# Recommending movies

Adam Tulling

5-1-2021

## Introduction

This paper is a part of a data analysis course on the topic of data science from Harvard University. The goal is to make a recommendation algorithm using a large database which includes film watchers, movies, movie genres and more. Firstly we explore the data and figure out which features are most relevant for further analyses and secondly we will make the analyses and evaluatie the accuracy of the recommendation system.

## Data set

The download of the data set on which we implement data analysis on was given by the code below. The data set comes from the Movielens data package. It is a dataset with 10 million ratings and was released on january of 2009.

*Data set install code*

```r
if(!require(tidyverse))
  install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret))
  install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table))
  install.packages("data.table", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(caret)
library(data.table)

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")

# if using R 3.6 or earlier:
movies <- as.data.frame(movies) %>%
  mutate(movieId = as.numeric(levels(movieId))[movieId],title = as.character(title),
                              genres = as.character(genres))


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

set.seed(1, sample.kind="Rounding")
```

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

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

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

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

Now that we have downloaded the data we also download the packages to pocess and analyse the data. The packages we will use are those in the code below.

```r
if(!require(Metrics))
  install.packages("Metrics", repos = "http://cran.us.r-project.org")
if(!require(lubridate))
  install.packages("lubridate", repos = "http://cran.us.r-project.org")
if(!require(ggpubr))
  install.packages("ggpubr", repos = "http://cran.us.r-project.org")
if(!require(imputeTS))
  install.packages("imputeTS", repos = "http://cran.us.r-project.org")
if(!require(permutations))
  install.packages("permutations", repos = "http://cran.us.r-project.org")
if(!require(utils))
  install.packages("utils", repos = "http://cran.us.r-project.org")
```

### Exploratory data analysis

Before we get into the real anlysis we need to assess what kind of data set we are dealing with. How many observation are present, which variables are there, what kind of class these variables belong to and so on. Firstly a quick summary of the variables. Data exploration and analysis will take place on the edx data set , the so called training set. Not on the validation set. The latter one we will only use to evaluate the final algorithm.

```r
summary(edx)
```

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

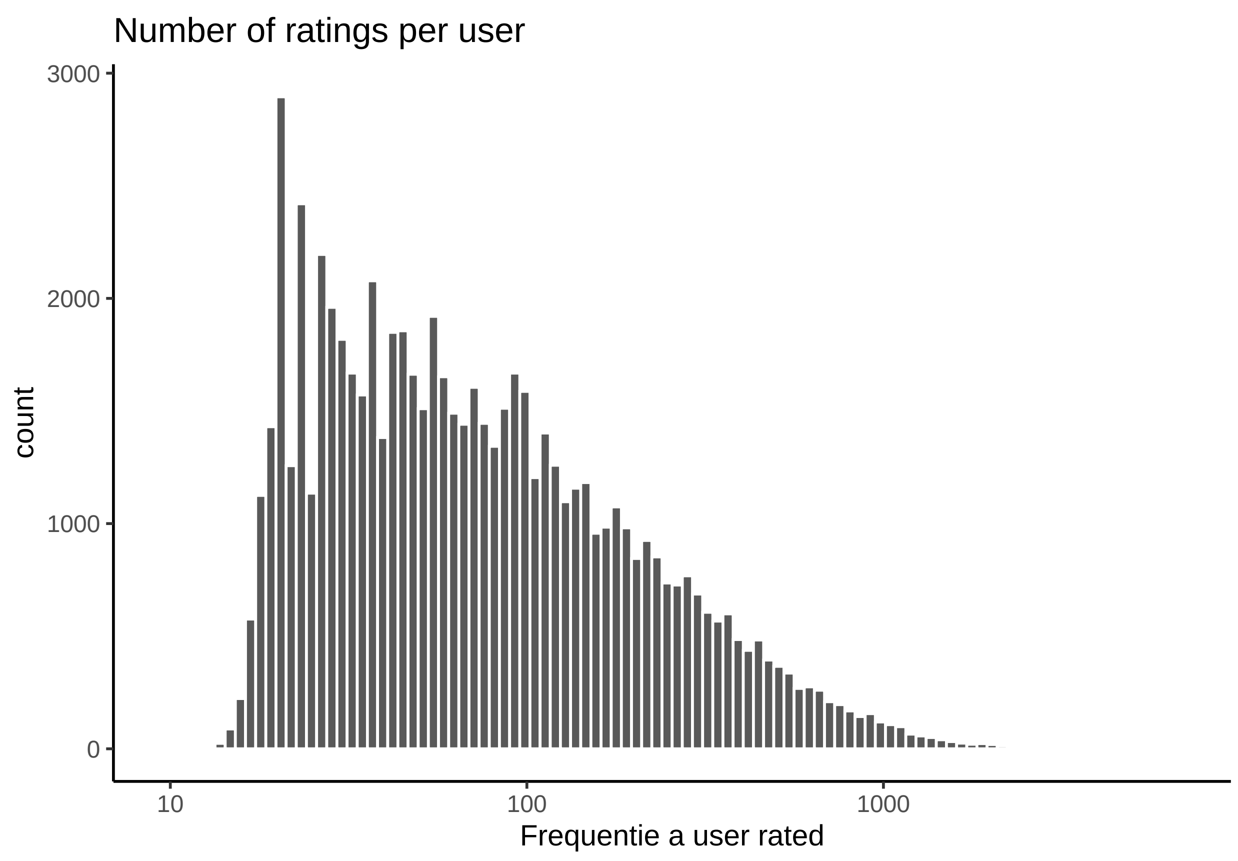
We are dealing with a data frame with 9,000,055 observations with 6 different variables. In the table below you can find the different kinds of variables and to which class they belong.

