Movielens Capstone Project

Vincenzo Grifo

31/05/2020

INTRODUCTION

Goal of the project

The goal of this project was to to create a movie recommendation system in R, using the 10M version of the MovieLens dataset.

Dataset description

This dataset was a collection of 10 million movie ratings given by users on the the MovieLens web site (movielens.umn.edu), during the seven-month period from 19th September 1997 through 22nd April 1998. These data were collected by the GroupLens Research Project at the University of Minnesota. The dataset consisted of 10,000 observations (rows) where each row represented a rating given by one user to one movie. More specifically, the dataset included information about 6 variables (columns):

- **userId**, unique numeric code that identified a singular user. One user was associated only to one userID;
- **movieId**, unique numeric code that identified a singular movie. One movie was associated only to one userID;
- **timestamp**, an integer that represented date and time when the rating was given. This was measured as seconds since midnight (UTC) of January 1, 1970
- **title**, character string that represented the movie title. It also included information of the year of release;
- **genres**, character string that listed all the genres associated with the movies;
- **rating**, a numeric value between 0 and 5 that represented the rating for the movie given by the user.

Key steps to build the movie recommendation system

To build the movie recommendation system a machine learning algorithm was implemented following 5 key steps. First, the data were retrieved by downloading and pre-processing the database from the grouplens.org website. Then, these data were explored through visualisation and summarisation techniques to understand which features could have had predictive power and, therefore, should have been considered in the algorithm. After that, the dataset have been processed once again to include the new features that had been engineered and selected during the data exploration and visualisation stages. During this stage, the training and test datasets that would have been used in the next step were created. In the next phase, different machine learning models have been implemented and compared - the model with the lowest value of the chosen loss function (i.e. RMSE) was selected. Finally, the accuracy of the final model was validated by using the given validation dataset.

ANALYSIS

Retrieving the data and preliminary data cleaning and pre-processing

The first step in building the recommendation system consisted in retrieving the dataset *movielens* from the grouplens.org website. After having loaded the required packages (i.e. *tidyverse*, *caret* and *data.table*), imported and converted the data from a raw to a tidy format, the dataset was split into 2 subsets: * the *edx* dataset (90% of the initial dataset), that was used to train and test the model * the *validation* dataset (10% of the initial dataset), that was instead only used in the last stage of the process to calculate the final value of the RMSE of the model.

Data Exploration

The newly created working dataset *edx* was used for the exploratory data analysis. This stage consisted of the initial investigation on the data so as to discover relationships and attributes present in the dataset, to formulate and check any assumptions, with the help of summary statistics and graphical representations. This was a key process as it helped gain precious insights on the predictive power of the variables and therefore enabled the formulation of the models that were built and compared in the subsequent stages.

In order to get a first grasp of the data, the initial investigations involved the analysis of the structure of the dataset and the visualisation of the first few rows.

```
# Analysing the structure of the dataset edx
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
```

```
# Having a look at the first lines of the dataset
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
```

This few simple lines of codes helped identify the initial features of the model (i.e. userId, movieId, title, genres, timestamp), the outcome variable (i.e. rating) and get some useful insights on their characteristics, for example:

- the dates of the ratings were in seconds, hence in future coding these would have need to be converted in a different format (e.g. date)
- the feature *genres* included all the possible genres associated to the movie and these separated by the special character /, meaning that a 'Comedy' and a 'Comedy|Romance' were two different genres despite being both a 'Comedy'.

Having identified the features, these became the focus of the data exploration analysis.

UserId and Movie feature exploration

The first two column of the *edx* dataset were *userId* and *movieid*, which represented the user who has given the rating and the rated movie respectively.

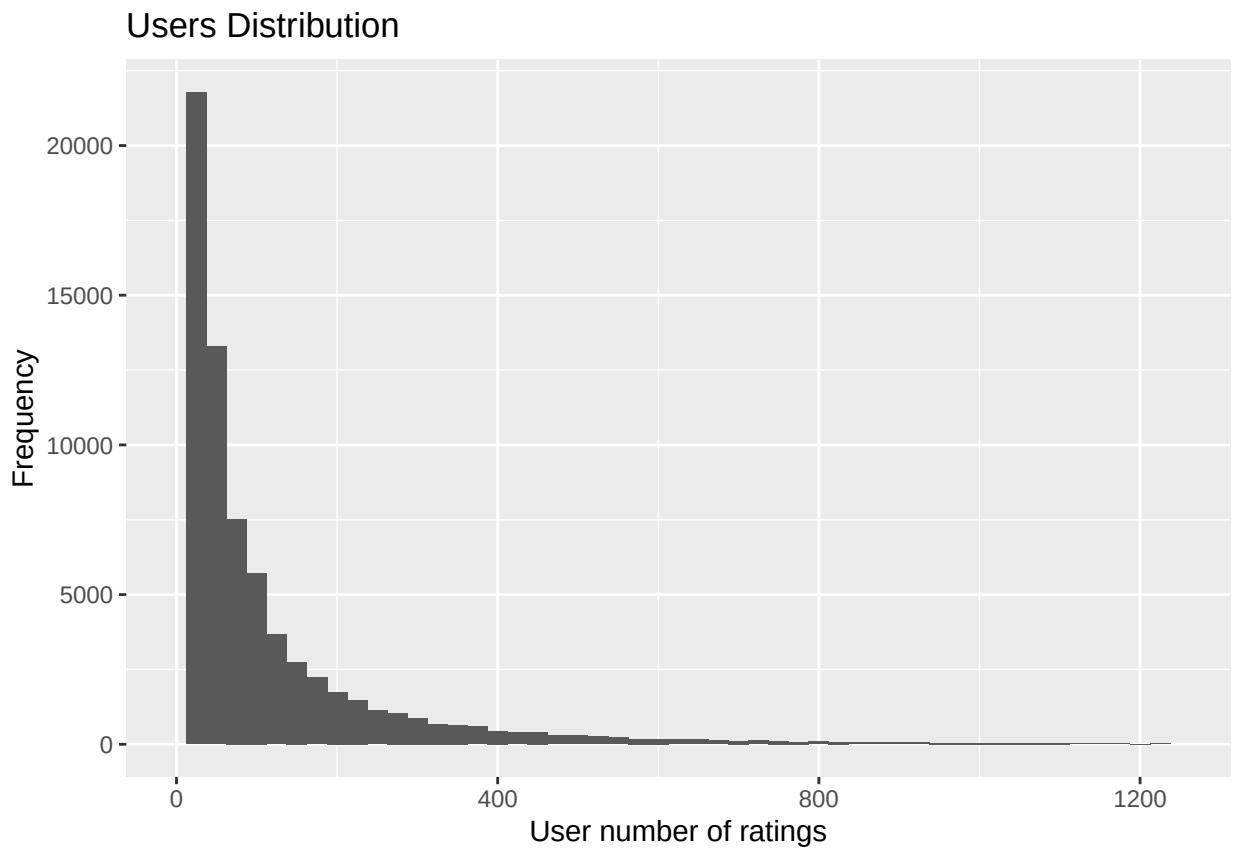
In the *edx* dataset there were 69878 users that have given a rating and 10677 movies that have been rated. This means that not all the users have rated all the movies in the dataset, otherwise the dataset would have had 69878 x 10677 = 746087406 number of rows instead of 9000055.

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

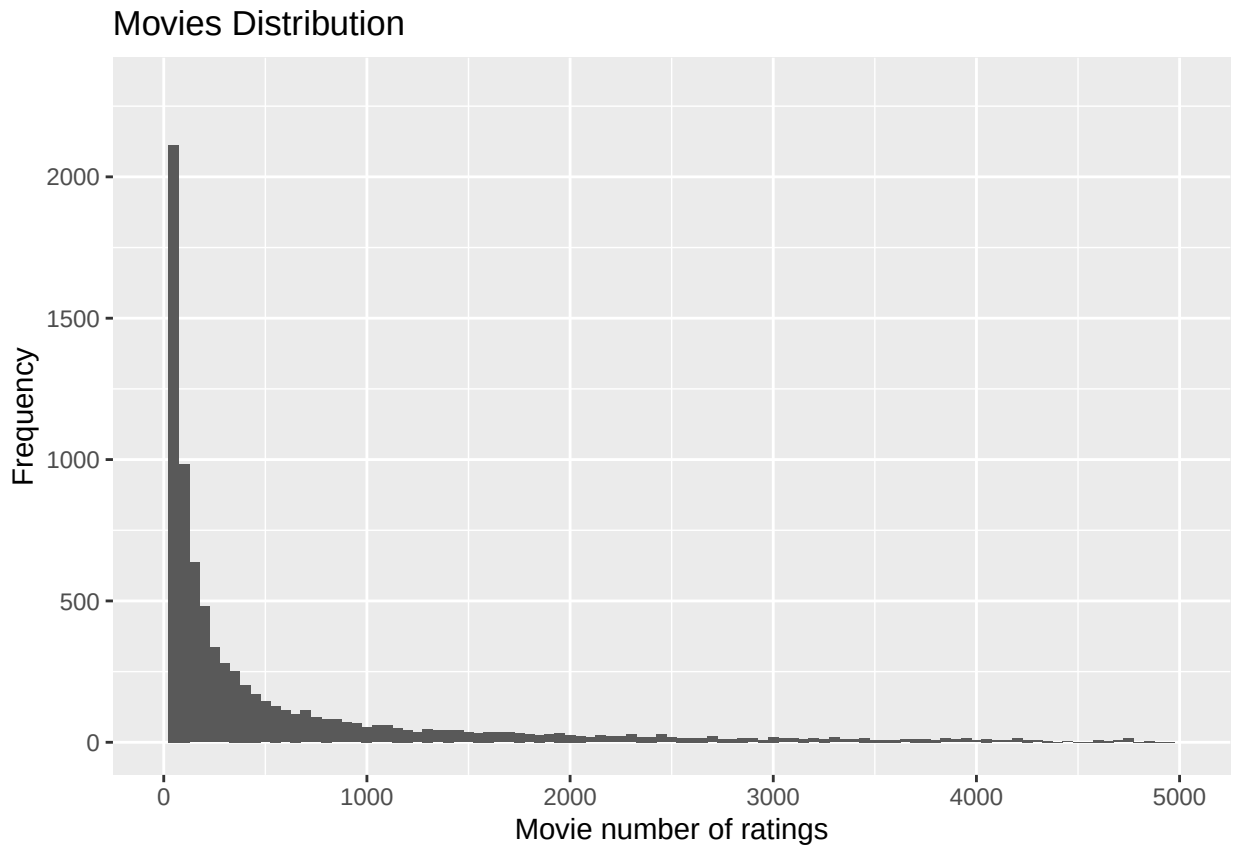
```
##   n_users n_movies
## 1   69878    10677
```

To understand how the ratings were distributed across users and movies the distribution of frequencies for userId and movieId were plotted.

```
# Plotting the userId distribution
edx %>%
  group_by(userId) %>%
  summarize(n = n() ) %>%
  ggplot(aes(x=n)) +
  geom_histogram(binwidth = 25) +
  xlim(0,1250) +
  ggtitle("Users Distribution") +
  xlab("User number of ratings") +
  ylab("Frequency")
```



