Movielens assignment report

Leonardo Tellaroli

7/5/2020

Contents

1 Introduction

In this document we will first analyze the data contained in the Movielens dataset provided and then run three different machine learning algorithms on it and compare the results, our target is obtaining an RMSE value lower than 0.8649. The data provided is a list of movie ratings in the form of a tibble, as we can see below:

```
## # A tibble: 6 x 9
##   userId movieId rating timestamp title genres release_year n_ratings_movie
##    <int>   <dbl>  <dbl>     <int> <chr> <chr>         <dbl>           <int>
## 1      1     122      5 838985046 Boom~ Comed~         1992            2178
## 2      1     185      5 838983525 Net,~ Actio~         1995           13469
## 3      1     292      5 838983421 Outb~ Actio~         1995           14447
## 4      1     316      5 838983392 Star~ Actio~         1994           17030
## 5      1     329      5 838983392 Star~ Actio~         1994           14550
## 6      1     355      5 838984474 Flin~ Child~         1994            4831
## # ... with 1 more variable: n_ratings_user <int>
```

For each review we can find:

- The rating (ranging from 0 stars to 5 stars)
- The user that gave the rating
- The time when the review was created
- The title of the rated movie
- A group of genres related to the movie
- The year the movie was released (we can extract it from the title)
- The total number of reviews for each movie (computed value)
- The total number of reviews for each user (computed value)

We will first study the correlations between those values and the given rating, develop a model that predicts a rating based on the other values and finally test the model on the validation set to measure their performance.

2 Data preparation and exploration

2.1 Preparation

The first step of the process is the preparation of the data. Data is downloaded from the Grouplens website and is then splitted into 2 different dataframes with the **Caret** *CreateDataPartition* function that randomly creates partition of our data.
The first partition, which contains 90% of the data (9000055 Reviews, 10677 Different movies and 69878 users) will be used to train and tune the parameters of our algorithms.
The second partition, formed by 10% of the data will be used to check the performance of the algorithms created.
The performance metric used for the tuning and for the evaluation of the algorithms is the Residual Mean Standard Error (RMSE). The RMSE is the square root of the squared mean difference between the prediction and the actual value found in the test or validation set. The lower the RMSE, the better the prediction.

$$RMSE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}\Big(prediction_i - rating_i\Big)^2}$$

Other data wrangling task performed are the extraction of the release year from the title thanks to the **Stringr** package as shown below:

```
 #extract years of release from the datase
release_years<-str_extract(str_trim(edx_small$title),"[(]([0-9]{4})[)]"))
#removes () and converts to numeric
release_years<-as.numeric(str_replace(release_years,  "[(]([0-9]{4})[)]", "\\1"))
```

The computation of the number of ratings for each movie and each user thanks to the **Dplyr** package *group_by* and *summarize* functions:

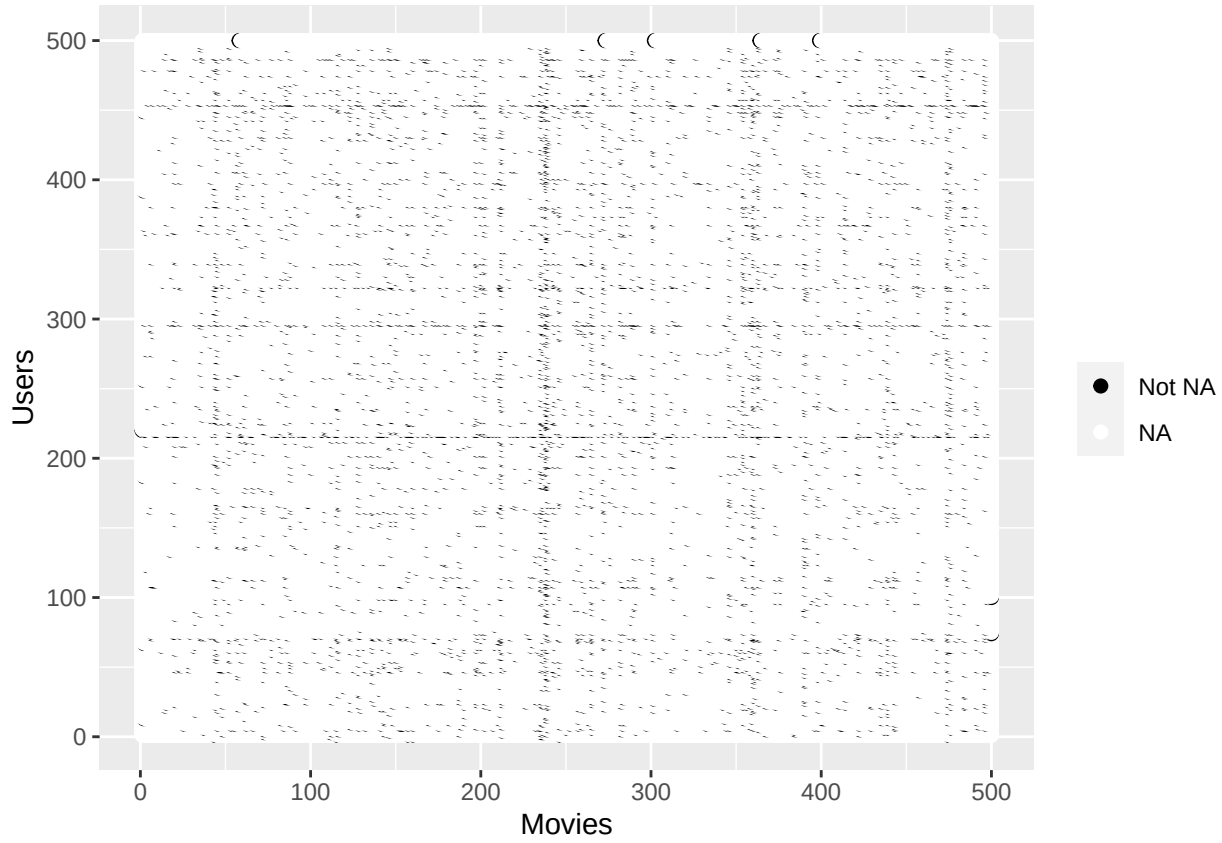
```
nratings_effect_plot <- edx_small %>% group_by(movieId) %>% summarize(mean = mean(rating),
    nratings = n())
```

