MovieLens Project

Rodrigo Moraes Kunrath

5/25/2021

Introduction

In order to create a movie recommendation system a machine learning algorithim was built in R. This project is part of the Data Science Professional Certificate from HarvardX.

The objective of the machine learning algorithim is to get the best possible rating guess. The general idea is to aim for the lowest possible root-mean-square error (RSME) without overtraining the model.

As the solution to the proposed problem was iterativelly determined during the coding development, this report must also to be followed through the code and its comments.

The dataset

The dataset used is MovieLens 10MB and it can be found in https://grouplens.org/datasets/movielens/10m/. This dataset was downloaded and subsequently divided into a validation and a training dataset. These were called *validation* and *edx* in the given order.

The dataset is comprised of 10000054 ratings and carries information as shown in the code bellow.

```
nrow(edx) + nrow(validation)
```

```
## [1] 10000054
```

```
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
```

Steps performed

In order to approach the problem, the first step performed was exploring the dataset and trying to see if any insight can be obtained from this exploration. Secondly, the data was wrangled for better prediction.

Subsequently, the *edx* dataset was divided in a training and a testing datasets and an evaluation RSME function was created.

Once all the steps above were performed, a modelling phase began. Different approaches were developed and a final cost-benefit solution was chosen.

Finally the model was applied to the *validation* dataset.

The whole operation is discribed in the following section and is structured in the comments.

Methods

Data Exploration

With a brief look at edx we can see that the column title also contains the year and genres are somehow aggregated.

```
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
```

```
nrow(edx)
```

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

Also, we can see that the 9000055 entries are comprized of 69878 users and 10677 movies, with a mean rating of 3.5 and a standard deviation of 1.

```
edx %>% summarize(n_row = nrow(.),
                  n_users = n_distinct(userId),
                  n_movies = n_distinct(movieId))
```

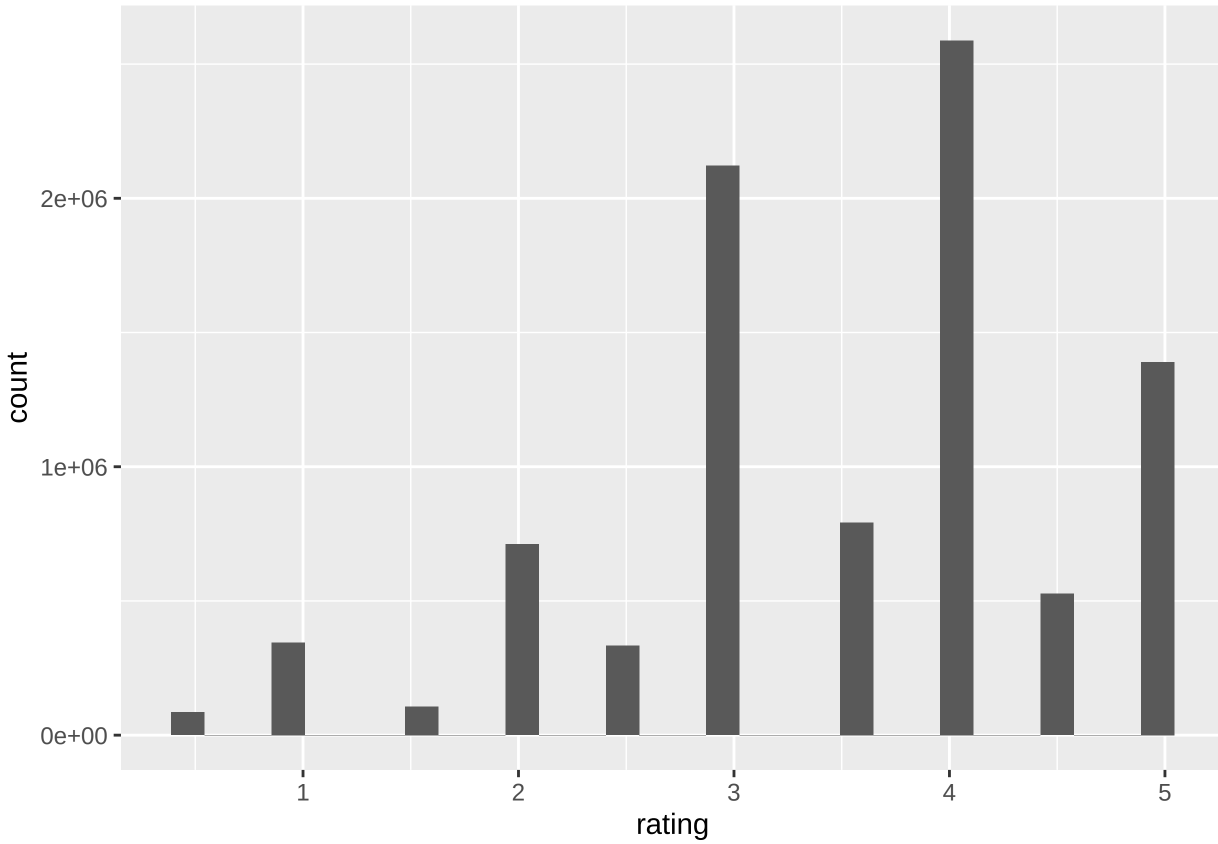
```
##     n_row n_users n_movies
## 1 9000055   69878    10677
```

```
edx %>% summarise(mean = mean(rating), sd = sd(rating))
```

```
##       mean       sd
## 1 3.512465 1.060331
```

