MovieLens Project

Florian Javelle

30/04/2021

1. Introduction

Recommendation systems can be described as algorithms using ratings that users have given items to emit specific recommendations [1]. The interest in these systems has greatly increased in the past 20 years. One of the most striking examples is the Netflix Prize competition launched in 2006. Through a machine learning and data mining competition, Netflix has challenged data scientists to increase the accuracy of its recommendation system (i.e., its root mean squared error [RMSE]) by a minimum of 10 percent [2]. The reward for the winner was 1 million dollars [2]. The team BellKor's Pragmatic Chaos decreased the RMSE of the Netflix recommendation system from .9525 to .8558 (i.e., 10.06 percent) and won the competition [3]. To achieve such a result, they used matrix factorization and restricted Boltzmann machines [3]. Working with similar techniques [1], this project aims to create a proper recommendation system using the 10 Million version of the MovieLens data set. This data set has been provided by GroupLens Research that has gathered and made publicly available rating data sets from the website MovieLens [4]. I will first explore the data set, reshape it if necessary, and then determine potential predictors to include in my recommendation system. I will continue by training my recommendation system using 90 percent of the MovieLens data set. Finally, I will test my algorithm on the 10 percent leftover of the data set.

2. Methods and Results

2.1 Training and validation sets - GIVEN CODE

First, I have to charge the MovieLens data set, and split it between a training set and a validation set using the given code by the Capstone course, Movielens unit.

```
#The following code assumes that all required libraries are already installed on R.
library(tidyverse)
library(caret)
library(data.table)

# 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 <- 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)
```

```r
colnames(movies) <- c("movieId", "title", "genres")

# if using R 3.6 or earlier:
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, sample.kind="Rounding") # if using R 3.5 or earlier, use `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)

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

2.2 Data exploration

I am going to explore the data set to learn more about it and see if some parts need to be reshaped to be usable. The following functions will help me to get an idea of the size, and characteristics of the data set and its variables.

```r
#To see the dimensions of edx
dim(edx)
```

```
## [1] 9000055       6
```

The data set "edx" has 6 variables and 9 000 055 observations.

```r
#To have a broad idea of the values included in each variable
glimpse(edx)
```

```
## Rows: 9,000,055
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, ~
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 420, ~
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ~
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83898~
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)", "S~
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|Sci~
```

Even though not very elaborated, the "glimpse" function provides interesting information. For example, it appears that the variables "timestamp" and "genres" are not in formats that are easy to use. I will have to change it. Furthermore, the variable "title" provides the titles from the rated movies and a piece of additional information (i.e., the year of release of the movie) that could be useful to consider for the recommendation system.

Before, reshaping the data set, I am going to finish this brief data exploration.

```
#To see how many distinct movies are included in the data set
n_distinct(edx$movieId)
```

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

```
#To see how many distinct users are included in the data set
n_distinct(edx$userId)
```

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

```
#To see how many distinct movie genres are included in the data set
n_distinct(edx$genres)
```

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

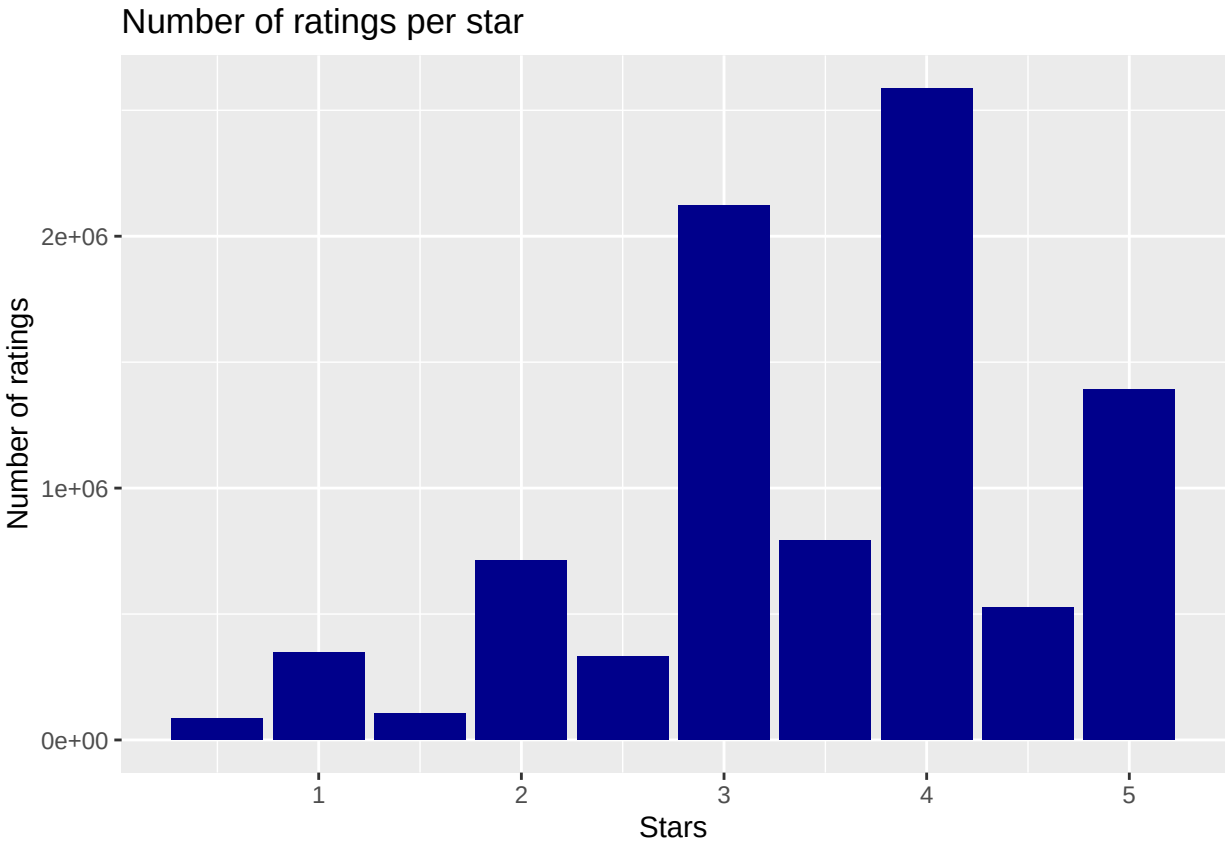
Within the data set "edx", 10 677 distinct movies having 797 different genres (but with mixed characteristics; e.g., "Action|Crime|Thriller", "Action|Drama|Sci-Fi|Thriller") were rated by 69 878 distinct users.

```
#To get a more accurate idea of the variables in edx
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

The rating of movies goes from 0.5 to 5, with an average of 3.512. I am going to check if some ratings (i.e., stars) are more frequent than others using a histogram.

```
#to have a histogram showing the number of ratings per star
edx %>%
  group_by(rating) %>%
  ggplot(aes(rating)) +
  geom_bar(fill="dark blue") +
  labs(x="Stars", y="Number of ratings") +
  ggtitle("Number of ratings per star")
```



From this plot, it appears that the "full star" ratings (i.e., 1,2,3,4, 5) are more frequent than the "half star" ratings.