And the conversion of the timestamp into a date thanks to the **Lubridate** package:

```
edx_small %>% mutate(date = round_date(as_datetime(timestamp),
    unit = "week"))
```

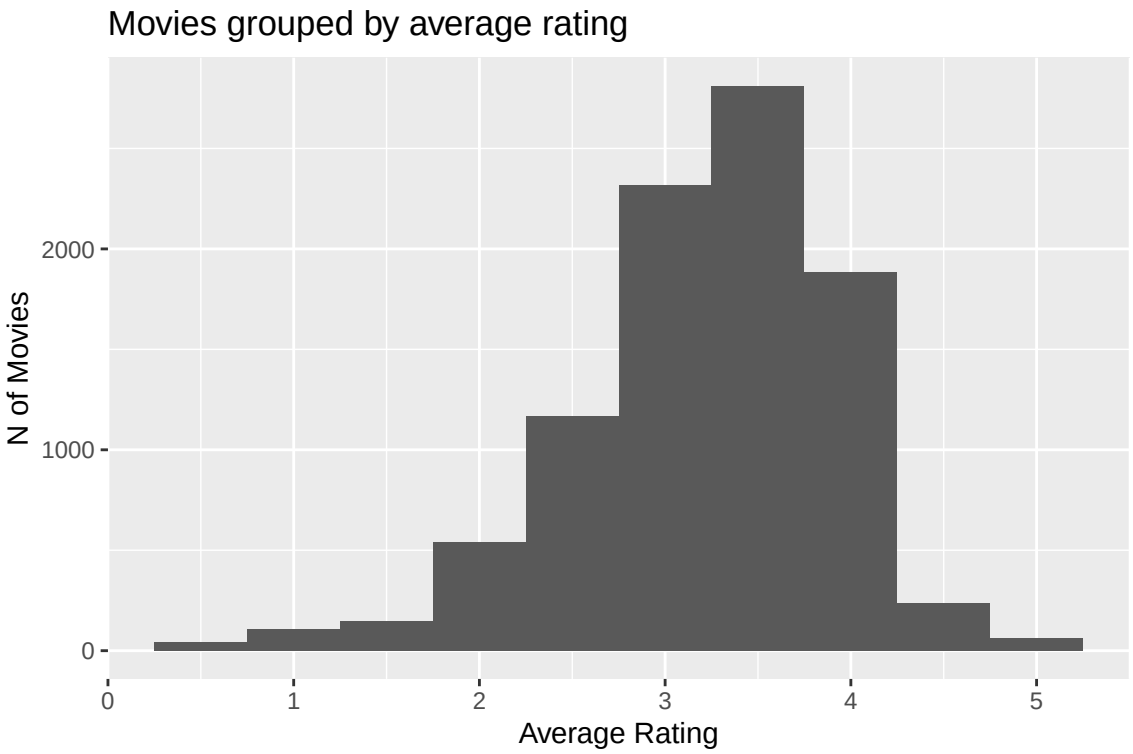
2.2 Data exploration

The first thing we notice is that the data is very sparse, we visualize this by plotting a matrix with users as rows and movies as column and higlihting the Non NA values (user - movie combination with a rating value available) in it:

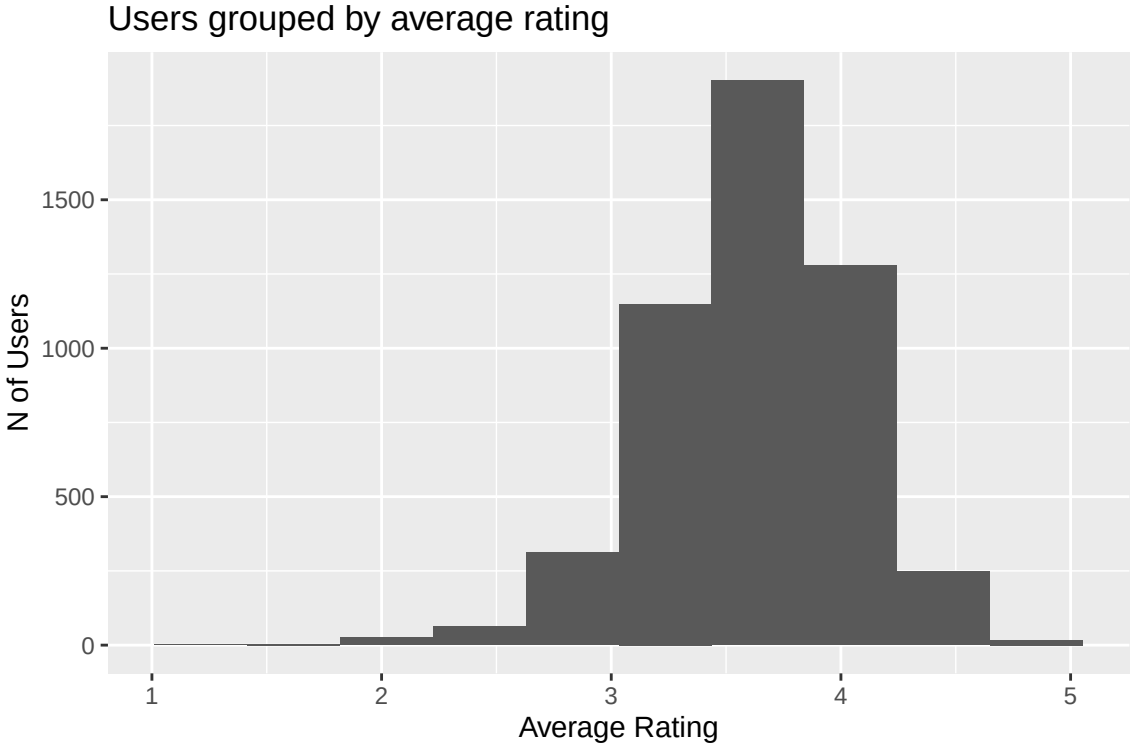


We can see that some movies have much more ratings than the others and also some user are much more active.

By plotting the distributions of movies grouped by mean rating we notice that the results are distributed across the whole rating range. This implies, as we can intuitively guess, that the rating given to a movie is related to the movie considered.



