The movieLens recommendation system

Jorge Amorós-Argos

14/6/2021

Contents

1 Introduction

As a wrap up exercise of the *Data Science* course, the direction has challenged alumni to train a recommendation system in order to predict movie afinities for a given user.

Such a system is at present time very common; we can all think our daily experience with streaming platforms like Netflix.

Ideally, if the system is fed with a user liking i.e. give ratings to different movies, then it can advise of other movies of our taste based on the community.

1.1 Data

1.1.1 Resources

We'd like to thank *grouplens* for their project movielens, which is a website that helps you find movies you may like.

Basically they keep available a database of movies and user ratings.

Since the data and computing time can be very high, the direction has chosen a subset of 10 million entries, available at the following link

1.1.2 Cleaning the data

The script to download and suit the data to R is already given.

However, we can read through the code and make the following explanation:

1. Original data is stored in 2 files:

Filename	Content
movies.dat	It's a database for each movie, holding the movieId, title, genre...
ratings.dat	This is the database of the reviews, connecting userId->movieId->rating

2. The data is ascii delimited with "::" and for each file, has a constant number of columns. Both objects are imported to R
3. At the end, a single dataframe is produced, combining both previous objects. We can see *ratings.dat* as the master file, complementary information for the movies is done via *left_join* command on *movies.dat*

1.1.2.1 Train and test set

To prepare our recommendation system, the previous single dataframe must be split into a train and a test set, i.e. 2 dataframes.

We will use 10% of available data for test.

This can be easily done in R via *createDataPartition*

Warning The partition function doesn't care about the simultaneous presence of the same movies and users on both train and test set.

In other words, if no action is done, the test set will probably fail as some users and movies haven't been detected during the training phase.

Therefore, we must manually drop those entries of the validation that are not in the train set. We can reuse these "rejected" for train purposes though.

The following code will do the trick:

```
# 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)
```

1.1.3 Structure of data

As an outcome of previous point, there will be 2 datasets aimed for different purposes:

Name	Purpose
edx	training
validation	testing

Inside each dataset, there're 6 fields (columns)

Field	Description
userId	Numerical, user that inputs the rating
movieId	Numerical, serial number of the movie given by the website
rating	Numerical, from 0 to 5, the user opinion/score for that movie
timestamp	Date when the user entered the review
title	Char, the name of the movie
genres	Char, category given for that movie e.g Action, Drama, Action

Please note that the R script attached to this report needs the files above to be available in a subdirectory */rda*, properly named as *edx.rda* and *validation.rda*. Otherwise, the script won't work.

1.2 Goal

The purpose of this exercise is to define and build the most accurate movie recommendation system.

Such a system will help any future user, based on his reviews, to find products of his taste.

To reach it, we will train and check the RMSE (accuracy) of the system by trying different models of growing complexity as seen in the course.

2 Methods & analysis

2.1 The model theory

Directly taken from the course, the model that we will use for prediction will have several layers of complexity in order to reach a satisfactory accuracy.

Broadly speaking, the model will grow as follows:

1. $Y_{u,i} = \hat{\mu} + \epsilon_{u,i}$
2. $Y_{u,i} = \hat{\mu} + \hat{b}_i + \hat{b}_u + \epsilon_{u,i}$
3. $Y_{u,i} = \hat{\mu} + \hat{b}_i + \hat{b}_u + \hat{\beta}_g + \epsilon_{u,i}$

2.1.1 Loss function

In order to evaluate the accuracy of the predictions, the RMSE of the errors will be used:

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

Needless to say that the final accuracy must be evaluated in the *test* set, not the training one.