Now, I am going to rank movies based on the number of ratings they received and display the top 10.

```
#to have the top 10 of movies with the biggest quantity of ratings
edx %>%
  group_by(movieId,title, genres) %>%
  summarize(n = n()) %>% top_n(10) %>%
  arrange(desc(n))
```

```
## # A tibble: 10,677 x 4
## # Groups:   movieId, title [10,677]
##    movieId title                                     genres                   n
##      <dbl> <chr>                                     <chr>                <int>
##  1     296 Pulp Fiction (1994)                       Comedy|Crime|Drama   31362
##  2     356 Forrest Gump (1994)                       Comedy|Drama|Romance|~ 31079
##  3     593 Silence of the Lambs, The (1991)          Crime|Horror|Thriller  30382
##  4     480 Jurassic Park (1993)                      Action|Adventure|Sci-~ 29360
##  5     318 Shawshank Redemption, The (1994)          Drama                  28015
##  6     110 Braveheart (1995)                         Action|Drama|War       26212
##  7     457 Fugitive, The (1993)                      Thriller               25998
##  8     589 Terminator 2: Judgment Day (1991)         Action|Sci-Fi          25984
##  9     260 Star Wars: Episode IV - A New Hope (a.k~ Action|Adventure|Sci-~ 25672
## 10     150 Apollo 13 (1995)                          Adventure|Drama        24284
## # ... with 10,667 more rows
```

This list shows that the movie "Pulp Fiction" is the most rated movie from the data set (31 362 ratings). I

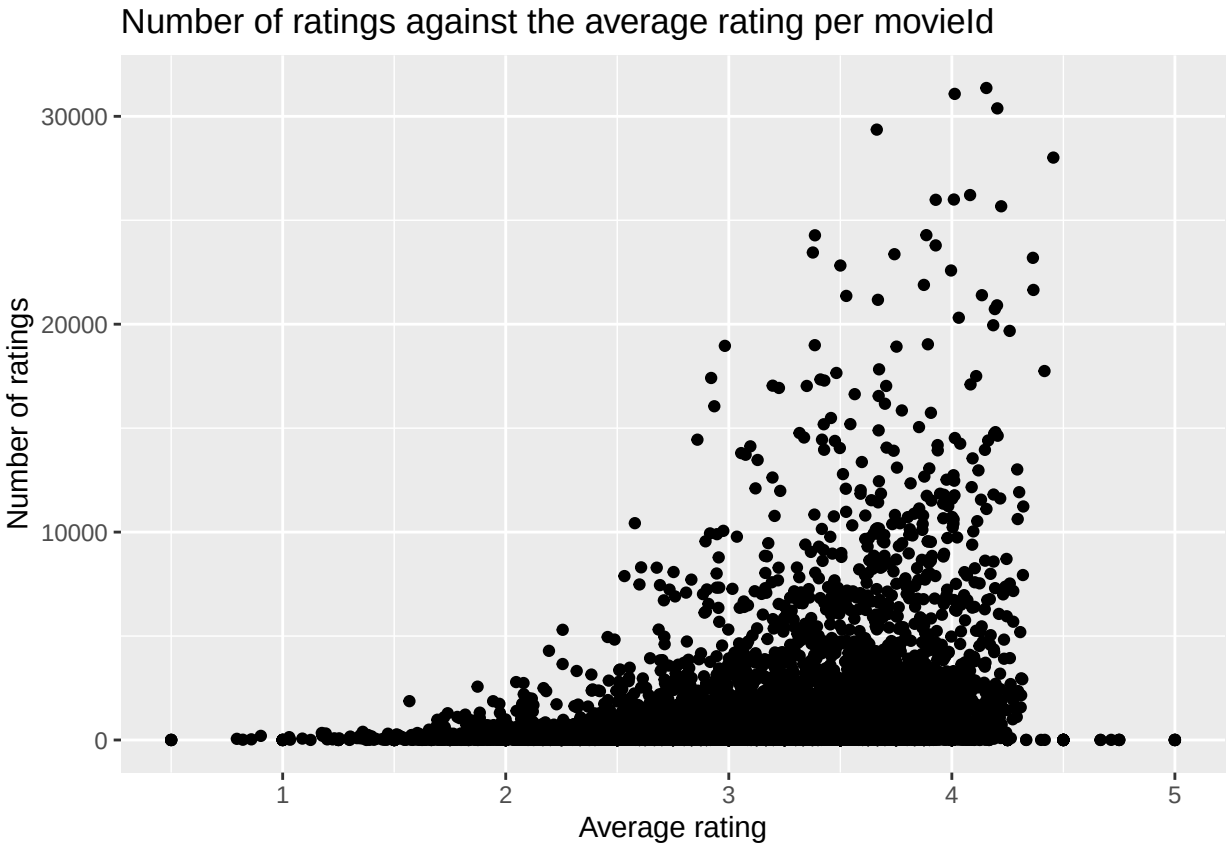
am, now, going to display the best-rated movies.

```
#to list movies based on their average ratings (highest at the top)
edx %>%
  group_by(movieId) %>%
  summarize(rating_avg=mean(rating), title, n=n()) %>%
  arrange(desc(rating_avg))
```

```
## # A tibble: 9,000,055 x 4
## # Groups:   movieId [10,677]
##    movieId rating_avg title                                                n
##      <dbl>      <dbl> <chr>                                            <int>
##  1    3226       5    Hellhounds on My Trail (1999)                        1
##  2   33264       5    Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)                   2
##  3   33264       5    Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)                   2
##  4   42783       5    Shadows of Forgotten Ancestors (1964)                1
##  5   51209       5    Fighting Elegy (Kenka erejii) (1966)                 1
##  6   53355       5    Sun Alley (Sonnenallee) (1999)                       1
##  7   64275       5    Blue Light, The (Das Blaue Licht) (1932)             1
##  8    5194       4.75 Who's Singin' Over There? (a.k.a. Who Sings Over Th~     4
##  9    5194       4.75 Who's Singin' Over There? (a.k.a. Who Sings Over Th~     4
## 10    5194       4.75 Who's Singin' Over There? (a.k.a. Who Sings Over Th~     4
## # ... with 9,000,045 more rows
```

Interestingly, the top 10 of this new list is entirely different (and unknown to me) compared to the previous one. It also shows that these "best" movies have just a few ratings. To have a better idea of the rating distribution per movieID, I am going to plot it.

