HarvardX: PH125.9x Data Science
Movie Recommendation System

Marc Bender

May 12, 2020

Contents

1 Introduction

Movie recommendation systems are a powerful approach to predict how a given user u will rate a movie i based on different features. These recommendation systems are used by famous streaming services like Netflix. Indeed, Netflix initiated a competition in 2006 offering $1,000,000 to any person or team which is able to improve Netflix' own recommendation algorithm by at least 10 %. This prize was awarded in 2009 to the winning team *BellKor's Pragmatic Chaos* achieving a 10.06 % improvement [1].

In this project we attempt to develop our own movie recommendation system with the publicly available MovieLens 10M dataset provided by the research group GroupLens. This dataset is a database with approximately 10 million ratings for over 10,000 movies from around 70,000 users and is used to train a machine learning algorithm utilizing the features in the edx subset to predict the outcome (movie ratings ranging from 0.5 to 5 stars with 0.5 star intervals) in the validation set (edx and validation set are provided by the staff; see 1.1).

The required metric to assess model performance is the Root Mean Square Error (RMSE). RMSE is defined as the standard deviation of the residuals (prediction errors) and is widely used to measure differences between observed and predicted values in modelling approaches. As the RMSE measures squared distances it is sensitive to outliers, which has to be accounted for. This can be achieved by regularization approaches penalizing estimates generated by a small number of observations (detailed explanation in section 2.2). The RMSE is calculated as follows:

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

where $y_{u,i}$ represents an observed value in the test set for user u and movie i, $\hat{y}_{u,i}$ represents the respective expected value and N represents the number of observations. It can simply be calculated in R using a function such as:

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

provided by Prof. Irizarry during the HarvardX: PH125.8x machine learning class [2]. A lower RMSE corresponds to an overall better model performance, therefore the aim in machine learning is to minimize this error term. For this course an RMSE < 0.86490 shall be achieved.

The modelling approach was conducted on a standard home laptop with limitations concerning computational capacity (just 8 GB RAM). Therefore algorithms like *lm* or *knn* were not feasible and an approach calculating least square estimates manually has been chosen.

After loading the dataset, different variables were analysed in respect to effects on the rating of users. Models incorporating these effects into the model term were generated and tuned (cross-validation by splitting edx into training and test set) and the best performing model was chosen.This final model was retrained on the whole edx set and validated with the retained data from the validation set.

1.1 Data ingestion

The code to load the dataset was provided by the staff. Some adjustments prior to splitting the dataset into edx and validation subsets have been made. These changes include extracting the release year of a movie from the title column, transforming the timestamp into a date (rounded to the respective week), calculating the age of movies by substracting the release year from 2009 (data was gathered until 2008) and calculating the Rate as number of ratings per year for each movie. Algorithm development was carried out solely on

the edx subset and the final model was independently validated at the end by applying the algorithm to the validation subset and comparing observed with predicted values within said subset.

```
#load required packages
library(Hmisc)
library(tidyverse)
library(data.table)
library(caret)
library(doParallel)
library(lubridate)
library(kableExtra)

#activate parallel computing to avoid CPU-bound limitations
cl <- makeCluster(detectCores(), type='PSOCK')
registerDoParallel(cl)

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

# as of R 4.0.0 stringsAsFactors in as.data.frame has to be set to TRUE
movies <- as.data.frame(movies, stringsAsFactors =TRUE) %>%
  mutate(movieId = as.numeric(levels(movieId))[movieId],
         title = as.character(title),
         genres = as.character(genres))

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

#### Transformation of the movielens dataset
movielens_trans <- movielens %>%
  # extract year from title column
  mutate(year = as.numeric(str_sub(title,-5,-2)))%>%
  # calculate age of movies
  mutate(age = 2009-year) %>%
  # change timestamp to date (week format)
  mutate(date = as_datetime(timestamp)) %>%
  mutate(date = round_date(date, unit = "week"))

# add average number of ratings/year to each movie
movielens_trans <- movielens_trans %>%
    # calculate average number of ratings/year
  group_by(movieId) %>%
  summarize(n = n(), years = 2009 - first(year),
            title = title[1],
            rating = mean(rating)) %>%
```

```
  mutate(rate = n/years) %>%
    # add Rate to the existing edx data
  select(movieId, rate) %>%
  right_join(movielens_trans,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)` instead
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens_trans[-test_index,]
temp <- movielens_trans[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, movielens_trans, removed)
```

2 Methods

2.1 Exploratory data analysis and data visualisation

Exploratory data analysis is an important step in machine learning as it allows us to familiarize ourselves with the structure of the dataset, parameters that might be useful for modeling and general trends within the data. To get a quick glance of the data we can inspect the data and view the first 6 rows with the *head()* command. As shown below, the edx subset contains 10 variables: *movieId*, *rate*, *userId*, *rating*, *timestamp*, *title*, *genres*, *year*, *age* and *date*.

```
## # A tibble: 6 x 10
##   movieId  rate userId rating timestamp title genres  year   age
##     <dbl> <dbl>  <int>  <dbl>     <int> <chr> <chr>  <dbl> <dbl>
## 1     122  142.      1      5 838985046 Boom~ Comed~  1992    17
## 2     185 1070.      1      5 838983525 Net,~ Actio~  1995    14
## 3     292 1148.      1      5 838983421 Outb~ Actio~  1995    14
## 4     316 1262.      1      5 838983392 Star~ Actio~  1994    15
## 5     329 1078.      1      5 838983392 Star~ Actio~  1994    15
## 6     355  358.      1      5 838984474 Flin~ Child~  1994    15
## # ... with 1 more variable: date <dttm>
```

A summary of the data shows that the edx subset contains 9,000,055 observations, the mean rating μ is 3.512 and there are no missing values in the dataset. The mean rating μ will be saved for later and is depicted by a red dashed line in subsequent plots.

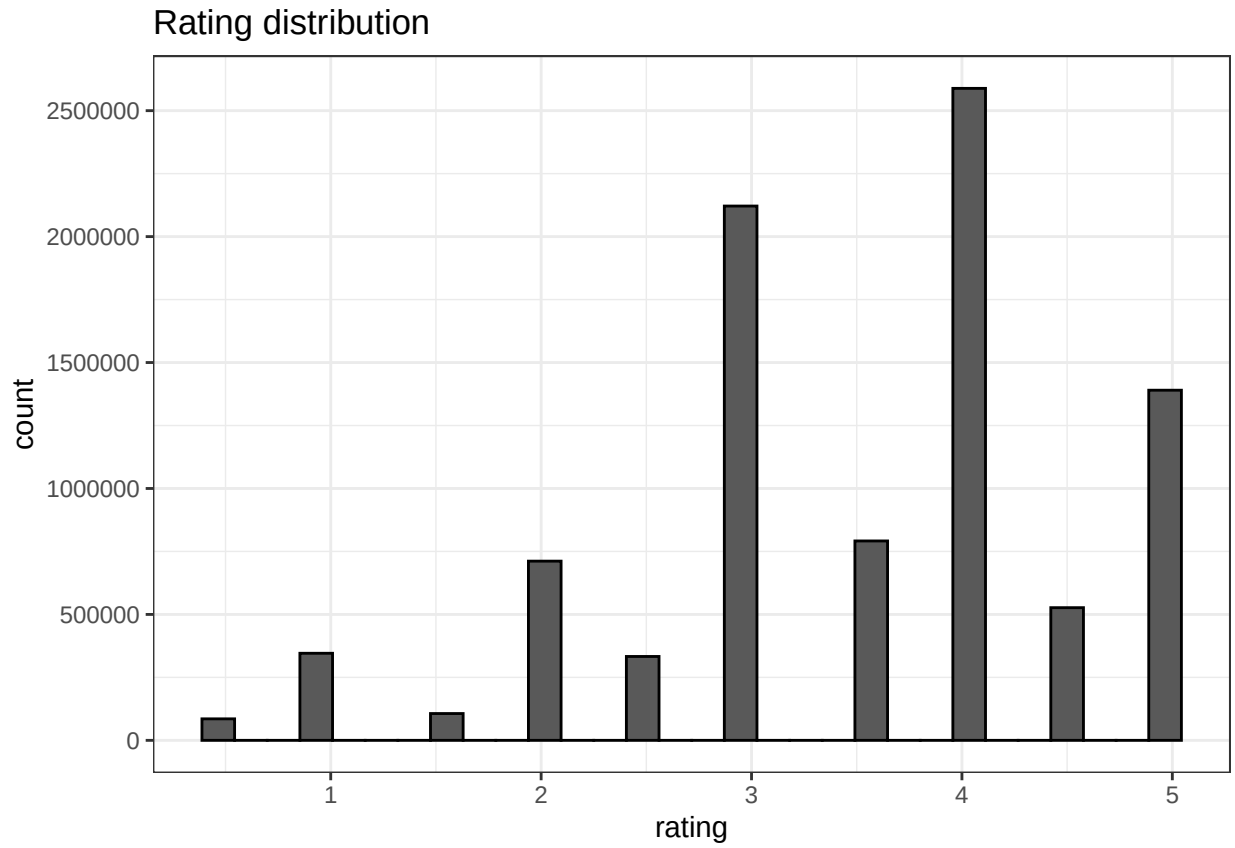
```
##     movieId           rate              userId          rating
##  Min.   :    1   Min.   :   0.0109   Min.   :    1   Min.   :0.500
##  1st Qu.:  648   1st Qu.: 110.6250   1st Qu.:18124   1st Qu.:3.000
##  Median : 1834   Median : 345.2414   Median :35738   Median :4.000
##  Mean   : 4122   Mean   : 533.0525   Mean   :35870   Mean   :3.512
##  3rd Qu.: 3626   3rd Qu.: 786.4167   3rd Qu.:53607   3rd Qu.:4.000
##  Max.   :65133   Max.   :2598.0000   Max.   :71567   Max.   :5.000
##    timestamp            title              genres              year
##  Min.   :7.897e+08   Length:9000055     Length:9000055     Min.   :1915
##  1st Qu.:9.468e+08   Class :character   Class :character   1st Qu.:1987
##  Median :1.035e+09   Mode  :character   Mode  :character   Median :1994
##  Mean   :1.033e+09                                         Mean   :1990
##  3rd Qu.:1.127e+09                                         3rd Qu.:1998
##  Max.   :1.231e+09                                         Max.   :2008
##       age             date
##  Min.   : 1.00   Min.   :1995-01-08 00:00:00
##  1st Qu.:11.00   1st Qu.:2000-01-02 00:00:00
##  Median :15.00   Median :2002-10-27 00:00:00
##  Mean   :18.78   Mean   :2002-09-21 12:42:48
##  3rd Qu.:22.00   3rd Qu.:2005-09-18 00:00:00
##  Max.   :94.00   Max.   :2009-01-04 00:00:00
```

The dataset contains 10,677 unique movies in 797 unique genres rated by 69,878 unique users.

n_users	n_movies	n_genres
69878	10677	797

In order to analyze the parameters in the dataset it is crucial to inspect the data distribution of the respective parameters.A histogram of movie ratings for instance reveals a left skewed distribution (users tend to give positive ratings more often than negative ones) and a higher frequency of full-star ratings compared to half-star ratings. The most frequent rating is 4 stars, followed by 3 and 5 stars. The least frequent rating is 0.5 stars.