| Variable | Class |
|---|---|
| userId | integer |
| movieId | numeric |
| rating | numeric |
| timestamp | integer |
| title | character |
| genres | character |

*userId*

Each user has their one opinion. Some users only give 5 star ratings on a couple of films. Others give a more variable rating. Giving 4 stars or 2 stars depending on whether they liked the film. To predict movie ratings we need to determine how many users there are in the data set. Furthermore what is the average movie-rating per user. There are 69878 unique users. In the graph below visualizes the number of users and how often they rate a movie.

```r
# Plotting the amount of movies users rated
edx %>%
  dplyr::count(userId) %>%
  ggplot(aes(n))+
  scale_x_log10()+
  geom_histogram(bins = 100,color="white")+
  xlab("Frequentie a user rated")+
  ggtitle("Number of ratings per user")+
  theme_classic()
```



You can see that there is a big difference in the frequentie individual users rated films.

```r
# Determining the users with the lowest number of ratings
bottom <- edx %>% dplyr::count(userId) %>% arrange(n)

# Determining the users with the highest number of ratings
top <- edx %>% dplyr::count(userId)%>% arrange(desc(n))

# Combining these two in a table
data.frame(Bottom = bottom$n[1:5], Top = top$n[1:5])%>%
  knitr::kable()
```

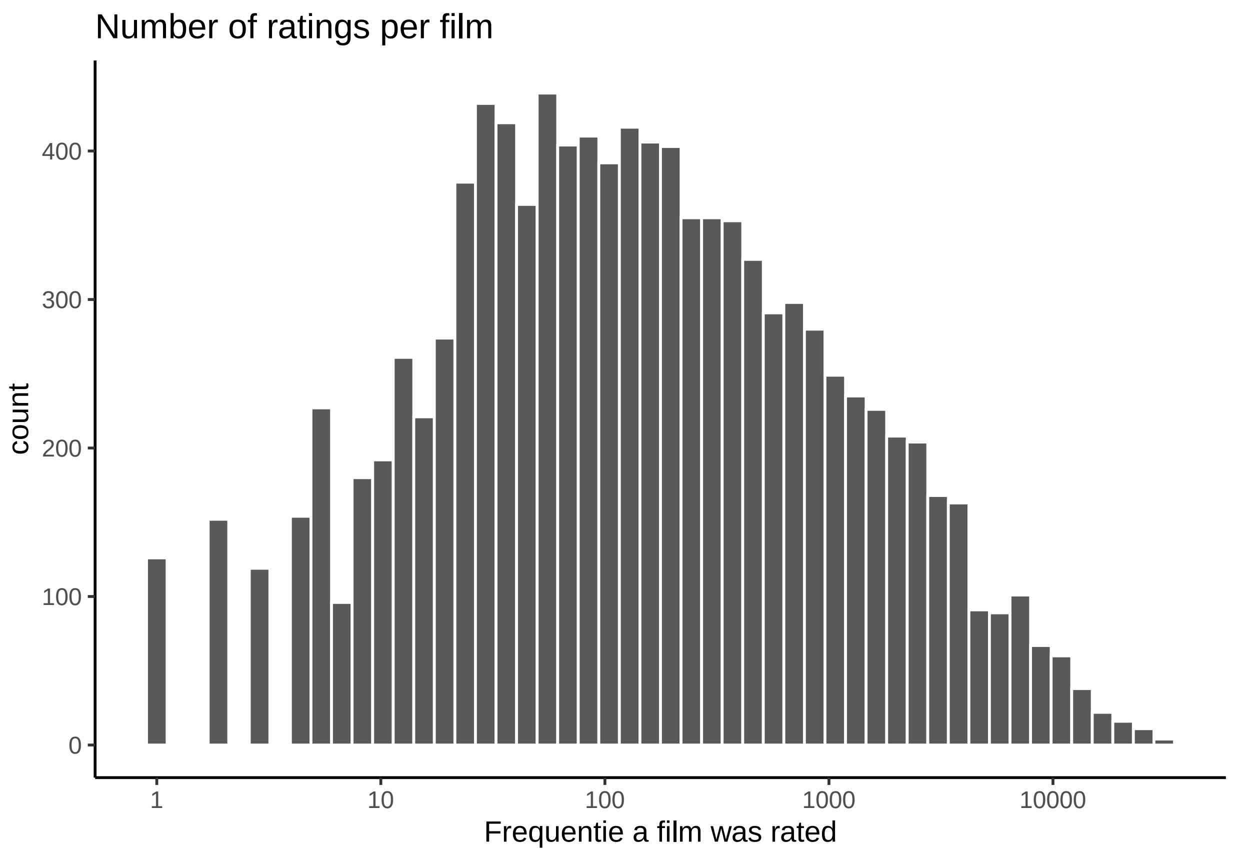
| Bottom | Top |
|---:|---:|
| 10 | 6616 |
| 12 | 6360 |
| 13 | 4648 |
| 13 | 4036 |
| 14 | 4023 |

The lowest number of ratings per user is 10 ratings and the largest number of ratings is 6616.