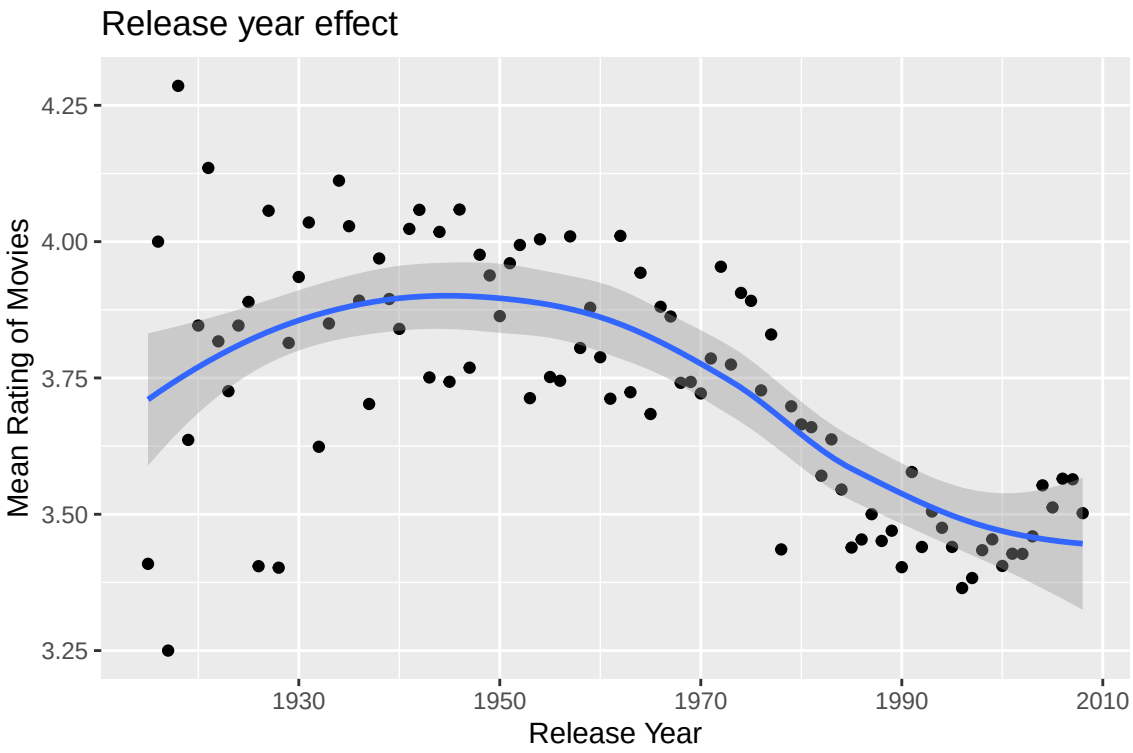
We also plot the distribution of users grouped by their mean rating and we see that different users tend to have different mean ratings, meaning that our prediction will have to include also a user effect accounting for the fact that some user often give ratings above or below the average.



Now we visualize the relationship between the release year of movies and their average rating.

We also run a correlation test between the two variables.

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



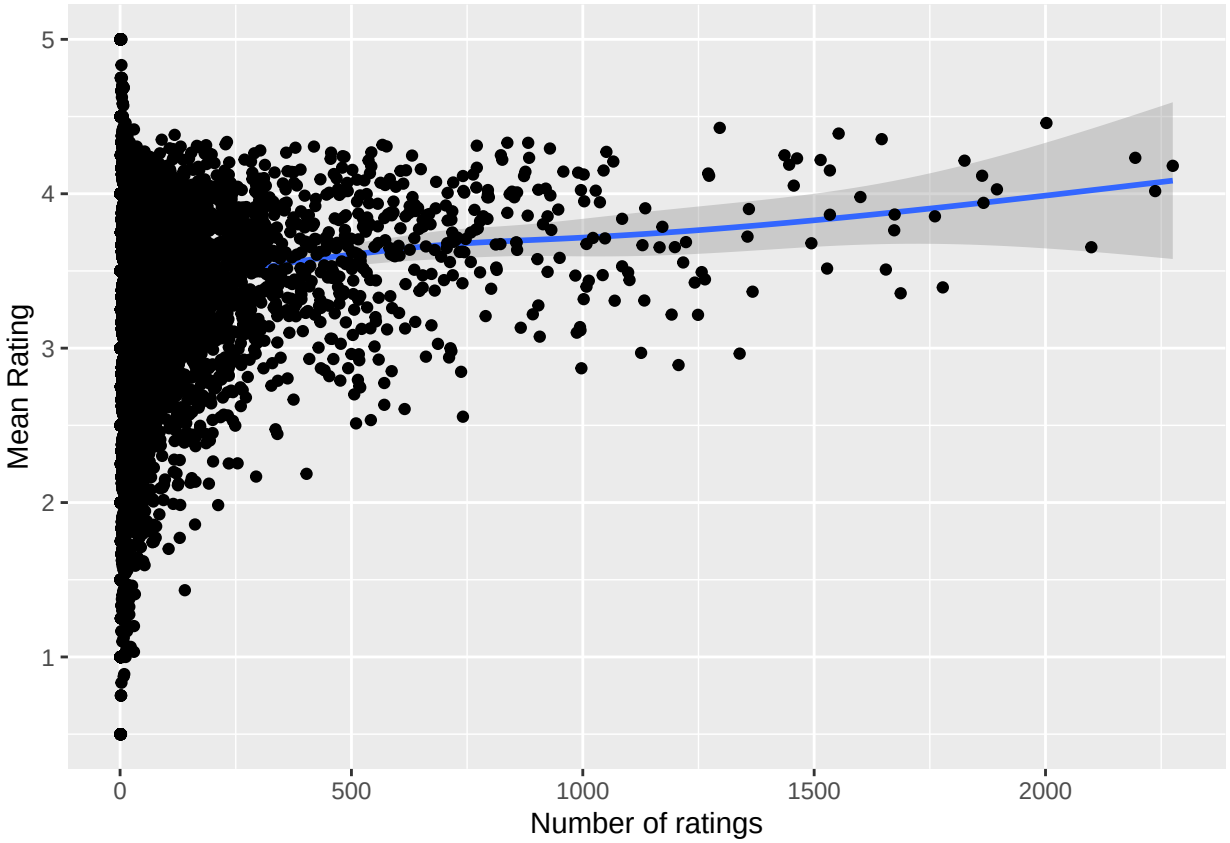

```
year_effect_corr <- cor.test(edx_small$release_year, edx_small$rating)
year_effect_corr
```

```
##
##  Pearson's product-moment correlation
##
## data:  edx_small$release_year and edx_small$rating
## t = -96.569, df = 653738, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.1209832 -0.1162032
## sample estimates:
##        cor
## -0.1185939
```

We see that older movies get higher ratings with a significant correlation estimate, this effect is the sign that Netflix usually offers old movies only if they are recognized as masterpieces, filtering out movies that are both old and not highly acclaimed.