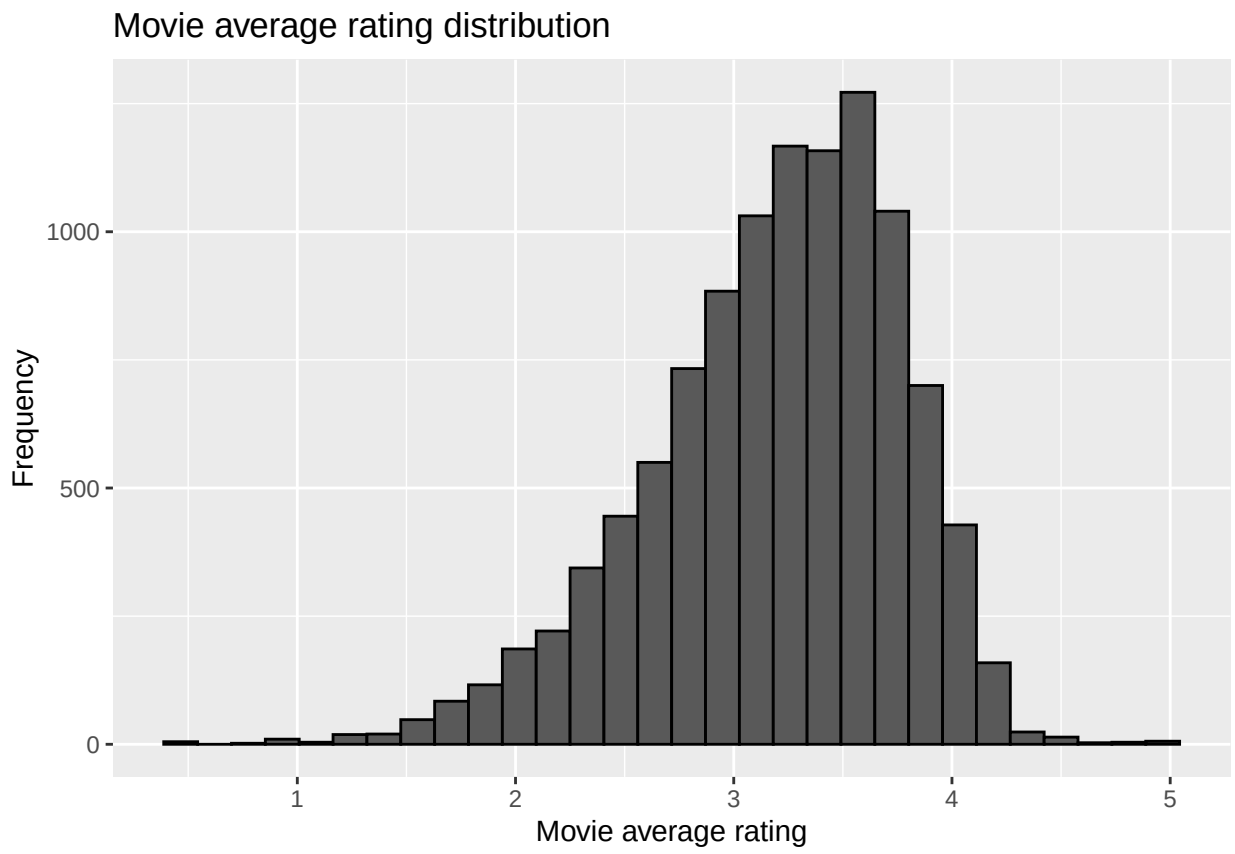

```
# Movies distribution
edx %>%
  group_by(movieId) %>%
  summarize(n = n() ) %>%
  ggplot(aes(x=n)) +
  geom_histogram(binwidth = 50) +
  xlim(0,5000) +
  ggtitle("Movies Distribution") +
  xlab("Movie number of ratings") +
  ylab("Frequency")
```



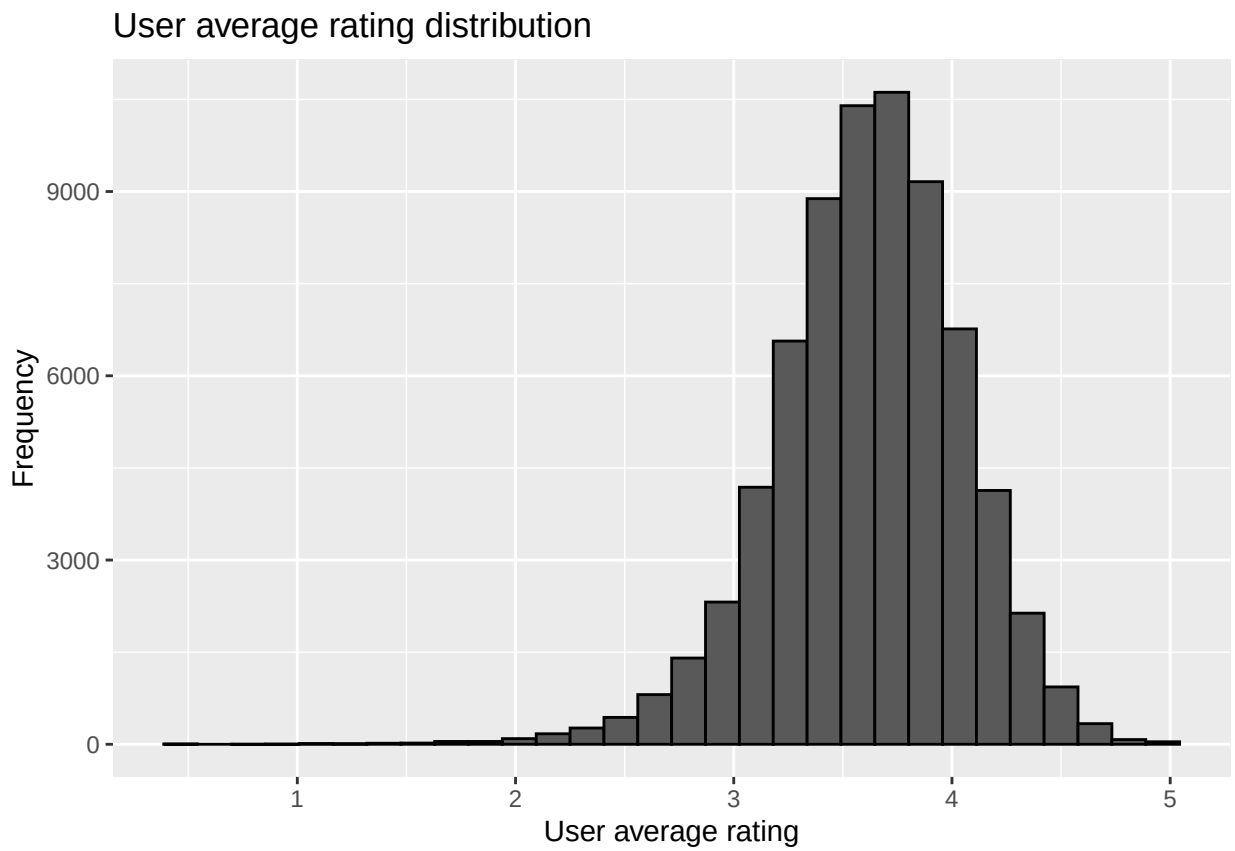
From the distribution charts we got some valuable insights: * Some movies got rated significantly more often than others. This made sense as there are movies that generally are highly distributed and appeal a vast audience (e.g. blockbusters), while there are also indie and more niche movies that are usually catered for a limited audience. * Some users gave remarkably more ratings than others. This is natural as some users tend to be more avid raters than others.

To understand how the ratings were distributed across the movies and users, the distribution of frequencies for the average rating per userId and per movieId were plotted.

```
# Movie average rating distribution
edx %>%
  group_by(movieId) %>%
  summarize(avg_rating = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(avg_rating)) +
  geom_histogram(bins = 30, color = "black") +
  ggtitle("Movie average rating distribution") +
  xlab("Movie average rating") +
  ylab("Frequency")
```




```
# User average rating distribution
edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black") +
  ggtitle("User average rating distribution") +
  xlab("User average rating") +
  ylab("Frequency")
```



From these it was noticed a significant variability of average rating across movies and users. 1. Most of the movies had an average score of around 3.5 stars. Just few movies had on average a very high rating, fore example 4.5 stars or more and there were only few movies with a poor rating below 1.5 stars. However, from the chart was clear that some movies appeared to be on average better than others. 2. Similarly to the movie average rating distribution, most of the users had an average rating of around 3.5-3.7 stars, although there were users who clearly gave more generous rating than others and the distribution appeared more skewed to the right.

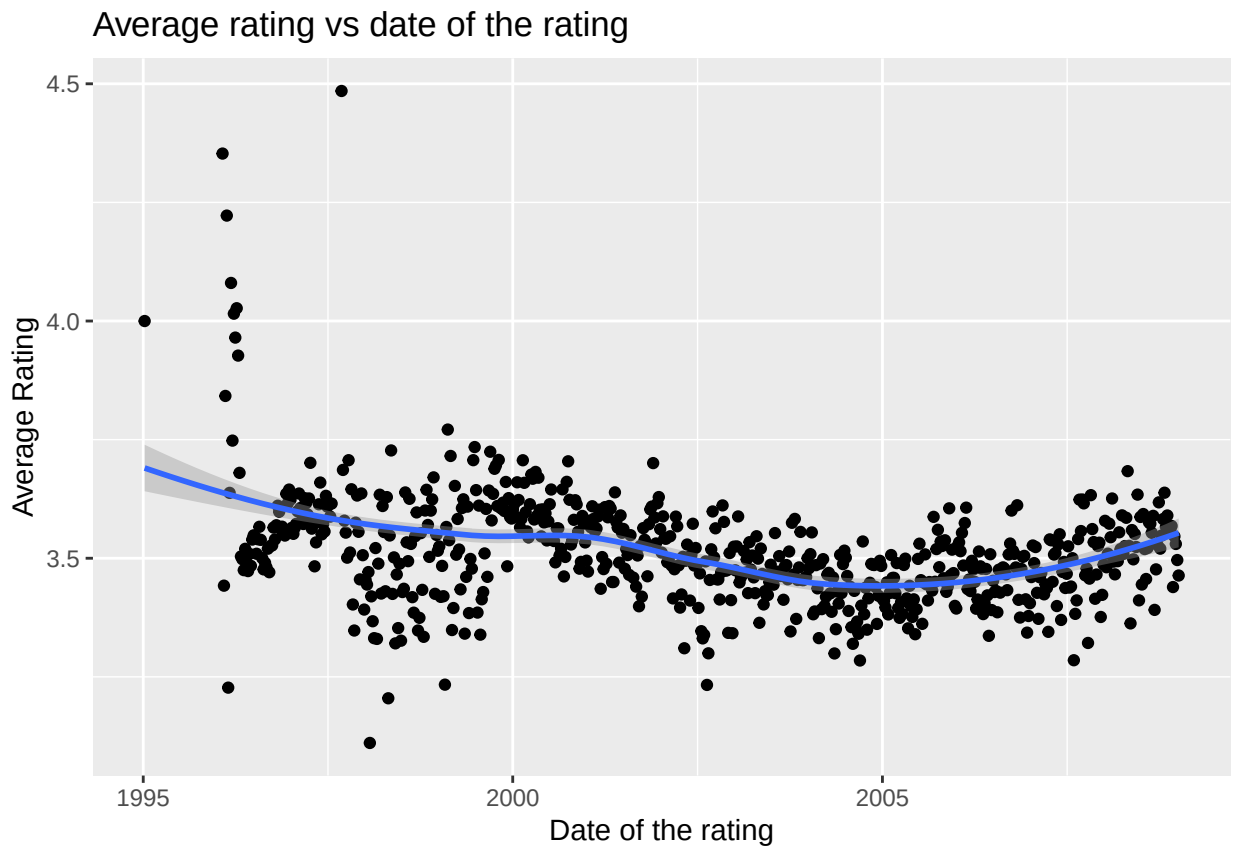
The understanding of these two distributions played a pivotal role in the development of the machine learning algorithm, because these gave evidence of the existence of **user** and **movie effects** which were the base of the machine learning algorithm - this will be explained in more details in the *Analysis* section.

Timestamp feature exploration

The third variable that was explored was *timestamp* to understand whether the date when the rating was given could have had an influence on the average rating. For this reason, a scatter plot of the average rating against the date of the rating was made.

```
edx %>% mutate(date = as_datetime(timestamp)) %>%
  mutate(date = round_date(date, unit = "week")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth(formula = y ~ x, method = 'loess') +
  ggtitle("Average rating vs date of the rating") +
```

```
  xlab("Date of the rating") +
  ylab("Average Rating")
```



From the plot it appeared that the average rating was somewhat influenced by a *time effect*, although not that strongly. It was therefore thought that the time when the rating was given could have hold some predictive power, however it was necessary some *feature engineering* in order to identify exactly which aspects of the time could affect the rating.

