HarvardX - PH125.9x - DataScience: Capstone - MovieLens

Jan Brettschneider

29th March 2020

1. Introduction

This document summarizes the MovieLens project for the Capstone Exam in course of the HardvardX Data Science Certificate.

The dataset used is the **10M version of the MovieLens dataset** which contains 10 million ratings from different users for different movies. We will examine the dataset in more depth in the analysis section.

Netflix offered a challenge for improving the algorithm of rating prediction, with a grand prize of US$ 1,000,000 up for grabs. The winning team bested the Netflix algorithm by 10,06%. The dataset used for this challenge was approximately 10 times larger than the one we use for this project.

During this project the following steps will be conducted:

1. Create/download the data
2. Explore the data and find variables which seem useful for the model creation
3. Build and optimize the model with help of the parameters identified and regularization
4. Use the best model for validation and discuss the results
5. Conclusion, limitations and comments

The ambition is to get a RMSE value below 0.86490.

The dataset

The course staff provides the following code to download and transform the dataset, which will be used as an input for the project:

```
#################################
# 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")
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)
```

Two datasets have been created. The edx dataset contains 9,000,055 observations, the validation set 999,999 observations. Each dataset contains the following variables:

- **userId:** Integer variable for identifying the user
- **movieId:** Numeric unique indentifier for the movie
- **rating:** Numeric variable from 0 to 5 in steps of 0.5 whereby 5 represents the best rating
- **timestamp:** Integer variable storing the time of rating, which has to be translated first. Base is 1970-01-01
- **title:** Char, which contains the title of the movie with the release year in brackets behind
- **genre:** Char, containing one or more genres seperated by "|"

Manipulating the dataset before the analysis

Before we can work with the dataset we have to perform some operations to manipulate the data but also to make computation (especially RAM usage) more efficient. Therefore we will transform **userId**, **movieId** and **genres** into factors instead of the afore mentioned data types. The timestamp will be transformed into a date. In this process we will reduce it to the year instead of a full date plus time. From the title column the year in which the movie has been released is extracted, put into a new column and the title is adapted accordingly. This gives us the possibility to access this metric much easier. Note that we do the computation on both sets, because we want the possibility to use the new created variables for the validation later. This is not an unallowed use of the validation dataset because we are not testing our model!

```
# Factorize userId, movieId and genre
edx$userId <- as.factor(edx$userId)
edx$movieId <- as.factor(edx$movieId)
```

```r
edx$genres <- as.factor(edx$genres)
validation$userId <- as.factor(validation$userId)
validation$movieId <- as.factor(validation$movieId)
validation$genres <- as.factor(validation$genres)

# Convert the timestamp and extracting the year of rating
# starting at the 1st of January 1970
edx$timestamp <- year(as.POSIXct(edx$timestamp, origin = "1970-01-01"))
validation$timestamp <- year(as.POSIXct(validation$timestamp, origin = "1970-01-01"))

# Extract the release year of the movie from the title -
# format: xxx (yyyy) and rename the title without year
edx$year<-substr(edx$title,nchar(as.character(edx$title))-4,
                 nchar(as.character(edx$title))-1)
edx$title<-paste0(substr(edx$title,1,nchar(as.character(edx$title))-6))
validation$year<-substr(validation$title,nchar(as.character(validation$title))-4,
                 nchar(as.character(validation$title))-1)
validation$title<-paste0(substr(validation$title,1,
                 nchar(as.character(validation$title))-6))

# Adding a unique identifier for each rating
edx <- edx %>% mutate(ratingId = row_number())
validation <- validation %>% mutate(ratingId = row_number())
```

Now the dataset looks like this:

```r
head(edx)
```

```
##   userId movieId rating timestamp                   title
## 1      1     122      5      1996               Boomerang
## 2      1     185      5      1996                Net, The
## 3      1     292      5      1996                Outbreak
## 4      1     316      5      1996                Stargate
## 5      1     329      5      1996 Star Trek: Generations
## 6      1     355      5      1996        Flintstones, The
##                          genres year ratingId
## 1                Comedy|Romance 1992        1
## 2         Action|Crime|Thriller 1995        2
## 3  Action|Drama|Sci-Fi|Thriller 1995        3
## 4       Action|Adventure|Sci-Fi 1994        4
## 5 Action|Adventure|Drama|Sci-Fi 1994        5
## 6       Children|Comedy|Fantasy 1994        6
```

2. Analysing the dataset

In this section we will have a look at the distribution of the variables and will get some insights which variables might be a good input for our model. Let's look at the distribution of the ratings.