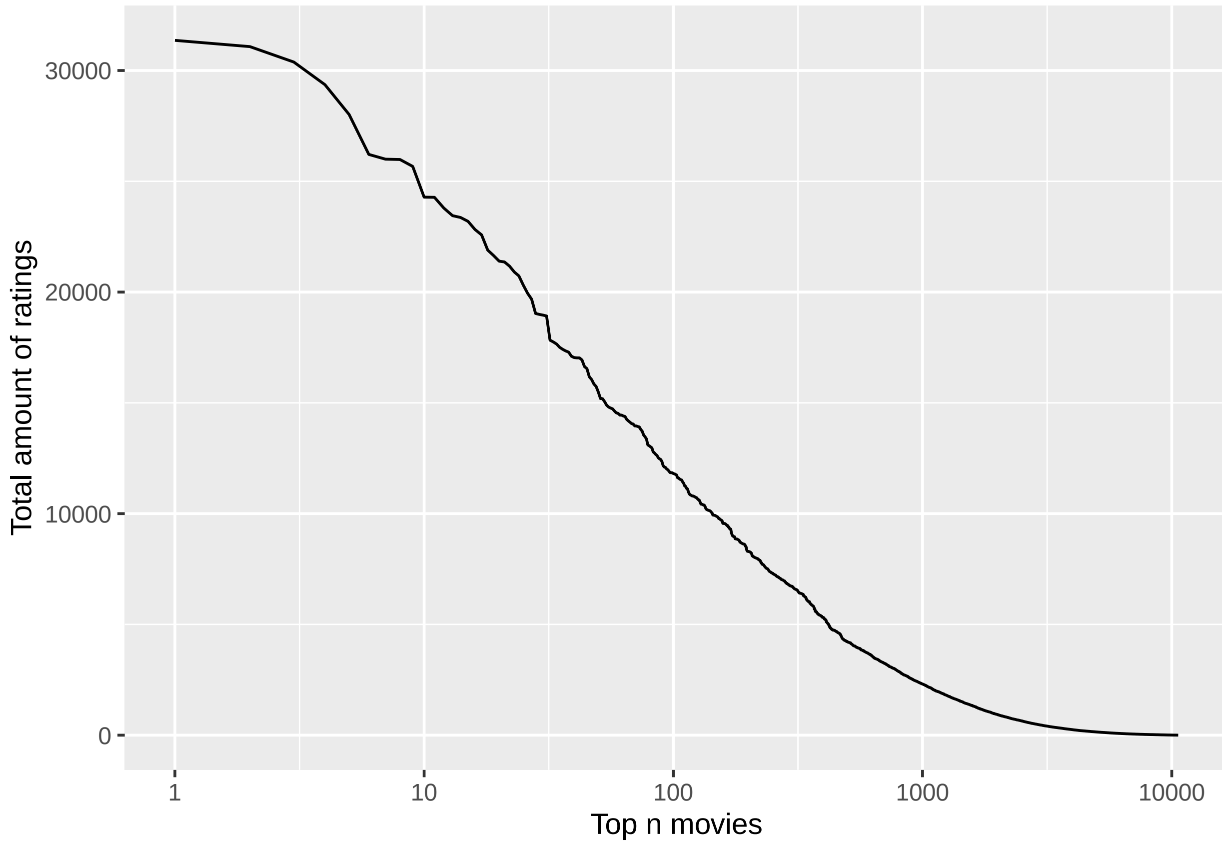
Users do not tend to give half star ratings.

```
edx %>% ggplot(aes(rating)) + geom_histogram()
```



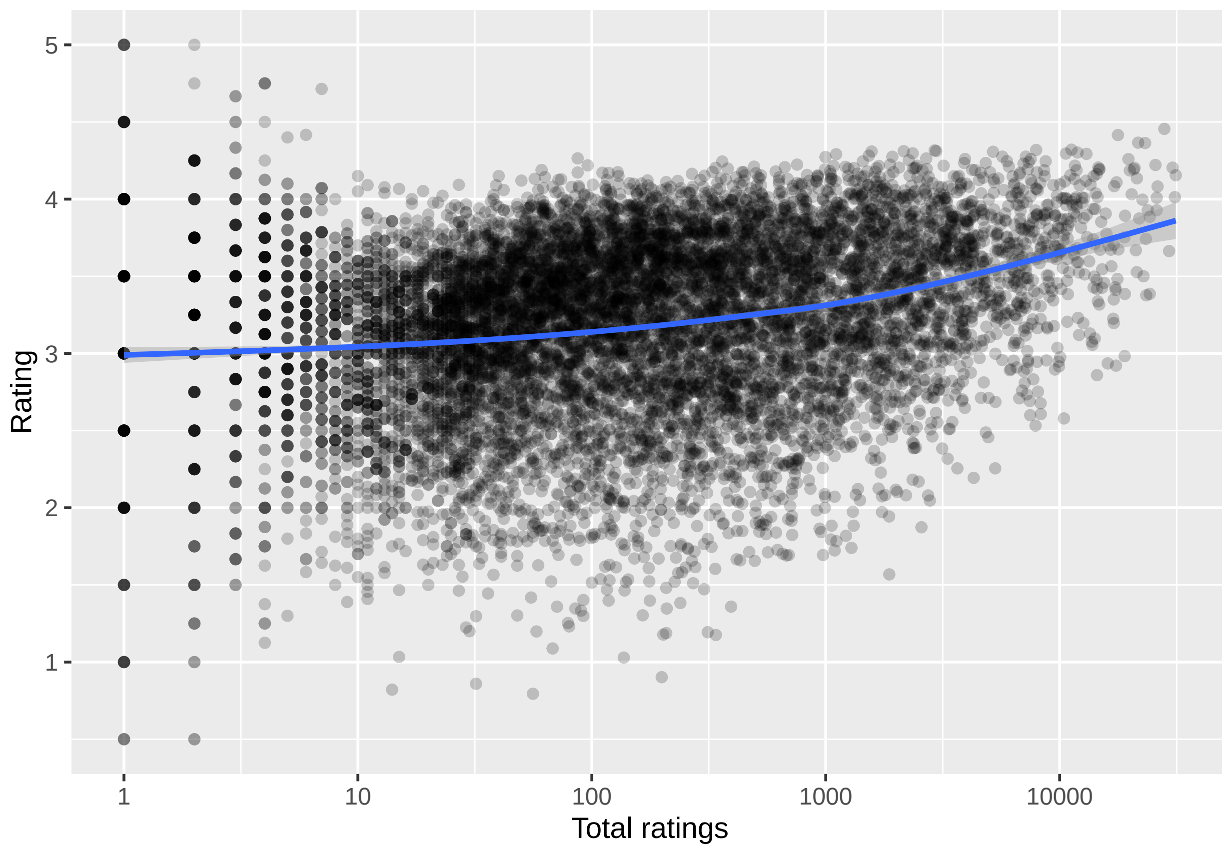
The movies have a rating distribution that is far from homogeneous. Most part of the ratings are comprized in a few movies.

```r
# Creating an object that contains total ratings and the mean rating per movie
dat_permovie <- edx %>%
  group_by(title) %>%
  summarise(totalratings = sum(userId != 0), mean_rate = mean(rating)) %>%
  arrange(desc(totalratings))

# Plotting the total amount of ratings per rank in logarithmic scale
dat_permovie %>%
  ggplot(aes(10677-rank(totalratings), totalratings)) +
  geom_line() +
  scale_x_log10() +
  xlab("Top n movies") +
  ylab("Total amount of ratings")
```



The most viewed movies are, unsurprisingly, the ones with better mean ratings.

```
# Plotting the ratings per rank in logarithmic scale.
dat_permovie %>%
  ggplot(aes(totalratings, mean_rate)) +
  geom_point(alpha = 0.2) +
  scale_x_log10() +
  geom_smooth() +
  xlab("Total ratings") +
  ylab("Rating")
```



We can see that the top 100 movies have a higher mean rating than top 1000 and the rest of the movies.

```
# Movies mean and sd ratings
dat_permovie %>%
        summarize(mean = mean(mean_rate), sd = sd(mean_rate))
```

```
## # A tibble: 1 x 2
##    mean    sd
##   <dbl> <dbl>
## 1  3.19 0.571
```

```
# Top 100 movies mean and sd ratings
dat_permovie %>%
        top_n(100, wt = totalratings) %>%
        summarize(mean = mean(mean_rate), sd = sd(mean_rate))
```

```
## # A tibble: 1 x 2
##    mean    sd
##   <dbl> <dbl>
## 1  3.75 0.387
```

```
# Top 1000 movies mean and sd ratings
dat_permovie %>%
        top_n(1000, wt = totalratings) %>%
        summarize(mean = mean(mean_rate), sd = sd(mean_rate))
```

```
## # A tibble: 1 x 2
##    mean    sd
##   <dbl> <dbl>
```

```
## 1  3.54 0.434
```