Specifically, two new feature were considered and explored:

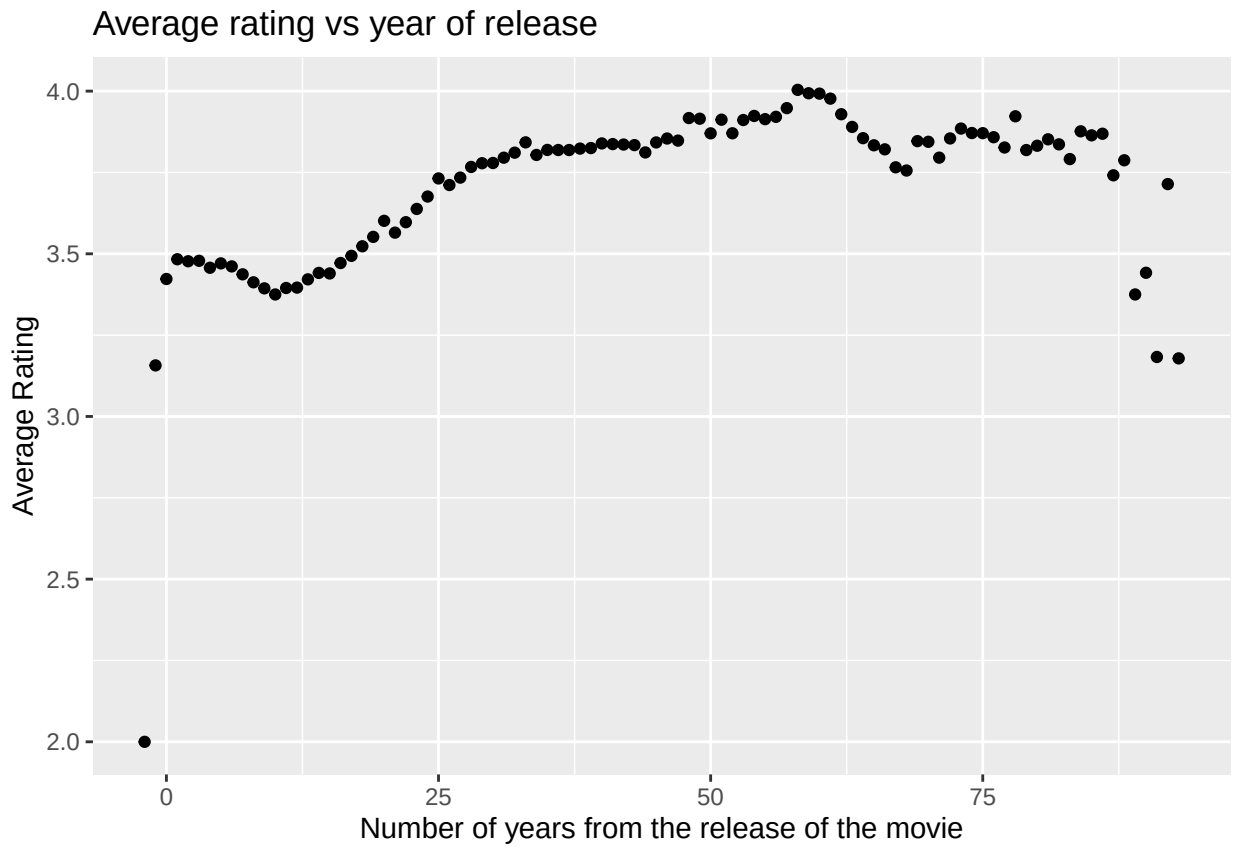
1. Time elapsed between the release of the movie and the rate given by the user
2. Time elapsed between the rate and the first rating given by the user

Feature engineering and exploration - 'Years from movie's release'

To understand whether and how a rating can be influenced by the amount of time elapsed since the movie's release, the average movie rating was plotted against the number of year's since the movie's release.

```
edx %>% mutate(date = as_datetime(timestamp)) %>%
        # extract year when the movie was filmed
        mutate(year_release = str_extract(title, "(?<=\\()\\d{4}+(?=\\))")) %>%
        # calculate time elapsed between review and movie's release
        mutate(years_from_release = as.numeric(year(date)-as.numeric(year_release))) %>%
        group_by(years_from_release) %>%
        summarize(rating = mean(rating)) %>%
```

```
        ggplot(aes(years_from_release, rating)) +
        geom_point() +
        ggtitle("Average rating vs year of release") +
        xlab("Number of years from the release of the movie") +
        ylab("Average Rating")
```

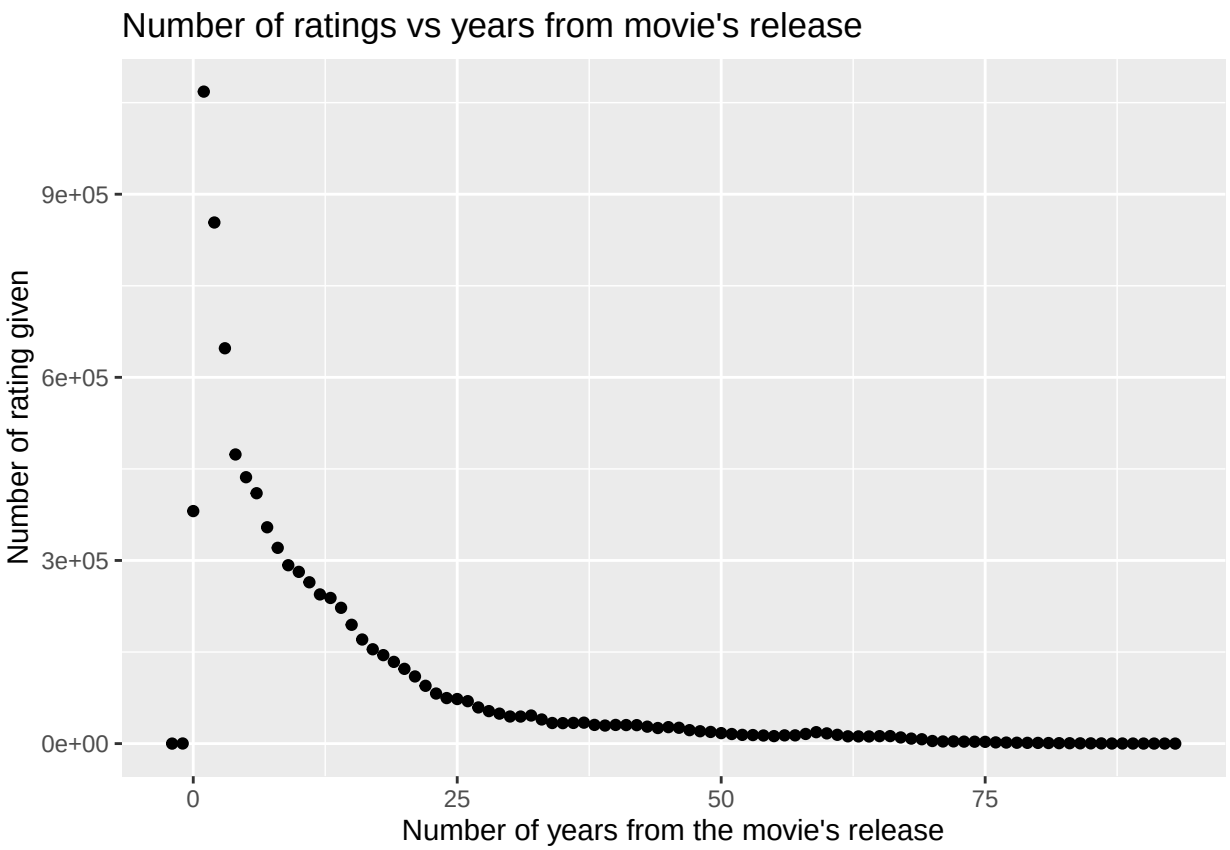


It appeared that the average rating was higher when the movies were rated within the first few years from their release and then dropped and reaches a minimum at around 10 years from the release. However, after that the average rating grew considerably and stabilized at around 3.7 - 3.8 stars. Intuitively, we can think that users might get quite excited for a movie that has just been released, maybe due to the media hype, and this might initially inflate the ratings. A few years after the release, if the movie is actually not great, users (having lost the initial excitement) start being less generous with their ratings. After around 10 years from the release, if a movie is actually good this could start to become a 'classic' and users would rate rate this accordingly. Conversely, if a movie is actually not great, after a few years from its release this might disappear from the market and not being rated any longer, thus not affecting the average movie rating of the classic old movies.

To test this last assumption, the distribution of ratings given across the time elapsed between the movie's release and the rating was plotted.

```
edx %>%
  mutate(date = as_datetime(timestamp),
         year_release = str_extract(title, "(?<=\\()\\d{4}+(?=\\))")) %>%
  mutate(years_from_release = as.numeric(year(date)-as.numeric(year_release))) %>%
  group_by(years_from_release) %>%
  summarize(n_rates = n()) %>%
```

```
  ggplot(aes(years_from_release, n_rates)) +
  geom_point() +
  ggtitle("Number of ratings vs years from movie's release") +
  xlab("Number of years from the movie's release") +
  ylab("Number of rating given")
```



It could be seen clearly that the majority of ratings happened for movie that had just been released recently and the distribution dropped significantly for movies that were at least 25 year's old when rated. As seen from the previous chart, these movies had extremely high average rating, therefore these could only have be classic movies which in general tend to survive the judgement of time. To confirm this hypothesis, the top 20 movies with most rating and that were rated after 25 years of their release were extracted and the results confirmed that those belonged to the category of classic movies (e.g. "Snow White and the Seven Dwarfs" and "The Godfather").

```
edx %>%
  mutate(date = as_datetime(timestamp),
         year_release = str_extract(title, "(?<=\\()\\d{4}+(?=\\))")) %>%
  mutate(years_from_release = as.numeric(year(date)-as.numeric(year_release))) %>%
  group_by(movieId, title,years_from_release) %>%
  summarize(n_rates = n()) %>%
  filter(years_from_release >25) %>%
  ungroup() %>%
  group_by(movieId, title) %>%
  arrange(desc(n_rates)) %>%
  head(20)
```

```
## # A tibble: 20 x 4
## # Groups:   movieId, title [15]
##    movieId title                                        years_from_relea~ n_rates
##      <dbl> <chr>                                                  <dbl>   <int>
##  1    1073 Willy Wonka & the Chocolate Factory (1971)                26    3434
##  2     594 Snow White and the Seven Dwarfs (1937)                    59    2814
##  3     858 Godfather, The (1972)                                     28    2616
##  4     260 Star Wars: Episode IV - A New Hope (a.k.a.~               28    2344
##  5     919 Wizard of Oz, The (1939)                                  61    2123
##  6     912 Casablanca (1942)                                         58    2056
##  7    1221 Godfather: Part II, The (1974)                            26    1985
##  8     924 2001: A Space Odyssey (1968)                              32    1983
##  9     858 Godfather, The (1972)                                     27    1945
## 10    1136 Monty Python and the Holy Grail (1975)                    30    1876
## 11     858 Godfather, The (1972)                                     33    1773
## 12     260 Star Wars: Episode IV - A New Hope (a.k.a.~               31    1771
## 13     596 Pinocchio (1940)                                          56    1702
## 14    1198 Raiders of the Lost Ark (Indiana Jones and~               27    1690
## 15     858 Godfather, The (1972)                                     36    1679
## 16    1196 Star Wars: Episode V - The Empire Strikes ~               28    1629
## 17     750 Dr. Strangelove or: How I Learned to Stop ~               36    1627
## 18    1304 Butch Cassidy and the Sundance Kid (1969)                 31    1623
## 19    1214 Alien (1979)                                             26    1586
## 20     260 Star Wars: Episode IV - A New Hope (a.k.a.~               29    1584
```

The result from this exploratory analysis gave confidence that a machine learning model that includes the feature engineered *Years from movie's release* should have been considered and tested.

