MovieLens Project

HarvardX - PH125.9x Data Science - Capstone - Project 1

Emil Efendiev

18/02/2020

Contents

1 Introduction

1.1 Project Overview

Given: A dataset of movie ratings prepared by GroupLens called MovieLens 10M. This dataset consists of 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users (according to the description on the GroupLens website, actual numbers are slightly different as we would discover later). Released 1/2009. This dataset is a smaller version of the Movielens dataset, which consists of 27,000,000 ratings and 1,100,000 tag applications applied to 58,000 movies by 280,000 users. A smaller version is used for our project to make computation a little easier. Dataset MovieLens 10M contains the following columns:

1. userId

2. movieId
3. rating
4. timestamp
5. title
6. genres

Need to do: Create a movie recommendation system. The code for downloading the dataset and splitting it into "train" (also called "edx" in this project) and "test" (also called "validation" in this project) subsets is provided by the organizers and presented in the relevant subsection below. We need to train a machine learning algorithm using the edx subset to predict movie ratings in the validation subset.

Goal: To minimize residual mean squared error (RMSE) on the validation subset. RMSE is defined below. In particular, we need to find an algorithm that would generate RMSE below 0.86490.

RMSE definition: Let's define $y_{u,i}$ as the rating for movie i by user u and denote our prediction with relevant $\hat{y}_{u,i}$. The RMSE is then defined as:

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

Note: for better transparency, almost all code for this project is explicitly shown below.

1.2 Project initialization code

Below is the code (provided by the organizers), which downloads the dataset, and splits it into train (edx) and test (validation) subsets.

```r
#################################
# Create edx set, validation set
#################################

# Note: this process could take a couple of minutes

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")

# 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")
 movies <- as.data.frame(movies) %>%
   mutate (movieId = as.numeric(levels(movieId))[movieId],
           title = as.character(title),
           genres = as.character(genres))

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

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
 edx <- movielens[-test_index,]
 temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
      semi_join(edx, by = "movieId") %>%
      semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
 edx <- rbind(edx, removed)

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

In addition, below is the code for ggthemes and knitr libraries installation in case they are not installed yet (might cause errors otherwise).

```
if(!require(ggthemes))
  install.packages("ggthemes", repos = "http://cran.us.r-project.org")
if(!require(knitr))
  install.packages("knitr", repos = "http://cran.us.r-project.org")
```

Let's also code a function in R that computes the RMSE. $y_{u,i}$ would be called TV (for True Value). $\hat{y}_{u,i}$ would be called PV (for Predicted Value).

```
RMSE <- function(PV, TV) {
    sqrt(mean((PV - TV)^2))
  }
```

1.3 Overview of the dataset

Let's now explore the dataset. We are going to review the edx object (contains most of the records of the movielens object (dropped in the initial code)), which is used for data training. The following code in R shows the first four records from the dataset.

```
head(edx,4)
```

```
##   userId movieId rating timestamp            title                        genres
## 1      1     122      5 838985046 Boomerang (1992)                Comedy|Romance
## 2      1     185      5 838983525  Net, The (1995)         Action|Crime|Thriller
## 4      1     292      5 838983421  Outbreak (1995)  Action|Drama|Sci-Fi|Thriller
## 5      1     316      5 838983392  Stargate (1994)       Action|Adventure|Sci-Fi
```

Also, it might be useful to have a look at the summary of the dataset.

```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
##
##
##
```

As a next step, let's have a look at the actual number of records.

```r
matrix(c(formatC(nrow(edx), digits = 0, format = 'f', big.mark = ','),
         formatC(ncol(edx), digits = 0, format = 'f', big.mark = ',')),
       dimnames = list(c("Number of rows","Number of colums"),""))
```

