Recommendation System for: MovieLens

Rolf Beutner, Germany, Bamberg

04/15/2021 18:50

Contents

1 Overview

This report is related to the MovieLens Project of the HarvardX: PH125.9x Data Science: Capstone.

It consists of seven parts:

It starts with this **Overview**, followed by a short **Introduction**.

A **Summary** section describes how to download and prepare the datasets **edx** and **validation**.

Next section **Investigation** describes the performed exploratory data analysis which provided an overview of the data and first approaches to optimize the machine learning algorithm.

In the **Methods and Analysis** section, **train**- and **test**-datasets are generated and a machine learning algorithm is stepwise developed that predicts movie ratings based on that sets.

The **Results** section explains the final model and applies it to the validation-dataset.

The report ends with a **Conclusion** on the findings and possible further steps.

2 Introduction

Today machine learning is becoming more and more important in business- and everyday life. It is used for business i.e. to explore data about customer bases, for single user it can personalize experiences like this for the selection of the next movie. Machine learning helps extract useful insights from data. Effective machine learning algorithms are used to obtain these insights within a short period of time.

In this project we aim to predict movie ratings for users based on a large dataset and develop an effective machine learning algorithm, suggest movies based on previous preferences of other uses, and ratings of similar movies.

Therefore a linear model is trained to generate these predictions. To evaluate the quality of the predictions we use the Root Mean Square Error (RMSE) to calculate the distance between the predicted ratings and the actual ratings.

3 Summary

The following dataset is used from the following directory:

- MovieLens 10M dataset
- ml-10m.zip

This dataset downloaded and then splitted into 2 subsets:

- **edx** - a subset to work with and to develop the models, and
- **validation** - a subset to test the final model

The **edx** dataset is then split further into 2 subsets:

- **trainset** - for training the models
- **testset** -for testing the models

Hint: The code is included in the RMD file but explicitly switched off (echo=FALSE).

3.1 Download

The embedded code downloads the data and creates the 2 subsets **edx** and **validation**.
It is instrumented for switching between:

- download and separate datasets from grouplens or
- load from local file system
- size of datasets:
  - large (the final 10M target dataset) or
  - small dataset to test the program (1M dataset, same format, 10% of size).

The final version will download AND use the 10M dataset of course.

4 Investigations

4.1 Exploratory analysis

The grouplens-dataset was split in the ratio 90-10 into the **edx**- and **validation**-dataset.
For the investigations here the **edx** dataset is taken.
For the development in the next chapter then the **trainset** and **testset** will be taken
The training set **edx** has 9,000,055 entries with 6 columns.

```
## [1] 9000055       6
```

Similarly, the test set **validation** has 999,999 entries and 6 columns.
The column information is shown below.

```
## [1] 999999      6
```

In the following the investigations are done with the **edx** dataset. The **validation** dataset will be used when a model is created and shall be tested.

The ratings in the edx dataset were applied by nearly 70.000 user and more than 10600 movies

no_users	no_movies
69878	10677

4.1.1 Quick preview of "edx"

Six columns with the following to be investigated more:

- rating - the center of interest - given by user with "userId" for movie with "movieId"
- genres - single pipe-delimited string containing the different genre categories 1:n
- timestamp - needs to be converted
- title - has release year as appendix, worth to split from title

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

4.1.2 Rating of movies

There is a broad set of ratings per movie - at the top *Pulp Fiction* with more than 31,000 ratings, here grouped by title and sorted descending by no. of ratings:

title	no_of_ratings
Pulp Fiction (1994)	31362
Forrest Gump (1994)	31079
Silence of the Lambs, The (1991)	30382
Jurassic Park (1993)	29360
Shawshank Redemption, The (1994)	28015

and on the end: 126 titles with just one single rating, here grouped by title, filtered by "no_of_ratings==1" and the result counted.

```
## [1] 126
```

Further checks:

- All movies are categorized with genres (no na(genres)):

```
## [1] 0
```

- There is no movie which has been never rated (no na(rating)):