*MovieId*

Not only the user itself is an important feature in the data set. Also the film it self has a large predictive power. There are 10677 unique movies in the data set. Here below we are trying to illustrate that soms films get rated often and others less.

```r
# Plotting the frequency movies were rated
edx %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n))+
  geom_histogram(bins = 50,color="white")+
  scale_x_log10()+
  xlab("Frequentie a film was rated")+
  ggtitle("Number of ratings per film")+
  theme_classic()
```



```r
# Films rated one time
edx %>% dplyr::count(movieId) %>% filter(n==1) %>%
  nrow(.)
```

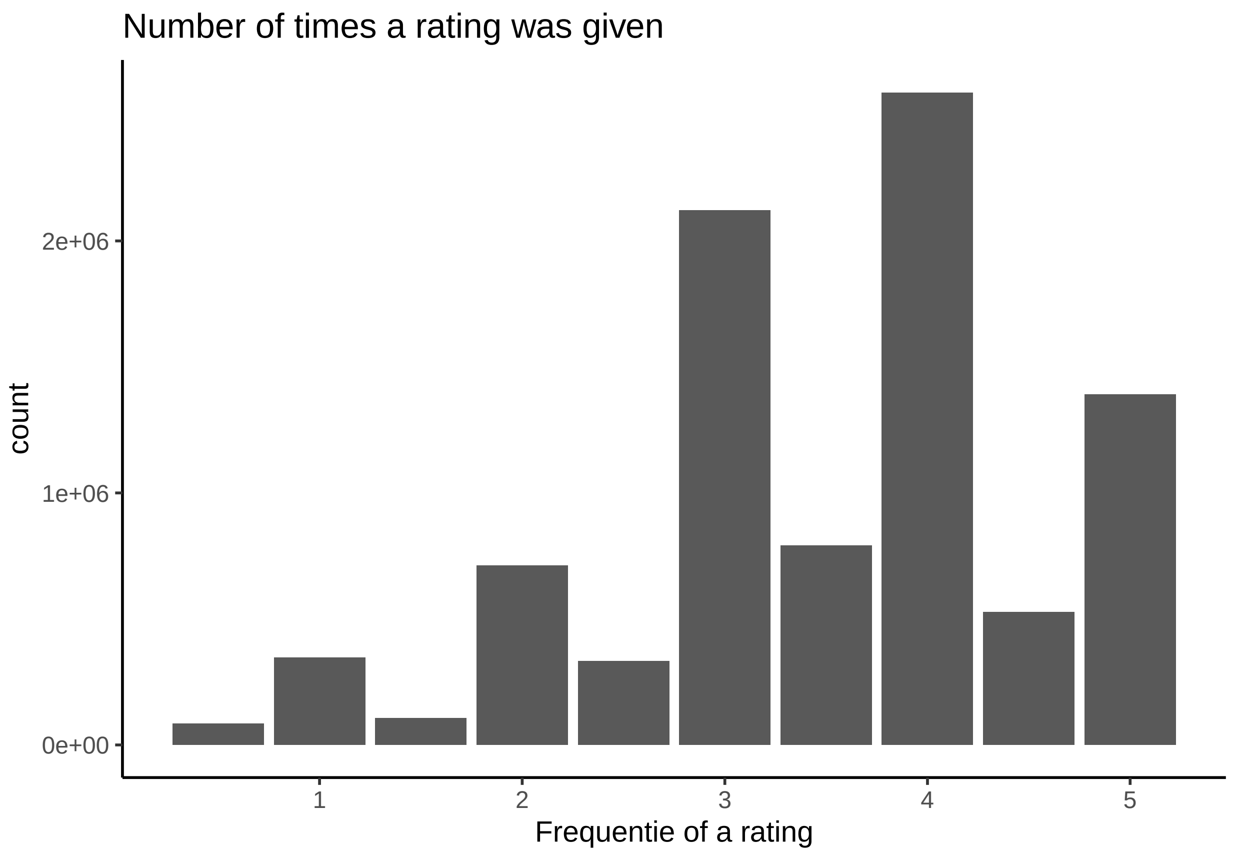
```
## [1] 126
```

There are 10677 unique movies and 126 movies were rated only once.

*Rating*

This is the variable we want to predict later on. In the graph below we will vizualize the amount of times a particular rating was given.

```
# The amount of times a particular rating was given
edx %>%
  group_by(rating) %>% summarise(count=n()) %>% ungroup() %>%
  ggplot(aes(rating,count)) +
  geom_col() +
  xlab("Frequentie of a rating") +
  ggtitle("Number of times a rating was given") +
  theme_classic()
```



You can see that the most frequent given rating is a 4, followed by a 3. Non-whole numbers, like 4.5 are less common than integers.

*Timestamp*

We already know before exploring the data itself that we would find a clear significant difference in the average rating per user and the average rating per film. There are better films and there are people that rate movies higher. An interseting question we are going to answer here is: Is there also a correlation between the time a movie has been rated and the rating itself. A user for example could rate a movie better when rated in the evening as opposed to the morning, On the other hand movies can be reviewed better or worse in different seasons. For now we are going to figure out if there is a difference in movie ratings depending on the year, season, month, week, day or hour of the day. The timestamp faeture is a continuous number depicting the seconds that have passed since 00:00 1 january of 1970 at midnight. The 'timestamp' ranges are determined below

```
# The lowest and highest number of the timestamp faeture
min(edx$timestamp)
```

```
## [1] 789652009
```