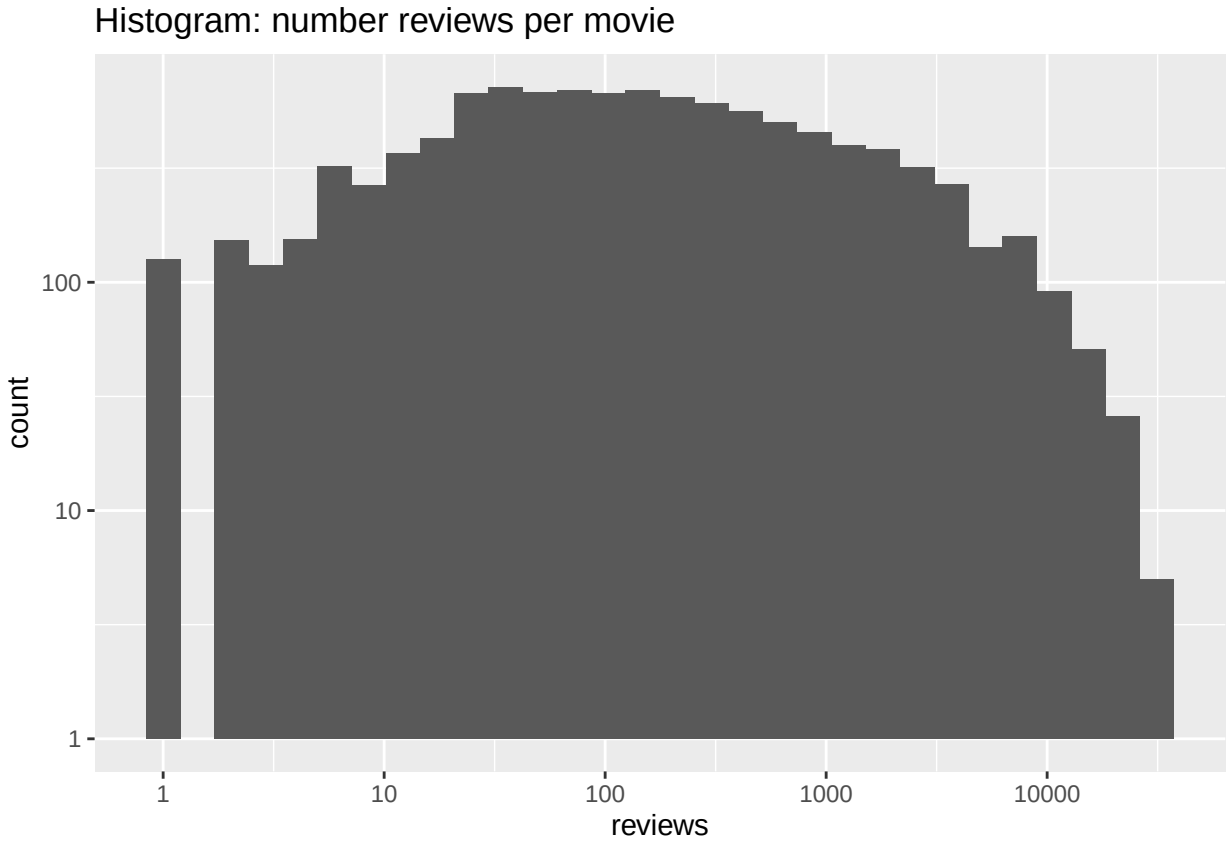
2.2 Exploring the data

We need to get familiar with the data. This will help us to choose the best model and also to explain possible mismatches.

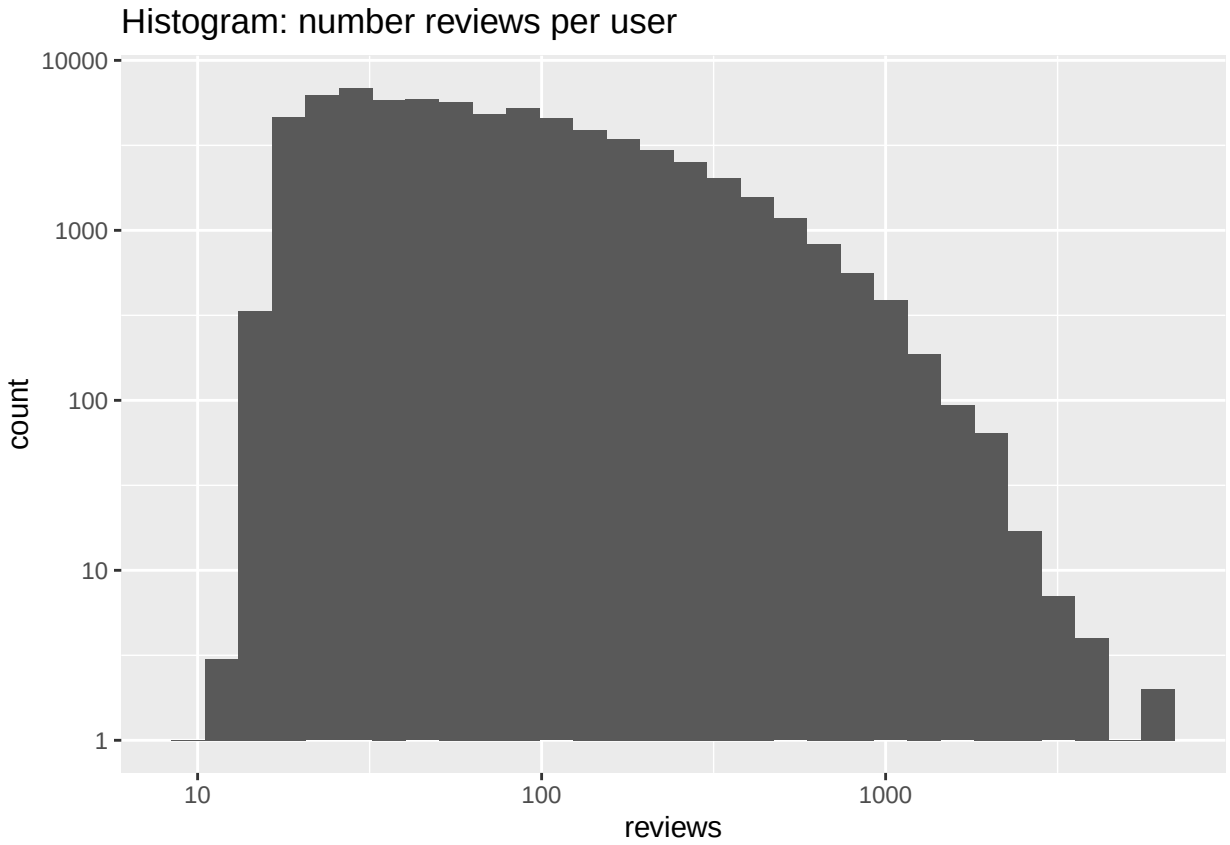
Note: we will always use the train set in the analysis shown here unless stated