```
##
## Number of rows   "9,000,055"
## Number of colums "6"
```

Let's also have a look at how many unique records for the number of users and the number of movies we have.

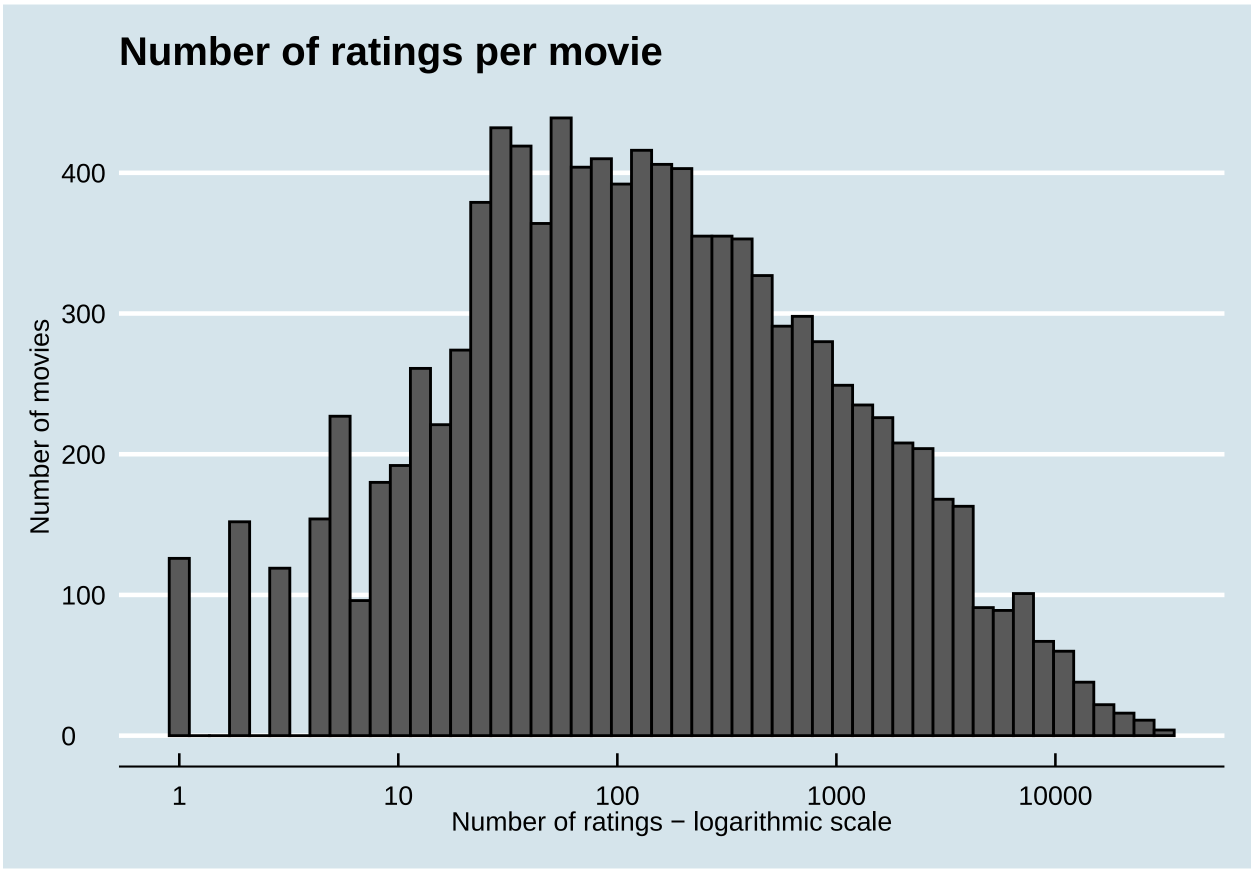
```r
edx %>%
  summarize(number_of_users = formatC(n_distinct(userId),
                                      digits = 0, format = 'f',
                                      big.mark = ','),
            number_of_movies = formatC(n_distinct(movieId),
                                       digits = 0, format = 'f',
                                       big.mark = ','))
```

