Report Recommender Project | HarvardX Data Science Professional Certificate

Martin Noetzel

10/23/2020

Contents

1. Introduction

This report is a requirement of the 9th course of the Data Science Professional Program provided by HarvardX (Irizarry, 2020). The goal was to build an algorithm (a so called model) which can be used to recommend unknown movies to specific user. In Detail the algorithm will predict a rating that a specific user would have most probably given if he had seen it. As a basis of the predictions the algorithm will be trained on a set of user ratings for other movies.

In practice I used a data set provided by GroupLens research lab (Sou, 2018) which contains of 10 million ratings applied to 10,681 movies by 71,567 users. The data set was randomly split into two separate sets:

1) A training set (referred to in later code as edx data set) which contains about 90% of the data
2) A validation set that contains 10% of the data

The training set was used to train different models in order to define a final model. The validation set however was only used to validate the final model and to ensure the comparison of the model performance through all course participants.

Before running the R code in your console you have to ensure that these packages are loaded.

Below you can see the code we used to generate the two sets.

```
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")

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(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))
movielens <- left_join(ratings, movies, by = "movieId")

set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

To ensure every user in the validation set has at least some ratings in the training set we run the following code provided by the course staff.

```
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

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

The project's goal is to develop a model which will calculate ratings equal or as close as possible to real ratings given by users. The model performance will be measured by rmse (root mean squared error). The rmse measures the sum of all deviations from the relating real values (referd to as loss function). It is defined as:

RMSE = sqrt(mean(y_hat-y)^2), where y is the real value (rating) and y_hat is the corresponding prediction.

In order to earn the most possible points (or highest grade) my aim was to find a model with an rmse below 0.8490.

To calculate the rmse a specific function was defined by the course instructor:

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

This report consist of four main section:

1) **Introduction** - overview, project goal and project structure
2) **Analysis** - data cleaning, data exploring, gained insights and approach
3) **Results** - results and model performance
4) **Conclusion** - summary, limitations and recommendations for further improvements

2. Analysis

2.1 Data Pre-processing

The first step for me was to families myself with the provided data set. The code below gives an overview of the data structure and the included variables.

```
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                            genres
```

```
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

The data consist of the 6 variables (items) and 9.000.055 items. The following points draw my attention while watching this table:

- The variable "timestamp" is not formatted at all
- The variable "year" (meaning the release year) is bind to the variable "title"
- The variable "genres" contains of items that consist of more then one genre

In order to address these issues I performed the steps below:

1) The variable "timestamp" was formatted as datetime. The original variable was overwritten with the new values:

2) I extracted the variable "release year" from the variable "title" and stored the results in a new variable called "year". Since I did not expect me to do something with the variable "title" and to reduce processing time of operations I deleted the variable from the edx data set:

```
## # A tibble: 6 x 2
##   movieId years
##     <dbl> <chr>
## 1       1 1995
## 2       2 1995
## 3       3 1995
## 4       4 1995
## 5       5 1995
## 6       6 1995
```

3) I generated dummy variables of all genres and stored them separately to hold the edx data clean and tidy:

```
## # A tibble: 6 x 15
## # Groups:   movieId [6]
##   movieId gComedy gAction gCrime gThriller gSci_Fi gDrama gFantasy gAdventure
##     <dbl>   <dbl>   <dbl>  <dbl>     <dbl>   <dbl>  <dbl>    <dbl>      <dbl>
## 1     122       1       0      0         0       0      0        0          0
## 2     185       0       1      1         1       0      0        0          0
## 3     292       0       1      0         1       0      1        0          0
## 4     316       0       1      0         0       0      0        0          1
## 5     329       0       1      0         0       0      1        0          1
## 6     355       1       0      0         0       0      0        1          0
## # ... with 6 more variables: gWar <dbl>, gChildren <dbl>, gRomance <dbl>,
## #   gAnimation <dbl>, gMystical <dbl>, gMusical <dbl>
```

4) Since one of my assumption was that the ratings have specific age effects I defined two new variables which measure the age of a every rating that was given to a specific movie (movie age effect) or the age of ratings from a specific user (user age effect):

- A variable “age_movie” - days past since a specific movie received its first rating
- A variable “age_user” - days past since a specific user rated his first movie

To generate the new variables I first had to identify the first timestamp each specific movie received to get a reference point. Then I calculated the new variable and defined the min and max values in order to be able to define appropriate cluster. In a final step I standardized the new variable and called it age_movie_s.

```
firstRating_movie <- edx %>%
  dplyr::select(movieId,timestamp) %>%
  group_by(movieId) %>%
  summarise(firstRating_movie=min(timestamp)) %>%
  arrange(-desc(movieId))

age_movie <- edx %>%
  left_join(firstRating_movie ,by="movieId") %>%
  mutate(age_i =round(difftime(timestamp,firstRating_movie,units = "days"))) %>%
  group_by(movieId) %>%
  dplyr::select(age_i,userId,rating)

max(age_movie$age_i)
```

```
## Time difference of 5110 days
```

```
min(age_movie$age_i)
```

```
## Time difference of 0 days
```

```
age_movie_s <- age_movie %>%
  mutate(days_i=ifelse(age_i<=1000,"1000",
                       ifelse(age_i<=2000,"2000",
                              ifelse(age_i<=3000,"3000",
                                     ifelse(age_i<=4000,"4000",
                                            ifelse(age_i<=5000,"5000","6000"))))))
```

So the new variable looked like this.

```
## # A tibble: 6 x 3
## # Groups:   movieId [6]
##   movieId userId days_i
##     <dbl>  <int> <chr>
## 1     122      1 1000
## 2     185      1 1000
## 3     292      1 1000
## 4     316      1 1000
## 5     329      1 1000
## 6     355      1 1000
```

The second variable was generated similarly.

```
## Time difference of 4122 days
```

```
## Time difference of 0 days
```

```
## # A tibble: 6 x 3
## # Groups:   userId [1]
##   userId movieId days_u
##    <int>   <dbl> <chr>
## 1      1     122 1000
## 2      1     185 1000
## 3      1     292 1000
## 4      1     316 1000
## 5      1     329 1000
## 6      1     355 1000
```

I then checked all variables for missing values with negative results.

```
any(is.na(year))
```

```
## [1] FALSE
```

```
any(is.na(genre))
```

```
## [1] FALSE
```

```
any(is.na(age_movie_s))
```

```
## [1] FALSE
```

```
any(is.na(age_user_s))
```

```
## [1] FALSE
```

2.2 Data Exploration

The goal of the data exploration step was to analyze the data in order to get insights which could be used to build better models. To make it easier for you to follow through I gave this section the fixed structure.

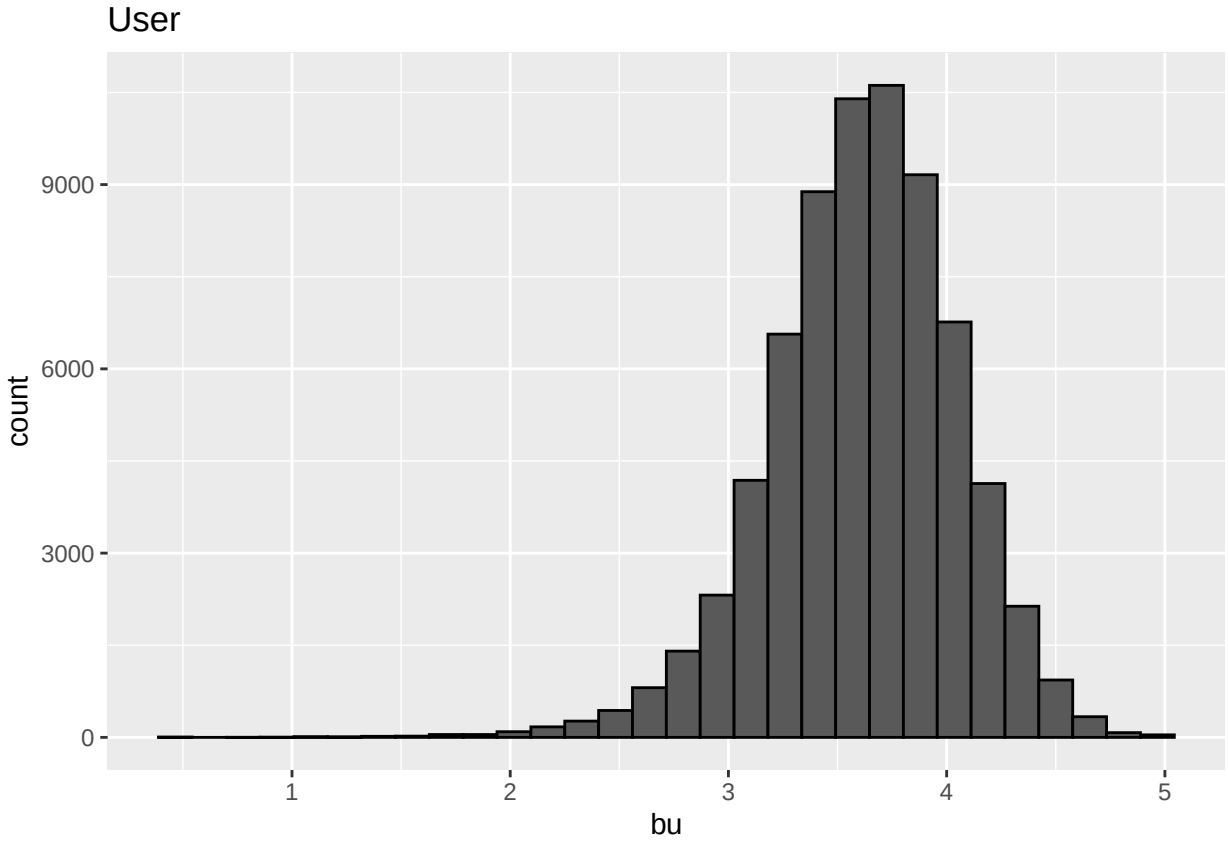
For each variable firstly I will give you overall information about the data structure. Secondly I will show you a visualization of of the data and a description of what we can see. And finally I will interpret the results, derive insights from it and will then conclude what it means for the modeling approach.

2.2.1 User Id

The overall structure of the variable "userId" looks like this:

```
## # A tibble: 6 x 2
##   userId     n
##    <int> <int>
## 1      1    19
## 2      2    17
## 3      3    31
## 4      4    35
## 5      5    74
## 6      6    39
```

The variable contains of 69878 specific numerical rows. Each one represents one specific user. You can see in the table that the times a specific user rated a movie differs a lot across user. The diagram below shows the rating distribution of all user.



Most users in the data set gave ratings between 3 and 4. But there are also some user with average ratings of 1 or 5 stars.

That means that the user effect (bu) should account for a significant part of the variance of the ratings. In other words. Including the "movie" variable in our model should increase the model performance.

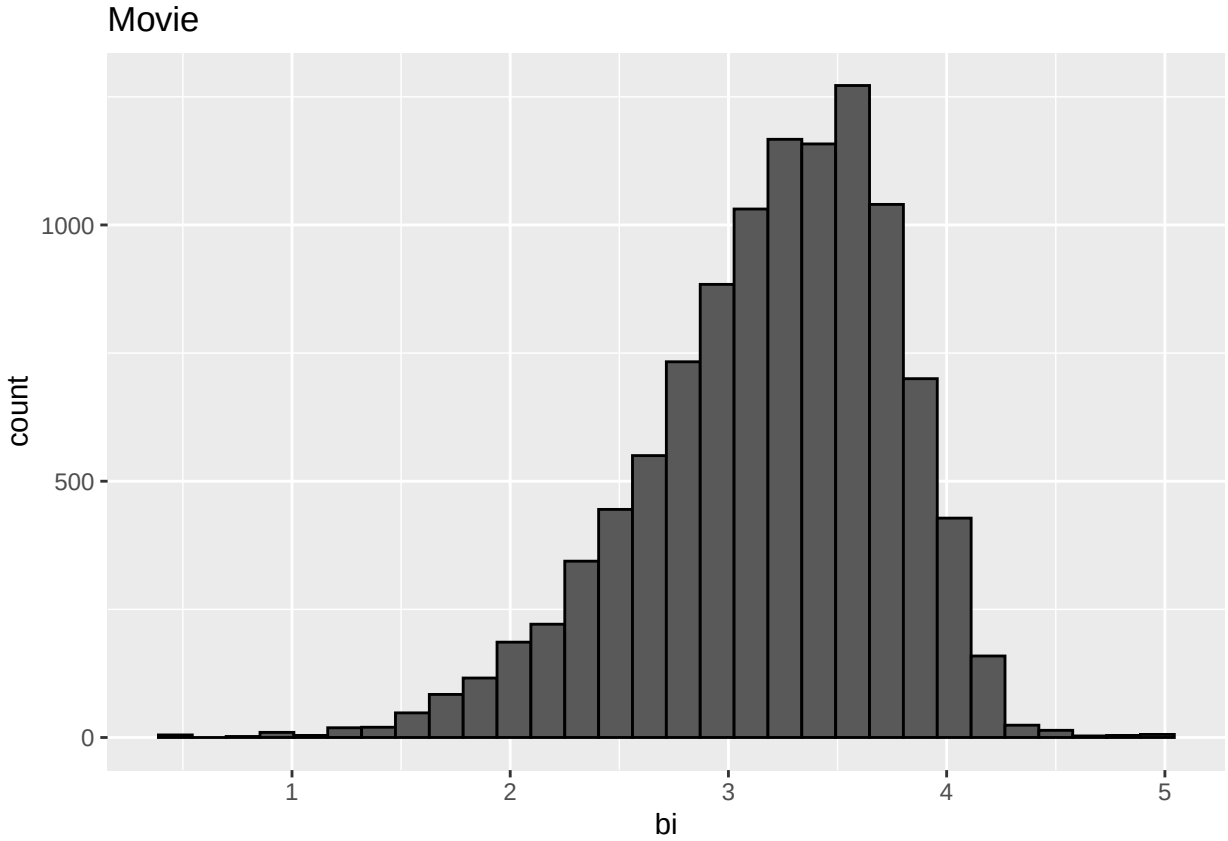
2.2.2 Movie Id

The table below shows an overview of the variable "movieId". The edx data set contains of 10677 different movies, each of them was rated at least from one user. But again you can see that there are movies that received substantially less or more ratings then the average movie.