Let's first plot some features:

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



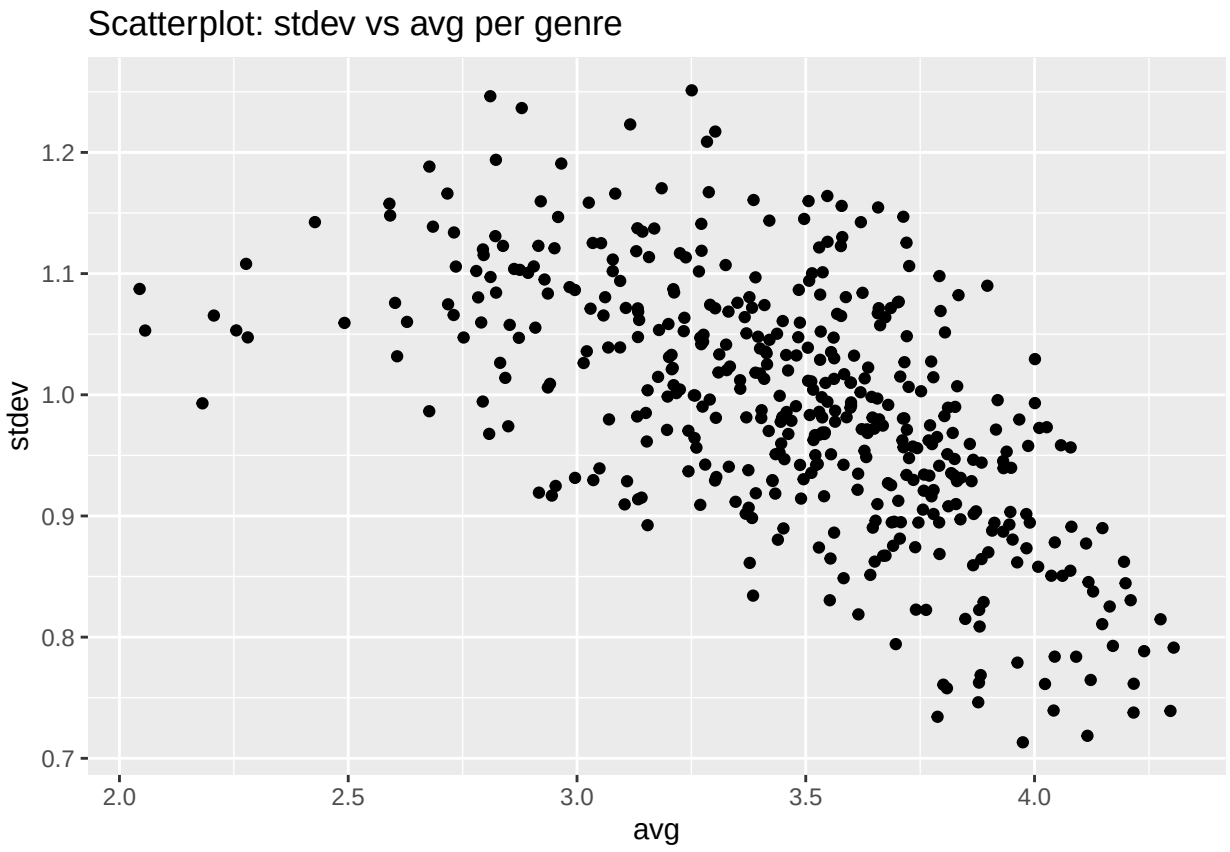
We can see that many movies only have 1 review. This will cause issues as the rating is not averaged/counter checked with other users and therefore can be biased. This will lead us to the *regularization* step.

Also, most of the users are very cooperative and have rated more than 10. In average 128 reviews, which is a lot.

```
## [1] 128.7967
```

Last not least, let's plot the rating per genre For this purpose, we will do a scatter plot of the standard deviation vs average grouped by popular genres, we get the following plot[1]:

[1]The genre data is in fact a composite of different genre tags. In order to simplify the problem, we're considering each combination as a unique tag. Also, only very very rated genres are considered.