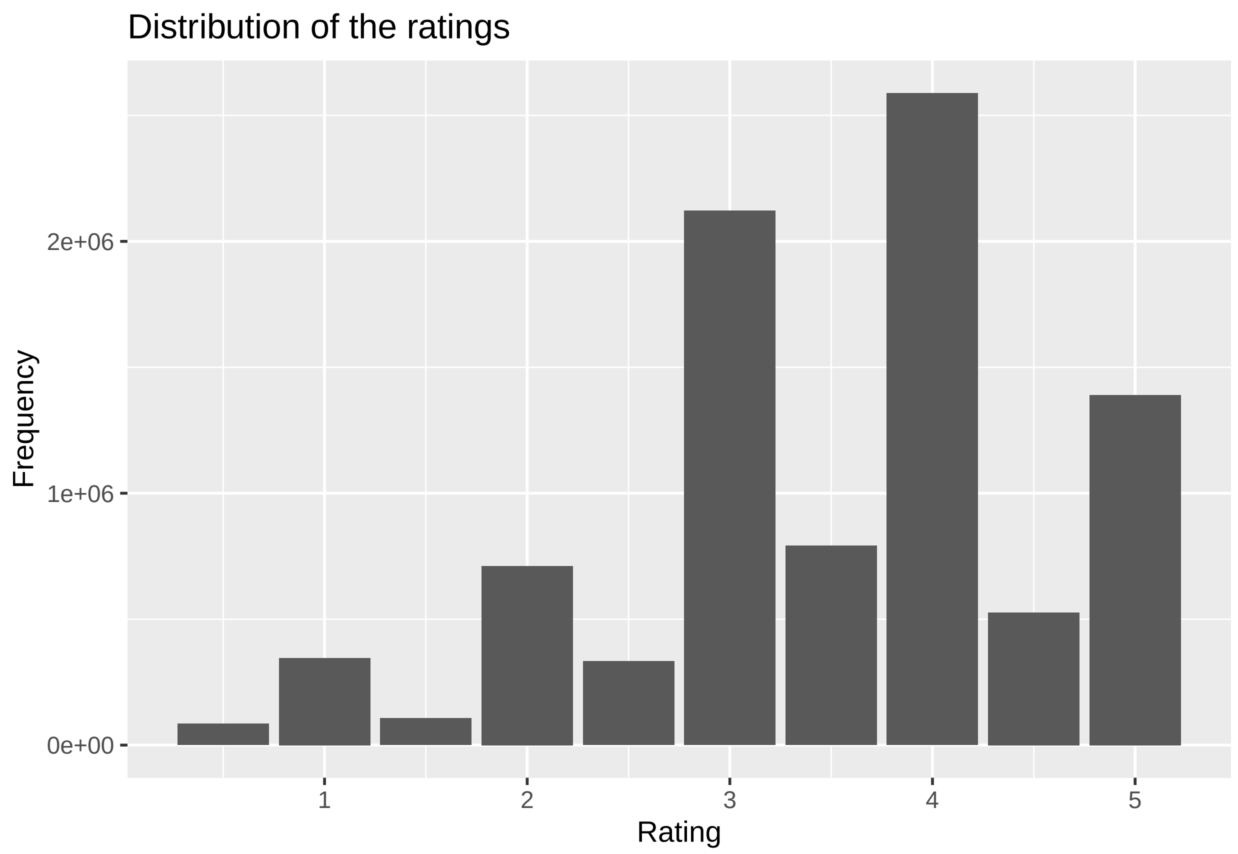
Before we look at the ratings we summarize the key metrics for the ratings data which will help to decide which factors to test for our model.

```r
summary(edx$rating)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.000   4.000   3.512   4.000   5.000
```

First we will take a look at how the ratings are distributed.

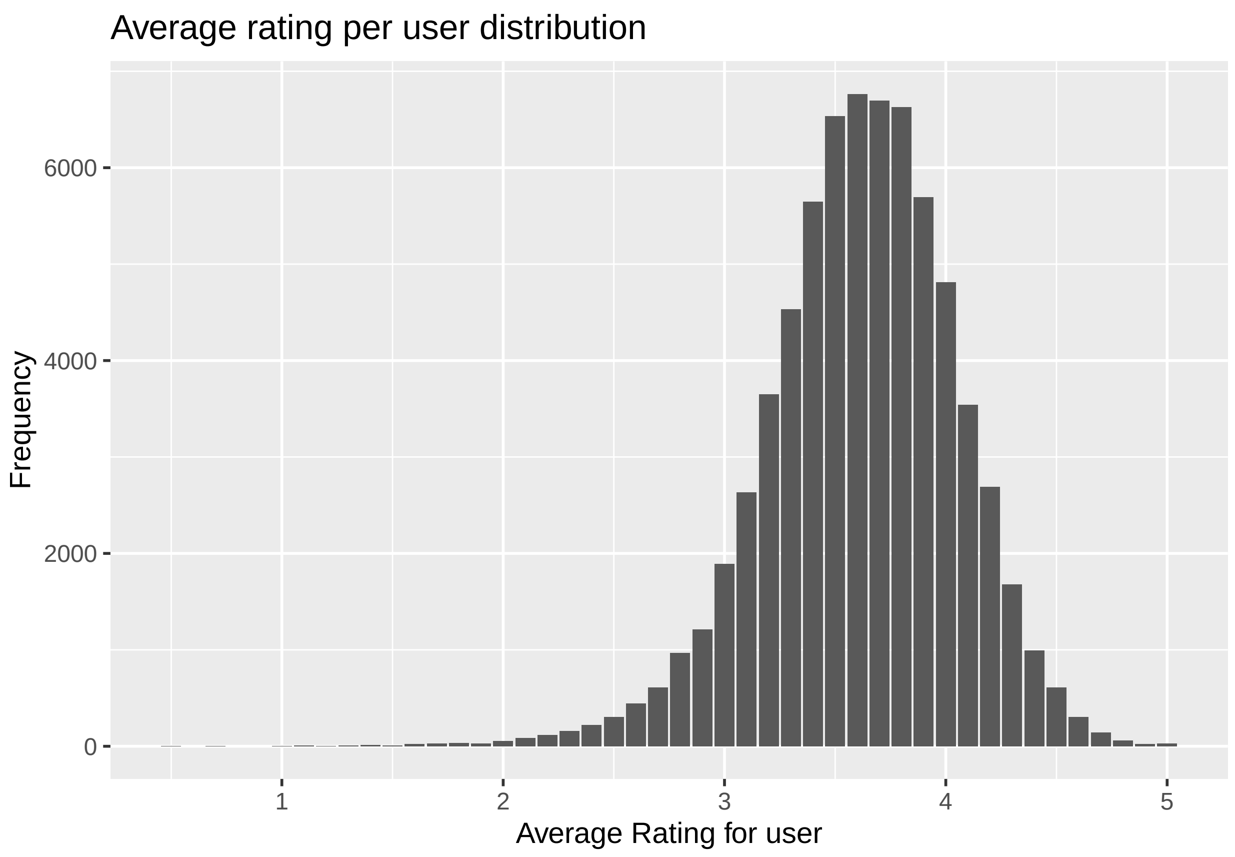
```r
edx %>%
  group_by(rating) %>%
  summarize(n = n()) %>%
  ggplot(aes(rating, n)) +
  geom_bar(stat = "identity") +
  labs(title = "Distribution of the ratings",
       x = "Rating",
       y = "Frequency")
```



We can see, that the ratings do vary significantly, as expected. High ratings seem to be more common than low ratings and full digits like 3.0 or 4.0 more common than 2.5 or 3.5 for example.