We continue the exploration of the relations between the available variables by searching for an effect of the number of times a movie is rated ant it's average rating:

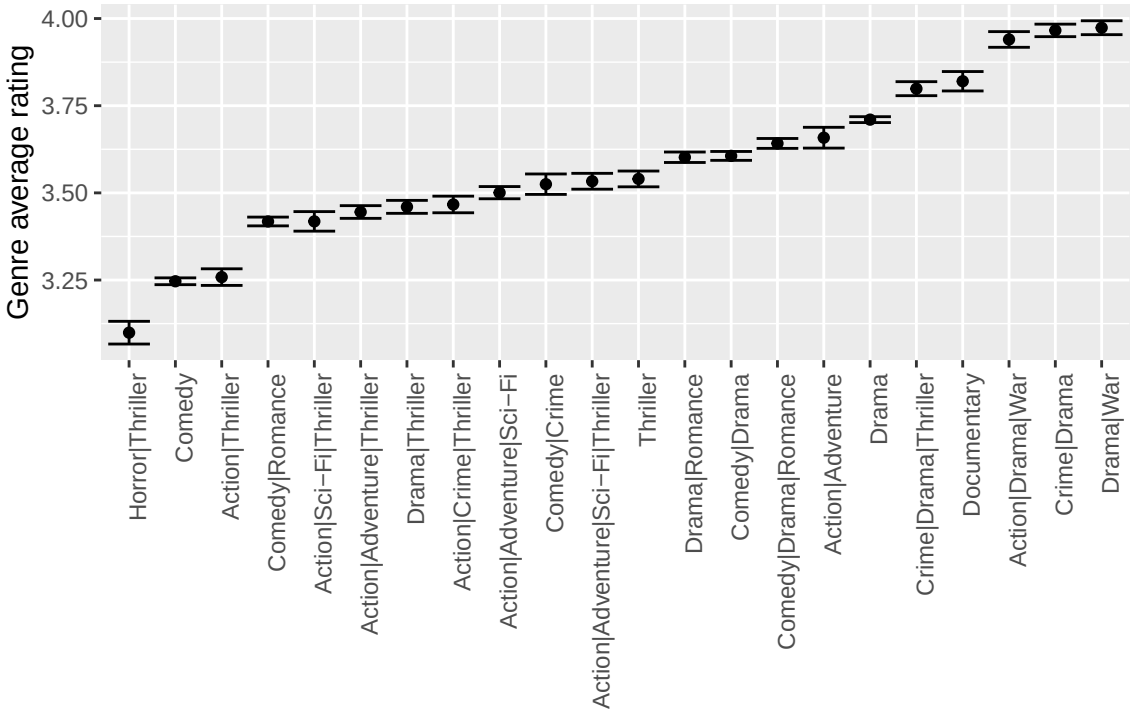
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```




```
##
##  Pearson's product-moment correlation
##
## data:  nratings_effect_cor$nratings and nratings_effect_cor$mean
## t = 15.801, df = 9307, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.1417815 0.1813496
## sample estimates:
##       cor
## 0.1616305
```

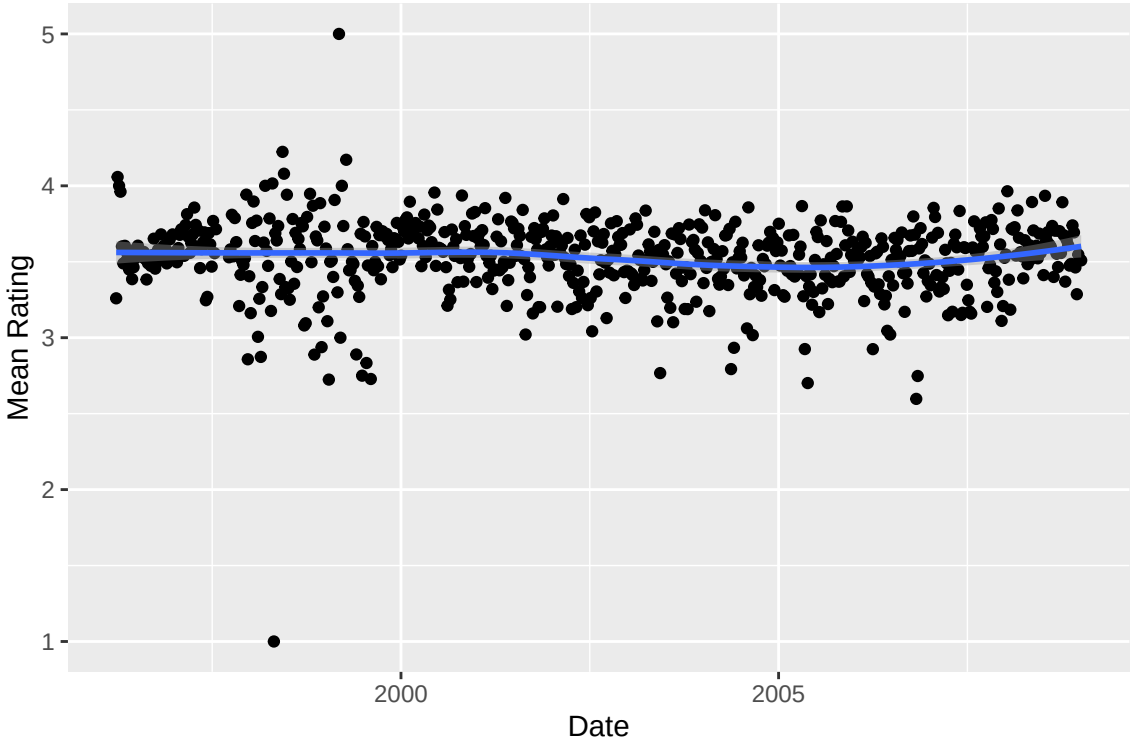
We see that movies with a higher number of ratings have an higher average rating and the variability between their ratings is much lower.