We could infere:

- Some genres are clearly better rated, with less variability than others
- There's a bias of the rating per genre

2.2.1 The genre approach

It is really challenging to deal with the genre information.

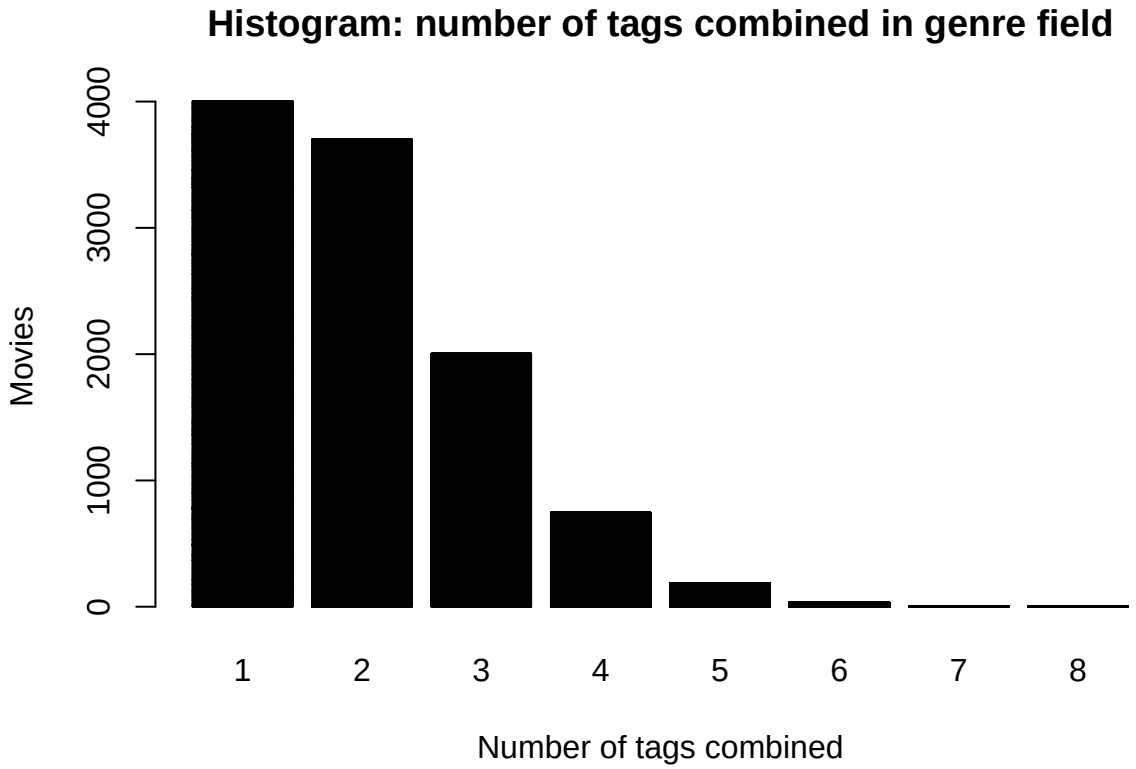
To start with, each movie has a combination of basic tags. We had to reconstruct them from the data set and they are 20.

We can list each tag (category) and the frequency of appearance in descending order:

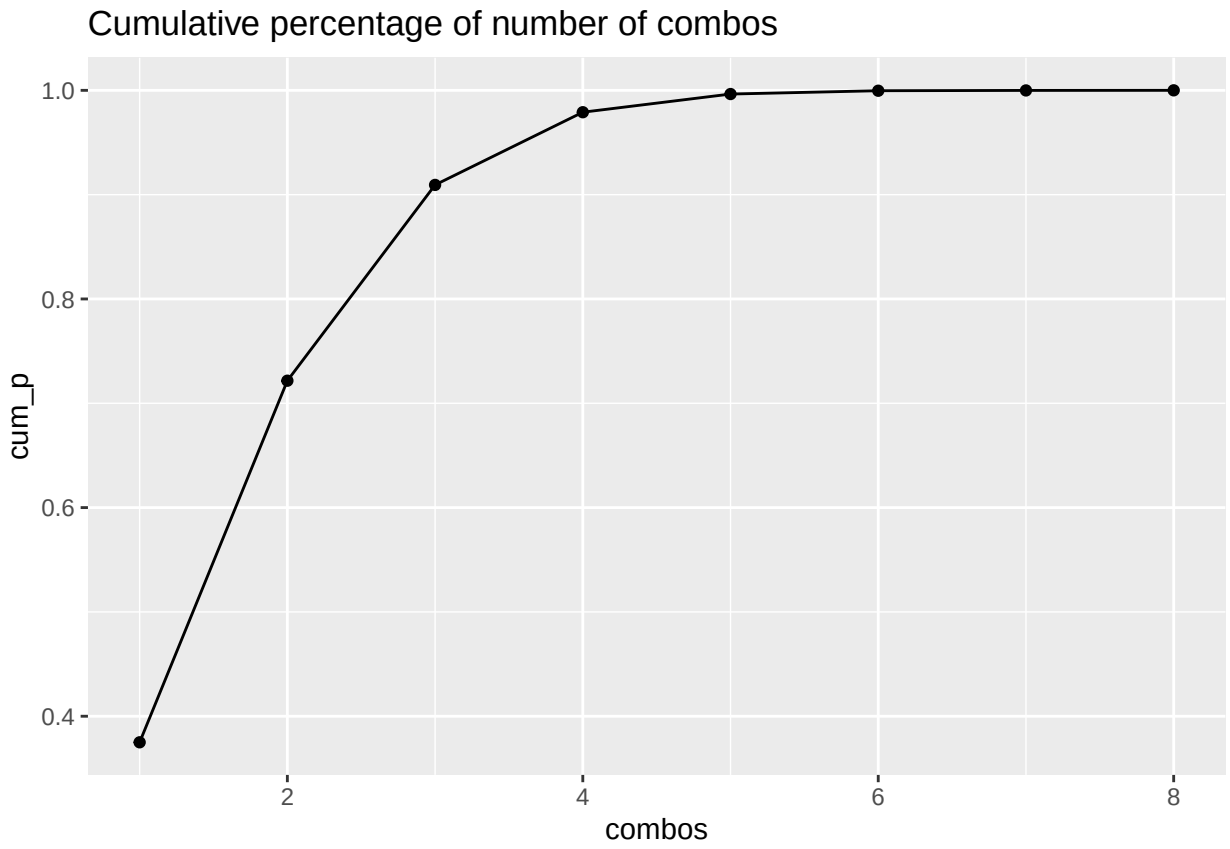
	Mean
Drama	0.4344559
Comedy	0.3934343
Action	0.2845033
Thriller	0.2584316
Adventure	0.2120978
Romance	0.1902322
Sci-Fi	0.1490194
Crime	0.1475230
Fantasy	0.1028479
Children	0.0819988

	Mean
Horror	0.0768312
Mystery	0.0631476
War	0.0567938
Animation	0.0519072
Musical	0.0481197
Western	0.0210436
Film-Noir	0.0131711
Documentary	0.0103406
IMAX	0.0009090
(no genres listed)	0.0000008

We can also know the frequency of tag combos i.e. how many tags are used per movie genre classification. The following histogram will be enlighting:



We can see that the most common combinations are with 1 and 2 tags (72%).



We could think of using for the model only a combination of 2 tags for the genre, ignoring the rest of the combinations.

This could represent an overload of programming since we must best sort each movie according only 2 tags.

However, we have discarded this idea because the genre is giving little information in the end. See the this discussion

The genre approach will consider each combination unique, not to be reduced to the basic tags

2.3 Model approach

We will compute the error for each complexity as shown before and make a final evaluation.

2.3.1 Mean

The model taking mean as the only estimator is as follows:

$$Y_{u,i} = \hat{\mu} + \epsilon_{u,i}$$

2.3.2 Movie & user bias

If the add to the model before the effect of the movie and user bias, the prediction model is as follows:

$$Y_{u,i} = \hat{\mu} + \hat{b}_i + \hat{b}_u + \epsilon_{u,i}$$

The *per movie* and *per user* estimates are affected by a regularization parameter λ as:

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

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

This *regularization* is needed because many movies are reviewed (rated) only once; therefore, it is not averaged with other users and therefore can be biased.

Because of the huge dataset processed, we have divided the *regularization* λ solution into two loops:

1. Coarse grid of 1, ranging from 0 to 10
2. Fine grid of 0.1, ranging ± 1 around estimated solution above

2.3.2.1 Genre bias A further refinement, after the data exploration that we have shown before, is to consider a new parameter β_g that reflects the genre of the movie. This will affect the predicted rating as follows:

$$Y_{u,i} = \hat{\mu} + \hat{b}_i + \hat{b}_u + \hat{\beta}_g + \epsilon_{u,i}$$

Obviously, this can only happen if the genre has been evaluated with the training set, otherwise must be considered 0

The variable is calculated after grouping *per genre*:

$$\hat{\beta_g} = \sum_{i=1}^{n_g} (Y_{u,i} - \hat{\mu} - \hat{b_i} - \hat{b_u})$$

3 Results

3.1 Training the system

It's worth stating again the procedure:

1. With the train set:

- Calculate $\hat{\mu}$
- Find λ_{opt} that minimizes the equation
- Once λ_{opt} is found, $\hat{b_i}$ and $\hat{b_u}$ are defined
- With all variables above defined, we can easily estimate $\hat{\beta_g}$

2. With $\hat{\mu}$, $\hat{b_i}$, $\hat{b_u}$ and $\hat{\beta_g}$ from above point, compute the RMSE with the **test** set.
3. We will prepare a breakdow of the RMSE per model and choose the best

3.1.1 Computing $\hat{\mu}$

It is straight forward to compute the $\hat{\mu}$ as it is the mean of the ratings.