The following script will show us how the average rating per user is distributed:

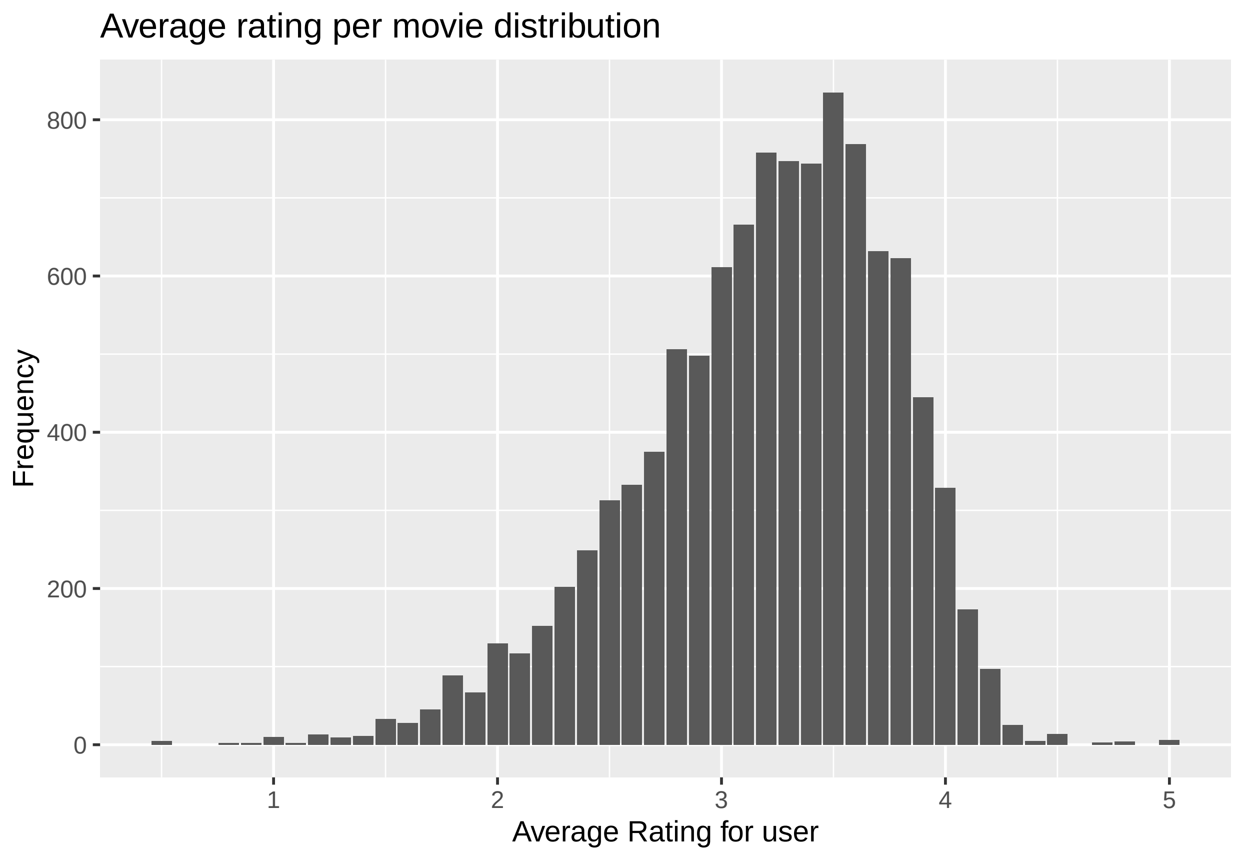
```r
edx %>%
  group_by(userId) %>%
  summarize(rating = round(mean(rating),1)) %>%
  ungroup() %>%
  group_by(rating) %>%
  summarize(n = n()) %>%
  ggplot(aes(rating, n)) +
  geom_bar(stat = "identity") +
  labs(title = "Average rating per user distribution",
       x = "Average Rating for user",
       y = "Frequency")
```



Looking at the average rating users are giving, some variance can be found, it almost looks like a bell curve. Average ratings per user seem to be to a large portion in the range of 3.0 to 4.0.

We can do the same for the average rating per movie:

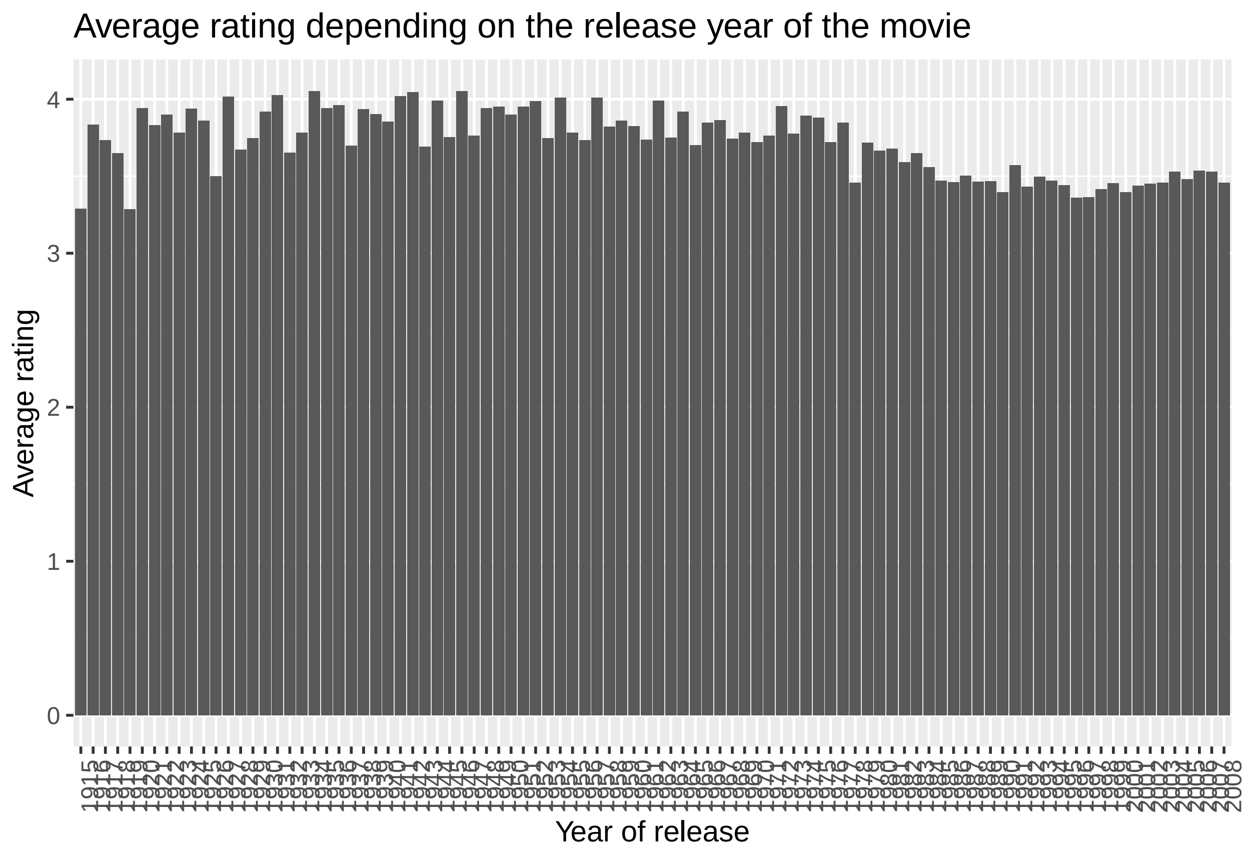
```
edx %>%
  group_by(movieId) %>%
  summarize(rating = round(mean(rating),1)) %>%
  ungroup() %>%
  group_by(rating) %>%
  summarize(n = n()) %>%
  ggplot(aes(rating, n)) +
  geom_bar(stat = "identity") +
  labs(title = "Average rating per movie distribution",
       x = "Average Rating for user",
       y = "Frequency")
```



The distribution for the average rating per movie seems to be distributed more widely, which indicates that we should take the individual movie into consideration when creating the model.

Does the year in which the movie has been published have an impact?:

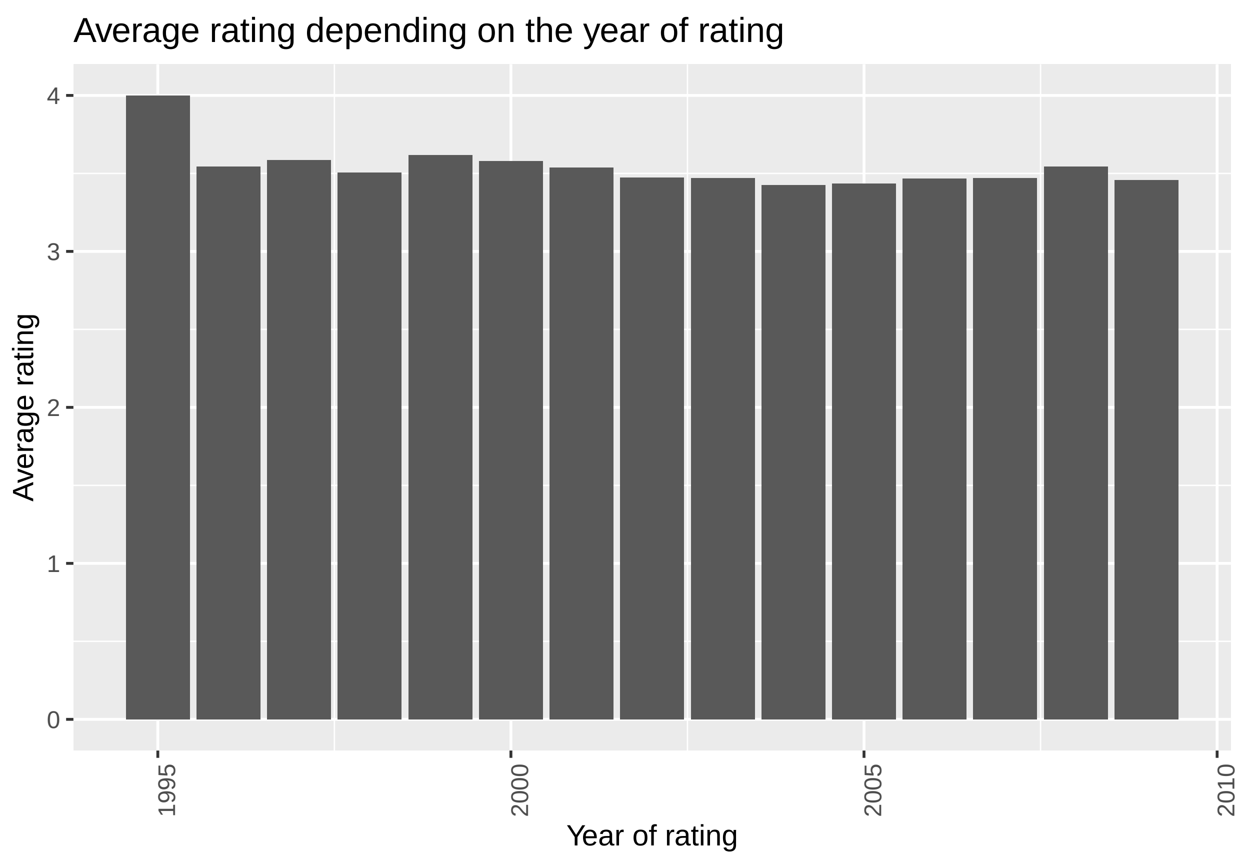
```
edx %>%
  group_by(year) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(year, rating)) +
  geom_bar(stat = "identity") +
  labs(title = "Average rating depending on the release year of the movie",
       x = "Year of release",
       y = "Average rating") +
 theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



The year in which the movie has been published also seems to have a slight influence. Newer movies (1980 and later) seem to be slightly lower rated than earlier movies, although the effect is not as strong as for the other two variables.

How about the year of the rating? Can we see some variance here?:

```r
edx %>%
  group_by(timestamp) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(timestamp, rating)) +
  geom_bar(stat = "identity") +
  labs(title = "Average rating depending on the year of rating",
       x = "Year of rating",
       y = "Average rating") +
 theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



The year in which the rating has been done seems to have no significant effect on the rating. Except for the higher rating in 1995 all other years seem to be relatively stable around 3.5, our average value.

A bit more complicated is the examination of the genres, due to the fact, that often more than one is listed for each movie. However we can do it like that. First we split up the ratings with more than one genre:

```r
temp <- edx %>% separate_rows(genres, sep = "\\|")
```

We can see that we have 19 unique genres and some movies without genre:

```r
length(unique(temp$genres))
```

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

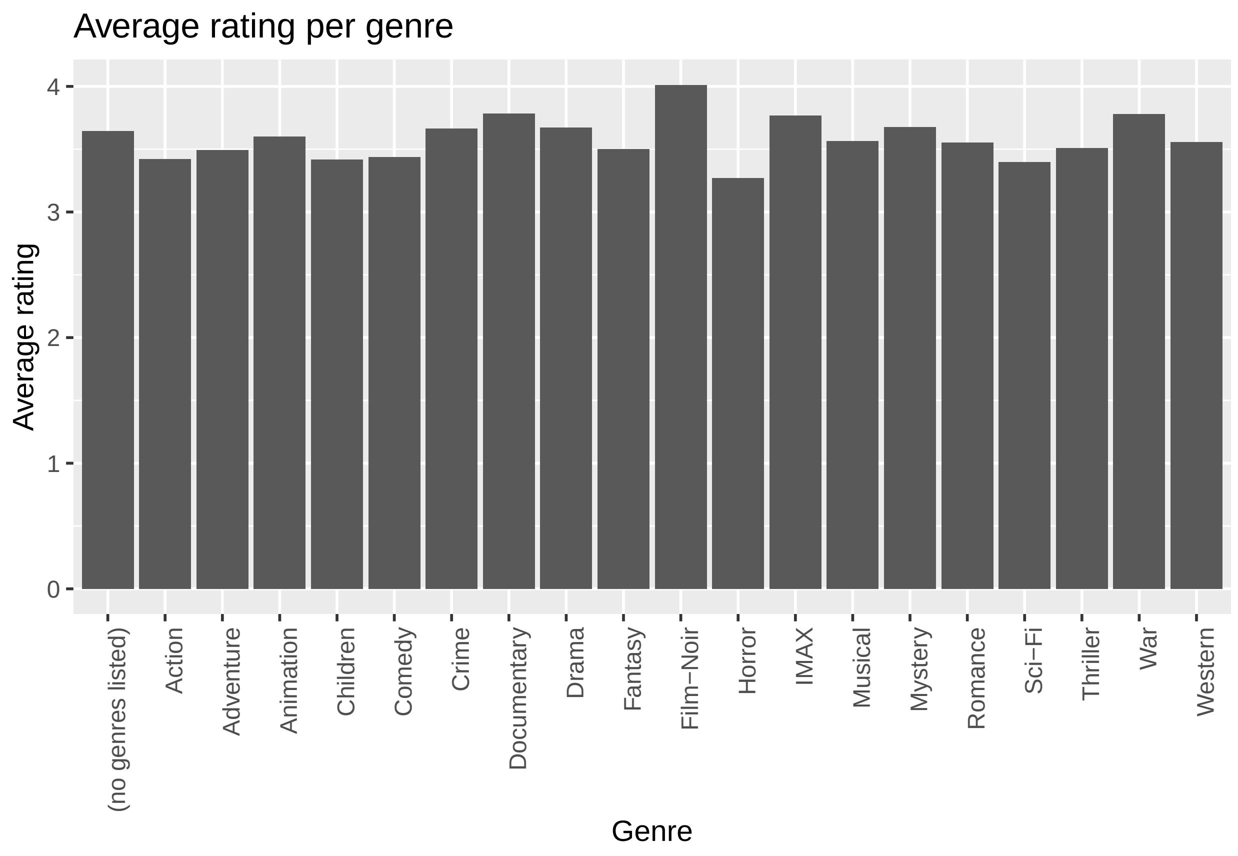
The average rating per genre and movies allocated movie per genre are as follows:

```r
temp %>% group_by(genres) %>% summarize(rating = mean(rating), n = n()) %>% arrange(rating)
```

```
## # A tibble: 20 x 3
##    genres             rating       n
##    <chr>               <dbl>   <int>
##  1 Horror               3.27  691485
##  2 Sci-Fi               3.40 1341183
##  3 Children             3.42  737994
##  4 Action               3.42 2560545
##  5 Comedy               3.44 3540930
##  6 Adventure            3.49 1908892
##  7 Fantasy              3.50  925637
##  8 Thriller             3.51 2325899
##  9 Romance              3.55 1712100
## 10 Western              3.56  189394
## 11 Musical              3.56  433080
## 12 Animation            3.60  467168
## 13 (no genres listed)   3.64       7
## 14 Crime                3.67 1327715
## 15 Drama                3.67 3910127
## 16 Mystery              3.68  568332
## 17 IMAX                 3.77    8181
## 18 War                  3.78  511147
## 19 Documentary          3.78   93066
## 20 Film-Noir            4.01  118541
```

