MovieLens Capstone

Mohammed Amine BOUSSETTA

08/04/2019

Introduction

The capstone project in the data science series of Harvardx is about creating a recommander system based on the Movielens dataset and will be evaluated using the root mean squared error metric. As defined by Wikipedia, a recommender system is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item. Those systems are used in many areas such as social networking, online markets, search queries, financial services and so many others… Our project focuses on using this system to predict a user's rating of a certain movie.

One of the events that energized research in recommender systems was the Netflix Prize. The company offered a 1M $ prize to the team who can build a recommander system 10% more accurate than those offered by Netflix at the time. On 21 September 2009, the prize was announced and was given to BellKor's Pragmatic Chaos team.

The MovieLens data sets were collected by the GroupLens Research Project at the University of Minnesota. Each user has rated at least 20 movies. The data was collected through the MovieLens web site (movielens.umn.edu) during the seven-month period from September 19th, 1997 through April 22nd, 1998. In this project, we will be working with the 10M version of the data set which contains 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users.

This report will be structured in three parts: the first one will be dedicated to getting the data ready to be cleaned, explored and analysed to get first thoughts about feature engineering and models to be chosen. The next part will cover the modeling approches used and their impact on the rmse results. The last part will discuss further approches to be tested in order to improve the scores.

Data exploration and analysis

1- Prepare the data

This piece of code was provided by edX to download the dataset and reorganise it into two data frames ("edx", "validation") containing the movie ID, the user ID, the rating and the timestamp variables.

```r
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- read.table(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                      col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data

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

# Make sure userId and movieId in validation set are also in edx set

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

# Add rows removed from validation set back into edx set

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

save_edx <- edx
save_valid <- validation

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

Let's see now how the data looks like:

```
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                          genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7       Children|Comedy|Fantasy
```

The dataset contains 9000055 and 6 variables:

- `userId`: Unique identification number for each user. `numeric` variable
- `movieId`: Unique identification number for each movie. `numeric` variable.
- `timestamp`: Code representing the date and time when a user rated a movie. `integer` variable.
- `title`: Title of the movie. `character` variable.

- `genres`: Motion-picture category associated to the film. `character` variable.

- `rating`: Rating given by the user to the movie. From 0 to 5 *stars* in steps of 0.5. `numeric` variable.

2- Processing the data

Let's first check the type of our variables and see if they need to be converted in other types to facilitate the modeling part or simply make the executions easier:

```
sapply(edx, class)
```

```
##      userId     movieId      rating   timestamp       title      genres
##   "integer"   "numeric"   "numeric"   "integer" "character" "character"
```

We remark that the userID and the movieID variables are not ordinal i.e. there is no order in the IDs and comparing movie number 623 with movie number 28 doesn't make sense. However, it does when comparing timestamps and ratings. We could then make the IDs as factors because this type is super useful in summary statistics.

The timestamp variable also must be converted to a more understandable number to use it in our analysis. The provided code can be converted to a date-time format then we can extract any time period. Let's go with the year for now.

Then, we can extract two useful variables which are the rating year and the release year. We can use those two variables for example to calculate the age of a certain movie when the rating was done by a user.

```
processDataset <- function(df){
  df$userId <- as.factor(df$userId)
  df$movieId <- as.factor(df$movieId)
  df$genres <- as.factor(df$genres)
  df$timestamp <- as.POSIXct(df$timestamp, origin = "1970-01-01")

  df <- df %>%
    mutate(title = str_trim(title), year_rate = year(timestamp)) %>% # remove whitespaces from title an
    extract(title, c("title_tmp", "year"), # separate the title and the relase year
            regex = "^(.*) \\(([0-9 \\-]*)\\)$",
            remove = F) %>%
    # Deal with NAs and errors
    mutate(year = if_else(str_length(year) > 4,
                          as.integer(str_split(year, "-",
                                               simplify = T)[1]),
                          as.integer(year))) %>%
    mutate(title = if_else(is.na(title_tmp), title, title_tmp), age = year_rate - year) %>%
    select(-title_tmp)  %>%
    mutate(genres = if_else(genres == "(no genres listed)",
                            `is.na<-`(genres), genres)) %>%
    select(-title, -timestamp)

  # check for outliers:
  numvectors <- data.frame(df$rating,df$year, df$year_rate)
  summary(numvectors)
  return(df)
}

# Applying the processing to the training set and the validation set
```

```r
edx <- processDataset(edx)
validation <- processDataset(validation)
```

Now our dataset looks like this:

```r
head(edx)
```

```
##   userId movieId rating year                        genres year_rate age
## 1      1     122      5 1992                Comedy|Romance      1996   4
## 2      1     185      5 1995         Action|Crime|Thriller      1996   1
## 3      1     292      5 1995  Action|Drama|Sci-Fi|Thriller      1996   1
## 4      1     316      5 1994       Action|Adventure|Sci-Fi      1996   2
## 5      1     329      5 1994 Action|Adventure|Drama|Sci-Fi      1996   2
## 6      1     355      5 1994       Children|Comedy|Fantasy      1996   2
```

We could check if the integer values year, year_rate and rating have no outliers by running a summary and we find that there are none (No rating outside the 0-5 range, no year outside 1915-2009!