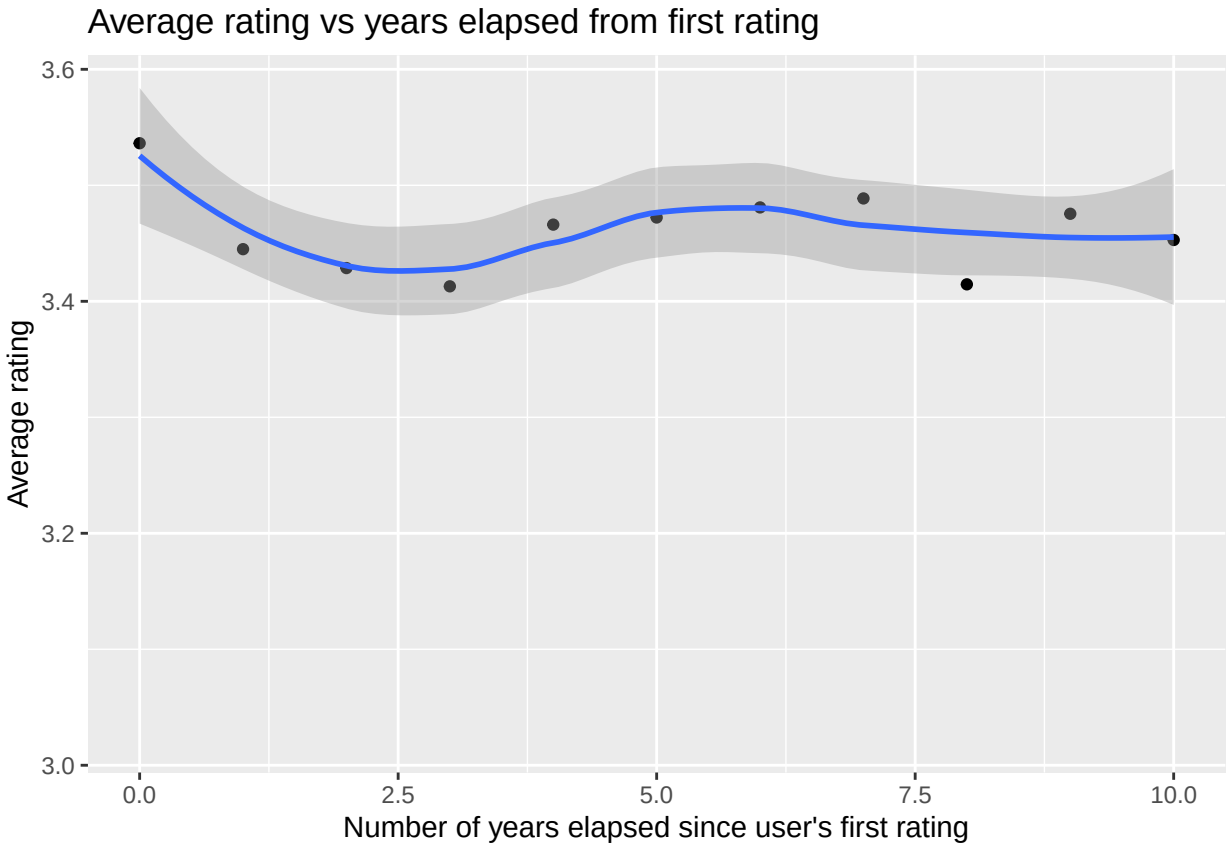
Feature engineering and exploration - 'Years from user's first rating'

Having the exact date and time of all the users' ratings, allowed to identify for each user the *number of years since the user's first rating* and therefore engineer and explore this new feature. The idea behind this feature engineering was simple: intuitively, users might become harsher critics over time because the more movies a user rates the more ratings the user could implicity or explicity refer to as a benchmark in future ratings.

The plot below, which shows the average rating given the number of years since the user's first rating, suggested that the user ratings tended to be higher during the first year of the user's rating experience. The average rating then dropped significantly during years 1 to 3 before recovering and finally stabilising in following years. In order to include this effect, the feature engineered *number of years since user's first review* was included in some of the machine learning algorithms that were developed afterwards.

```
# Function to calculate how many months have elapsed between 2 dates
elapsed_months <- function(end_date, start_date) {
  ed <- as.POSIXlt(end_date)
  sd <- as.POSIXlt(start_date)
  12 * (ed$year - sd$year) + (ed$mon - sd$mon)
}

#Plotting the average rating vs number of years elapsend since the user's first rating
edx %>%
  group_by(userId) %>%
  mutate(date = as_datetime(timestamp), date_first_rate=min(date)) %>%
  ungroup() %>%
  mutate(years_elapsed_first_review = ceiling(elapsed_months(date, date_first_rate)/12)) %>%
```

```
  group_by(years_elapsed_first_review) %>%
  summarise(avg_rate = mean(rating)) %>%
  ggplot(aes(years_elapsed_first_review, avg_rate )) +
  geom_point() +
  geom_smooth(formula = y ~ x, method = 'loess') +
  ggtitle("Average rating vs years elapsed from first rating") +
  xlab("Number of years elapsed since user's first rating") +
  ylab("Average rating") +
  xlim(0,10)
```



Genres feature exploration

The last variable from the *edx* dataset that was explored was *genres*. At the very beginning of the data exploration analysis, it was noted that the column *genres* included all the genres associated with the movies separated by the special character /, meaning that each combination of genres would have been treated as one single genre itself (i.e. Action|Adventure|Sci-Fi would have been a completely distinct genre from Action|Adventure|Drama|Sci-Fi). This also meant that within the dataset there were a considerable number of genres - 797 to be precise.

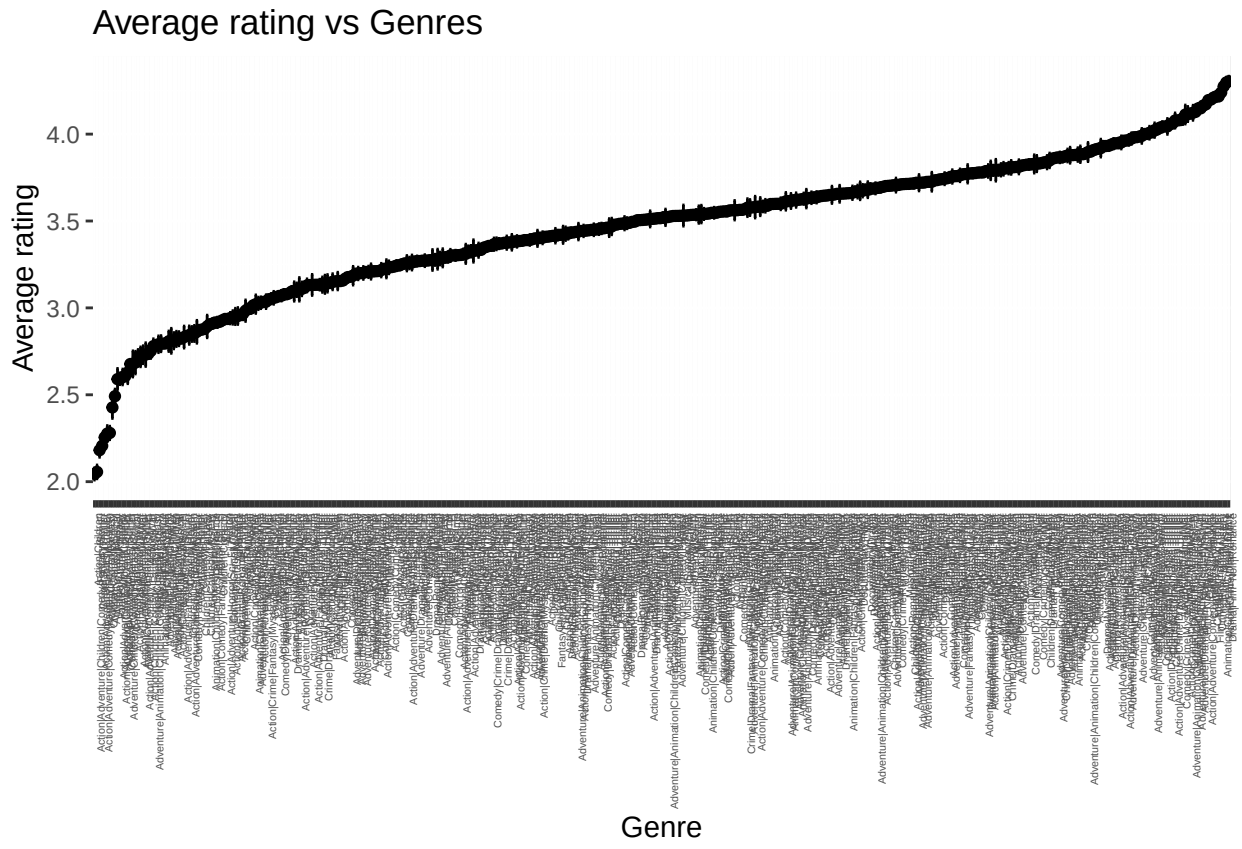
```
# Determining how many genres are included in the dataset
length(unique(edx$genres))
```

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

Compared to the features analysed in the previous sections, *genres* was a categorical variable, which meant that could only take on one of a limited number of possible values (797 as previously mentioned). To understand whether this variable could actually influence the rating, a bar plot of the average and standard error of each genre was generated.

Even though the cluttered labels in the x-axis were harder to read, the shape of the curve indicated the possible presence of a *genre* effect. This can be intuitively explain if we think that certain genres, like comedies, usually are liked by a wider audience than a genre like horror movies. Therefore, this feature was considered during the implementation and evaluation of the machine learning algorithms.

```
edx %>%
  group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 1000) %>%
  # Visualising only only genres with a minimum of 1000 reviews
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(size =4,angle = 90, hjust = 1)) +
  ggtitle("Average rating vs Genres") +
  xlab("Genre") +
  ylab("Average rating")
```



Key insights and modelling approach

The key insights that were gained from the data exploration analysis could be summarised as follow:

- Rating could be influenced by the specific **user** who gives the rating and by the **movie** itself that was rated
- The time when the rating was given could also influence a rating. Specifically, ratings given in the same year of the **movie's release** tend to be on average inflated. In addition, *classic* movies keep receive a good amount of ratings overtime, which are usually higher than the average.
- The time elapsed **since the user's first** rating seemed to play a role in the rating process. Ratings appear to be inflated during the first few years of a user's rating experience.
- Finally, movies belonging to a certain **genre** are usually rated higher than others. Basically, the data exploration analysis provided the indications for the set of predictors to consider for the machine learning algorithm.

The modelling approach was to think of the rating as it was composed of several parts - a baseline rating which then got adjusted by the action of the effects highlighted before. In essence:

baseline average rating + movie effect + user effect + years since movie's release effect + years since user's first review effect + genre effect

The idea was not to predict the rating itself, but rather predicting the value that the effects add/remove to the selected baseline rating (in this case the average rating of all movies). In order to understand which effect actually contributed to the right prediction of the rating, more models were studied. Starting from the simplest model (baseline rating), each effect was added one by one. If the accuracy of the model improved, then the predictor was permanently considered within the model.

Further data pre-processing

The data exploration analysis identified some new features that could be engineered from the original dataset that could have predictive power and could therefore be used in the machine learning models. These new feature where number of *Years from movie's release* and number of *Years from user's first rating*. Before starting the development of the machine learning models, the original dataset *edx* was once again processed in order to include the new features that were going to be used systematically during the creation and testing of the algorithms. Calculating and storing the new variables in this stage avoided to repeat the same calculation over and over again, thus making the code sleeker and saving precious computational time.

Further data pre-processing also included the creation of two complementary datasets from the *edx* dataset - the *training* and *test* sets. The former was the subset that was going to be used to train the algorithms (tuning the models' parameters), while the latter was the subset that was going to be used to measure the accuracy of the models and therefore select the final machine learning model.

When slitting the training and test sets, the proportion of the data split is usually arbitrary. In most cases, the larger the *training set* the better, because adding more examples in general add diversity within the data, thus improving the fit and generalisation of the model. However, at the same time, the *test set* in order to to yield statistically meaningful results must be large enough and representative of the *edx* data set as a whole. For this reason the *edx* dataset was split randomly with a 95-5 train-test ratio. As the *edx* dataset contained a vast amount of observations (9000055), picking 5% of observations from it allowed the creation of a subset with 450001 observations, which should be sufficient. To make sure that users and movies in the test set that did not appear in the training set were not included in the test set, these entries were removed by using the semi_join function.

```
# Splitting the edx dataset into TEST and TRAINING set
# Setting the seed to enable the reproducibility of results
set.seed(17, sample.kind="Rounding")
```

```r
# The test set will be 5% of the training set
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.05,
                                  list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

# Making sure userId and movieId in test set are also in train set
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Adding rows removed from test set back into train set
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)

# Removing the temporary tables
rm(test_index, temp, removed)
```

Building the machine learning models

Defining the loss function - RMSE

After having prepared the *training* and *test* set, the first step toward the development of the machine learning model was choosing the appropriate *loss function*. The project rubric clearly stated that the loss function to be used was the *residual mean squared error* (**RMSE**).

In the case of the movie recommendation system, defined $y_{u,i}$ as the rating for movie i given by user u and $\hat{y}_{u,i}$ as the predicted rating for movie i by user u, the RMSE is defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y_{ui}} - y_{ui})^2}$$

where N is the total number of observations.

```r
# Defining the Loss Function
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

The RMSE represents the error that could occur when making a prediction. It has a meaning similar to the standard deviation - the larger the RMSE the larger will be on average how the predictions will deviate from the real rating. If the RMSE is, for example, equal to 1.5, the predicted ratings will be on average 1.5 stars higher or lower than the real ratings.

First model: baseline rating of mean over all user-movie ratings

The first model that was built consisted of the simple baseline prediction of assigning the same rating to all movies. The rating assigned was the average of all user-movie ratings. From a mathematical perspective, the model looked like this:

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

where where the variance of the rating Y given by user u for movie i from the *true* rating for all movies is caused by random variability, represented by the independent error $\epsilon_{u,i}$.

In this specific case, the RMSE is minimised by the least square estimate of the average of all user-movie ratings. Therefore, the predicted rating was the simply the average of all user-movie ratings. The RMSE for this model was .