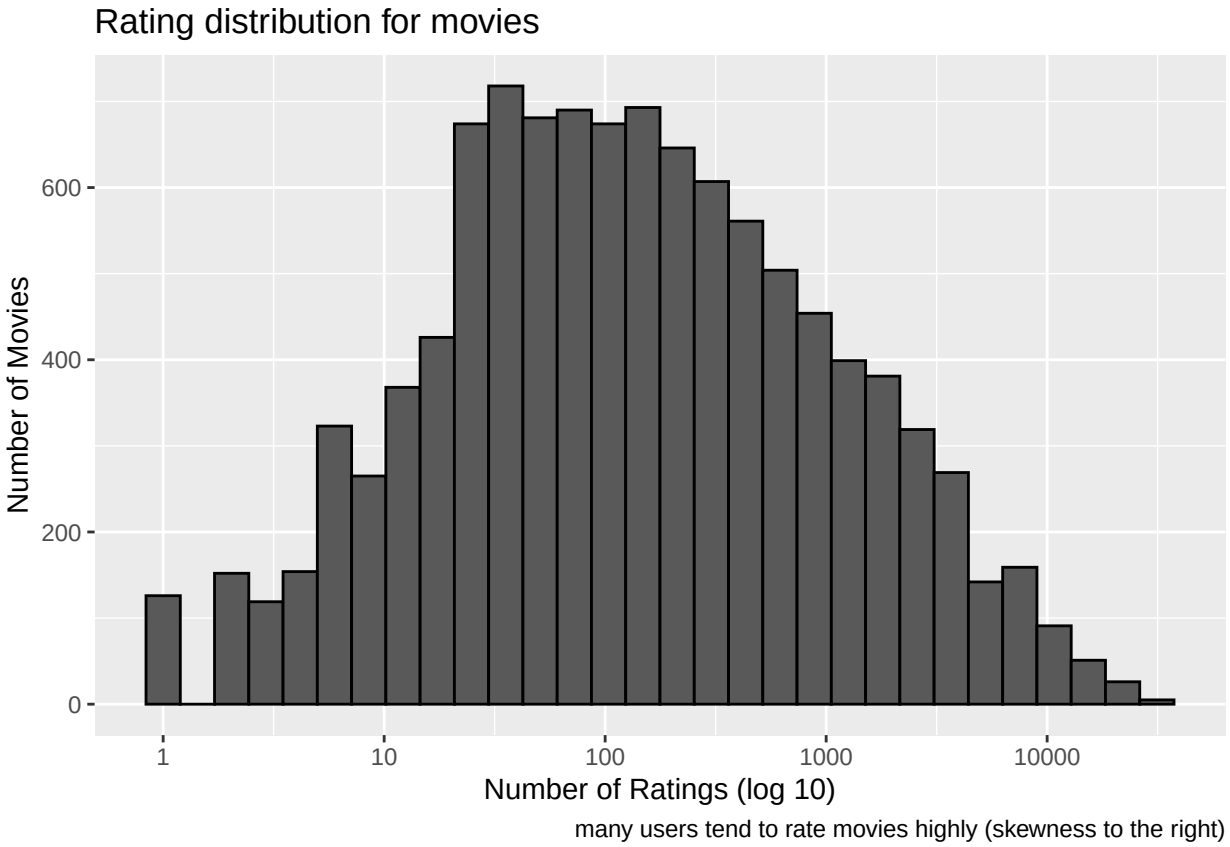
```
## [1] 0
```

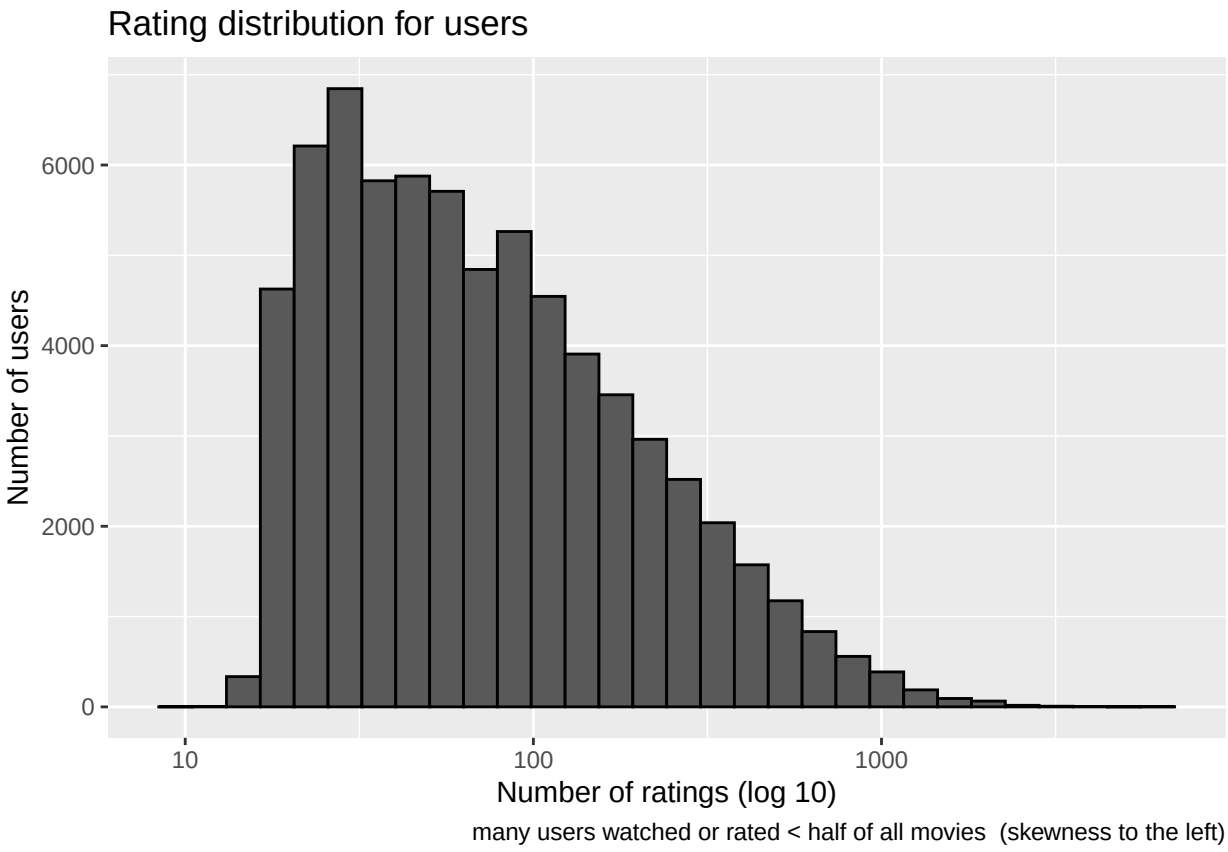
4.1.3 Movies vs. Users rating - distribution, histograms and crosstab

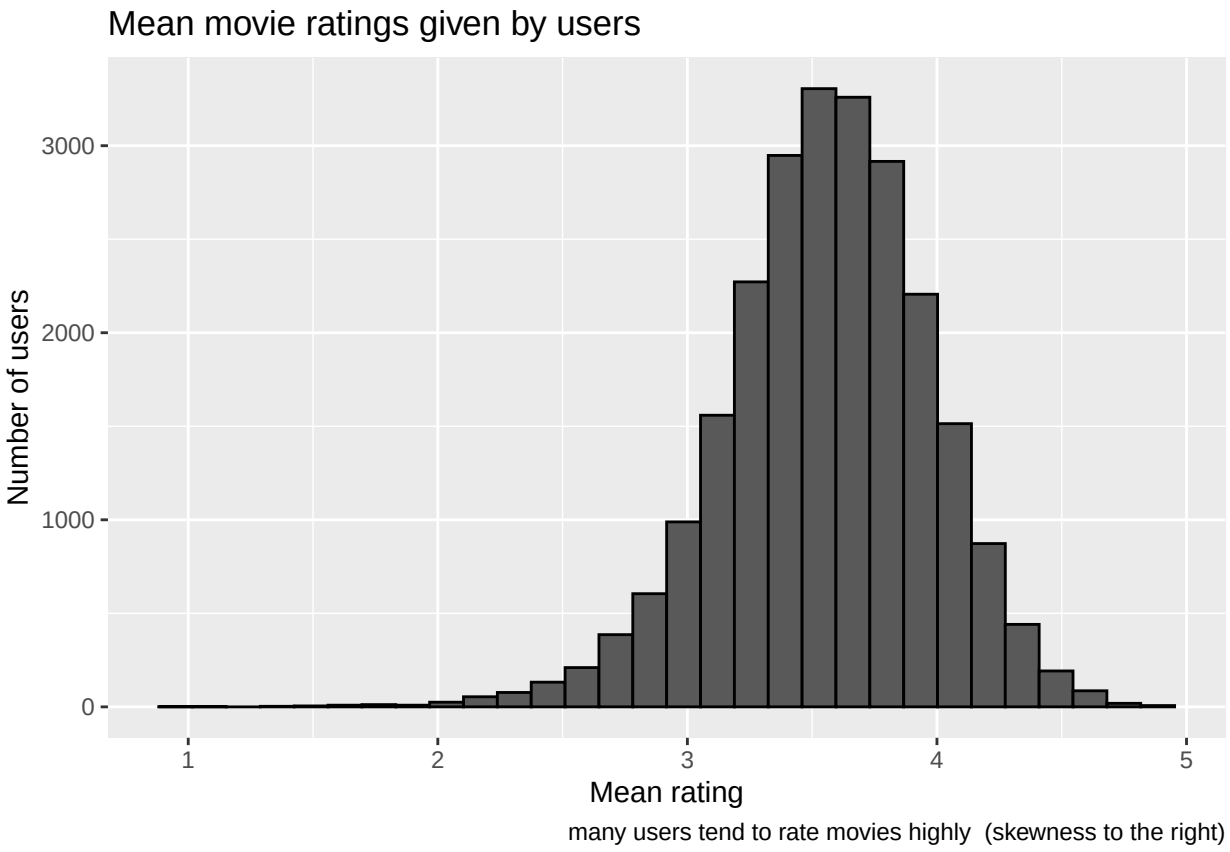
We have the following distribution of the ratings and the histograms of ratings related to users and movies. So there are users who rate more than others and also movies which are rated more than others. When we put both in a cross tab to see what user has rated what movie we see a sparse - but very big matrix.

4.1.3.1 Histograms The mean rating distribution show slightly skewness - that can be used as influencing factor for our predicting algorithm.

The distribution of user ratings shows that many users have watched or rated less than half of all movies.

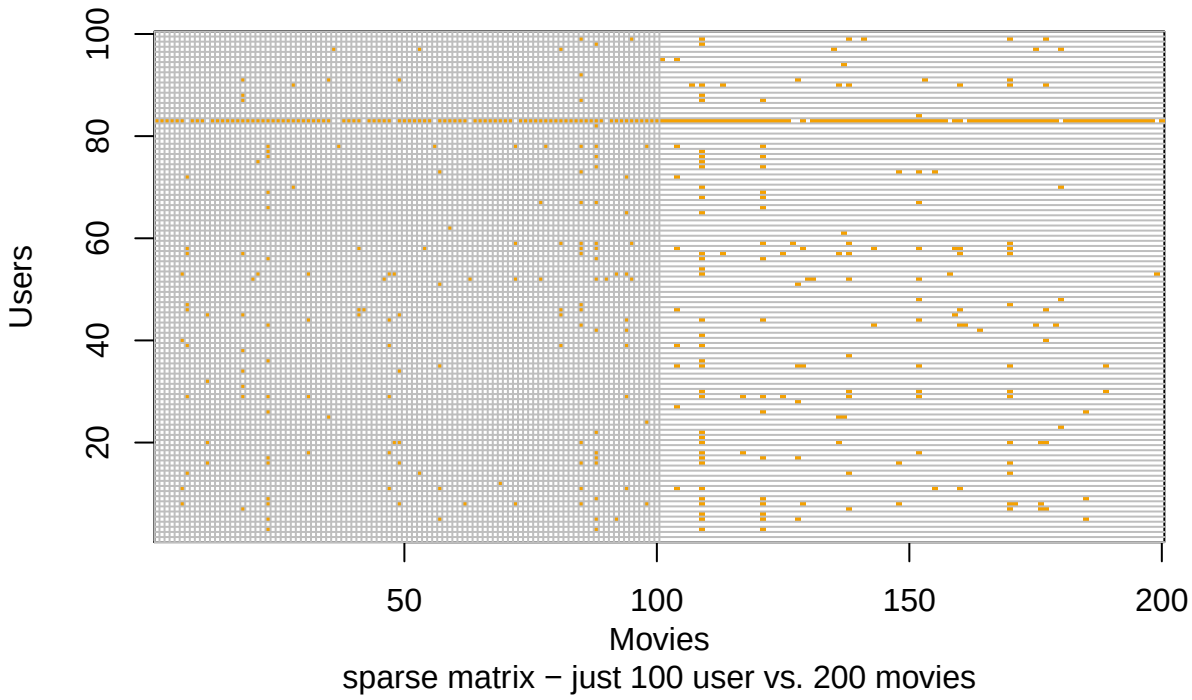



Rating distribution for users
Number of users
6000
4000
2000
0
10
100
1000
Number of ratings (log 10)
many users watched or rated < half of all movies (skewness to the left)