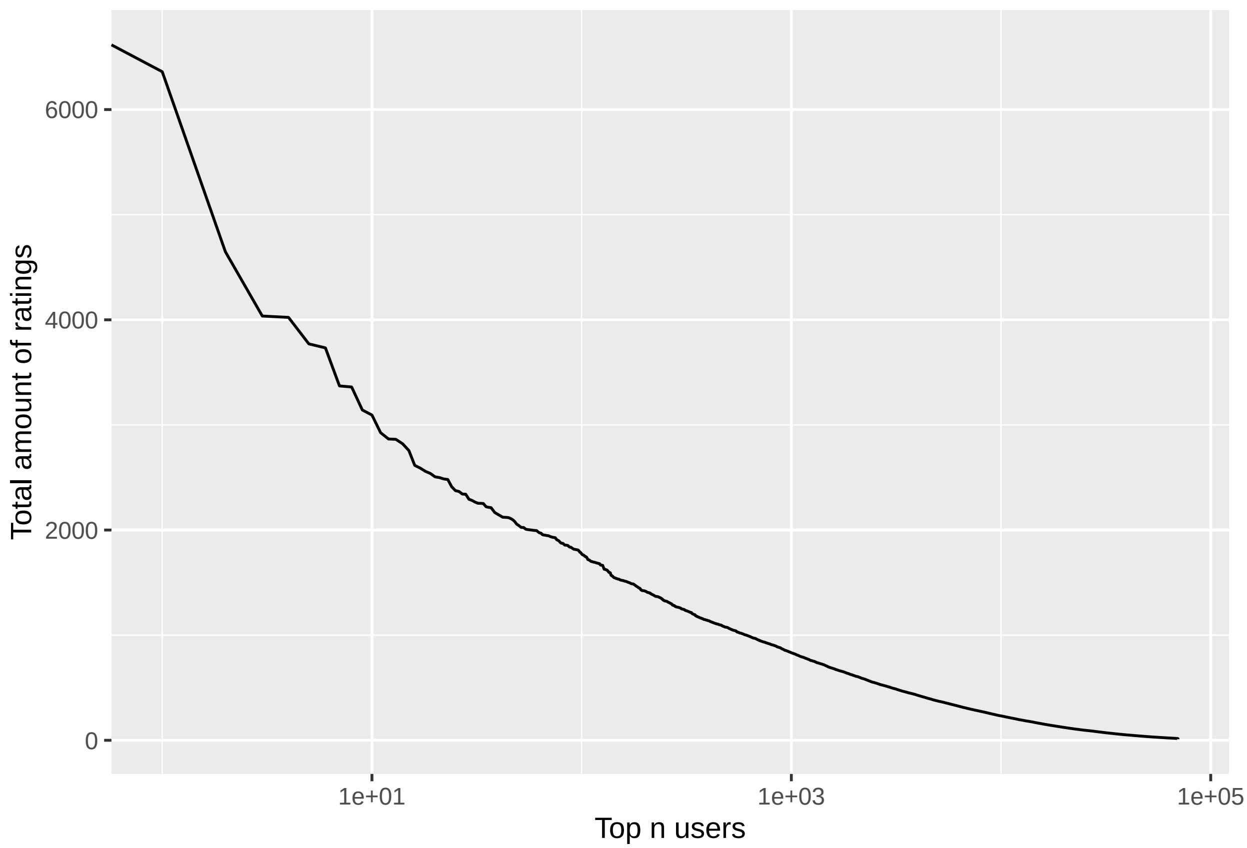
```
# Rest of users mean and sd ratings
dat_permovie %>%
        top_n(-(10676-1000), wt = totalratings) %>%
        summarize(mean = mean(mean_rate), sd = sd(mean_rate))
```

```
## # A tibble: 1 x 2
##    mean    sd
##   <dbl> <dbl>
## 1  3.16 0.572
```

The users also have a rating distribution that is far from uniform.

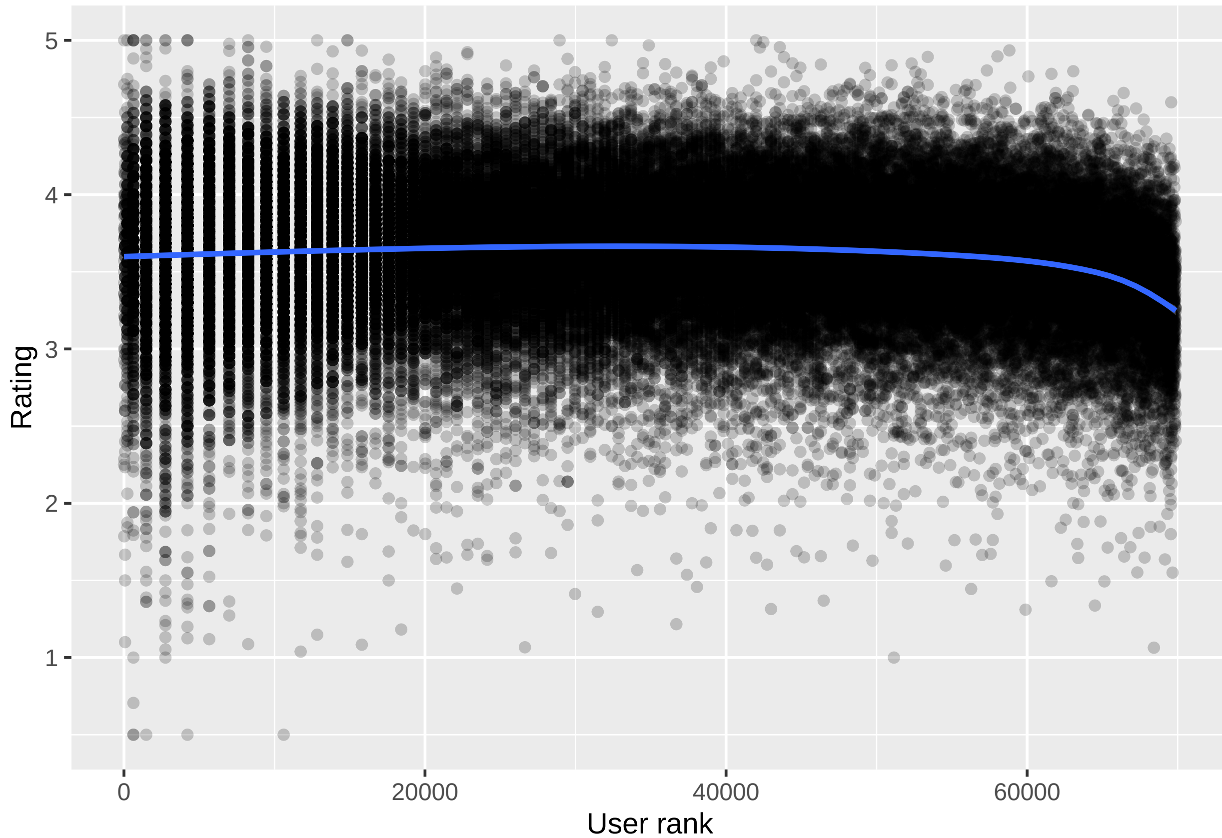
Most part of the ratings are comprized in few users.

```
# Creating an object that contains total ratings and the mean rating per user.
dat_peruser <- edx %>%
  group_by(userId) %>%
  summarise(totalratings = sum(userId != 0), mean_rate = mean(rating)) %>%
  arrange(desc(totalratings))

# Plotting the total amount of ratings per rank in logarithmic scale
dat_peruser %>%
  ggplot(aes(69878-rank(totalratings), totalratings)) + # 69878 is the total user count
  geom_line() +
  scale_x_log10() +
  xlab("Top n users") +
  ylab("Total amount of ratings")
```