```
## # A tibble: 6 x 2
##   movieId     n
##     <dbl> <int>
## 1       1 23790
## 2       2 10779
## 3       3  7028
## 4       4  1577
## 5       5  6400
## 6       6 12346
```

When we evaluate the distribution of ratings for different movies we see that most of them were rated on

average between 3 and 4 stars. Nevertheless there are also movies which average ratings of just one star or also movies that got the highest avarege rating of 5 stars.

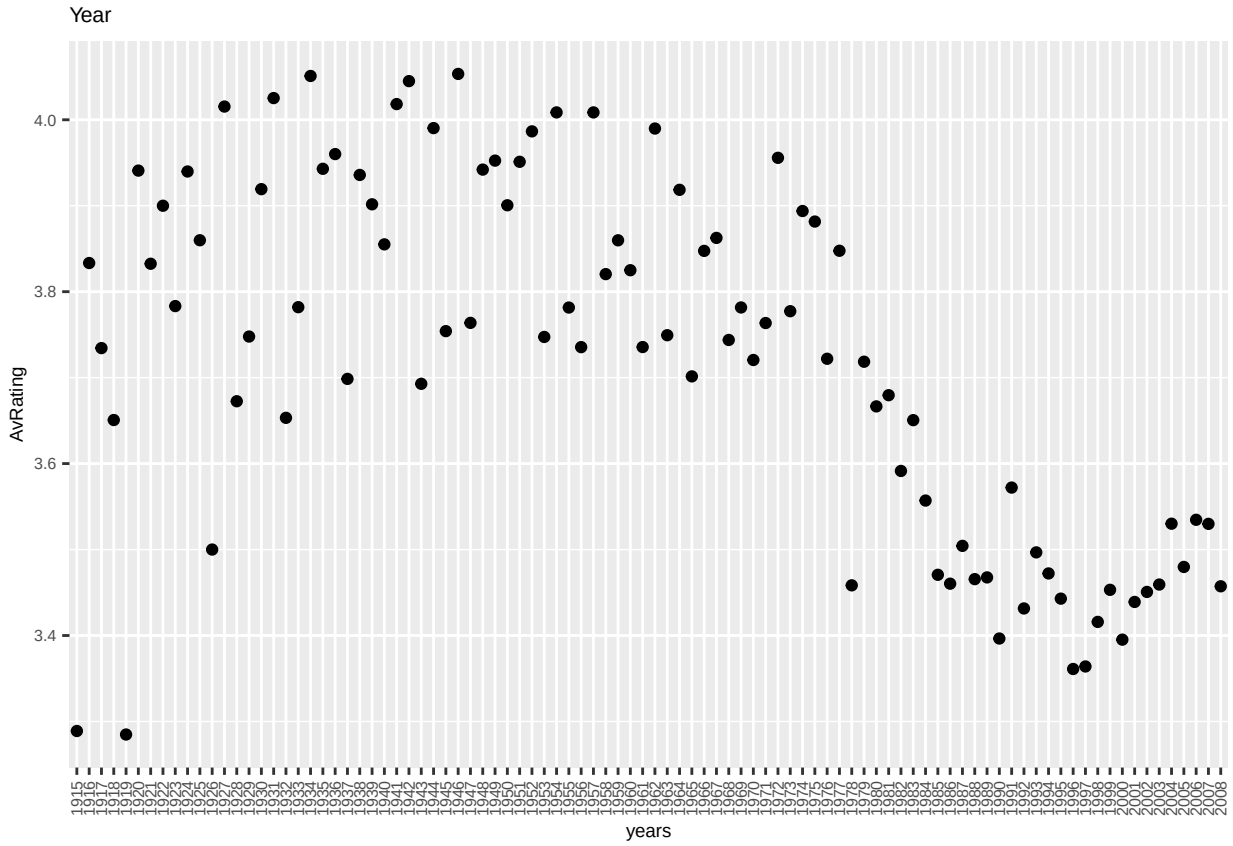


Given that plot it is likely that the movie effect plays a fundamental role in the predictability of movie ratings. So I assume it will account for a big part of the performance of our final model.

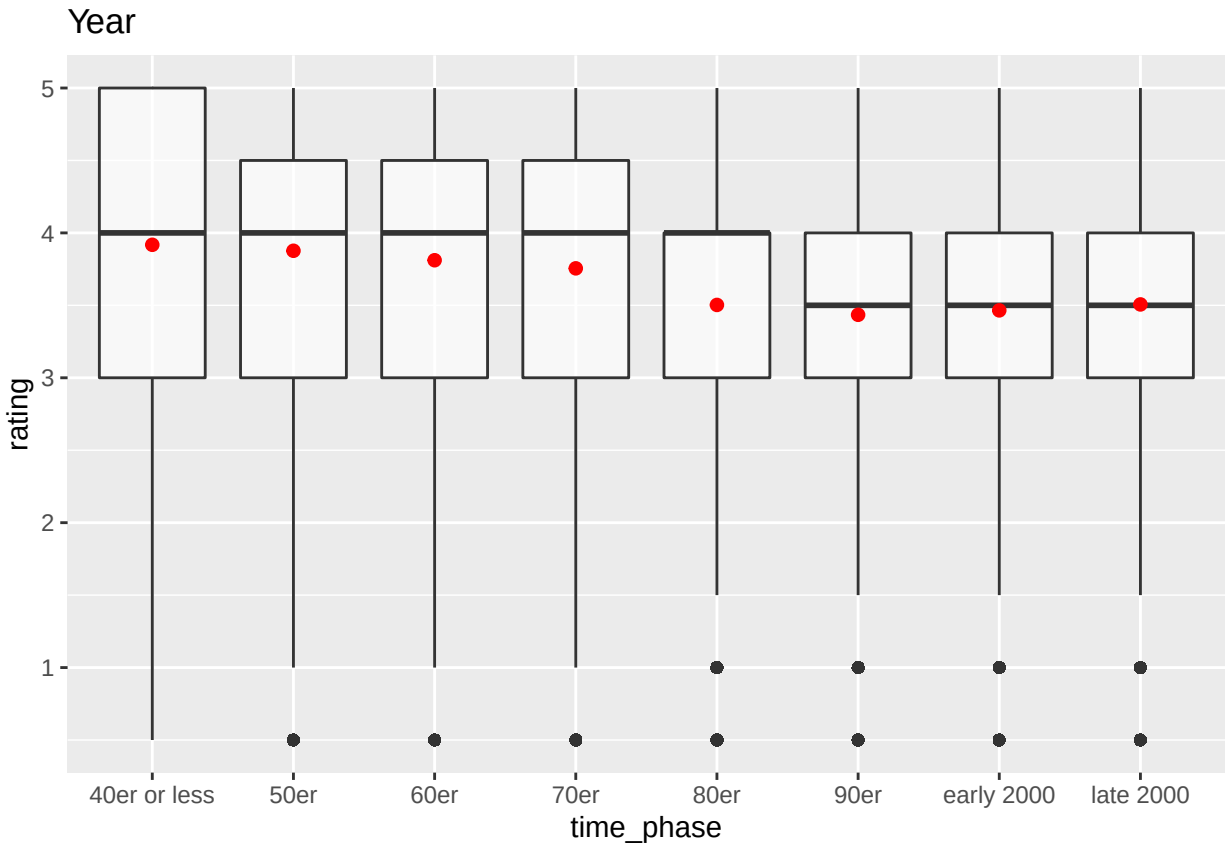
2.2.3 Year

The variable year represents the release year and was extracted from the title. The data consists of 10677 individual release years, one for each movie.

If you look at the plot shown below you will recognize that the average rating per year (by) negatively correlates with the year.



As “by” is only the average rating per year I would expect that very old movies have a greater variance since the sample size of ratings is small compared to years in the nearer past. To account for that issue I decided to cluster the year in different time phases. The boxplot diagram below shows the distribution. It appears that the average rating drops during the time phases. Nevertheless the effect is not very strong.



I decided to still include this variable in my models, in good hope that it will improve the performance to at least some amount.

2.2.4 Genres

The variable "genres" in the edx data set exist of 16 different genres. The fixed binding of genre combination results in 797 individual genre combinations.

```
edx %>% group_by(genres) %>% summarise(n=n()) %>% nrow()
```

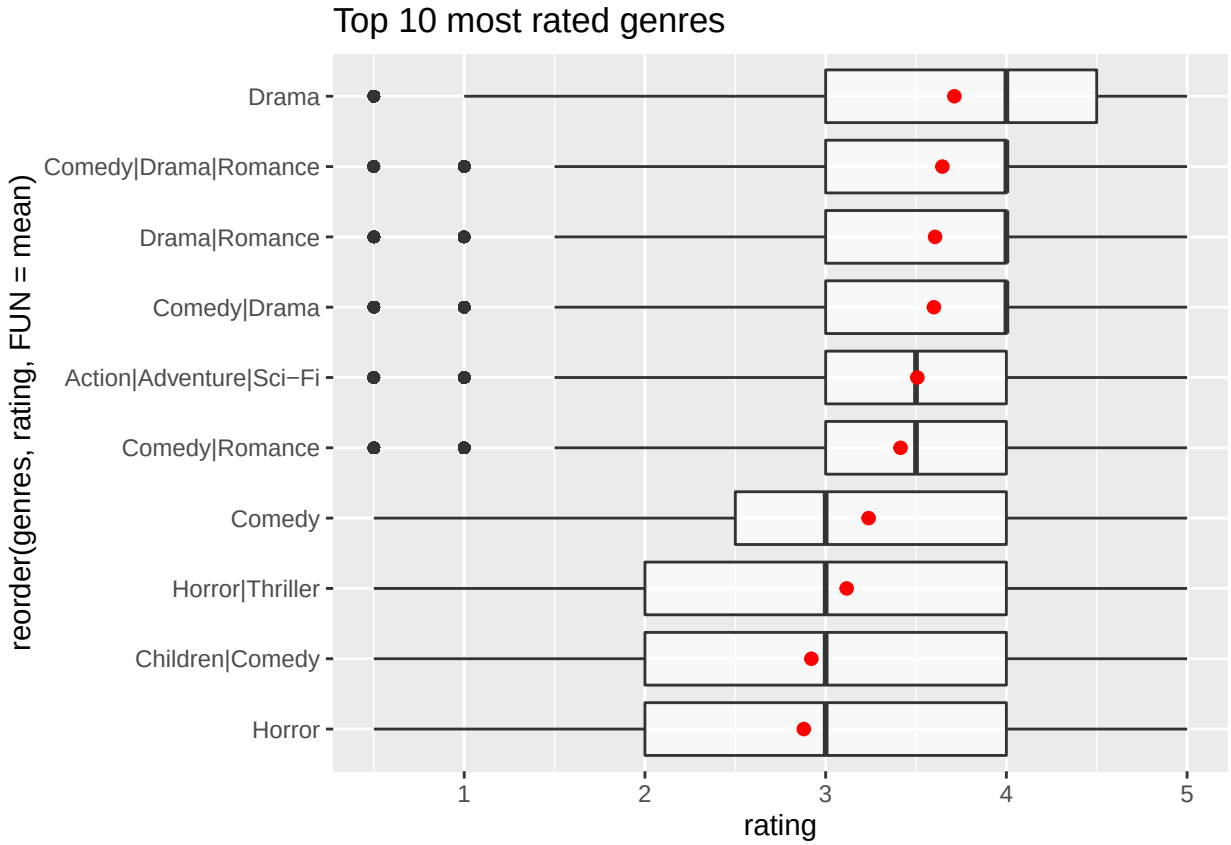
```
## [1] 797
```

The table below shows the most important genres as they received the most user ratings.

