Model for the Prediction of User Ratings within the Movielense Dataset

Jörg Duschmalé

5/19/2021

Abstract

Recommendation systems are an indispensable feature of the digital economy from commerce (e.g. Amazon) to personalized advertisement (e.g. google) to movie recommendations for streaming services (e.g. Netflix). In this paper the development of a simple model to predict movie ratings of users based on the MovieLens dataset is described. Data analysis reveals predictive features that are then used to build a model. Finally, the obtained insights are combined and the resulting model is assessed.

Introduction

In 2006 Netflix announced a competition (the "Netflix prize") for data scientists, where they offered one million dollars to the winning team that managed to improve their in-house movie rating algorithm by a margin of at least 10% (Lohr 2009). Inspired by the Netflix prize it is the capstone project of a HarvardX Professional Certificate in data science (Irizarry 2021) to build a recommendation system using the MovieLens dataset (Harper and Konstan 2016) making use of the the content thought throughout the course. The MovieLens dataset is the collection of movie ratings of users on the MovieLens recommender system over the course of years. There are several differently sized datasets. The one used below is the ML 10M dataset which consists of more than 10 million ratings by nearly 70'000 users on more than 10'000 movies between the years of 1995 and 2009. The data can be obtained at http://files.grouplens.org/datasets/movielens/ml-10m.zip .

While the solution actually winning the Netflix prize was a combination of techniques some of which are beyond the scope of this course (Chen n.d.) a decent recommenation system can be constructed using a relatively simple approach of normalization of different effects followed by matrix factorization on the remaining variability. The goal of the project was to use nine tenth of the available dataset in order to build a recommendation system able to predict movie ratings within the remaining 10% of the data with an root mean sqared error (RMSE) of less than 0.86490.

Herein the building of a recommendation system based on the normalization of several factors (such as user effects, movie effects, date effects and genre effects) is described. Finally, the remaining errors are subjected to a matrix factorization algorithm, greatly improving the final result.

Downloading The Movielens Dataset and Data Analysis

Downloading the data and data partition

The data is downloaded and separated into a training and a test set using code provided in the course material of the HarvardX Professional Certificate in data science (Irizarry 2021). First the necessary packages are loaded.

```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(dslabs)
library(caret)
library(dplyr)
library(tidyverse)
library(lubridate)
library(recosystem)
```

After loading of the required packages, the dataset is downloaded and separated into a training set (edx) as well as a test set (validation) for the final validation of the code. This is achieved by the code below (Irizarry 2021).

```r
# 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(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)`
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)
```

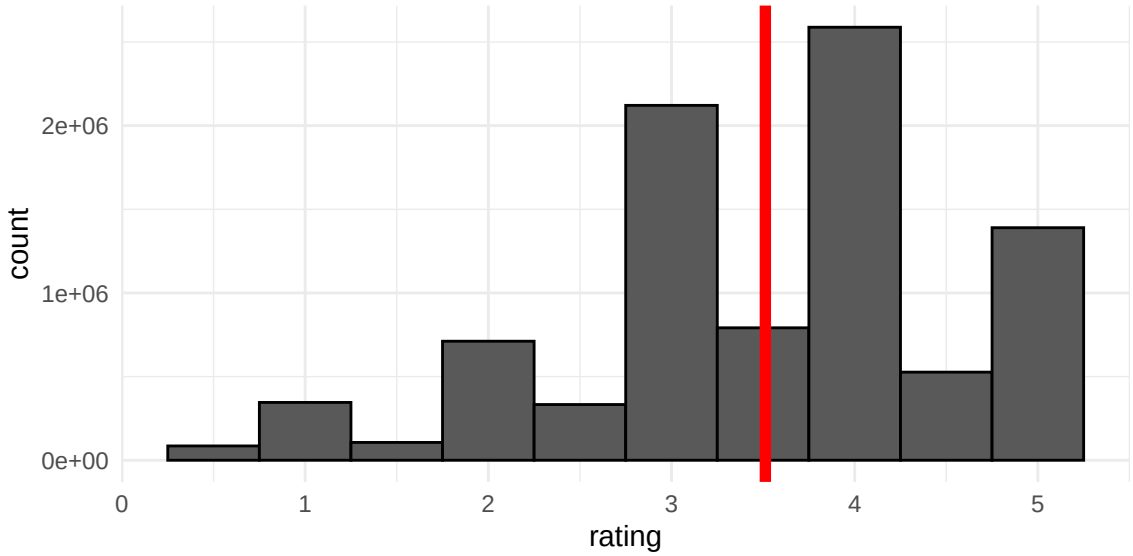
Data analysis

Structure and Summary The edx dataset, which is used here to create and train the prediction model is a collection of individual unique movie ratings (i.e. movie-user combinations). A closer look at the structure and the summary of the dataset (see below) reveals that it consists of 6 columns representing an Id for the user giving the rating, an Id of the movie in question, the rating itself (as a mark between 0.5 and 5 with 0.5 steps possible), a timestamp when the rating was given (in seconds after "1970-01-01 00:00:00 UTC"), the title (and the year of appearance) of the movie as well as a set of genres collectively describing the movie in question.

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
##  - attr(*, ".internal.selfref")=<externalptr>
```

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

Ratings Below is a histogram of the ratings with the average rating depicted as a vertical red line.



From the histogram it is immediately obvious that people are much more likely to give integer marks rather than 0.5 steps and that there is a certain bias to rate movies favourably (also reflected in the mean mark of 3.51).

Movies When only looking at movies that received at least 150 ratings (thus excluding "speciality" movies only of interest to a few people) the most favoured movies are the following:

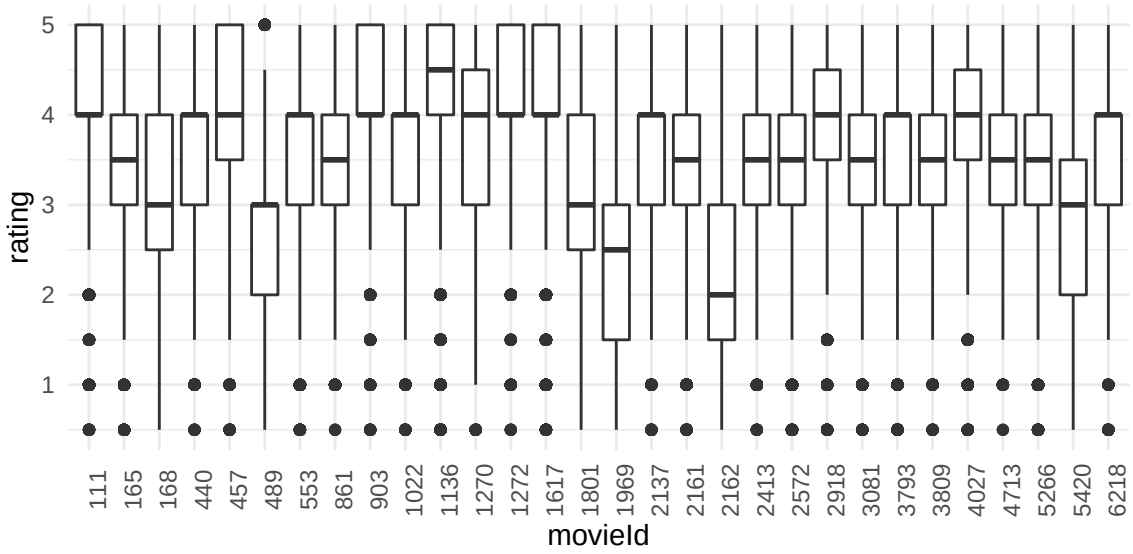
```
edx %>% group_by(movieId) %>% filter(n()>150) %>% summarize(title,rating=mean(rating)) %>%
  unique() %>% arrange(desc(rating)) %>% head(10)
```