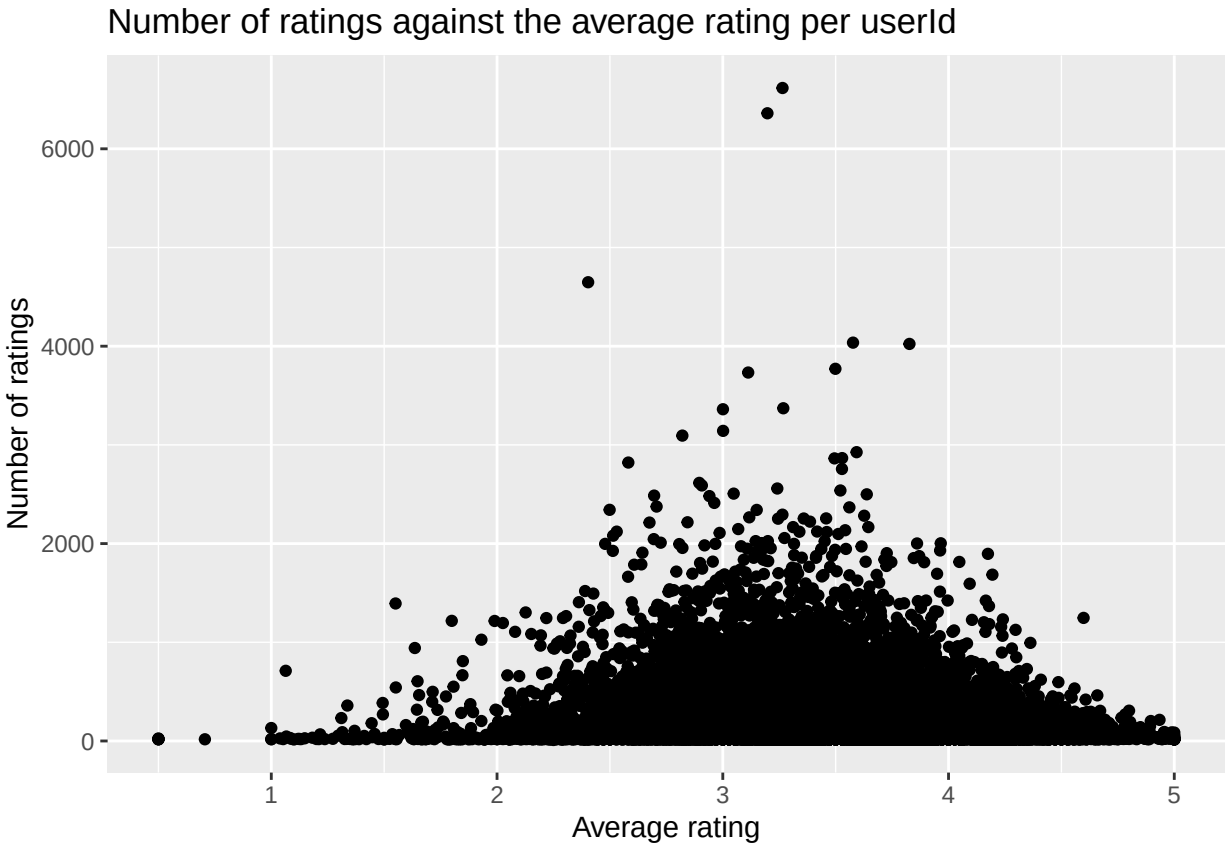
```
#to plot the number of ratings against the average rating per movieID
edx %>%
  group_by(movieId) %>%
  summarize(rating=mean(rating), n=n()) %>%
  ggplot(aes(rating, n))+ geom_point()+labs(x="Average rating", y="Number of ratings")+
  ggtitle("Number of ratings against the average rating per movieId")
```



This very simple dot-plot displays two important pieces of information. 1) It confirms the previous point showing that the best average ratings, but also the worst ones, have only a few ratings. 2) Up to a specific cutoff (around 4.25 stars), the more ratings a movie has, the better its average rating tends to be. This information suggests that it might be important to implement "regularization techniques"(i.e., to penalize large estimates that are created because of small sample sizes) in the recommendation system.

I am going to create the same graph but per userID.

```
#to plot the number of ratings against the average rating per userID
edx %>%
  group_by(userId) %>%
  summarize(rating=mean(rating), n=n()) %>%
  ggplot(aes(rating, n))+ geom_point()+labs(x="Average rating", y="Number of ratings")+
  ggtitle("Number of ratings against the average rating per userId")
```



Here again, it appears clearly that users that have only a few ratings are either very positive (between 4 and 5 stars) or very negative (between 1 and 2 stars). UserId is also a variable that might benefit from regularization.

To continue my exploration of the data set, I will have to adapt a few parameters. I am going to change the format of "timestamp", and then extract the year of release from the variable "titles".

```
library(lubridate)
```

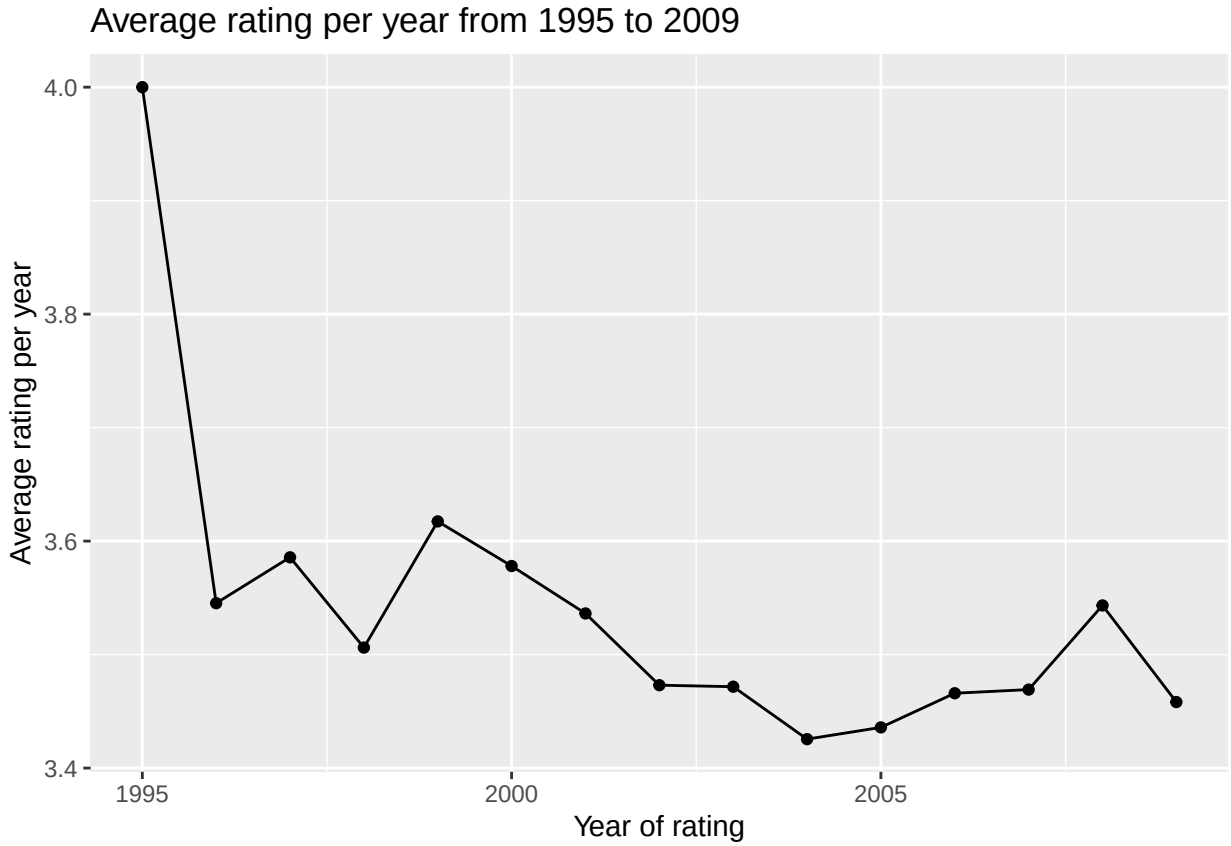
```
edx <- mutate(edx, year_2 = year(as_datetime(timestamp)))
head(edx)
```

```
##    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 year_2
## 1:                Comedy|Romance   1996
## 2:         Action|Crime|Thriller   1996
## 3:  Action|Drama|Sci-Fi|Thriller   1996
## 4:       Action|Adventure|Sci-Fi   1996
## 5: Action|Adventure|Drama|Sci-Fi   1996
## 6:       Children|Comedy|Fantasy   1996
```

I have create a seventh variable (i.e., "year_2") in the edx data set reporting the year of the timestamp. I will evaluate in a later stage if I need to be more accurate with this new variable (i.e., include months or weeks).