```
## # A tibble: 10 x 3
## # Groups:   genres [10]
##    genres                    avRating     n
##    <chr>                        <dbl> <int>
##  1 Comedy                        3.24 45299
##  2 Drama                         3.71 17751
##  3 Comedy|Romance                3.41 13637
##  4 Comedy|Drama                  3.60  9216
##  5 Horror                        2.88  9025
##  6 Action|Adventure|Sci-Fi       3.51  8371
##  7 Drama|Romance                 3.61  8353
```

```
##  8 Horror|Thriller             3.12  6796
##  9 Children|Comedy             2.92  6459
## 10 Comedy|Drama|Romance        3.65  6143
```

The average rating varies across these genres which indicates some amount of predictive power. To clarify this I calculated a boxplot chart with the corresponding underlying data.



T his chart highlights very clearly that both, the variance as well as the mean varies across those genres.

Thus I will include the variable "genres" in my models.

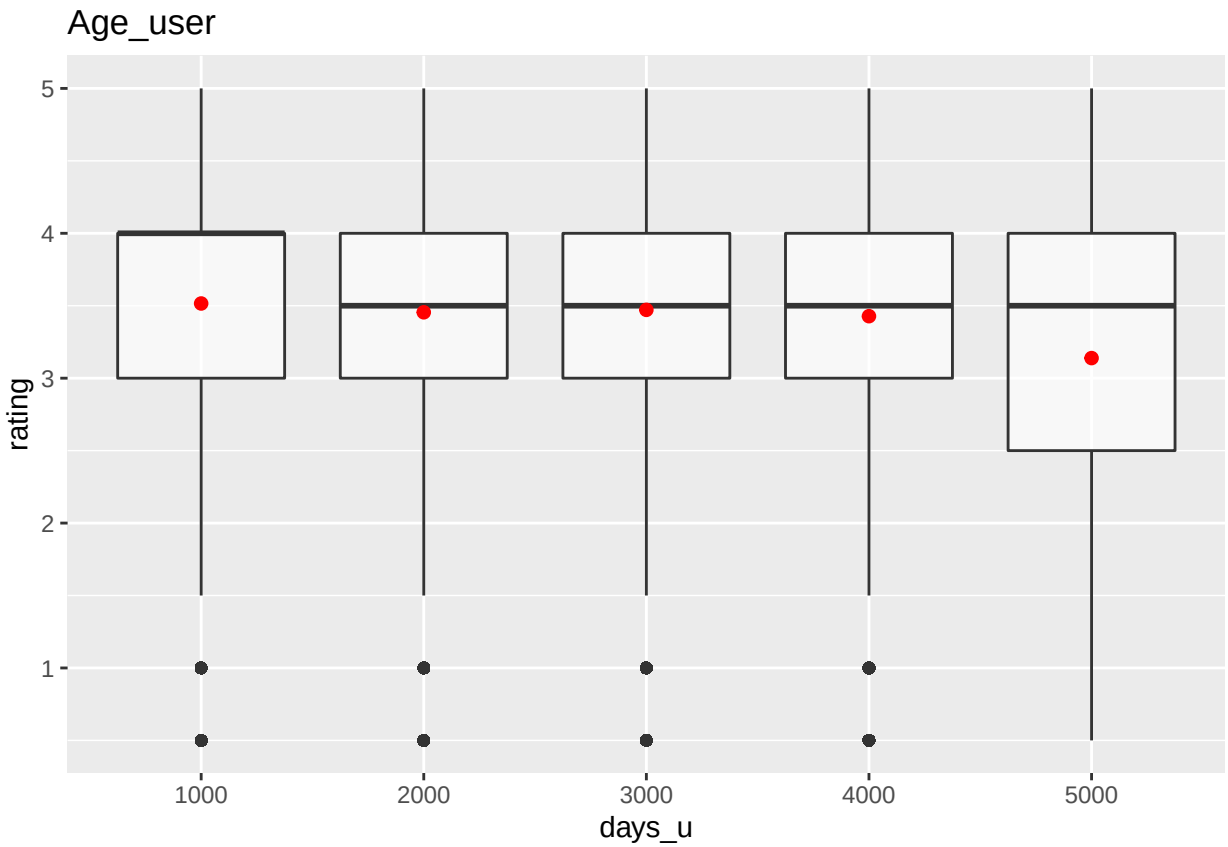
2.2.5 Age effect

As the age variables measure the past days since the first rating from each user (variable "age_user") or for each movie (variable "age_movie") respectively, both variables have 7200091 items. Below I looked at each variable separately:

1) **Age_user** The following table shows the prevalence of ratings within each defined cluster:

```
## # A tibble: 5 x 2
##   days_u       n
##   <chr>    <int>
## 1 1000   8522022
## 2 2000    356815
## 3 3000    106436
## 4 4000     14555
## 5 5000       227
```

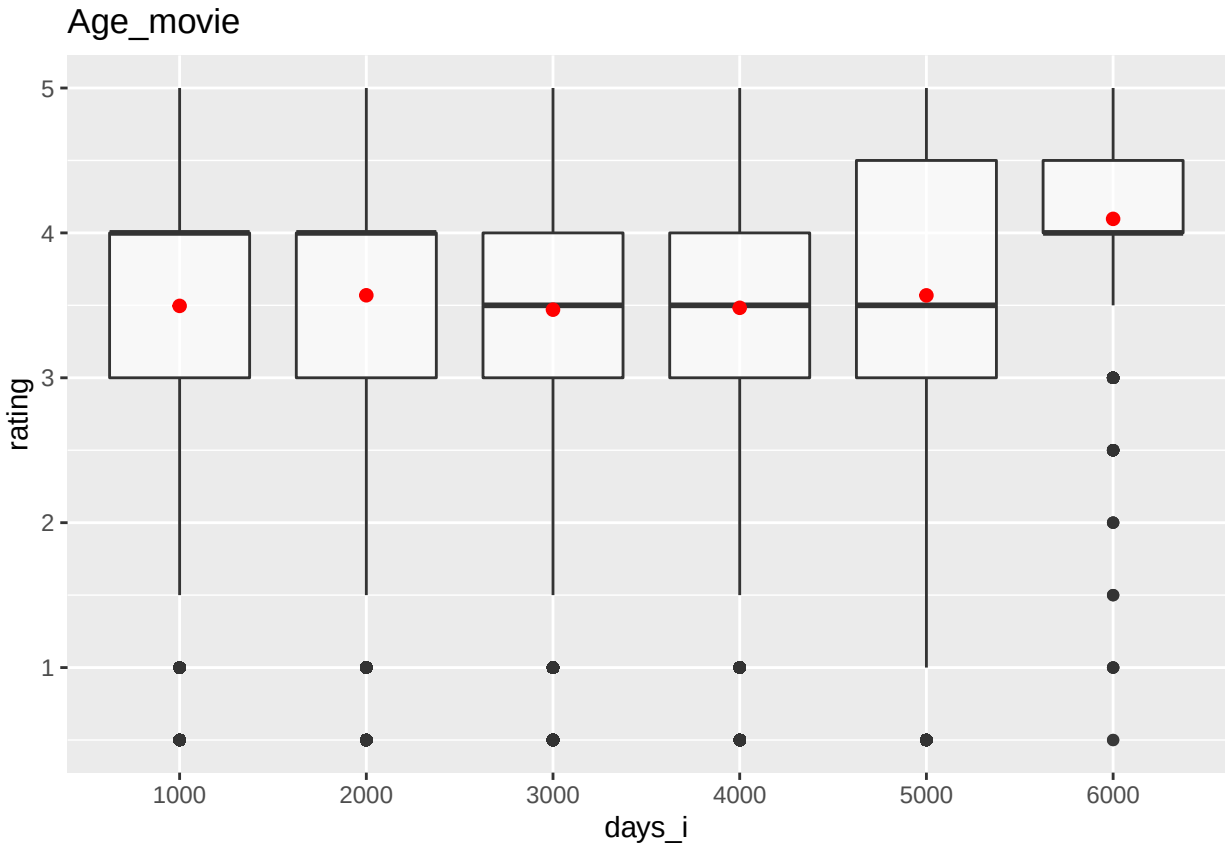
What we see is that the distribution is very skewed and the most of the values are in the first, second or third cluster. To evaluate the variance within each cluster and to compare the defined cluster I calculated a corresponding boxplot chart.



As you can derive from the figure above the variable "age_user" has nearly zero variance across all cluster. Only the last cluster shows some degree of variance and a slight shift of the mean. However as we have seen above the last cluster has no weight in the data set as it accounts for only a very small amount of ratings. Therefore I decided to not use these variable any further.

2) **Age_movie** I performed the same analysis for the second age variable.