```
# Average of all ratings
mu_hat <- mean(train_set$rating)

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

```
## [1] 1.061957
```

Second model: introduction of the *movie effect*

In the exploratory analysis stage, it was introduced the idea that a movie, for example The Godfather, could simply be better than an average movie, and therefore be rated on average above the average user-movie rating. This deviation from the average rating due exclusively to the value of the movie itself could be interpreted as a *movie effect*. Therefore, the baseline prediction model could be improved by adding the parameter b_i to represent the *movie effect*, basically the number of stars above or below the average given by the *quality* of the movie i itself.

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

While the model above could have been fitted by using the code below:

```
fit <- train_set %>% lm(rating ~ as.factor(movieId))
```

Due to the required computational resources needed to run this code, the b_i were instead estimated as the average of the variation of the rating $Y_{u,i}$ from the average of all ratings $\hat{\mu}$:

$$\hat{b_i} = Y_{u,i} - \hat{\mu}$$

```
# Calculating the least square estimates b_i

mu <- mean(train_set$rating) #Average of all ratings

movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# Making predictions
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

# RMSE calculation
movie_effect_rmse <- RMSE(predicted_ratings, test_set$rating)
```

```
# Adding this RMSE to the table where we are storing all RMSE previously calculated
rmse_results <- tibble(method = c("Just the average","Movie Effect"),
                       RMSE = c(naive_rmse,movie_effect_rmse))
rmse_results
```

```
## # A tibble: 2 x 2
##   method            RMSE
##   <chr>            <dbl>
## 1 Just the average 1.06
## 2 Movie Effect     0.945
```

The introduction of the *movie effect* improved the RMSE by 0.117.

Third model: introduction of the *user effect*

In the exploratory analysis it was also noted that a user, for example Vinnie, could be extremely stingy with his ratings, thus rating a movie 0.2 stars lower than the average. This deviation from the average rating due exclusively to the attitude of the user towards rating could be interpreted as the *user effect*. This meant that, similarly to the case of the *movie effect*, the model could be augmented:

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

where the parameter b_u represented the *user effect*.

As mentioned earlier, the *lm()* function could have been used to fit the model, however b_u were estimated following the same approach seen in the case of the *movie effect*.

$$\hat{b_u} = Y_{u,i} - \hat{\mu} - \hat{b_i}$$

```
# Estimating b_u
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# Calculating predictions
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Calculating RMSE
user_effect_rmse <- RMSE(predicted_ratings, test_set$rating)

# Adding this RMSE to the table where we are storing all RMSE previously calculated
rmse_results <- tibble(method = c("Just the average","Avg + Movie Effect",
                                  "Avg + Movie Eff + User Eff"),
                       RMSE = c(naive_rmse,movie_effect_rmse, user_effect_rmse))
rmse_results
```

```
## # A tibble: 3 x 2
##   method                      RMSE
##   <chr>                      <dbl>
## 1 Just the average           1.06
## 2 Avg + Movie Effect         0.945
## 3 Avg + Movie Eff + User Eff 0.866
```

The addition of the *user effect* improved the RMSE by 0.079.

Fourth model: using *regularised movie effects*

To further improve the RMSE, the process of *regularisation* was applied on the least square estimates b_i. The idea behind the regularisation is to constrain the total variability of the effect sizes by penalizing large estimates that come from small sample sizes.

Before applying the regularisation on the movie effects, some of the predictions were explored in order to understand how the performance of algorithm was affected by large estimates that came from small sample sizes. Specifically, looking at the 10 best and worst rated movies according to the model, it could be observed that those movies were not famous at all.

```
# Worst 10 movies according to the predictions
movie_avgs %>% left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  slice(1:10)  %>%
  pull(title)
```

```
##  [1] "Besotted (2001)"
##  [2] "Hi-Line, The (1999)"
##  [3] "Accused (Anklaget) (2005)"
##  [4] "Confessions of a Superhero (2007)"
##  [5] "War of the Worlds 2: The Next Wave (2008)"
##  [6] "SuperBabies: Baby Geniuses 2 (2004)"
##  [7] "Hip Hop Witch, Da (2000)"
##  [8] "Disaster Movie (2008)"
##  [9] "From Justin to Kelly (2003)"
## [10] "Criminals (1996)"
```

```
# Best 10 movies according to the predictions
movie_avgs %>% left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  slice(1:10)  %>%
  pull(title)
```

```
##  [1] "Hellhounds on My Trail (1999)"
##  [2] "Satan's Tango (Sátántangó) (1994)"
##  [3] "Shadows of Forgotten Ancestors (1964)"
##  [4] "Fighting Elegy (Kenka erejii) (1966)"
##  [5] "Sun Alley (Sonnenallee) (1999)"
##  [6] "Blue Light, The (Das Blaue Licht) (1932)"
##  [7] "Hospital (1970)"
##  [8] "Human Condition II, The (Ningen no joken II) (1959)"
##  [9] "Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)"
## [10] "Human Condition III, The (Ningen no joken III) (1961)"
```

Next, it was calculated how many times these top 10 best and worst movies were rated to confirm the assumptions that those were niche movies rated only few times. The result showed that all of these movies were rated less than 5 times in total. Therefore, to improve the predictions it was decided to penalise these large estimates coming from sample sizes through the regularisation process.

```
# Worst 10 movies number of ratings
train_set %>% count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  slice(1:10) %>%
  pull(n)
```

```
## Joining, by = "movieId"
```

```
##  [1]   2   1   1   1   2  54  14  29 186   2
```

```
# Best 10 movies number of ratings
train_set %>% count(movieId) %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  slice(1:10) %>%
  pull(n)
```

```
##  [1] 1 2 1 1 1 1 1 3 4 4
```

The regularisation on the movie effects was implemented by calculating the b_i that minimised the following equation:

$$\frac{1}{N}\sum_{u,i}(y_{ui}-\mu-b_i)^2+\lambda\sum_i b_i^2$$

where $\lambda\sum_i b_i^2$ was the penalty term.

The value of b_i that minimised the equation can be calculated as:

$$\hat{b}_i(\lambda)=\frac{1}{\lambda+n_i}\sum_{u=1}^{n_i}(Y_{u,i}-\hat{\mu})$$

with n_i the number of ratings given for movie i.

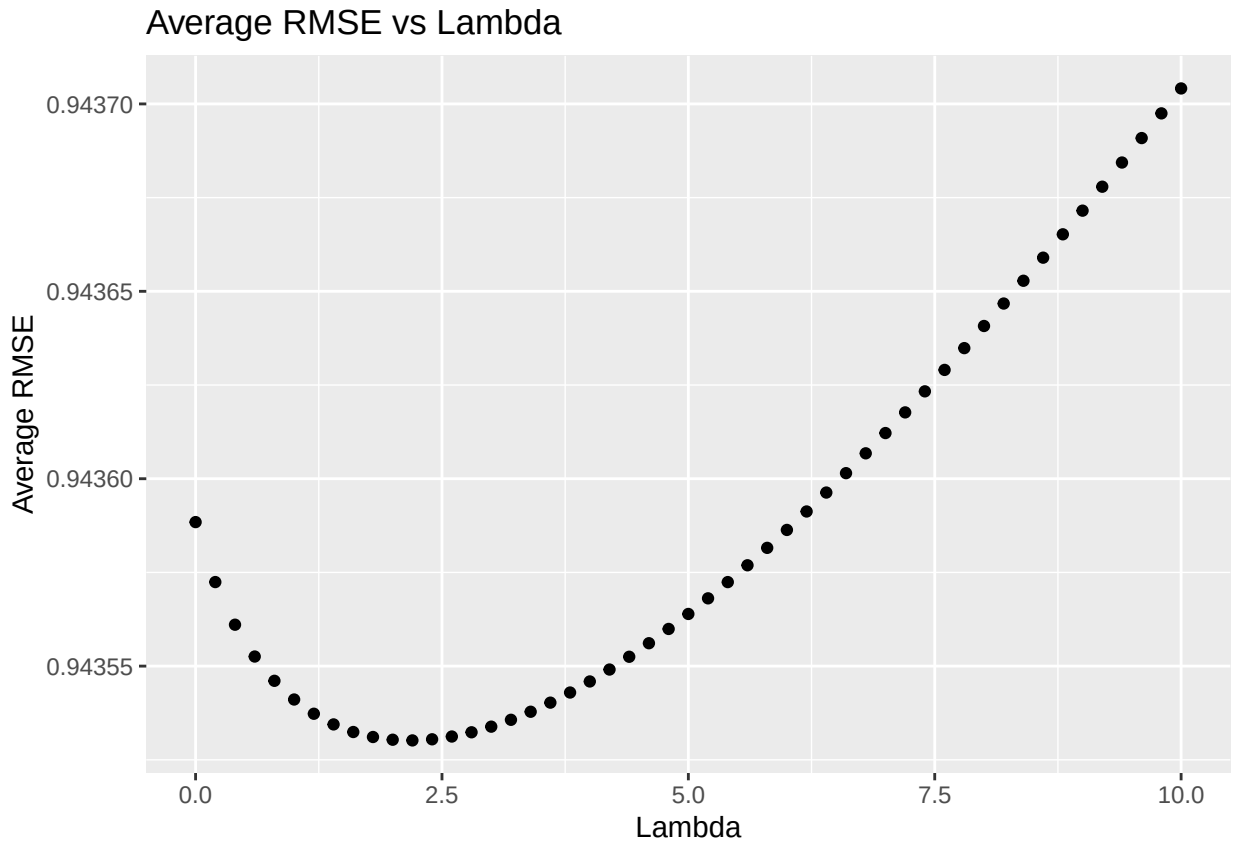
This equation perfectly fitted the intended goal - for large samples n >> λ, thus the penalty term is practically ignored; for smaller size, instead, the penalty term λ would cause the estimate to shrink towards 0. The level of shrinkage will depend on the chosen value of λ.

Thus, prior to the estimates of the regularised movie effects b_i, it was required to tune the parameter λ. The tuning process was implemented through a **k-fold cross-validation**. The idea behind this procedure is to split the *training* dataset into k subsets (*folds*), which will all be used at some point as a test set to evaluate a model trained for each given value of λ.

In detail, given a sequence of values for λ, the process consisted of a loop of 10 iterations (as many as the folds) and in each iteration: 1. 9 folds were combined and used a training set to train the models, while the remaining fold was used as a test set; 2. for each value of λ, a model was trained by using the *regularised* values of b_i and the relative RMSE was calculated. At the end of the loop, for each value of λ 10 values of RMSE were obtained The λ with the minimum average RMSE was eventually selected (λ =2.2).

```r
# Tuning the penalty term lambda through cross-validation of traning set

# Creating the index 'folds' that will be used to split the training set into
# 10 equally sized folds
set.seed(87, sample.kind = "Rounding")
folds <- createFolds(train_set$rating, k = 10, list = TRUE, returnTrain = FALSE)


# Calculating the value of lambda that maximise the RMSE in each folds
# Initialising the vector that will contain all the possible values of lambda that will be
# used during the tuning process
lambdas <- seq(0, 10, 0.20)

# Initialising the dataframe that will store the value of the RMSE for each value of lambda
# obtained in each folds
rmse_folds <- as.data.frame(matrix(ncol=length(lambdas), nrow=10))
colnames(rmse_folds) <- lambdas


# Creating a loop to implement the cross-validation process.
for (i in 1:10) {
  #create the train and test folds
  cv_train_set <- train_set[ -folds[[i]], ]
  temp <-  train_set[ folds[[i]], ]

  # Make sure userId and movieId in test set are also in train set
  cv_test_set <- temp %>%
    semi_join(cv_train_set, by = "movieId") %>%
    semi_join(cv_train_set, by = "userId")

  # Add rows removed from test set back into train set
  removed <- anti_join(temp, cv_test_set)
  cv_train_set <- rbind(cv_train_set, removed)

  #removing the temporary tables
  rm( temp, removed)

  #training the model
  mu <- mean(cv_train_set$rating)
  just_sum <- cv_train_set %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())

  #calculating the regularised b_i for each value of lambda
  rmses <- sapply(lambdas, function(l){
  predicted_ratings <- cv_test_set %>%
    left_join(just_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, cv_test_set$rating))
  })
```

```
  #storing the RMSEs of each lambdas that have been calculated using the fold-i
  rmse_folds[i,] <- rmses
}

# Calculating the average RMSEs for each value of lambda
 avg_rmse_folds <- colMeans(rmse_folds)

# Visualising  avg_RMSE vs lambda
qplot(lambdas, avg_rmse_folds, main= "Average RMSE vs Lambda",
      xlab ="Lambda",
      ylab = "Average RMSE" )
```




```
# Calculating and storing the value of lambda that minimise the avg RMSE (lambda = 2.2)
lambda <- lambdas[which.min(avg_rmse_folds)]
lambda
```

```
## [1] 2.2
```