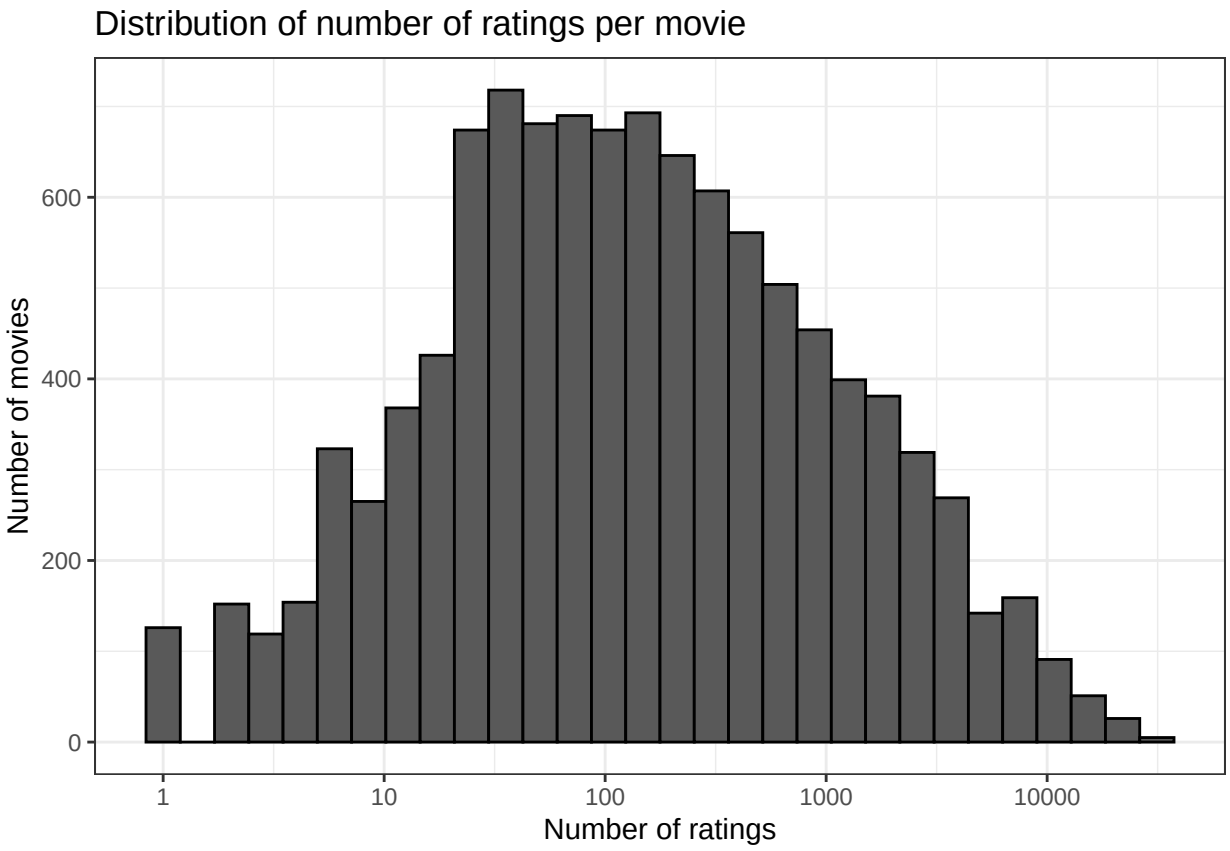
```
# distribution of movie ratings
edx %>%
  group_by(rating) %>%
  ggplot(aes(rating)) +
  geom_histogram(bins = 30, color = "black") +
  theme_bw() +
  scale_y_continuous(breaks = seq(0,2500000,by=500000)) +
  ggtitle("Rating distribution")
```



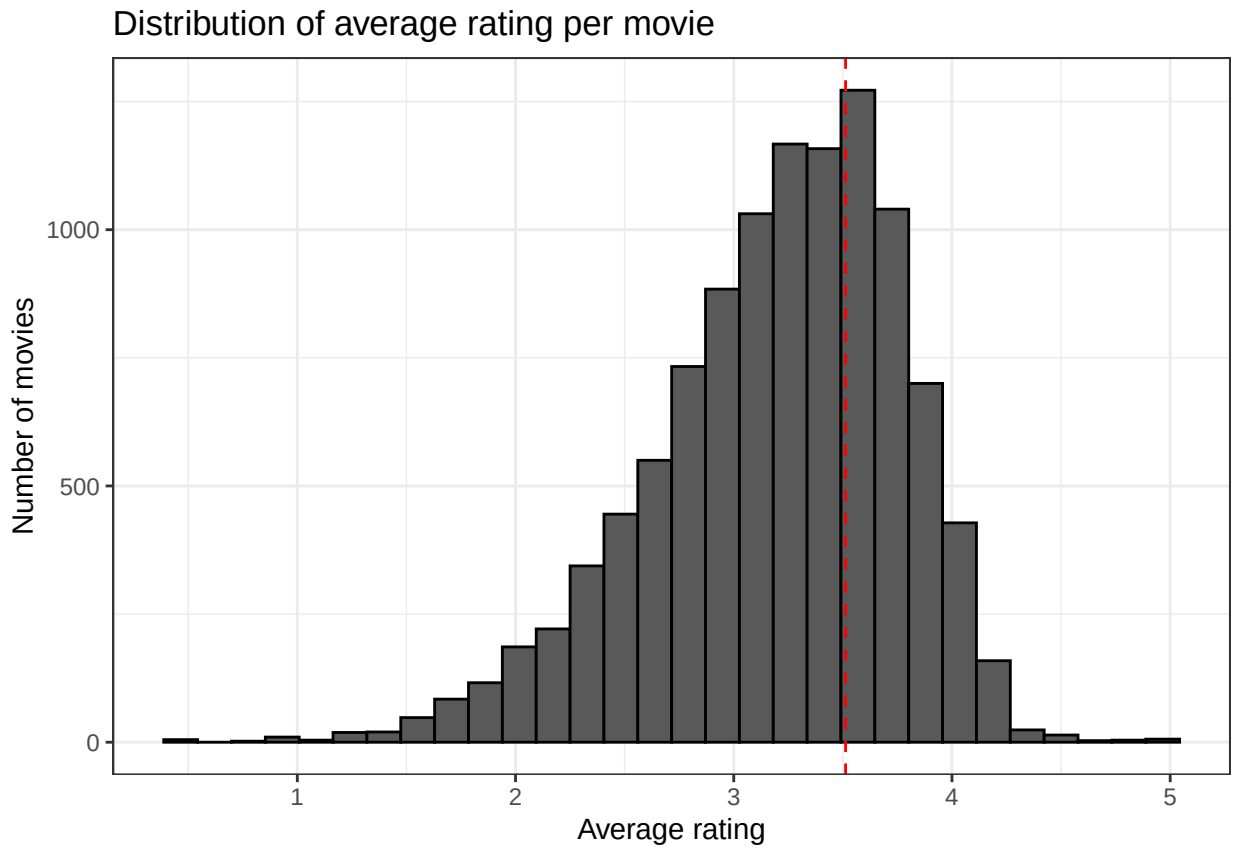
2.1.1 Movie Effect

To analyze the data for a putative movie effect, a histogram depicting the number of ratings per movie and a histogram showing the average rating per movie were generated. Some movies were rated over 10,000 times whereas other movies were rated very few times, sometimes even just once leading to unstable representations of the true movie rating and large errors. Furthermore some movies were generally rated higher (good movies) and some were generally rated lower (bad movies) confirming a suggested movie effect.

```
# distribution of number of ratings per movieId
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10()+
  theme_bw() +
  xlab("Number of ratings") +
  ylab("Number of movies") +
  ggtitle("Distribution of number of ratings per movie")
```



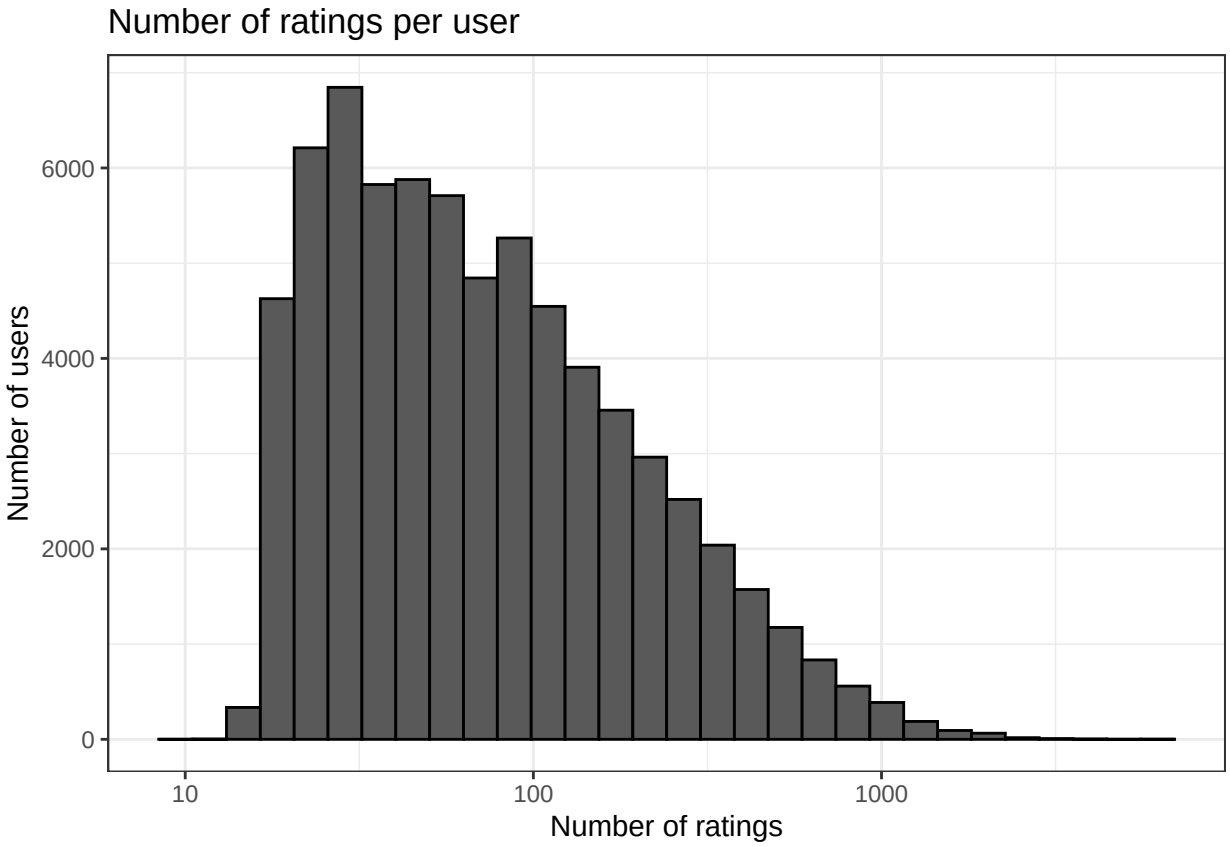
```r
# movie effect
edx %>%
  group_by(movieId) %>%
  summarize(b_movie = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_movie)) +
  geom_histogram(bins = 30, color = "black") +
  geom_vline(xintercept = mu, color = "red", lty = 2) +
  theme_bw() +
  xlab("Average rating") +
  ylab("Number of movies") +
  ggtitle("Distribution of average rating per movie")
```



