Capstone_MovieLens

Section 1

Introduction:

In this R project I will create a prediction system through machine learning training. Datasource is the Movielens 10M Dataset, released 1/2009. As algorithm, linear regression will be used. Further methods will be only be used if the results are not satisfying. As validation of the models, the RMSE (Root Mean Square Error) is used.

Note: Any code provided is based on R v3.6.1

```
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")
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(dslabs)) install.packages("dslabs", repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(kableExtra)) install.packages("kableExtra", repos = "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-project.org")
if(!require(tictoc)) install.packages("tictoc", repos = "http://cran.us.r-project.org")

# Load libraries
library(dslabs)
library(dplyr)
library(caret)
library(tidyverse)
library(ggplot2)
library(data.table)
library(kableExtra)
library(knitr)
library(tictoc)
```

At this point we have installed all the necessarry packages and are ready to go.

Data preparation

Now we load the original dataset and take a quick look at it to get an idea of the data structure.

```
tic("downloading data") # set timer for download
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
toc() # end timer for download
```

```
## downloading data: 9.25 sec elapsed
```

```
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))
# join content of both files
movielens <- left_join(ratings, movies, by = "movieId")
```

The movielens object now contains all the moviedata and their ratings. This is how the datastructure we are dealing with looks like:

```
str(movielens)
```

```
## 'data.frame':    10000054 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983653 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller"
```

In sum, we have 10000054 rows of data. Each row, representing one rating for a movie by a user (="observation"), consisting of 6 variables:

- 2 variabels of type integer (userId, timestamp).
  - The timestamp represents the moment in time the rating was given. In order to have a human-readable datetime information like year, month and day, we will convert the timestamp later.
- 2 variables of type numeric (movieId, rating)
- 2 variables of type character (title, genres).
  - The title information also seems to include the year the movie was published. So - beyond the given 6 variabels - this is one more potential information that could be used for analysis.
  - As we can see, genres are a concenated string, that can contain multiple values. (A movie can be categorized into more than one genre.) In order to extract all the genres-information, we will separat these later.

For our later goal to train model that can predict ratings, we need 2 subsets of the complete "movielens" dataset, one for training purposes, one to validate our trained model. The training set will be 90% of the movielens data, validation set 10%.

```
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, removed) #cleanup
#(not removing movielens dataset for analysis)
```

Section 2

Overview Dataset:

Now that we have our original data available and know basic the data structure, let's learn something about the dataset itself by exploring what range of values the 6 variables have. How many different users and movies exists? What genres does exists? What period of ratings are available? What is the scale of ratings? ...

Starting point is a simple summary of the dataset.