```
## # A tibble: 10 x 3
## # Groups:   movieId [10]
##    movieId title                                          rating
##      <dbl> <chr>                                           <dbl>
##  1     318 Shawshank Redemption, The (1994)                 4.46
##  2     858 Godfather, The (1972)                            4.42
##  3      50 Usual Suspects, The (1995)                       4.37
##  4     527 Schindler's List (1993)                          4.36
##  5     912 Casablanca (1942)                                4.32
##  6     904 Rear Window (1954)                               4.32
##  7     922 Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)    4.32
##  8    1212 Third Man, The (1949)                            4.31
##  9    3435 Double Indemnity (1944)                          4.31
## 10    1178 Paths of Glory (1957)                            4.31
```

There is clearly a bias towards classics from the 1990ies as well as from the 1940ies-1950ies.

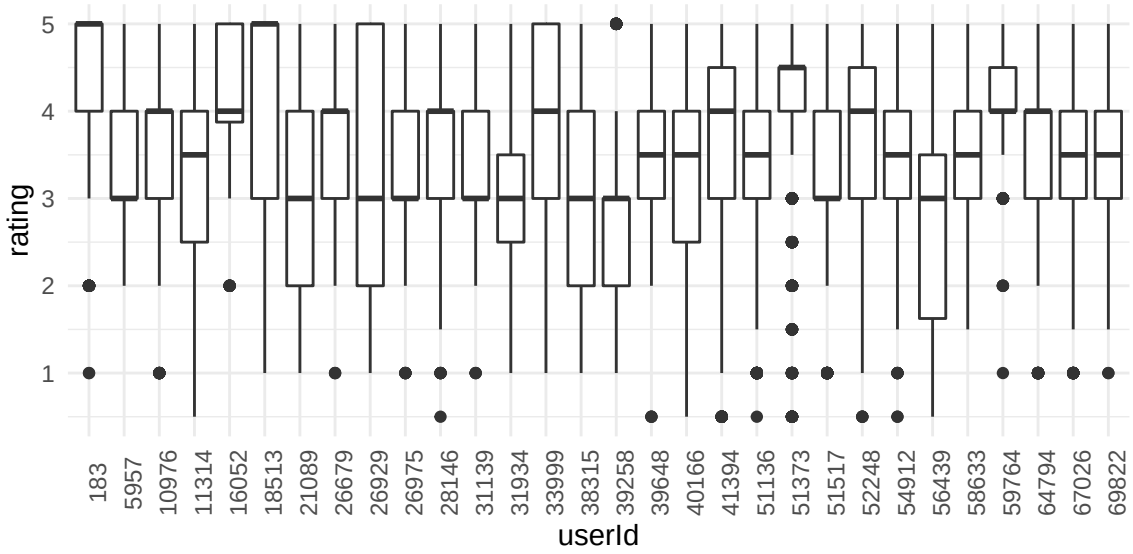
Below is the code and the resulting plot for a boxplot of the ratings for a randomly chosen set of 30 movies from the group of movies with more than 150 ratings. The plot clearly demonstrates that individual tastes aside there are more and less preferred movies, suggesting that such a preference might be useful in model building (see below).

```
edx %>%  group_by(movieId) %>% filter(n()>150) %>% ungroup() %>%
  filter(movieId %in% sample(movieId, 30, replace=FALSE)) %>%
  ggplot(aes(as.factor(movieId), rating)) + geom_boxplot() + theme_minimal() +
  theme(axis.text.x=element_text(angle=90)) + xlab("movieId")
```



Users Similarly to what has been described above, when looking at a boxplot of the ratings of 30 randomly chosen users it becomes clear that there are differences in the user behaviour. Whereas there are users that appear to love each and every movie they see, there are other users that turn out to rate movies in a much stricter fashion. Again this finding can be made use of later when trying to build a prediction model.

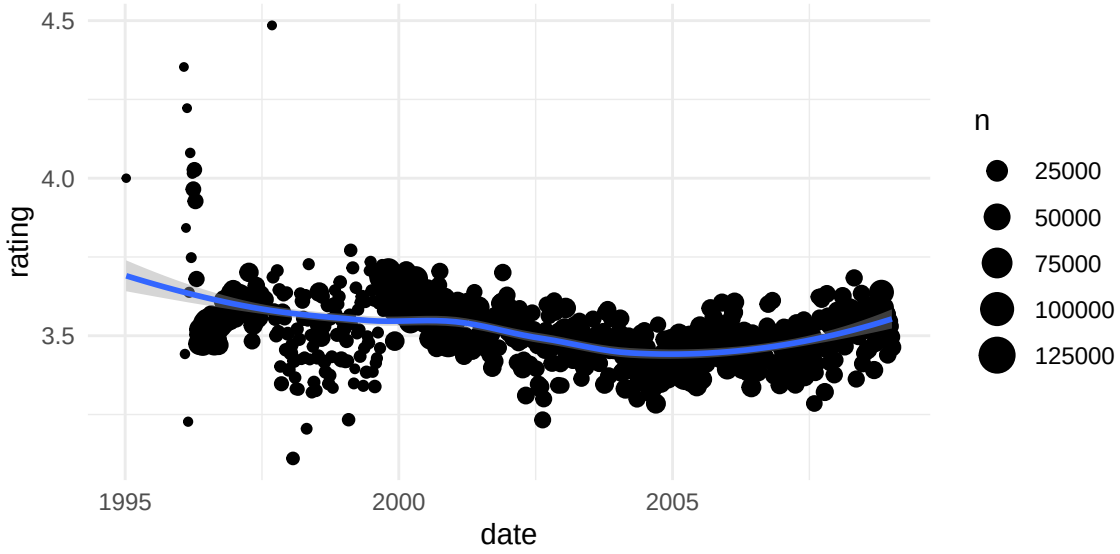
```
edx %>%  group_by(userId) %>% filter(n()>150) %>% ungroup() %>%
  filter(userId %in% sample(userId, 30, replace=FALSE)) %>%
  ggplot(aes(as.factor(userId), rating)) + geom_boxplot() + theme_minimal() +
  theme(axis.text.x=element_text(angle=90)) + xlab("userId")
```



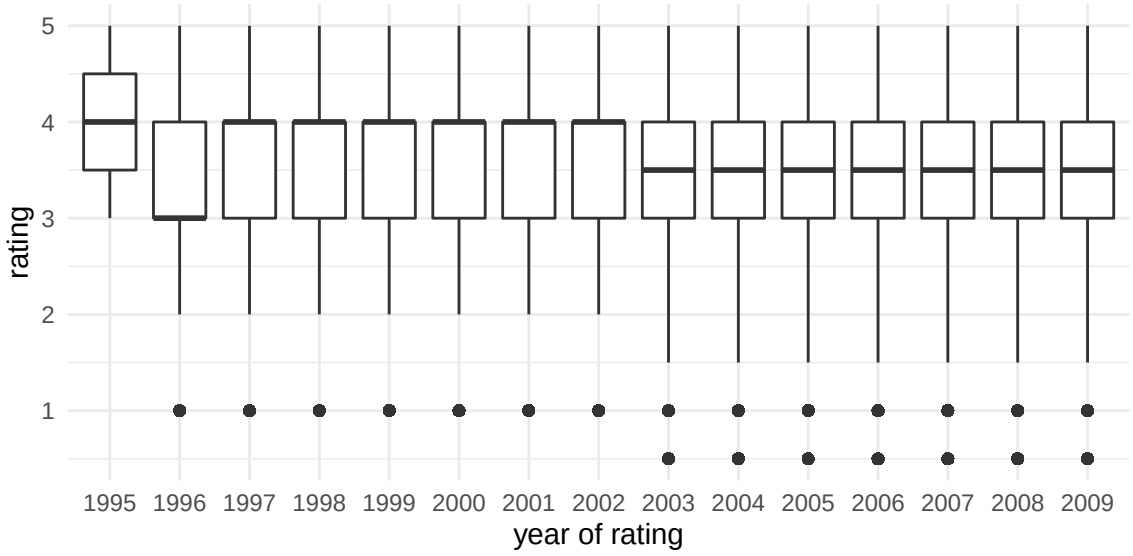
Time For the analysis of time effects, a dataframe is created using the code below, where the timestamps are converted to time and date and where the years of appearance are extracted from the title column.