We continue the analysis computing the average rating for genres with more than 5000 associated movies and we get the following results:



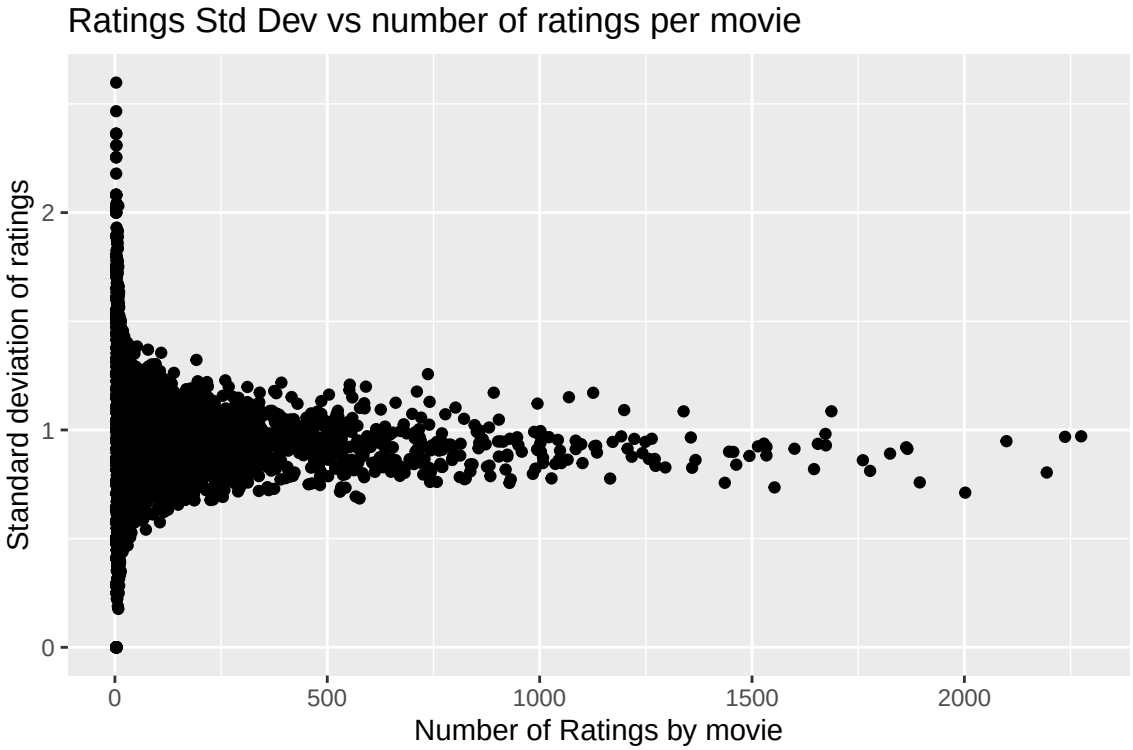
We indeed see that some genres get higher ratings than others, so there is a genre effect on ratings. Now the only question left on the correlations between our available variables and the ratings is: Are the given ratings influenced by the time when the rating was given?

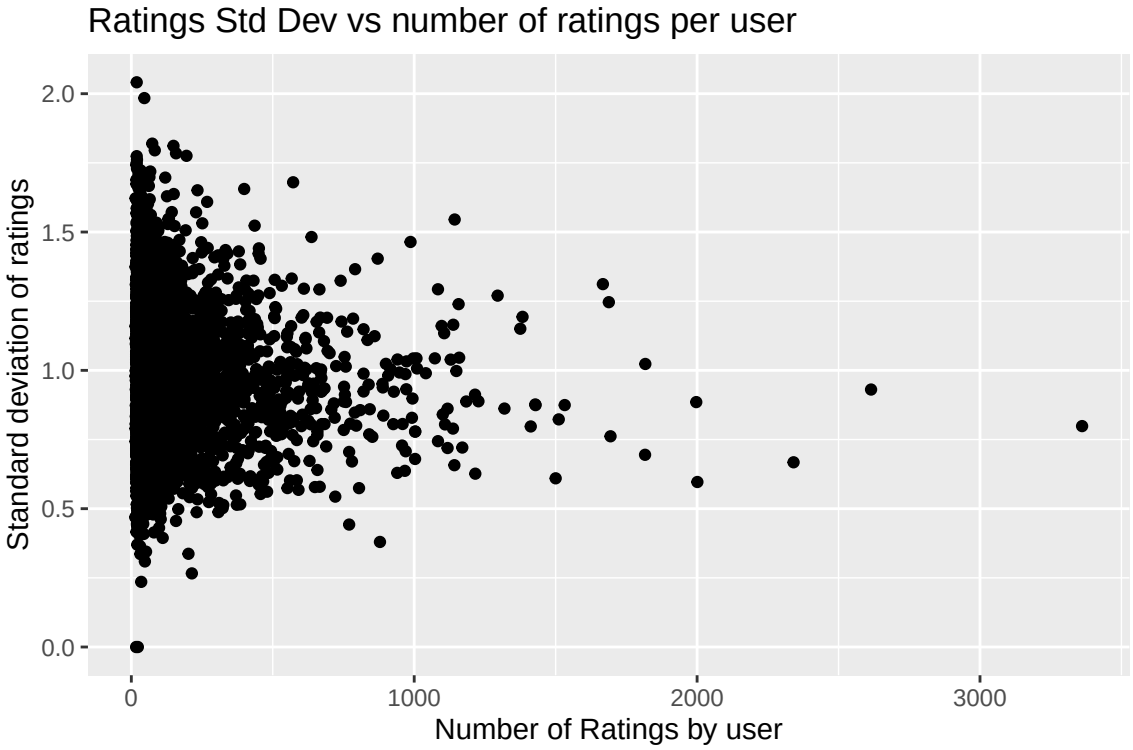
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



The plot above groups the ratings by week and then plots the mean of every week. As we can see there is a lot of week to week variance and we cannot say that there is a clear connection between the rating date and the rating value.