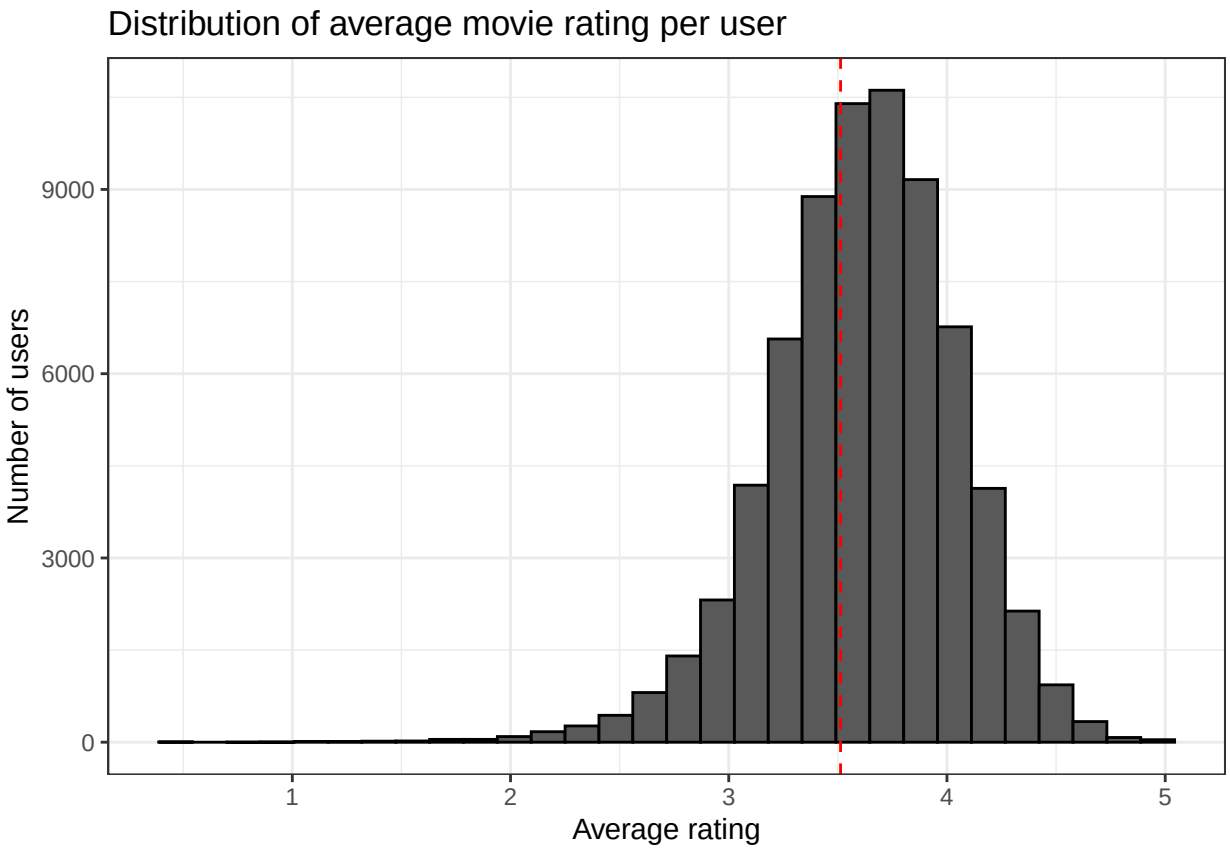
2.1.2 User Effect

Analogously to the movie effect, differences in the amount of ratings per user, ranging from over 1,000 to as low as 10 ratings per user could be detected. On top of that the average rating distribution reveals a user bias as some users are very critical and some users seem to enjoy every movie.

```
# distribution of userIds
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10()+
  theme_bw() +
  xlab("Number of ratings") +
  ylab("Number of users") +
  ggtitle("Number of ratings per user")
```



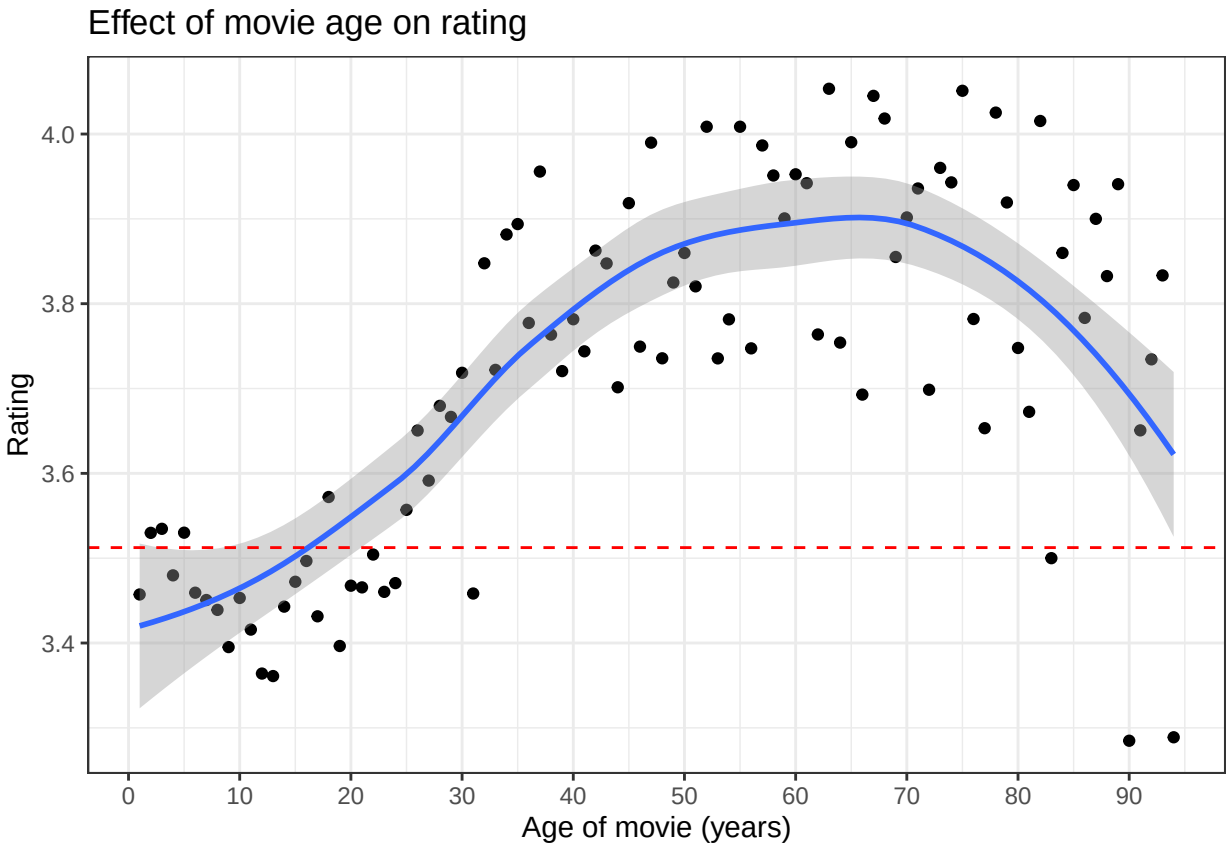
```r
# User effect
edx %>%
  group_by(userId) %>%
  summarize(b_user = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_user)) +
  geom_histogram(bins = 30, color = "black") +
  geom_vline(xintercept = mu, color = "red", lty = 2)+
  theme_bw() +
  xlab("Average rating") +
  ylab("Number of users") +
  ggtitle("Distribution of average movie rating per user")
```



2.1.3 Age Effect

The age of movies also affects the movie rating as more recent movies (age < 20 years) are rated below average with an increasing rating until the curve peaks at around 70 years (maximum rating of approximately 3.9 stars) and starts to converge towards the average again. This can be explained by a selection effect as old movies, which are still watched today, most likely are considered good movies. For recent movies this selection effect is not yet completed and newly published movies are watched by a bigger portion of the population therefore rated closer to the average.

```
# Age effect
edx %>% group_by(age) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(age, rating)) +
  geom_point() +
  geom_smooth()  +
  geom_hline(yintercept = mu, lty =2,color="red") +
  theme_bw() +
  scale_x_continuous(breaks = seq(0,100, by=10)) +
  xlab("Age of movie (years)") +
  ylab("Rating") +
  ggtitle("Effect of movie age on rating")
```



2.1.4 Rate Effect

Another possible effect influencing movie ratings is the number of ratings per year for each movie. To explore this putative effect we can take a look at the most and least rated movies as shown in the following tables. The depicted ratings are the averages over all users that rated the respective movie.

```
#### Rate
# check for movies with the highest number of ratings per year
most_rated <- edx %>%
  group_by(title) %>%
  summarize(mean_rate = mean(rate),
            mean_rating= mean(rating),n = n()) %>%
  top_n(25, mean_rate) %>%
  arrange(desc(mean_rate))

# print table
most_rated %>% kable(format = 'markdown')
```

title	mean_rate	mean_rating	n
Dark Knight, The (2008)	2598.000	4.297068	2353
Pulp Fiction (1994)	2324.267	4.154789	31362
Matrix, The (1999)	2322.900	4.202578	20908
Forrest Gump (1994)	2297.133	4.012822	31079
American Beauty (1999)	2212.000	4.185664	19950
Braveheart (1995)	2082.429	4.081852	26212
Shawshank Redemption, The (1994)	2075.067	4.455131	28015
Lord of the Rings: The Return of the King, The (2003)	2061.000	4.155224	11113
Lord of the Rings: The Two Towers, The (2002)	2055.571	4.119400	12969
Jurassic Park (1993)	2039.438	3.663522	29360
Iron Man (2008)	2013.000	3.992488	1797
Independence Day (a.k.a. ID4) (1996)	2003.231	3.376903	23449
Lord of the Rings: The Fellowship of the Ring, The (2001)	1992.250	4.162884	14406
Sixth Sense, The (1999)	1940.000	4.109089	17504
Apollo 13 (1995)	1931.071	3.885789	24284
Toy Story (1995)	1889.214	3.927638	23790
Silence of the Lambs, The (1991)	1870.444	4.204101	30382
Fargo (1996)	1830.308	4.134821	21395
Shrek (2001)	1816.625	3.899066	13063
Fugitive, The (1993)	1809.438	4.009155	25998
Pirates of the Caribbean: The Curse of the Black Pearl (2003)	1774.000	3.893930	9555
12 Monkeys (Twelve Monkeys) (1995)	1742.643	3.874743	21891
Saving Private Ryan (1998)	1720.364	4.083903	17103
Gladiator (2000)	1717.444	3.936975	13931
Usual Suspects, The (1995)	1716.929	4.365854	21648

```
# check for movies with the lowest number of ratings per year
least_rated <- edx %>%
  group_by(title) %>%
  summarize(mean_rate = mean(rate),
            mean_rating= mean(rating), n=n()) %>%
  top_n(-25, mean_rate) %>%
  arrange(desc(mean_rate))

# print table
least_rated %>% kable(format = 'markdown')
```