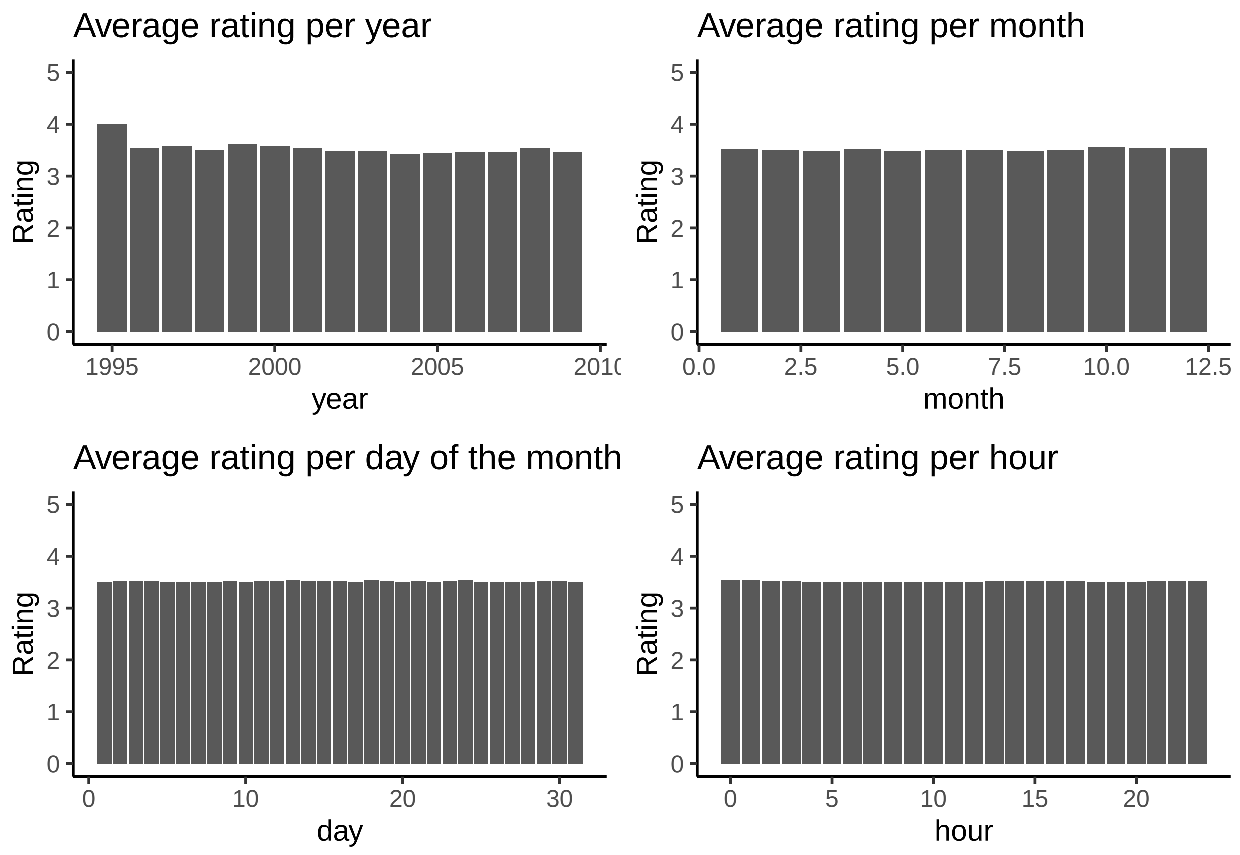
```
max(edx$timestamp)
```

```
## [1] 1231131736
```

The graph below illustrates whether the correlation between the time of rating is significant to the heigth of the average rating.

```
# Converting the timestamp variable into years, months, days and hours
edx_timestamp <- edx %>% mutate(date_time = as_datetime(timestamp)) %>%
        mutate(year = year(date_time), month = month(date_time),
         day = day(date_time), hour = hour(date_time))

# Calculating the mean ratings per year, month, day and hour
Rating_per_year<- edx_timestamp %>% group_by(year) %>%
  summarise( Rating = mean(rating)) %>% ungroup()
Rating_per_month <-edx_timestamp %>% group_by(month) %>%
  summarise( Rating = mean(rating)) %>% ungroup()
Rating_per_day <- edx_timestamp %>% group_by(day) %>%
  summarise( Rating = mean(rating)) %>% ungroup()
Rating_per_hour<- edx_timestamp %>% group_by(hour) %>%
  summarise( Rating = mean(rating)) %>% ungroup()

# Plotting the average ratings per year, month, day and hour
Graf_1<-Rating_per_year%>%
  ggplot(aes(year,Rating)) +
  geom_col() +
  ggtitle("Average rating per year") +
  scale_y_continuous(limits = c(0,5))+
  theme_classic()

Graf_2<-Rating_per_month%>%
  ggplot(aes(month,Rating)) +
  geom_col() +
  ggtitle("Average rating per month") +
  scale_y_continuous(limits = c(0,5))+
  theme_classic()

Graf_3<-Rating_per_day%>%
  ggplot(aes(day,Rating)) +
  geom_col() +
  ggtitle("Average rating per day of the month") +
  scale_y_continuous(limits = c(0,5))+
  theme_classic()

Graf_4<-Rating_per_hour%>%
  ggplot(aes(hour,Rating)) +
  geom_col() +
  ggtitle("Average rating per hour") +
  scale_y_continuous(limits = c(0,5))+
  theme_classic()
```

```r
ggarrange(Graf_1, Graf_2, Graf_3, Graf_4,
          ncol = 2, nrow = 2)
```



It's seems like there is little to no variablity in the feauture time in this data set. Only in the graph comparing different years you see a little difference. The reason for this is probably the fact that a small number of users rated films in 1995. Only c(36955, 36955), c(47, 1079), c(5, 3), c(789652009, 789652009), c("Seven (a.k.a. Se7en) (1995)", "Fish Called Wanda, A (1988)"), c("Crime|Horror|Mystery|Thriller", "Comedy|Crime"), c(789652009, 789652009), c(1995, 1995), c(1, 1), c(9, 9), c(11, 11) ratings were given that year.