```
## # A tibble: 6 x 2
##   days_i       n
##   <chr>    <int>
## 1 1000   4128332
## 2 2000   2322611
## 3 3000   1276322
## 4 4000    966789
## 5 5000    305185
## 6 6000       816
```



We can see a similar pattern here. However the variance within the cluster above 5000 days appears to be more stable and accounts for far more ratings. These are the reasons I decided to include this variable in my models.

2.3 Modeling approach

2.3.1 Baseline model

A typical modeling approach as described in the book "Super Forecasting" by Philip Tetlock and Dan Gardner (2015) contains of a baseline prediction as well as additional terms which can decrease or increase the baseline prediction and accounts for additional identified biases within the data set (or additional specific information if the context is different from machine learning). Another term, called the error term, will account for all the variance that can not be explained by the data. As we will consider the predictors "movieId", "userId", "year" (time phase), "genres" and "age_movie" in our model it can be formally written as:

y_hat = mu + bi + bu + by + bg + ba + e

What follows is a short description of the formula parts:

- **y_hat** - the prediction of a specific rating
- **mu** - the mean of all ratings in the training data set
- **bi** - the movie bias, which represents the variance of ratings due to a movie effect
- **bu** - the user bias, which represents the variance of ratings due to a user effect
- **by** - the year bias, which represents the variance of ratings due to a time phase effect
- **bg** - the genre bias, which represents the variance of ratings due to a genre effect
- **ba** - the age bias, which represents the variance of ratings that can be explained with the days past since a specific movie received its first rating

2.3.2 Regularization

When we calculate biases, eg. the user bias, we analyze specific (user) ratings in order to find correlations between specific users (groups of users) and their related ratings. The smaller the sample, the less robust will be the bias. Or in other words. We should only trust the calculated biases, and thus adapt our baseline prediction, if the pattern is based on a great enough sample so the effect is robust.

To make sure we give more robust biases a greater weight we will use the regularization concept and constrain the total variability of the effect sizes. We will use lambda as a the regularization parameter that shrinks the deviation from the mean (baseline) towards zero. In order to find the best lambda I minimized the following formula, where y is the real value (rating):

```
# (y - mu - bi - bu - by - bg - ba)^2 + lambda(bi^2 + bu^2 + by^2 + bg^2 + ba^2)
```

2.3.3 Data Partition

To ensure that we can properly evaluate our models in order to find the final model we separated our edx data set in two parts:

1) **training_set** - 80% of all edx items, randomly picked to train our models, tune the model parameter and define the final model
2) **test_set** - 20 % of the edx data set, randomly picked to evaluate the final model

The figure below shows the corresponding R code:

```
set.seed(1234, sample.kind = "Rounding")
```

```
## Warning in set.seed(1234, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
test_index <- createDataPartition(edx$rating,times = 1,p=0.2,list = F)
train_set <- edx[-test_index,]
test_set0 <- edx[test_index,]

test_set <- test_set0 %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId") %>%
  semi_join(train_set, by="genres")

removed <- anti_join(test_set0, test_set)
edx <- rbind(train_set, removed)

rm(removed, test_set0)
```

As soon as I defined the final model I rerun the code with the complete edx data set and evaluated the model with the validation set. This operation delivered the final RMSE.

3. Results

In this section I will show the results of all models I trained during the training phase. I trained them first on the train_set and tuned possible model parameters. In a second step I evaluated every model on the

validation set in order to compare results with all course participants. Of course I never used the validation set for training purposes as I know this would yield to over-fitting.

The simplest prediction would be to use the average of all ratings in the edx set to predict the ratings in the validation set:

```
## [1] 3.512565
```

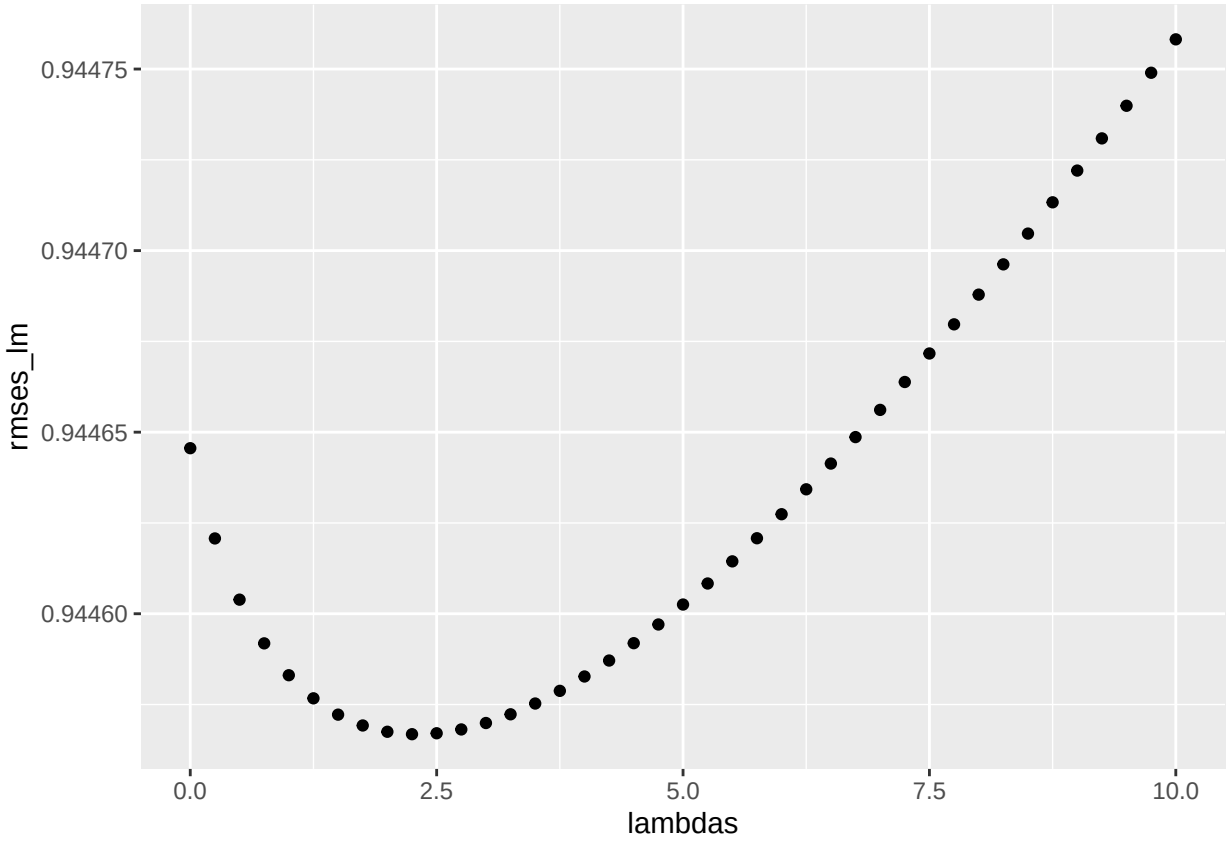
I stored all results in a table called "rmse_results".

```
## # A tibble: 1 x 2
##   Model    RMSE
##   <chr>   <dbl>
## 1 Average  1.06
```

So the first approach would yield to an rmse greater then 1, which means that our predictions would be on average 1 star above or below the real rating. I think there is much room for improvement.

3.1 Model 1 - movie effect and regularization

The first real model I trained was the linear model with regularization on the variable "movieId". I used cross-validation to find the best lambda.




```
## [1] 2.25
```

You can see that lambda = 2.25 minimizes the rmse and was the best tuning parameter. With the code below I trained the final model.

```
mu <- mean(edx$rating)

  bi <- edx %>%
    group_by(movieId) %>%
    summarize(bi = sum(rating - mu)/(n()+2.25))