4.1.3.2 Crosstab/Heatmap user vs. movies When we put the relation of user vs. movies in a cross tab to see what user has rated what movie, the result a sparse, but very big matrix - here only shown a small subset of 100 user (rows) and 200 movies (columns):

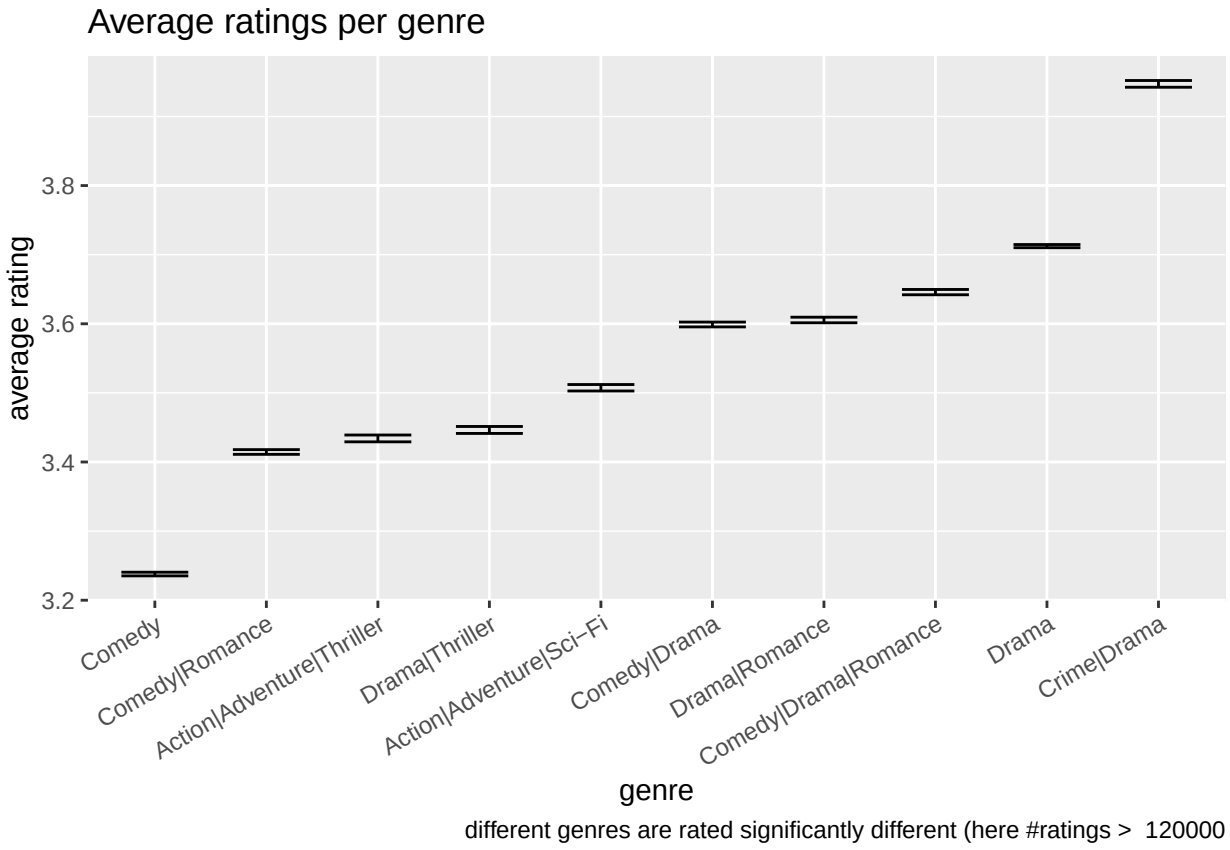
All nonzero values are represented by a dot. Empty spaces means no ratings were given by the user.



4.1.4 Genres

The genres have a broad set too - here those where the movies have **more than 120.000 ratings**. The average rating per genre is significantly at different levels.

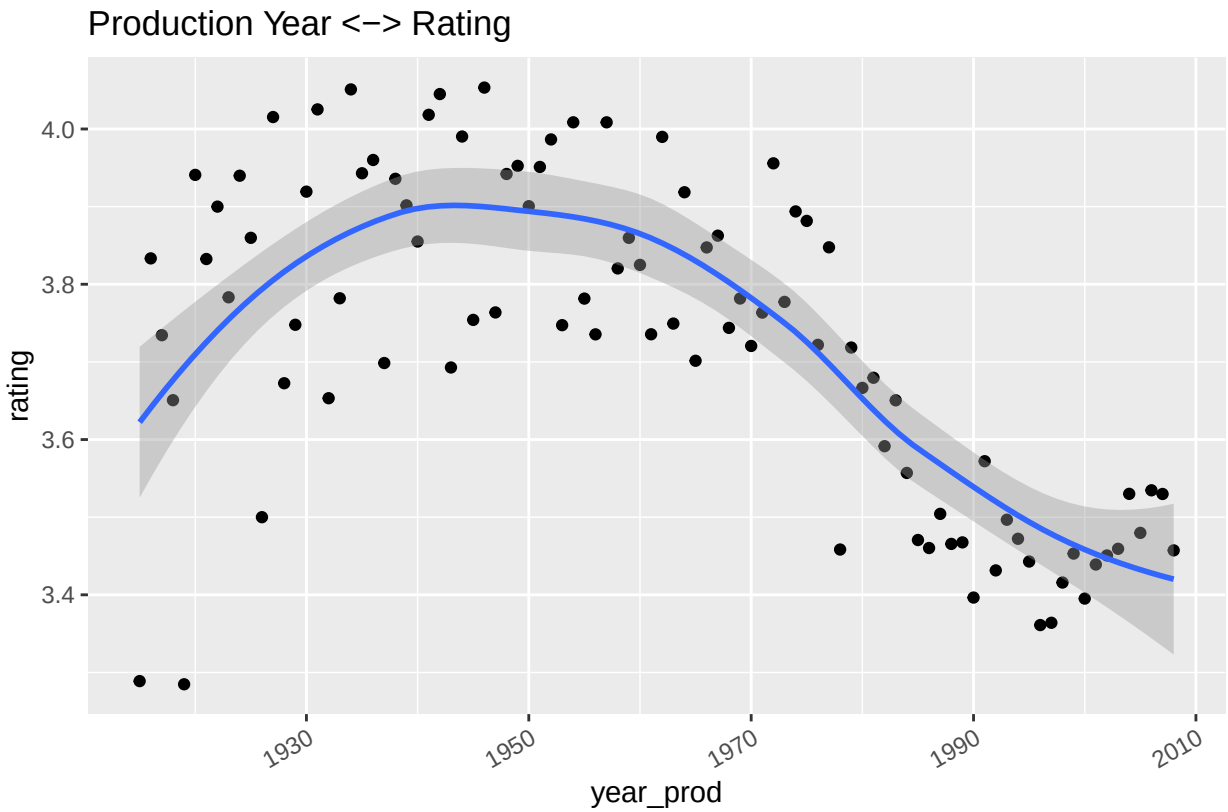
So as one can see the average ratings for **Crime/Drama** are substantially higher than for **Comedy**. **This will be considered later as tuning parameter**.