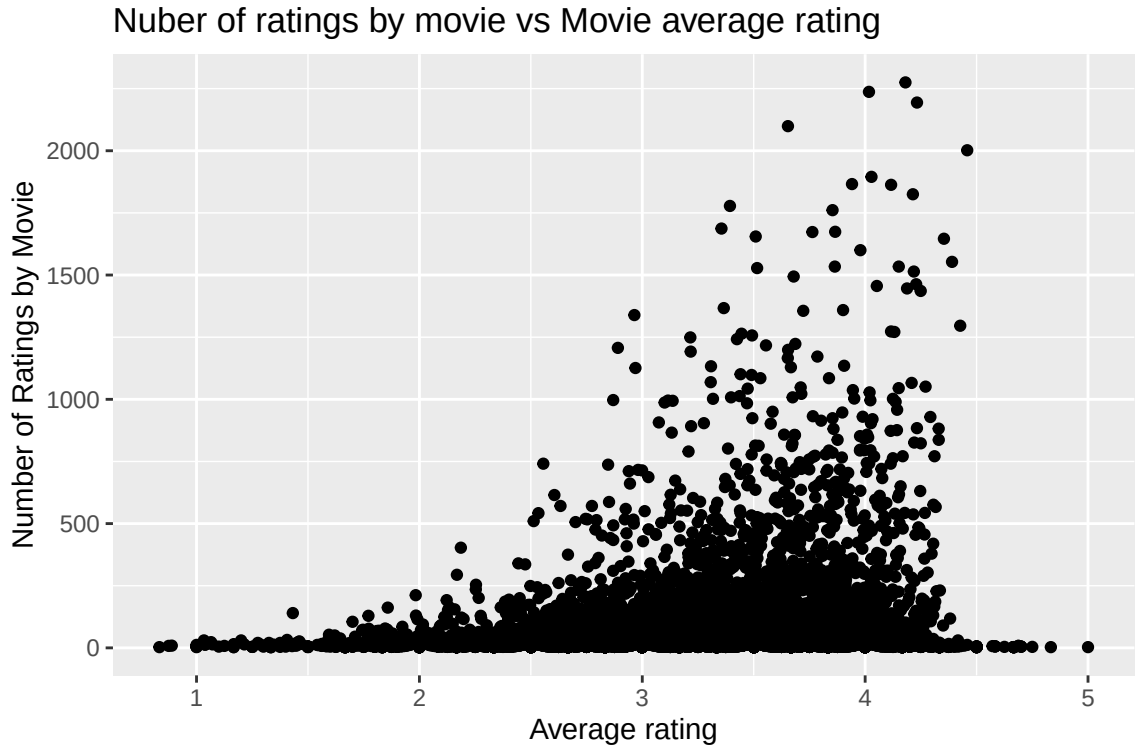
Another interesting feature to visualize is the standard deviation of the ratings vs the number of ratings related to each movie and to each user.





As expected the variability related to movies and users with a low number of ratings is higher, and with increasing number of ratings the standard deviation is much more stable. This is a clear example of *Regression to the mean* .
We will have to consider this feature of our data when developing our models, to avoid giving too much weight to ratings related to movies with a low number of ratings.

This characteristics of the data is also noticed when plotting the average rating of each movie on the x-axis and the number of ratings on the y-axis. We now see that the extremes of our rating scale are reached only by movies with a low number of ratings.



2.3 Further data preparation

In the following algorithms we'll have to tune some parameters. To carry out this tuning we will train our algorithm on a training set and then compute the RMSE on a test set, choosing the parameters that minimize its value. Because our validation dataset will be used only for the final assessment of the RMSE of our algorithms we can't use it for the tuning process so a further splitting of the edx dataset is needed to build our training and test sets.
The CreateDataPartition function will be used again to create a test set with 10% of the data from the edx dataframe. The code used is the following:

```
set.seed(1, sample.kind = "Rounding")
test_index2 <- createDataPartition(y = edx$rating, times = 1,
    p = 0.1, list = FALSE)
edx_train <- edx[-test_index2, ]
temp <- edx[test_index2, ]
edx_test <- temp %>% semi_join(edx_train, by = "movieId") %>%
    semi_join(edx_train, by = "userId")
removed <- anti_join(temp, edx_test)
edx_train <- rbind(edx, removed)
```

3 Methods

3.1 Regularized linear regression

The first model that will be implemented is the Regularized linear regression. In this model the predicted rating is computed starting from the average of the ratings from the training set and then adding a movie related term (b_i) and a user related term(b_u) to account for movie to movie and user to user variability. The usual model is showed in the following formula:

$Y_{ui} = \mu + b_u + b_i$

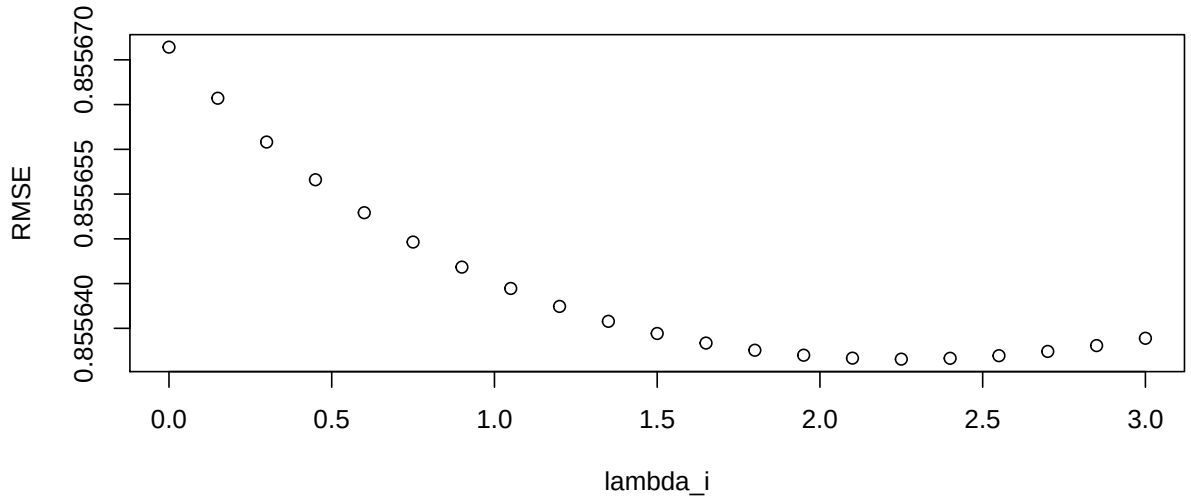
In this formula Y_{ui} is the predicted rating for user u and movie i, μ is the mean of all the ratings calculated on the training set.
b_i is equal to $\frac{1}{n_i}\sum_{u=1}^{n_i}(y_{u,i} - \mu)$ where n_i is the total number of ratings associated with movie i, $y_{u,i}$ are the ratings received by movie i. This formula is applied for every movie on the training set data.

b_u is equal to $\frac{1}{n_u}\sum_{i=1}^{n_u}(y_{u,i} - \mu - b_i)$ where n_u is the number of ratings for user u and $y_{u,i}$ are the ratings given by user u taken from the training set. This formula is used for every user based on training set data.