```r
numvectors <- data.frame(edx$rating,edx$year, edx$year_rate)
summary(numvectors)
```

```
##    edx.rating       edx.year    edx.year_rate
##  Min.   :0.500   Min.   :1915   Min.   :1995
##  1st Qu.:3.000   1st Qu.:1987   1st Qu.:2000
##  Median :4.000   Median :1994   Median :2002
##  Mean   :3.512   Mean   :1990   Mean   :2002
##  3rd Qu.:4.000   3rd Qu.:1998   3rd Qu.:2005
##  Max.   :5.000   Max.   :2008   Max.   :2009
```

3- Exploring the data

3-1 Summary statistics on the dataset

First, let's see how many observations, unique users, unique movies and genres we are dealing with:

```r
# number of observations and variables:
dim(edx)
```

```
## [1] 9000055       7
```

```r
#number of uniques users
n_distinct(edx$userId)
```

```
## [1] 69878
```

```r
# number of uniques movies
n_distinct(edx$movieId)
```

```
## [1] 10677
```

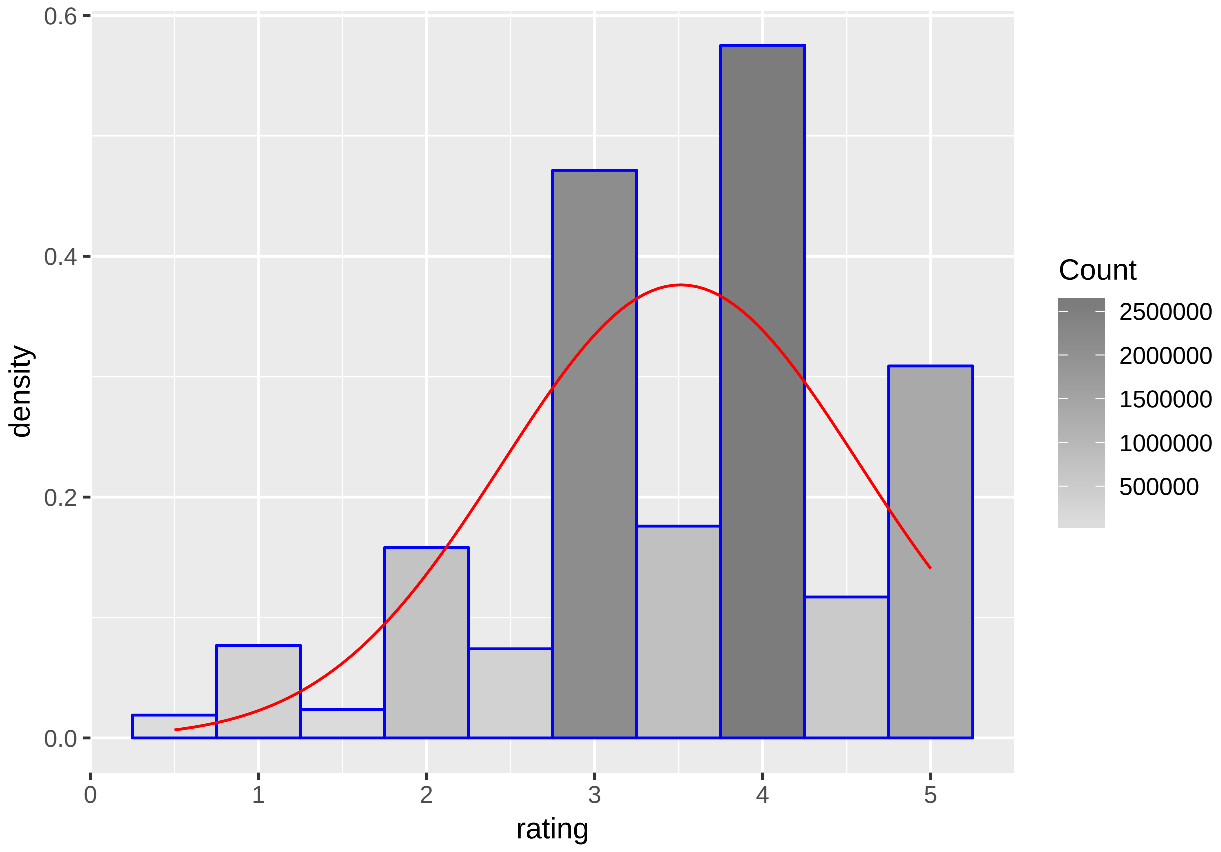
```r
# number of distinct genres| genres can be a combination of many so we split them to count:
n_distinct(edx %>% separate_rows(genres, sep= "\\|") %>% select(genres))
```