```
##   number_of_users number_of_movies
## 1          69,878           10,677
```

Given that our database contains 9,000,055 records and that multiplication of the number of users by the number of movies gives us 746,087,406 potential records, we can say that not every user rated every movie. Let's actually, have a look at how many ratings are there per movie and how many ratings are given by users by making relevant plots.

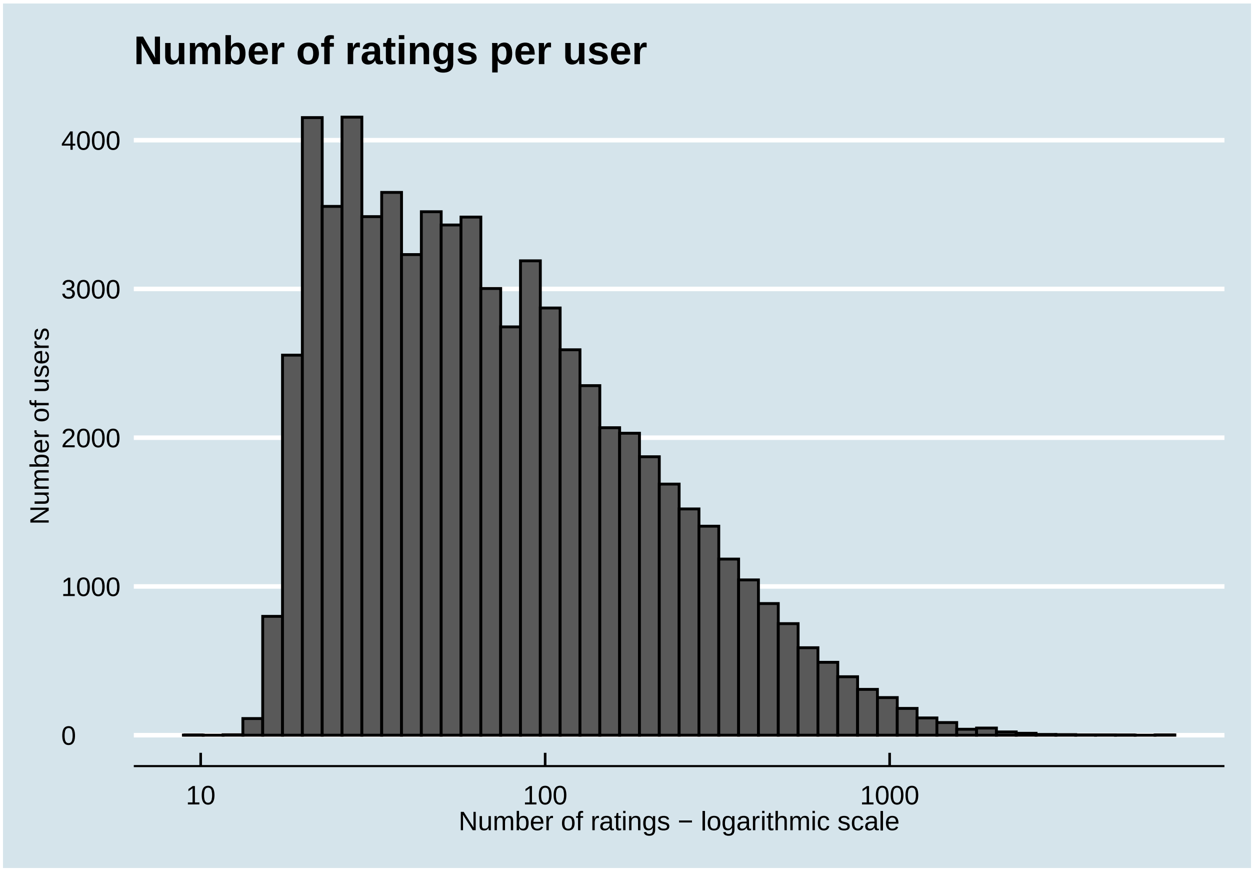
The following code in R generates a chart showing the number of ratings per movie.

```r
edx %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
    geom_histogram(bins = 50, color = "black") +
    scale_x_log10() +
    xlab("Number of ratings - logarithmic scale") +
    ylab("Number of movies") +
    ggtitle("Number of ratings per movie") +
    theme_economist()
```



The following code in R generates a chart showing the number of ratings per user.

```r
edx %>%
  dplyr::count(userId) %>%
  ggplot(aes(n)) +
    geom_histogram(bins = 50, color = "black") +
    scale_x_log10() +
    xlab("Number of ratings - logarithmic scale") +
    ylab("Number of users") +
    ggtitle("Number of ratings per user")+
    theme_economist()
```

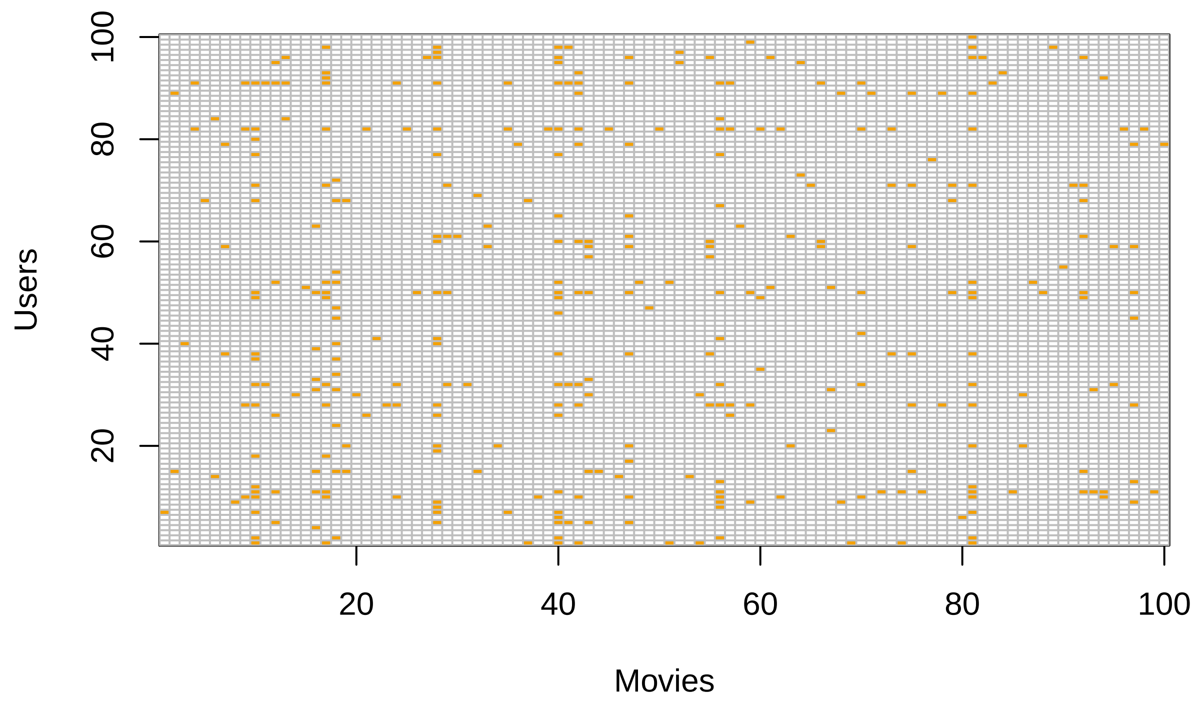


An interesting way to visualize our dataset is to create a random matrix of random 100 records of each movies and users (thus having 10,000 records in total) and see how sparse this matrix is. Below is the code which creates the relevant chart/matrix (highlighted are the cells of the chart where a user left a rating for a movie).