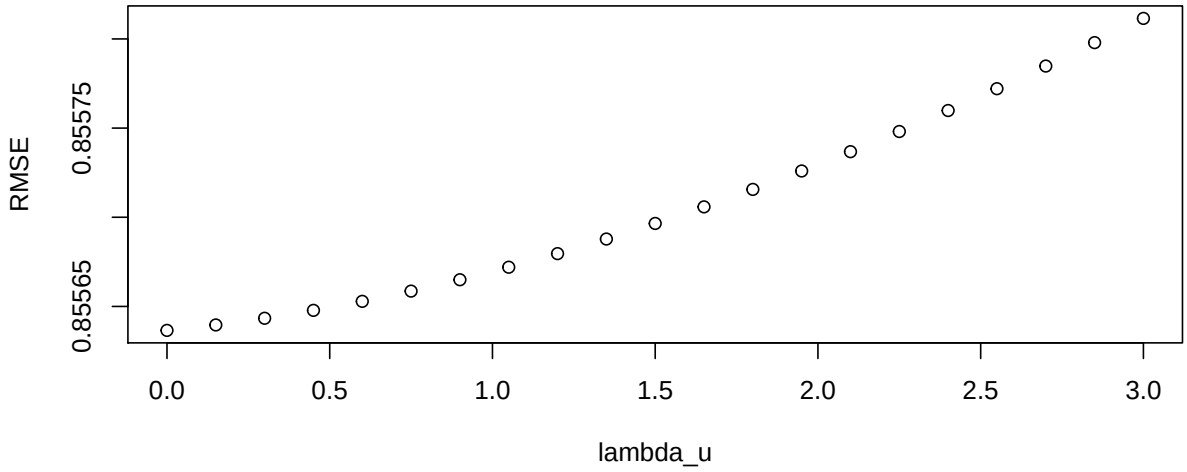
To account for the fact that movie with a low number of ratings and users that made a low number of reviews provide more viable estimates, leading to a large final RMSE of the model, we introduce some variations to the algorithm above to penalize movie and users with small sample sizes.
The modifications implemented are the following:

- μ is substituted by μ_w which is the average of the ratings weighted by number of ratings for the movie considered, calculated with the function *weighted.mean()*
- b_i is now computed as $b_i(\lambda_i) = \frac{1}{\lambda_i + n_i}\sum_{u=1}^{n_i}(y_{u,i} - \mu_w)$
- b_u is now computed as $b_u(\lambda_u) = \frac{1}{\lambda_u + n_u}\sum_{i=1}^{n_u}(y_{u,i} - \mu_w - b_i)$

As we can see we now have two parameters in our algorithm, λ_i and λ_u . To choose those parameters we run our model trying various values in a double for cycle and then choosing the combination of λ_i and λ_u that returns the lowest value of RMSE by testing the algorithm on our test set (As explained above this test set is different from the validation set).
For those interested the tuning code can be found from row 128 to row 145 in the mainscript.R file.
Here we plot the value of RMSE vs λ_i having fixed λ_u at it's best value:



And the value of RMSE vs λ_u having fixed λ_i at it's best value:



As a last step in our prediction algorithm we transform predictions>5 into 5 and predictions<0 into 0 because we know that ratings can't be higher than 5 or lower than 0. The code to generate predictions is shown below:

```
regularized_linear_prediction <- edx_test %>% left_join(b_mov,
    by = "movieId") %>% left_join(b_user, by = "userId") %>%
    mutate(prediction = wmean + b_mov + b_user) %>% mutate(prediction = ifelse(prediction >
    5, 5, prediction)) %>% mutate(prediction = ifelse(prediction <
    0, 0, prediction))
# last part of the line sets to 5 predictions that are higher
# than 5 and to 0 preditions that are negative
```

The RMSE found using this algorithm with the best values on the test set is:0.8556366

The code used to implement the release year effect can be found in the mainscript.r file between rows 157 and 163, I decided not to overload this report with code to stay focused on the method characteristics and results.

3.2 Adding Release Year effect

We can also try to include a term accounting for the release year effect. To do so we compute $b_{year} = average_y\,(y_{u,i} - \mu - b_i - b_u)$ with $average_y$ as the average over all ratings regarding movies released in year y. Then we introduce those year dependent term in our model linear equation:

$Y_{ui} = \mu_w + b_u + b_i + b_{year}$

The RMSE obtained by evaluating this model on the test set is equal to 0.8552975
This is an improvement compared with the RMSE obtained not considering the year effect.

3.3 Matrix Factorization

To obtain a better prediction we can transform our datasets as matrices with movies as columns and user as rows (as we did before to show the small number of non NA values).
This matrix can be decomposed in the product of two smaller dimensional matrices P and Q, those matrices can be found with factorization techniques such as Single Value Decomposition. The vector by vector products are then used to complement our linear model as follows:

$Y_{u,i} = \mu + b_i + b_u + p_{u,1}q_{1,i} + p_{u,2}q_{2,i} + \cdots + p_{u,m}$

By doing so we consider the preferences of the users because vectors q represent movie to movie correlation and vectors p represent user to user correlations.

Managing the edx dataset in it's movie-user matrix form is not possible on a common laptop and will return an error stating that not enough memory is available to allocate the object. To overcome this limitation we use the recosystem package. This package is specifically developed to build recommendation systems based on matrix factorization.