```
edx_time <- edx %>% mutate(date = round_date(as_datetime(timestamp), unit="week")) %>%
  mutate(date_1= as.numeric(year(as_datetime(timestamp))),
         appearance=str_extract(title, "\\([0-9]{4}\\)")) %>%
  mutate(appearance=as.numeric(str_extract(appearance, "[0-9]+"))) %>%
  mutate(lapse=date_1-appearance)
```

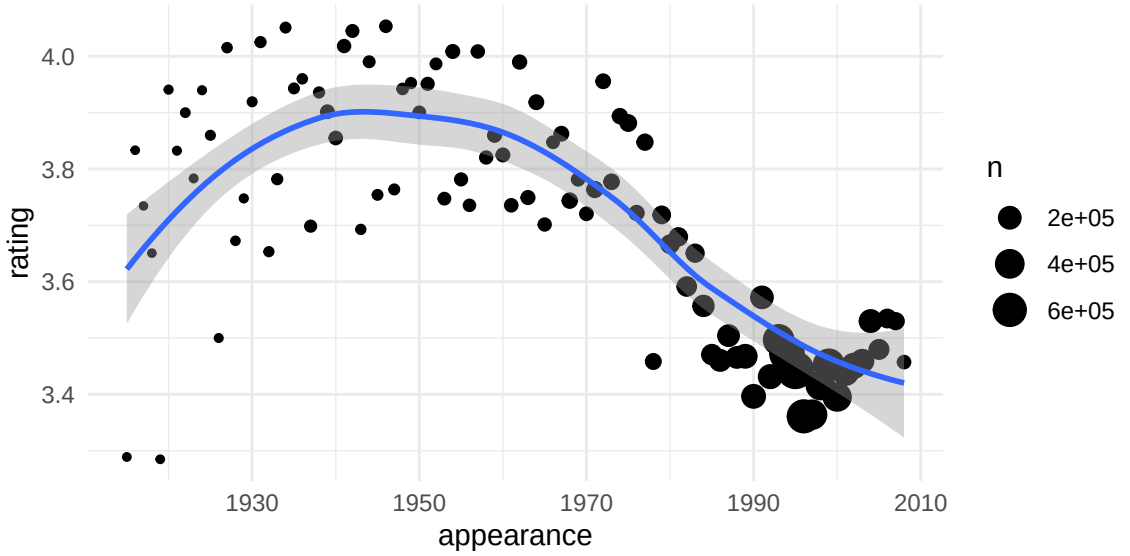
Plotting the rating of movies versus the date of the rating (timestamp) shows a tendency, that the later a movie was rated the lower the rating is likely to be. This trend might be fitted when creating a model.



However, when looking at corresponding boxplots and the respective quartile distances such an effect might turn out to be rather small.



Likewise, plotting the rating versus the year of appearance suggests, that movies that appeared before the year ~1980 should be treated separately from newer movies, as the yearly average rating appears to change significantly. This might be due to the fact that bad movies from before 1980 are not watched any more and are thus not rated today. This effect might also be used to improve a model.



Genre The edx data set contains a column "genres". In this column each movie is assigned to one or a combination of genres, that describe the movie in question. The code below takes the edx data set and expands it to a new data set called edx_genres, where each row is a unique combination of user, movie, rating and individual genre. In order to reduce computation time and in order to avoid memory overflow this analysis is only conducted on users that have rated at least 150 movies. This is justified, as any user-genre preferences used later for optimizing a prediction algorithm will be most impactful for users that have rated many movies.

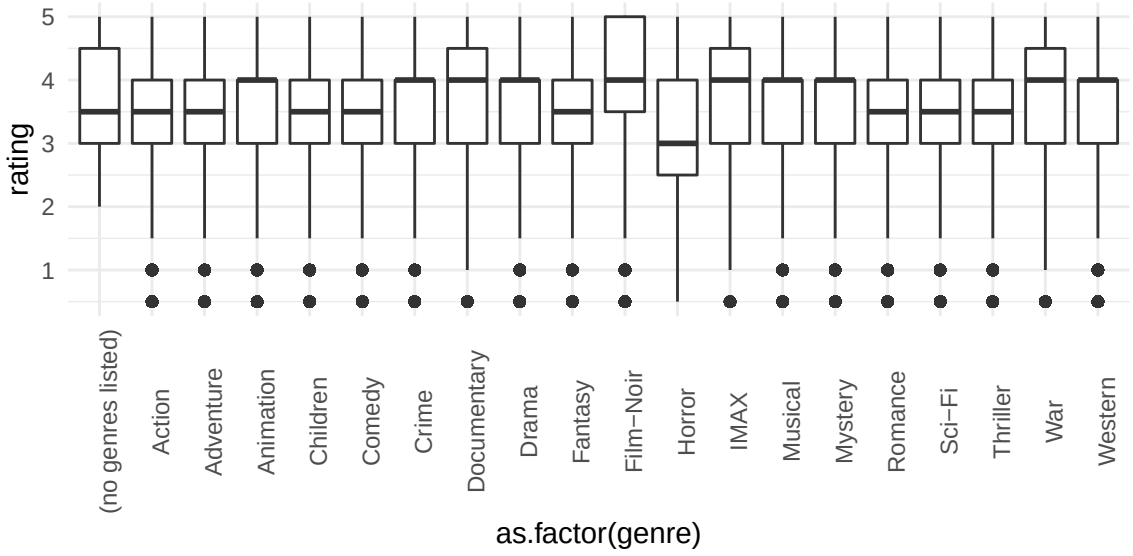
Analysis of edx_genres shows, that there are 20 individual genres, that are combined in order to describe a movie. Of these 20 genres most movies contain an element of Drama and Comedy, with Action, Thriller, Adventure and Romance also being common genres.

```
edx_genres %>% group_by(genre) %>% summarize(n=n()) %>% arrange(desc(n))
```

```
## # A tibble: 20 x 2
##    genre                    n
##    <chr>                <int>
##  1 Drama              2621472
##  2 Comedy             2383823
##  3 Action             1641830
##  4 Thriller           1506736
##  5 Adventure          1210668
##  6 Romance            1109708
##  7 Sci-Fi              868185
##  8 Crime               865598
##  9 Fantasy             619275
## 10 Horror              499876
## 11 Children            475299
## 12 Mystery             390734
## 13 War                 322685
## 14 Animation           296933
## 15 Musical             286753
## 16 Western             123087
## 17 Film-Noir            82853
```

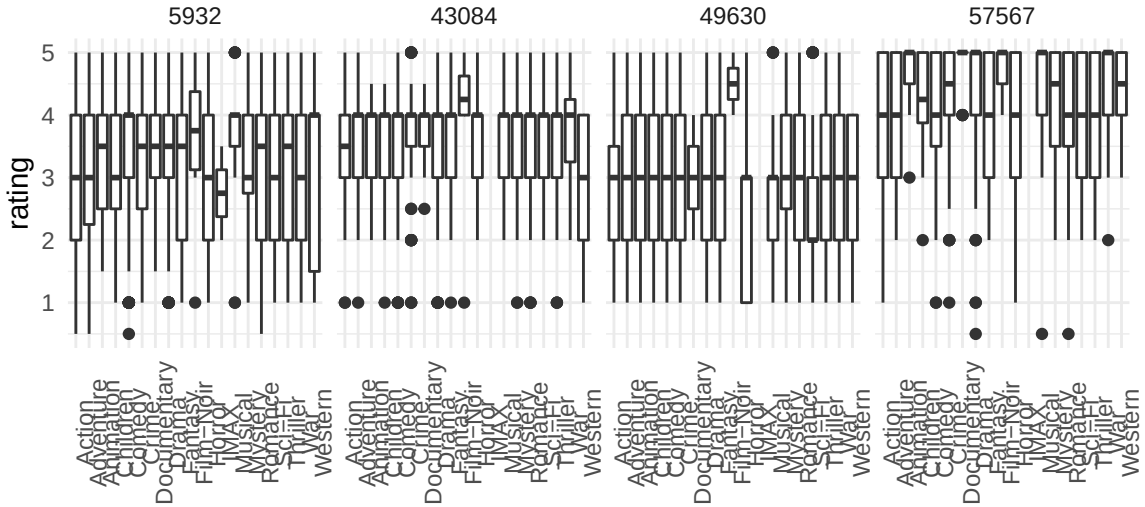
```
## 18 Documentary          71216
## 19 IMAX                  5797
## 20 (no genres listed)       7
```

A boxplot of the average movie ratings given any individual genre component reveals that there is only a limited bias in favour of or against individual genres across the whole edx_genre dataset. This reflects well the variability of taste among people.



However, when looking at 4 randomly chosen individual users it becomes imedeately clear, that some people do have preferences for certain genres. Thus such a user-genre preference could be made use of, when building a prediction model later.