Users that assess more movies also tend to be more critical with their reviews.

```
# Plotting the ratings per rank. Polynom of order 5
dat_peruser %>%
  ggplot(aes(rank(totalratings), mean_rate)) + # 69878 is the total movie count
  geom_point(alpha = 0.2) +
  geom_smooth() +
  xlab("User rank") +
  ylab("Rating")
```




```
# Users mean and sd ratings
dat_peruser %>%
        summarize(mean = mean(mean_rate), sd = sd(mean_rate))
```

```
## # A tibble: 1 x 2
##    mean    sd
##   <dbl> <dbl>
## 1  3.61 0.431
```

```
# Top 1000 users mean and sd ratings
dat_peruser %>%
        top_n(1000, wt = totalratings) %>%
        summarize(mean = mean(mean_rate), sd = sd(mean_rate))
```

```
## # A tibble: 1 x 2
##    mean    sd
##   <dbl> <dbl>
## 1  3.26 0.412
```

```
# Rest of users mean and sd ratings
dat_peruser %>%
        top_n(-(69878-1000), wt = totalratings) %>%
        summarize(mean = mean(mean_rate), sd = sd(mean_rate))
```

```
## # A tibble: 1 x 2
##    mean    sd
##   <dbl> <dbl>
## 1  3.62 0.429
```

Exploring whether the genre can affect mean rating.

```r
# Creating and object with movie ratings per genre.
dat_pergenres <- edx %>% group_by(genres) %>%
  summarize(mean = mean(rating), sd = sd(rating), total_ratings = sum(userId != 0)) %>%
  arrange(desc(mean))

head(dat_pergenres)
```

```
## # A tibble: 6 x 4
##   genres                          mean    sd total_ratings
##   <chr>                          <dbl> <dbl>         <int>
## 1 Animation|IMAX|Sci-Fi           4.71 0.567             7
## 2 Drama|Film-Noir|Romance         4.30 0.791          2989
## 3 Action|Crime|Drama|IMAX         4.30 0.739          2353
## 4 Animation|Children|Comedy|Crime 4.28 0.815          7167
## 5 Film-Noir|Mystery               4.24 0.788          5988
## 6 Crime|Film-Noir|Mystery         4.22 0.762          4029
```

Different genres have also different means and standard deviations.

```r
edx %>% summarise(mean = mean(rating), sd = sd(rating))
```

```
##       mean       sd
## 1 3.512465 1.060331
```

```r
edx %>% filter(str_detect(genres, "Drama")) %>%
        summarise(mean = mean(rating), sd = sd(rating))
```

```
##       mean       sd
## 1 3.673131 0.995397
```

```r
edx %>% filter(str_detect(genres, "Film-Noir")) %>%
        summarise(mean = mean(rating), sd = sd(rating))
```

```
##       mean        sd
## 1 4.011625 0.8871659
```

```r
edx %>% filter(str_detect(genres, "Mystery")) %>%
        summarise(mean = mean(rating), sd = sd(rating))
```

```
##       mean       sd
## 1 3.677001 1.000263
```

```r
edx %>% filter(str_detect(genres, "Horror")) %>%
        summarise(mean = mean(rating), sd = sd(rating))
```

```
##       mean       sd
## 1 3.269815 1.149955
```

```r
edx %>% filter(str_detect(genres, "Comedy")) %>%
        summarise(mean = mean(rating), sd = sd(rating))
```

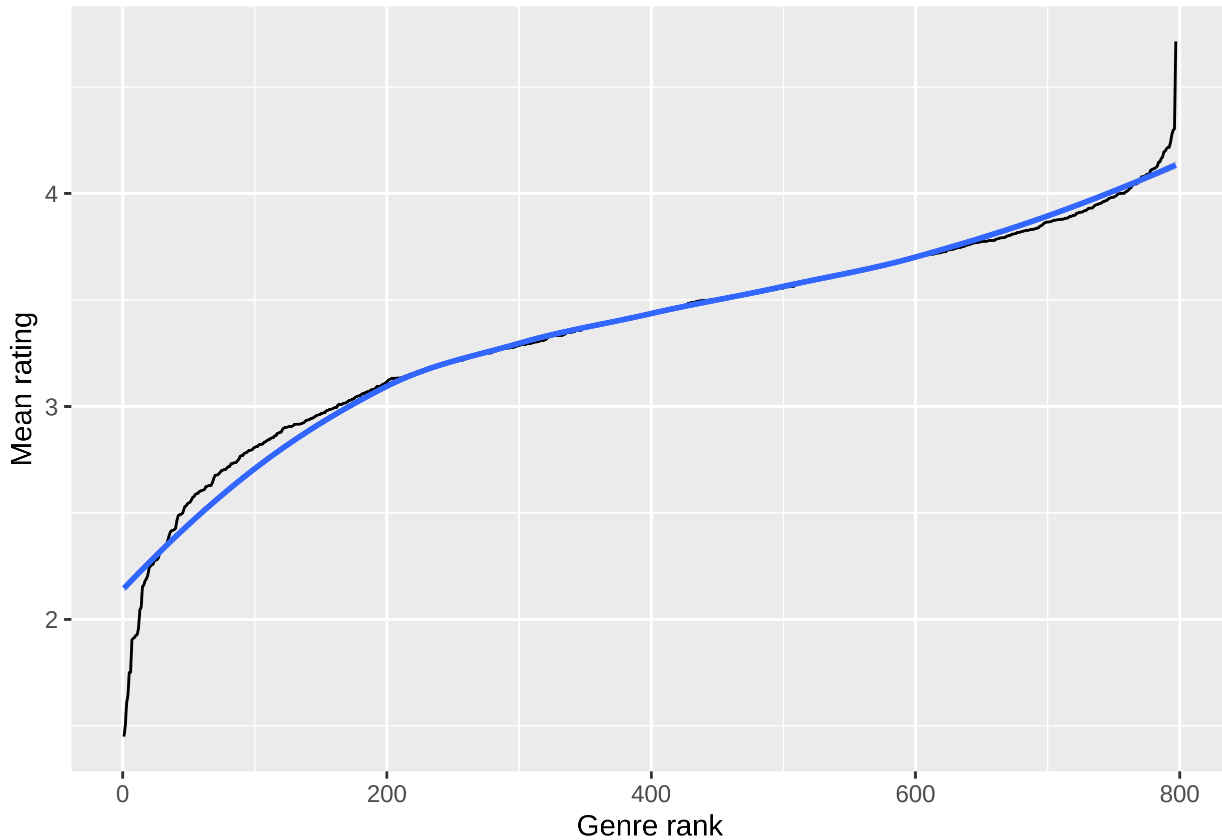
```
##       mean       sd
## 1 3.436908 1.074651
```

```r
edx %>% filter(str_detect(genres, "Children")) %>%
        summarise(mean = mean(rating), sd = sd(rating))
```

```
##       mean       sd
```

```
## 1 3.418715 1.092398
```

```
# Plotting aggregated genres and their mean.
# It looks like a cotangent function. Polynom of order 5
dat_pergenres %>%
  ggplot(aes(rank(mean), mean)) + # 797 is the total genre count
  geom_line() +
  geom_smooth() +
  xlab("Genre rank") +
  ylab("Mean rating")
```