I am going to plot the average rating per year over time to assess if the latter is worth implementing to the recommendation system.

```
#To do a graph plotting the average rating per year over time
edx %>%
  group_by(year_2) %>%
  summarize(rating=mean(rating))%>%
  ggplot(aes(year_2, rating))+
  geom_point()+ geom_line()+labs(x="Year of rating", y="Average rating per year")+
  ggtitle("Average rating per year from 1995 to 2009")
```



The plot displays that the average rating decreased over time. Nevertheless, it is worth noting that the year of release of MovieLens website (1995) has a higher average rating than all other years.
To have a better idea of the overall effect size, I am going to run a linear regression.

```
#Linear regression to evaluate if the variable "year of release" can explain some variance
#in the rating results
model  <- lm(rating ~ year_2, data = edx)
summary(model)
```

```
##
## Call:
## lm(formula = rating ~ year_2, data = edx)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0044 -0.5448  0.4249  0.5259  1.5563
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 23.7483054  0.1906154   124.6   <2e-16 ***
## year_2      -0.0101068  0.0000952  -106.2   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.06 on 9000053 degrees of freedom
## Multiple R-squared:  0.001251,   Adjusted R-squared:  0.001251
## F-statistic: 1.127e+04 on 1 and 9000053 DF,  p-value: < 2.2e-16
```

The results are significant but with a very small effect size (R^2=.125%, p<.001). By curiosity, I am going to verify if being more accurate on the variable "timestamp" changes the results.

```
#Transform the variable "timestamp" to a usable format with months as measuring unit.
edx <- mutate(edx, month_2 = month(as_datetime(timestamp)))
#Linear regression to evaluate if the variable "year of release" can explain some variance
#in the rating results
model2  <- lm(rating ~ month_2, data = edx)
summary(model2)
```

```
##
## Call:
## lm(formula = rating ~ month_2, data = edx)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0312 -0.5204  0.4688  0.5048  1.5084
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.4880373  0.0007661 4553.18   <2e-16 ***
## month_2     0.0035996  0.0001002   35.94   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.06 on 9000053 degrees of freedom
## Multiple R-squared:  0.0001435,  Adjusted R-squared:  0.0001434
## F-statistic:  1292 on 1 and 9000053 DF,  p-value: < 2.2e-16
```

```
#Transform the variable "timestamp" to a usable format with weeks as measuring unit.
edx <- mutate(edx, week_2 = week(as_datetime(timestamp)))
#Linear regression to evaluate if the variable "year of release" can explain some variance
#in the rating results
model3  <- lm(rating ~ week_2, data = edx)
summary(model3)
```

```
##
```

```
## Call:
## lm(formula = rating ~ week_2, data = edx)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0335 -0.5210  0.4665  0.5048  1.5098
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.4893589  0.0007269 4800.20   <2e-16 ***
## week_2      0.0008330  0.0000229   36.38   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.06 on 9000053 degrees of freedom
## Multiple R-squared:  0.000147,   Adjusted R-squared:  0.0001469
## F-statistic:  1323 on 1 and 9000053 DF,  p-value: < 2.2e-16
```

The results stay significant but the effect sizes are 10 times smaller (R^2=.0147%, p<.001) than when using the measuring unit "year". Therefore, I am going to keep the unit "year" for further analyses. The graph also displays that movies released between the years 1990 and 2000 received more ratings than others.

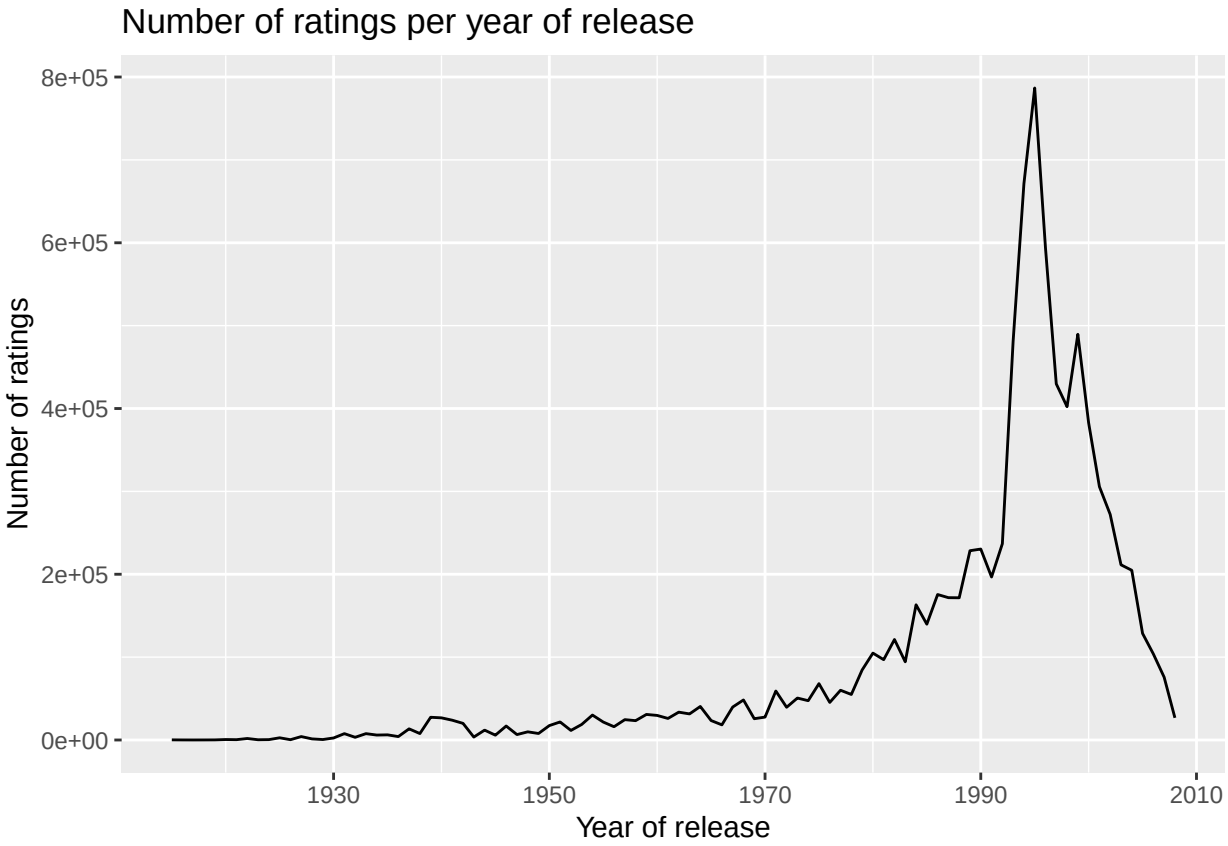
Now, I am going to split the variable "title" in order to obtain titles and years of release.

```
#To split the variable "title" into 2 variables (title and year of release)
edx <- edx%>%
  extract(title, c("title", "year_release"), regex = "(.*)\\s\\((\\d+)\\)", convert = TRUE)
head(edx)
```

```
##    userId movieId rating timestamp                title year_release
## 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 year_2 month_2 week_2
## 1:                Comedy|Romance   1996       8     31
## 2:          Action|Crime|Thriller   1996       8     31
## 3:  Action|Drama|Sci-Fi|Thriller   1996       8     31
## 4:       Action|Adventure|Sci-Fi   1996       8     31
## 5: Action|Adventure|Drama|Sci-Fi   1996       8     31
## 6:       Children|Comedy|Fantasy   1996       8     31
```

I am going to plot the number of ratings per year of release. Then, I am going to run another linear regression to evaluate if the year of release is an important predictor in explaining the variance of the rating results.

```
#to plot the number of ratings per year of release
edx %>%
  group_by(year_release) %>%
  summarize(count = n()) %>%
  ggplot(aes(year_release, count)) +
  geom_line()+
  ggtitle("Number of ratings per year of release")+
  labs(x="Year of release", y="Number of ratings")
```




```
#linear regression
model  <- lm(rating ~ year_release, data = edx)
summary(model)
```

```
##
## Call:
## lm(formula = rating ~ year_release, data = edx)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.7212 -0.5051  0.1174  0.5797  1.6550
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.226e+01  5.140e-02   433.2   <2e-16 ***
## year_release -9.422e-03  2.583e-05  -364.8   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.053 on 9000053 degrees of freedom
## Multiple R-squared:  0.01457,    Adjusted R-squared:  0.01457
## F-statistic: 1.331e+05 on 1 and 9000053 DF,  p-value: < 2.2e-16
```

The results are significant and have a small effect size (R^2=1.46%, p<.001). It is, however, 10 times bigger than the effect size for the variable "timestamp" (unit: year).