*Genres*

Genres are also an important faeture in the dataset. There are 797 unique genres. These include the genre, 'no genre listed'. These genres are most often created by combining different genres. For example the genre could only be 'comedy', but a film genre could also be 'Romantic|Comedy'. There are 20 core genres in the database. Some genres are listed in only one particular film. For example, the film 'The Host' has 8 different genres listed. 'Action', 'Adventure', 'Comedy', 'Drama', 'Fantasy', 'Horror', 'Sci-Fi' and 'Thriller'. Others have only one. Here you can see the most popular genres in the dataset.

Below a list of the most frequently rated genres

```r
# Most frequently rated genres
edx %>% group_by(genres) %>% summarise( Rating = mean(rating), times_rated = n())%>% ungroup()%>%
  arrange(desc(times_rated)) %>% top_n(10) %>% knitr::kable()
```

| genres | Rating | times_rated |
|---|---|---|
| Drama | 3.712364 | 733296 |
| Comedy | 3.237858 | 700889 |
| Comedy\|Romance | 3.414486 | 365468 |
| Comedy\|Drama | 3.598961 | 323637 |
| Comedy\|Drama\|Romance | 3.645824 | 261425 |
| Drama\|Romance | 3.605471 | 259355 |
| Action\|Adventure\|Sci-Fi | 3.507407 | 219938 |
| Action\|Adventure\|Thriller | 3.434101 | 149091 |
| Drama\|Thriller | 3.446345 | 145373 |
| Crime\|Drama | 3.947135 | 137387 |

Below a list of the highest rated genres

```
# Highest rated genres
edx %>% group_by(genres) %>% summarise( Rating = mean(rating), times_rated = n())%>% ungroup() %>%
  arrange(desc(Rating)) %>% top_n(10) %>% knitr::kable()
```

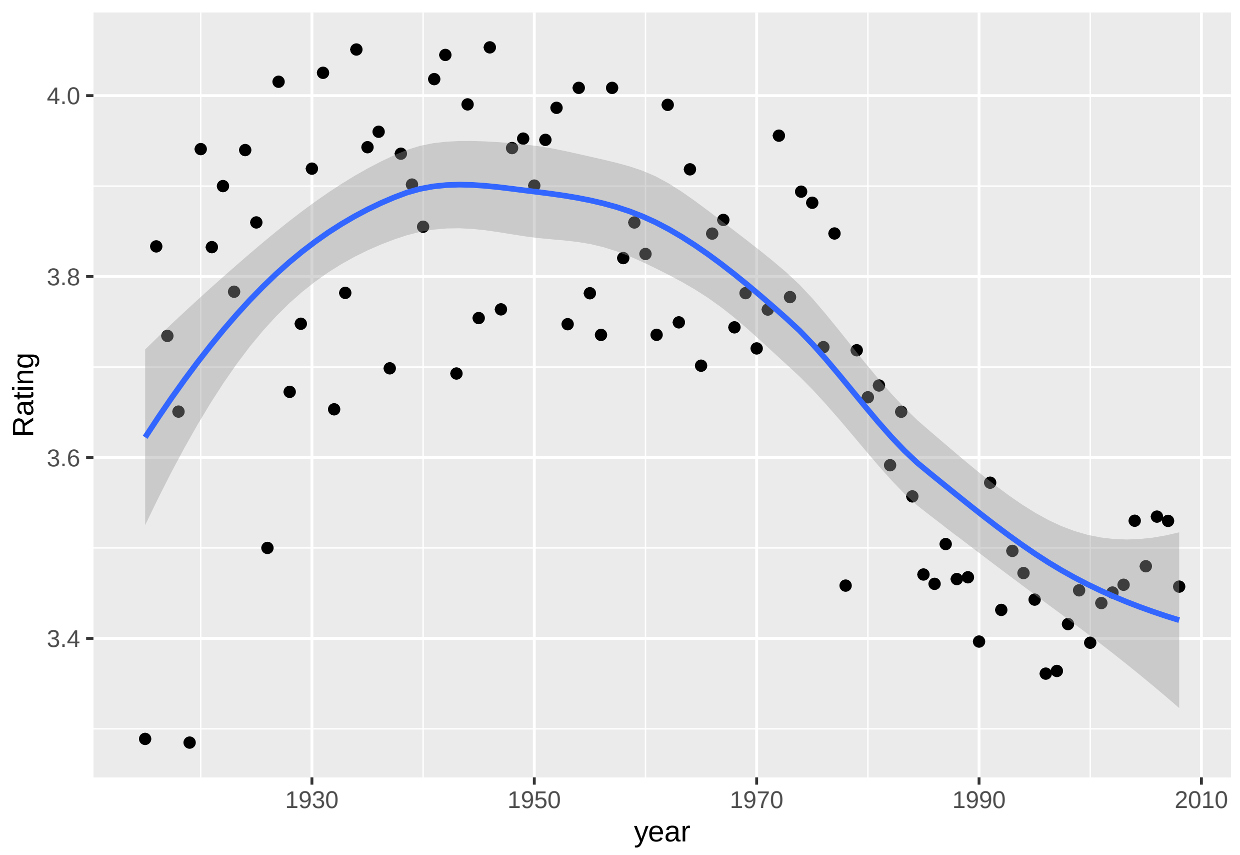
| genres | Rating | times_rated |
|---|---|---|
| Crime\|Drama | 3.947135 | 137387 |
| Drama | 3.712364 | 733296 |
| Comedy\|Drama\|Romance | 3.645824 | 261425 |
| Drama\|Romance | 3.605471 | 259355 |
| Comedy\|Drama | 3.598961 | 323637 |
| Action\|Adventure\|Sci-Fi | 3.507407 | 219938 |
| Drama\|Thriller | 3.446345 | 145373 |
| Action\|Adventure\|Thriller | 3.434101 | 149091 |
| Comedy\|Romance | 3.414486 | 365468 |
| Comedy | 3.237858 | 700889 |