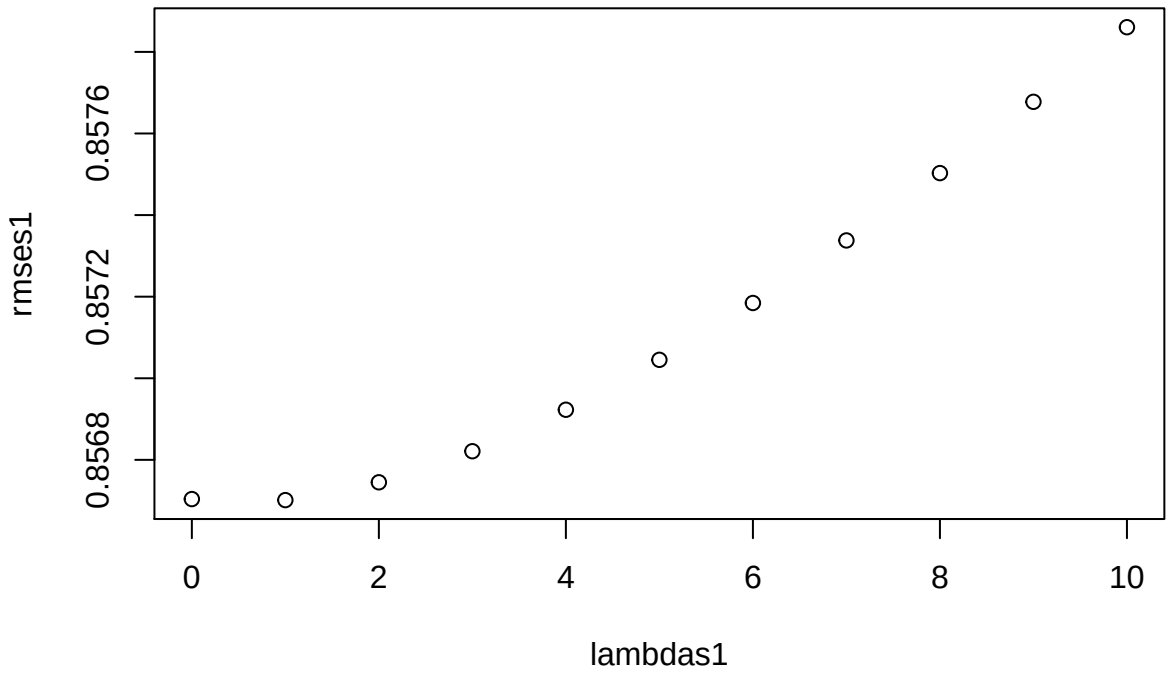
```
mean(edx$rating)
```

```
## [1] 3.512465
```