```
edx_genres %>% filter(userId %in% sample(edx_genres$userId,4, replace=FALSE)) %>%
  ggplot(aes(as.factor(genre), rating)) + geom_boxplot() + facet_grid(. ~ userId) +
    theme_minimal() +
    theme(axis.text.x =element_text(angle = 90)) + xlab(" ")
```



Model Building

The loss function

For any machine learning problem one of the most important decisions is what to optimize. Here the goal is to construct an algorithm that predicts the ratings for user-movie-combinations. Therefore, a loss function needs to be defined, which gives a measure of how good a prediction is. Here, we use the root-mean-square-error (RMSE) as a loss function. The RMSE is defined as

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

, where N is the overall number of predictions, $\hat{y}_i$ is the predicted value of rating i and y_i is the actual value of rating i.

In R a function is defined for the calculation of RMSEs:

```
RMSE <- function(test,pred){sqrt(mean((test-pred)^2))}
```

Train and test set

In order to avoid overtraining, the edx data set, which is supposed to be used for model building, is again split into a train and a test set. Throughout the model building process this train and test sets are used for construction of the prediction algorithm. Only in the very last step (see results section), the finally obtained model is retrained on the full edx data set and ultimately assessed on its ability to predict the ratings within the validation dataset. The code below creates a test and a train dataset:

```
ind<-createDataPartition(edx$rating, times=1, p=0.2, list=FALSE)
train <- edx[-ind,]
test <- edx[ind,] %>% semi_join(train, by="movieId") %>% semi_join(train, by="userId")
```

Random ratings

The simplest possible prediction model is simply guessing the ratings. While this is of course not a useful approach, it demonstrates how later models perform versus pure chance. Below are the corresponding code and the resulting RMSE.

```
pred <- sample(seq(0.5,5,0.5), nrow(test), replace=TRUE)

rmse <- RMSE(test$rating, pred)
```

```
results <- data.frame(what="random rating", rmse=rmse)
results
```

```
##            what    rmse
## 1 random rating 1.94111
```

Mean

As a first step attempting to improve on pure random ratings, we can assume, that the mean over all predictions is a better call. This means we create a model, where it is assumed, that each rating consists of the mean over all ratings and a deviation (or error). The model looks thus as in the equation below:

$$\hat{Y}_{u,i} = \mu + \epsilon_{u,i}$$

, where $Y_{u,i}$ is the rating of user u for movie i, μ is the mean over all ratings by all users and $\epsilon_{u,i}$ is a user and movie specific deviation from this mean.

Applying simply the mean of all movies as a predictor yields the following result:

```
mu <- mean(test$rating) #rating mean

rmse <- RMSE(test$rating, mu)

results<- bind_rows(results, data.frame(what="mu", rmse=rmse))
results
```

```
##            what     rmse
## 1 random rating 1.941110
## 2            mu 1.060702
```

We can see that this approach improves significantly on random ratings.

Movie effects and user effects

As suggested above, the parameter $\epsilon_{u,i}$ contains all the user and movie specifics that go into the rating of user u for movie i. Thus it should be possible to improve on simply the mean, when the model is first extended with a parameter taking into account the movie effect or movie bias b_i (i.e. how popular a given movie is on average). The resulting new model is given below:

$$\hat{Y}_{u,i} = \mu + b_i + \epsilon_{u,i}$$

Similarly, the remaining error $\epsilon_{u,i}$ contains the user effect or user bias b_u (i.e. how strict is a user generally in his or her ratings). This gives us the model below:

$$\hat{Y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

In R the above is achieved with the following code:

```
## Movie effect Y_hat = mu + bi + e ##################################

movie_effects <- train %>% group_by(movieId) %>% summarize(bi = mean(rating - mu))

pred <- test %>% inner_join(movie_effects, by="movieId") %>% mutate(pred=mu+bi) %>%
  .$pred

rmse <- RMSE(test$rating, pred)

results<- bind_rows(results, data.frame(what="mu+bi", rmse=rmse))

## Adding user effect  Y_hat = mu + bi + bu + e ###################################

user_effects <- train %>% left_join(movie_effects, by="movieId") %>%
  group_by(userId) %>% summarize(bu=mean(rating-mu-bi))

pred <- test %>% left_join(movie_effects, by="movieId") %>%
```

```
  left_join(user_effects, by="userId") %>% mutate(pred=mu+bu+bi) %>% .$pred

rmse <- RMSE(test$rating,pred)

results <- bind_rows(results, data.frame(what="mu+bi+bu", rmse=rmse))
results
```

```
##            what      rmse
## 1 random rating 1.9411095
## 2            mu 1.0607020
## 3         mu+bi 0.9441287
## 4      mu+bi+bu 0.8661754
```

Both, including a movie effect and including a user effect greatly improve the overall accuracy of the prediction system.

Regularization

In the above model, the overall means within the movie and user biases are somewhat overproportionally driven by movies with only few ratings as well as users rating only very few movies. One approach to avoid this is regularization. This is achieved by, when calculating b_i and b_u, not dividing by the number of ratings N but by the sum of N and a parameter λ. When N is large, this has only a minimal effect. For movies with few ratings and users rating only few movies, however, this decreases the overall effect.

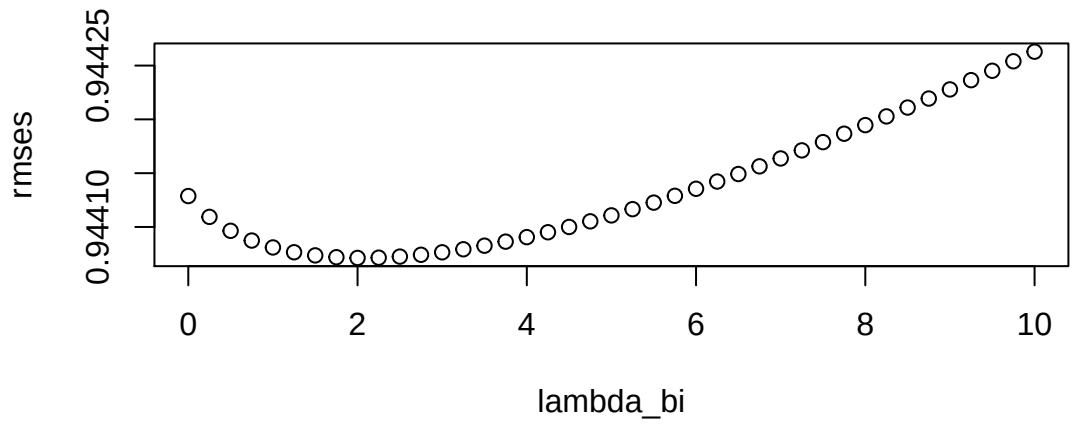
Using regularization, the movie and user biases are calculated as shown below:

$$b_i = \frac{\sum_{u=1}^{n_i}(Y_{u,i} - \hat{\mu})}{\lambda_i + n_i}$$

$$b_u = \frac{\sum_{u=1}^{n_u}(Y_{u,i} - \hat{\mu} - b_i)}{\lambda_u + n_u}$$

The best value for the regularization parameter λ is not *a priori* known. Therefore, a function is used, which tests values for λ between 0 and 10. Then the λ resulting in the lowest RMSE is chosen. For the movie bias b_i this is achieved by the code below:

```
lambda_bi<-seq(0,10,0.25)

rmses<- sapply(lambda_bi, function(lam){
  reg_movie_effects <- train %>% group_by(movieId) %>%
    summarize(bi = sum(rating - mu)/(n()+lam))
  pred <- test %>% inner_join(reg_movie_effects, by="movieId") %>%
    mutate(pred=mu+bi) %>% .$pred
  return(RMSE(test$rating, pred))})

plot(lambda_bi,rmses)
```




```
lambda_bi<- lambda_bi[which.min(rmses)]
```

Consequently the regularized movie effects can be obtained using the optimal λ_i of 1.75.