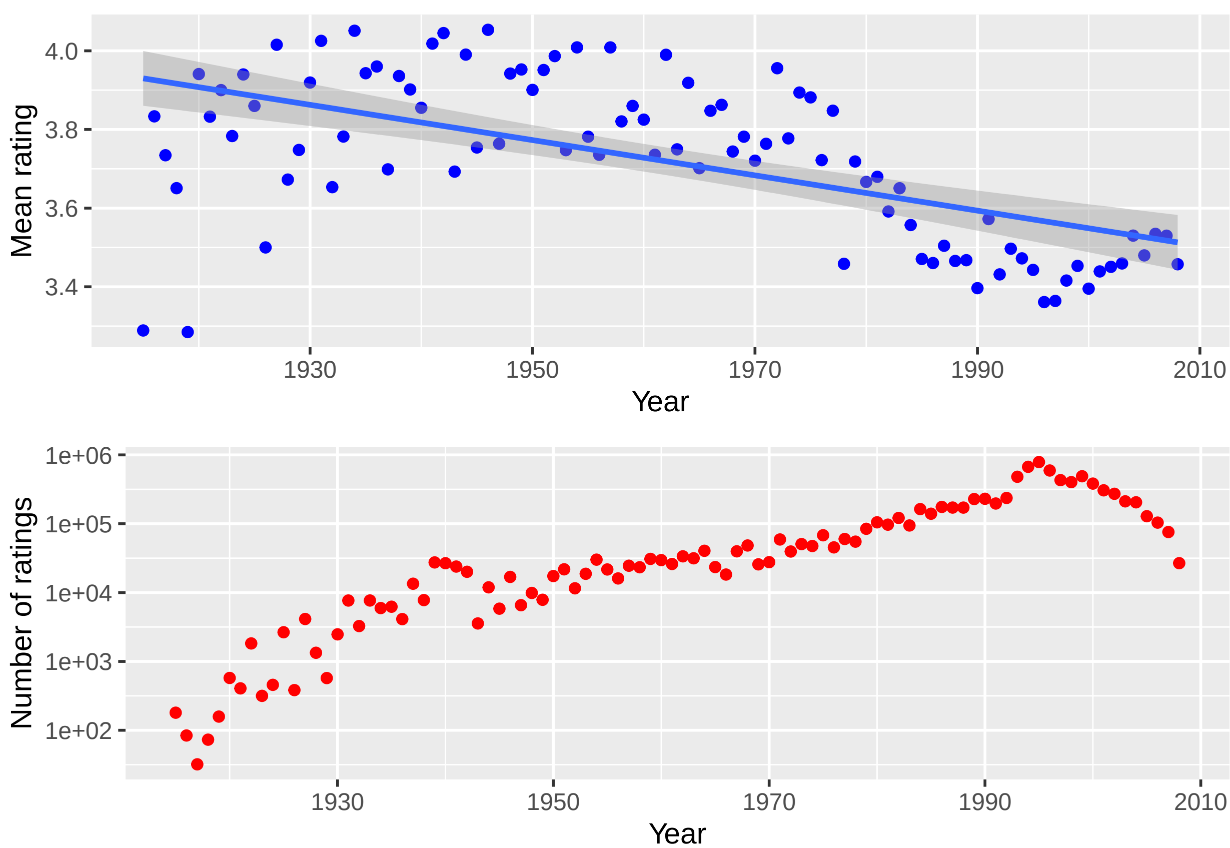
Verifying if the year influences the mean rating. To do that, it is necessary to extract the year from the table.

```
# Getting the year within the parenthesis
year_vector <- str_extract(edx$title, "\\(\\d\\d\\d\\d\\)")
# Removing the parenthesis
year_vector <- substring(year_vector, 2, nchar(year_vector)-1)
# Converting to numeric
year_vector <- as.numeric(year_vector)

# Creating an object that contains the year
dat_peryear <- edx %>% mutate(year = year_vector)
dat_peryear <- dat_peryear %>% group_by(year) %>%
        summarise(totalratings = sum(userId != 0), mean_rate = mean(rating))

head(dat_peryear)
```

```
## # A tibble: 6 x 3
```

```
##    year totalratings mean_rate
##   <dbl>        <int>     <dbl>
## 1  1915          180      3.29
## 2  1916           84      3.83
## 3  1917           32      3.73
## 4  1918           73      3.65
## 5  1919          158      3.28
## 6  1920          575      3.94
```

```r
# Plot object of mean rating per year.
a <- ggplot(data = dat_peryear, aes(x = year, y = mean_rate)) +
  geom_point(alpha = 1, color = "blue") +
  geom_smooth(method='lm', formula = y~poly(x,1)) +
  xlab("Year") +
  ylab("Mean rating")
# Plot object of total ratings per year
b <- ggplot(data = dat_peryear, aes(x = year, y = totalratings)) +
  geom_point(alpha = 1, color = "red") +
  xlab("Year") +
  scale_y_log10() +
  ylab("Number of ratings")

# Arranged plot of mean ratings and year
require(gridExtra)
grid.arrange(a, b)
```



When verifying if timestamp affects rating, it was seen that the rating year affects the mean, but not much.

Nonetheless, the month and weekday seems to not alter it substantially.

```
edx %>%
  mutate(rating_year = year(as_datetime(timestamp))) %>%
  group_by(rating_year) %>%
  summarise(mean = mean(rating))
```

```
## # A tibble: 15 x 2
##    rating_year  mean
##          <dbl> <dbl>
##  1        1995  4
##  2        1996  3.55
##  3        1997  3.59
##  4        1998  3.51
##  5        1999  3.62
##  6        2000  3.58
##  7        2001  3.54
##  8        2002  3.47
##  9        2003  3.47
## 10        2004  3.43
## 11        2005  3.44
## 12        2006  3.47
## 13        2007  3.47
## 14        2008  3.54
## 15        2009  3.46
```

```
edx %>%
  mutate(rating_month = month(as_datetime(timestamp))) %>%
  group_by(rating_month) %>%
  summarise(mean = mean(rating))
```

```
## # A tibble: 12 x 2
##    rating_month  mean
##           <dbl> <dbl>
##  1            1  3.52
##  2            2  3.51
##  3            3  3.48
##  4            4  3.52
##  5            5  3.48
##  6            6  3.50
##  7            7  3.50
##  8            8  3.48
##  9            9  3.50
## 10           10  3.56
## 11           11  3.54
## 12           12  3.53
```

```
edx %>%
  mutate(rating_weekday = wday(as_datetime(timestamp))) %>%
  group_by(rating_weekday) %>%
  summarise(mean = mean(rating))
```

```
## # A tibble: 7 x 2
##   rating_weekday  mean
##            <dbl> <dbl>
## 1              1  3.52
```

```
## 2              2  3.52
## 3              3  3.51
## 4              4  3.50
## 5              5  3.50
## 6              6  3.51
## 7              7  3.53
```

Now the user gender preference is going to be verifyed. It is somewhat logic that users may have specific taste per genre.