title	mean_rate	mean_rating	n
Bell Boy, The (1918)	0.0219780	3.5	2
Deadly Companions, The (1961)	0.0208333	4.0	1
Black Tights (1-2-3-4 ou Les Collants noirs) (1960)	0.0204082	3.0	1
Cruel Story of Youth (Seishun zankoku monogatari) (1960)	0.0204082	2.5	1
Testament of Orpheus, The (Testament d'OrphÃ©e) (1960)	0.0204082	4.5	1
Face of a Fugitive (1959)	0.0200000	3.0	1
Du cÃ´tÃ© de la cÃ´te (1958)	0.0196078	2.5	1
Music Room, The (Jalsaghar) (1958)	0.0196078	4.0	1
Double Dynamite (1951)	0.0172414	2.0	1
Gold Raiders (1951)	0.0172414	2.0	1
Man of Straw (Untertan, Der) (1951)	0.0172414	4.0	1
Borderline (1950)	0.0169492	3.0	1
Variety Lights (Luci del varietÃ) (1950)	0.0169492	4.0	1
Malaya (1949)	0.0166667	3.0	1
Long Night, The (1947)	0.0161290	3.0	1
Rockin' in the Rockies (1945)	0.0156250	2.0	1
Battle of Russia, The (Why We Fight, 5) (1943)	0.0151515	3.5	1
Chapayev (1934)	0.0133333	1.5	1
Death Kiss, The (1933)	0.0131579	2.5	1
Blue Light, The (Das Blaue Licht) (1932)	0.0129870	5.0	1
One Hour with You (1932)	0.0129870	3.0	1
Gaucho, The (1927)	0.0121951	3.5	1
Mr. Wu (1927)	0.0121951	3.0	1
Ace of Hearts, The (1921)	0.0113636	3.5	1
Father Sergius (Otets Sergiy) (1917)	0.0108696	3.0	1

The 25 movies with the highest number of ratings per year seem to be generally rated higher than average with a lot of 4-star ratings and only 1 movie rating lower than 3.5 stars. The least rated movies on the other hand seem to have a higher variance and are rated in a range from 1.5 to 5 stars. This high variance is explained by the low number of ratings (1 rating for each movie except for the movie "The Bell Boy").

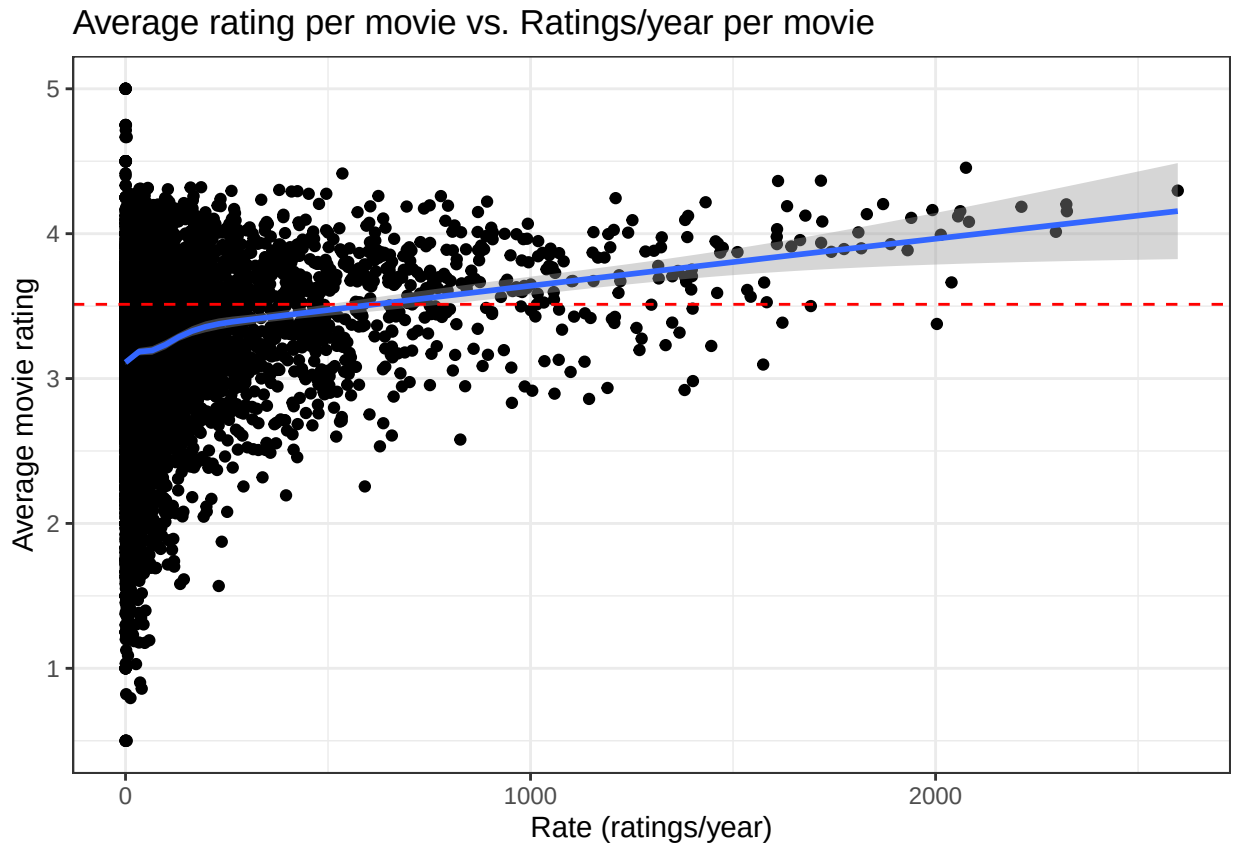
Indeed, the 25 movies with the highest number of ratings per year have an average rating of 4.05 stars, whereas the least rated movies have an average rating as low as 3.14.

```
# compare average rating of most 25 and least 25 rated movies
data.frame(most_rated = mean(most_rated$mean_rating),
           least_rated =mean(least_rated$mean_rating))
```

```
##   most_rated least_rated
## 1   4.047416        3.14
```

This trend is confirmed for the whole population by plotting the rating versus the rate (number of ratings per year) which generally shows that movies with more than ~600 ratings per year are ranked above average as they most likely represent popular movies and movies with fewer ratings per year are ranked below average as these are rather obscure movies. The plot also shows a higher variance of ratings for low rates as these ratings often represent the opinion of only a handful of individuals and are therefore representative for the whole population.

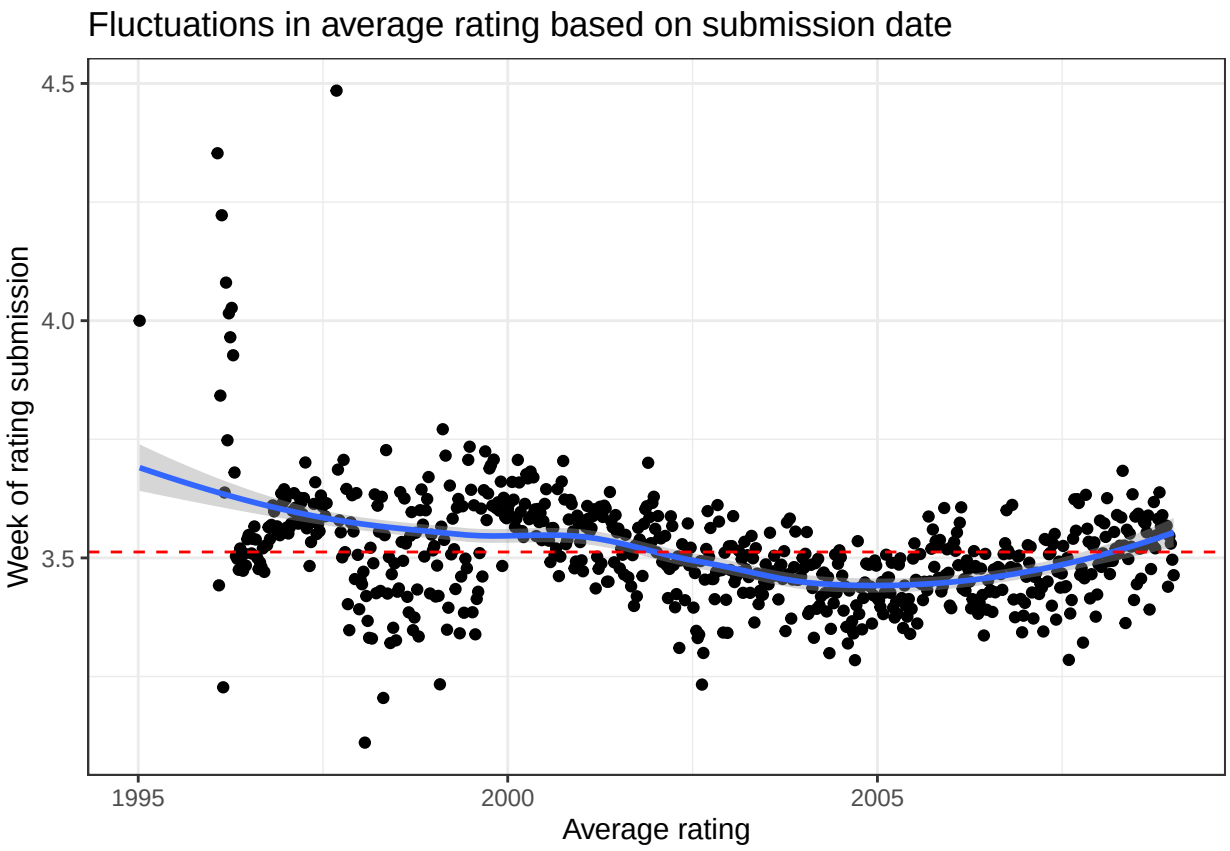
```
# Plot rating of the movies versus the rate per year
edx %>%
  group_by(movieId) %>%
  summarize(mean_rate = mean(rate),
            mean_rating = mean(rating)) %>%
  ggplot(aes(mean_rate, mean_rating)) +geom_point()+
  geom_smooth() +
  geom_hline(yintercept = mu, lty =2,color="red") +
  theme_bw() +
  xlab("Rate (ratings/year)")+
  ylab("Average movie rating") +
  ggtitle("Average rating per movie vs. Ratings/year per movie")
```



2.1.5 Time Effect

Plotting rating versus the date of submission date of the rating reveals a time dependent effect of movie ratings. Until ~2001 users rated movies slightly above average; afterwards users were more critical and gave below average ratings. At the end of the data gathering period (2008), the trend converged towards the average and users started to rate movies above average again. These changes look like dynamic fluctuations around the average and might be due to differences how the population perceives movies in general at a given time. It should also be noted that this time data was only gathered for the period of 1995-2008.