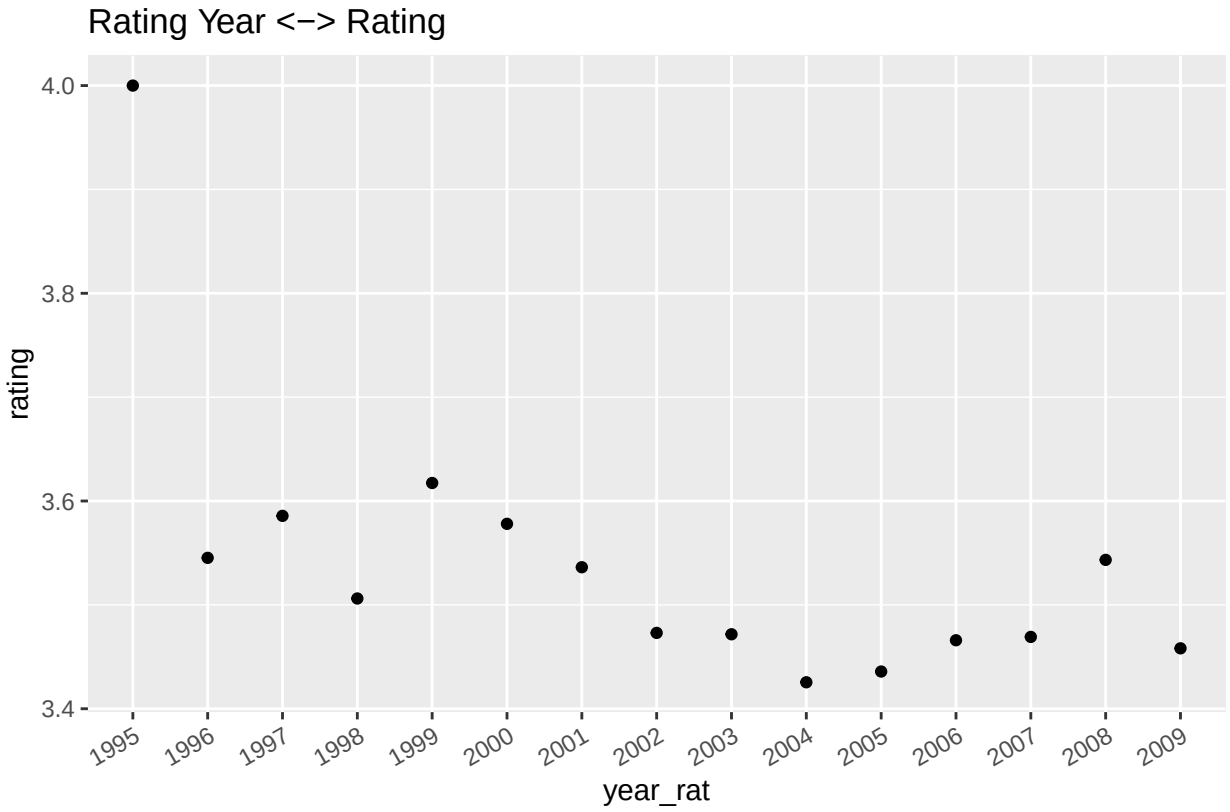
4.1.5 Timestamp and year tag

We put some effort in investigations around the two timestamps given by the data, the unix timestamp of the rating and the year of production / release of the movie and we check against the genres to detect patterns. The date we need later as join criterion.

We see that older movies have better ratings - maybe an approach to penalize. The second graph shows that the date of the rating has not much effect, see the x- and y-scale: it does just cover a short time period and has just a small offset to the median around 3.5.

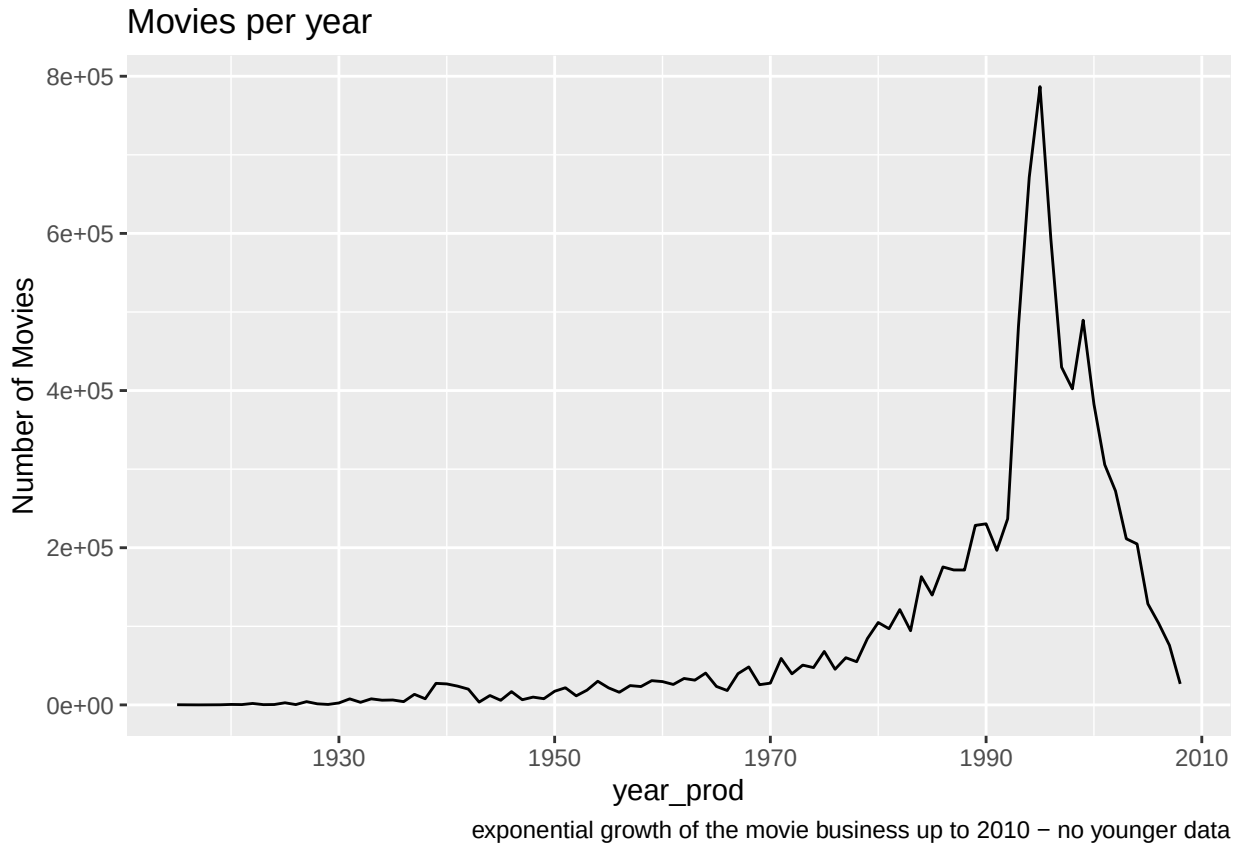


Older movies have better ratings − maybe an approach to penalize



4.1.6 Movies per year

When we check the frequency of movies per year we see an exponential growth of the movie business up to 2010 and then a sharp decrease, does not mean the movie business went bancrupt but lack of data, in the dataset were no younger data.



4.1.7 Genres per year

We could separate the genres-string into own attributes / features. We will check later - first analysis showed: the **1:n relation blows up to more than 23 mio rows** and the genres per year seem to follow the movies per year. So here must be investigated more work, i.e. normalize per movie per year.

5 Methods and Analysis

The **edx**-dataset is very big as shown above.

To find out the **best recommendation for a new movie** based on movies or people "close" to the asking person could lead to the idea to use the methods for solving a "nearest neighbor problem". My attempt with a nearest neighbor algorithm failed due to lack of resources.

So the approach recommended in the book for large datasets was taken: **a linear model**.

https://rafalab.github.io/dsbook/large-datasets.html#recommendation-systems

As prerequisite to develop the machine learning models we will create now the **train**- and **test**-datasets from the **edx** dataset.

5.1 Create Train- and Test-dataset

The **edx**-dataset is split in the ratio 90-10 into **trainset** and **testset** for the further development of the algorithm.

```r
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
trainset <- edx[-test_index,]  # 90%
temp     <- edx[test_index,]   # 10%

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

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

rm(temp)
```

5.2 Linear Models

The RMSE function as described in chapter 33.7.3 is used for measuring the accuracy of different approaches.

5.2.1 Model 0: Naive model

Following the book we start with the simplest model, the naive approach: We predict the same rating for all movies independent of user, movie or genre. The estimate that minimizes the RMSE is the least squares estimate of mu and, in this case, is the average of all ratings.