Once λ was tuned, the regularised estimates b_i were obtained.

```
# Calculating the regularised estimates of b_i for the optimal lambda using the whole training set
mu <- mean(train_set$rating)
movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
```

As expected, the RMSE of the model including the *regularised movie effect* was lower than the RMSE of the model without regularisation.

```
# a) Regularized Movie Effect model (no users-effect)
# Making the predictions
predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)

#Calculating the RMSE of the regularised model
regularised_movie_effect_rmse <- RMSE(predicted_ratings, test_set$rating)

# b) Regularized movie Effect model (including users-effect)
# Making the predictions
predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Calculating the RMSE of the new model
regularised_movie_and_user_effect_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- tibble(method = c("Just the average", "Avg + Movie Effect",
                                  "Avg + Movie Eff + User Eff", "Avg + Regul. Movie Eff",
                                  "Avg + Regul. Movie Eff + User Eff"),
                       RMSE = c(naive_rmse, movie_effect_rmse, user_effect_rmse,
                                regularised_movie_effect_rmse,
                                regularised_movie_and_user_effect_rmse))
rmse_results
```

```
## # A tibble: 5 x 2
##   method                              RMSE
##   <chr>                              <dbl>
## 1 Just the average                   1.06
## 2 Avg + Movie Effect                 0.945
## 3 Avg + Movie Eff + User Eff         0.866
## 4 Avg + Regul. Movie Eff             0.945
## 5 Avg + Regul. Movie Eff + User Eff 0.866
```

In addition, the predictions resulted were more robust. The top 10 best/worst movies after applying the regularisation had more ratings.

```
# Top 10 best movies based on the penalized estimates b_i
train_set %>%
  count(movieId) %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  left_join(movie_titles, by = "movieId") %>%
  arrange(desc(b_i)) %>%
  slice(1:10) %>%
  pull(title)
```

```
##  [1] "Shawshank Redemption, The (1994)"
```

```
##  [2] "More (1998)"
##  [3] "Godfather, The (1972)"
##  [4] "Usual Suspects, The (1995)"
##  [5] "Schindler's List (1993)"
##  [6] "Casablanca (1942)"
##  [7] "Rear Window (1954)"
##  [8] "Paths of Glory (1957)"
##  [9] "Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)"
## [10] "Human Condition III, The (Ningen no joken III) (1961)"
```

```r
# Worst 10 best movies based on the penalized estimates b_i
train_set %>%
  count(movieId) %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  left_join(movie_titles, by = "movieId") %>%
  arrange(desc(-b_i)) %>%
  slice(1:10) %>%
  pull(title)
```

```
##  [1] "SuperBabies: Baby Geniuses 2 (2004)"
##  [2] "From Justin to Kelly (2003)"
##  [3] "Pokémon Heroes (2003)"
##  [4] "Disaster Movie (2008)"
##  [5] "Barney's Great Adventure (1998)"
##  [6] "Carnosaur 3: Primal Species (1996)"
##  [7] "Hip Hop Witch, Da (2000)"
##  [8] "Glitter (2001)"
##  [9] "Gigli (2003)"
## [10] "Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie) (2002)"
```

```r
# Calculating the number or reviews of the top/worst movies
#  Best 10 movies number of ratings
train_set %>%
  count(movieId) %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  left_join(movie_titles, by = "movieId") %>%
  arrange(desc(b_i)) %>%
  slice(1:10) %>%
  pull(n)
```

```
##  [1] 26710     7 16858 20573 22042 10660  7469  1489     4     4
```

```r
#  Worst 10 movies number of ratings
train_set %>%
  count(movieId) %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  left_join(movie_titles, by = "movieId") %>%
  arrange(desc(-b_i)) %>%
  slice(1:10) %>%
  pull(n)
```

```
##  [1]  54 186 132  29 198  64  14 327 298 196
```

The regularisation process was applied to the *user effects* b_u. In this case, the equation to minimise was:

$$\frac{1}{N}\sum_{u,i}(y_{ui}-\mu-b_i-b_u)^2+\lambda(\sum_i b_i^2+\sum_u b_u^2)$$

The estimates that minimise the above equations were obtained in a way similar to the case of the *regularised movie effects*. K-fold cross-validation was used once again used to tuned the value of the parameter λ for the full model.

```
# Tuning the penalty term lambda through cross-validation of traning set

# 1) Creating the index 'folds' that can be used to split the training set
# into  10 equally sized folds
set.seed(107, sample.kind = "Rounding")
folds <- createFolds(train_set$rating, k = 10, list = TRUE, returnTrain = FALSE)


# 2) Calculating the value of lambda that maximise the RMSE in each folds
# Initialising the vector that will contain all the possible values of lambda
# that will be used during the tuning process
lambdas <- seq(0, 10, 0.20)

# Initialising the dataframe that will store the value of the RMSE
# for each value of lambda obtained in each folds
rmse_folds <- as.data.frame(matrix(ncol=length(lambdas), nrow=10))
colnames(rmse_folds) <- lambdas


# Creating a loop to implement the cross-validation process.
for (i in 1:10) {
  #create the train and test folds
  cv_train_set <- train_set[ -folds[[i]], ]
  temp <-  train_set[ folds[[i]], ]

  # Make sure userId and movieId in test set are also in train set
  cv_test_set <- temp %>%
    semi_join(cv_train_set, by = "movieId") %>%
    semi_join(cv_train_set, by = "userId")

  # Add rows removed from test set back into train set
  removed <- anti_join(temp, cv_test_set)
  cv_train_set <- rbind(cv_train_set, removed)

  #removing the temporary tables
  rm( temp, removed)


  #training the model
  mu <- mean(cv_train_set$rating)

  #calculating the regularised b_i and b_u for each value of lambda
  rmses <- sapply(lambdas, function(l){
    b_i <- train_set %>%
      group_by(movieId) %>%
      summarise(b_i=sum(rating-mu)/(n()+l))
```

```
    b_u <- train_set %>%
      left_join(b_i, by="movieId") %>%
      group_by(userId) %>%
      summarise(b_u= sum(rating-b_i-mu)/(n()+l))

    predicted_ratings <- cv_test_set %>%
      left_join(b_i, by = "movieId") %>%
      left_join(b_u, by = "userId") %>%
      mutate(pred = mu + b_i +b_u) %>%
      pull(pred)
    return(RMSE(predicted_ratings, cv_test_set$rating))
  })

  #storing the RMSEs of each lambdas that have been calculated using the fold-i
  rmse_folds[i,] <- rmses
}

# Calculating the average RMSEs for each value of lambda
avg_rmse_folds <- colMeans(rmse_folds)

# Visualising  avg_RMSE vs lambda
qplot(lambdas, avg_rmse_folds,
      main= "Average RMSE vs Lambda - User effects",
      xlab ="Lambda",
      ylab = "Average RMSE" )

# Calculating and storing the value of lambda that minimise the avg RMSE
lambda <- lambdas[which.min(avg_rmse_folds)]
lambda
```

Once λ for the full model was calculated ($\lambda = 0.6$), the regularised estimates b_i and b_u were calculated using the entirety of the training set and used to make the predictions and calculate the new RMSE.

```
# Calculating the regularised estimates of b_i and b_u for the optimal lambda
# for the full model using the whole training set
mu <- mean(train_set$rating)

movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())

user_reg_avgs <- train_set %>%
  group_by(userId) %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  summarize(b_u = sum(rating - b_i- mu)/(n()+lambda), n_u = n())

# Making the predictions
predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  left_join(user_reg_avgs, by='userId')%>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
```

```r
# Calculating the RMSE of the new model
regularised_movie_and_user_effect_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- tibble(method = c("Just the average","Avg + Movie Effect",
                                  "Avg + Movie Eff + User Eff", "Avg + Regul. Movie Eff",
                                  "Avg + Regul. Movie Eff + Regul. User Eff"),
                       RMSE = c(naive_rmse,movie_effect_rmse,
                              user_effect_rmse, regularised_movie_effect_rmse,
                              regularised_movie_and_user_effect_rmse))
rmse_results
```

```
## # A tibble: 5 x 2
##   method                                 RMSE
##   <chr>                                 <dbl>
## 1 Just the average                      1.06
## 2 Avg + Movie Effect                    0.945
## 3 Avg + Movie Eff + User Eff            0.866
## 4 Avg + Regul. Movie Eff                0.945
## 5 Avg + Regul. Movie Eff + Regul. User Eff 0.866
```

The regularisation of the *user effects* improved the RMSE only marginally by 0.000046. However, the computational resources used to run the code increased exponentially, thus making the code nearly impossible to run on machines with low specifications. For this reason, the regularisation of the user effect was discarded and only the regularisation of the movie effects was instead included in the model.

Fifth model: including the feature engineered *years from movie's release*

In the exploratory data analysis it was pointed out that the time when the rating was given could hold predictive power. For this reason, the new feature *years since movie's release* was engineered to keep track of how many years had elapsed since the movie's release. It was eventually suggested to include this engineered feature instead of the rating *timestamp*, as it showed to have a larger influence on the average rating.

Defined b_r as the least square estimate that represented the *years from the movie's release* effect, the new model was given by the following equation:

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

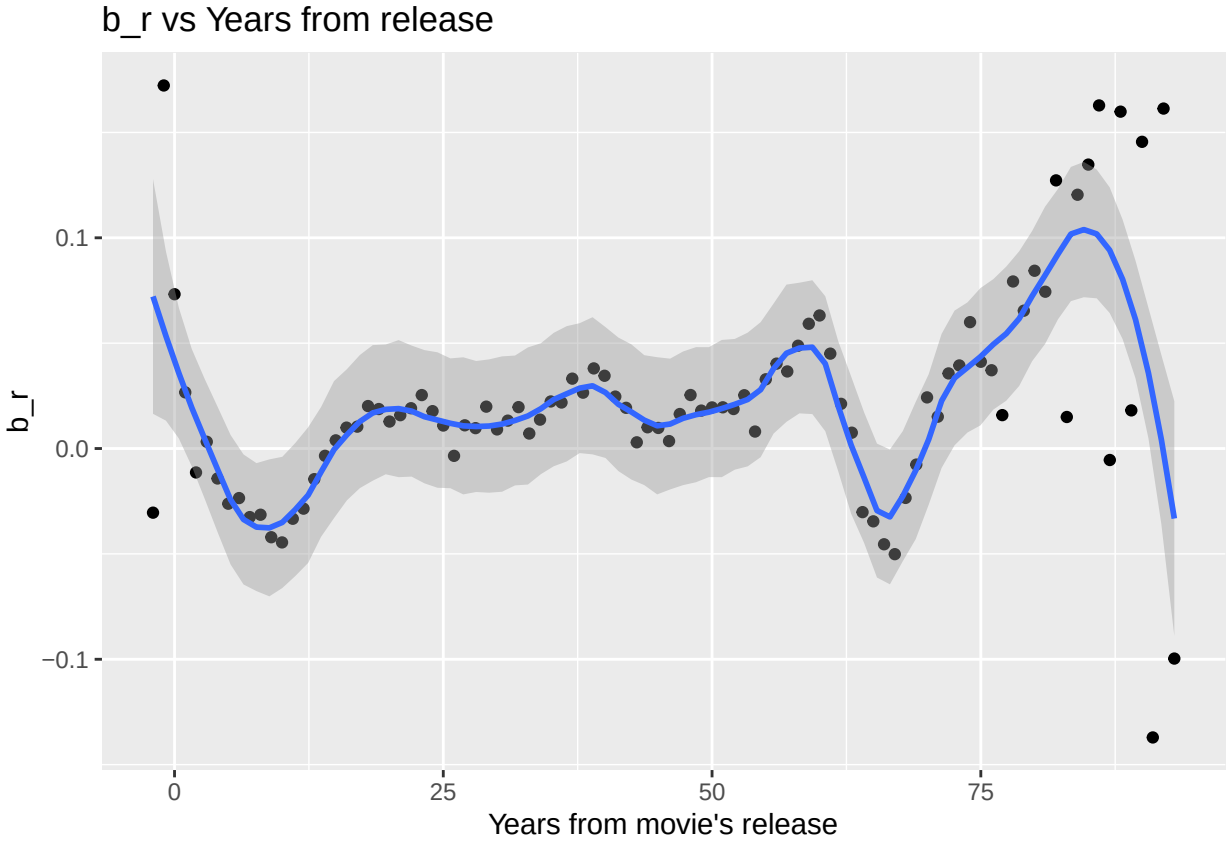
The process to estimate the effects b_r was similar to the process that was carried out to calculate the movie and user effects described above.

```r
# Estimating the least square estimates b_r and save these in the table release_avgs
release_avgs <- train_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(years_from_release = as.numeric(year(date)-as.numeric(year_release))) %>%

  group_by(years_from_release) %>%
  summarize(b_r = mean(rating - mu - b_i - b_u))
```

However, there was an important difference - this process could only estimate the effects b_r for a limited number of *years from the movie's release* (i.e. only the values included in the training set). By plotting the value of the estimate b_r for each value of *years from the movie's release* it was clear that only the b_r for

movies released within 95 years had been calculated. This meant that the model, as it was, could not predict any value for ratings given to movies that had been released over 95 years from the day that the rating was given.

```
# Plotting b_r
release_avgs %>%
  ggplot(aes(x=years_from_release, y=b_r)) +
  geom_point() +
  geom_smooth(formula = y ~ x, method = 'loess', span=0.2) +
  ggtitle("b_r vs Years from release") +
  xlab("Years from movie's release") +
  ylab("b_r")
```



Machine learning algorithms must be able to adapt properly to new and previously unseen data. For this reason, a *smooth* function was used to identify a function that generalised the value of b_r for any given *years from the movie's release*.