```
reg_movie_effects <- train %>% group_by(movieId) %>%
  summarize(bi_reg = sum(rating-mu)/(n()+lambda_bi))
pred <- test %>% inner_join(reg_movie_effects, by="movieId") %>%
  mutate(pred=mu+bi_reg) %>% .$pred
rmse <- RMSE(test$rating,pred)
results<- bind_rows(results, data.frame(what="mu+bi_reg", rmse=rmse))
results
```

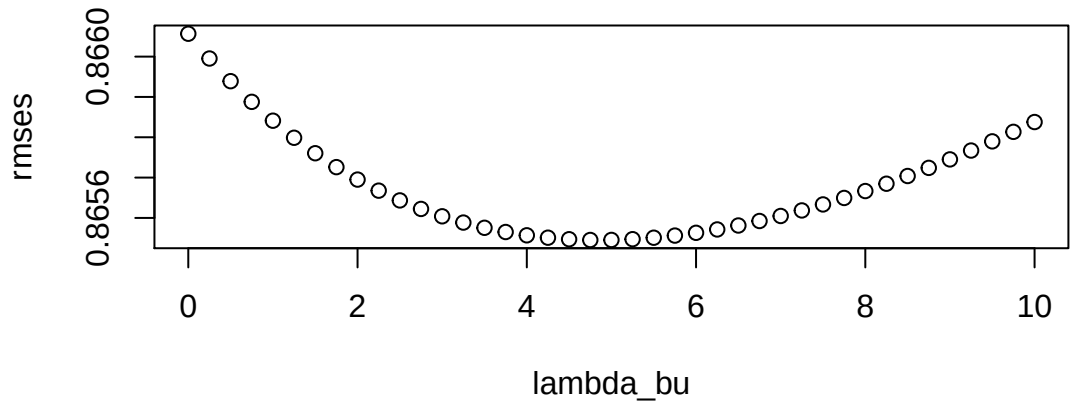
```
##            what      rmse
## 1 random rating 1.9411095
## 2            mu 1.0607020
## 3         mu+bi 0.9441287
## 4      mu+bi+bu 0.8661754
## 5     mu+bi_reg 0.9440712
```

As can be seen regularization of the movie effect does lead to an overall improvement of the corresponding RMSE, however this effect is rather small.

The same procedure can also be applied to the user effect described previously. Again, values between 0 and 10 are examined and the λ resulting in the lowest RMSE is selected.

```
lambda_bu<-seq(0,10,0.25)

rmses<- sapply(lambda_bu, function(lam){
  reg_user_effects <- train %>% left_join(reg_movie_effects, by="movieId") %>%
    group_by(userId) %>% summarize(bu_reg = sum(rating - mu - bi_reg)/(n()+lam))
  pred <- test %>% left_join(reg_user_effects, by="userId") %>%
    left_join(reg_movie_effects, by="movieId") %>% mutate(pred=mu+bi_reg+bu_reg) %>%
    .$pred
  return(RMSE(test$rating, pred))})
```

```
plot(lambda_bu,rmses)
```




```
lambda_bu<- lambda_bu[which.min(rmses)]
```

Then the optimized λ_u of 5.0 is applied to the user effects and the resulting model containing regularized movie and user effects is tested on the test set.

```
reg_user_effects <- train %>% left_join(reg_movie_effects, by="movieId") %>%
  group_by(userId) %>% summarize(bu_reg = sum(rating - mu - bi_reg)/(n()+lambda_bu))
pred <- test %>% left_join(reg_user_effects, by="userId") %>%
  left_join(reg_movie_effects, by="movieId") %>% mutate(pred=mu+bi_reg+bu_reg) %>%
  .$pred

rmse <- RMSE(test$rating,pred)
results<- bind_rows(results, data.frame(what="mu+bi_reg+bu_reg", rmse=rmse))
results
```

```
##               what      rmse
## 1    random rating 1.9411095
## 2               mu 1.0607020
## 3           mu+bi 0.9441287
## 4         mu+bi+bu 0.8661754
## 5        mu+bi_reg 0.9440712
## 6 mu+bi_reg+bu_reg 0.8655453
```

Again, a small improvement on the RMSE of the model is obtained.

Time effects

The analysis above regarding the dependence of the ratings on the timestamp suggests, that the model might be improved using a linear model of rating versus timestamp in weeks.

$$\hat{Y}_{u,i} = \mu + b_i + b_u + f(rd) + \epsilon_{u,i}$$

, where rd is the rating date.

The code below includes this time dependence:

```
train <- mutate(train, date = round_date(as_datetime(timestamp), unit="week"))
test <- mutate(test, date = round_date(as_datetime(timestamp), unit="week"))

train_1<- train %>% left_join(reg_movie_effects, by="movieId") %>%
  left_join(reg_user_effects, by="userId") %>% mutate(diff=rating-mu-bi_reg-bu_reg)

fit_date <- lm(diff~date, data=train_1)

pred<-test %>% left_join(reg_user_effects, by="userId") %>%
  left_join(reg_movie_effects, by="movieId") %>%
  mutate(date=round_date(as_datetime(timestamp), unit="week")) %>%
  mutate(time_dep=predict(fit_date, test)) %>%
  mutate(pred=mu+bi_reg+bu_reg+time_dep) %>% .$pred

rmse <- RMSE(test$rating,pred)
results<-bind_rows(results,data.frame(what="mu+bi_reg+bu_reg+time_dep", rmse=rmse))
results
```

```
##                          what      rmse
## 1              random rating 1.9411095
## 2                          mu 1.0607020
## 3                      mu+bi 0.9441287
## 4                   mu+bi+bu 0.8661754
## 5                  mu+bi_reg 0.9440712
## 6           mu+bi_reg+bu_reg 0.8655453
## 7 mu+bi_reg+bu_reg+time_dep 0.8655335
```

Again a slight improvement of the overall RMSE can be obtained.

Analysis of the year of appearance showed that there appears to be a difference in average ratings in movies that appeared before or after ~ 1980. Interestingly, movies that appeared before 1980 ("classics") are rated more favorably. This led to the suggestion that a model could be improved simply by adding a term giving the mean over all movies that appeared before 1980 and after respectively.

The resulting model is shown below:

$$\hat{Y}_{u,i} = \mu + b_i + b_u + b_{1980} + \epsilon_{u,i}$$

This is achieved by the code below:

```
train_1980 <- train %>% left_join(reg_movie_effects, by="movieId") %>%
  left_join(reg_user_effects, by="userId") %>% mutate(diff=rating-mu-bi_reg-bu_reg) %>%
  mutate(date_1= as.numeric(year(as_datetime(timestamp))),
         appearance=str_extract(title, "\\([0-9]{4}\\)")) %>%
  mutate(appearance=as.numeric(str_extract(appearance, "[0-9]+")))

effect_1980 <- train_1980 %>% mutate(is1980=ifelse(appearance>1980,0,1)) %>%
  group_by(is1980) %>% summarize(effect_1980=mean(diff))
```