Result: 3.512 (`mean(trainset$rating)`).

(using the code of Course Section 6: Model Fitting and Recommendation Systems / 6.2: Recommendation Systems)

```
## [1] 3.512456
```

If we naivly predict all unknown ratings with mu_hat we obtain the following RMSE.

We collect the results in the **rmse_results** data structure

```r
naive_rmse <- RMSE(testset$rating, mu_hat)
naive_rmse
```

```
## [1] 1.060054
```

```r
predictions <- rep(2.5, nrow(testset))
RMSE(testset$rating, predictions)
```

```
## [1] 1.465938
```

```r
predictions <- rep(3  , nrow(testset))
RMSE(testset$rating, predictions)
```

```
## [1] 1.177464
```

```
rmse_results <- tibble(method = "Just the average", RMSE = naive_rmse)
rmse_results
```

```
## # A tibble: 1 x 2
##   method            RMSE
##   <chr>            <dbl>
## 1 Just the average  1.06
```

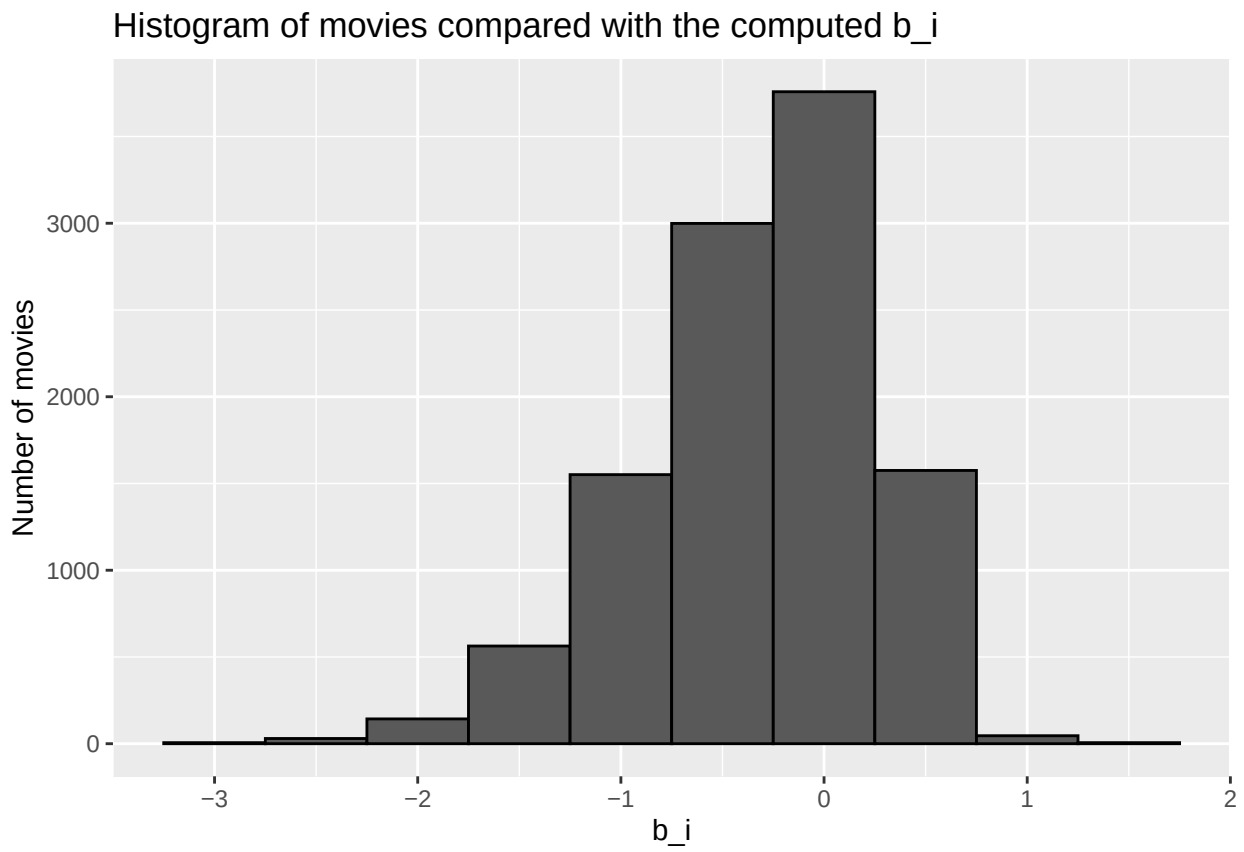
```
rm(predictions) # cleanup, is big table!
```

5.2.2 Model 1: Movie effect - multi-variate model

Now we want to improve the model and add to it movie effects. Different movies are rated differently as shown in previous chapter. This is done by adding an error term b_i to represent average ranking for movie i:

$$Y_i = \mu + b_i => b_i = Y_{i,u} - \mu \quad (1)$$

We can see that these estimates vary substantially:



Check how much it improves the model, we test it with the validation-set, compare it by RSME with reality and add the following result for RMSE to the summary table.

```
## [1] 0.9429615
```

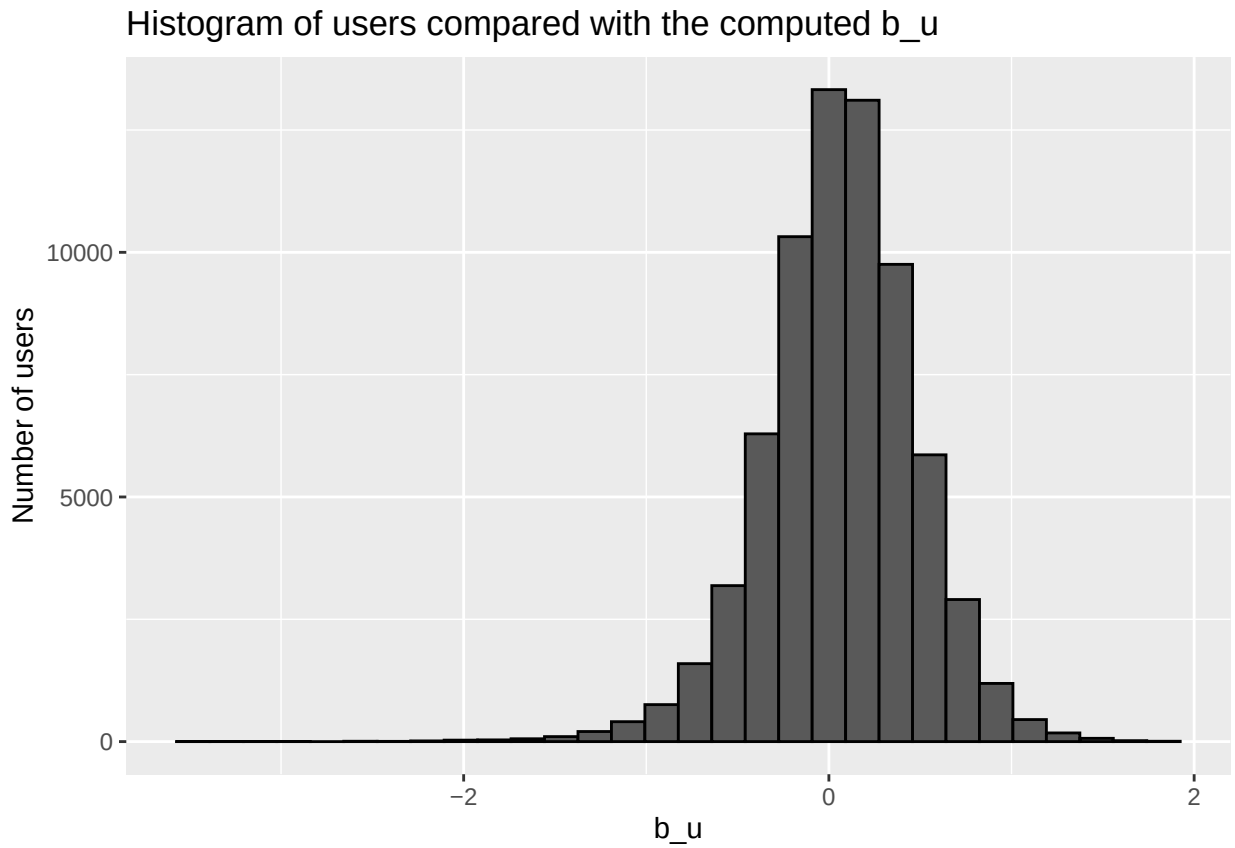
method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615