```
set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead

users <- sample(unique(edx$userId), 100) # creates a temporary object for calculation

edx %>%
  filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>%
  select(sample(ncol(.), 100)) %>%
  as.matrix() %>%
  t(.) %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users")
  abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
```



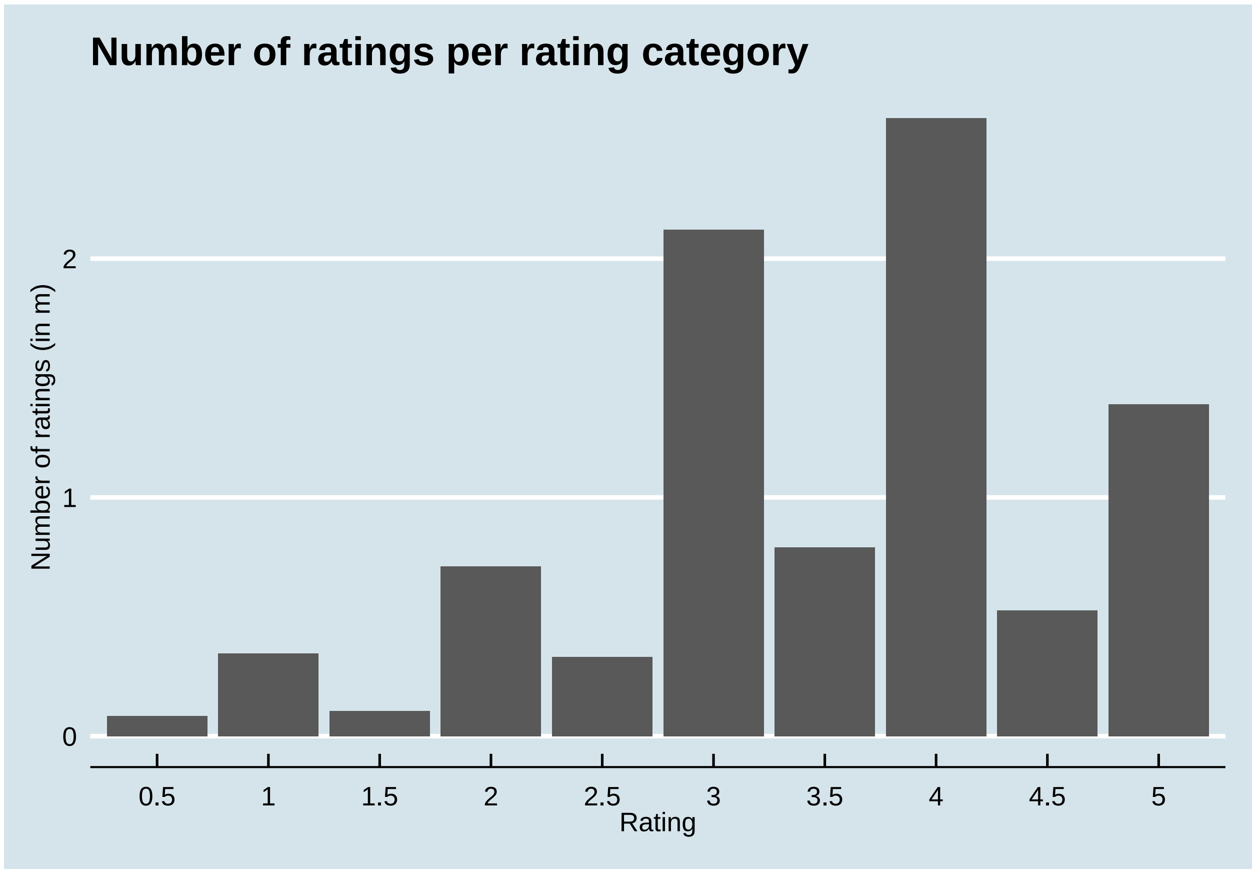
```r
rm(users) # removes unused object
```

This chart clearly shows that average user rates only a small portion of movies in the dataset, which makes perfect logical sense (especially if one considers the amount of time required for a user to review all movies in the dataset).

Let's also have a look at some additional information from the dataset. First, let's see what a common rating for a movie is.