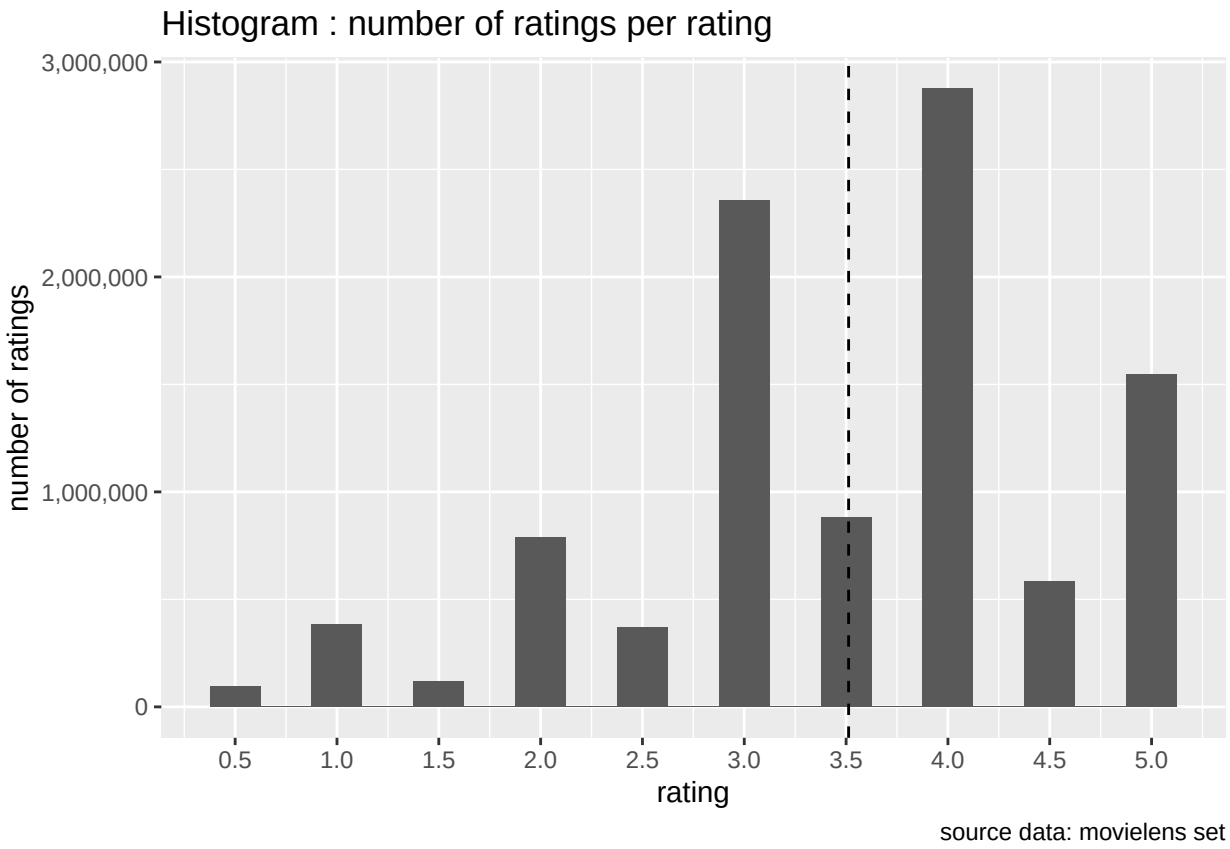
```
summary(movielens)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18123   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35741   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53608   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:10000054    Length:10000054
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

We see the distributions of the numerical variables. Nothing very insightful here, but we can see that the ratings given are on a scale from 0.5 to 5; and half-point ratings are possible.

Let's have a closer look on the ratings.

```
ggplot(movielens, aes(x= movielens$rating)) +
    geom_histogram( binwidth = 0.25) +
    scale_x_continuous(breaks=seq(0.5, 5, by= 0.5)) +
    scale_y_continuous(labels = function(x) format(x, big.mark = ",", scientific = FALSE)) +
    labs(x="rating", y="number of ratings", caption = "source data: movielens set") +
    ggtitle("Histogram : number of ratings per rating") +
    geom_vline(xintercept = mean(movielens$rating), linetype = "dashed")
```



- 4.0 is the most given rating
- whole point ratings are for more common than half point ratings
- the average rating itself (3.5) is given not so often

