Data Science Capstone Movielens Project

Florian Kern

16 May 2019

Table of contents

1. Summary

The Movielens 10M Dataset with more than 10 million ratings of more than 10 000 movies was used to generate an algorithm to predict movie ratings. 90% of the dataset was used to train the algorithm, the remaining 10% were used as a validation set. After taking into account the average movie rating per movie called the movie bias (b_i) and the average user rating per user called the user bias (b_u) the Baysian prediction model was able to achieve a residual mean squared error (RMSE) of 0.8653488. Using regularization the RMSE was further improved to 0.8648170. Compared to using just the native average movie rating as a prediction value this is an impressive improvement of 18.5%.

2. Introduction

Creating a movie recommendation system is a wonderful way to show the capabilities of data science, statistics and machine learning using a practical example with a real life use case. The data used is the MovieLens 10M Dataset[ˆ1] which is a smaller random subset of the much larger MovieLens dataset of users that had at least rated 20 movies. It was obtained from the the grouplens website (https://grouplens.org/datasets/movielens/10m/). The 10M dataset contains: 10000054 ratings, 95580 tags, 10681 movies and 71567 users. It is comprised of userId, movieId, rating, timestamp, title and genres. For the purpose of this project the MovieLens 10M dataset was split into a training set called edx and a testing set called validation. The edx subset contains 9,000,055 ratings, while the validation subset contains only 999,999 ratings or roughly 10% of the data. The task is to generate an algoritm to correctly estimate movie ratings in the validation set using the edx set to train the ML prediction algorithm. As an output the RMSE on the validation set achieved is reported.

[ˆ1] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

3. Methods

The tidyverse library will be used which includes the following packages: * ggplot2, for data visualisation * dplyr, for data manipulation * tidyr, for data tidying * readr, for data import * purrr, for functional programming * tibble, for tibbles, a modern re-imagining of data frames * stringr, for strings * forcats, for factors

```r
library(tidyverse)
```

```
## -- Attaching packages -----------------------------------------------------------------------

## v ggplot2 3.1.1     v purrr   0.3.2
## v tibble  2.1.1     v dplyr   0.8.1
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked _by_ '.GlobalEnv':
##
##     RMSE

## The following object is masked from 'package:purrr':
##
##     lift
```

The Movielens 10M dataset is already in a tidy format, the training and the testing set was created following the given instructions to download the data and to use 10% for validation purposes. Code is only shown not executed, this was done separatly.

Create edx set, validation set, and submission file ##

Note: this process could take a couple of minutes

```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")
```

Download the MovieLens 10M dataset: https://grouplens.org/datasets/movielens/10m/ http://files.grouplens.org/datasets/movielens/ml-10m.zip

```r
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- read.table(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

```r
set.seed(1)
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
```

```r
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

Make sure userId and movieId in validation set are also in edx set

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

Add rows removed from validation set back into edx set

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

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

Exploring the edx dataset showed the following:

```r
# check dimensions and column names
dim(edx)
```

```
## [1] 9000055       6
```

```r
colnames(edx)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

It contains 9,000,055 rows and 6 columns containing userId, movieId, rating, timestamp, title and genres.

```r
# count 0 ratings
length(which(edx$rating ==0))
```

```
## [1] 0
```

It contains no 0 ratings.

```r
# how many distinct movies
edx %>% summarise(n_movies = n_distinct(movieId))
```

```
##   n_movies
## 1    10677
```

```r
n_distinct(edx$movieId)
```

```
## [1] 10677
```

There are 10677 different movies in the dataset and

```r
# how many distinct users
n_distinct(edx$userId)
```

```
## [1] 69878
```

69,878 different users.

```r
# count how many movies in the following genres
gnames <- c("Drama", "Comedy", "Thriller", "Romance")
gnumbers <- sapply(gnames, function(g){
  edx_sg %>% filter(genres == g) %>% tally()
})
gnumbers
```

```
## $Drama.n
## [1] 3910127
##
## $Comedy.n
## [1] 3540930
##
## $Thriller.n
## [1] 2325899
##
## $Romance.n
## [1] 1712100
```

Looking at individual genres Drama contains 3,910,127, Comedy 3,540,930, Thriller 2,325,899, and Romance 1,712,100 ratings.

```
# which is the most rated movie and how many ratings does it have
edx %>% group_by(movieId, title) %>%
    summarize(count = n()) %>%
    arrange(desc(count))
```

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                        count
##      <dbl> <chr>                                                        <int>
##  1     296 Pulp Fiction (1994)                                          31362
##  2     356 Forrest Gump (1994)                                          31079
##  3     593 Silence of the Lambs, The (1991)                             30382
##  4     480 Jurassic Park (1993)                                         29360
##  5     318 Shawshank Redemption, The (1994)                             28015
##  6     110 Braveheart (1995)                                            26212
##  7     457 Fugitive, The (1993)                                         25998
##  8     589 Terminator 2: Judgment Day (1991)                            25984
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (19~ 25672
## 10     150 Apollo 13 (1995)                                             24284
## # ... with 10,667 more rows
```

Pulp fiction (1994) with 31,362 ratings is the movie with the most ratings.