```r
CPR <- as.data.frame(table(edx$rating)) # creates a temporary object for calculation

ggplot(CPR, aes(x=Var1,y = Freq/1000000)) +
  geom_bar(stat = "identity") +
  xlab("Rating") +
  ylab("Number of ratings (in m)") +
  ggtitle("Number of ratings per rating category")+
  theme_economist()
```

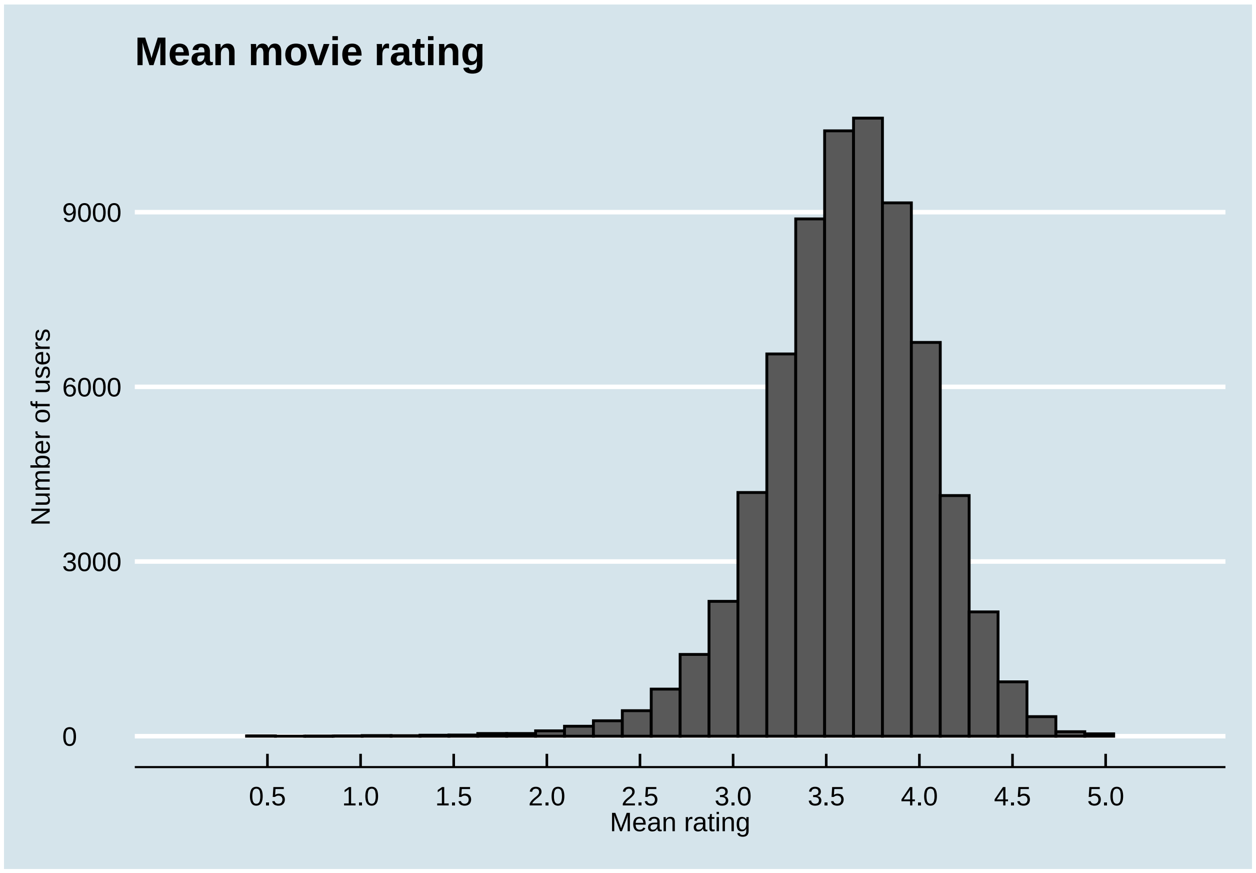