In these two tables you can see that the film category 'Drama' is the most rated film genre followed by 'Comedy' and the best rated film genre is 'Crime|Drama' followed by 'Drama'.

*Title*

Normally the title would not be of much use because there is already a faeture that specifices which film is being analysed, the movieId. Fortunately for us, there is another piece of information in the title faeture, the year of release. On the graph below you can see the correlation between the year of release of the film and their average ratings.

```
# Plot of the year of release of the film and their average ratings
edx %>% mutate(year = as.numeric(str_sub(edx$title,-5,-2))) %>%
  group_by(year) %>% summarise(Rating = mean(rating)) %>% ungroup() %>%
  ggplot(aes(year,Rating))+
  scale_x_continuous()+
  geom_point()+
  geom_smooth(method = "loess")
```



The graph depicts a significant correlation between the year of release and the average given rating.

**Building an algorithm**

**RMSE**

The goal of the paper is to make an algorithm that can predict movie ratings. The effectivity of the algorithm is determined by the root mean squared error (RMSE). The end goal of this paper is to predict movie ratings resulting in a RMSE lower than 0.86490. The rmse has a general formula.

$$\sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

In the Metrics package there is a function to calculate this and further on we will use this RMSE function. It is called by writing ‘rmse()’.

**Guessing movie ratings**

Before we start doing real predictions based on the data we need to figure out what the RMSE is when no data is looked at. This proces is also called random guessing. Guessing the ratings 0.5 till 5 whithout any consideration. This is easy to do. We will be guessing the ratings in a range from 0.5 to 5 with increments of 0.5 because these are the only values the users can choose from. In further analyses we start using intermediate values to eventualy minimize the RMSE. To guarantee replicability we will be using the ‘set.seed()’ function.

```r
# Setting a seed for randomization
set.seed(123, sample.kind = "Rounding")
```

```r
# Guessing the rating
y_hat <- sample(seq(0.5 ,5 ,0.5), nrow(validation) , replace = TRUE)

# Calculating the RMSE using random guessing
RMSE_by_guessing <- rmse(validation$rating, y_hat)

# We will save this RMSE value for comparing purposes
(RMSE <- data.frame( Method = "RMSE by guessing",
                     Result = round(RMSE_by_guessing,4))) %>% knitr::kable()
```

| Method | Result |
|---|---|
| RMSE by guessing | 1.9431 |

It results in a high RMSE rating, as expected.

**Incorporating the mean of ratings**

There is much we can do to improve this first RMSE rating. Firstly we will always predict the same value, the average of all ratings.

```r
#  The average of all ratings
mu <- mean(edx$rating)

# Predicting the average
y_hat <- rep(mu, nrow(validation))

# Calculating the RMSE using this method
RMSE_by_average <- rmse(validation$rating, y_hat)

# We will save this RMSE value for comparing purposes
(RMSE <- bind_rows(RMSE, data.frame(Method = "RMSE by average rating", Result = round(RMSE_by_average,4
  knitr::kable()
```

| Method | Result |
|---|---|
| RMSE by guessing | 1.9431 |
| RMSE by average rating | 1.0612 |

We can see a big improvement over the method of random guessing.

**Correcting for user- and movie ratings**

Some users rate movies more positive than others. Some movies get consistent good ratings others do not. These are one of the most inportant variables an algorithm needs to take into account. We will start with the movie specific ratings.

```r
# Calculating the average rating per movie
edx_MovieRating <- edx %>% group_by(movieId) %>%
  summarise( MovieRating = mean(rating)) %>% ungroup()

# Predicting the average rating for every movie in the validation set
Validation_MovieRating <- left_join(validation, edx_MovieRating, by = "movieId")
```

```r
# Calculating the RMSE using this method
RMSE_by_movie_average <- rmse(validation$rating, Validation_MovieRating$MovieRating)

# We will save this RMSE value for comparing purposes
(RMSE <- bind_rows(RMSE, data.frame( Method = "RMSE by movie averages", Result = round(RMSE_by_movie_ave
```

| Method | Result |
|---|---|
| RMSE by guessing | 1.9431 |
| RMSE by average rating | 1.0612 |
| RMSE by movie averages | 0.9439 |

Also doing the same for the individual users.

```r
# Calculating the average rating per user
edx_UserRating<-edx %>% group_by(userId) %>%
  summarise(UserRating=mean(rating)) %>% ungroup()

# Predicting the average rating for every user in the validation set
Validation_UserRating<-left_join( validation,
                                  edx_UserRating, by = "userId")

# Calculating the RMSE using this method
RMSE_by_user_average <- rmse(validation$rating,
                             Validation_UserRating$UserRating)

# We will save this RMSE value for comparing purposes
(RMSE <- bind_rows(RMSE, data.frame( Method = "RMSE by user averages",
                                     Result = round(RMSE_by_user_average,4))))%>%
  knitr::kable()
```

| Method | Result |
|---|---|
| RMSE by guessing | 1.9431 |
| RMSE by average rating | 1.0612 |
| RMSE by movie averages | 0.9439 |
| RMSE by user averages | 0.9783 |

It looks like the average rating of a movie is more important than the average rating per user at predicting movie ratings. The RMSE is lower using average ratings per movie.