A visual representation of the rating per genre:

```
temp %>% group_by(genres) %>% summarize(rating = mean(rating)) %>%
  ggplot(aes(genres, rating)) +
  geom_bar(stat = "identity")+
  labs(title = "Average rating per genre",
       x = "Genre",
       y = "Average rating") +
 theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



The data analysis suggests, that **movieId**, **userId** and **release year** seem to have the most significant impact on the rating. In addition the more complex genres column is worth a try. For this we need to seperate the genres column into several rows and put it together after model building and prediction More on that in the model building section.

3. Model building

In this section we will follow a stepwise approach for building the model. For testing the results we will use the RMSE function provided in course of this lecture.

```r
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

3.1 Creating the datasets

Defining test and train sets as well as splitting up all datasets by genres for including them into the model:

```r
# Define the index for train and test set
train_index <- sample(1:nrow(edx), 0.8 * nrow(edx))
test_index <- setdiff(1:nrow(edx), train_index)

# Create train and test set
train_set <- edx[train_index,]
test_set <- edx[test_index,]

# Make sure we don't have users and movies in the test set
# which are not in the training set
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Seperating the genres
edx_split <- edx %>% separate_rows(genres, sep = "\\|")
train_set_split <- train_set %>% separate_rows(genres, sep = "\\|")
test_set_split <- test_set %>% separate_rows(genres, sep = "\\|")
```

3.2 Models without regularization

We start with just taking the average rating across the dataset. **mu** contains the mean of all ratings.

```r
mu <- mean(train_set$rating)

# Predict ratings
predicted_ratings <- test_set %>%
    mutate(pred = mu)

# Calculate the RMSE
RMSE_mu <- RMSE(predicted_ratings$pred, test_set$rating)
```

Table 1: RMSEs of models

Method	RMSE
Average	1.060789

Next we add the movie effect to the model. We do that by using **b__i** as the mean rating difference of each movie to the mean ratings. We then predict the rating by adding the **b__i** term to our calculation.

```r
# Movie effect without regularization
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# Predict ratings
predicted_ratings <- test_set %>%
  left_join(b_i, by = "movieId") %>%
  mutate(pred = mu + b_i)

# Calculate the RMSE
RMSE_movie <- RMSE(predicted_ratings$pred, test_set$rating)
```

Table 2: RMSEs of models

Method	RMSE
Average	1.0607886
Movie Effect	0.9443593

We can see that we get a significant improvement over our baseline by just adding the movie effect to the model.

The same can be done for the user effect:

```r
# User effect without regularization
b_u <- train_set %>%
  left_join(b_i, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# Predict ratings
predicted_ratings <- test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u)

# Calculate the RMSE
RMSE_user <- RMSE(predicted_ratings$pred, test_set$rating)
```

Table 3: RMSEs of models

Method	RMSE
Average	1.0607886
Movie Effect	0.9443593
Movie + User Effect	0.8671375

Adding the user effect to the model gives a further significant improvement compared to the previous model.

Our analysis showed, that the release year also could have some impact on the prediction accuracy. It's implemeted as follows:

```r
# Release year effect without regularization
b_y <- train_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  group_by(year) %>%
  summarize(b_y = mean(rating - mu - b_i - b_u))

# Predict ratings
predicted_ratings <- test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_y, by = "year") %>%
  mutate(pred = mu + b_i + b_u + b_y)

# Calculate the RMSE
RMSE_release_year <- RMSE(predicted_ratings$pred, test_set$rating)
```

Table 4: RMSEs of models

Method	RMSE
Average	1.0607886
Movie Effect	0.9443593
Movie + User Effect	0.8671375
Movie + User + Release Year Effect	0.8667976

In this case the improvement is much smaller, but considering the fact, that the overall rating for RMSE in this project is very narrowly splitted it's seems worth the implementation.

Taking the genre into consideration is a bit more difficult. Herefor we need the splitted models which we created earlier. These models contain more than one row per rating if more than one genre was listed in the original dataset. With help of the splitted datasets we then calculate the genre effect and predict the ratings. Now in some cases we have more than one prediction for a certain ratingId. To make our final prediction we therefore have to take the average predicted rating per ratingId and make it our final prediction.

```r
# Genre effect without regularization
b_g <- train_set_split %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_y, by = "year") %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u - b_y))

# Predict ratings
predicted_ratings <- test_set_split %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_y, by = "year") %>%
  left_join(b_g, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_y + b_g)

# Combine predictions by taking the mean of the predicted rating per ratingId
predicted_ratings <- predicted_ratings %>% group_by(ratingId) %>%
  summarize(pred = mean(pred))

# Calculate the RMSE
RMSE_genre <- RMSE(predicted_ratings$pred, test_set$rating)
```

Table 5: RMSEs of models

Method	RMSE
Average	1.0607886
Movie Effect	0.9443593
Movie + User Effect	0.8671375
Movie + User + Release Year Effect	0.8667976
Movie + User + Release Year + Genre Effect	0.8666935

As you can see we again improve our model, although the effort for this improvement seems very high. Nevertheless we will keep these four variables and see what improvement we can get through regularization.