5.2.3 Model 2: User effect - multi-variate model

Add the user bias term b_u to improve the model. This minimizes the effect of extreme ratings made by users that love or hate every movie

$$Y_{i,u} = \mu + b_i + b_u => b_u = Y_{i,u} - mu - b_i \quad (2)$$

We can see that these estimates changes indeed:



Check how much it improves the model, we test it with the validation-set, compare it by RSME with reality and add the following result for RMSE to the summary table.

```
## [1] 0.8646843
```

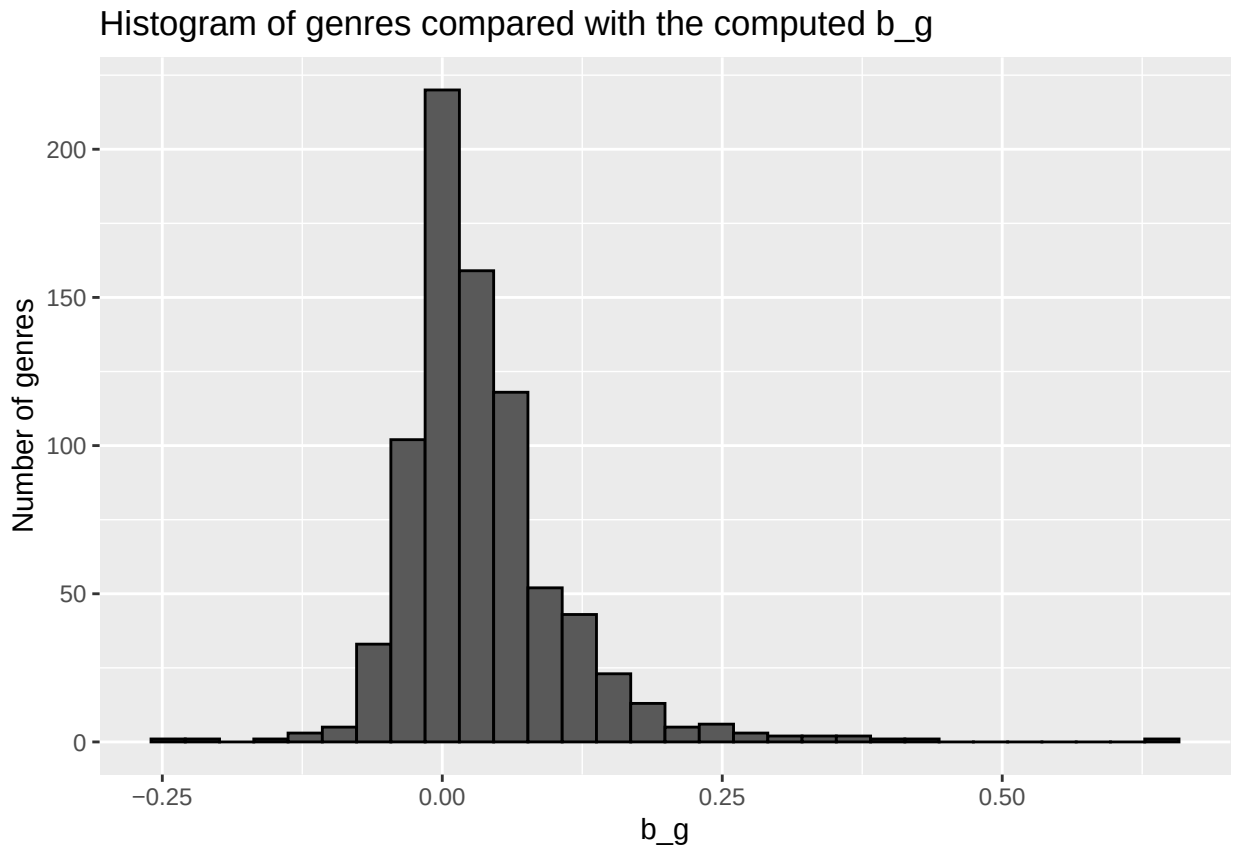
method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646843

5.2.4 Model 3: Genre effect - multi-variate model

Next we add a genres bias term b_g to possibly further improve our model. This term considers that same genres get similar ratings, some genres tend to get better ratings than others. The updated model is:

$$Y_{i,u,g} = \mu + b_i + b_u + b_g => b_g = Y_{i,u,g} - mu - b_i - b_u \quad (3)$$

We can see that these estimates vary slightly:



Check how much it improves the model, we test it with the validation-set, compare it by RSME with reality and add the following result for RMSE to the summary table.

```
## [1] 0.8643241
```

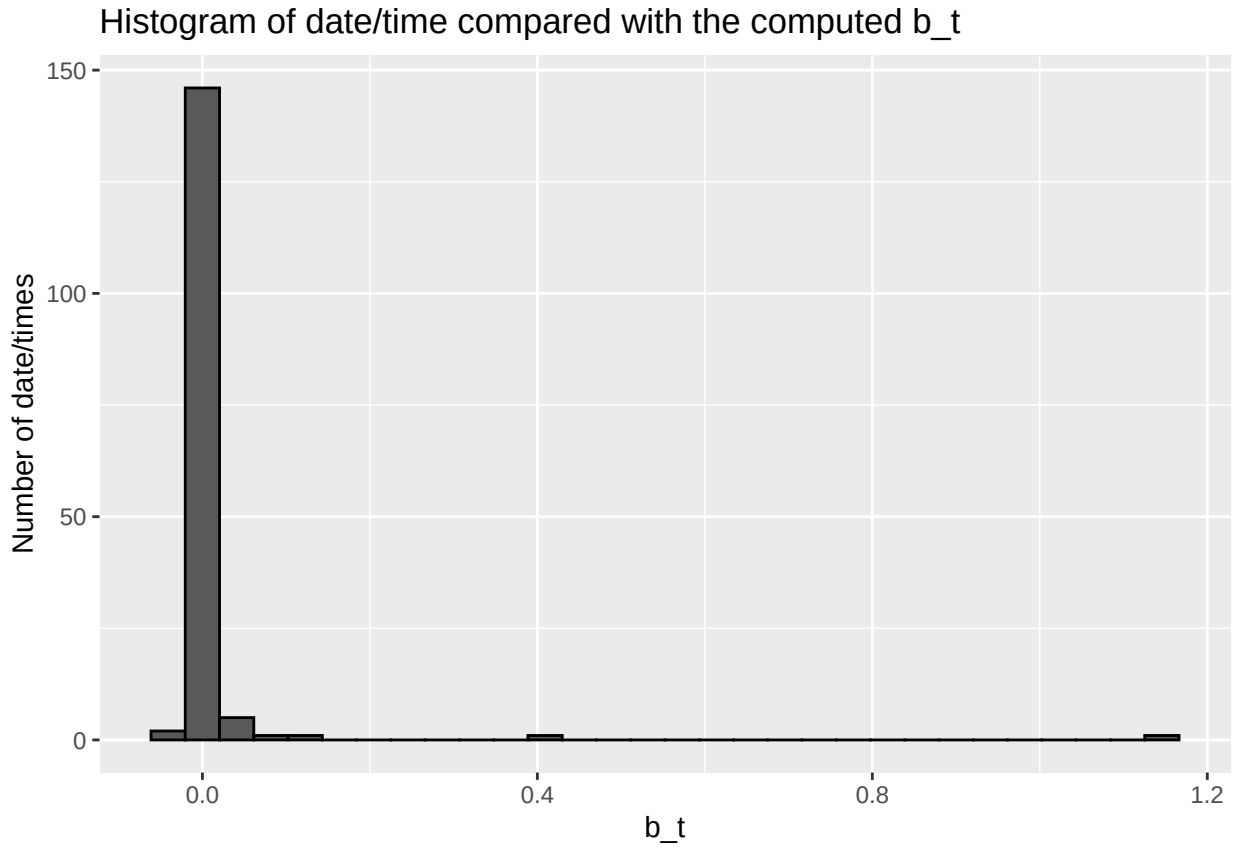
method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646843
Movie + User + Genre Effects Model	0.8643241

5.2.5 Model 4: Time effect - multi-variate model

Next we add a time bias term b_t to possibly further improve our model. This term considers that at similar times ratings are similar, as per spirit of that time, as fad. The updated model is:

$$Y_{i,u,g,t} = \mu + b_i + b_u + b_g + b_t => b_t = Y_{i,u,g,t} - \mu - b_i - b_u - b_g \quad (4)$$

We can see that these estimates vary very slightly, tend just to show outliers:



That histogram above does not show any improvement a time based term could offer, it seems it is showing just noise but no skewness. But we apply it to the model and check.