```
## [1] 20
```

3-2 Discovering the data distribution

Since our objective is to predict the ratings, let's first draw a histogram of the ratings:

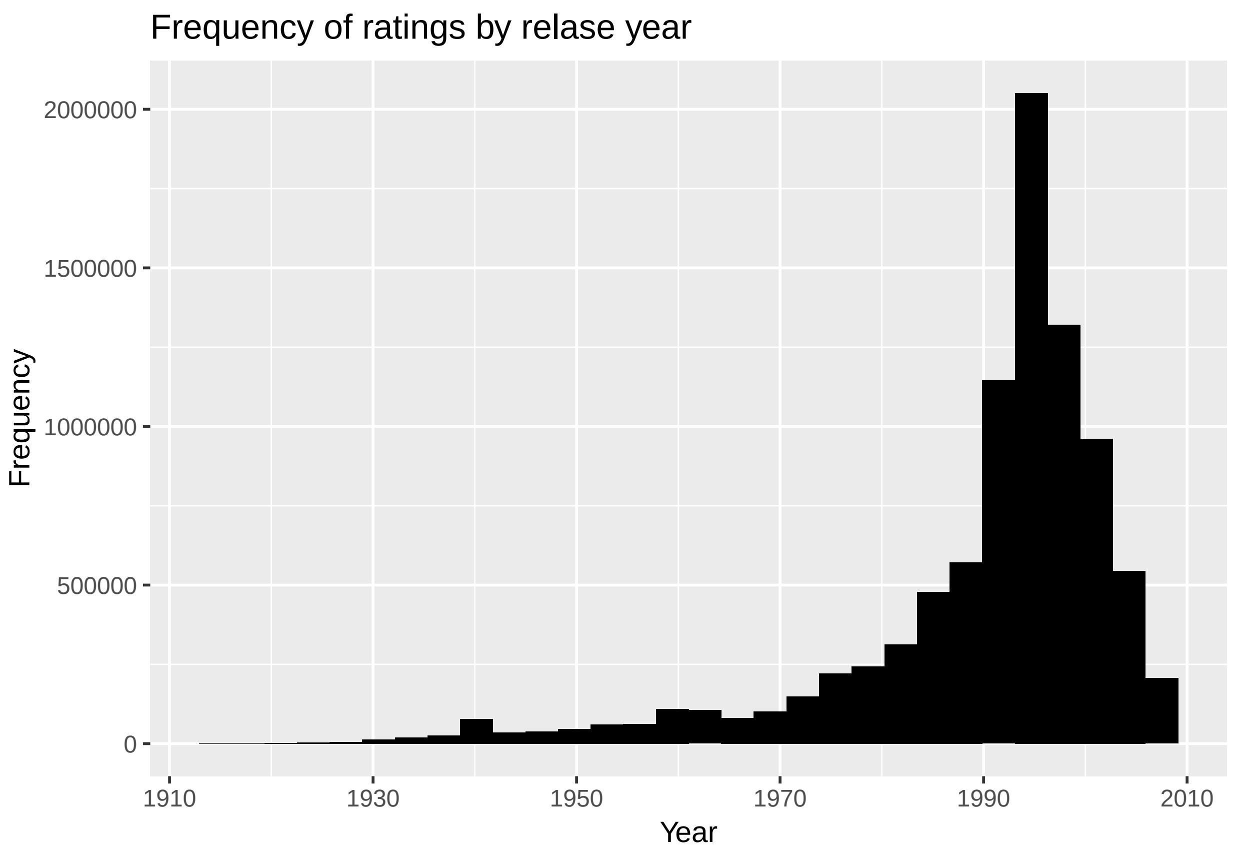
```r
mean = mean(edx$rating)
stdv = sd(edx$rating)
edx %>% ggplot(aes(x=rating)) + geom_histogram(binwidth=0.5, colour="blue",
                                               aes(y=..density.., fill=..count..)) +
                      scale_fill_gradient("Count", low="#DCDCDC", high="#7C7C7C") +
                      stat_function(fun=dnorm, color="red",args=list(mean, stdv))
```



The distribution is not really normal shaped and we see a negative skewness. There is also no rating with the value 0 and most of the ratings comes exactly at 3 and 4. Why there are more good ratings than bad ones? This could be explained by the fact that when a user watch a movie and like it, he finds the time to rate this movie. But when a user does not like a movie, he simply doesn't bother and pass.

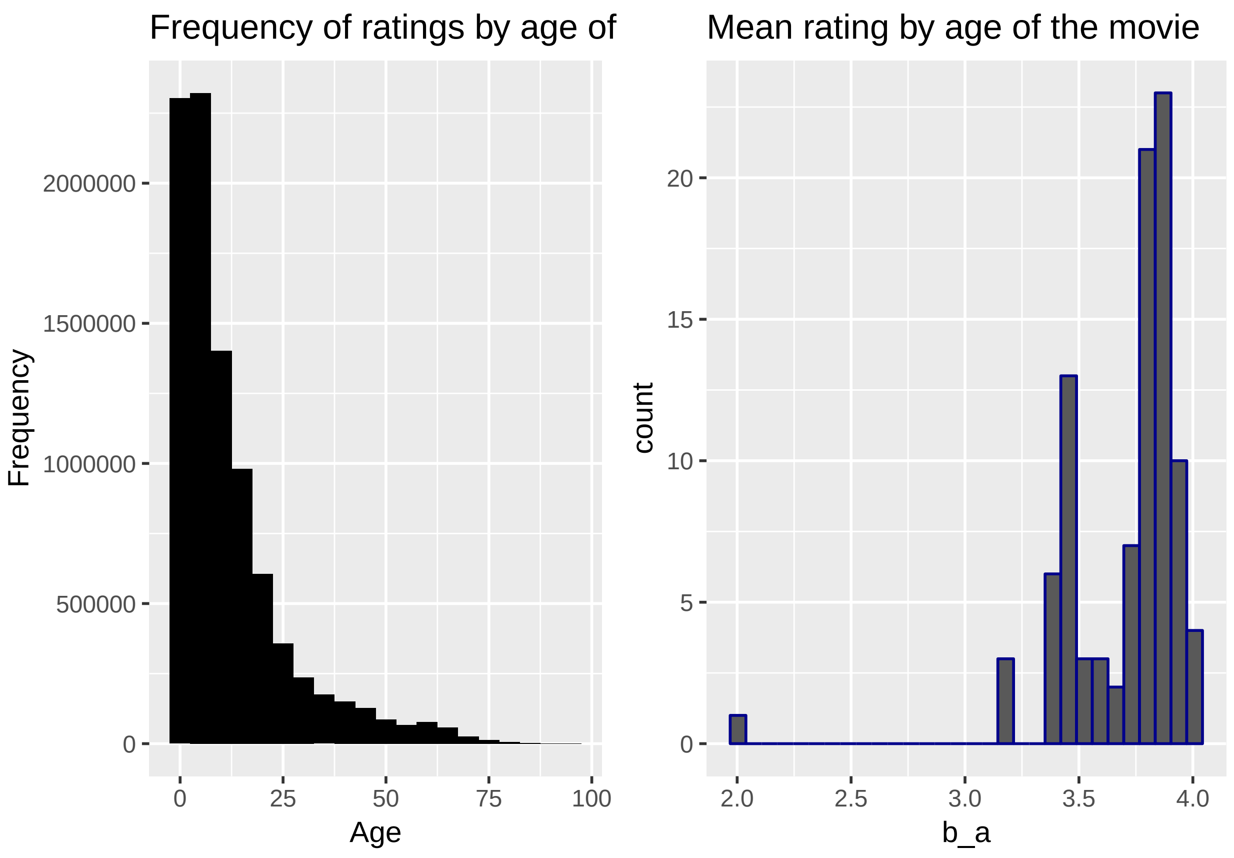
Now let's see how the number of ratings evolve with the release year of the movies:

```r
edx %>% ggplot(aes(year)) +
  geom_histogram(fill = "black") +
  labs(title = "Frequency of ratings by relase year",
       x = "Year",
       y = "Frequency")
```



Again we have a left skewed distribution with most of the ratings happening between 1980 and 2009. Now let's introduce the age variable and see its distribution:

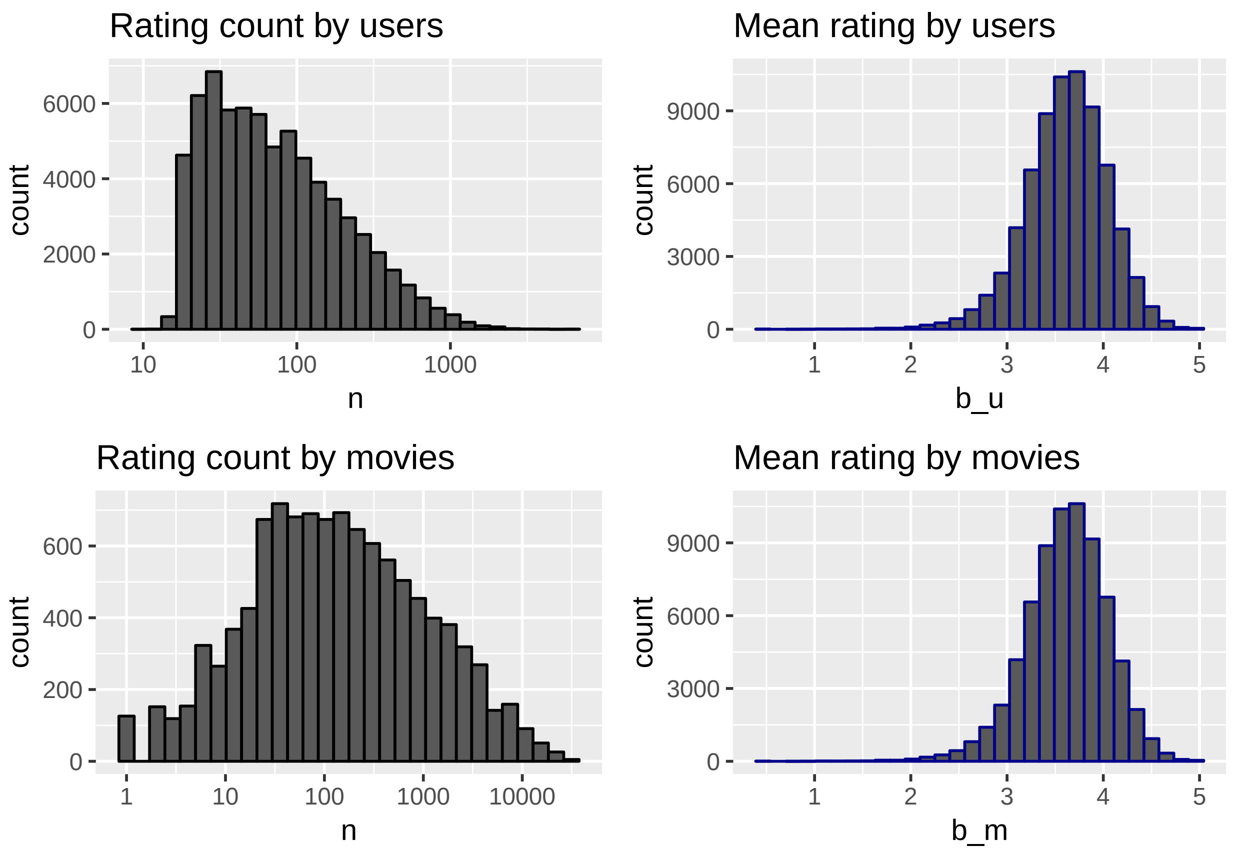
```
edx <- edx %>% mutate(age = year_rate - year)