For instance, user 11129 has more than 375 ratings and a mean rating of 4.1. Nevertheless, the user doesn't like Horror movies. This should be taken into account.

```
edx %>% filter(userId == 11129) %>% nrow()
```

```
## [1] 375
```

```
edx %>% filter(userId == 11129) %>% summarize(mean(rating))
```

```
##   mean(rating)
## 1     4.109333
```

```
edx %>% filter(userId == 11129 & genres == "Horror")
```

```
##    userId movieId rating  timestamp
## 1:  11129    1974    0.5 1055555302
## 2:  11129    1983    0.5 1055301097
## 3:  11129    1984    0.5 1055301106
## 4:  11129    1985    0.5 1055301088
## 5:  11129    1986    0.5 1055301103
## 6:  11129    6220    0.5 1055129973
## 7:  11129    6290    0.5 1055129971
##                                              title genres
## 1:                          Friday the 13th (1980) Horror
## 2:                             Halloween II (1981) Horror
## 3:        Halloween III: Season of the Witch (1982) Horror
## 4:  Halloween 4: The Return of Michael Myers (1988) Horror
## 5: Halloween 5: The Revenge of Michael Myers (1989) Horror
## 6:                                  Willard (2003) Horror
## 7:                     House of 1000 Corpses (2003) Horror
```

Data Wrangling

It has been seen that there is information that seems to impact on the mean rating of a movie. The information is:

```
Movie mean rating (1)
User mean rating (2)
Genre mean (3)
User mean rating per genre (4)
Year of the movie (5)
```

To make value of this information, it is going to be extracted from the data and added to the table. In short, the following code was developed to wrangle the data. It was developed in a function in order to better replicate in the final dataset.

```
add_useful_columns <- function(table){

# (1)
```

```r
# Movie mean rating

mu <- mean(table$rating)

dat_permovie <- table %>%
  group_by(movieId) %>%
  summarise(movie_mean_rating = mean(rating))

# Adding the column
table <- left_join(table, dat_permovie, by = "movieId")

# (2)
# user mean rating
dat_peruser <- table %>%
  group_by(userId) %>%
  summarise(user_mean_rating = mean(rating))

# Adding the column
table <- left_join(table, dat_peruser, by = "userId")

# (3)
# Getting genre mean
dat_pergenres <- table %>%
  group_by(genres) %>%
  summarise(genre_mean_rating = mean(rating))

# Adding the column
table <- left_join(table, dat_pergenres, by = "genres")

# (4)
# Getting user mean per genre
# To not overtrain the mean, only considering where there is more than 4 ratings
# per genre.
# If not, considering the average between user mean rating and movie mean rating
table <- table %>%
  group_by(genres, userId) %>%
  mutate(user_genre_mean = ifelse(n() >= 4, mean(rating),
                                  (user_mean_rating+movie_mean_rating)/2)) %>%
  ungroup()


# (5)
# Getting the year
# Getting the year within the parenthesis
year_vector <- str_extract(table$title, "\\(\\d\\d\\d\\d\\)")
# Removing the parenthesis
year_vector <- substring(year_vector, 2, nchar(year_vector)-1)
# Converting to numeric
year_vector <- as.numeric(year_vector)

# Adding the year column
table <- table %>% mutate(year = year_vector)
```

```r
return(table)
}
# End of function


# Adding the useful columns.
edx <- add_useful_columns(edx)
```

Partitioning the data in test and train set and creating the RSME function

Now we are ready to split the data into test and train set. We are also going to create a function to generate the RSME.

```r
# Leaving 10% of the data to test and creating test and train set
test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.1, list = FALSE)
train_set <- edx %>% slice(-test_index)
test_set <- edx %>% slice(test_index)

rm(test_index)

# Creating a function to return the root-mean-square deviation
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Creating the model

The data cleaning was done applying the function proposed in the data wrangling session.

From now on, a series of steps are going to be performed to better model the problem. The logic must also be followed with the given code and its outputs.

Determining a mean rate and predicting it

```r
mu_hat <- mean(train_set$rating)
mu_hat
```

```
## [1] 3.512509
```

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

```
## [1] 1.061135
```

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

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

Predicting using user mean rating

```
user_mean_rmse <- RMSE(test_set$rating, test_set$user_mean_rating)
user_mean_rmse
```

```
## [1] 0.971246
```

```
rmse_results <- bind_rows(rmse_results, data_frame(method="User mean rating",
                                                   RMSE = user_mean_rmse ))
rmse_results
```

```
## # A tibble: 2 x 2
##   method            RMSE
##   <chr>            <dbl>
## 1 Just the average 1.06
## 2 User mean rating 0.971
```

Predicting using movie mean rating

```
movie_mean_rmse <- RMSE(test_set$rating, test_set$movie_mean_rating)
movie_mean_rmse
```