```r
rm (CPR) # removes unused object
```

Interestingly, users prefer integers (1, 2, 3, 4, 5) to decimal numbers (0.5, 1.5, 2.5, 3.5, 4.5).

Then let's have a look at the average rating:

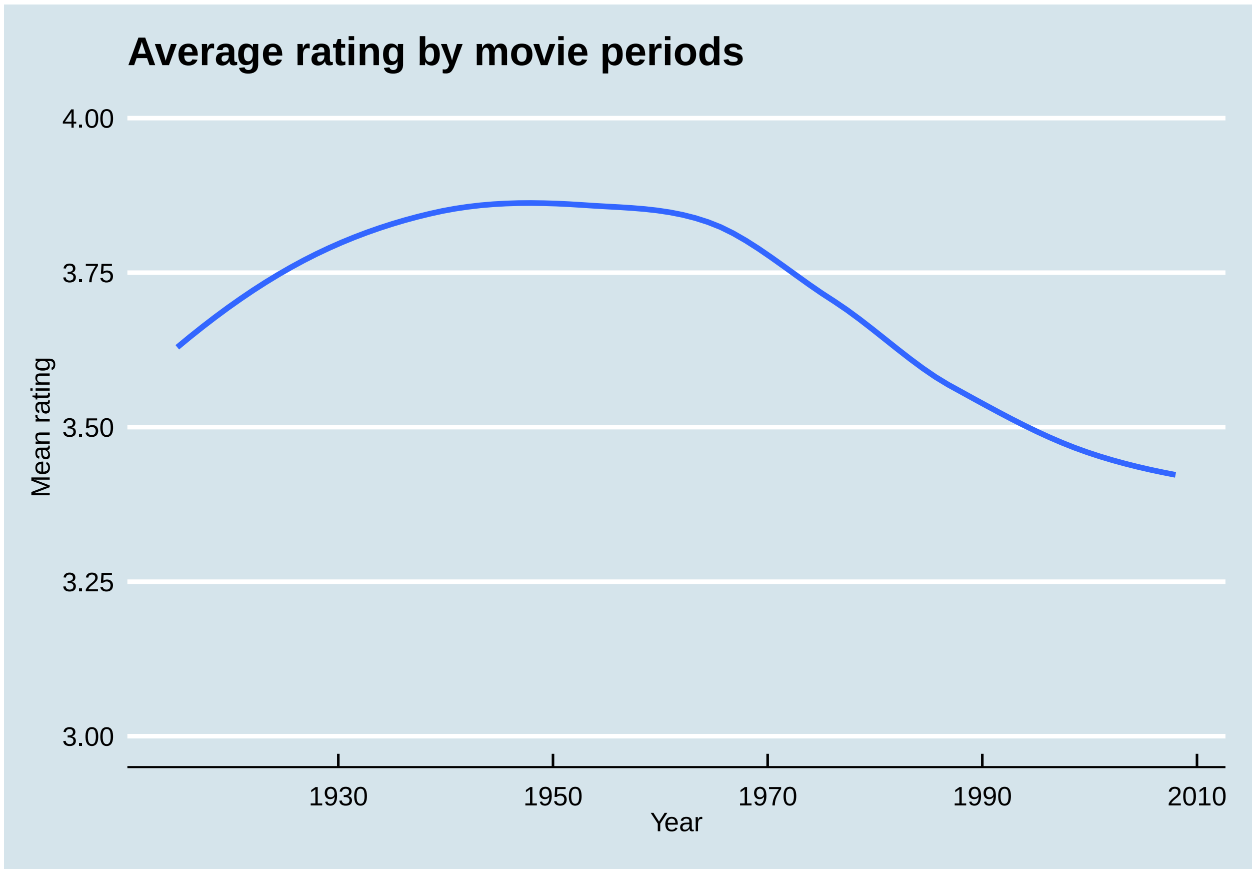
```r
edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
    geom_histogram(bins = 30, color = "black") +
    xlab("Mean rating") +
    ylab("Number of users") +
    ggtitle("Mean movie rating") +
    scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
    theme_economist()
```



And as the last step, we'll have a look at the distribution of ratings by movie release year (we are going to smooth the line to see the trend). But before doing so, we need to extract years from the "title" column of the dataset. Let's save data to a new dataset called edx_with_year to make sure that we do not modify the initial dataset without need. We'll drop edx_with_year dataset after making the chart to save memory but can reintroduce this object in the future in case we need it. Below the code in R.