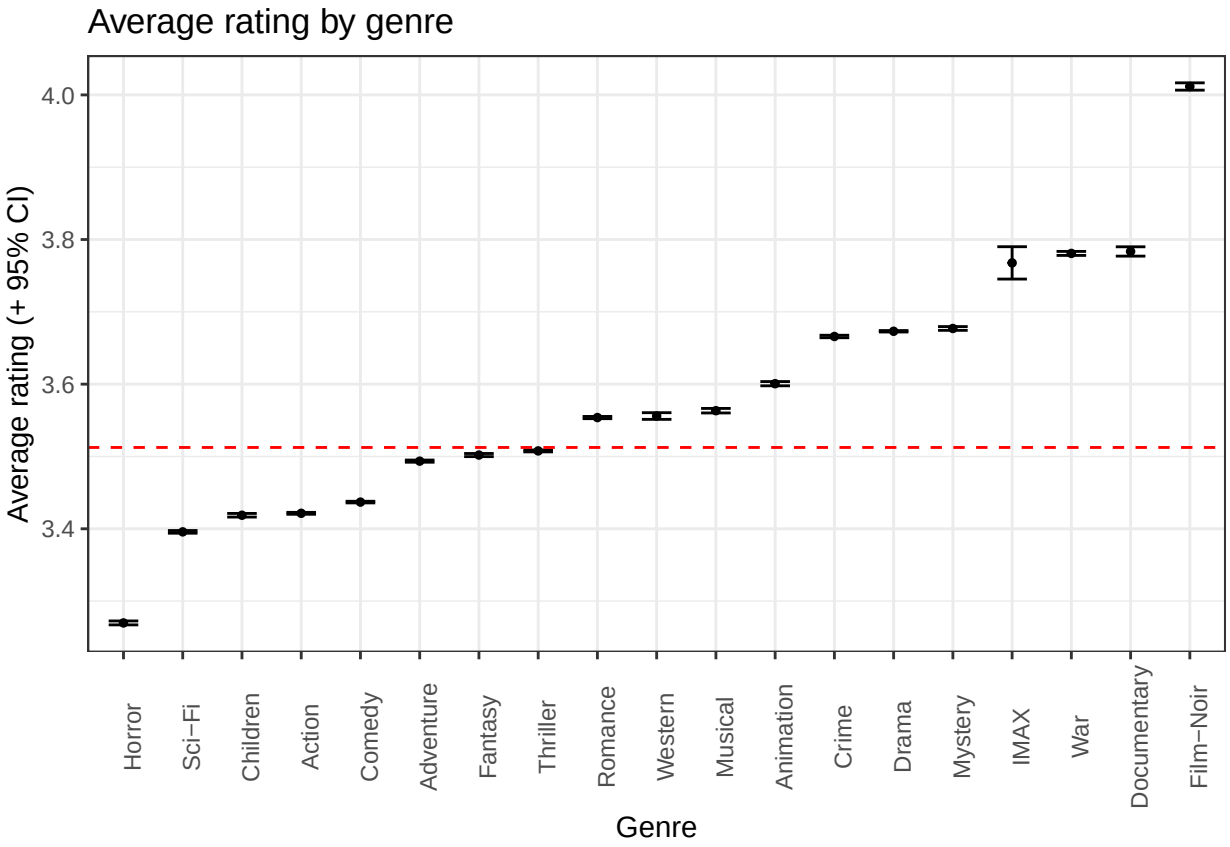
```
#### Time Effect
# Plot showing bias based on when the data was submitted
edx %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth() +
  geom_hline(yintercept = mu, lty =2,color="red") +
  theme_bw()+
  xlab("Average rating") +
  ylab("Week of rating submission") +
  ggtitle("Fluctuations in average rating based on submission date")
```



2.1.6 Genre Effect

The last parameter that was analysed for an effect on movie ratings was the movie genre. The genre column is comprised of 797 unique genre combinations as previously explained. These combinations are formed by 19 distinct "base" genres such as Drama, Action, Comedy, etc. The following plot shows the average rating for a given "base" genre with error bars representing 95 % confidence intervals and indicates a genre effect with Horror movies ranking the lowest and movies of the Film-Noir genre ranking the highest.

```
#### Genre
# as str_split on the whole data.frame requires too much RAM this is a workaround
# to extract mean and 95% ci for each respective genre

# extract genre names
genre_names <- unique(edx$genres) %>%
  str_split("\\|") %>%
  unlist() %>%
  unique() %>%
  .[-length(.)]

# apply function over each genre to calculate mean and 95% ci
stat_genre <- sapply(genre_names, function(x){
edx %>%
  filter(str_detect(edx$genres, x)) %>%
  # the confidence intervals could also be calculated by a bootstrap approach but were #
  # using way to much computation time. Although data is not normal, based on CLT this #
  # approximation of the cis should work well                                           #
  summarize(ci = list(mean_cl_normal(rating))) %>%
  unlist()
})%>%
  t() %>%
  data.frame() %>%
  rownames_to_column("Genre")

# plot average genre rating with 95% confidence intervals
stat_genre %>% mutate(Genre = reorder(Genre, ci.y)) %>%
    ggplot(aes(Genre, ci.y)) +
    geom_point(size=1) +
    geom_errorbar(aes(ymin = ci.ymin, ymax = ci.ymax),width=0.5)  +
    geom_hline(yintercept = mu, lty =2,color="red") +
    theme_bw() +
    theme(axis.text.x = element_text(angle = 90)) +
    ylab("Average rating (+ 95% CI)") +
    ggtitle("Average rating by genre")
```


Average rating by genre
Average rating (+ 95% CI)
4.0
3.8
3.6
3.4
Horror
Sci−Fi
Children
Action
Comedy
Adventure
Fantasy
Thriller
Romance
Western
Musical
Animation
Crime
Drama
Mystery
IMAX
War
Documentary
Film−Noir
Genre

2.2 Modelling Approach

The aim of the modelling approach was to minimize the RMSE as described in the introduction. To compare modelling approaches to each other, a reference model needs to be definded. In this case this baseline model was the prediction of the average μ for each rating. Afterwards the different effects were added to analyze an improvement over this baseline model using the insights gained in the previous chapter.

The exploratory data analysis demonstrated that there are a lot of movie ratings with only few observations, leading to inflated errors as squared distances were calculated. Therefore a regularization approach was applied to minimize the effect of these estimates generated by small sample sizes. Regularization shrinks errors of small sample sizes towards zero by adding a penalty term without significantly affecting observations obtained by large sample sizes and therefore yields a more robust and meaningful estimate.

2.2.1 Splitting edx into training and test set for model tuning

To perform cross-validation and define the best tuning parameters (if present in the respective model) the edx dataset was split into training and test set using the caret pipeline.

```
# split edx data into training and test set for parameter tuning
set.seed(12)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

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

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

rm(temp, removed)
```

2.2.2 Baseline Model

The baseline model used as reference was to predict the average μ of the training set for each rating. This model explains all differences by random variation:

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

where $\epsilon_{u,i}$ is the independent error from the same distribution centered at 0

Using this model a naive RMSE was obtained:

```
# predict all values to be mean mu of training set
mu_hat <- mu
naive_rmse <- RMSE(test_set$rating, mu_hat)
naive_rmse
```

```
## [1] 1.060842
```

```
# show results in table
rmse_results <- data_frame(method = "Just the average", RMSE = naive_rmse)
rmse_results %>% kable(format = 'markdown')
```

method	RMSE
Just the average	1.060842

2.2.3 Movie Model

In the data exploration section, differences in ratings for different movies could be observed, indicating a movie effect. The model incorporating this effect is:

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

where b_i is the movie effect or movie bias for each respective movie. A negative bias b_i indicates a bad movie therefore ranking below average. A positive bias corresponds to a good movie ranking above average. Therefore this model better represents the "true" rating of each movie shown by an improvement of the RMSE over the baseline model:

```
# define model taking movie bias into account
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_movie = mean(rating - mu))

# prediction of ratings based on movie bias
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_movie

# calculate RMSE for movie model
model_2_rmse <- RMSE(test_set$rating, predicted_ratings)

# bind results to rmse table
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = model_2_rmse ))
rmse_results %>% kable(format = 'markdown')
```

method	RMSE
Just the average	1.0608425
Movie Effect Model	0.9438581

2.2.4 Movie + User Model

Another insight gained during data exploration was the difference in ratings by user as some users tend to enjoy most movies and rate movies generally higher, whereas other users are more critical and rate movies generally lower than average. To account for these user specific differences the following model was developed:

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

where b_u is the user bias with a positive b_u corresponding to a generous user and a negative b_u indicating a critical user. The combination of movie and user bias greatly improved model performance:

```
#### 3.: User + Movie Model
# define model taking movie and user bias into account
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_user = mean(rating - mu - b_movie))

# prediction of ratings based on movie and user bias
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_movie + b_user) %>%
  .$pred

# calculate RMSE for movie model
model_3_rmse <- RMSE(test_set$rating, predicted_ratings)

# bind results to rmse table
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effects Model",
                                     RMSE = model_3_rmse ))
rmse_results %>% kable(format = 'markdown')
```

method	RMSE
Just the average	1.0608425
Movie Effect Model	0.9438581
Movie + User Effects Model	0.8661512

2.2.5 Regularized Movie + User Model

To further improve model performance, regularization was applied to minimize error inflation caused by small sample sizes. The regularisation formula for movie bias $b_i(\lambda)$ for a given λ is:

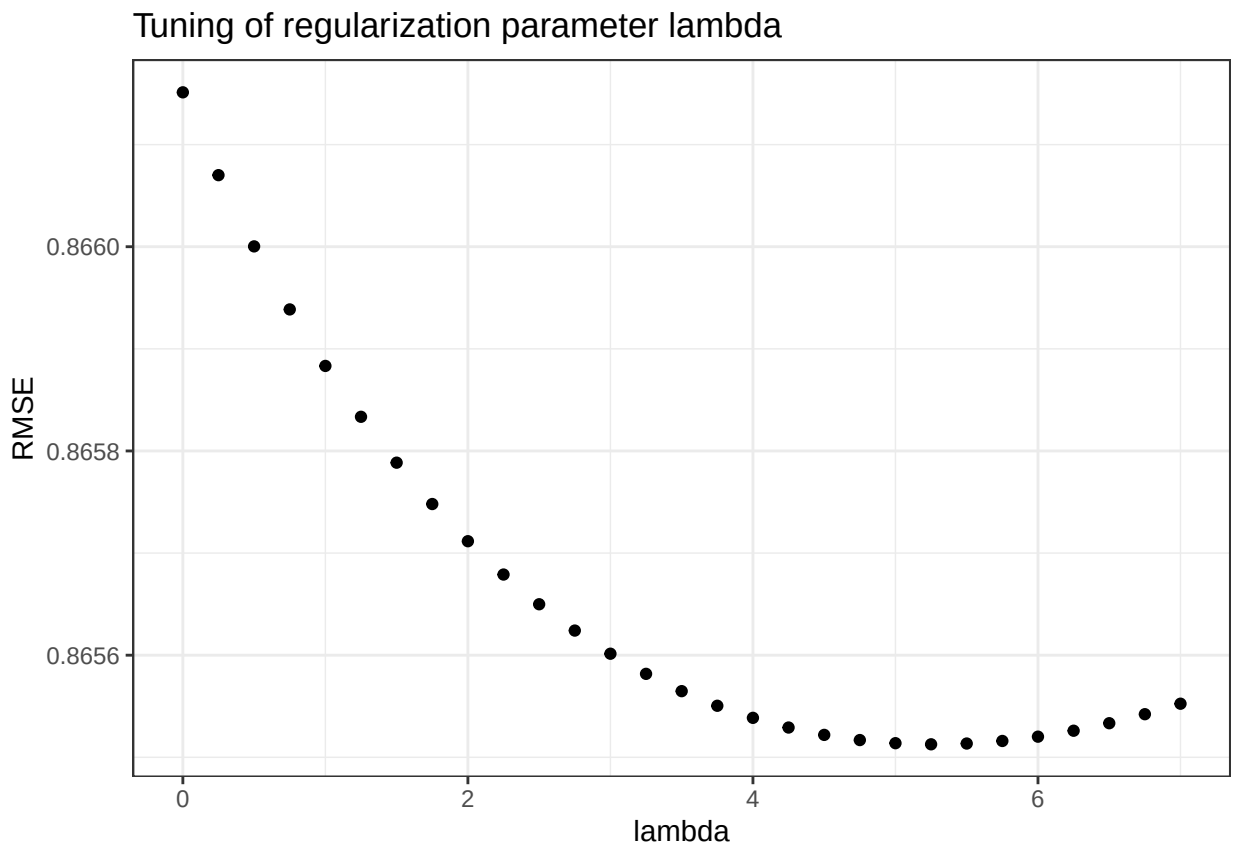
$$b_i(\lambda) = \frac{1}{\lambda + ni} \sum (Y_i - \mu)$$

where n_i is the number of ratings per movie i and λ is a hyperparameter that is object to tuning by a least square approach. Analogously the regularized user bias $b_u(\lambda)$ can be calculated as:

$$b_u(\lambda) = \frac{1}{\lambda + n_u} \sum (Y_i - (\mu + b_i(\lambda)))$$

using the same λ as for the regularized movie bias with n_u being the number of movies user u rated. The best hyperparameter λ was chosen by training the model using different values for λ and chosing that value λ which minimzes the RMSE on the test set.

```
# Regularized Movie and User Model

# tuning hyperparameter lambda using training set
lambdas <- seq(0, 7, 0.25)

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

  #Calculate the mean of ratings from the train_set training set
  mu <- mean(train_set$rating)

  #Adjust mean by movie effect and penalize low number on ratings
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  #ajdust mean by user and movie effect and penalize low number of ratings
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  #predict ratings in the training set to derive optimal penalty value 'lambda'
  predicted_ratings <-   test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

  return(RMSE(test_set$rating, predicted_ratings))
})

# plot rmses vs lambdas
data.frame(lambda = lambdas, rmse = rmses) %>%
  ggplot(aes(lambda, rmse)) +
  geom_point() +
  theme_bw() +
  ylab("RMSE") +
  ggtitle("Tuning of regularization parameter lambda")
```



The optimal value for λ using this model was $\lambda = 5.25$ resulting in an RMSE of 0.8655127:

```
# extract best value lambda
model_4_lambda  <- lambdas[which.min(rmses)]

# extract RMSE of the model using the optimal lambda
model_4_rmse <- min(rmses)

# bind results to the RMSE table
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie + User Effect Model",
                                     RMSE = model_4_rmse ))
rmse_results %>% kable(format = 'markdown')
```

method	RMSE
Just the average	1.0608425
Movie Effect Model	0.9438581
Movie + User Effects Model	0.8661512
Regularized Movie + User Effect Model	0.8655127

2.2.6 Regularized Movie + User + Age+ Genre+ Rate + Time Model

The last model tested, used the variables *movieId*, *userId*, *genres*, *age*, *rate* and *date* as data exploration suggested an influence on ratings by all of these parameters. On top of that regularization was applied to minimize small sample errors. The model is defined as follows:

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

where b_g is the genre bias: $\sum_{k=1}^{K}(X_{u,i}\beta_k)$ (with $x_{u,i}^k = 1$ if $g_{u,i}$ is genre k),
b_a is the age of movie bias: $b_a = f(a_i)$ (with $f(a_i)$ a smooth function of a_i),
b_r is the rate (ratings/year) per movie: $b_r = f(r_i)$ (with $f(r_i)$ a smooth function of r_i)
and b_t is the date user u rated movie i: $b_t = f(t_{u,i})$ (with $f(t_{u,i})$ a smooth function of t_{u,i}).

The regularized effects were obtained as follows:

Regularized genre effect:

$$b_g(\lambda) = \frac{1}{\lambda + n_g}\sum(Y_i - (\mu + b_i(\lambda) + b_u(\lambda)))$$

where n_g is the number of ratings per genre.

Regularized age of movie effect:

$$b_a(\lambda) = \frac{1}{\lambda + n_a}\sum(Y_i - (\mu + b_i(\lambda) + b_u(\lambda) + b_g(\lambda)))$$

where n_a is the number of ratings per movie age.

Regularized rate effect:

$$b_r(\lambda) = \frac{1}{\lambda + n_r}\sum(Y_i - (\mu + b_i(\lambda) + b_u(\lambda) + b_g(\lambda) + b_a(\lambda)))$$

where n_r is the number of ratings per rate.

Regularized date of submission effect:

$$b_t(\lambda) = \frac{1}{\lambda + n_t}\sum(Y_i - (\mu + b_i(\lambda) + b_u(\lambda) + b_g(\lambda) + b_a(\lambda) + b_r(\lambda)))$$

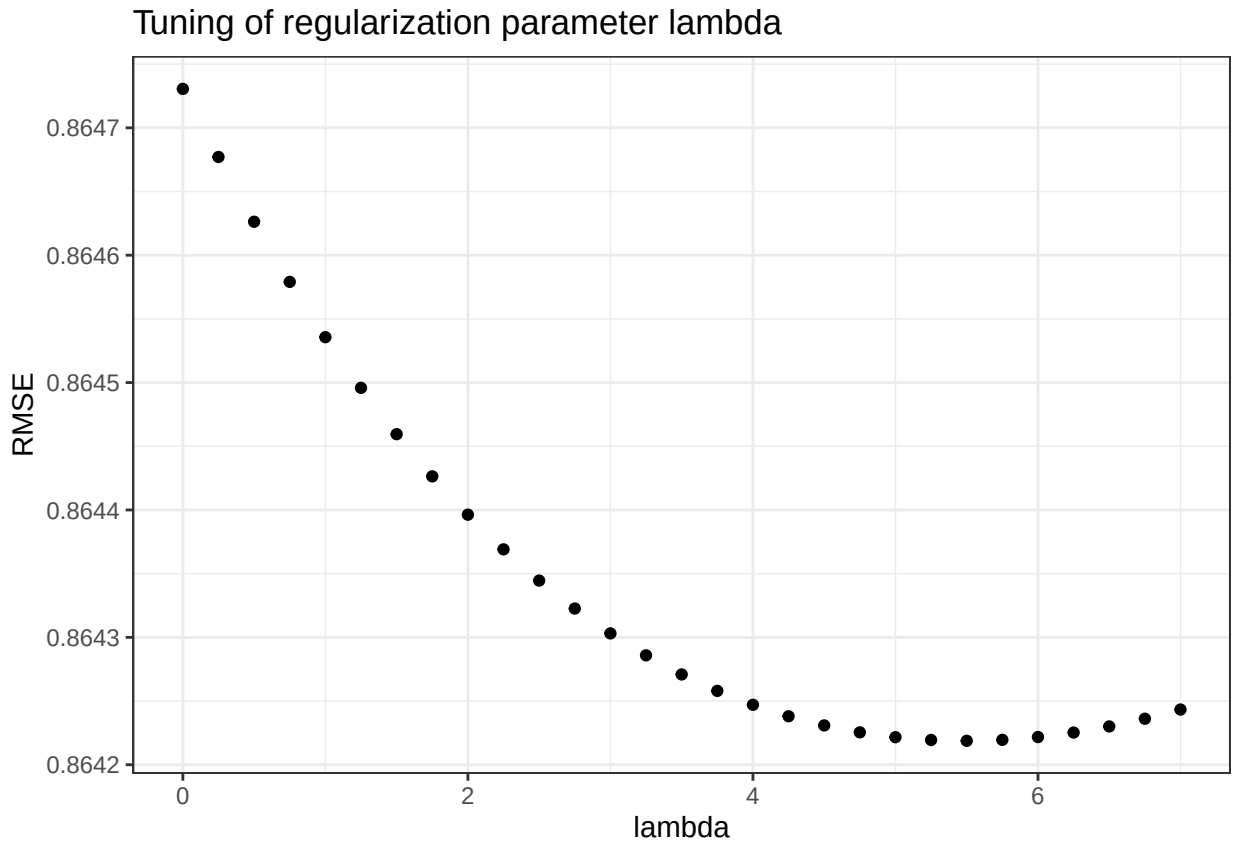
where n_t is the number of ratings per submission date.