```
## [1] 0.9428615
```

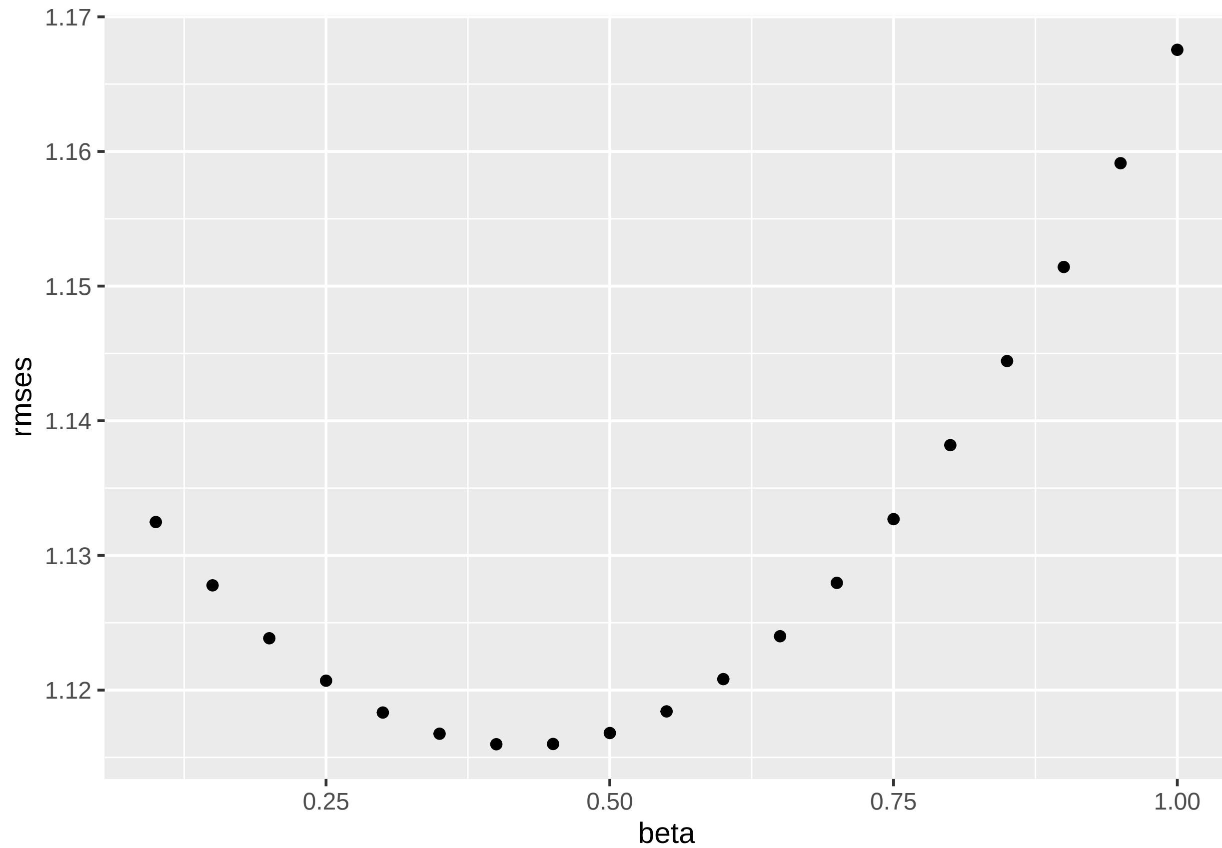
```
rmse_results <- bind_rows(rmse_results, data_frame(method="Movie mean rating",
                                                   RMSE = movie_mean_rmse ))
rmse_results
```

```
## # A tibble: 3 x 2
##   method             RMSE
##   <chr>             <dbl>
## 1 Just the average  1.06
## 2 User mean rating  0.971
## 3 Movie mean rating 0.943
```

```
rm(movie_mean_rmse, naive_rmse, user_mean_rmse)
```

Applying movie mean rating and user mean rating and finding a coeficient to get the best result

```
# beta is the fraction for the movie
beta <- seq(0.1, 1, 0.05)

# Applying beta
rmses <- sapply(beta, function(l){
  val <- RMSE(test_set$rating, (train_set$movie_mean_rating*l +
                                  (1-l)*train_set$user_mean_rating))
  return(val)
})

# Best beta
qplot(beta, rmses)
```




```
beta <- beta[which.min(rmses)]

movie_user_mean_rmse <- RMSE(test_set$rating,
                             (test_set$movie_mean_rating*beta +
                                (1-beta)*test_set$user_mean_rating))
movie_user_mean_rmse
```

```
## [1] 0.9142978
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie & user mean rating",
                                     RMSE = movie_user_mean_rmse ))
rmse_results
```

```
## # A tibble: 4 x 2
##   method                    RMSE
##   <chr>                    <dbl>
## 1 Just the average         1.06
## 2 User mean rating         0.971
## 3 Movie mean rating        0.943
## 4 Movie & user mean rating 0.914
```

Applying a linear regression and assesing the error

```
lm_fit <- train_set %>%
  lm(rating ~ movie_mean_rating +
       user_mean_rating +
```

```
        genre_mean_rating +
        user_genre_mean +
        year, data=.)

mu_hat_linear <- predict(lm_fit, newdata = test_set, type = "response")

lm_rmse <- RMSE(test_set$rating, mu_hat_linear)
lm_rmse
```

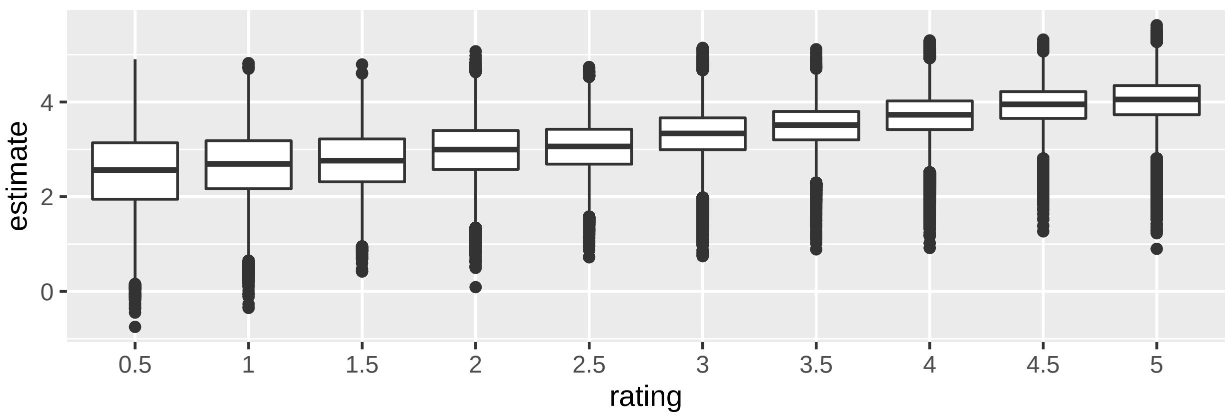
```
## [1] 0.8506361
```

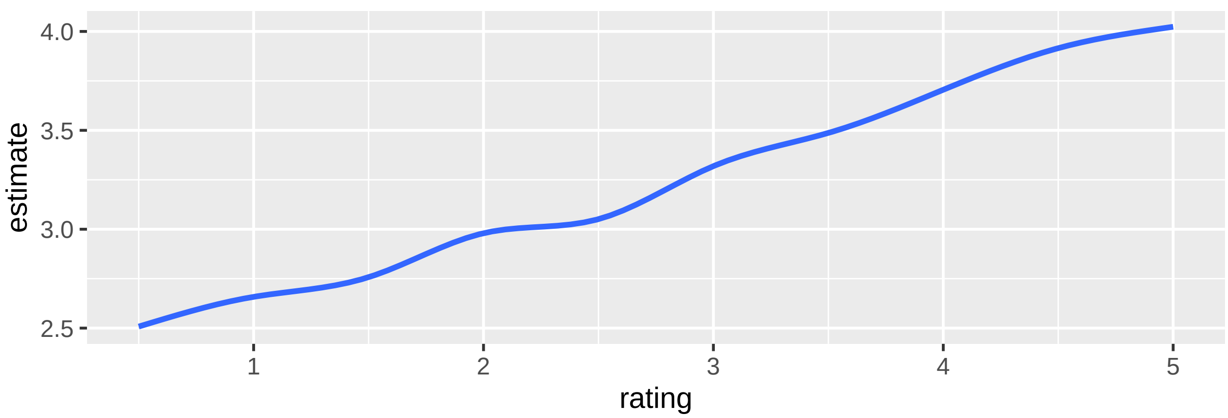
```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Linear regression",
                                     RMSE = lm_rmse ))
rmse_results
```

```
## # A tibble: 5 x 2
##   method                    RMSE
##   <chr>                    <dbl>
## 1 Just the average         1.06
## 2 User mean rating         0.971
## 3 Movie mean rating        0.943
## 4 Movie & user mean rating 0.914
## 5 Linear regression        0.851
```

To assess the error a plot that compares the predicted rating and the actual rating was built. It is possible to observe that for higher ratings the model generally predicts a lower value, while for lower ratings the predicted value is higher.