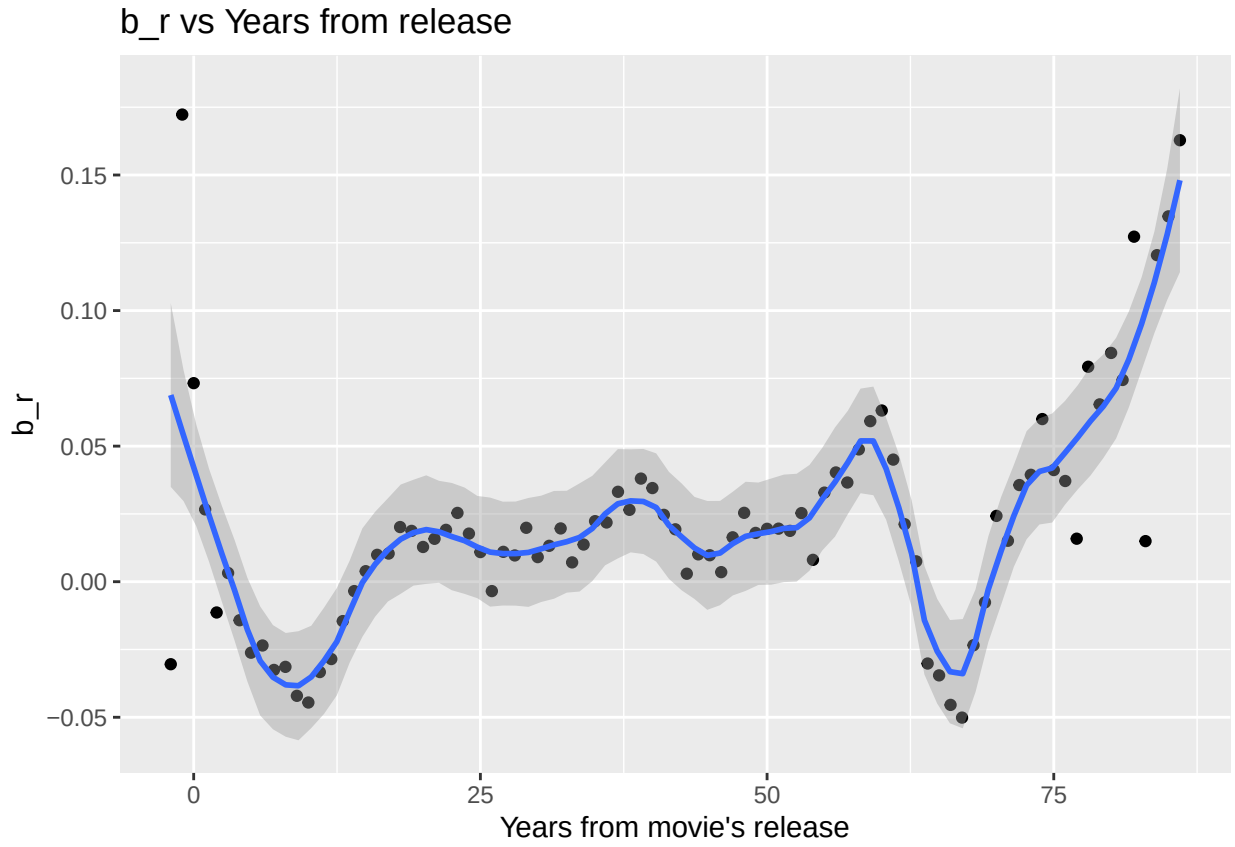
The chart gave alright some insights on the shape of the regression curve. Particularly, the smooth function struggled to fit adequately observations for value of *year's from movie's release* > 85. This issue was due to small sample sizes for the largest value of *years from movie's release* - as a consequence those values were excluded from the regression when the model was fitted.

```
# Veryfing the sample sizes for the largest values of 'Years from movie's release'
  train_set %>%
  mutate(years_from_release = as.numeric(year(date)-as.numeric(year_release))) %>%
  group_by(years_from_release) %>%
```

```
  summarize(n_obs = n()) %>%
  tail(20)
```

```
## # A tibble: 20 x 2
##    years_from_release n_obs
##                 <dbl> <int>
##  1                 74  2947
##  2                 75  2760
##  3                 76  1844
##  4                 77  1612
##  5                 78  1332
##  6                 79  1172
##  7                 80  1072
##  8                 81   876
##  9                 82   664
## 10                 83   591
## 11                 84   398
## 12                 85   384
## 13                 86   284
## 14                 87   133
## 15                 88   160
## 16                 89   100
## 17                 90    55
## 18                 91    40
## 19                 92    27
## 20                 93    13
```

```
  # Plotting smooth function for b_r  with years_from_release<87
  release_avgs %>%
    filter(years_from_release<87) %>%
    ggplot(aes(x=years_from_release, y=b_r)) +
    geom_point() +
    geom_smooth(formula = y ~ x, method = 'loess', span=0.2) +
    ggtitle("b_r vs Years from release") +
    xlab("Years from movie's release") +
    ylab("b_r")
```




```
fit <- release_avgs %>%
  filter(years_from_release<87) %>%
  #selecting only 'years_from_release' with at least 200 observations
  loess(formula = b_r~years_from_release,control=loess.control(surface="direct"), span=0.2)
  # As we have seen from the chart, the smooth function with span = 0.2 gives us
  # a good aproximation of the b_fr estimates
```

The model thus became:

$$Y_{u,i} = \mu + b_i + b_u + f(y_{fr}) + \epsilon_{u,i}$$

with $f(y_{fr})$ being a smooth function of the number of years from the movie's release (y_{fr}).

```
# Making predictions
predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(release_avgs, by='years_from_release') %>%
  mutate(b_r = predict(fit, years_from_release), pred = mu + b_i + b_u + b_r)  %>%
  # The predict function is needed to estimate the value of b_fr for any given value of
  # 'years_from_first_rate'
  pull(pred)

# Calculating RMSE
years_from_rel_effect_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- tibble(method = c("Just the average","Avg + Movie Effect",
                                  "Avg + Movie Eff + User Eff", "Avg + Regul. Movie Eff",
```

```
                                  "Avg + Regul. Movie Eff + User Eff",
                                  "Regul. Movie Eff + User Eff+Years from Rel eff"),
                        RMSE = c(naive_rmse,movie_effect_rmse, user_effect_rmse,
                                 regularised_movie_effect_rmse,
                                 regularised_movie_and_user_effect_rmse,
                                 years_from_rel_effect_rmse))
rmse_results
```

```
## # A tibble: 6 x 2
##   method                                      RMSE
##   <chr>                                      <dbl>
## 1 Just the average                           1.06
## 2 Avg + Movie Effect                         0.945
## 3 Avg + Movie Eff + User Eff                 0.866
## 4 Avg + Regul. Movie Eff                     0.945
## 5 Avg + Regul. Movie Eff + User Eff          0.866
## 6 Regul. Movie Eff + User Eff+Years from Rel eff 0.865
```

The new model improved the RMSE by 0.00037.

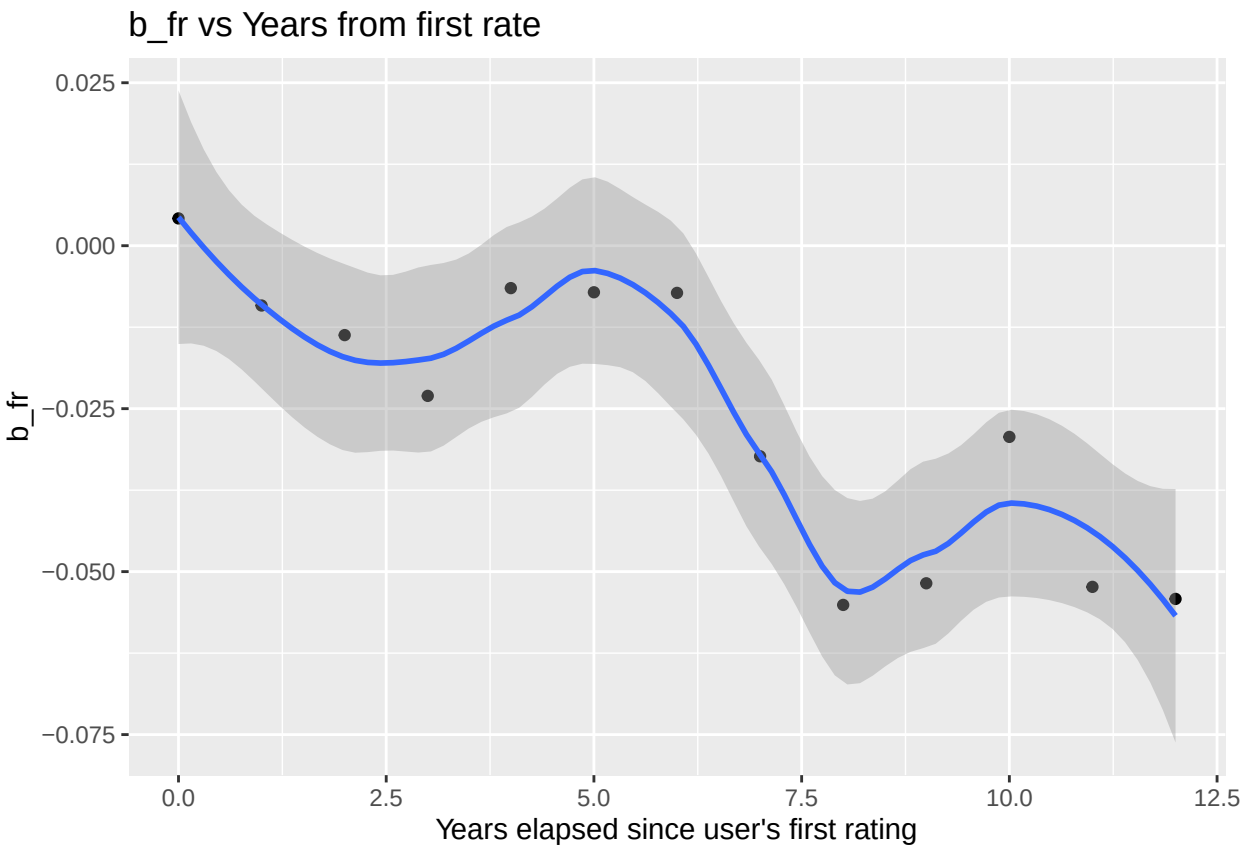
Sixth model: including the feature engineered *years from user's first rating*