  predicted_ratings_1 <- validation %>%
    left_join(bi, by = "movieId") %>%
    mutate(pred = mu + bi) %>%
    pull(pred)
```

Assuming I would submit this model as my final one I would reach the following rmse on the validation set:

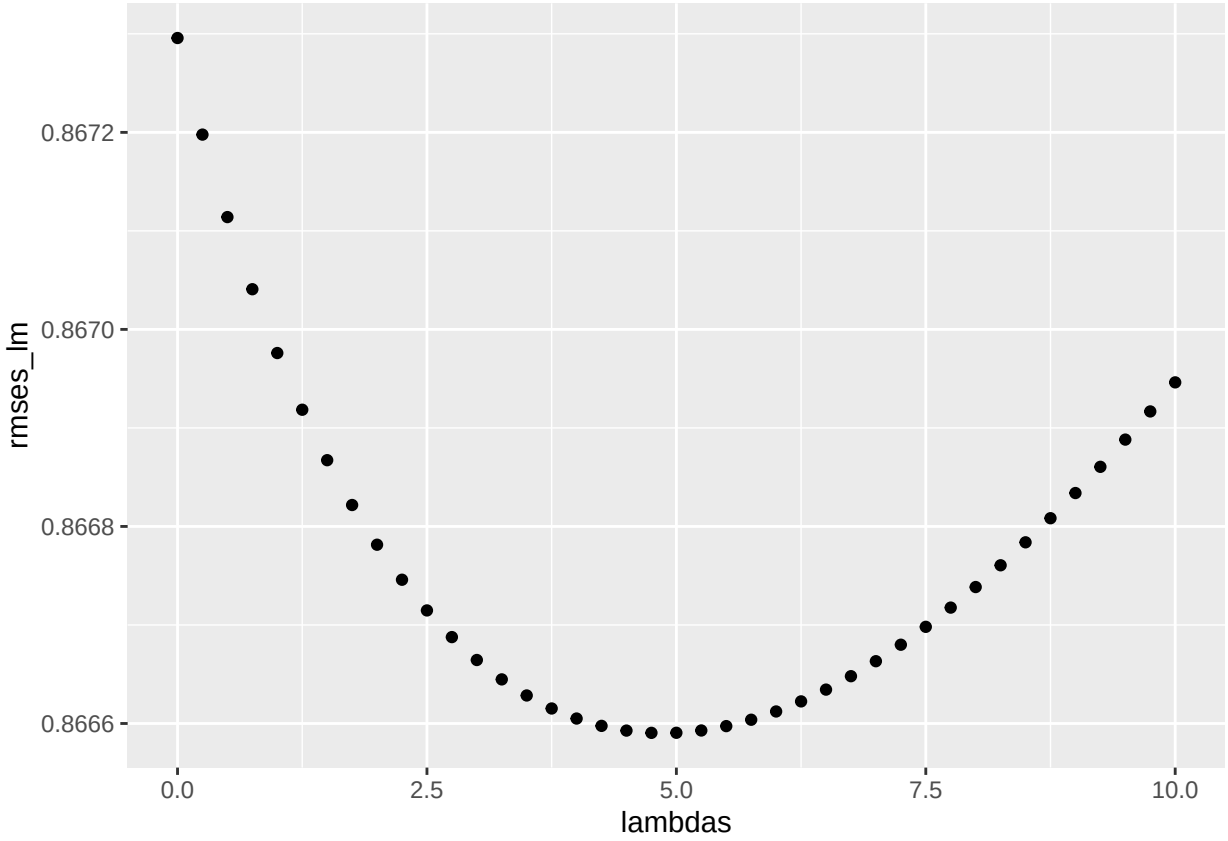
```
rmse_1 <- RMSE(predicted_ratings_1,validation$rating)

rmse_results <- bind_rows(rmse_results,
                          tibble(Model="Movie effect and reg",
                                     RMSE = rmse_1))
rmse_results
```

```
## # A tibble: 2 x 2
##   Model                 RMSE
##   <chr>                <dbl>
## 1 Average              1.06
## 2 Movie effect and reg 0.944
```

The first model archived an rmse of approximately 0.94, which is significantly better then just the average.

3.2 Model 2 - movie effect, user effect and regularization

I repeated the procedure above a couple of times, always adding another variable to the model to measure the impact on the model performance. In model 2 I included the variable “userId”.




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

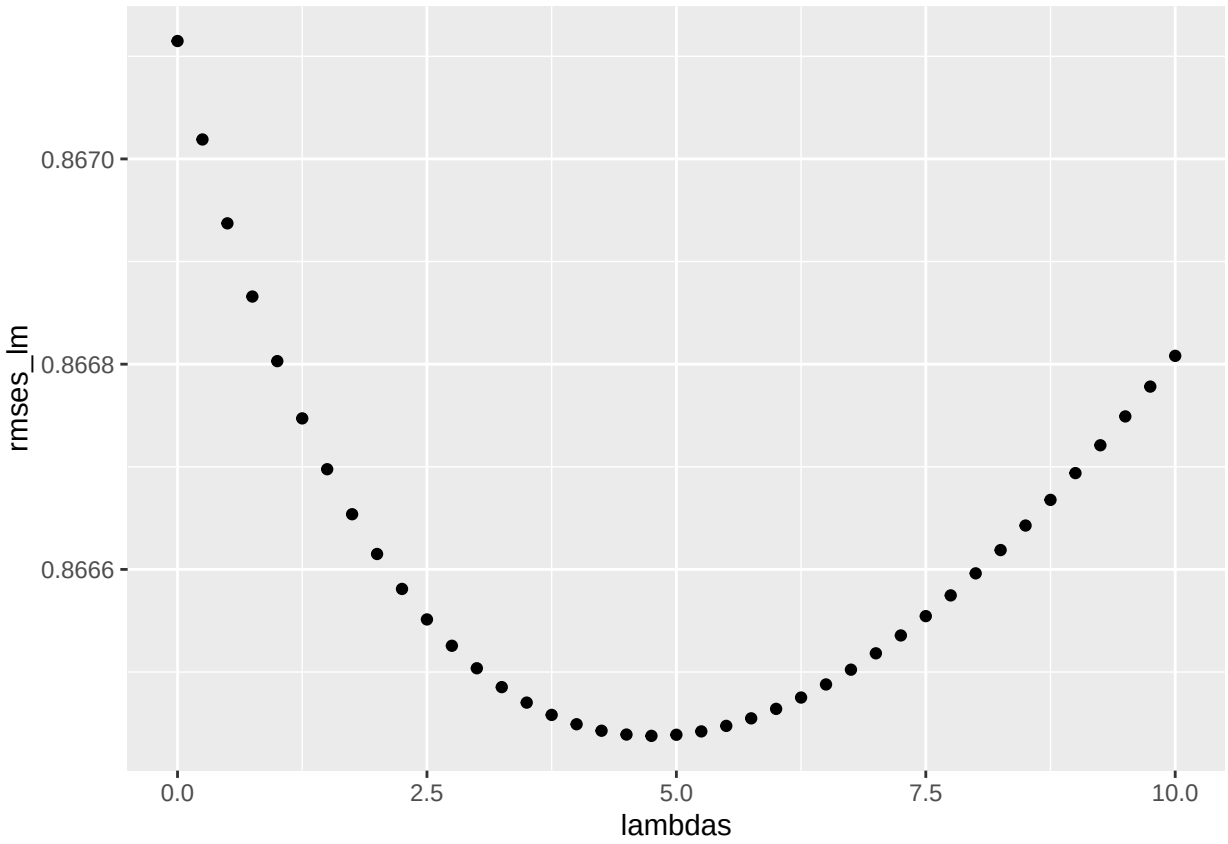
So the best value for the tuning parameter lambda turned out to be 4.75. Let us check how model 2 would perform on the validation set.

The user effect reduced the rmse significantly. That means we observe a strong user effect within the data. Or in other words, the variable "userId" had strong predictive power.

```
## # A tibble: 3 x 2
##   Model                              RMSE
##   <chr>                             <dbl>
## 1 Average                           1.06
## 2 Movie effect and reg              0.944
## 3 Movie effect, user effect and reg 0.866
```

3.3 Model 3 - movie effect, user effect, time effect and regularization

In model 3 I added the variable "time_phase" to model 2.




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

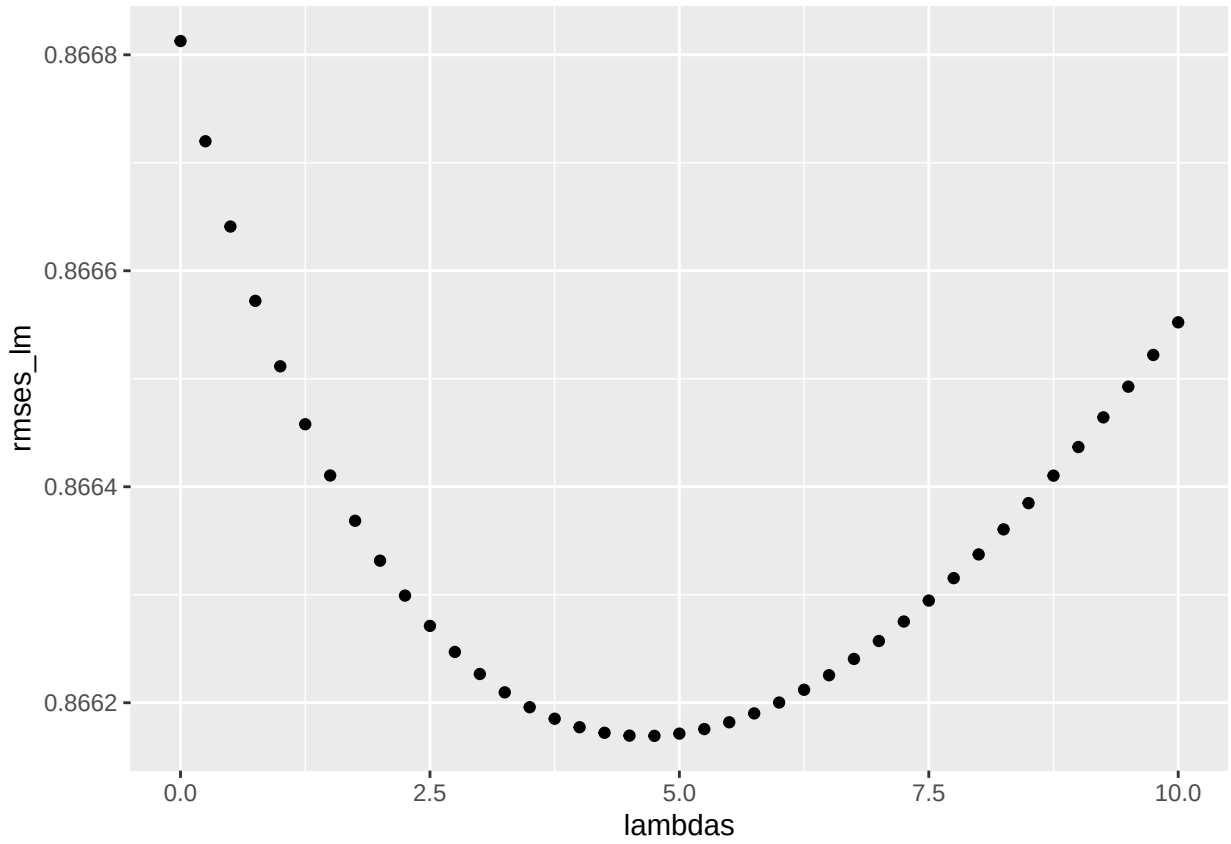
As before the best value for the tuning parameter was 4.75.

The added time variable had nearly no effect at all. The rmse improved just a tiny bit. Focusing on the pre-processing-process could had helped to separate the signal better from the noise. Nevertheless I thought this variable had just not that much predictive power overall.