The hyperparameter λ was tuned and the optimal λ was chosen as described previously.

```
### Regularized Movie + User + Age+ Genre+ Rate + Time Effect Model
lambdas <- seq(0, 7, 0.25)


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

  #Calculate the mean of ratings from the train_set training set
  mu <- mean(train_set$rating)

  #Adjust mean by movie effect and penalize low number on ratings
  b_i <- train_set %>%
    group_by(movieId) %>%
```

```
  summarize(b_i = sum(rating - mu)/(n()+l))

#Ajdust mean by user and movie effect and penalize low number of ratings
b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+l))

#Ajdust mean by genre, user and movie effect and penalize low number of ratings
b_g <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+l))

#Ajdust mean by age, genre, user and movie effect and penalize low number of ratings
b_a <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by="genres") %>%
  group_by(age) %>%
  summarize(b_a = sum(rating - b_i - b_u- b_g - mu)/(n()+l))

#Ajdust mean by rate, time, age, genre, user and movie effect and penalize low number
# of ratings
b_r <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by="genres") %>%
  left_join(b_a, by="age") %>%
  group_by(rate,movieId) %>%
  summarize(b_r = sum(rating - b_i - b_u- b_g - b_a- mu)/(n()+l))

#Ajdust mean by time, age, genre, user and movie effect and penalize low number of ratings
b_t <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by="genres") %>%
  left_join(b_a, by="age") %>%
  left_join(b_r, by="movieId")  %>%
  group_by(date) %>%
  summarize(b_t = sum(rating - b_i - b_u- b_g - b_a - b_r - mu)/(n()+l))

test_set %>%
  left_join(b_i, by="movieId") %>% filter(is.na(b_i)) %>% .$b_i %>% sum()

#predict ratings in the training set to derive optimal penalty value 'lambda'
predicted_ratings <- test_set %>%
  left_join(b_i, by="movieId") %>% mutate(b_i = ifelse(is.na(b_i),0,b_i)) %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by="genres") %>%
  left_join(b_a, by="age") %>%
  left_join(b_r, by="movieId") %>%mutate(b_r = ifelse(is.na(b_r),0,b_r)) %>%
```

```
    left_join(b_t, by="date") %>%
    mutate(pred = mu + b_i + b_u + b_g +b_a+b_r+b_t) %>%
    .$pred

  return(RMSE(test_set$rating, predicted_ratings))
})


# plot rmses vs lambdas
data.frame(lambda = lambdas, rmse = rmses) %>%
  ggplot(aes(lambda, rmse)) +
  geom_point() +
  theme_bw() +
  ylab("RMSE") +
  ggtitle("Tuning of regularization parameter lambda")
```



Using this approach, the RMSE could be further minimized. Modelling approaches leaving out one or more of the parameters from the previous model were also tested but did not yield any better results and were therefore left out to increase lucidity.

```
# extract optimal lambda
model_5_lambda <- lambdas[which.min(rmses)]

# extract RMSE of the model using the optimal lambda
model_5_rmse <- min(rmses)
```

```r
# bind results to the RMSE table
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized  Movie + User + Genre + Age + Rate + Time Effe
                                     RMSE = model_5_rmse ))
rmse_results %>% kable(format = 'markdown')
```

method	RMSE
Just the average	1.0608425
Movie Effect Model	0.9438581
Movie + User Effects Model	0.8661512
Regularized Movie + User Effect Model	0.8655127
Regularized Movie + User + Genre + Age + Rate + Time Effect Model	0.8642188

3 Results

The model using *movieId*, *userId*, *genres*, *age*, *rate* and *date* performed best on the training set. Thus, this model was retrained on the whole edx dataset with the optimal value of $\lambda = 5.5$ determined by cross-validation. Subsequently this model was applied to the validation dataset to determine the out of sample error for this model. With an RMSE of 0.8634613 the final model improved predictions by 18.54 % compared to the baseline model.

```
# optimal tuning parameter lambda (acquired in the last paragraph of the methods section)
lambda <- model_5_lambda

# calculate RMSE on validation set
predicted_ratings <- sapply(lambda,function(l){

  #Calculate the mean of ratings from the train_set training set
  mu <- mean(edx$rating)

  #Adjust mean by movie effect and penalize low number on ratings
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  #Ajdust mean by user and movie effect and penalize low number of ratings
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  #Ajdust mean by genre, user and movie effect and penalize low number of ratings
  b_g <- edx %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+l))

  # Ajdust mean by age, genre, user and movie effect and penalize low number of ratings
  b_a <- edx %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_g, by="genres") %>%
    group_by(age) %>%
    summarize(b_a = sum(rating - b_i - b_u- b_g - mu)/(n()+l))

  # Ajdust mean by rate effect and penalize low number of ratings
  b_r <- edx %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_g, by="genres") %>%
    left_join(b_a, by="age") %>%
    group_by(rate,movieId) %>%
    summarize(b_r = sum(rating - b_i - b_u- b_g - b_a- mu)/(n()+l))

  # Ajdust mean by time effect and penalize low number of ratings
  b_t <- edx %>%
```

```
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_g, by="genres") %>%
    left_join(b_a, by="age") %>%
    left_join(b_r, by=c("rate","movieId"))%>%
    group_by(date) %>%
    summarize(b_t = sum(rating - b_i - b_u- b_g - b_a - b_r - mu)/(n()+l))

  #predict ratings in the validation set with the final value of lambda
  predicted_ratings <- validation %>%
    left_join(b_i, by="movieId") %>%  #mutate(b_i = ifelse(is.na(b_i),0,b_i)) %>%
    left_join(b_u, by="userId") %>%
    left_join(b_g, by="genres") %>%
    left_join(b_a, by="age") %>%
    left_join(b_r, by="movieId") %>%  #mutate(b_r = ifelse(is.na(b_r),0,b_r)) %>%
    left_join(b_t, by="date") %>%
    mutate(pred = mu +  b_i + b_u + b_g + b_a+ b_r + b_t) %>%
    .$pred

   return(predicted_ratings)
})

#calculate RMSE
model_5_validation  <- RMSE(validation$rating, predicted_ratings)

# bind results to the RMSE table
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Validation: Regularized  Movie + User + Genre +
                                     Age + Rate + Time Effect Model",
                                     RMSE = model_5_validation))
rmse_results %>% kable(format = 'markdown')
```

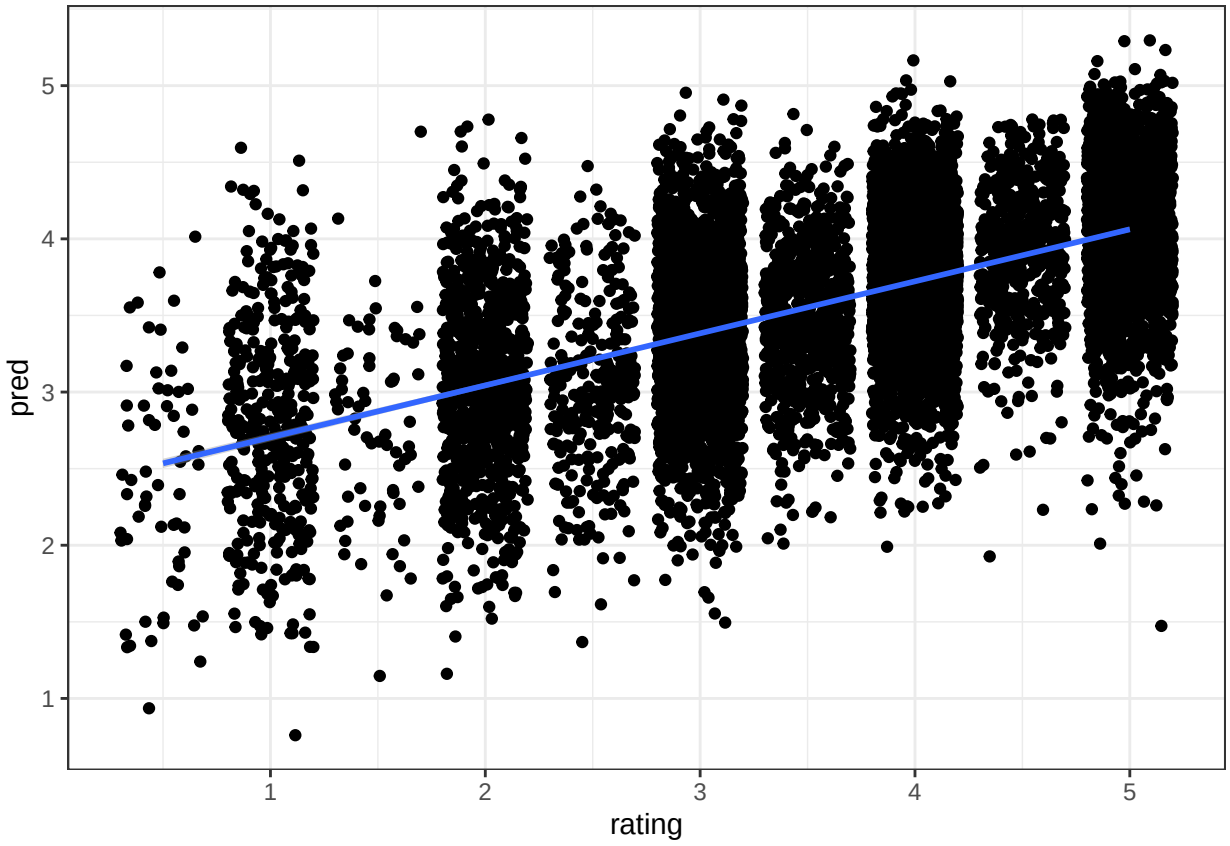
method	RMSE
Just the average	1.0608425
Movie Effect Model	0.9438581
Movie + User Effects Model	0.8661512
Regularized Movie + User Effect Model	0.8655127
Regularized Movie + User + Genre + Age + Rate + Time Effect Model	0.8642188
Validation: Regularized Movie + User + Genre + Age + Rate + Time Effect Model	0.8634613

Furthermore the pearson correlation between predicted and observed ratings was calculated to analyze if the general movie taste could be correctly predicted. A correlation coefficient of 0.581 was obtained suggesting a strong correlation between predicted and observed ratings. This correlation and a linear relationship between the two variables is depicted for visualization purposes for the first 10,000 entries.

```
# calculate correlation of predicted and observed ratings
corr <- cor(validation$rating, predicted_ratings)
colnames(corr) <- "pearson correlation coefficent"
corr %>% round(3) %>% kable(format = 'markdown')
```

pearson correlation coefficent
0.581

```
# plot correlation of predicted and observed ratings (for visualization purposes: only the first 10000 o
validation[1:10000,] %>%
  mutate(pred = predicted_ratings[1:10000]) %>%
  ggplot(aes(rating, pred)) + geom_jitter() +
  geom_smooth(method="lm") +
  theme_bw()
```



4 Conclusion

The aim of this project was to develop an algorithm improving predictions over a baseline model (baseline model: suggesting the average rating μ and explaining differences by random variation) and achieving an RMSE < 0.86490. Incorporation of six different bias terms - using the features *movieId*, *userId*, *genres*, *age*, *rate* and *date* - into the baseline model combined with a regularization approach lead to an improvement of 18.54 % over the baseline model with a final RMSE of 0.8634613, thus fulfilling the requirements for a successful model regarding the class guidelines.

Although the results still suggest a relatively large error concerning predictions of the acutal rating, correlation analysis revealed a strong correlation of 0.581 between predicted and observed ratings, indicating that the general movie taste can be predicted confidently. As the goal of a movie recommendation system is to suggest movies a user might enjoy it does not matter too much to not predict the actual rating correctly as long as the general direction is correct.

This applies especially for very high or very low actual ratings as a rating of 0.5 or 1.5 is both considered very bad and a rating of 4 or 5 stars is considered great. However, the model might fail to suggest movies correctly if the actual rating is somewhere around 2.5 or 3 stars as this can be seen as a threshold between good and bad movies.