3.3 Models with regularization

In this section we will follow the same process as in the previous chapter but regularize the data. We define the vector **lambdas** which contains the test values for the penalization term of the regularization:

```
# Vector for the tuning parameters
lambdas <- seq(0, 10, 0.25)
```

Then we will use the sapply function to optimize the result by minimizing the RMSEs. Movie model:

```
# Compute the RMSE for all elements in our lambdas vector
rmses <- sapply(lambdas, function(l){

  # Mean value of the rating for the train_set
  mu <- mean(train_set$rating)

  # Movie effect including regularization
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  # Predict the rating for the test set
  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)

  # Return the RMSE between prediction and real rating
  return(RMSE(predicted_ratings, test_set$rating))
})

# Storing the lowest RMSE and the associated lambda
RMSE_movie_reg <- min(rmses)
lambda_movie <- lambdas[which.min(rmses)]
```

Adding the user effect:

```
# Compute the RMSE for all elements in our lambdas vector
rmses <- sapply(lambdas, function(l){

  # Mean value of the rating for the train_set
  mu <- mean(train_set$rating)

  # Movie effect including regularization
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  # User effect including regularization
  b_u <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n()+l))

  # Predict the rating for the test set
  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  # Return the RMSE between prediction and real rating
  return(RMSE(predicted_ratings, test_set$rating))
})

# Storing the lowest RMSE and the associated lambda
RMSE_user_reg <- min(rmses)
lambda_user <- lambdas[which.min(rmses)]
```

The release year effect:

```
# Compute the RMSE for all elements in our lambdas vector
rmses <- sapply(lambdas, function(l){
```

```r
  # Mean value of the rating for the train_set
  mu <- mean(train_set$rating)

  # Movie effect including regularization
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  # User effect including regularization
  b_u <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n()+l))

  # Release year effect including regularization
  b_y <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - mu - b_i - b_u)/(n()+l))

  # Predict the rating for the test set
  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = "year") %>%
    mutate(pred = mu + b_i + b_u + b_y) %>%
    pull(pred)

  # Return the RMSE between prediction and real rating
  return(RMSE(predicted_ratings, test_set$rating))
})

# Storing the lowest RMSE and the associated lambda
RMSE_release_year_reg <- min(rmses)
lambda_release_year <- lambdas[which.min(rmses)]
```

And finally the genre, which is as mentioned in the previous section calculated based on the split data:

```r
# Compute the RMSE for all elements in our lambdas vector
rmses <- sapply(lambdas, function(l){

  # Mean value of the rating for the train_set
  mu <- mean(train_set$rating)

  # Movie effect including regularization
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  # User effect including regularization
  b_u <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n()+l))

  # Release year effect including regularization
  b_y <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - mu - b_i - b_u)/(n()+l))

  # Genre effect based on the split data including regularization
  b_g <- train_set_split %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = "year") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u - b_y)/(n()+l))

  # Predict the rating for the splitted test set
  predicted_ratings <- test_set_split %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = "year") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_y + b_g)

  # Taking the mean per ratingId
  predicted_ratings <- predicted_ratings %>% group_by(ratingId) %>%
    summarize(pred = mean(pred), rating = mean(rating))

  # Return the RMSE between prediction and real rating
  return(RMSE(predicted_ratings$pred, predicted_ratings$rating))
})

# Storing the lowest RMSE and the associated lambda
RMSE_genre_reg <- min(rmses)
lambda_genre <- lambdas[which.min(rmses)]
```

We can take a look at the optimization parameter for the full model by plotting lambda against RMSE:

```r
qplot(lambdas, rmses,
      main = "Effect of regularization on RMSE",
      xlab = "Lambda", ylab ="RMSE")
```