```
pred <- test %>% left_join(reg_movie_effects, by="movieId") %>%
  left_join(reg_user_effects, by="userId") %>%
  mutate(date_1= as.numeric(year(as_datetime(timestamp))),
         appearance=str_extract(title, "\\([0-9]{4}\\)")) %>%
  mutate(appearance=as.numeric(str_extract(appearance, "[0-9]+"))) %>%
  mutate(is1980=ifelse(appearance>1980,0,1)) %>% left_join(effect_1980, by="is1980") %>%
  mutate(pred=mu+bi_reg+bu_reg+effect_1980) %>% .$pred

rmse <- RMSE(test$rating,pred)
results<-bind_rows(results,data.frame(what="mu+bi_reg+bu_reg+effect_1980", rmse=rmse))
results
```

```
##                            what      rmse
## 1                random rating 1.9411095
## 2                            mu 1.0607020
## 3                        mu+bi 0.9441287
## 4                     mu+bi+bu 0.8661754
## 5                    mu+bi_reg 0.9440712
## 6              mu+bi_reg+bu_reg 0.8655453
## 7    mu+bi_reg+bu_reg+time_dep 0.8655335
## 8 mu+bi_reg+bu_reg+effect_1980 0.8655198
```

As can be seen an even better improvement is obtained with a factor as simple as deciding if a movie appeared before or after the year 1980.

Genre dependence

Probably the most important predictor for movie ratings is the individual preference of a user. As discussed above, each movie is assigned to a combination of genres out of a list of 20. Based on the assumption that every user has a certain preference for each of the 20 genres and that his or her movie preference can be approximated by averaging over the individual genre preferences, the model can be expanded by the addition of a "user specific term for movie-genre preference" $b_{u,g,i}$:

$$\hat{Y}_{u,i} = \mu + b_i + b_u + b_{1980} + b_{u,g,i} + \epsilon_{u,i}$$

In order to achieve this, the regularized movie and user effects as well as the 1980-effect are added to the data and the corresponding differential to the actual rating is determined.

```
train_1 <- train_1980 %>% mutate(is1980=ifelse(appearance>1980,0,1)) %>%
  left_join(effect_1980, by="is1980") %>%
  mutate(diff=rating-mu-bi_reg-bu_reg-effect_1980) %>%
  select(userId, movieId, rating, timestamp, title, genres, bi_reg, bu_reg, effect_1980, diff)
```

Then in order to avoid memory overflow, the analysis is restricted to users that have at least rated 150 movies. The data set is expanded to obtain individual genre entries (see data analysis part above) and the specific average user-genre ratings are calculated.

```
train_subset <- train_1 %>% group_by(userId) %>%  filter(n()>=150) %>% ungroup()

train_expanded <- train_subset %>%
  separate(genres, into=c("a","b","c","d","e","f","g","h"), sep="\\|", fill="right") %>%
```

```
  gather(key="bla", value="genre", 6:13, na.rm=TRUE) %>% select(-bla)

user_genre_rating<-train_expanded %>% group_by(userId, genre) %>%
  summarize(avg_diff=mean(diff))
```

The manipulations described above are then performed on the test set.

```
test_expanded <- test %>% left_join(reg_user_effects, by="userId") %>%
  left_join(reg_movie_effects, by="movieId") %>%
  mutate(appearance=str_extract(title, "\\([0-9]{4}\\)")) %>%
  mutate(appearance=as.numeric(str_extract(appearance, "[0-9]+"))) %>%
  mutate(is1980=ifelse(appearance>1980,0,1)) %>% left_join(effect_1980, by="is1980") %>%
  select(-is1980) %>%
  separate(genres, into=c("a","b","c","d","e","f","g","h"), sep="\\|", fill="right") %>%
  gather(key="bla", value="genre", 6:13, na.rm=TRUE) %>% select(-bla)
```

Finally, the average user-genre ratings are combined to match the individual movies and the corresponding values for the user genre dependence $b_{u,g,i}$ are calculated. Then, the RMSE using this model is calculated.

```
pred <-test_expanded %>% left_join(user_genre_rating, by=c("userId","genre"))  %>%
  mutate_if(is.numeric, funs(ifelse(is.na(.), 0, .))) %>% group_by(userId, movieId) %>%
  mutate(ugr=mean(avg_diff)) %>% summarize(bu_reg,bi_reg,effect_1980,ugr) %>% unique()  %>%
  mutate(pred=mu+bi_reg+bu_reg+effect_1980+ugr) %>%
  .$pred

rmse1 <- RMSE(test$rating,pred)
results<-bind_rows(results,data.frame(
          what="mu+bi_reg+bu_reg+effect_1980+user_genre_dependance", rmse=rmse1))
results
```

```
##                                                 what      rmse
## 1                                      random rating 1.9411095
## 2                                                 mu 1.0607020
## 3                                             mu+bi 0.9441287
## 4                                          mu+bi+bu 0.8661754
## 5                                         mu+bi_reg 0.9440712
## 6                                    mu+bi_reg+bu_reg 0.8655453
## 7                          mu+bi_reg+bu_reg+time_dep 0.8655335
## 8                       mu+bi_reg+bu_reg+effect_1980 0.8655198
## 9 mu+bi_reg+bu_reg+effect_1980+user_genre_dependance 0.8556056
```

As can be seen, inclusion of the individual genre preferences leads to a quite dramatic improvement of the model.

Matrix Factorization

A more powerful technique to take into account individual preferences is matrix factorization, where the whole rating matrix (in this case user vs movie) is approximated by the dot product of two matrices of lower dimensions (here a user matrix and a movie matrix). In R for example the recosystem package allows to do this (Qiu 2021).

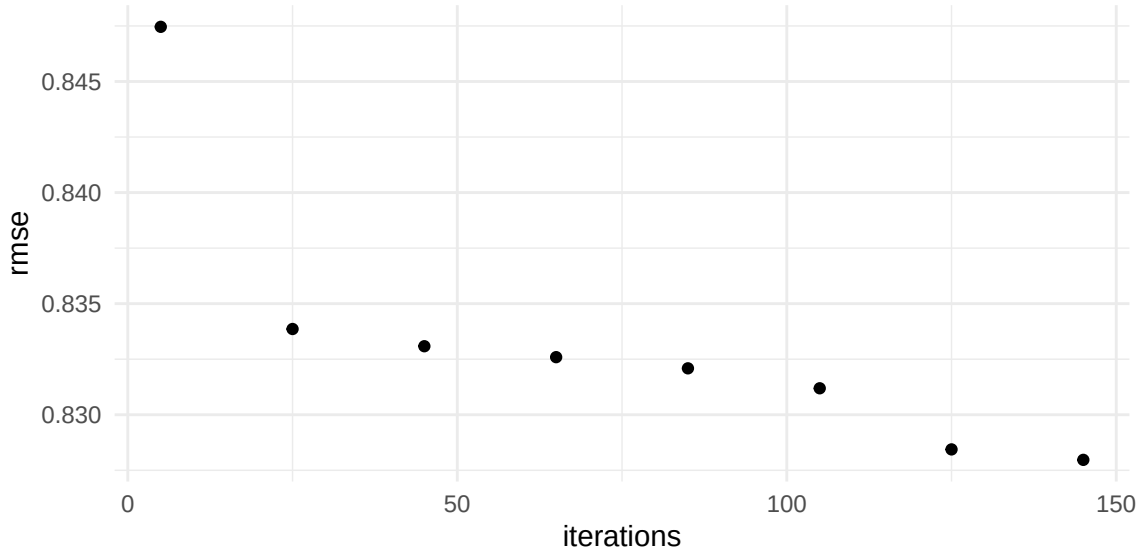
When using the recosystem package, the first step is to define a rating matrix from the train and the test data.