```
movielens %>% summarize(distinct_users = n_distinct(userId), distinct_movies = n_distinct(movieId))
```

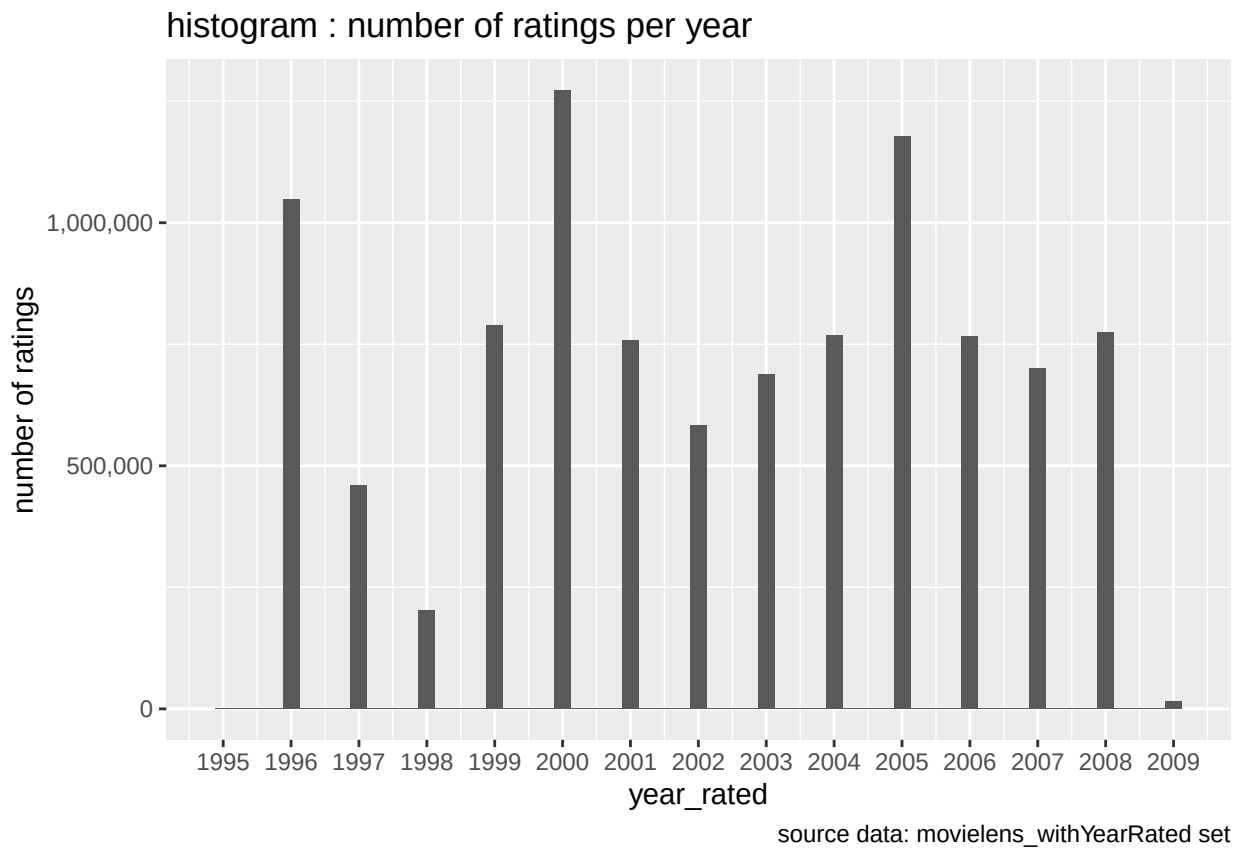
```
##   distinct_users distinct_movies
## 1          69878           10677
```

- These ratings where given by 69878 different users, 10677 different movies were rated.

To get an impression, when the ratings were given and therefore get an idea how old the observations are, we need to manipulate the timestamp variable to get extract the year information.

```
movielens_withYearRated <- movielens
movielens_withYearRated <- movielens_withYearRated %>% mutate(year_rated = as.POSIXct(timestamp, origin
movielens_withYearRated$year_rated <- as.numeric(format(movielens_withYearRated$year_rated, "%Y"))

ggplot(movielens_withYearRated, aes(x= movielens_withYearRated$year_rated)) +
    geom_histogram( binwidth = 0.25) +
    scale_x_continuous(breaks=seq(1995, 2009, by= 1)) +
    scale_y_continuous(labels = function(x) format(x, big.mark = ",", scientific = FALSE)) +
    labs(x="year_rated", y="number of ratings", caption = "source data: movielens_withYearRated set") +
    ggtitle("histogram : number of ratings per year")
```



(Hint: 1995 has 3 ratings)

- We see three dominant peaks in the years 1996, 2000 and 2005.

What movies where rated in these years?