Another example of featured engineered from the variable *timestamp* and that was shown during the data exploration to have the potential to influence users' ratings was the number of *years from user's first rating*. Similarly to the *years from movie's release* effect, due to the variable *time* being involved , this factor could be modeled with a smooth function of the number of year since the user's first rating (y_{fufr}). Thus, the new model was represented by the new equation:

$$Y_{u,i} = \mu + b_i + b_u + f(y_{fr}) + f(y_{fufr}) + \epsilon_{u,i}$$

```
# Function to calculate how many months have elapsed between 2 dates
elapsed_months <- function(end_date, start_date) {
  ed <- as.POSIXlt(end_date)
  sd <- as.POSIXlt(start_date)
  12 * (ed$year - sd$year) + (ed$mon - sd$mon)
}

# Estimating the least square  b_fr and save these stored in the table years_from_first_rate_avg
years_from_first_rate_avg <- train_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(first_rates, by='userId') %>%
  mutate(years_from_first_rate = ceiling(elapsed_months(date, date_first_rate)/12)) %>%
  # we are using the function created previously to calculate how many years have elapsed
  # since the user's rating - we divide by 12 as there are 12 months in a year
  left_join(release_avgs, by='years_from_release') %>%
  group_by(years_from_first_rate) %>%
  summarize(b_fr = mean(rating - mu - b_i - b_u - b_r))

# Plotting b_fr
years_from_first_rate_avg %>%
  ggplot(aes(x=years_from_first_rate, y=b_fr)) +
```

```
  geom_point() +
  geom_smooth(formula = y ~ x, method = 'loess', span=0.5) +
  ggtitle("b_fr vs Years from first rate") +
  xlab("Years elapsed since user's first rating") +
  ylab("b_fr")
```




```
fit <- years_from_first_rate_avg %>%
  loess(formula = b_fr~years_from_first_rate,
        control=loess.control(surface="direct"),
        span=0.5)
# As we have seen from the chart, the smooth function with span=0.5 gives us
# a good aproximation of the b_fr estimates

# Making predictions
predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(first_rates, by='userId') %>%
  mutate(years_from_first_rate = ceiling(elapsed_months(date, date_first_rate)/12)) %>%
  left_join(release_avgs, by='years_from_release') %>%
  left_join(years_from_first_rate_avg, by='years_from_first_rate') %>%
  mutate(b_fr = predict(fit, years_from_first_rate),
         pred = mu + b_i + b_u + b_r+ b_fr)  %>%
  # The predict function is needed to estimate the value of b_fr
  # for any given value of 'years_from_first_rate'
```

```
  pull(pred)

years_from_first_rate_effect_rmse <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- tibble(method = c("Just the average","Avg + Movie Effect",
                                  "Avg + Movie Eff + User Eff", "Avg + Regul. Movie Eff",
                                  "Avg + Regul. Movie Eff + User Eff",
                                  "Regul. Movie Eff + User Eff+Years from Rel eff",
                                  "Avg. + Reg. Movie + User + Years from rel. + Years from first rate")
                       RMSE = c(naive_rmse,movie_effect_rmse, user_effect_rmse,
                              regularised_movie_effect_rmse,
                              regularised_movie_and_user_effect_rmse,
                              years_from_rel_effect_rmse,
                              years_from_first_rate_effect_rmse))
rmse_results
```

```
## # A tibble: 7 x 2
##   method                                                          RMSE
##   <chr>                                                          <dbl>
## 1 Just the average                                               1.06
## 2 Avg + Movie Effect                                             0.945
## 3 Avg + Movie Eff + User Eff                                     0.866
## 4 Avg + Regul. Movie Eff                                         0.945
## 5 Avg + Regul. Movie Eff + User Eff                              0.866
## 6 Regul. Movie Eff + User Eff+Years from Rel eff                 0.865
## 7 Avg. + Reg. Movie + User + Years from rel. + Years from first rate 0.865
```

The model that included the feature *years from user's first rating* had a RMSE of 0.8653707, an improvement of 0.000092.

Seventh model: including the *genre effect*

The final model that was fitted included the *genre effects* b_g. The equation representing the final model was the follow:

$$Y_{u,i} = \mu + b_i + b_u + f(y_{fr}) + f(y_{fufr}) + b_g + \epsilon_{u,i}$$

The approach used to estimate the least square estimated b_g was the same used to calculate the factors b_i and b_u.

```
# Estimating the least square the least square estimates b_g and save these in the table genre_avg
genre_avgs <- train_set %>%
  left_join(first_rates, by='userId') %>%
  mutate(years_from_first_rate = ceiling(elapsed_months(date, date_first_rate)/12)) %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  group_by(userId) %>%
  left_join(user_avgs, by='userId') %>%
  left_join(release_avgs, by='years_from_release') %>%
  left_join(years_from_first_rate_avg, by='years_from_first_rate') %>%
  summarize(b_g = mean(rating - mu - b_i - b_u - b_r- b_fr))

# Calculating predictions
```

```
predicted_ratings <- test_set %>%
  left_join(first_rates, by='userId') %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='userId') %>%
  mutate(years_from_first_rate = ceiling(elapsed_months(date, date_first_rate)/12)) %>%
  left_join(release_avgs, by='years_from_release') %>%
  left_join(years_from_first_rate_avg, by='years_from_first_rate') %>%
  mutate(b_fr = predict(fit, years_from_first_rate), pred = mu + b_i + b_u + b_r+ b_fr +b_g) %>%
  pull(pred)

# Calculating RMSE
genre_effect_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- tibble(method = c("Just the average","Avg + Movie Effect",
                                  "Avg + Movie Eff + User Eff",
                                  "Avg + Regul. Movie Eff",
                                  "Avg + Regul. Movie Eff + User Eff",
                                  "Regul. Movie Eff + User Eff+Years from Rel eff",
                                  "Avg. + Reg. Movie + User + Years from rel. + sems. from first rate",
                                  "Avg. + Reg. Movie + User + Years from rel. + sems. from first rate,Y
                       RMSE = c(naive_rmse,movie_effect_rmse, user_effect_rmse,
                                regularised_movie_effect_rmse,
                                regularised_movie_and_user_effect_rmse,
                                years_from_rel_effect_rmse,
                                years_from_first_rate_effect_rmse,
                                genre_effect_rmse))
rmse_results
```

```
## # A tibble: 8 x 2
##   method                                                                RMSE
##   <chr>                                                                <dbl>
## 1 Just the average                                                     1.06
## 2 Avg + Movie Effect                                                   0.945
## 3 Avg + Movie Eff + User Eff                                           0.866
## 4 Avg + Regul. Movie Eff                                               0.945
## 5 Avg + Regul. Movie Eff + User Eff                                    0.866
## 6 Regul. Movie Eff + User Eff+Years from Rel eff                       0.865
## 7 Avg. + Reg. Movie + User + Years from rel. + sems. from first rate   0.865
## 8 Avg. + Reg. Movie + User + Years from rel. + sems. from first rate,Year~ 0.865
```

Using the feature *genres* as a predictor improved once again the model. The final RMSE of the model when using the *test set* to make predictions was 0.8653368.

RESULTS

All the features highlighted during the data exploration analysis were examined and their inclusion in the algorithm somewhat improved the predictions made on the *test set*. As a consequence, the final model included the following predictors:

- a baseline rating μ which would be the same for user-movie predicted ratings and equal to the average of user-movie ratings calculated from the training set;

- a factor b_i which depended on the specific movie i and that represented a *movie effect* that adjusted the baseline predictor based on the *quality* of the movie i;
- a factor b_u which depended uniquely on the user who was giving the rating and that represented a *user effect* that adjusted the baseline predictor based on the *attitude of the user* toward the rating activity;
- a function of time $f(y_{fr})$ that adjusted the baseline rating depending on how many *years elapsed from the movie's release* the moment when the rating was given;
- a function of time $f(y_{fr})$ that adapted the baseline rating based on how many *years elapsed from the user's first rating*;
- a factor b_g which depended uniquely on the genre g associated with the movie and that represented a *genre effect* that adjusted the baseline rating based on the genre.

The predicted rating $\hat{Y}_{u,i}$ were therefore calculated by the following equation:

$$\hat{Y}_{u,i} = \mu + b_i + b_u + f(y_{fr}) + f(y_{fufr}) + b_g$$

The final test of the algorithm was to predict the movie ratings by using the *validation* set as if these were unknown. The RMSE was once again used to evaluate how close the predictions were to the true values in the validation set. The result was considered as the RMSE of the final model. This was a fundamental step of the process because it gave some key insights on the *gereralisability* of the final model outside the original dataset *edx* from which it was trained and tested. For instance, a RMSE significantly higher than the RMSE obtained from the test set could have been an indication of *over-fitting* - a situation given usually by complex models that minimise the RMSE of the test set by explaining the random error in the data rather than the relationships between predictors. Low generalisation ability could also be due *under-fitting*, basically the opposite of over-fitting - a model too simple that *did not learn enough* to be able to generalise predictions from unseen data.

Before calculating the RMSE of the final model, it was essential to pre-process the *validation* set, so as to be able to apply the same operations that were executed with the *test* set.

```
# ------ Data preprocessing ------
validation <- validation %>% mutate(date = as_datetime(timestamp))

# 1. Extracting the year when the movie was released
validation <- validation %>%
  mutate(year_release = str_extract(title, "(?<=\\()\\d{4}+(?=\\))")) %>%
# 2. Calculating the new feature 'years from release'
mutate(years_from_release = as.numeric(year(date)-as.numeric(year_release)))
```

Once the *validation* dataset was in the right format, the predictions were made and the final RMSE was calculated.

```
# ------ Making predictions ------
predicted_ratings <- validation %>%
  left_join(first_rates, by='userId') %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='userId') %>%
  mutate(years_from_first_rate = ceiling(elapsed_months(date, date_first_rate)/12)) %>%
  left_join(release_avgs, by='years_from_release') %>%
  left_join(years_from_first_rate_avg, by='years_from_first_rate') %>%
  mutate(b_fr = predict(fit, years_from_first_rate),
         pred=mu + b_i + b_u + b_r+ b_fr +b_g) %>%
  pull(pred)
```

```r
# ----- Calculating RMSE ------
final_RMSE <- RMSE(predicted_ratings, validation$rating)
final_RMSE
```

```
## [1] 0.8648948
```

The RMSE of the final machine learning model using the *validation* set was **0.8648948**.

$$RMSE_{final-model} = 0.8648948$$

This meant that the ratings predicted by the model were on average 0.865 (stars) higher or lower than the real ratings contained in the *validation* set. Looking at the RMSE, this is not best-performing movie recommendation system that could have been built, however the results was quite satisfactory and in line with the expectations - the value of the RMSE was in fact similar to the RMSE on test set (0.8653368). The fact that the two RMSEs are quite similar also suggests that the model might not suffer of *over-fitting*.

The are other positive aspects of the model performance that cannot be highlighted from the RMSE, but that deserve some attention:

- the model is highly **interpretable**. It is very easy to understand how the algorithm assigns a certain rating. By decomposing the rating into *baseline rating* and *effects*, what has driven a specific rating can be easily determined (e.g. an below the average rating due to a harshly critical user). Furthermore, the ability to interpret an erroneous prediction could help understand the cause of an error, thus delivering a direction for how to fix the algorithm should something go wrong in the future when the model is used.

- the code underlying the machine learning model is *quick to execute*. The first advantage of this property is that this movie system recommendation could be run by nearly anyone without heavily relying on the machine's specification. Secondly, this allows to easily *retrain* the algorithm once new data become available. Retraining is an important process in machine learning, as things change over time (e.g. new movies get release, users' preferences change, new genres become popular, etc.) and the data distributions can at some point deviate significantly from those of the original training set. Therefore, the accuracy of the model can start to decline, if something changes in the context used to train the algorithm. Since the code to train the model is simple and quick to execute, the algorithm can be retrained often and without major difficulties.

CONCLUSION

In summary, a movie recommendation system was developed in R. After having explored the dataset and identified the predictors to use in the machine learning algorithm, the modelling approach was described. The idea underlying the model was to decompose the predicted rating into a baseline rating equal for all user-movie rating and then adjust this by adding the effect of the predictors chosen in the exploratory data analysis. The effects were added one-by-one to the baseline model and kept in the following models only if the inclusion of the predictor improved the RMSE. All the models were trained on a partition of the original dataset *edx* (training set), while the remaining portion of this dataset was used as a *test set* to make predictions and calculate the RMSE. All the effects were kept in the final model - specifically, a user effect, a movie effect, an effect that depends on the number of year's elapsed between the movie's release and the rating, an effect that depends on the number of years elapsed between the rating and the user's first rating, a genre effect. The process of regularisation was used to adjust the movie-specific effect, in order to constrain its total variability, by penalizing large estimates that come from small sample sizes. Finally, the performance of the model were tested by making the predictions using a validation set composed by unseen data; these prediction were used to calculate the RMSE of the final model.

The final model did not reveal any signs of over-training, however it showed some limitations with the accuracy of the predictions. The predicted ratings were on average 0.865 stars higher or lower than the real ratings, in a 0-5 star range. While the simplicity of the model represented one of its strengths, in terms of interpretability and retraining abilities, it was also one of its limitations as it precluded the possibility to take into consideration more sophisticated patterns and effects that could determine a rating. For this reason, the possibility of using more advanced machine learning methods and including more elaborated effects should be explored in future work. For instance, a matrix factorisation approach combined with a principal component analysis could help identify *hidden structures* within the data that could explain variability in ratings due to *latent* components (for example similarity in user's preferences towards actors or nature of the movie). Using a neighborhood model could also be another approach that could be explored in the future.