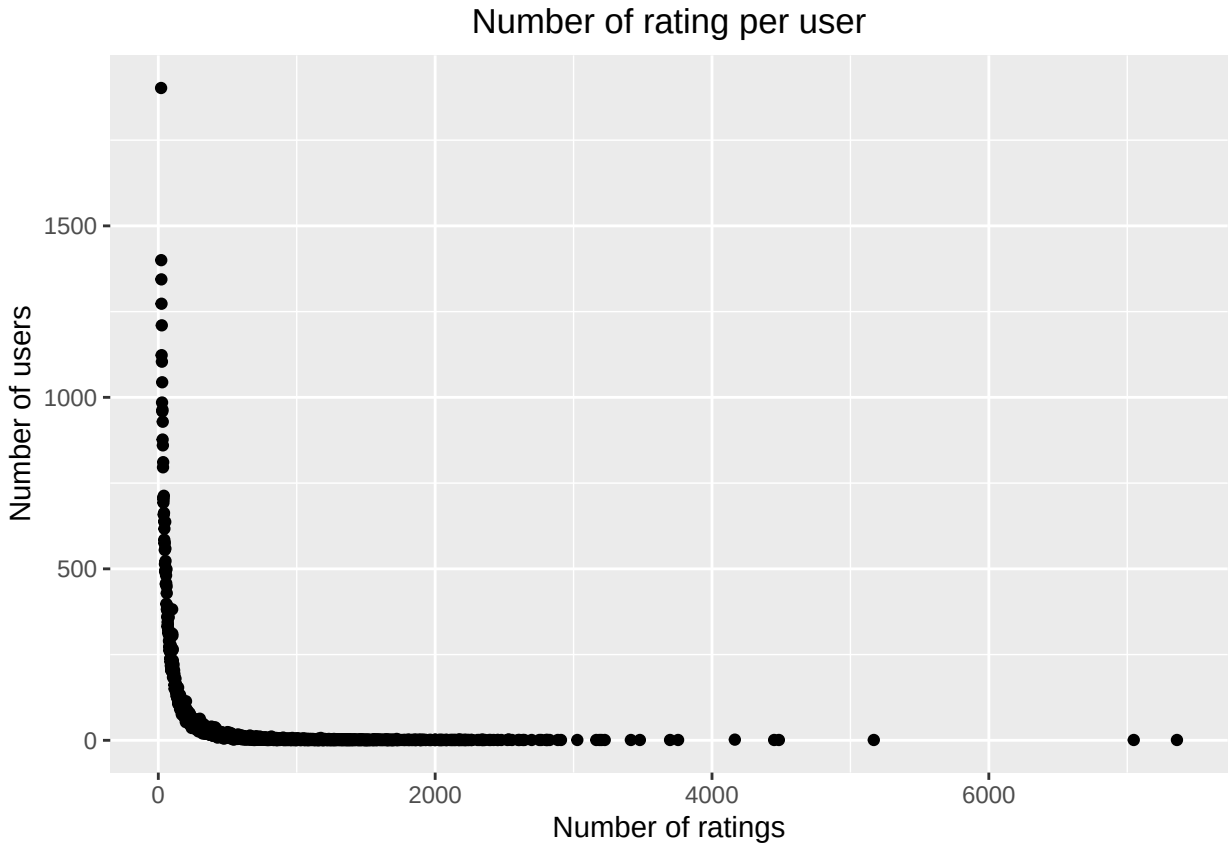
Ratings per user (the user effect)

A couple of users were really active, they rated a large number of films:

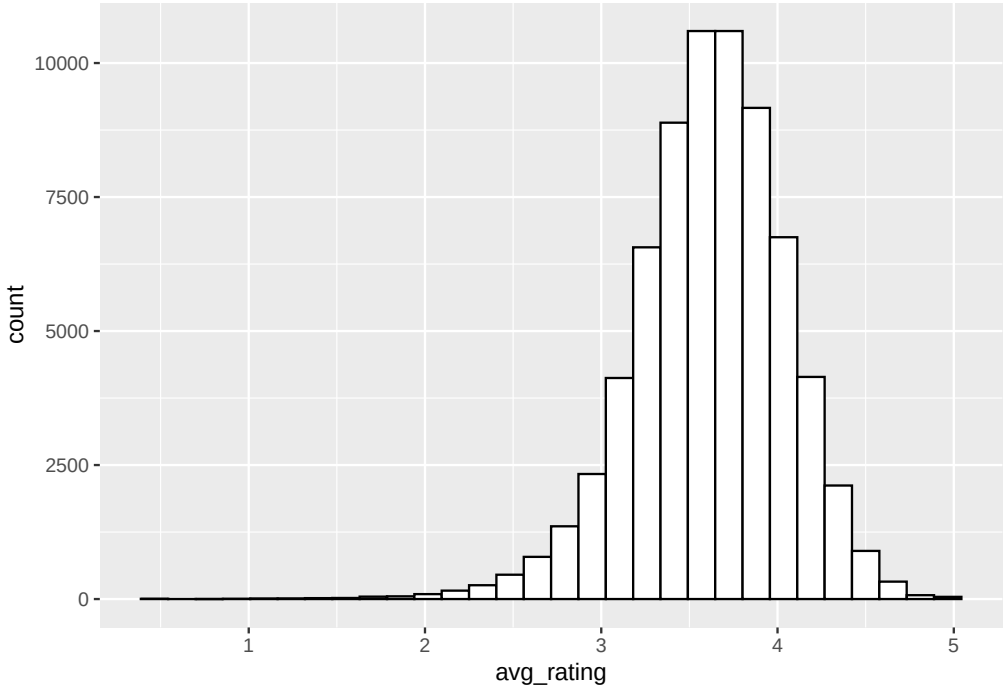
userId	n
59269	7359
67385	7047
14463	5169
68259	4483
27468	4449
3817	4165
19635	4165
63134	3755
58357	3697
27584	3479

While others gave only a couple of ratings, most users rating about 20 films:

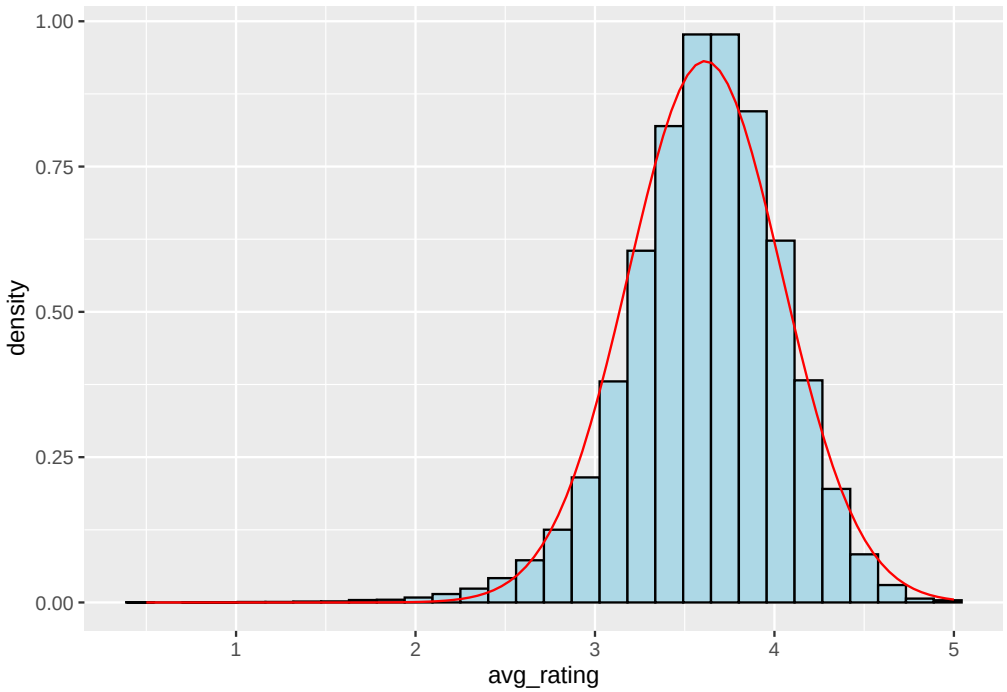
number_of_ratings	number_of_users
20	1902
21	1400
22	1344
23	1273
25	1210
24	1123
26	1104
28	1044
27	985
30	964



Some users tend to give better ratings then others, while some users are more kinky with their ratings:



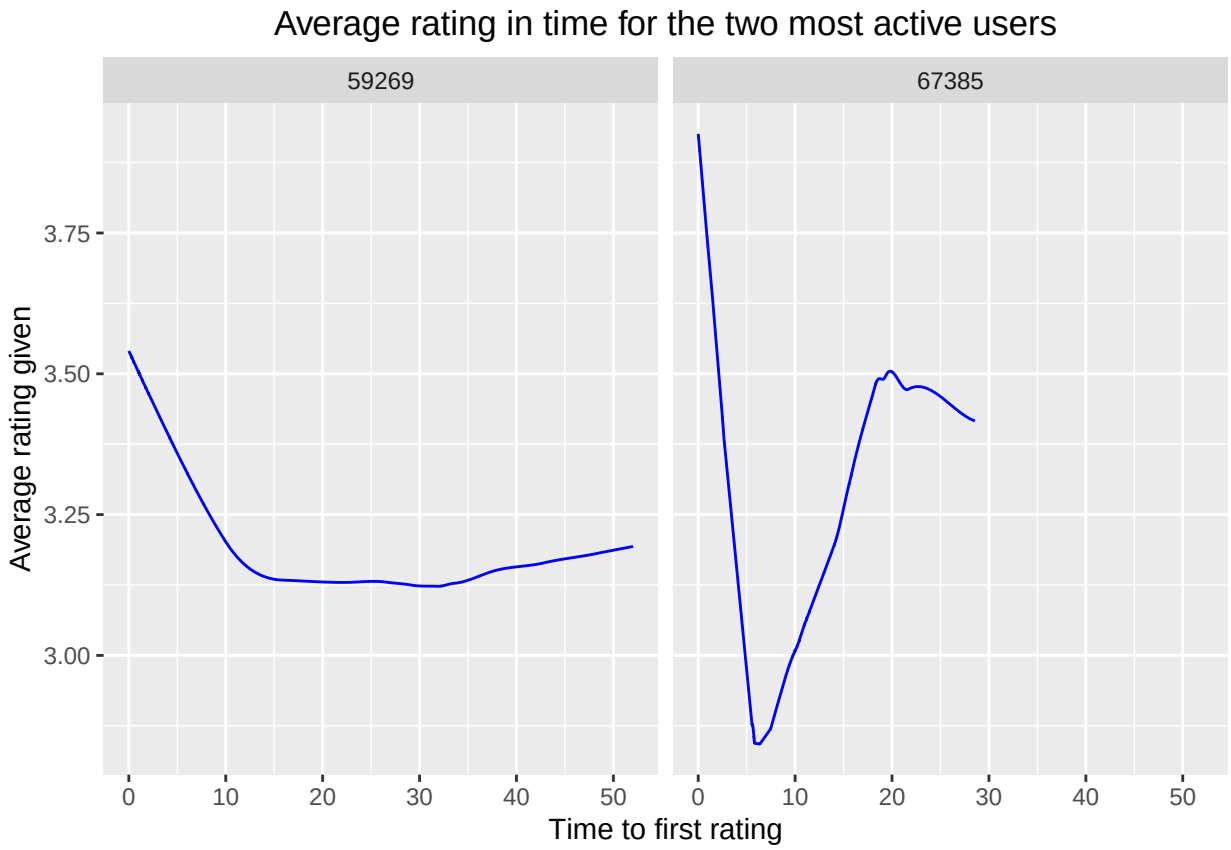
This distribution seems quite normal. We can verify it visually by drawing a normal density on the same graph:



People tend to give on average more average or slightly better than average ratings, kinky users (ratings about 2 - 2.5) are also more common then a normal distribution would suggest. The distribution seems negaively skewed and leptokurtic (kurtosis > 3). This can be easily verified:

skewness	kurtosis
-0.5607792	4.492671

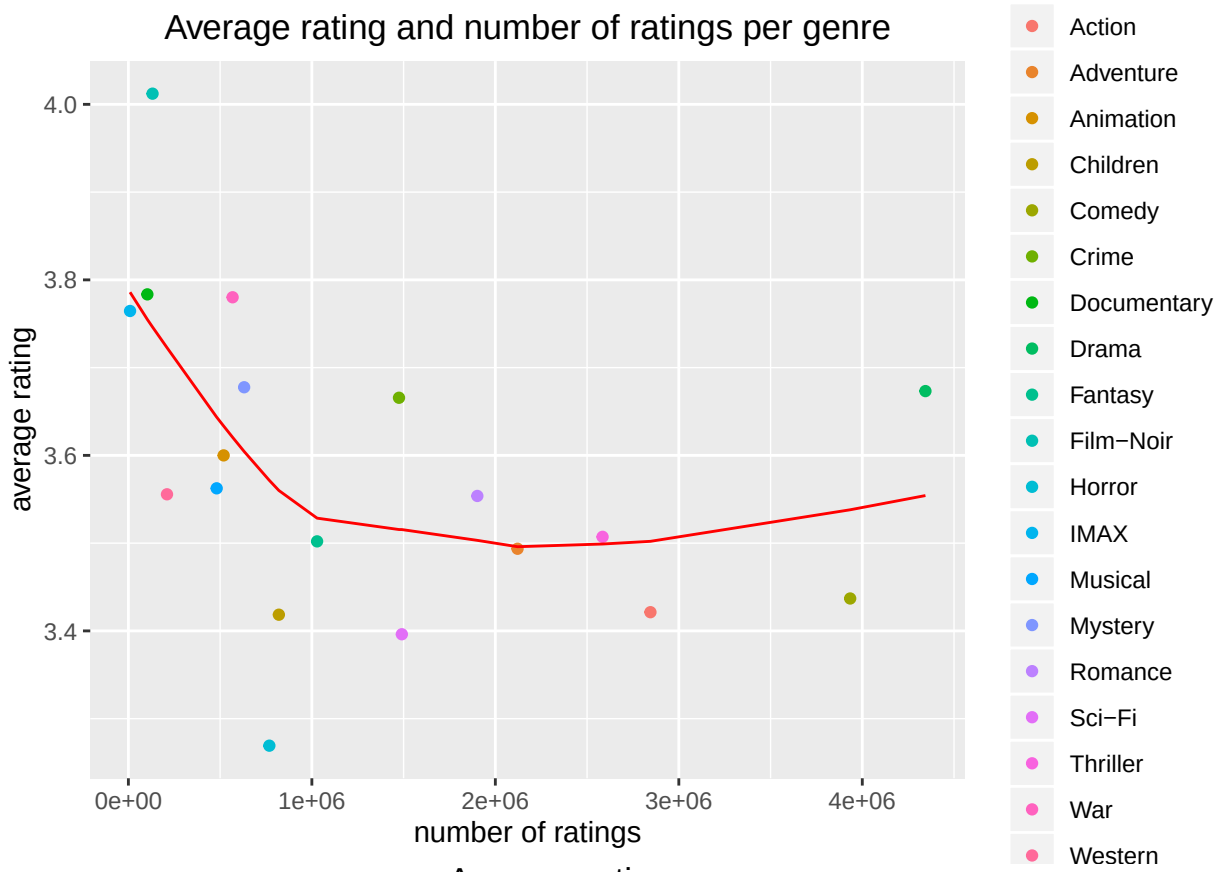
Let's examine whether users tend to become kinkier or softer with their ratings over time. We can visualize the ratings given by the two most active users over the sqrt of time in days:

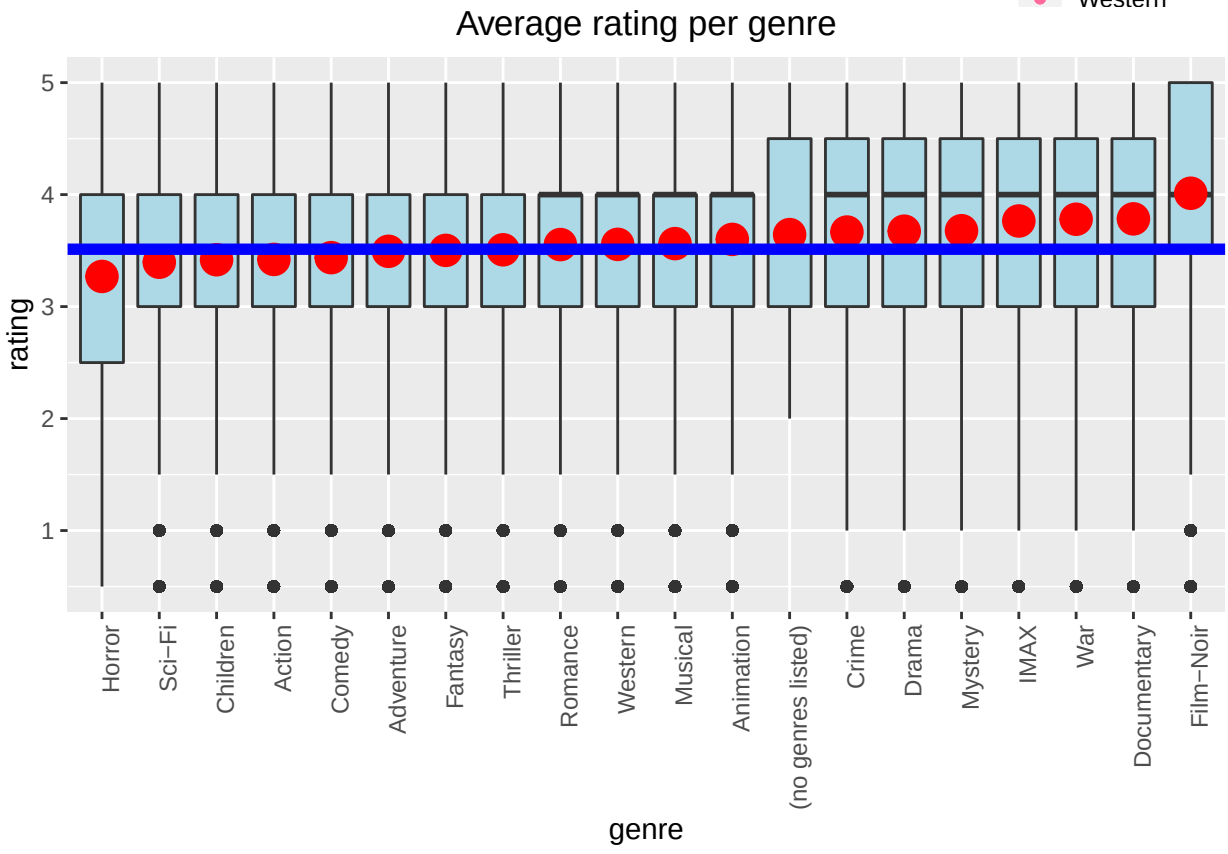


It seems that the **user effect in ratings is time dependent**. Users tend to change their given average ratings over time, so the user bias can be **modeled as a function of time:** $u(t)$.

Ratings per genre (the genre effect)

A possible genre effect can be included as well, but to treat **with caution as films can have more than one genre**.


Average rating and number of ratings per genre
average rating
number of ratings
4.0
3.8
3.6
3.4
0e+00
1e+06
2e+06
3e+06
4e+06
Action
Adventure
Animation
Children
Comedy
Crime
Documentary
Drama
Fantasy
Film−Noir
Horror
IMAX
Musical
Mystery
Romance
Sci−Fi
Thriller
War
Western


Average rating per genre
rating
genre
5
4
3
2
1
Horror
Sci−Fi
Children
Action
Comedy
Adventure
Fantasy
Thriller
Romance
Western
Musical
Animation
(no genres listed)
Crime
Drama
Mystery
IMAX
War
Documentary
Film−Noir

On the first graph above, we can clearly see, **genres with less ratings tend to be biaised upward**, while the **most popular genres are rated higher** as well. In the middle of the graph, we can observe a **regression to the mean**.

On the second graph we see a boxplot. The blue line is the global average, while the red dots inside the boxes show the average rating of each genre. This graph shows for example that horror is rated on average lower than the average 3.5 of all films, but **ratings are not consistent**, some films are rated 0.5 while others received a 5. Science-Fiction received more consistent ratings, the box is smaller.

The EDX and validation datasets

The EDX and validation datasets have the same structure as the MovieLens dataset. To create them we splitted randomly the MovieLens dataset keeping 90% of the lines in the EDX dataset and making sure that all the user and movie identifiers present in the validation set are also present in the EDX dataset.

```
## '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|Ac
```