A better prediction could be made when combining these two modalities.

```r
# For conveiniance we will use the mu as the average of all ratings,
# b_i as the movie bias and b_u as the user bias.

mu <- mean(edx$rating)

b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu)) %>% ungroup()
```

```r
b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean( rating - b_i - mu)) %>% ungroup()

# Joining the predictions with the movies en user in the validation set
y_hat <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  summarise(y_hat <- mu + b_i + b_u) %>%
  .$y_hat

# Calculating the RMSE using this method
RMSE_by_user_movie_average <- rmse(validation$rating, y_hat)

# We will save this RMSE value for comparing purposes
(RMSE <- bind_rows(RMSE, data.frame( Method = "RMSE by user and movie averages",
                                      Result = round(RMSE_by_user_movie_average,4))))%>%
  knitr::kable()
```

| Method | Result |
|---|---|
| RMSE by guessing | 1.9431 |
| RMSE by average rating | 1.0612 |
| RMSE by movie averages | 0.9439 |
| RMSE by user averages | 0.9783 |
| RMSE by user and movie averages | 0.8653 |

As you can see. Joining data from the average user and movie ratings improves the predictive outcome.

**Incorporating the genre and year of release variable**

Some users score some genres of film better than others. For example I like comedies more than drama's. Therefore there is a higher chance I will rate comedy movies higher than for instance drama films. To make a better prediction this is a variable we could look at. The last variable we also want to incorporate is the year of release. This latter faeture is not in the timestamp variable but in the title variable.

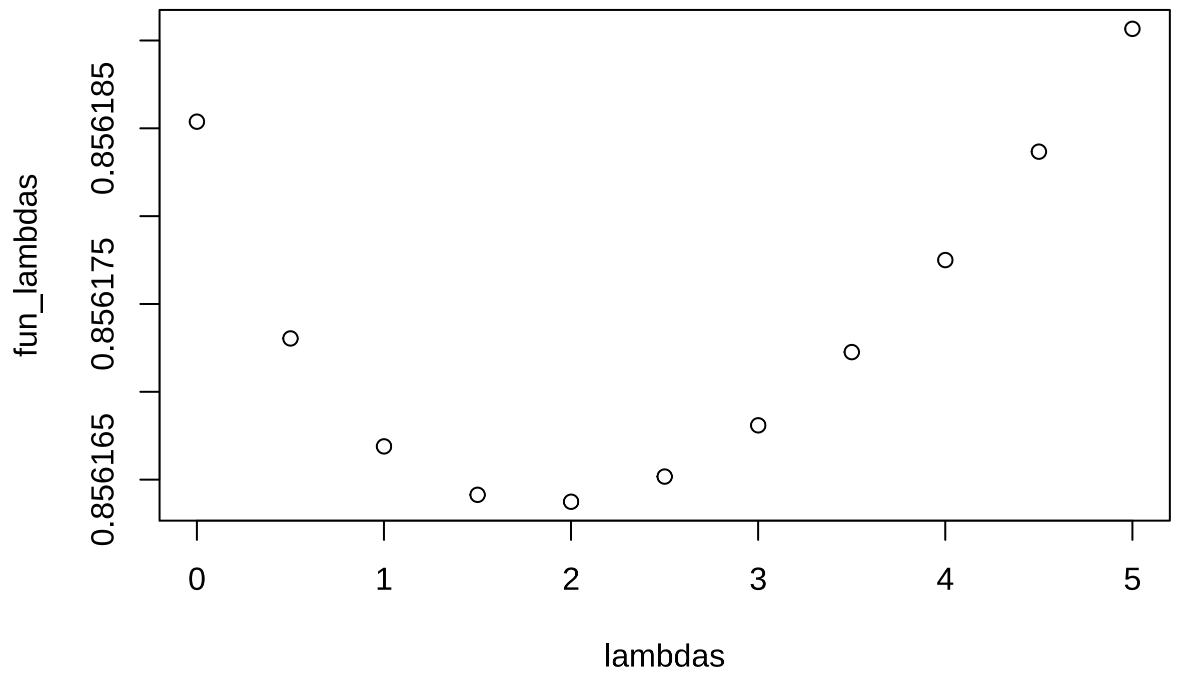
```r
# Again for conveiniance we will use the mu as the average of all ratings,
# b_i as the movie bias and b_u as the user bias Now we will add b_g for the genre bias
# and b_y for the year variable

mu <- mean(edx$rating)

edx <- edx %>% mutate(year = as.numeric(str_sub(edx$title,-5,-2)))
validation <- validation %>% mutate(year = as.numeric(str_sub(validation$title,-5,-2)))

b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu)) %>% ungroup()

b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - b_i - mu)) %>% ungroup()

b_g <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(genres)%>%
  summarise(b_g = mean(rating - b_i - b_u - mu)) %>% ungroup()

b_y <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by= "genres")%>%
  group_by(year) %>%
  summarise(b_y = mean(rating - b_i - b_u - b_g - mu)) %>% ungroup()

# Joining the predictions with the movies,
# users, genres and year of release in the validation set

y_hat <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_y, by = "year")%>%
  mutate(pred = mu + b_i + b_u + b_g + b_y) %>%
  .$pred

# Calculating the RMSE using this method
RMSE_by_movie_user_genre_year <- rmse(validation$rating,y_hat)

# We will save this RMSE value for comparing purposes
(RMSE <- bind_rows(RMSE, data.frame( Method = "RMSE by user, movie, enre and year",
                          Result = round(RMSE_by_movie_user_genre_year,4))))%>%
  knitr::kable()
```