To use this package we first have to convert our data in a package specific object format with the function *data_memory()*, then we initialize the model with the *Reco()* function and we set tuning parameters such as the number of CPU threads available on our machine and the number of iterations to perform with the *opts()* function, to have control over computational time.
The model is then trained with the *Recomodel$train* function and predictions are generated with *Recomodel$train*. The code is reported below:

```
# setting seed again to prepare for next analysis using the
# recosystem package
set.seed(1, sample.kind = "Rounding")

# converting train and test set in format compatible with
# recosystem package
train_reco <- data_memory(user_index = edx_train$userId, item_index = edx_train$movieId,
    rating = edx_train$rating)

test_reco <- data_memory(user_index = edx_test$userId, item_index = edx_test$movieId,
    rating = edx_test$rating)

recomodel <- Reco()  #creating recosystem model as indicated in the package docs


# optimizing parameters for recosystem
opts = recomodel$tune(train_reco, opts = list(nthread = 4, niter = 5))

# nthread is the number of CPU threads available and niter
# the number of iterations

# training the model
recomodel$train(train_reco, opts = c(opts$min, nthread = 4, niter = 10))  #train model
# make predictions on the test set from the model
recopred = recomodel$predict(test_reco, out_memory())
rmsereco <- rmse(edx_test$rating, recopred)  #computes rmse
```

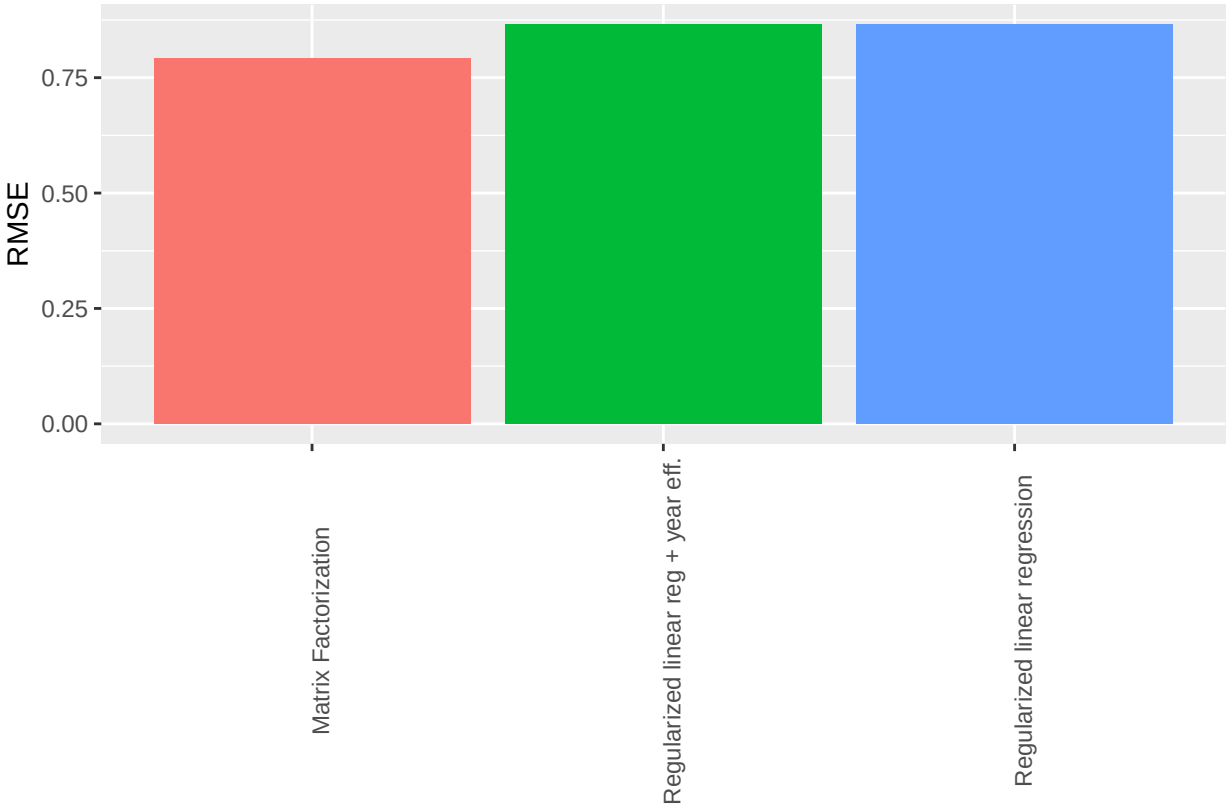
4 Results

We now evaluate the Regularized Linear Regression with and without year effect and the Matrix Factorization RMSE on the **validation** set, obtaining the following results:

*RMSE obtained with **Recosystem**: 0.7909384.

*RMSE obtained with **Regularised linear Regression + year effect**: 0.8647064.

*RMSE obtained with **Regularized Linear Regression**: 0.8650367.



As we see the Regularized linear Regression does not obtain an RMSE meeting our target of 0.8649 specified in the Movielens Grading Rubric related to this assessment.

A simple modification to the previous method introducing the release year effect that we noticed during data exploration finally creates a model meeting and slightly exceeding the performance target of 0.8649, obtaining a lower RMSE. This method obtains a 0.04 % improvement over the previous method.

If instead we use the model developed ad hoc for recommendation system by Yixuan Qiu and available trough the Recosystem package we manage to get a significantly better RMSE. In this way we obtain a further 8.53 % improvement over the Regularized linear Regression + year effect.

5 Conclusions

As we can see the best predictions for the Regularized Linear Predictions algorithm are obtained by considering only the movie and the user effects with a different tuning for the λ_i and λ_u parameters even at the expense of a larger computation time. The PC performance problem related with the handling of large dataset is also noticed when trying to handle large matrices, that go beyond the usual allocation memory available if not handled with ad-hoc methods such as storing them as sparse matrices, like it's done by the Recoystem algorithm. Base R can only use 1/4 of the CPU power since it can only work on a single thread. Recosystem instead uses multi-thread computing, significantly decreasing computation time, notice that when the recosytem section of the script starts running the CPU usage of a normal quad-core PC will go from around 30% to more than 90%.

We also see that specific algorithms developed to handle those large sparse matrices are needed to perform efficient calculations. The **Recosystem** package developed by *Yixuan Qiu* came to our help enabling much better results with the limited resources available.

On a 2.4Ghz Quad-core processor the scripts takes around 50 minutes to run. Also notice that if you have a

processor with less or more cores you should set the nthread argument in lines 176 and 179 to the number of processor cores you have.

All the files related to this report are available here .