Now, I am interesting myself in the variable "genres". First, I have to separate the mixed genres into a single genre cell to have a proper analysis of the different types.

```
# to separate genres into single genre cell
edx1 <- edx %>% separate_rows(genres, sep ="\\|")
n_genres<-unique(edx1$genres)
length(n_genres)
```

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

```
# to display the number of ratings per genre
edx1 %>%
  group_by(genres) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
```

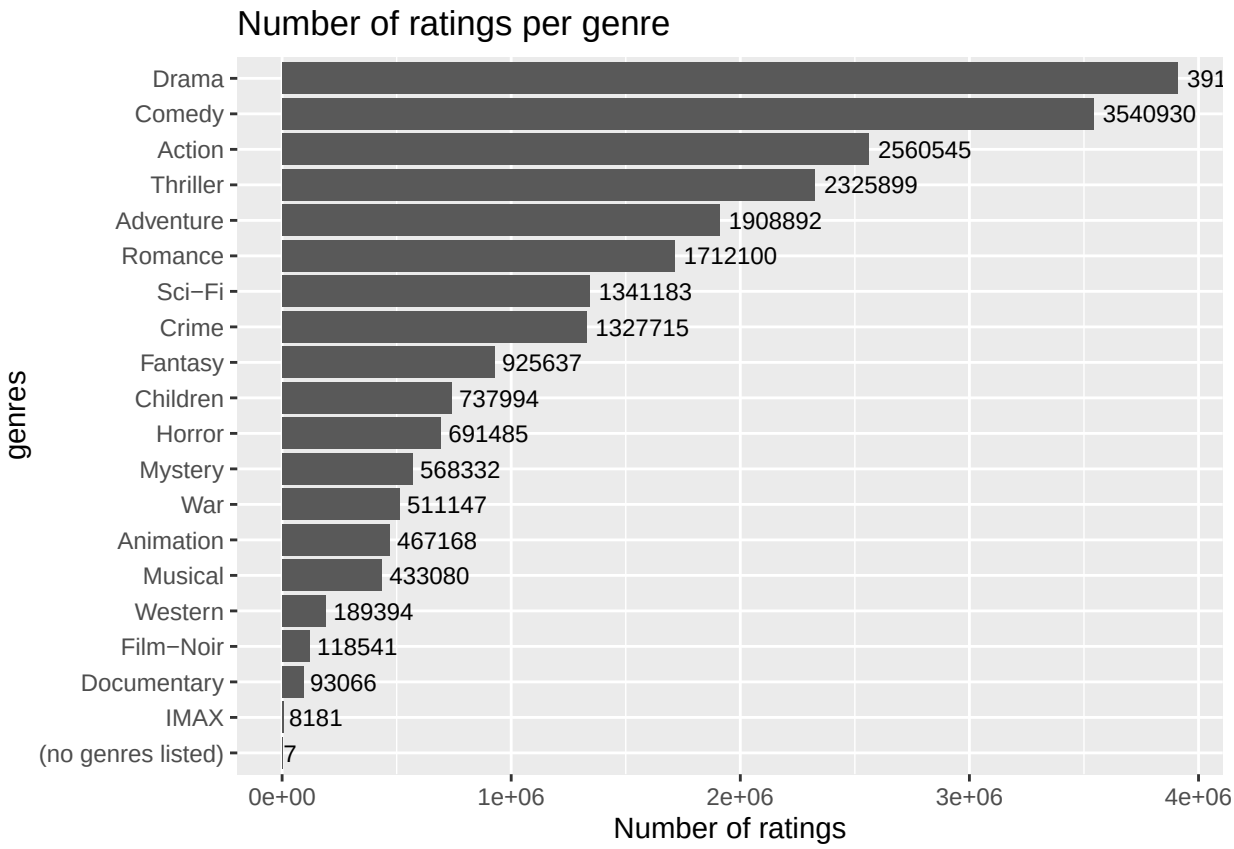
```
## # A tibble: 20 x 2
##    genres               count
##    <chr>                <int>
##  1 Drama              3910127
##  2 Comedy             3540930
##  3 Action             2560545
##  4 Thriller           2325899
##  5 Adventure          1908892
##  6 Romance            1712100
##  7 Sci-Fi             1341183
##  8 Crime              1327715
##  9 Fantasy             925637
## 10 Children            737994
## 11 Horror              691485
## 12 Mystery             568332
## 13 War                 511147
## 14 Animation           467168
## 15 Musical             433080
## 16 Western             189394
## 17 Film-Noir           118541
## 18 Documentary          93066
## 19 IMAX                  8181
## 20 (no genres listed)       7
```

Twenty different genres were detected in the data set. The genres "Drama", "Comedy" and "Action" are the top three the most rated.

However, presented this way, it is difficult to evaluate the differences between the number of ratings per genre. To be able to visualize it more quickly, I am going to create a bar plot presenting the same information.

```
# Number of ratings per genre
edx1 %>%
  group_by(genres)%>%
  summarize(count = n()) %>%
  ggplot(aes(x=count, y=reorder(genres, count))) +
  geom_bar(stat="identity")+
  labs(x= "Number of ratings", y="genres")+
```

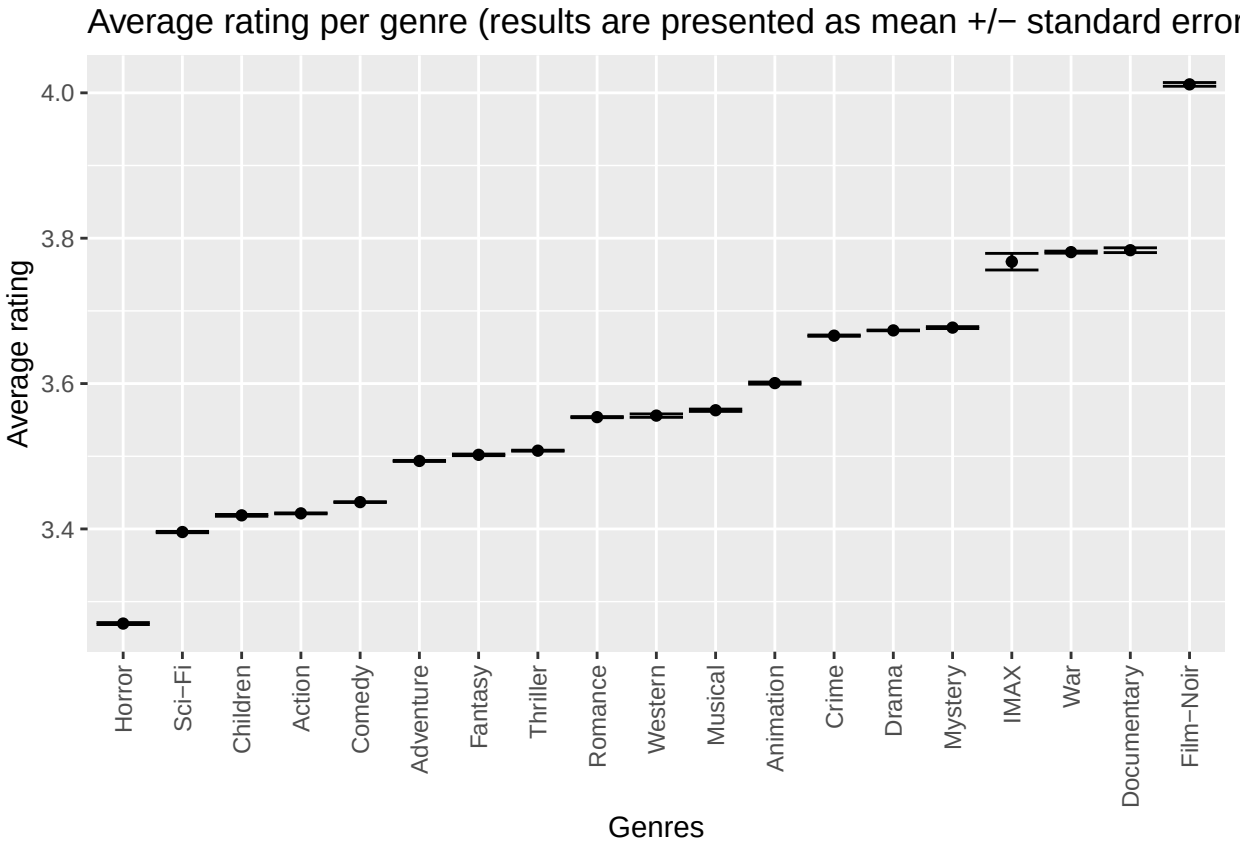
```
  geom_text(aes(label=count), hjust=-0.1, size=3)+
  ggtitle("Number of ratings per genre")
```