```r
# creates a temporary object for calculation
edx_with_year <- edx %>% mutate(year = as.numeric(str_sub(title,-5,-2)))

edx_with_year %>%
  group_by(year) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(year, rating)) +
    scale_y_continuous(limits = c(3, 4)) +
    xlab("Year") +
    ylab("Mean rating") +
    ggtitle("Average rating by movie periods") +
    geom_smooth(se=FALSE) +
    theme_economist()
```



```r
rm(edx_with_year) # removes unused object
```

The information reviewed above provides us with a good understanding of the dataset, so we can move forward and start working on our recommendation system.

1.4 Key steps to building a recommendation system

In the next section, we'll start with a simple approach of applying the same rating for all movies as our prediction. Then we'll try to enhance this algorithm by introducing "biases" which would add movie- and user- specific dimensions to our average, thus theoretically improving results.

2 Building recommendation system / data analysis

2.1 Approach 1 - Simple model

In our case, the easiest way to predict a rating is to assume that it is equal to the average of the edx dataset (please note that taking any other number would decrease the accuracy of the algorithm). With this approach, we assume that all the differences are explained by random variation. Mathematically, this is going to look like this:

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

Where $Y_{u,i}$ - predicted rating for movie i by user u, μ is the "true" rating for all movies, and $\epsilon_{u,i}$ - independent errors sampled from the same distribution centered at 0.

We are going to make relevant calculations and get results using the following R code.

```r
edx_av <- mean(edx$rating)

cat('The average of the edx dataset is ',
    formatC(edx_av, digits = 2, format = 'f', big.mark = ','), sep = "")
```

```
## The average of the edx dataset is 3.51
```

Then let's calculate RMSE for this approach.

```r
AP1_RMSE <- RMSE(edx_av, validation$rating)

cat('RMSE is equal to ',
    formatC(AP1_RMSE, digits = 5, format = 'f', big.mark = ','), sep = "")
```

```
## RMSE is equal to 1.06120
```

Clearly, not very accurate system and very far from our RMSE target.

2.2 Approach 2 - Enhancing the system using "movie effect"

As we know from the data exploration exercise in the previous section, movies are rated differently by users. Some of them have higher ratings than others. To enhance our algorithm, we can introduce a movie specific rating (so-called movie bias) that will adjust the dataset average. Below is the adjusted formula for this approach:

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

In this formula, b_i stands for movie bias / effect and represents the average difference between the actual rating of a movie i by user u and average dataset rating.

Let's calculate b_i for our dataset. We'll be using the edx dataset for bias calculation and then applying this bias to the validation dataset. Please note that since the initial code (provided by the organizers; which downloads and splits data) adjusts data to make sure that all unique values from userId and movieId fields in the validation dataset are also present in the edx dataset, we can be sure that joining operations would not result in any errors (e.g. "NA" error). Below is the relevant code in R.

```r
AP2_movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - edx_av))
```

Now, let's build our predictions for the validation dataset using the average and the movie bias as calculated above.

```r
predicted_ratings_BI <- edx_av + validation %>%
  left_join(AP2_movie_avgs, by='movieId') %>%
  pull(b_i)
```

To see how our enhanced approach performs, let's calculate relevant RMSE.

```r
AP2_RMSE <- RMSE(predicted_ratings_BI, validation$rating)

cat('RMSE for our enhanced model is equal to ',
    formatC(AP2_RMSE, digits = 5, format = 'f', big.mark = ','), sep = "")
```

```
## RMSE for our enhanced model is equal to 0.94391
```

Our enhanced approach provides us with lower RMSE than in approach 1, but still far from our target RMSE. In the next subsection, we are going to introduce "user bias" to improve our algorithm even further.

2.3 Approach 3 - Enhancing the system using "user effect"

In addition to some movies being higher rated than others, some users are more generous than others. To account for this factor, we are going to introduce user bias. By essence, it is very similar to movie bias. Mathematically, our formula will be modified the following way:

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

where b_u represents user bias.

Now we need to calculate b_u for every user and then amend our predictions. Let's start with the calculation of b_u using the following R code.

```r
AP3_user_avgs <- edx %>%
  left_join(AP2_movie_avgs, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - edx_av - b_i))
```

As the next step, we need to apply user bias (as well as movie bias) to the validation dataset.

```r
predicted_ratings_BI_BU <- validation %>%
  left_join(AP2_movie_avgs, by='movieId') %>%
  left_join(AP3_user_avgs, by = 'userId') %>%
  mutate(PRBIBU = edx_av + b_i + b_u) %>%
  pull(PRBIBU)
```

Then we calculate RMSE to see how the updated algorithm works.

```r
AP3_RMSE <- RMSE(predicted_ratings_BI_BU, validation$rating)

cat('RMSE for our model enhanced with movie and user effects is equal to ',
    formatC(AP3_RMSE, digits = 5, format = 'f', big.mark = ','), sep = "")
```

```
## RMSE for our model enhanced with movie and user effects is equal to 0.86535
```

We see improvements in RMSE, which is now close to our target. In the next subsection, we are going to review additional steps we can take to improve RMSE.

2.4 Approach 4 - Regularization

One of the issues of the dataset is that some movies are rated only by very few users. Also some users rated only a very small number of movies. In our previous approaches, we allowed such entries to influence our adjustments to the average. However, such records are not reliable and can be viewed as noise. To penalize such entries, we are going to use the regularization technique. Regularization permits us to penalize large estimates that are formed using small sample sizes. Mathematically, regularization in our particular case means minimization of the following formula:

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

where λ represents an element related to penalty.

Before we start building a new algorithm, we need to split the edx dataset into train and test partitions that we are going to use to determine λ, which minimizes RMSE. We cannot use the validation dataset for this purpose since it will result in overtraining. Then once we determine λ, we are going to make predictions for the validation dataset using this λ as an approximation that would make data more reliable. Then we are going to calculate RMSE. Below is the code which splits the edx dataset into train and test partitions.