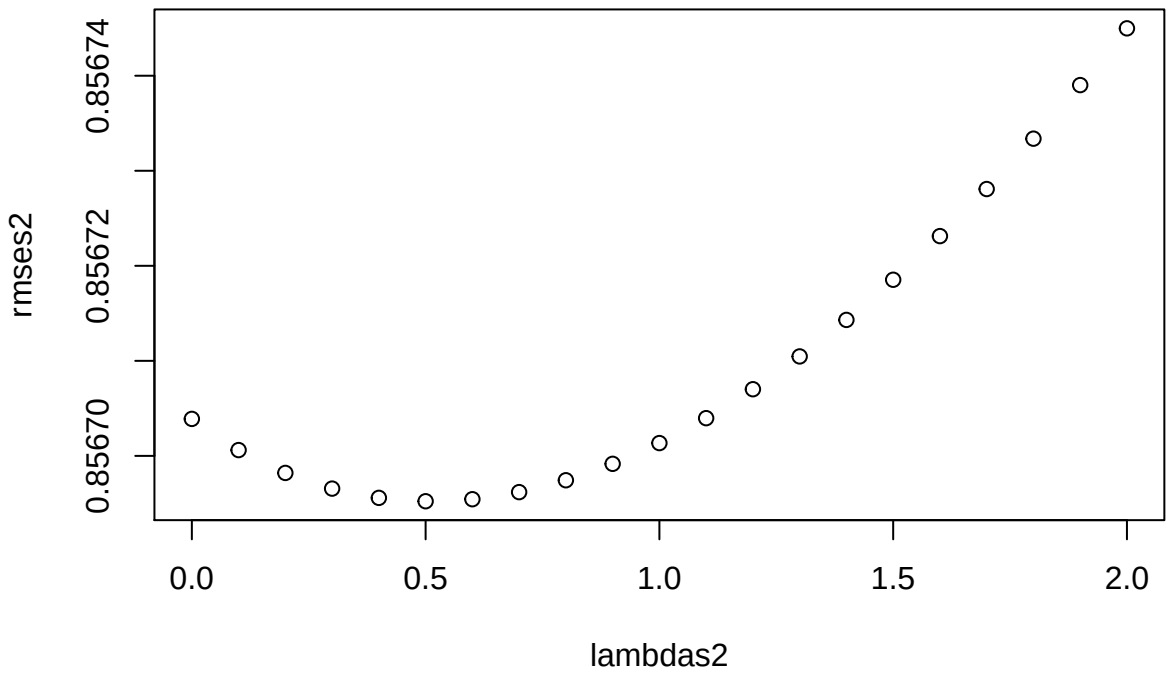
3.1.2 Computing λ_{opt}

With computing resources in mind, we have divided the solving into 2 loops

3.1.2.1 First loop The grid is chosen randomly from 0 to 10 with a step of 1. After visual inspection, the minimum must be between 0 and 2.



3.1.2.2 Second loop Refining the grid to a step of 0.1 in the range determined in the first loop. Here's the result of the graph and also the optimal values found:




```
rmse_opt
```

```
## [1] 0.8566952
```

```
lambda_opt
```

```
## [1] 0.5
```

3.1.3 Computing $\hat{\beta}_g$

Once determined the previous variables, it is immediate to calculate $\hat{\beta}_g$ if the data is grouped by genres.

3.1.4 Validation of the model: RMSE

Therefore, taking all the parameters from the train set, we will compute the RMSE with the *test* set:

method	RMSE
Only mean	1.0612018
Movie & user & regularization	0.8652226
Movie & user & regularization & genre	0.8648454

4 Conclusion

4.1 The best model

As the refinement of the model grows, so it does the accuracy of the predictions.

Therefore the best model includes movie, user and genre bias:

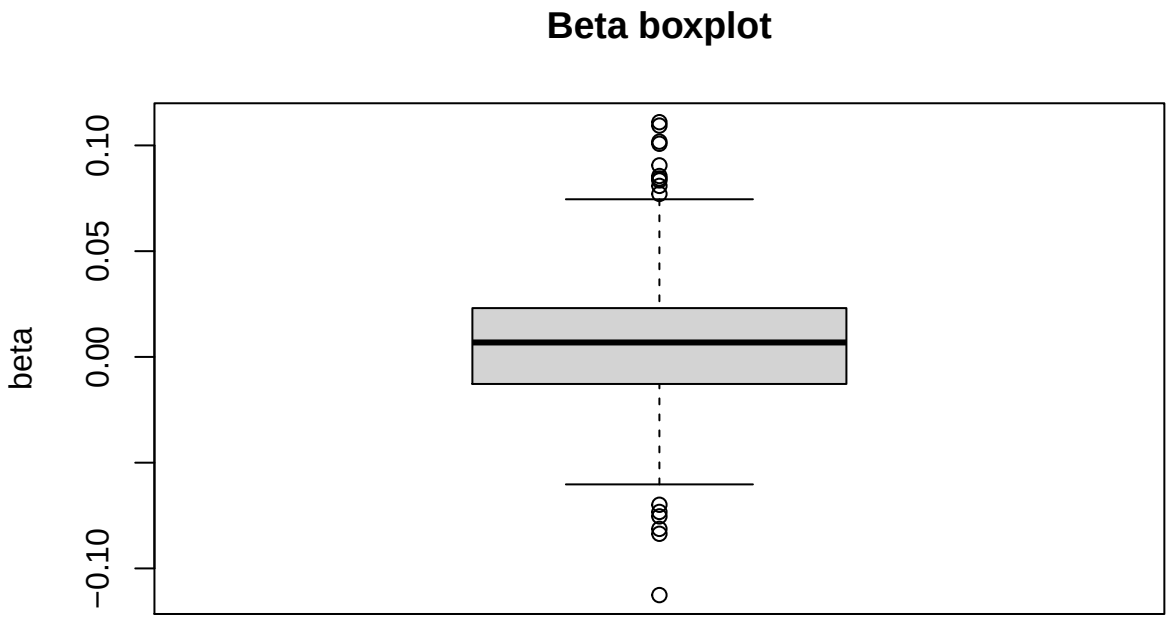
$$Y_{u,i} = \hat{\mu} + \hat{b}_i + \hat{b}_u + \hat{\beta}_g + \epsilon_{u,i}$$

4.2 Critical thinking: diving into genre bias $\hat{\beta}_g$

It's worth noting that each refinement introduces less benefit and we could reach the point that the computing time doesn't pay off.