It looks better this way.

I am now going to evaluate if there is a difference in the average rating between genres. For this analysis, I am going to remove the category "(no genres listed)" that has only 7 ratings and is a clear outlier.

```
#To plot the average rating per genre for 19 genres ("(no genres listed)" excluded)
plot<-edx1 %>%
  group_by(genres)%>%
  filter(genres !="(no genres listed)")%>%
  summarize(count = n(), means=mean(rating), std=sd(rating)/sqrt(count)) %>%
  mutate(genres=reorder(genres, means))%>%
  ggplot(aes(x=genres, y=means,ymin=means-std, ymax=means+std)) +
  geom_point()+
  geom_errorbar()+
  labs(x= "Genres", y="Average rating")+
  ggtitle("Average rating per genre (results are presented as mean +/- standard error)")
plot + theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```



The category "Film-Noir" has the highest average rating. This category was, however, one of the genres with the lowest quantity of ratings. Once again, implementing regularization to the recommendation system seems to be suitable.

3. Results

Based on this data exploration step realized in the previous section, I am going to build my recommendation system. The accuracy of recommendation systems is evaluated by the RMSE (see function below). The RMSE has the same unit as the predicted variable (the rating) and thus can be interpreted in the same manner as a standard deviation. Therefore, the bigger it is, the least accurate the prediction will be.

```
#RMSE function
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2, na.rm = TRUE))}
```

To begin, I am going to create the simplest model possible and I will use it as a reference. In this model, I assume that all movies are rated the same and that all the variance can be explained by the random error. Thus, my first and only predictor for this model is the average rating of my training set.

```
#model 1 - reference rmse (naive)
mean_movies_mvl<-mean(edx$rating)

ref_rmse <- RMSE(validation$rating, mean_movies_mvl)
print(ref_rmse)
```

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

```
rmse_results <- data_frame(method = "Just the average rating", RMSE = ref_rmse)
```

Now, I have a reference value (ref_rmse=1.0612). Nevertheless, as displayed during the data exploration step, this first model does not properly consider the variance across the ratings. Indeed, in the previous section, I have displayed potential predictors that could be used to give a better fit to our model.

1) Some movies were rated higher than others. Thus, I am going to add a movie bias (b_m) term (i.e., or effect when using the statistic textbooks terminology [1]) to the model. As explained in the Machine Learning course, in such a situation the least square estimate of b_m is the difference between values from the validation set and the average rating from the training set [1].

```
#model 2 - average rating + movie bias (b_m)
b_m <- edx %>%
  group_by(movieId) %>%
  summarize(b_m = mean(rating - mean_movies_mvl))

predicted_ratings <- mean_movies_mvl + validation %>%
  left_join(b_m, by='movieId') %>%
  .$b_m

rmse_2<-RMSE(predicted_ratings, validation$rating)

rmse_results <- data_frame(method = c("Just the average rating", "movie error"),
                           RMSE = c(ref_rmse, rmse_2))
format.data.frame(rmse_results, digits=4)
```

```
##                     method   RMSE
## 1 Just the average rating 1.0612
## 2             movie error 0.9439
```

The RMSE has largely decreased (rmse_2=.9439).

2) The data exploration has also shown important variability between users. Thus, I am going to add a user bias (b_u) term (i.e., effect [1]) to the model.

```
#model 3 - average rating + movie bias (b_m) + user bias (b_u)
b_u <- edx %>%
  left_join(b_m, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mean_movies_mvl - b_m))

predicted_ratings <- validation %>%
  left_join(b_m, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  mutate(pred = mean_movies_mvl + b_m + b_u) %>%
  .$pred

rmse_3 <- RMSE(predicted_ratings, validation$rating)


rmse_results <- data_frame(method = c("Just the average rating", "movie error", "user error"),
                           RMSE = c(ref_rmse, rmse_2, rmse_3))
format.data.frame(rmse_results, digits=4)
```

```
##                     method   RMSE
## 1 Just the average rating 1.0612
## 2             movie error 0.9439
## 3              user error 0.8653
```

The RMSE has largely decreased, down to 0.8653.

3) The data exploration step has also suggested the year of release as a potential predictor of the variance in the ratings. Thus, I am going to attribute a bias term (i.e., effect [1]) for the year of release.

```
#model 4 - average rating + movie bias (b_m) + user bias (b_u) + year of release (b_y)
#First I have to extract the year of release in the validation set
validation <- validation%>%
  extract(title, c("title", "year_release"), regex = "(.*)\\s\\((\\d+)\\)", convert = TRUE)

#Now I am going to create the new model
b_y <- edx %>%
  left_join(b_m, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  group_by(year_release) %>%
  summarize(b_y = mean(rating - mean_movies_mvl - b_m - b_u))

predicted_ratings <- validation %>%
  left_join(b_m, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(b_y, by='year_release')%>%
  mutate(pred = mean_movies_mvl + b_m + b_u + b_y) %>%
  .$pred

rmse_4 <- RMSE(predicted_ratings, validation$rating)


rmse_results <- data_frame(method = c("Just the average rating", "movie error", "user error",
                                      "year of release error"), RMSE = c(ref_rmse, rmse_2,
                                                                         rmse_3, rmse_4))
format.data.frame(rmse_results, digits=4)
```

```
##                     method   RMSE
## 1 Just the average rating 1.0612
## 2             movie error 0.9439
## 3              user error 0.8653
## 4   year of release error 0.8650
```

The RMSE improved a little (rmse_4=0.8650).

4) Using linear regression in the previous section, I have shown that the year of rating might also be a useful predictor. However, considering the small effect size detected, it is likely that the influence of this variable on the recommendation system will be reduced.