```
top_titles_1996 <- movielens_withYearRated %>%
    filter(year_rated == 1996) %>%
    group_by(title) %>%
    summarize(no_ratings=n()) %>%
    top_n(10, no_ratings) %>%
    arrange(desc(no_ratings))
top_titles_2000 <- movielens_withYearRated %>%
    filter(year_rated == 2000) %>%
    group_by(title) %>%
    summarize(no_ratings=n()) %>%
    top_n(10, no_ratings) %>%
    arrange(desc(no_ratings))
top_titles_2005 <- movielens_withYearRated %>%
    filter(year_rated == 2005) %>%
    group_by(title) %>%
    summarize(no_ratings=n()) %>%
    top_n(10, no_ratings) %>%
    arrange(desc(no_ratings))

top_table <- knitr::kable(list(top_titles_1996, top_titles_2000, top_titles_2005),
```

Table 1: Top 10 most rated movies in the years 1996, 2000 and 2005

title	no_ratings
Batman (1989)	13365
Dances with Wolves (1990)	12744
Apollo 13 (1995)	12711
Fugitive, The (1993)	12142
Pulp Fiction (1994)	12076
True Lies (1994)	12050
Forrest Gump (1994)	11105
Batman Forever (1995)	10974
Aladdin (1992)	10957
Jurassic Park (1993)	10850

title	no_ratings
American Beauty (1999)	4243
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	3930
Star Wars: Episode V - The Empire Strikes Back (1980)	3820
Star Wars: Episode VI - Return of the Jedi (1983)	3647
Saving Private Ryan (1998)	3493
Silence of the Lambs, The (1991)	3484
Jurassic Park (1993)	3451
Terminator 2: Judgment Day (1991)	3417
Matrix, The (1999)	3407
Fargo (1996)	3374

title	no_ratings
Lord of the Rings: The Two Towers, The (2002)	3612
Lord of the Rings: The Fellowship of the Ring, The (2001)	3226
Lord of the Rings: The Return of the King, The (2003)	3222
Matrix, The (1999)	3139
Pulp Fiction (1994)	2956
Shrek (2001)	2956
Forrest Gump (1994)	2911
Shawshank Redemption, The (1994)	2864
Silence of the Lambs, The (1991)	2819
Incredibles, The (2004)	2807