plot1 <- edx %>% ggplot(aes(age)) +
  geom_histogram(binwidth = 5, fill = "black") +
  labs(title = "Frequency of ratings by age of the movie",
       x = "Age",
       y = "Frequency")

plot2 <- edx %>%
  group_by(age) %>%
  summarize(b_a = mean(rating)) %>%
  ggplot(aes(b_a)) +
  geom_histogram(color = "darkblue") +
  ggtitle("Mean rating by age of the movie")

grid.arrange(plot1, plot2, ncol=2)
```



The histogram explains that recent movies are rated more enough than older ones. This can be explained by the fact that people tend to watch popular movies and searching for new movies each time. People are usually not very interested in old movies because either they have watched many mavies before they started ratings movies or no one in their social network recommands an old movie.

Now let's see if the users and the movies have some effect on the ratings:

```r
plot1 <- edx %>%
dplyr::count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Rating count by users")
plot2 <- edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "darkblue") +
  ggtitle("Mean rating by users")
plot3 <- edx %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Rating count by movies")
plot4 <- edx %>%
  group_by(userId) %>%
  summarize(b_m = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_m)) +
  geom_histogram(bins = 30, color = "darkblue") +
  ggtitle("Mean rating by movies")
grid.arrange(plot1, plot2, plot3, plot4, ncol=2, nrow=2)
```



Let's discuss the first row of the facet plot: we see that some users are more active than others, some users are very cranky and others love every movie. The same analysis applies for movies: some movies are rated and appreciated more than others. This makes the variables users, movies and age have a strong impact on the ratings.

Finally, let's print the genres and their corresponding average rating by descending order to introduce a technique that will be relevant in our modeling:

```
edx %>%
  select(genres, rating) %>%
  group_by(genres) %>%
  summarize(mean = mean(rating), median = median(rating), n = n()) %>%
  arrange(desc(mean)) %>%
  head(10)
```

```
## # A tibble: 10 x 4
##    genres                                      mean median     n
##    <fct>                                      <dbl>  <dbl> <int>
##  1 Animation|IMAX|Sci-Fi                       4.71    5       7
##  2 Drama|Film-Noir|Romance                     4.30    4.5  2989
##  3 Action|Crime|Drama|IMAX                     4.30    4.5  2353
##  4 Animation|Children|Comedy|Crime          4.28    4.5  7167
##  5 Film-Noir|Mystery                        4.24    4    5988
##  6 Crime|Film-Noir|Mystery                  4.22    4    4029
##  7 Film-Noir|Romance|Thriller               4.22    4    2453
##  8 Crime|Film-Noir|Thriller                 4.21    4    4844
##  9 Crime|Mystery|Thriller                   4.20    4   26892
## 10 Action|Adventure|Comedy|Fantasy|Romance  4.20    4   14809
```

In this table, the genre "Animation|IMAX|Sci-Fi" has the highest mean and median rating but this is not significant because it only has 7 ratings. To deal with this problem, we introduce regularization which permits us to penalize large estimates that are formed using small sample sizes.

Constructing the models

1- Evaluation metric: loss function (RMSE)

In the NETFLIX challenge, they used the residual mean squared error to evaluate the participant's models on a test set. Let's define $Y_{u,i}$ as the rating of movie i by user u and $\hat{Y}_{u,i}$ as the estimate of the same variable. The RMSE is formulated as follow:

$$RMSE = \frac{1}{N}\sum_{u,i}\left(\hat{Y}_{u,i} - Y_{u,i}\right)^2$$

with N being the number of user/movie combinations and the sum occurring over all these combinations.

The RMSE is similar to a standard deviation. If a prediction model has an RMSE larger than 1, this simply means that our errors are typically far from the true rating by more than 1 star and this is not good since 1 constitute a big interval considering ratings from 0 to 5.

In the next sections, we will try different models and report their RMSE in the variable rmse_results.

2- Basic model

The simplest model one can think of is predicting one single value to every input. It can be any statistic but people tend to choose the median and the mean. Of course we can choose to generate random ratings but in general, random predictions are often evaluated negatively. A model that assumes the same rating for all movies and users with all the differences explained by random variation would look like this:

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

with $\epsilon_{u,i}$ independent errors sampled from the same distribution centered at 0 and μ the "true" rating for all movies. The best estimator in this case would be the mean of all the ratings.

Let's build this model and report its result:

```r
#Predicting the average for all users and movies
mu_hat <- mean(edx$rating)
naive_rmse <- rmse(validation$rating, mu_hat)
naive_rmse
```

```
## [1] 1.061202
```

```r
# Creating a dataframe containing the rmse results of all methods
rmse_results <- data_frame(method = "Just the average", RMSE = naive_rmse)
```

With just the average, the model is very bad because we are making a typical error of 1.0612018 stars. We will improve this model by adding some of the variables that had a clear impact on the ratings.

3- Including other variables in the model

In this model, we will incorporate the effect of the movie, the user and their combination to see whether our score is improving.

The movie effect model

This model will be formulated as:

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

where b_i is the average of the difference between the movie rating and the global mean rating.

```r
#first, estimate the effect of each movie bi
validation$movieId <- as.factor(validation$movieId)
mu <- mean(edx$rating)
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
# then, make the prediction using the new model y= mu + b
predicted_ratings <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i
model_1_rmse <- rmse(predicted_ratings, validation$rating)
model_1_rmse
```

```
## [1] 0.9439087
```

```r
# Add the rmse result of the new model to the rmse data frame
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = model_1_rmse ))
```

The user effect model

This model will be formulated as:

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

where b_u is the average of the difference between the user rating and global mean rating μ

```r
#first, estimate the effect of each movie bi
mu <- mean(edx$rating)
users_avgs <- edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu))
# then, make the prediction using the new model y= mu + b
predicted_ratings <- mu + validation %>%
  left_join(users_avgs, by='userId') %>%
  .$b_u
model_2_rmse <- rmse(predicted_ratings, validation$rating)
model_2_rmse
```