In particular, we're curious about the genre bias.

Let's get some insight of the $\hat{\beta}_g$ constants with a boxplot:



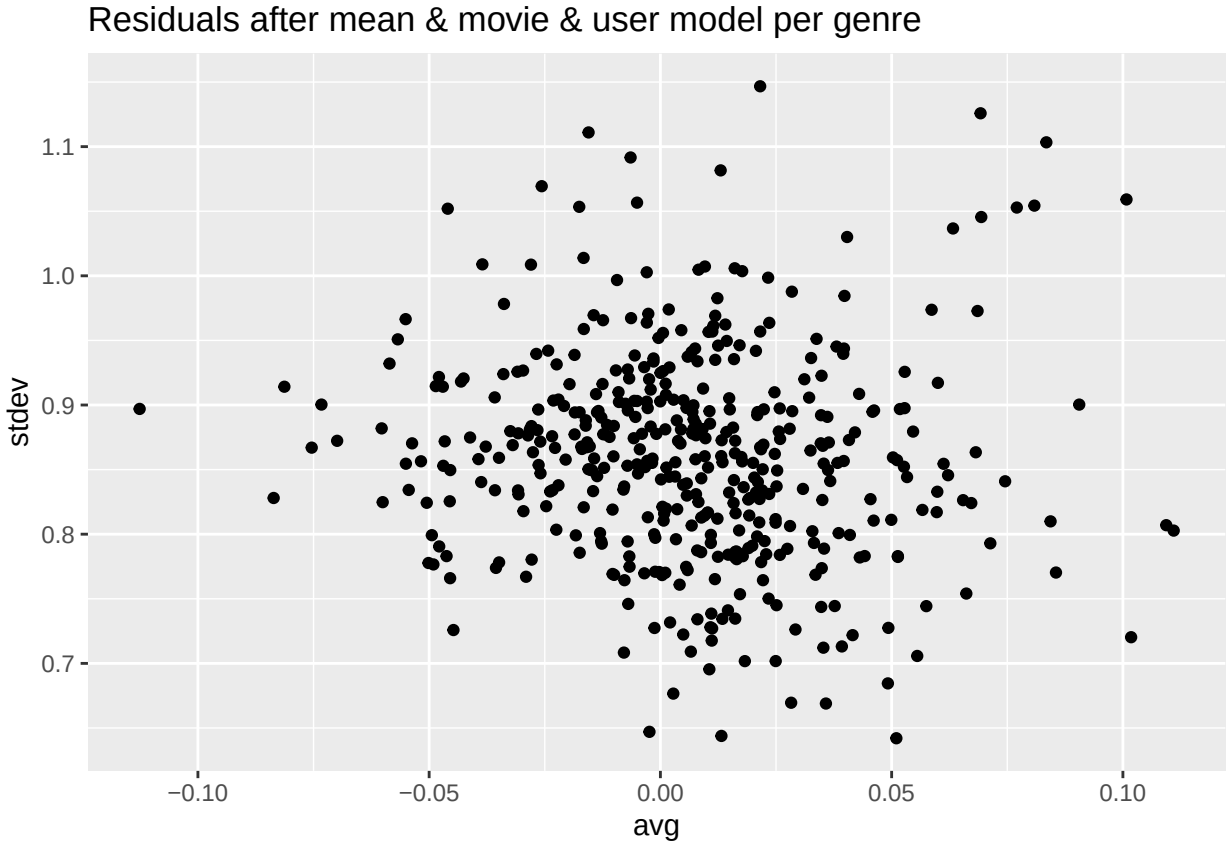

```
range(beta_hat$beta_hat)
```

```
## [1] -0.1125980  0.1109806
```

So in the extreme cases, only ± 0.11 of the rating will be affected, in many most cases less than that.

What happened to our assumption that the genre had a considerable weight?

We graph now the same scatter plot but for the residuals after an estimation with $\hat{\mu} + \hat{b}_i + \hat{b}_u$



We can see now that after the adjustment, the trend vanishes being now rounded and centered on 0.

4.2.1 Explanation

What has happened? The trend that we detected at first stage is real, however we can argue that *the genre is something that is intrinsic with the movie* and therefore is already estimated with $\hat{b}_i$. Also a given user may consume and rate mostly a certain kind of genre therefore $\hat{b}_u$ is somehow detecting some genre bias.

In the end, all the significant information has already been captured before $\hat{\beta}_g$.

In our opinion, this estimator could be dropped as it is arguably useful.

4.3 Future work

There could be other variables that could refine the prediction but those need exploration and final checking. To point out out a few:

- Check genre tags as predictors: decompose each genre into it's a combination of basic tags and treat them as predictors as seen in *machine learning* course
- Check if *timestamp* of the ratings define the user: maybe night owl users are more cranky than early birds?