We have to prepare the testset set too with the "date" to be able to join, then check how much it improves the model, we test it with the testset-set, compare it by RSME with reality and add the following result for RMSE to the summary table.

```
## [1] 0.8642838
```

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646843
Movie + User + Genre Effects Model	0.8643241
Movie + User + Genres + Time Effects Model	0.8642838

Indeed it does not improve the model substantially, only at 4th place behind the decimal point. We will not use this parameter further.

5.3 Regularized approach

This section is following the book, chapter for **regularization**

https://rafalab.github.io/dsbook/large-datasets.html#regularization

Regularization allows us to penalize large estimates from small samples. The idea is to add a penalty for large values of movie/user/genre (b_i/b_u/b_g) to the sum of squares that we minimize, means we penalize big distances from mean by an influencing parameter.

A more precise estimate of b_u, b_i and b_g treats them symmetrically by solving a least squares problem

Lambda is this tuning parameter for the penalty and by cross validation we get the optimal(=minimal) value. We test `lamda <- seq(from=0, to=10, by=0.25)`

We try the following in 3 approaches:

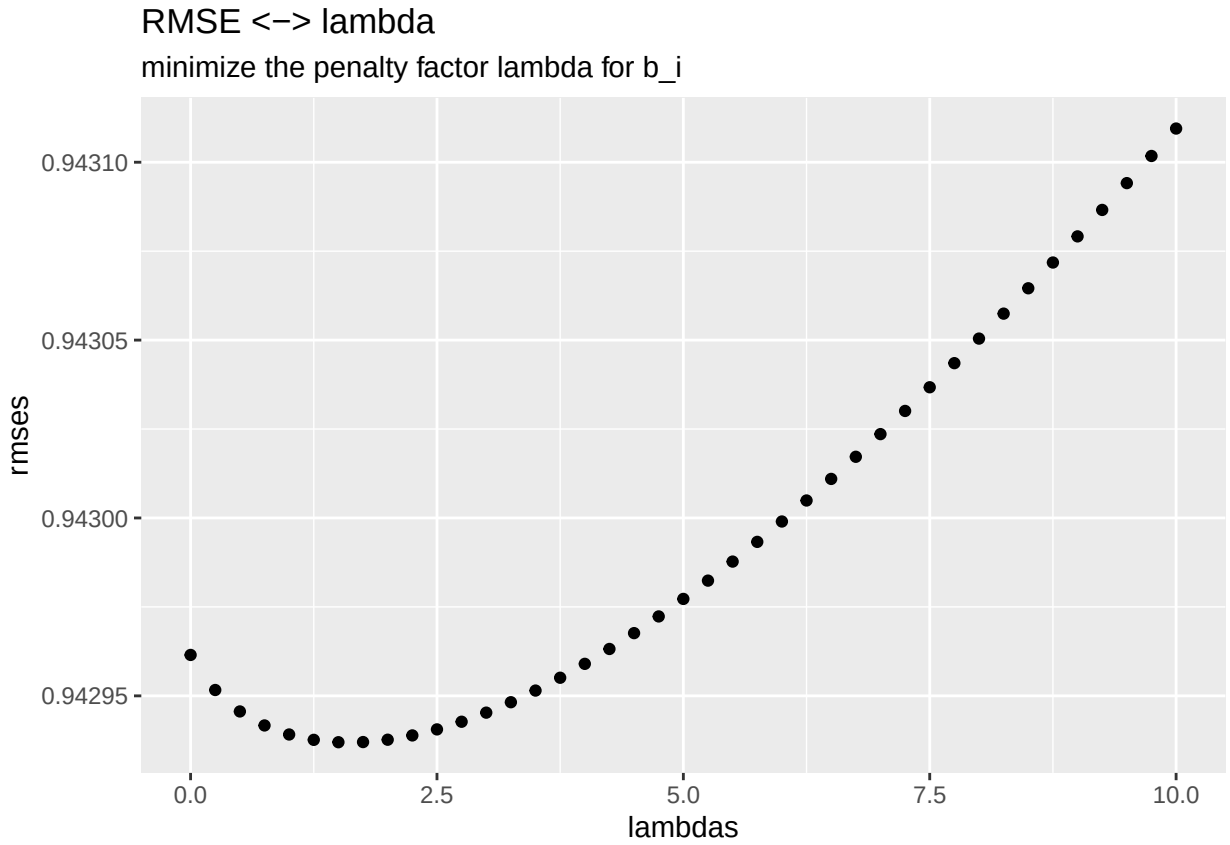
- starting just with b_i (movie effect)
- then add b_u (user effect)
- and finally add b_g (genres effect)

5.3.1 Model 5: Regularized movie effect

We use the movie effect b_i in our model. We do a loop with lambda and check where the result is minimal.

$$\frac{1}{N}\sum_{u,i}(Y_{u,i}-\mu-b_i)^2+\lambda(\sum_i b_i^2)$$

We can see the typical curve and where it has a minimum - quick plot of RMSE vs lambdas



and the minimum / optimal lambda is:

```
## [1] 1.5
```

which leads to RSME of the following, which we add to the result table:

```
## [1] 0.942937
```

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646843
Movie + User + Genre Effects Model	0.8643241
Movie + User + Genres + Time Effects Model	0.8642838
Regularized Movie Effects Model	0.9429370

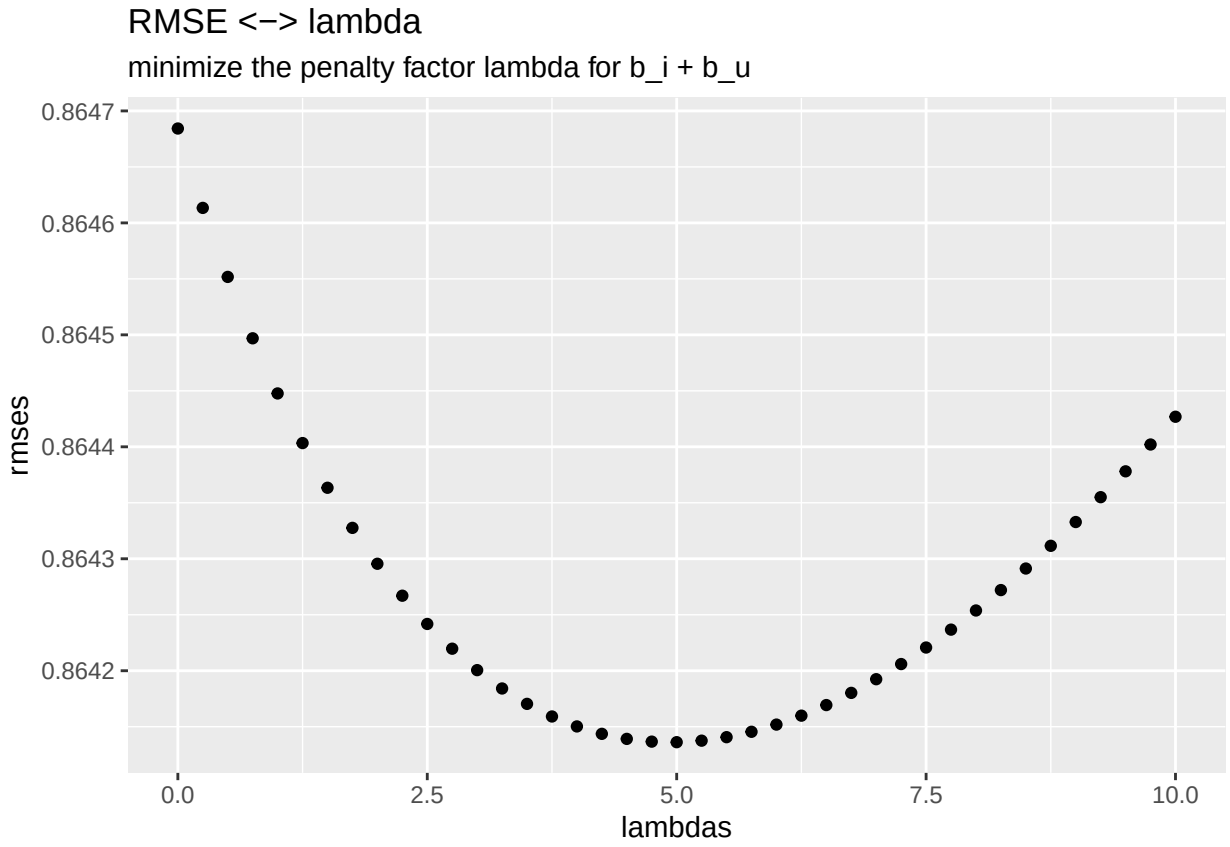
5.3.2 Model 6: Regularized movie and user effect

We use additionally to the movie b_i the user b_u in our model. We do as before a loop with lambda and check where the result is minimal.

$$\frac{1}{N}\sum_{u,i}(Y_{u,i}-\mu-b_i-b_u)^2+\lambda(\sum_i b_i^2+\sum_u b_u^2)$$

This is taking time!!!