```
set.seed(130484)
train_2 <- data_memory(train$userId, train$movieId, rating = train$rating)
test_2 <- data_memory(test$userId, test$movieId)
```

In order to get a feeling for this package and the optimizable options several parameters are examined for their sensitivity. First, when a model is trained in the recosystem package, this is done with iterations. In order to examine, if more iterations always lead to a better result the following function is run over 20 different amounts of iterations:

```
iter <- seq(5,145,20)

doiter<-function(n){
  r<-Reco()
  r$train(train_2, opt=list(niter=c(n)))
  pred<- r$predict(test_2,out_memory())
  rmse <- RMSE(test$rating,pred)
  rmse
}

iterationtable <- data.frame(iterations=iter, rmse=sapply(iter,doiter))
```

```
iterationtable %>% ggplot(aes(iterations, rmse)) + geom_point()+theme_minimal()
```



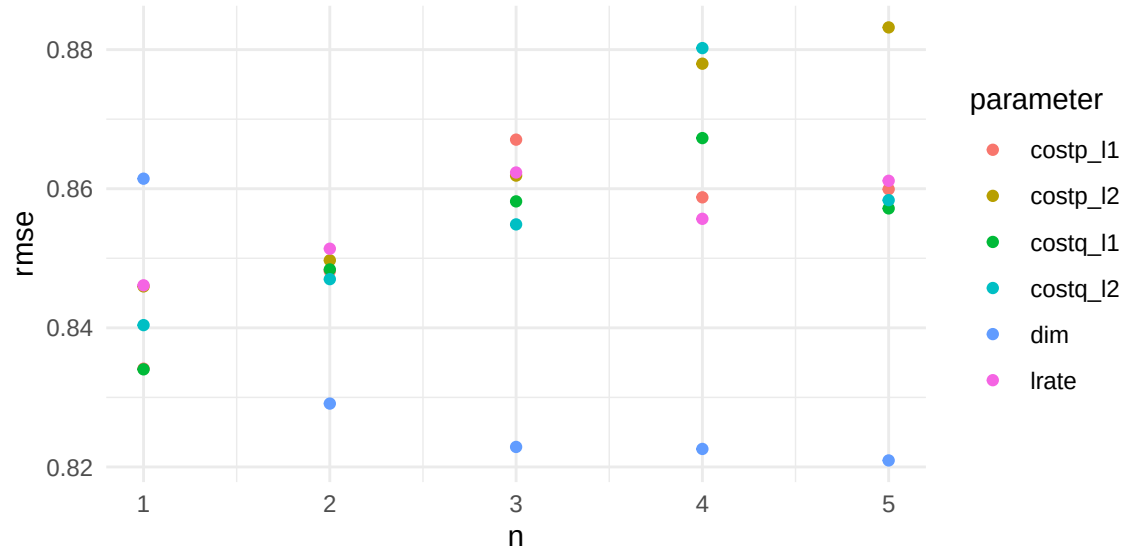
As can be seen from the plot, generally, the more iterations are used, the better the final result. However, increasing the number of iterations, of course, leads to significantly longer computation time. The plot shows that in the beginning increasing the amount of iterations leads rapidly to a better result. The magnitude of the effect then decreases, reaching a plateau around between ~50 and ~100 iterations. Then towards signifiantly higher iteration numbers (> 100) the improvement per added iteration increases again. However, for the sake of computation time, a iteration number of 75 is chosen.

Similarly, the individual tunable parameters are analyzed in order to find out which ones of them show a high sensitivity towards the final result. Every parameter is looked at individually and 5 values between the lowest and 150% of the highest default value are examined.

```
m<-5 # how many data points to generate

dim <- seq(5,30,length.out=m)
dimfunc <- function(n){
  r<-Reco()
  r$train(train_2,opt=list(dim=c(n)))
  pred <- r$predict(test_2,out_memory())

  rmse <- RMSE(test$rating,pred)

  rmse
}

costp_l1 <- seq(0,0.15,length.out=m)
costp_l1func <- function(n){
  r<-Reco()
  r$train(train_2,opt=list(costp_l1=c(n)))
  pred <- r$predict(test_2,out_memory())

  rmse <- RMSE(test$rating,pred)

  rmse
}

costp_l2 <- seq(0.01,0.15,length.out=m)
costp_l2func <- function(n){
  r<-Reco()
  r$train(train_2,opt=list(costp_l1=c(n)))
  pred <- r$predict(test_2,out_memory())

  rmse <- RMSE(test$rating,pred)

  rmse
}

costq_l1 <- seq(0,0.15,length.out=m)
costq_l1func <- function(n){
  r<-Reco()
  r$train(train_2,opt=list(costp_l1=c(n)))
  pred <- r$predict(test_2,out_memory())

  rmse <- RMSE(test$rating,pred)

  rmse
}

costq_l2 <- seq(0.01,0.15,length.out=m)
costq_l2func <- function(n){
  r<-Reco()
  r$train(train_2,opt=list(costp_l1=c(n)))
  pred <- r$predict(test_2,out_memory())

  rmse <- RMSE(test$rating,pred)
```

```
  rmse
}

lrate <- seq(0.01,0.15, length.out=m)
lratefunc <- function(n){
  r<-Reco()
  r$train(train_2,opt=list(costp_l1=c(n)))
  pred <- r$predict(test_2,out_memory())

  rmse <- RMSE(test$rating,pred)

  rmse
}

sensitivity <- data.frame(n=seq(1,m), dim=sapply(dim, dimfunc),
                          costp_l1=sapply(costp_l1, costp_l1func),
                          costp_l2=sapply(costp_l2, costp_l2func),
                          costq_l1=sapply(costq_l1, costq_l1func),
                          costq_l2=sapply(costq_l2, costq_l2func),
                          lrate=sapply(lrate,lratefunc))
```

```
sensitivity %>% gather(key="parameter", value="rmse",2:7) %>%
  ggplot(aes(n,rmse, col=parameter)) + geom_point() +theme_minimal()
```



Interestingly, only the dim parameter turns out to lead to an improvement in RMSE within the values examined above. As a consequence, for further model building activities only this parameter is tuned.

With these findings, a matrix factorization approach is conducted on the original train data.

```
set.seed(130484)

train_2 <- data_memory(train_1$userId, train_1$movieId, rating = train_1$rating)
test_2 <- data_memory(test$userId, test$movieId)
```

```
r<-Reco()

params <- r$tune(train_2, opt=list(dim=seq(5,25,length.out=4), costp_l1=c(0),
                                   costp_l2=c(0.01), costq_l1=c(0), costq_l2=c(0.01),
                                   lrate=c(0.01)))

r$train(train_2, opt=list(params$min, niter=c(75)))

pred<- r$predict(test_2,out_memory())
```

```
rmse <- RMSE(test$rating,pred)
results<- bind_rows(results, data.frame(what="matrix_factorization_only", rmse=rmse))
results
```

```
##                                                 what      rmse
## 1                                     random rating 1.9411095
## 2                                                mu 1.0607020
## 3                                            mu+bi 0.9441287
## 4                                         mu+bi+bu 0.8661754
## 5                                        mu+bi_reg 0.9440712
## 6                                  mu+bi_reg+bu_reg 0.8655453
## 7                      mu+bi_reg+bu_reg+time_dep 0.8655335
## 8                   mu+bi_reg+bu_reg+effect_1980 0.8655198
## 9  mu+bi_reg+bu_reg+effect_1980+user_genre_dependance 0.8556056
## 10                     matrix_factorization_only 0.8323970
```

It is evident that matrix factorization is a very powerful technique. An improvement of RMSE of more than 0.02 is obtained with matrix factorization alone compared to the best previously used model (including the user genre dependence).

After matrix factorization using the recosystem package led to a significant improvement of RMSE compared to the model constructed before, it is interesting to examine if combining the two models might lead to yet another improvement. Thus, matrix factorization using the same parameters as above was performed not on the original data, but on the differentials of the model constructed before (but instead of the user genre parameter). As within the model, these differentials are supposed to contain the individual movie preferences, performing matrix factorization on these might be particularly valuable. The data is therefore fitted to the model below.

$$\hat{Y}_{u,i} = \mu + b_i + b_u + b_{1980} + b_{\text{recosystem}} + \epsilon_{u,i}$$

This is achieved using the code below.