```
# Graph that plots true rating and estimate
a <- test_set %>%
  mutate(estimate = mu_hat_linear) %>%
  mutate(diff = rating - estimate)

box <- a %>% select(estimate, rating) %>%
  mutate(rating = as_factor(rating)) %>%
  ggplot(aes(rating, estimate)) +
  geom_boxplot()

smooth <- a %>% select(estimate, rating) %>%
  ggplot(aes(rating, estimate)) +
  geom_smooth()

require(gridExtra)
grid.arrange(box, smooth)
```






```r
rm(a, box, smooth)
```

Analysing the linear regression coeficcients it is possible to see that the movie mean rating and the user preference for an specific genre are the factors that most contribute for the prediction.

The coefficient was also multiplied to the mean value, so it is possible to understand as it increases or decreases the predicted rating. It is curious that the year and the genre mean rating have such coefficients when we compare it to results from the data exploration step. It is a way that the linear regression adjusted so other coefficients could better predict the rating.

```r
# Analysing the linear regression
tidy(lm_fit)
```

```
## # A tibble: 6 x 5
##   term               estimate std.error statistic  p.value
##   <chr>                 <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)       -2.31     0.0461        -50.0 0
## 2 movie_mean_rating  0.751    0.000835      899.  0
## 3 user_mean_rating   0.369    0.00102       361.  0
## 4 genre_mean_rating -0.261    0.00134      -195.  0
## 5 user_genre_mean    0.658    0.00102       645.  0
## 6 year               0.000245 0.0000229      10.7 7.93e-27
```

```r
# Actually seeing as the coefficients relate to prediction

lm_fit$coefficients[1]
```

```
## (Intercept)
##   -2.306324
```

```r
lm_fit$coefficients[2] * mean(test_set$movie_mean_rating )
```

```
## movie_mean_rating
##          2.636002
```

```r
lm_fit$coefficients[3] * mean(test_set$user_mean_rating)
```

```
## user_mean_rating
##         1.296239
```

```r
lm_fit$coefficients[4] * mean(test_set$genre_mean_rating)
```

```
## genre_mean_rating
##         -0.916607
```

```r
lm_fit$coefficients[5] * mean(test_set$user_genre_mean)
```

```
## user_genre_mean
##        2.314877
```

```r
lm_fit$coefficients[6] * mean(test_set$year)
```

```
##      year
## 0.4878279
```

Although good, the estimate can get better. The minimum rate is 0.5 and the maximum is 5.

```r
# Set limits
minV <- 0.5
maxV <- 5

# Limit vector
mu_hat_linear <- sapply(mu_hat_linear, function(y) min(max(y,minV),maxV))

lm_limited_rmse <- RMSE(test_set$rating, mu_hat_linear)
lm_limited_rmse
```

```
## [1] 0.8505627
```

```r
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Linear regression limited",
                                     RMSE = lm_limited_rmse ))
rmse_results
```

```
## # A tibble: 6 x 2
##   method                     RMSE
##   <chr>                     <dbl>
## 1 Just the average          1.06
## 2 User mean rating          0.971
## 3 Movie mean rating         0.943
## 4 Movie & user mean rating  0.914
## 5 Linear regression         0.851
## 6 Linear regression limited 0.851
```

Predicting with smooth

Can we still get a better prediction? During data exploration we saw that many values, like the year, were not linear. Let us try with the very smooth used in *ggplot mgcv::gam()*.

```r
library(mgcv)

gam_fit <- train_set %>%
  gam(rating ~ movie_mean_rating +
        user_mean_rating +
        genre_mean_rating +
        user_genre_mean +
        year, data=.)

mu_hat_gam <- predict(gam_fit, newdata = test_set)

gam_rmse <- RMSE(test_set$rating, mu_hat_gam)
gam_rmse
```

```
## [1] 0.8506361
```

There is no great change, so it does not worth the computational effort.

```r
rmse_results <- bind_rows(rmse_results, data_frame(method="GAM", RMSE = gam_rmse ))
rmse_results
```

```
## # A tibble: 7 x 2
##   method                     RMSE
##   <chr>                     <dbl>
## 1 Just the average          1.06
## 2 User mean rating          0.971
## 3 Movie mean rating         0.943
## 4 Movie & user mean rating  0.914
## 5 Linear regression         0.851
## 6 Linear regression limited 0.851
## 7 GAM                       0.851
```

Predicting the model

The best cost-benefit method was linear regression with minimum and maximum values.

Validating the model:

```r
validation <- add_useful_columns(validation)
```

Predicting and showing the RMSE:

```r
mu_hat_linear <- predict(lm_fit, newdata = validation)
# Set limits
minV <- 0.5
maxV <- 5
mu_hat_linear <- sapply(mu_hat_linear, function(y) min(max(y,minV),maxV))

lm_rmse <- RMSE(validation$rating, mu_hat_linear)
lm_rmse
```

```
## [1] 0.8446204
```

Results

Using linear regression that trimmed maximum and minimun values allied with a proper data wrangling, a good result was obtained. The model performance was still very good when compared with other methods.

```
## [1] 0.8446204
```

As there are less genres and the prediction cut was 4 movies, it is arguable the the regression could be using to much from the rating. For this reason, a new regression without considering the genre at all is presented.

```
lm_fit_without_genre <- train_set %>%
  lm(rating ~ movie_mean_rating + user_mean_rating + genre_mean_rating + year, data=.)

mu_hat_linear_without_genre <- predict(lm_fit_without_genre,
                                       newdata = validation, type = "response")
```

The prediction is still very good:

```
lm_rmse <- RMSE(validation$rating, mu_hat_linear_without_genre)
lm_rmse
```

```
## [1] 0.8452198
```

Conclusion

Using the knowledge aquired in the course a reasonable method and model was proposed to predict movie ratings. The data wrangling session was key to obtain the result shown. The main limitation of the model is that it is better predicting ratings for users that have more reviews. The final RSME is 0.84.

```
## [1] 0.8446204
```

Different approaches were used, being linear regression the one that best balances computational effort with results.

```
rmse_results
```

```
## # A tibble: 7 x 2
##   method                     RMSE
##   <chr>                     <dbl>
## 1 Just the average          1.06
## 2 User mean rating          0.971
## 3 Movie mean rating         0.943
## 4 Movie & user mean rating  0.914
## 5 Linear regression         0.851
## 6 Linear regression limited 0.851
## 7 GAM                       0.851
```

A possible future work would be to separate users by amount of ratings per genre they make. This way users with lower ratings could be adressed better. Another approach can also be developed to lower rating movies.