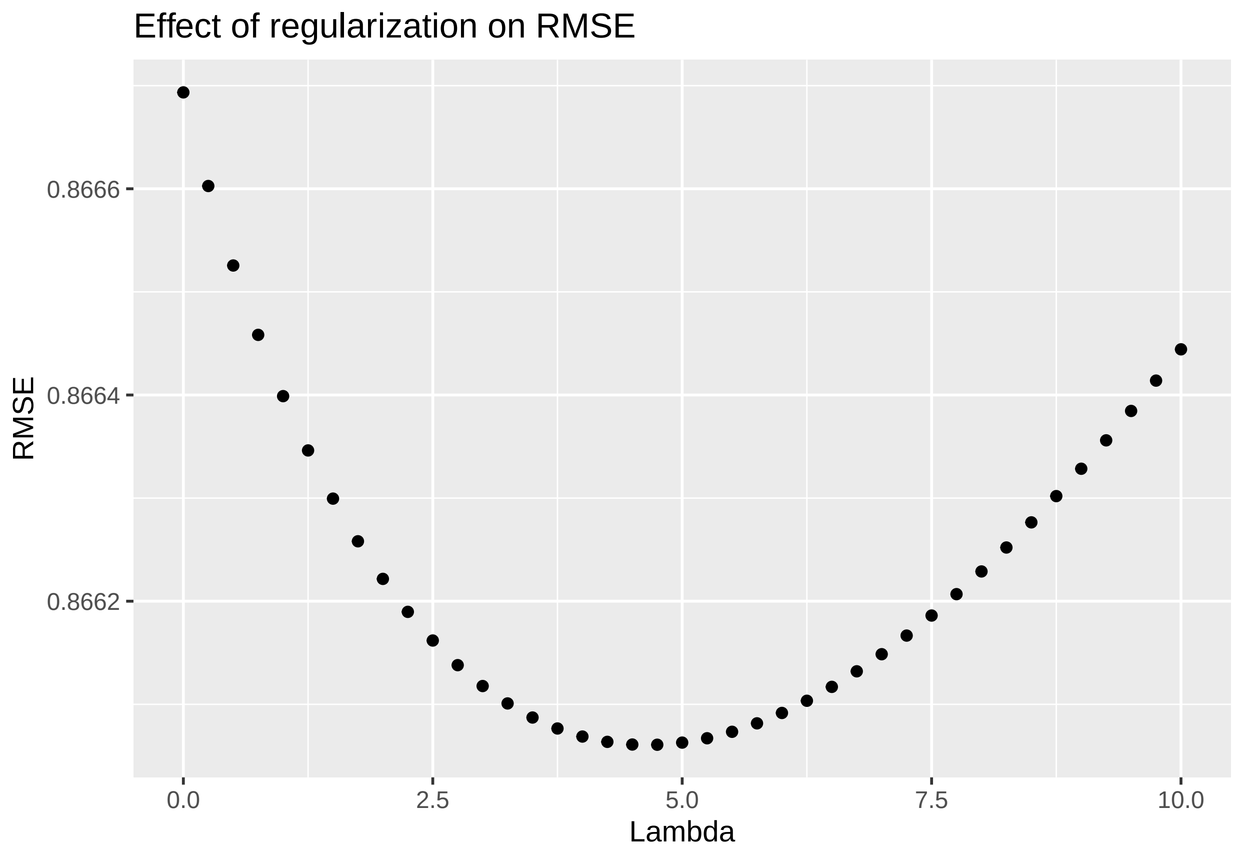


Table 6: RMSEs of models

Method	RMSE
Average	1.0607886
Movie Effect	0.9443593
Movie + User Effect	0.8671375
Movie + User + Release Year Effect	0.8667976
Movie + User + Release Year + Genre Effect	0.8666935
Movie Effect Reg.	0.9443031
Movie + User Effect Reg.	0.8664463
Movie + User + Release Year Effect Reg.	0.8661582
Movie + User + Release Year + Genre Effect Reg.	0.8660608

The table above summarizes the results of the different models. As we can see, the best model is the one taking all four variables into consideration and regularizing all the terms. This is also the model we will now use for the validation dataset.

4. Results

For validating the model we will calculate the factors on the whole edx dataset using the optimized lambda from the full model above. The program looks exactly the same as before, excluding the optimization loop of course.

```r
### Using the whole edx dataset for the building of the new model
### which we will then test on the validation set
# Mean value of the rating for the whole edx set
lambda <- lambda_genre
mu <- mean(edx$rating)

# Movie effect including regularization
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

# User effect including regularization
b_u <- edx %>%
  left_join(b_i, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n()+lambda))

# Release year effect including regularization
b_y <- edx %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  group_by(year) %>%
  summarize(b_y = sum(rating - mu - b_i - b_u)/(n()+lambda))

# Genre effect based on the split data including regularization
b_g <- edx_split %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_y, by = "year") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - b_i - b_u - b_y)/(n()+lambda))

# Predict the rating for the validation set
predicted_ratings <- validation
# Seperate by genre
predicted_ratings <- predicted_ratings %>% separate_rows(genres, sep = "\\|")

# Prediction for the split data
predicted_ratings <- predicted_ratings %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_y, by = "year") %>%
  left_join(b_g, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_y +b_g)

# Summarizing the data to get one prediction per ratingId
predicted_ratings <- predicted_ratings %>% group_by(ratingId) %>%
  summarize(pred = mean(pred))

# Calculation the RMSE on the validation set
RMSE_validation <- RMSE(predicted_ratings$pred, validation$rating)
```

Table 7: RMSEs of models

Method	RMSE
Average	1.0607886
Movie Effect	0.9443593
Movie + User Effect	0.8671375
Movie + User + Release Year Effect	0.8667976
Movie + User + Release Year + Genre Effect	0.8666935
Movie Effect Reg.	0.9443031
Movie + User Effect Reg.	0.8664463
Movie + User + Release Year Effect Reg.	0.8661582
Movie + User + Release Year + Genre Effect Reg.	0.8660608
Validation - full model	0.8644231

As we can see, the model achieves our set target to get an RMSE < 0.8649, by using the optimized λ value of 4.75. Using the full edx dataset for a final calculation of the parameters for the optimized approach leads to an even better result on the validation set than on the test done before.

The model building process in the previous section shows a very consistent improvement by adding in more variables for the model bulding. The most significant improvements were made using **movieId** and **userId** and the model accuracy was raised even more using **release year** and **genre**. The improvement achieved with these variables was much smaller.

Futher improvements were made by implementing regularization to the model and optimizing the penelization term. The direction of the results were consistent with the ones obtained without this term. The RMSEs while using the regularization were better/lower than without.

Testing the optimized model on the validation set (the model was redefined before using the whole edx dataset with the optimized lambda for the best model) brought the targeted result.

5. Conclusion, limitations and comments

The model build during this project was complex in the end. Most of the complexity was added by the need to split up the genre column due to multiple assignments. Allocating every movie to just one genre would reduce the effort significantly. We can also see, that a majority of the accuracy can be achieved by a very small effort, the unregularized model just using the movie and user effect achieved an RMSE of 0.8671375 compared to the final result of **0.8644231**. The computational effort added for this minimal improvement was huge.

These methods are very basic in my point of view although they achieve an accuracy which is sufficient for maximum points on the grading scale. Trying more complex/ressource intensive (RAM) methods was unfortunately not possible. Setting up the **ratingMatrix** with the **recommenderlab package** (required 5.5 GB of RAM) worked, but further computation was not possible due to hardware limitations. I wanted to try out matrix factorization (SVD / PCA) in order to identify patterns in movie or user groups and use them for further redefining the model. Unfortunately it was not possible.

First of all, I really liked the project and the challenges that came with it. Going step by step through some of the things learned during the pervious courses and applying them by trial and error was really fun. Maybe it would be better to just use the 1 Mio. dataset for this project and set the bar higher in terms of grading, making it possible and mandatory to implement more advanced methods. It's a real shame that I couldn't dig in deeper when the best part was about to come due to the afore mentioned limitations.

Hope you enjoyed the project and this report as well and maybe got some new insights you can use in the future. You can find the data **here**