In summary, the developed model fulfills the purpose to recommend movies based on different features in a satisfactory manner. The recommendation system, however, could be improved by application of different algorithms like *knn* or neural networks, if the computational capacity was available.

5 Appendix

5.1 References

[1] https://www.netflixprize.com/
[2] https://courses.edx.org/courses/course-v1:HarvardX+PH125.8x+1T2020/course

5.2 Used Packages

- Hmisc
  - summary of useful functions for data analysis
- tidyverse
  - Uniform way to process, clean, transform, . . . , data
- data.table
  - Handling big datasets
- caret
  - Provides machine learning pipelines
- doParallel
  - Allowing R to use all CPU cores available
- lubridate
  - Facilitates handling time data
- kableExtra
  - Display of tables in the Report

5.3 Dataset

The MovieLens data set is available at https://grouplens.org/datasets/movielens/10m/. The R script, R markdown file and this report can be accessed at https://github.com/MBender1992/MovieLens_Edx.

5.4 Session Info

```
## - Session info ---------------------------------------------------------------
##  setting  value
##  version  R version 4.0.0 (2020-04-24)
##  os       Windows 10 x64
##  system   x86_64, mingw32
##  ui       RTerm
##  language (EN)
##  collate  German_Germany.1252
##  ctype    German_Germany.1252
##  tz       Europe/Berlin
##  date     2020-05-13
##
## - Packages -------------------------------------------------------------------
##  package      * version    date       lib source
##  acepack        1.4.1      2016-10-29 [1] CRAN (R 4.0.0)
##  assertthat     0.2.1      2019-03-21 [1] CRAN (R 4.0.0)
##  backports      1.1.6      2020-04-05 [1] CRAN (R 4.0.0)
##  base64enc      0.1-3      2015-07-28 [1] CRAN (R 4.0.0)
##  broom          0.5.6      2020-04-20 [1] CRAN (R 4.0.0)
##  caret        * 6.0-86     2020-03-20 [1] CRAN (R 4.0.0)
##  cellranger     1.1.0      2016-07-27 [1] CRAN (R 4.0.0)
##  checkmate      2.0.0      2020-02-06 [1] CRAN (R 4.0.0)
##  class          7.3-16     2020-03-25 [2] CRAN (R 4.0.0)
##  cli            2.0.2      2020-02-28 [1] CRAN (R 4.0.0)
##  cluster        2.1.0      2019-06-19 [2] CRAN (R 4.0.0)
##  codetools      0.2-16     2018-12-24 [2] CRAN (R 4.0.0)
##  colorspace     1.4-1      2019-03-18 [1] CRAN (R 4.0.0)
##  crayon         1.3.4      2017-09-16 [1] CRAN (R 4.0.0)
##  data.table   * 1.12.8     2019-12-09 [1] CRAN (R 4.0.0)
##  DBI            1.1.0      2019-12-15 [1] CRAN (R 4.0.0)
##  dbplyr         1.4.3      2020-04-19 [1] CRAN (R 4.0.0)
##  digest         0.6.25     2020-02-23 [1] CRAN (R 4.0.0)
##  doParallel   * 1.0.15     2019-08-02 [1] CRAN (R 4.0.0)
##  dplyr        * 0.8.5      2020-03-07 [1] CRAN (R 4.0.0)
##  ellipsis       0.3.0      2019-09-20 [1] CRAN (R 4.0.0)
##  evaluate       0.14       2019-05-28 [1] CRAN (R 4.0.0)
##  fansi          0.4.1      2020-01-08 [1] CRAN (R 4.0.0)
##  farver         2.0.3      2020-01-16 [1] CRAN (R 4.0.0)
##  forcats      * 0.5.0      2020-03-01 [1] CRAN (R 4.0.0)
##  foreach      * 1.5.0      2020-03-30 [1] CRAN (R 4.0.0)
##  foreign        0.8-78     2020-04-13 [2] CRAN (R 4.0.0)
##  Formula      * 1.2-3      2018-05-03 [1] CRAN (R 4.0.0)
##  fs             1.4.1      2020-04-04 [1] CRAN (R 4.0.0)
##  generics       0.0.2      2018-11-29 [1] CRAN (R 4.0.0)
##  ggplot2      * 3.3.0      2020-03-05 [1] CRAN (R 4.0.0)
##  glue           1.4.0      2020-04-03 [1] CRAN (R 4.0.0)
##  gower          0.2.1      2019-05-14 [1] CRAN (R 4.0.0)
##  gridExtra      2.3        2017-09-09 [1] CRAN (R 4.0.0)
##  gtable         0.3.0      2019-03-25 [1] CRAN (R 4.0.0)
##  haven          2.2.0      2019-11-08 [1] CRAN (R 4.0.0)
##  highr          0.8        2019-03-20 [1] CRAN (R 4.0.0)
##  Hmisc        * 4.4-0      2020-03-23 [1] CRAN (R 4.0.0)
```

```
##  hms             0.5.3      2020-01-08 [1] CRAN (R 4.0.0)
##  htmlTable       1.13.3     2019-12-04 [1] CRAN (R 4.0.0)
##  htmltools       0.4.0      2019-10-04 [1] CRAN (R 4.0.0)
##  htmlwidgets     1.5.1      2019-10-08 [1] CRAN (R 4.0.0)
##  httr            1.4.1      2019-08-05 [1] CRAN (R 4.0.0)
##  ipred           0.9-9      2019-04-28 [1] CRAN (R 4.0.0)
##  iterators     * 1.0.12     2019-07-26 [1] CRAN (R 4.0.0)
##  jpeg            0.1-8.1    2019-10-24 [1] CRAN (R 4.0.0)
##  jsonlite        1.6.1      2020-02-02 [1] CRAN (R 4.0.0)
##  kableExtra    * 1.1.0      2019-03-16 [1] CRAN (R 4.0.0)
##  knitr           1.28       2020-02-06 [1] CRAN (R 4.0.0)
##  labeling        0.3        2014-08-23 [1] CRAN (R 4.0.0)
##  lattice       * 0.20-41    2020-04-02 [2] CRAN (R 4.0.0)
##  latticeExtra    0.6-29     2019-12-19 [1] CRAN (R 4.0.0)
##  lava            1.6.7      2020-03-05 [1] CRAN (R 4.0.0)
##  lifecycle       0.2.0      2020-03-06 [1] CRAN (R 4.0.0)
##  lubridate     * 1.7.8      2020-04-06 [1] CRAN (R 4.0.0)
##  magrittr        1.5        2014-11-22 [1] CRAN (R 4.0.0)
##  MASS            7.3-51.5   2019-12-20 [2] CRAN (R 4.0.0)
##  Matrix          1.2-18     2019-11-27 [2] CRAN (R 4.0.0)
##  mgcv            1.8-31     2019-11-09 [2] CRAN (R 4.0.0)
##  ModelMetrics    1.2.2.2    2020-03-17 [1] CRAN (R 4.0.0)
##  modelr          0.1.7      2020-04-30 [1] CRAN (R 4.0.0)
##  munsell         0.5.0      2018-06-12 [1] CRAN (R 4.0.0)
##  nlme            3.1-147    2020-04-13 [2] CRAN (R 4.0.0)
##  nnet            7.3-13     2020-02-25 [2] CRAN (R 4.0.0)
##  pillar          1.4.4      2020-05-05 [1] CRAN (R 4.0.0)
##  pkgconfig       2.0.3      2019-09-22 [1] CRAN (R 4.0.0)
##  plyr            1.8.6      2020-03-03 [1] CRAN (R 4.0.0)
##  png             0.1-7      2013-12-03 [1] CRAN (R 4.0.0)
##  pROC            1.16.2     2020-03-19 [1] CRAN (R 4.0.0)
##  prodlim         2019.11.13 2019-11-17 [1] CRAN (R 4.0.0)
##  purrr         * 0.3.4      2020-04-17 [1] CRAN (R 4.0.0)
##  R6              2.4.1      2019-11-12 [1] CRAN (R 4.0.0)
##  RColorBrewer    1.1-2      2014-12-07 [1] CRAN (R 4.0.0)
##  Rcpp            1.0.4.6    2020-04-09 [1] CRAN (R 4.0.0)
##  readr         * 1.3.1      2018-12-21 [1] CRAN (R 4.0.0)
##  readxl          1.3.1      2019-03-13 [1] CRAN (R 4.0.0)
##  recipes         0.1.12     2020-05-01 [1] CRAN (R 4.0.0)
##  reprex          0.3.0      2019-05-16 [1] CRAN (R 4.0.0)
##  reshape2        1.4.4      2020-04-09 [1] CRAN (R 4.0.0)
##  rlang           0.4.6      2020-05-02 [1] CRAN (R 4.0.0)
##  rmarkdown       2.1        2020-01-20 [1] CRAN (R 4.0.0)
##  rpart           4.1-15     2019-04-12 [2] CRAN (R 4.0.0)
##  rstudioapi      0.11       2020-02-07 [1] CRAN (R 4.0.0)
##  rvest           0.3.5      2019-11-08 [1] CRAN (R 4.0.0)
##  scales          1.1.0      2019-11-18 [1] CRAN (R 4.0.0)
##  sessioninfo     1.1.1      2018-11-05 [1] CRAN (R 4.0.0)
##  stringi         1.4.6      2020-02-17 [1] CRAN (R 4.0.0)
##  stringr       * 1.4.0      2019-02-10 [1] CRAN (R 4.0.0)
##  survival      * 3.1-12     2020-04-10 [2] CRAN (R 4.0.0)
##  tibble        * 3.0.1      2020-04-20 [1] CRAN (R 4.0.0)
##  tidyr         * 1.0.3      2020-05-07 [1] CRAN (R 4.0.0)
##  tidyselect      1.0.0      2020-01-27 [1] CRAN (R 4.0.0)
```

```
##  tidyverse    * 1.3.0      2019-11-21 [1] CRAN (R 4.0.0)
##  timeDate       3043.102   2018-02-21 [1] CRAN (R 4.0.0)
##  utf8           1.1.4      2018-05-24 [1] CRAN (R 4.0.0)
##  vctrs          0.2.4      2020-03-10 [1] CRAN (R 4.0.0)
##  viridisLite    0.3.0      2018-02-01 [1] CRAN (R 4.0.0)
##  webshot        0.5.2      2019-11-22 [1] CRAN (R 4.0.0)
##  withr          2.2.0      2020-04-20 [1] CRAN (R 4.0.0)
##  xfun           0.13       2020-04-13 [1] CRAN (R 4.0.0)
##  xml2           1.3.2      2020-04-23 [1] CRAN (R 4.0.0)
##  yaml           2.2.1      2020-02-01 [1] CRAN (R 4.0.0)
##
## [1] C:/Users/marcb/Documents/R/win-library/4.0
## [2] C:/Program Files/R/R-4.0.0/library
```