```
set.seed(130484)

train_2<- data_memory(train_1$userId, train_1$movieId, rating = train_1$diff)
test_2 <- data_memory(test$userId, test$movieId)

r<-Reco()

params <- r$tune(train_2, opt=list(dim=seq(5,25,length.out=4), costp_l1=c(0),
                                   costp_l2=c(0.01), costq_l1=c(0),
                                   costq_l2=c(0.01), lrate=c(0.01)))
```

```
r$train(train_2, opt=list(params$min, niter=c(75)))

pred<- r$predict(test_2,out_memory())

pred<- test %>% left_join(reg_user_effects, by="userId") %>%
  left_join(reg_movie_effects, by="movieId") %>%
  mutate(appearance=str_extract(title, "\\([0-9]{4}\\)")) %>%
  mutate(appearance=as.numeric(str_extract(appearance, "[0-9]+"))) %>%
  mutate(is1980=ifelse(appearance>1980,0,1))%>%
  left_join(effect_1980, by="is1980") %>%
  mutate(pred_diff=pred) %>% mutate(pred=mu+bi_reg+bu_reg+effect_1980+pred_diff) %>%
  .$pred
```

```
rmse_1 <- RMSE(test$rating,pred)
results<- bind_rows(results, data.frame(what="mu+bi_reg+bu_reg+effect_1980+matFact(diffs)",
                                        rmse=rmse_1))
results
```

```
##                                                 what      rmse
## 1                                     random rating 1.9411095
## 2                                                mu 1.0607020
## 3                                             mu+bi 0.9441287
## 4                                          mu+bi+bu 0.8661754
## 5                                        mu+bi_reg 0.9440712
## 6                                   mu+bi_reg+bu_reg 0.8655453
## 7                        mu+bi_reg+bu_reg+time_dep 0.8655335
## 8                  mu+bi_reg+bu_reg+effect_1980 0.8655198
## 9  mu+bi_reg+bu_reg+effect_1980+user_genre_dependance 0.8556056
## 10                       matrix_factorization_only 0.8323970
## 11        mu+bi_reg+bu_reg+effect_1980+matFact(diffs) 0.8246685
```

This combination of the previous model with matrix factorization on the resulting differentials leads to the best model obtained thus far. Consequently, this model is now to be trained on the full edx dataset and then ultimately tested on the original validation set.

Results

The model developped above is now trained on the full edx training set and then finally validated on the validation set. The model to be used is given below:

$$\hat{Y}_{u,i} = \mu + b_i + b_u + b_{1980} + b_{\text{recosystem}} + \epsilon_{u,i}$$

, where a rating $\hat{Y}_{u,i}$ of user u on movie i is given by the mean of all ratings (μ), a regularized ($\lambda_{b,i} = 1.75$) movie effect term, a regularized ($\lambda_{b,u} = 5.0$) user effect term, a term if the movie appeared before or after the year 1980 (b_{1980}) as well as a term derived by matrix factorization on the remaining differentials ($b_{\text{recosystem}}$).

Training of this model on the edx data set is achieved using the following code.

```
#rating mean
mu <- mean(edx$rating)

#regularized movie and user effects
```

```
reg_movie_effects <- edx %>% group_by(movieId) %>%
  summarize(bi_reg = sum(rating-mu)/(n()+lambda_bi)) # movie effect

edx <- edx %>% left_join(reg_movie_effects, by="movieId")

reg_user_effects <- edx %>% group_by(userId) %>%
  summarize(bu_reg = sum(rating - mu - bi_reg)/(n()+lambda_bu)) # user effect

edx <- edx  %>% left_join(reg_user_effects, by="userId")

#1980 effect
edx <- edx %>% mutate(appearance=str_extract(title, "\\([0-9]{4}\\)")) %>%
  mutate(appearance=as.numeric(str_extract(appearance, "[0-9]+")),
         diff_1=rating-mu-bi_reg-bu_reg)

effect_1980 <- edx %>% mutate(is1980=ifelse(appearance>1980,0,1)) %>%
  group_by(is1980) %>% summarize(effect_1980=mean(diff_1))

edx <- edx %>% mutate(is1980=ifelse(appearance>1980,0,1)) %>%
  left_join(effect_1980, by="is1980") %>%
  mutate(diff=rating-mu-bi_reg-bu_reg-effect_1980) %>% select(-is1980)


#matrix factorization on differentials
set.seed(130484)

edx_2<- data_memory(edx$userId, edx$movieId, rating = edx$diff)

r<-Reco()

params <- r$tune(edx_2, opt=list(dim=seq(5,25,length.out=4), costp_l1=c(0),
                                 costp_l2=c(0.01), costq_l1=c(0),
                                 costq_l2=c(0.01), lrate=c(0.01)))

r$train(edx_2, opt=list(params$min, niter=c(75)))
```

Finally, the model is applied to the validation data predicting the ratings in this data set, which has never been used neither in the construction of the model nor in the training of the algorithm.

```
validation <- validation %>% left_join(reg_user_effects, by="userId") %>%
  left_join(reg_movie_effects, by="movieId")
validation <- validation %>% mutate(appearance=str_extract(title, "\\([0-9]{4}\\)")) %>%
  mutate(appearance=as.numeric(str_extract(appearance, "[0-9]+"))) %>%
  mutate(is1980=ifelse(appearance>1980,0,1)) %>%
  left_join(effect_1980, by="is1980") %>%  mutate(diff=rating-mu-bi_reg-bu_reg-effect_1980) %>%
    select(-is1980)
validation_1 <- data_memory(validation$userId, validation$movieId, rating = validation$diff)
validation <- validation %>% mutate(mat_fact_diff = r$predict(validation_1,out_memory()))

pred <- validation %>% mutate(pred=mu+bi_reg+bu_reg+effect_1980+mat_fact_diff) %>% .$pred
```

Ultimately, the quality of the prediction model is assessed by calculating the resulting RMSE.

```
final_rmse <- RMSE(validation$rating, pred)
final_rmse
```

```
## [1] 0.8244087
```

Conclusion

In summary, a recommendation system was created allowing for the prediction of user ratings of movies based on the Movielens 10M dataset. The final model,

$$\hat{Y}_{u,i} = \mu + b_i + b_u + b_{1980} + b_{\text{recosystem}} + \epsilon_{u,i}$$

using a combination of the mean overall ratings, a movie effect, a user effect, an effect based on the year of appearance of the movie and matrix factorization, was able to predict ratings with a RMSE of 0.82441. This results fulfills the goal of the project to reach a RMSE of less than 0.86490 for predictions on the final validation set.

Of course this model is still rather crude and future work should be aiming at the use of more sophisticated modelling approaches (Chen n.d.). Also, the author's understanding of matrix factorization in general, the recosystem package, the algorithms behind it and its optimal use are admittedly somewhat limited. Therefore, in a first attempt at future improvement of the model a better understanding would most certainly be useful. As a very first step, one could also invest computation time into more iterations for the training of the matrix factorization (see above) which would most certainly yield a certain improvement. Also a more sophisticated tuning of the corresponding parameters in matrix factorization (combined with a better understanding what the individual parameters mean) would be helpful.

Nonetheless, the relatively simple model described above does a surprisingly good job at predicting the ratings within the validation set. Thus it was a very helpful capstone project for the course not only allowing the participant to actively use the knowledge learned but also showing how far one can get with seemingly simple models.

References

Chen, Edwin. n.d. "Winning the Netflix Prize: A Summary." Accessed May 10, 2021. http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/.

Harper, F. Maxwell, and Joseph A. Konstan. 2016. "The MovieLens Datasets: History and Context." *ACM Transactions on Interactive Intelligent Systems* 5 (4): 1–19. https://doi.org/10.1145/2827872.

Irizarry, Rafael A. 2021. "EdX/Harvardx, Professional Certificate Program - Data Science: Capstone." https://www.edx.org/professional-certificate/harvardx-data-science.

Lohr, Steve. 2009. "Netflix Awards $1 Million Prize and Starts a New Contest. Bits: Business, Innovation, Technology, Society." September 21, 2009. https://bits.blogs.nytimes.com/2009/09/21/netflix-awards-1-million-prize-and-starts-a-new-contest/.

Qiu, Yixuan. 2021. "Recosystem: Recommender System Using Parallel Matrix Factorization." May 14, 2021.