```
## # A tibble: 4 x 2
##   Model                                                RMSE
##   <chr>                                               <dbl>
## 1 Average                                             1.06
## 2 Movie effect and reg                                0.944
## 3 Movie effect, user effect and reg                   0.866
## 4 Movie effect, user effect, time phase effect and reg 0.866
```

3.4 Model 4 - movie effect, user effect, time effect, genre effect and regularization

In this section I added the genre variable in their raw form.




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

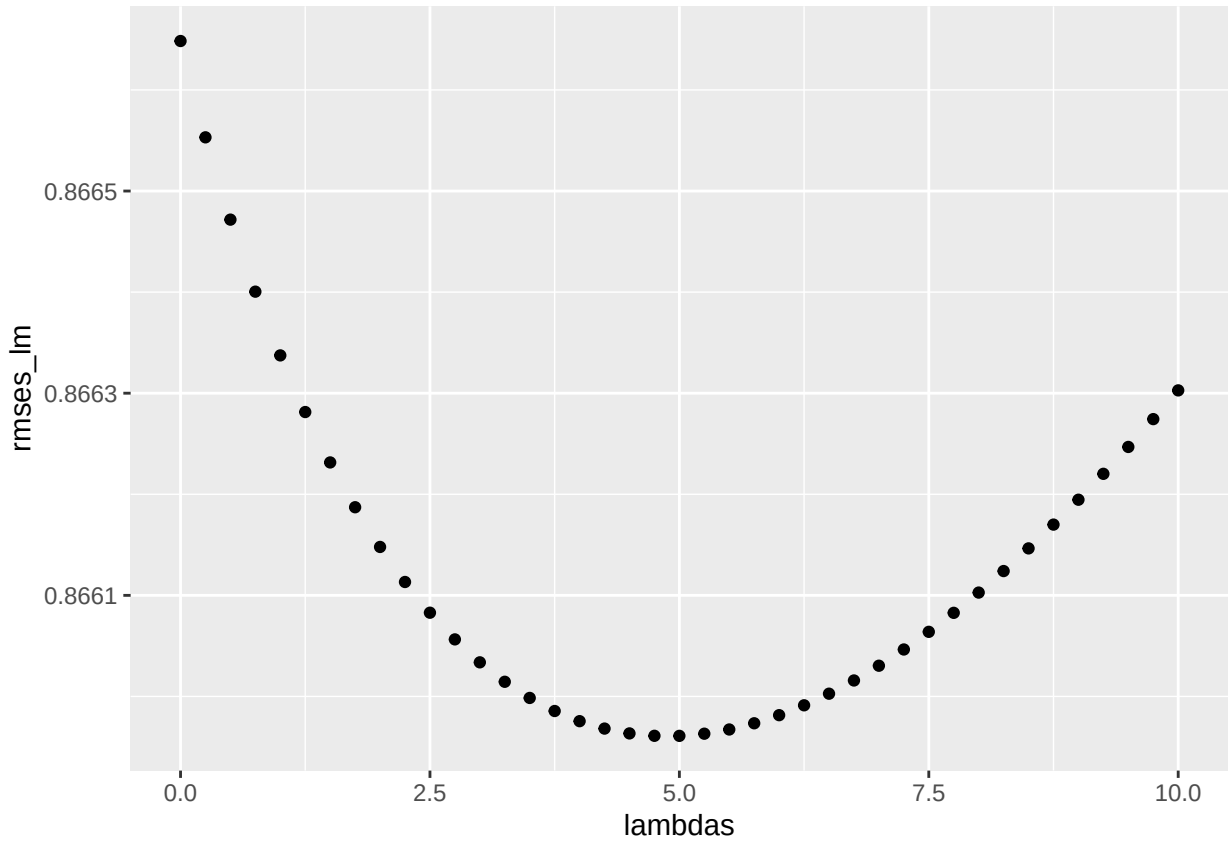
The tuning parameter lambda did not change.

Again the added variable genre decreased the rmse just slightly and had no significant effect on the model performance.

```
## # A tibble: 5 x 2
##   Model                                                          RMSE
##   <chr>                                                         <dbl>
## 1 Average                                                       1.06
## 2 Movie effect and reg                                          0.944
## 3 Movie effect, user effect and reg                             0.866
## 4 Movie effect, user effect, time phase effect and reg          0.866
## 5 Movie effect, user effect, time phase effect, genre effect and reg 0.865
```

3.5 Model 5 - movie effect, user effect, time effect, genre effect, age effect and regularization

Finally I included the variable "age_movie" variable to the model and was excited about whether it would push us over the rmse target line of < 0.8649.



The tuning parameter changed to 5.

```
bi <- edx %>%
  group_by(movieId) %>%
  summarize(bi = sum(rating - mu)/(n()+5))

bu <- edx %>%
  left_join(bi, by="movieId") %>%
  group_by(userId) %>%
  summarize(bu = sum(rating - bi - mu)/(n()+5))

by <- edx %>%
  left_join(bi, by="movieId") %>%
  left_join(bu, by="userId") %>%
  left_join(year_s, by="movieId") %>%
  group_by(time_phase) %>%
  summarize(by = sum(rating - bi - bu - mu)/(n()+5))

bg <- edx %>%
  left_join(bi, by="movieId") %>%
  left_join(bu, by="userId") %>%
  left_join(year_s, by="movieId") %>%
  left_join(by,by="time_phase") %>%
  group_by(genres) %>%
  summarize(bg = sum(rating - bi - bu - by - mu)/(n()+5))

ba <- edx %>%
```

```
  left_join(bi, by="movieId") %>%
  left_join(bu, by="userId") %>%
  left_join(year_s, by="movieId") %>%
  left_join(by, by="time_phase") %>%
  left_join(bg, by="genres") %>%
  left_join(age_movie_s, by=c("userId","movieId")) %>%
  group_by(days_i) %>%
  summarize(ba = sum(rating - bi - bu - by - bg - mu)/(n()+5))

prediction_5 <- validation %>%
  left_join(bi, by = "movieId") %>%
  left_join(bu, by = "userId") %>%
  left_join(year_s, by="movieId") %>%
  left_join(by, by = "time_phase") %>%
  left_join(bg, by = "genres") %>%
  left_join(firstRating_movie ,by="movieId") %>%
  mutate(timestamp=as_datetime(timestamp)) %>%
  mutate(age_i = round(difftime(timestamp,firstRating_movie,units = "days"))) %>%
  mutate(days_i=ifelse(age_i<=1000,"1000",
                       ifelse(age_i<=2000,"2000",
                              ifelse(age_i<=3000,"3000",
                                     ifelse(age_i<=4000,"4000",
                                            ifelse(age_i<=5000,"5000","6000")))))) %>%
  left_join(ba, by="days_i") %>%
  mutate(pred = mu + bi + bu + by + bg + ba) %>%
  pull(pred)

rmse_5 <- RMSE(prediction_5, validation$rating)
```

It turned out that the added variable "age_movie" reduced the rmse further. However not enough to reach the performance to earn the highest grade. In order to reach that goal I had to describe, test and train a new model, that focuses on matrix factorization.