```
#model 5 - average rating + movie bias (b_m) + user bias (b_u) + year of release (b_y)
# + year of rating (b_yr)
#First I have to transform the variable "timestamp" from the validation set
```

```
validation <- mutate(validation, year_2 = year(as_datetime(timestamp)))

#Now I am going to create the new model
b_yr <- edx %>%
  left_join(b_m, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(b_y, by='year_release') %>%
  group_by(year_2) %>%
  summarize(b_yr = mean(rating - mean_movies_mvl - b_m - b_u - b_y))

predicted_ratings <- validation %>%
  left_join(b_m, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(b_y, by='year_release')%>%
  left_join(b_yr, by='year_2')%>%
  mutate(pred = mean_movies_mvl + b_m + b_u + b_y + b_yr) %>%
  .$pred

rmse_5 <- RMSE(predicted_ratings, validation$rating)

rmse_results <- data_frame(method = c("Just the average rating", "movie error", "user error",
                                      "year of release error", "year of rating error"),
                           RMSE = c(ref_rmse, rmse_2, rmse_3, rmse_4, rmse_5))
format.data.frame(rmse_results, digits=4)
```

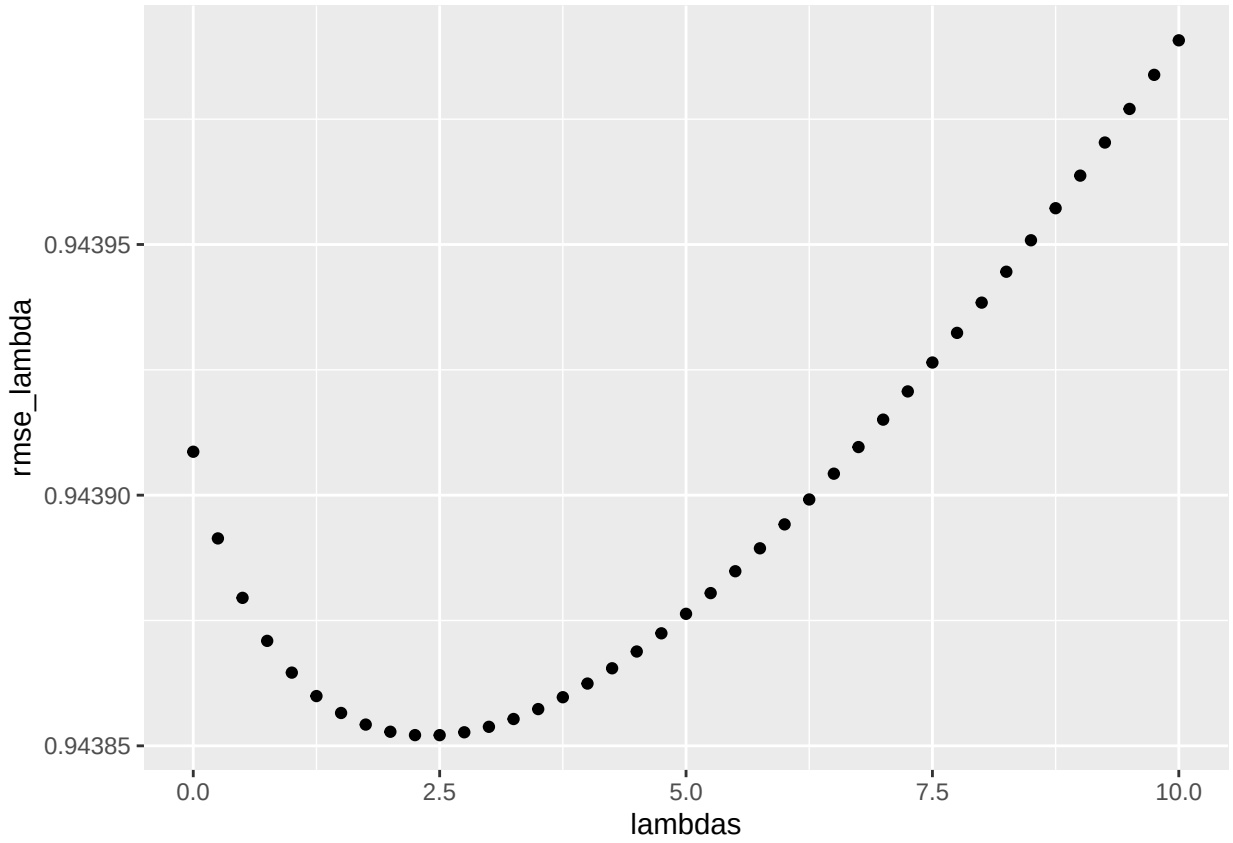
```
##                     method   RMSE
## 1 Just the average rating 1.0612
## 2             movie error 0.9439
## 3              user error 0.8653
## 4   year of release error 0.8650
## 5    year of rating error 0.8649
```

As expected the RMSE improved but in a very reduced manner (rmse_4=0.8650 vs rmse_5=0.8649). Another variable that has been shown to influence the ratings is the genres. Nevertheless, using genres will force me to create several rows for one observation creating duplicates. Thus, I a going to leave this potential predictor out of my analysis.

Even if the RMSE is already acceptable, I should be able to continue improving it. Indeed, during the data exploration step, I have demonstrated that the movies with the best and the worst ratings were also the ones with the least ratings. In the next model, I am going to include a concept that penalizes large estimates that are formed in reason of small sample sizes: the regularization. I am going to take a step back and first regularize the model including only the movie bias. I am also going to create a function that will help me to choose the most suitable lambda.

```
#model 6 - regularization ==> movies
#determine the lambda
lambdas <- seq(0, 10, 0.25)

rmse_lambda <- sapply(lambdas, function(l){

  mean_movies_mvl<-mean(edx$rating)

  b_m <- edx %>%
```

```
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mean_movies_mvl)/(n()+l))

  predicted_ratings <- validation %>%
    left_join(b_m, by = "movieId") %>%
    mutate(pred = mean_movies_mvl + b_m) %>%
    pull(pred)

  return(RMSE(predicted_ratings, validation$rating))
})
#to plot rmse against lambda
qplot(lambdas, rmse_lambda)
```




```
#to determine what lambda corresponds to the smallest RMSE
lambda<-lambdas[which.min(rmse_lambda)]

#RMSE with the selected lambda
b_m <- edx %>%
  group_by(movieId) %>%
  summarize(b_m = sum(rating - mean_movies_mvl)/(n()+lambda), n_m =n())

predicted_ratings <- validation %>%
  left_join(b_m, by='movieId') %>%
  mutate(pred = mean_movies_mvl + b_m) %>%
  .$pred
```

```
rmse_6 <- RMSE(predicted_ratings, validation$rating)


rmse_results <- data_frame(method = c("Just the average rating", "movie error", "user error",
                                      "year of release error", "year of rating error",
                                      "regularization - movieID"),
                           RMSE = c(ref_rmse, rmse_2, rmse_3, rmse_4, rmse_5, rmse_6))

format.data.frame(rmse_results, digit=4)
```

```
##                       method   RMSE
## 1  Just the average rating 1.0612
## 2              movie error 0.9439
## 3               user error 0.8653
## 4    year of release error 0.8650
## 5     year of rating error 0.8649
## 6 regularization - movieID 0.9439
```

Unfortunately, when only considering the model with movieID, the regularization did not change the RMSE (rmse_6=0.9439=rmse_2).

Nonetheless, all the variables that I have included in the previous models were not homogeneously spread across ratings, thus I am going to include them in the regularization system. One at a time. I continue with the userID.