```
                        caption = "Top 10 most rated movies in the years 1996, 2000 and 2005")
top_table
```

- In both, 2000 and 2005, we see that very popular Hollywood-Blockbuster triologies are among the most rated movies (star Wars and Lord of the Rings respectively). This could imply the effect, that new releases of those series draw attention also to their predecessors, resulting in ratings by users.
- In 1996, the same effect could be present in the case of "Batman".
  - We also see much higher number of ratings in 1996 for the most rated movies in general.
  - This can not be explained that the database was "new" and therefore more popular, because the number of ratings per year dont show a descending trend.
  - Instead, just a higher percentage of the ratings per year given fall to the most popular movies.
  - This would mean that the diversity of movies should have grown bigger in the later years, and in the first years mostly the popular movies were rated.

Lets check this by exermining the movie-diversity growth by year. Therefore we extract the release year information that is included in the title column and add it as separat column.

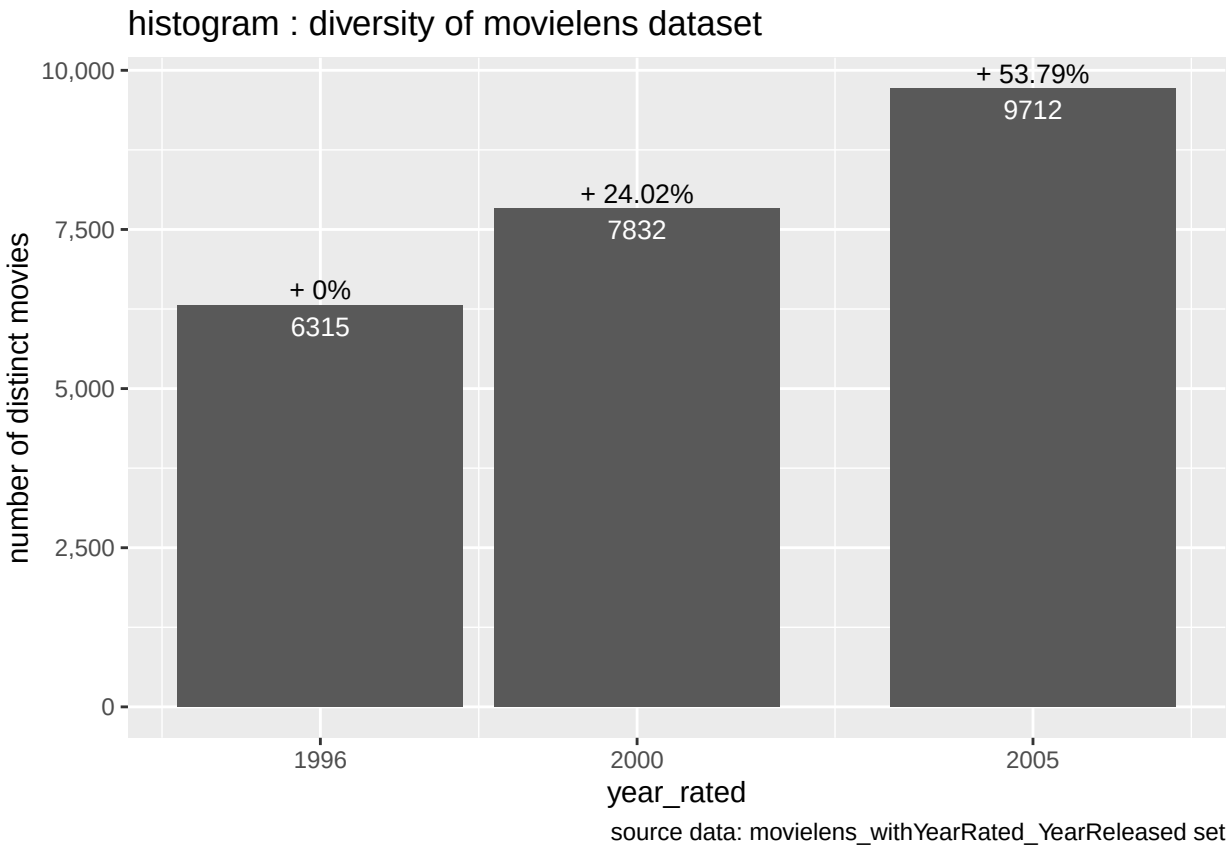
```
year_released <- stringi::stri_extract(movielens_withYearRated$title, regex = "(\\d{4})", comments = TRU

movielens_withYearRated_YearReleased <- movielens_withYearRated
movielens_withYearRated_YearReleased <-  movielens_withYearRated_YearReleased %>%
  mutate(year_released = year_released)

years <- seq(1995, 2009, 1)
for(year in years){
  distinct_movies <- movielens_withYearRated_YearReleased %>% filter(year_released <= year) %>% summari
  resultlist_distinct_movies_per_year <- if(exists("resultlist_distinct_movies_per_year")) c(resultlist_
}
# re-structure data
results_distinct_movies_per_year <- as.data.frame(resultlist_distinct_movies_per_year)
names(results_distinct_movies_per_year) <- seq(1,15,1)
results_distinct_movies_per_year <- results_distinct_movies_per_year %>% t() %>% cbind(as.data.frame(yea
colnames(results_distinct_movies_per_year) <- c("distinct_movies", "year")