We can see the typical curve and where it has a minimum - quick plot of RMSE vs lambdas



and the minimum / optimal lambda is:

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

which leads to RSME of the following, which we add to the result table:

```
## [1] 0.8641362
```

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646843
Movie + User + Genre Effects Model	0.8643241
Movie + User + Genres + Time Effects Model	0.8642838
Regularized Movie Effects Model	0.9429370
Regularized Movie + User Effects Model	0.8641362

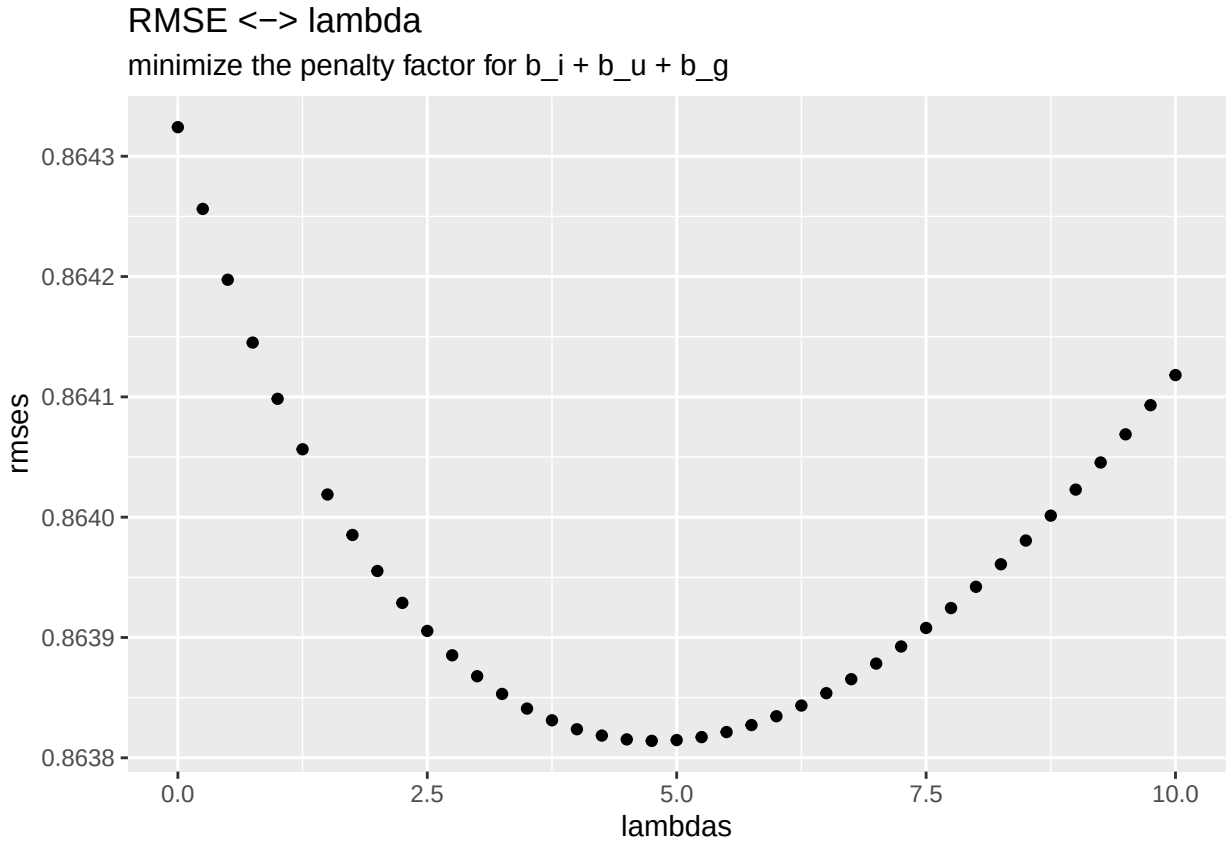
5.3.3 Model 7: Regularized movie, user and genres effect

This is finally the "full blown up model" - we use additionally the genres b_g. We do as before a loop with lambda and check where the result is minimal.

$$\frac{1}{N}\sum_{u,i,g}(Y_{u,i,g} - \mu - b_i - b_u - b_g)^2 + \lambda(\sum_i b_i^2 + \sum_u b_u^2 + \sum_g b_g^2)$$

WARNING: This needs time!!!

We can see the typical curve and where it has a minimum - quick plot of RMSE vs lambdas



and the minimum / optimal lambda is at:

```
## [1] 4.75
```

which leads to RSME of the following, which we add to the result table:

```
## [1] 0.8638141
```

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646843
Movie + User + Genre Effects Model	0.8643241

method	RMSE
Movie + User + Genres + Time Effects Model	0.8642838
Regularized Movie Effects Model	0.9429370
Regularized Movie + User Effects Model	0.8641362
Regularized Movie + User + Genres Effects Model	0.8638141

5.3.4 Final lambda

This value will be used in the next section for the final model.

```
final_lambda = lambda
final_lambda
```

```
## [1] 4.75
```

6 Results

6.1 Execute final model 7 now with the original data

Here is the code explicitely shown to demonstrate how the algorithm works:

- in 4 steps.
- choose minimized **final_lambda** from best model 7
- train with the **edx** dataset (step 1-3)
- test with the **validation** dataset (step 4)

```
# 1. step: compute regularize movie bias term
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating             - mu)/(n() + final_lambda))
b_i[is.na(b_i)] <- 0 # remove NA

# 2. step: compute regularize user bias term
b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i       - mu)/(n() + final_lambda))
b_u[is.na(b_u)] <- 0 # remove NA

# 3. step: compute regularize genre bias term
b_g <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - b_u - mu)/(n() + final_lambda))
b_g[is.na(b_g)] <- 0 # remove NA
```

```r
# 4. step: compute predictions on **validation** based on the terms above = TEST
predicted_ratings <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId" ) %>%
  left_join(b_g, by = "genres" ) %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)
```

6.2 RMSE of best model = model 7:

```
## [1] 0.8644514
```

Generate output file with predicted ratings to fulfill the requirement:
"a script in R format that generates your predicted movie ratings and RMSE score"
is done by the R file, not in the Rmd file.

7 Conclusion

With the linear model the movie ratings can be predicted without too much use of computer resources.

The best model is the "Regularized Movie + User + Genres Effect Model" with RMSE:

BESTRMSE:

```
## [1] 0.8644514
```

The more complicated the considered effects are, the longer it takes to calculate.

So one has to think about whether it makes sense to consider i.e. genres or dates because that improves the predictions only at the 3rd or 4th place behind the decimal point.

For future work: the distribution of the ratings per genres over time was just slightly touched, remarked they seem to follow the curve of the movies per year. A further approach would be to normalize these genre curves (divide by numbers of movies in the year) and see whether that has an effect on quality of the predictions.

7.1 Addendum

7.1.1 Benchmark: time to generate Pdf from Rmd with the small and big dataset

Machine	Dataset	Time
Laptop, Pentium 7 12GB	1M	2'41
Mac M1 8GB	1M	1'10
Old Dual Xeon 64GB	1M	3'35
---:	---:	---:
Laptop, Pentium 7 12GB	10M	19'56
Mac M1 8GB	10M	12'58
Old Dual Xeon 64GB	10M	28'12