```
#model 7 - regularization ==> movies and users
#determine the lambda
lambdas <- seq(0, 10, 0.25)

rmse_lambda <- sapply(lambdas, function(l){

  mean_movies_mvl<-mean(edx$rating)

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

  b_u <- edx %>%
    left_join(b_m, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_m - mean_movies_mvl)/(n()+l))

  predicted_ratings <-
    validation %>%
    left_join(b_m, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mean_movies_mvl + b_m + b_u) %>%
    pull(pred)

  return(RMSE(predicted_ratings, validation$rating))
})
#if I want to plot the results
#qplot(lambdas, rmse_lambda)
```

```
#to determine what lambda corresponds to the smallest RMSE
lambda<-lambdas[which.min(rmse_lambda)]

#RMSE with the selected lambda
b_u <- edx %>%
  left_join(b_m, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mean_movies_mvl - b_m)/(n()+lambda), n_m =n())

predicted_ratings <- validation %>%
  left_join(b_m, by='movieId') %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mean_movies_mvl + b_m + b_u) %>%
  .$pred

rmse_7 <- RMSE(predicted_ratings, validation$rating)

rmse_results <- data_frame(method = c("Just the average rating", "movie error", "user error",
                                      "year of release error", "year of rating error",
                                      "regularization - movieID", "regularization - movieID - userID"),
                           RMSE = c(ref_rmse, rmse_2, rmse_3, rmse_4, rmse_5, rmse_6, rmse_7))
format.data.frame(rmse_results, digit=4)
```

```
##                              method   RMSE
## 1           Just the average rating 1.0612
## 2                      movie error 0.9439
## 3                       user error 0.8653
## 4             year of release error 0.8650
## 5              year of rating error 0.8649
## 6          regularization - movieID 0.9439
## 7 regularization - movieID - userID 0.8648
```

This time, the regularization has proven to be useful.The RMSE has never been so small so far (rmse_7=0.8648).

I am going to continue trying to reduce it and include the year of release.

```
#model 8 - regularization ==> movies, users, and year of release
#determine the lambda
lambdas <- seq(0, 10, 0.25)

rmse_lambda <- sapply(lambdas, function(l){

  mean_movies_mvl<-mean(edx$rating)

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

  b_u <- edx %>%
    left_join(b_m, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_m - mean_movies_mvl)/(n()+l))
```

```
  b_y <- edx %>%
    left_join(b_m, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(year_release) %>%
    summarize(b_y = sum(rating - b_m - mean_movies_mvl - b_u)/(n()+l))

  predicted_ratings <-
    validation %>%
    left_join(b_m, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = "year_release") %>%
    mutate(pred = mean_movies_mvl + b_m + b_u + b_y) %>%
    pull(pred)

  return(RMSE(predicted_ratings, validation$rating))
})
#if I want to plot the results
#qplot(lambdas, rmse_lambda)

#to determine what lambda corresponds to the smallest RMSE
lambda<-lambdas[which.min(rmse_lambda)]

#RMSE with the selected lambda
b_y <- edx %>%
  left_join(b_m, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(year_release) %>%
  summarize(b_y = sum(rating - mean_movies_mvl - b_m - b_u)/(n()+lambda), n_m =n())

predicted_ratings <- validation %>%
  left_join(b_m, by="movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_y, by = "year_release") %>%
  mutate(pred = mean_movies_mvl + b_m + b_u + b_y) %>%
  .$pred

rmse_8 <- RMSE(predicted_ratings, validation$rating)

rmse_results <- data_frame(method = c("Just the average rating", "movie error", "user error",
                                      "year of release error", "year of rating error",
                                      "regularization - movieID", "regularization - movieID - userID",
                                      "regularization - movieID - userID - year of release"),
                           RMSE = c(ref_rmse, rmse_2, rmse_3, rmse_4, rmse_5, rmse_6, rmse_7, rmse_8))
format.data.frame(rmse_results, digit=4)
```

```
##                                                 method   RMSE
## 1                            Just the average rating 1.0612
## 2                                        movie error 0.9439
## 3                                         user error 0.8653
## 4                          year of release error 0.8650
## 5                           year of rating error 0.8649
## 6                        regularization - movieID 0.9439
## 7             regularization - movieID - userID 0.8648
```

```
## 8 regularization - movieID - userID - year of release 0.8645
```

Excellent, the RMSE continues decreasing (rmse_8=0.8645). Unfortunately, I've reached the limits of what the my R interface can support (i.e., too large vectors), thus I will not be able to include the variable year of rating to the previous model. Nonetheless, I am going to swap the variable "year of release" by "year of rating" to verify that I have the best RMSE possible.

```
#model 9 - regularization ==> movies, users, and year of rating
#determine the lambda
lambdas <- seq(0, 10, 0.25)

rmse_lambda <- sapply(lambdas, function(l){

  mean_movies_mvl<-mean(edx$rating)

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

  b_u <- edx %>%
    left_join(b_m, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_m - mean_movies_mvl)/(n()+l))

  b_yr <- edx %>%
    left_join(b_m, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(year_2) %>%
    summarize(b_yr = sum(rating - b_m - mean_movies_mvl - b_u)/(n()+l))


  predicted_ratings <-
    validation %>%
    left_join(b_m, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_yr, by = "year_2") %>%
    mutate(pred = mean_movies_mvl + b_m + b_u + b_yr) %>%
    pull(pred)

  return(RMSE(predicted_ratings, validation$rating))
})
#if I want to plot the results
#qplot(lambdas, rmse_lambda)

#to determine what lambda corresponds to the smallest RMSE
lambda<-lambdas[which.min(rmse_lambda)]

#RMSE with the selected lambda
b_yr <- edx %>%
  left_join(b_m, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(year_2) %>%
  summarize(b_yr = sum(rating - mean_movies_mvl - b_m - b_u)/(n()+lambda), n_m =n())
```

```r
predicted_ratings <- validation %>%
  left_join(b_m, by="movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_yr, by = "year_2") %>%
  mutate(pred = mean_movies_mvl + b_m + b_u + b_yr) %>%
  .$pred

rmse_9 <- RMSE(predicted_ratings, validation$rating)


rmse_results <- data_frame(method = c("Just the average rating", "movie error", "user error",
                                      "year of release error", "year of rating error",
                                      "regularization - movieID", "regularization - movieID - userID",
                                      "regularization - movieID - userID - year of release",
                                      "regularization - movieID - userID - year of rating"),
                           RMSE = c(ref_rmse, rmse_2, rmse_3, rmse_4, rmse_5, rmse_6, rmse_7,
                                    rmse_8, rmse_9))
format.data.frame(rmse_results, digit=4)
```

```
##                                                method   RMSE
## 1                            Just the average rating 1.0612
## 2                                        movie error 0.9439
## 3                                         user error 0.8653
## 4                                year of release error 0.8650
## 5                                 year of rating error 0.8649
## 6                              regularization - movieID 0.9439
## 7                  regularization - movieID - userID 0.8648
## 8 regularization - movieID - userID - year of release 0.8645
## 9  regularization - movieID - userID - year of rating 0.8648
```

The RMSE increased showing that "year of release" is a better predictor than "year of rating".

4. Conclusion

In the data exploration step, I did put in evidence several potential predictors of movie ratings and determinate that the use of regularization might be beneficial to my recommendation system. When implementing these potential predictors to the recommendation system, it appeared that the movie and user biases (i.e., effect [1]) have the most important effects on the RMSE. The regularization has also been useful, especially for the variable "userId". The lowest RMSE obtained was 0.8645 using regularization over three predictors: "movieId", "userId", and "year of release". Considering that the variable "genres" has not been included in the recommendation system (even though potentially useful), further studies might consider using it. Going even further, using matrix factorization might very useful to consider movies that are rated in the same manner.

5. References

[1] https://rafalab.github.io/dsbook/
[2] https://towardsdatascience.com/
[3] https://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf
[4] https://grouplens.org/datasets/movielens/