# for plot, only the three years in question are needed
results_distinct_movies_per_year <- results_distinct_movies_per_year  %>%
  filter(year==1996 | year==2000 | year == 2005)
# add growthrates
results_distinct_movies_per_year$growth_rate_from1996 = 0
results_distinct_movies_per_year[2,3] = (results_distinct_movies_per_year[2,1] / results_distinct_movies
results_distinct_movies_per_year[3,3] = (results_distinct_movies_per_year[3,1] / results_distinct_movies

ggplot(results_distinct_movies_per_year, aes(x= year, y=distinct_movies, Z=growth_rate_from1996)) +
    geom_bar(stat="identity") +
    geom_text(aes(label=distinct_movies), vjust=1.6, color="white", size=3.5) +
    geom_text(aes(label=sprintf("+ %s",paste((round(growth_rate_from1996,4)*100), "%", sep=""))), vjust=
    scale_x_continuous(breaks=c(1996, 2000, 2005)) +
    scale_y_continuous(labels = function(x) format(x, big.mark = ",", scientific = FALSE)) +
    labs(x="year_rated", y="number of distinct movies", caption = "source data: movielens_withYearRated_
    ggtitle("histogram : diversity of movielens dataset")
```



- The number of movies represented in the movielens data has grown from 1996 to 2000 by 24%, from 1996 to 2005 by 59%.
- Since the number of ratings per year did not grow by these rates, this can a reason why there are less rating per year per movie in the later years (for the most popular movies we looked at here)

There can be done a lot more analysis of the movielens dataset, i.e. looking at the users or genres etc. But lets jump to the core target of this project, which is to build a model to predict user ratings. Let's get to the training part and train a model, that can predict ratings, how users would rate a movies.

Training Method

As stated before, we use the RMSE, as defined in the Machine Learning Course, lesson "Recommendation Systems" to evaluate a models performance:

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

where:

- $\hat{y}_{u,i}$ is the true rating of user u for movie i
- $y_{u,i}$ is the predicted rating of user u for movie i

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

> Comment: In the Machine Learning Course, lesson "Regularization", Prof. Irizarry showcased code using linear regression models with different tweaks with the following results:

method	RMSE
Just the average	1.048
Movie Effect Model	0.986
Movie + User Effects Model	0.885
Regularized Movie + User Effect Model	0.881

> The results just got down to a RMSE of 0.881, with Models with Regularized Movie + User Effect performing best. So, going from this, I will continue using a Regression Model incorporating both Regularized Movie Effect & Regularized User Effect and try to optimise the model to reach a better RMSE.

As target performance, an RMSE of 0.8649 was defined.

For explanation, the Movie effect is a penelizing factor to capture the associated movie bias. For example some movies are rated higher than others (i.e. blockbusters are rated higher then niche movies). The User effect captures the associated user bias. So user ratings may be influenced by personal liking/disliking (of a genre, an actor...) regardless of the movie itself.

So in summary, the prediction model will follow this approach: $y_{u,i} = \mu + b_i + b_u$ where:

- μ the mean rating
- b_i movie effect of movie i
- b_u user effect of user u with
- $b_i = \sum_i(\hat{y} - \mu)/(n_i + \lambda)$
- $b_u = \sum_{u,i}(\hat{y} - \mu - b_i)/(n_i + \lambda)$

```r
target_RMSE <- 0.8649 # given in the project assignment
mean_edx <- mean(edx$rating) #define mean of ratings in the dataset used for training

lambda_tuning <- seq(0, 10, 0.1) #define interval for optimise tuning parameter lambda
for(lambda in lambda_tuning){    # loop through values for lambda to optimise model
  b_movie <-
    edx %>%
    group_by(movieId) %>%
    summarize(b_movie = sum(rating - mean_edx)/(n()+lambda)) # calculate reg. movie effect per movieId
  b_user <-
    edx %>%
    left_join(b_movie, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_user = sum(rating - b_movie - mean_edx)/(n()+lambda)) # calculate user effect per userI
  predicted_ratings <-
    validation %>%
```