```
## [1] 0.978336
```

```r
# Add the rmse result of the new model to the rmse data frame
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="User Effect Model",
                                     RMSE = model_2_rmse ))
```

The movie and user effect model

This model will be formulated as:

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

where b_a is the average of the difference between the user rating and the sum of the global mean rating μ and b_i

```
mvuser_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_a = mean(rating - mu - b_i))
# Now, compute the predicted ratings using the new model with user and movie effects:
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(mvuser_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_a) %>%
  .$pred
model_3_rmse <- RMSE(predicted_ratings, validation$rating)
model_3_rmse
```

```
## [1] 0.8653488
```

```
# Add the rmse result of the new model to the rmse data frame
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effects Model",
                                     RMSE = model_3_rmse ))
```

The movie, user and age effect model

This model will be formulated as:

$$Y_{u,i} = \mu + b_i + b_a + b_g + \epsilon_{u,i}$$

where b_g is the average of the difference between the age rating and the sum of the global mean rating μ, b_i and b_a

```
# Let's compute the new term bu in our mode y= mu+bi+ba+bg +eps
age_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(mvuser_avgs, by='userId') %>%
  group_by(age) %>%
  summarize(b_g = mean(rating - mu - b_i - b_a))

# process the variable age so that leftjoin finds values in LHS
listofages <- unique(age_avgs$age)
ages <- validation$age
validation$age <- sapply(ages, function(x){ listofages[which.min(abs(listofages-x))]})
# Now, compute the predicted ratings using the new model with user and movie effects:
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(mvuser_avgs, by='userId') %>%
  left_join(age_avgs, by='age') %>%
  mutate(pred = mu + b_i + b_a + b_g) %>%
  .$pred
model_4_rmse <- RMSE(predicted_ratings, validation$rating)
model_4_rmse
```

```
## [1] 0.8649038
```

```
# Add the rmse result of the new model to the rmse data frame
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User + Age Effects Model",
                                     RMSE = model_4_rmse ))
```

Using regularization to improve the score

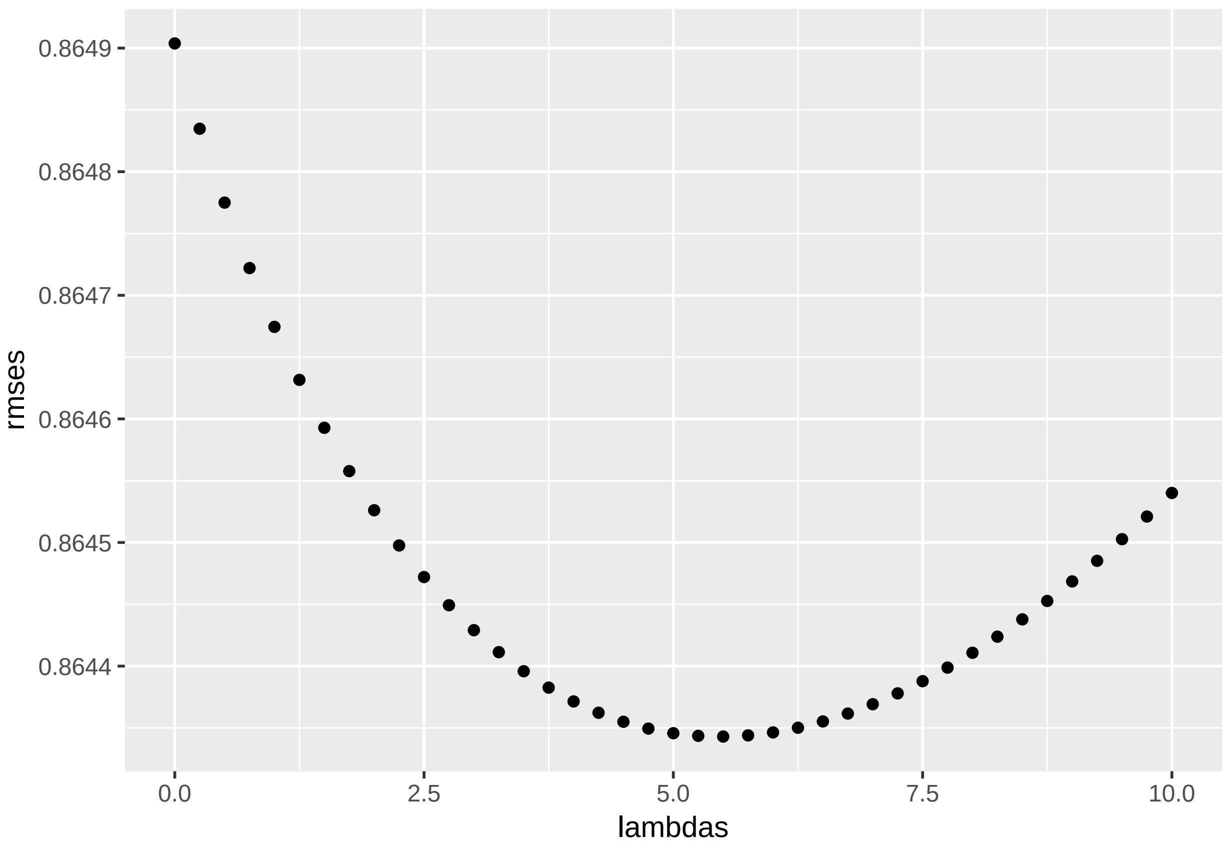
The rmse results up to now are still not very good. Maybe we are making some errors in our predictions. To make sure, let's take the first model (movie effect) and print the movies with the best ratings:

```
save_edx$movieId <- as.factor(save_edx$movieId)

movie_titles <- save_edx %>%
  select(movieId, title) %>%
  distinct()

save_edx %>% dplyr::count(movieId) %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:10)
```

```
## # A tibble: 10 x 3
##    title                                                          b_i     n
##    <chr>                                                        <dbl> <int>
##  1 Hellhounds on My Trail (1999)                                 1.49     1
##  2 "Satan's Tango (S\u00e1t\u00e1ntang\u00f3) (1994)"            1.49     2
##  3 Shadows of Forgotten Ancestors (1964)                         1.49     1
##  4 Fighting Elegy (Kenka erejii) (1966)                          1.49     1
##  5 Sun Alley (Sonnenallee) (1999)                                1.49     1
##  6 Blue Light, The (Das Blaue Licht) (1932)                      1.49     1
##  7 Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko~  1.24     4
##  8 Human Condition II, The (Ningen no joken II) (1959)           1.24     4
##  9 Human Condition III, The (Ningen no joken III) (1961)         1.24     4
## 10 Constantine's Sword (2007)                                    1.24     2
```

We see that those movies were only rated by a very few users so our mispredictions comes from this small samples. We can deal with those errors by introducing regularization. The general idea behind it is to constrain the total variability of the effect sizes by adding a penalty term to the least square equation, that gets larger when b_i are large. The penalty term is a parameter that should be tuned to optimise the objective.

```
# Using regularization and tuning the penalty term

lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){
  mu <- mean(edx$rating)

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

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

  b_g <- edx %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_a, by="userId") %>%
    group_by(age)%>%
    summarize(b_g = sum(rating - b_i - b_a - mu)/(n()+l))


  predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_a, by = "userId") %>%
    left_join(b_g, by = "age") %>%
    mutate(pred = mu + b_i + b_a + b_g) %>%
    .$pred

  return(rmse(predicted_ratings, validation$rating))
})

# Plot the values or RMSE against lambdas
qplot(lambdas, rmses)
```



```r
# Find the value of lambda that minimizes the RMSE
lambdas[which.min(rmses)]
```

```
## [1] 5.5
```

```r
min(rmses)
```

```
## [1] 0.864343
```

```r
# save the model with the minimal RMSE to the results data frame

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie + User + age Effect Model",
                                     RMSE = min(rmses)))
```

Result discussion

In this section, we show our results for the five models we implemented:

```r
rmse_results %>% knitr::kable(caption = "Summary of the RMSEs")
```

Table 1: Summary of the RMSEs

method	RMSE
Just the average	1.0612018
Movie Effect Model	0.9439087
User Effect Model	0.9783360

method	RMSE
Movie + User Effects Model	0.8653488
Movie + User + Age Effects Model	0.8649038
Regularized Movie + User + age Effect Model	0.8643430

As expected, all the models are better than the one value prediction based on the average. The movieId variable had a strong impact on the rmse and combining it with the usedId made the rmse smaller. The age variable a very small impact comparing to the first two. The best rmse 0.864343 is then obtained by the last model

Conclusion

In this project, we had the chance to apply many of the techniques learned in data exploration, visualization and machine learning courses. We tried to get to an RMSE result below 0.87 and we achieved this result. We can move forward and try new algorithms like matrix factorization, gradient boosting machines, random forest on a larger dataset. However, such work will certainly need more computation power. We can try those techniques in a future work.

I would like to thank Professor Rafael A. Irizarry for this wonderful and well detailed course on the edX plateforme. Special thanks for the students who made all the problems clear and made the forums a complementary learning space.