| Method | Result |
|---|---|
| RMSE by guessing | 1.9431 |
| RMSE by average rating | 1.0612 |
| RMSE by movie averages | 0.9439 |
| RMSE by user averages | 0.9783 |
| RMSE by user and movie averages | 0.8653 |
| RMSE by user, movie, enre and year | 0.8648 |

### Regularization

There is a way to improve this already good RMSE value. We will be using regularization as the final step to make a better prediction algorithm. The theory behind this is that some movies, users and genres are present in the database a very few times. Some movies are rated 1 time, some users rate less than others and some genres are present only in one particular film. These rare features can have very variable ratings. If a film is rated once with 5 stars it does not mean that another user would also rate it 5 stars. With the parameter lambda we will be trying the regularize this problem. Important to note is that we use the lambda parameter only on one variable, the movieId. This is because every variable has a different best lambda and using one regularization parameter for all the variables would not improve the algorithm. Furthermore it is important to realize we are picking our parameter lambda solely based on the edx data set, not on the validation set.

```r
# Making a list of lambdas
lambdas <- seq(0,5,0.5)

# Making a function to determine which parameter is most effective

REG <- function(lambda){

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

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

b_g <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(genres)%>%
  summarise(b_g = sum(rating - b_i - b_u - mu)/(n())) %>% ungroup()

b_y <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by= "genres")%>%
  group_by(year)%>%
  summarise(b_y = sum(rating - b_i - b_u - b_g - mu)/(n())) %>% ungroup()

predicted_ratings <- edx %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_y, by = "year")%>%
  mutate(pred = mu + b_i + b_u + b_g + b_y) %>%
  .$pred

rmse(edx$rating,predicted_ratings)
}

# Testing all the lambdas on the function
fun_lambdas <- sapply(lambdas, REG)

# Plotting which lambda results in the lowest RMSE
plot(lambdas,fun_lambdas)
```



```r
# Establishing the best lambda parameter
final_lambda <- lambdas[which.min(fun_lambdas)]
```

As you can see in the graph there is some improvement regularizing the movieId variable. In the calculation below we will figure out what our final RMSE is based on regularization.

```r
# Calculating all averages and implementing them and incorparating the parameter lambda

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

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

b_g <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(genres)%>%
  summarise(b_g = sum(rating - b_i - b_u - mu)/(n())) %>% ungroup()

b_y <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by= "genres")%>%
  group_by(year)%>%
  summarise(b_y = sum(rating - b_i - b_u - b_g - mu)/(n())) %>% ungroup()

# Joining the predictions with the movies,
# users, genres and year of release in the validation set

predicted_ratings <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_y, by = "year")%>%
  mutate(pred = mu + b_i + b_u + b_g + b_y) %>%
  .$pred

# Calculating the RMSE using this method
RMSE_with_regularization <- rmse(validation$rating,predicted_ratings)

# We will save this RMSE value for comparing purposes
(RMSE <-  bind_rows(RMSE, data.frame( Method = "RMSE with regularization",
                                      Result = round(RMSE_with_regularization,4)))) %>%
  knitr::kable()
```

| Method | Result |
|---|---|
| RMSE by guessing | 1.9431 |
| RMSE by average rating | 1.0612 |
| RMSE by movie averages | 0.9439 |
| RMSE by user averages | 0.9783 |
| RMSE by user and movie averages | 0.8653 |
| RMSE by user, movie, enre and year | 0.8648 |
| RMSE with regularization | 0.8646 |

The RMSE has a slight improvement over the method without regularization.

**Result summary**

```
RMSE %>% knitr::kable()
```

| Method | Result |
|---|---|
| RMSE by guessing | 1.9431 |
| RMSE by average rating | 1.0612 |
| RMSE by movie averages | 0.9439 |
| RMSE by user averages | 0.9783 |
| RMSE by user and movie averages | 0.8653 |
| RMSE by user, movie, enre and year | 0.8648 |
| RMSE with regularization | 0.8646 |

## Conclusion

In this paper we explored the world of movie recommendation systems, like we see with Netflix. Our analyses were performed on a well known data set named Movielens. After explorating all variables in the data set we began making predictions. As you can see in the result summary our goal of a rmse below 0.86490 was reached. By performing a precise linear model with regularization we reached a final RMSE of 0.8646.

## Discusion

For now we only used a linear approuch and regularization to establish an algorithm. For further research on this topic it is advised to perform more advanced machine learning tasks. For example k-nearest neighbour, random forest and matrix factorization. The latter one has in my opinion the highest potential for a movie recommendation system.

## Table of referrence

https://grouplens.org/datasets/movielens/10m/