```
    left_join(b_movie, by = "movieId") %>%                # append movie effect to validation set
    left_join(b_user, by = "userId") %>%                  # append user effect to validation set
    mutate(prediction = mean_edx + b_movie + b_user) %>%  # calculate prediction
    .$prediction

  model_rmse <- calculateRMSE(validation$rating, predicted_ratings) # calculate model performance

  results_rmse <- if(exists("results_rmse")) c(results_rmse, model_rmse) else c(model_rmse) # create/ap

  if(model_rmse <= target_RMSE){ # stopping algorithm as soon as target RMSE is reached in order to sho
    break()
  }
}
```

Section 3

Results

```
lambda_optimal <- lambda_tuning[which.min(results_rmse)] # value of lambda for best model trained
print("Value of lambda for the best model trained:")
```

```
## [1] "Value of lambda for the best model trained:"
```

```
lambda_optimal
```

```
## [1] 2.9
```

```
rmse_optimal <- min(results_rmse) # RMSE of best model trained
print("RMSE of the best model trained.")
```

```
## [1] "RMSE of the best model trained."
```

```
rmse_optimal
```

```
## [1] 0.8648972
```

- The model performance improved and the target RMSE was reached.

Finally, let's have a look at some of the predicted ratings to get a feeling how precise they are. (Since the algorithm stopped with the final result, the predicted ratings of this model are still in the predicted_ratings variable.) For comparison, they are rounded to the nearest possible rating value (only full and half star ratings are possible) and also displayed side by side with the real rating.

```
pred25 <- predicted_ratings[1:25]
pred25_rounded <- round(pred25/0.5)*0.5
realRatings25 <- validation$rating[1:25]
rating_comparsion_25 <- data.frame(pred25, pred25_rounded, realRatings25)
names(rating_comparsion_25) <- c("prediction rating", "rounded prediction rating", "real rating")
```

Table 3: Comparsion of 25 ratings predictions and real ratings:

prediction rating	rounded prediction rating	real rating
4.391980	4.5	5.0
5.120261	5.0	5.0
4.512502	4.5	5.0
3.328065	3.5	3.0
4.213232	4.0	2.0
2.743516	2.5	3.0
3.984630	4.0	3.5
4.148017	4.0	4.5
4.307772	4.5	5.0
3.336911	3.5	3.0
3.677784	3.5	3.0
3.592929	3.5	3.0
3.678907	3.5	3.0
4.148959	4.0	3.0
3.485113	3.5	3.0
4.223516	4.0	3.0
3.752477	4.0	3.0
3.468601	3.5	3.0
4.286329	4.5	4.0
3.878108	4.0	5.0
3.858222	4.0	3.0
4.083048	4.0	4.0
4.186953	4.0	4.0
4.259045	4.5	4.0
4.543153	4.5	5.0

```
rating_comparsion_table <- knitr::kable(rating_comparsion_25,
                          caption = "Comparsion of 25 ratings predictions and real ratings:")
rating_comparsion_table
```

In this small sample we see that the prediction often is off by a half star, but differs only once by two stars. So we get an impression that the model is in fact able to do predictions for users's ratings of movies.

Section 4

Conclusion

In this MovieLens project some insights on the dataset itself were provided as well as a recommender system using a regression algorithm to predict movie ratings for the 10M version of the data. Based on separate training set (edx) and validation set, linear regression models were successfully trained. The model's performances were evaluated through the RMSE (root mean squared error) and showed that the Linear regression model with regularized effects on users and movies was good enough to reach the performance we aimed at. With a penelizing factor of lambda = 2.9, a RMSE of 0.8648972 was reached.

Limitations & Suggestions for future work

The data analysis does not make claim to be complete, but should give some ideas who some questions to the data could be answered. As mentioned before, there might be other interesting insigts to the data not

covered here. Coming to the prediction model, it was shown that linear regression doesperform pretty good on the data. But, to shorten calculation time and given limited computation resources, we did to sqeeze to algorithm to the limit here. Maybe, the same linear regression model can be optimised further. Also, there were on other algorithms or recommender systems evaluated. It might be interesting to see, how other algorithms perform in comparison.