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

```
## 'data.frame':    999999 obs. of  6 variables:
##  $ userId   : int  1 1 1 2 2 2 3 3 4 4 ...
##  $ movieId  : num  231 480 586 151 858 ...
##  $ rating   : num  5 5 5 3 2 3 3.5 4.5 5 3 ...
##  $ timestamp: int  838983392 838983653 838984068 868246450 868245645 868245920 1136075494 1133571200
##  $ title    : chr  "Dumb & Dumber (1994)" "Jurassic Park (1993)" "Home Alone (1990)" "Rob Roy (1995)"
##  $ genres   : chr  "Comedy" "Action|Adventure|Sci-Fi|Thriller" "Children|Comedy" "Action|Drama|Roman
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18096   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.467e+08
##  Median :35768   Median : 1827   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4108   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53621   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:999999      Length:999999
##  Class :character   Class :character
##  Mode  :character   Mode  :character
```

```
##
##
##
```

The model used

For our prevision I used a model with **regularization: the average rating is the starting point** that I rectifed with a **regularized user and movie effect. Lambda is the shrinkage factor**: for simplicity the same factor was used for the user and the movie effect.

To determine lambda, I executed a **10-fold cross validation on the edx dataset**: I created 10 sets of randomly selected values equivalent of 10% of lines from the dataset and used for optimization of lambda. The lambda resulting the smallest RMSE was selected, then the average was taken as the final lambda to use and the resulting RMSE of the training set.

The final RMSE was computed on the validation set, then it was compared to the RMSE obtained during training. If the two RMSEs were close, then we could not speak about overtraining of our algorithm and we could accept results.

To keep results the same at each execution, I chose to set the seed to 1 when training the algorithm for lambda.

For evaluation of results, we will compute the RMSE the following way:

```
#for final validation we sill use the RMSE (Residual Means Squares Errors)
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Results

Cross-validation to determine lambda

As a first step I executed a 10-fold cross validation on the edx training set to determine the optimal shrinkage factor, lambda:

```
#We will use a model with regularized movie + user effect (lambda is a degressive regularization factor
#We can use 10_fold cross validation without the validation set to determine the optimal lambda that mi
#We set the seed to receive a stable result
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 10, p = 0.1, list = FALSE)
results <- sapply(seq(1,10),function(i){

  train_set <- edx[-test_index[,i],]
  temp <- edx[test_index[,i],]

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

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

  lambdas <- seq(0, 10, 0.25)

  rmses <- sapply(lambdas, function(l){

    mu <- mean(train_set$rating)

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

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

    predicted_ratings <-
      test_set %>%
      left_join(b_i, by = "movieId") %>%
      left_join(b_u, by = "userId") %>%
      mutate(pred = mu + b_i + b_u) %>%
      pull(pred)

    return(RMSE(predicted_ratings, test_set$rating))
  })

  return(c(lambdas[which.min(rmses)],min(rmses)))
})

#results: the lambda to use will be the average lambda of the 10 sets
```

```r
#and the average RMSE needs to be compared with the RMSE of the validation set to see whether we overtr
results
mean(results[1,]) #this is the lambda to use: 4.775
mean(results[2,]) #this is the average RMSE on the training set: 0.8649748
lambda <- mean(results[1,])
```

```
##    Lambda      RMSE
## 1    5.00 0.8641362
## 2    4.75 0.8651805
## 3    4.50 0.8658975
## 4    5.00 0.8658025
## 5    4.75 0.8651570
## 6    5.00 0.8638235
## 7    5.00 0.8633388
## 8    4.50 0.8661761
## 9    4.75 0.8649827
## 10   4.50 0.8652531
```

```
## [1] 4.775
```

```
## [1] 0.8649748
```

Final RMSE

As a last step the model was evaluated with the validation dataset to report the final RMSE:

```r
lambda <- mean(results[1,])

#global average
mu <- mean(edx$rating)

#penalized movie effect
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

#penalized user effect
b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

#predicted ratings
predicted_ratings <-
  validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

#evaluation
RMSE(predicted_ratings, validation$rating)
```

With a lambda of 4.775 we obtained the **final RMSE of 0.8648198 on the validation dataset**.

Limitations

The model I used is really simple but provides decent results. The final RMSE obtained means that on average we made an error of less than 1 when we predicted the rating of each user for the validation set. For comparison the RMSE obtained when predicting the average rating for each film is 1.0612018.

Possibilities for improvement are the following:

- instead of using a constant user effect, we can use a time dependent user effect
- use different shrinkage factor (lambda) for user and film effects, train them independently
- use different shrinkage factor (lambda) for over and under average rated films, train them independently
- use PCA and SVD to create user specific effects.