```
## # A tibble: 6 x 2
##   Model                                                                RMSE
##   <chr>                                                               <dbl>
## 1 Average                                                             1.06
## 2 Movie effect and reg                                                0.944
## 3 Movie effect, user effect and reg                                   0.866
## 4 Movie effect, user effect, time phase effect and reg                0.866
## 5 Movie effect, user effect, time phase effect, genre effect and reg  0.865
## 6 Movie effect, user effect, time phase effect, genre effect, age effect ~ 0.865
```

3.6 Model 6 - matrix factorisation

I decided to try the matrix factorization technique as it is well known as a good fit for recommender system and it was also a topic we discussed in our course (see Introduction to Data Science, Rafael A. Irizarry, 2020).

The goal of the algorithm is to decompose a user-item interaction matrix into the product of two rectangular matrices with lower dimensions. So linear algebra is used to reduce the complexity of the matrix and to calculate factors like movie popularity or user activeness and their specific weights. I will not describe the

whole concept mathematically in this report. But interested readers can find a good introduction into that topic in the recommended course book above.

3.6.1 Data Preparation

To build a matrix factorization model I used the "recosystem" package. The package was developed by Yu-Chin Juan, Wei-Sheng Chin, Yong Zhuang, Bo-Wen Yuan, Meng-Yuan Yang, and Chih-Jen Lin in 2015 and can be installed and loaded similar to other packages at the Rstudio environment. I picked this package, because it is designed for matrix factorization problems and also very resource efficient, which is fundamentally important in our case (think about how big our data set is!).

Three steps were necessary to prepare the data for the training phase:

1) The recosystem package needs the data to be in a sparse matrix triplet form. Meaning both the training set and the validation set needed to have 3 columns. One for the user, one for the movie and the third for the rating.

```
edx_m <- edx %>% dplyr::select(userId,movieId, rating)
edx_m <- as.matrix(edx_m)

validation_m <- validation %>% dplyr::select(userId, movieId,rating)
validation_m <- as.matrix(validation_m)
```

2) It was also necessary to write the data into a hard disc before training the model to ensure efficiency.

```
write.table(edx_m ,file = "trainset.txt",sep = " " ,row.names = F,col.names = F)
write.table(validation_m,file = "validset.txt",sep = " " ,row.names = F, col.names = F)
```

3) The final data sets had to be generated

```
set.seed(202)
training <- data_file("trainset.txt")
validating <- data_file("validset.txt")
```

3.6.2 Building the model

First of all I built a recommender model using this code:

```
r <- Reco()
```

Then I tuned the training set to get the best possible parameter for the training phase:

```
opts <- r$tune(training, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                     costp_l1 = 0, costq_l1 = 0,
                                     nthread = 1, niter = 10))
opts
```

```
## $min
## $min$dim
## [1] 30
```

```
##
## $min$costp_l1
## [1] 0
##
## $min$costp_l2
## [1] 0.01
##
## $min$costq_l1
## [1] 0
##
## $min$costq_l2
## [1] 0.1
##
## $min$lrate
## [1] 0.1
##
## $min$loss_fun
## [1] 0.8055348
##
##
## $res
##    dim costp_l1 costp_l2 costq_l1 costq_l2 lrate  loss_fun
## 1   10        0     0.01        0     0.01   0.1 0.8309401
## 2   20        0     0.01        0     0.01   0.1 0.8191950
## 3   30        0     0.01        0     0.01   0.1 0.8309894
## 4   10        0     0.10        0     0.01   0.1 0.8326496
## 5   20        0     0.10        0     0.01   0.1 0.8137919
## 6   30        0     0.10        0     0.01   0.1 0.8147565
## 7   10        0     0.01        0     0.10   0.1 0.8330510
## 8   20        0     0.01        0     0.10   0.1 0.8111756
## 9   30        0     0.01        0     0.10   0.1 0.8055348
## 10  10        0     0.10        0     0.10   0.1 0.8408312
## 11  20        0     0.10        0     0.10   0.1 0.8338562
## 12  30        0     0.10        0     0.10   0.1 0.8304942
## 13  10        0     0.01        0     0.01   0.2 0.8740454
## 14  20        0     0.01        0     0.01   0.2 1.0137669
## 15  30        0     0.01        0     0.01   0.2 0.9369970
## 16  10        0     0.10        0     0.01   0.2 0.8260700
## 17  20        0     0.10        0     0.01   0.2 0.9181461
## 18  30        0     0.10        0     0.01   0.2 0.9195215
## 19  10        0     0.01        0     0.10   0.2 0.8291477
## 20  20        0     0.01        0     0.10   0.2 0.8098741
## 21  30        0     0.01        0     0.10   0.2 0.8114899
## 22  10        0     0.10        0     0.10   0.2 0.8438771
## 23  20        0     0.10        0     0.10   0.2 0.8290994
## 24  30        0     0.10        0     0.10   0.2 0.8748880
```

I finally trained the matrix factorization model using the parameters above that minimizes the rmse.

```
r$train(training,opts = c(opts$min,nthread=1,niter=20))
```

```
## iter      tr_rmse          obj
##    0       0.9943   1.0066e+07
```

```
##    1       0.8783   8.0821e+06
##    2       0.8465   7.5030e+06
##    3       0.8255   7.1625e+06
##    4       0.8087   6.9117e+06
##    5       0.7955   6.7310e+06
##    6       0.7843   6.5886e+06
##    7       0.7747   6.4747e+06
##    8       0.7664   6.3779e+06
##    9       0.7592   6.3004e+06
##   10       0.7529   6.2327e+06
##   11       0.7473   6.1764e+06
##   12       0.7423   6.1253e+06
##   13       0.7378   6.0825e+06
##   14       0.7337   6.0432e+06
##   15       0.7299   6.0097e+06
##   16       0.7265   5.9766e+06
##   17       0.7233   5.9511e+06
##   18       0.7204   5.9245e+06
##   19       0.7177   5.9031e+06
```

3.6.3 Validation of the model

As I had the final model I used the predict function on the r object to generate the prediction for the validation set. Then I run these predictions through our RMSE function in order to calculate the final rmse_mf.

```
pred_file <- tempfile()
r$predict(validating,out_file(pred_file))
predicted_ratings_mf <- scan(pred_file)

rmse_mf <- RMSE(predicted_ratings_mf, validation$rating)
```

```
## # A tibble: 7 x 2
##   Model                                                                RMSE
##   <chr>                                                               <dbl>
## 1 Average                                                             1.06
## 2 Movie effect and reg                                                0.944
## 3 Movie effect, user effect and reg                                   0.866
## 4 Movie effect, user effect, time phase effect and reg                0.866
## 5 Movie effect, user effect, time phase effect, genre effect and reg  0.865
## 6 Movie effect, user effect, time phase effect, genre effect, age effect ~ 0.865
## 7 Matrix factorization                                                0.790
```

I was surprised about the result and how much the matrix factorization model improved the rmse. I could archive a really good result of approximately 0.79 on the validation set. That is an improvement of nearly 8% in comparison to model 5. Note that I only used the variable “userId” and “movieId” in matrix factorization model.

4. Conclusion

In summary I trained and tuned 6 different models on the training data set and validated them on the validation set. It turned out, that linear models are appropriate for our data in general. Especially to built a good baseline model, that is highly superior compared to just predicting the average. But the final goal of an rmse of less then 0.86490 could I only archive through introducing the matrix factorization technique that got me to an rmse of 0.79.

The matrix factorization model turned out to be superior to linear models in gathering the signal when it comes to the dynamic interaction of user, movie pairs. The reason is that this technique is able to capture factors, eg. user activeness or movie popularity. I think a combination of both techniques, the linear models for the baseline prediction and matrix factorization for the more dynamic relationship, would yield to even better results.

If I were able to spent more time on that project I would have focused longer on the data pre-processing step. I think there is a great potential to isolate the signals of different predictors further. I read about "feature engineering" and it seems like it is a good approach to harvest the most possible amount of the signal from a specific predictor. Thus I will dig deeper into that topic in the future.

For further improvements I will also suggest to include ensembles of models, because it is very likely that the "many model approach", especially if the individual models are uncorrelated, improve the performance compared to even the best single model.

Actually I tried to train some ensemble models with the "caretEnsemble" package but was not able to run the code due to limitations of my computer resources.

To sum up my report I have to say that I really liked the project, learned a lot and want to thank the whole instructor team, especially Prof. Irizarry, for putting together such a valuable program. I also want to thank all my peers for the interesting discussions (even if online) and their support.