```r
# Train set will be 10% of edx data
set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
 edx_train <- edx[-test_index,]
 temp <- edx[test_index,]

# Make sure userId and movieId in edx_test set are also in edx_train set
edx_test <- temp %>%
      semi_join(edx_train, by = "movieId") %>%
      semi_join(edx_train, by = "userId")

# Add rows removed from edx_test set back into edx_train set
removed <- anti_join(temp, edx_test)
 edx_train <- rbind(edx_train, removed)

rm(test_index, temp, removed)
```

Let's now write a function in R, which would take λ as an argument, make predictions (adjusting dataset average with user and movie biases) and calculate relevant RMSE. Then, we'll write a code that would try multiple λ values to determine the one which minimizes RMSE. As mentioned above, we calculate RMSE only based on the edx dataset (consisting of train and test partitions).

```r
Reg_Fun_RMSE_mod <- function(l){

  mu <- mean(edx_train$rating)

  # movie bias element
  b_i <- edx_train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  # user bias element
  b_u <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n()+l))

  # predictions
  predicted_ratings <-
    edx_test %>%
    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, edx_test$rating))
}
```

Now, let's try lambdas from 0 to 10 with a step of 0.2 to determine the one which minimizes RMSE.

```r
lambdas <- seq(0, 10, 0.2)

rmses <- sapply(lambdas, Reg_Fun_RMSE_mod)

LMRMSE <- lambdas[which.min(rmses)] # LMRMSE - lamda min RMSE

cat('Lambda which minimizes RMSE is equal to ', LMRMSE, sep = "")
```

```
## Lambda which minimizes RMSE is equal to 5
```

Then we need to make predictions (for the validation dataset) and calculate RMSE using the code below (which utilizes lambda that theoretically should minimize RMSE).

```r
# movie bias element
b_i_m <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - edx_av)/(n()+LMRMSE))

# user bias element
b_u_m <- edx %>%
  left_join(b_i_m, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - edx_av - b_i)/(n()+LMRMSE))

# predictions
predicted_ratings_m <-
  validation %>%
  left_join(b_i_m, by = "movieId") %>%
  left_join(b_u_m, by = "userId") %>%
  mutate(pred = edx_av + b_i + b_u) %>%
  pull(pred)

AP4_RMSE <- RMSE(predicted_ratings_m, validation$rating)

cat('RMSE is equal to ',
    formatC(AP4_RMSE, digits = 5, format = 'f', big.mark = ','), sep = "")
```

```
## RMSE is equal to 0.86482
```

This approach results in the lowest RMSE observed so far. Moreover, this RMSE is lower than our benchmark of 0.86490. Thus we have reached the goal.

3 Results

Let's summarize our RMSEs in a table using the following R code.

```
RMSE_models <- c('1. Simple Average',
                 '2. Adjusted by movie effect',
                 '3. Adjusted by movie and user effects' ,
                 '4. Regularized movie and user effects')

RMSE_values <- formatC(c(AP1_RMSE,AP2_RMSE,AP3_RMSE,AP4_RMSE), digits = 5, big.mark=",")

RMSE_table  <- data.frame(RMSE = RMSE_values, row.names = RMSE_models)

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

	RMSE
1. Simple Average	1.0612
2. Adjusted by movie effect	0.94391
3. Adjusted by movie and user effects	0.86535
4. Regularized movie and user effects	0.86482

The best result is achieved in approach 4, where we regularize movie and user effects.

4 Conclusion

We have successfully built a machine learning algorithm that predicts movie ratings. We have tried four models ranging from the simplest approach of taking an average as a prediction (approach 1) to the model adjusting such average with movie and user effects (biases) and also mitigating noisy entries (approach 4). This approach (which uses regularization) results in the lowest error estimated by RMSE, which is also lower than our benchmark of 0.86490.

However, even with RMSE below the benchmark, we can further improve the accuracy of our model by adding other elements from our dataset to our algorithm as well as by using more sophisticated methods for predicting ratings.

5 Sources / Literature used

1. Introduction to Data Science - Data Analysis and Prediction Algorithms with R by Rafael A. Irizarry (available at https://rafalab.github.io/dsbook/)
2. MovieLens 10M dataset by GroupLens