Looking at the ratings:

```
# looking at the distribution of the ratings
ratings <- edx %>% group_by(rating) %>% summarize(count = n()) %>%
    arrange(desc(count))
# obtaining the percentage the 3 most common ratings have over total ratings
sum(ratings$count[1:3])/sum(ratings$count)
```

```
## [1] 0.6777496
```

```
ratings
```

```
## # A tibble: 10 x 2
##    rating   count
##     <dbl>   <int>
##  1    4   2588430
##  2    3   2121240
##  3    5   1390114
##  4    3.5  791624
##  5    2    711422
##  6    4.5  526736
## 7     1     345679
## 8     2.5   333010
## 9     1.5   106426
## 10    0.5    85374
```

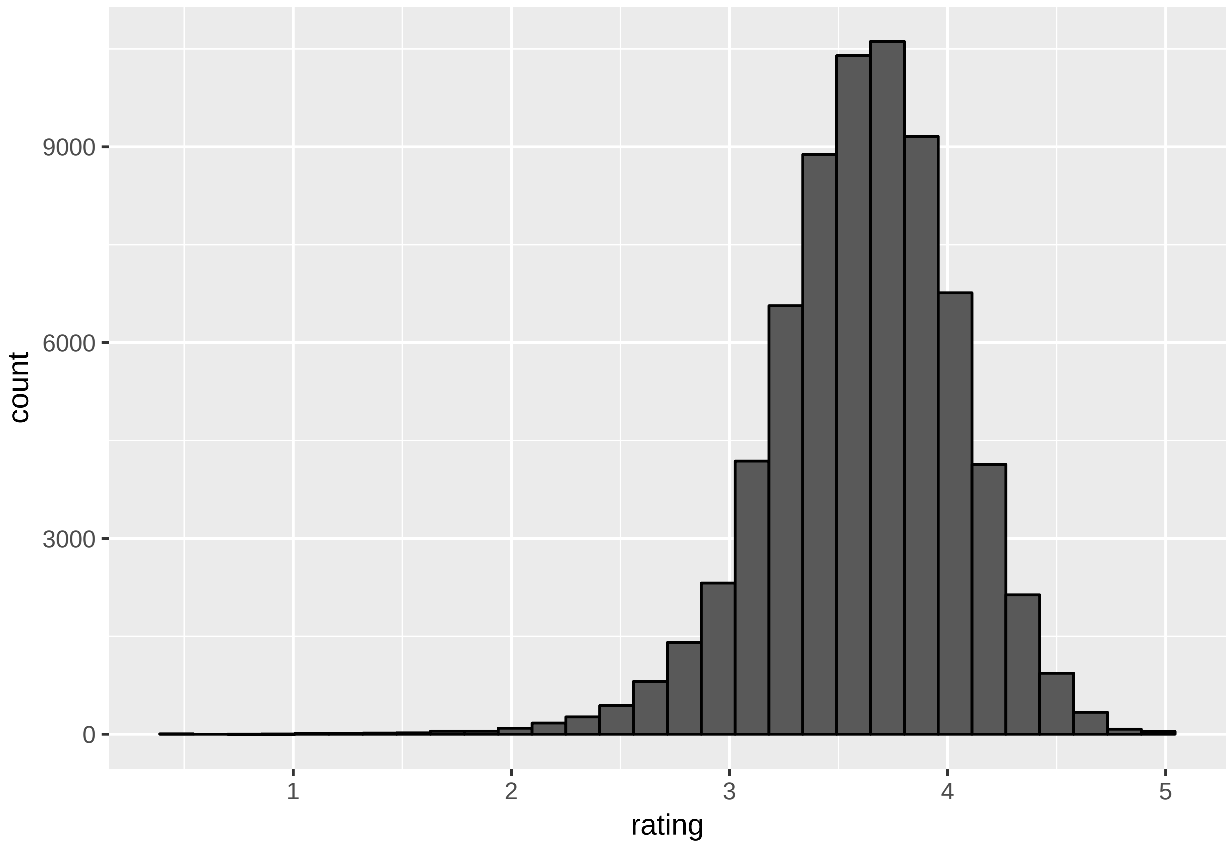
4.0, 3.0 and 5.0 are the most commonly given ratings and account for more than 67% of the ratings.

Looking at a histogram of user ratings:

```
# the average rating per user
mean(edx$rating)
```

```
## [1] 3.512465
```

```
# plot the average ratings per user as histogram
edx %>%
  group_by(userId) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(rating)) +
  geom_histogram(bins = 30, color = "black")
```



The average user rating is about 3.51 and it looks normally distributed with a longer tail towards 0.

All this shows that the dataset at hand is quite large with more than 9 million rows and several features including userId, movieId, rating, title, timestamp and genre that can be used to build a valid prediction model. From previous experience the user and movie bias should have the most profound impact on the prediction accuracy. Therefore a machine learning (ML) algorithm using the naive Bayes approach to account for the bias introduced by the user and movie effects.

RMSE is used as a readout, therefore we create a RMSE function:

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

4. Results

Let's begin by determining the naive RMSE by taking the average rating as prediction value to get an idea of what we are dealing with:

```r
# calculate the average rating
mu_hat <- mean(edx$rating)
mu_hat
```

```
## [1] 3.512465
```

```r
# "predict" ratings by just using the average rating
naive_rmse <- RMSE(validation$rating, mu_hat)
naive_rmse
```

```
## [1] 1.061202
```

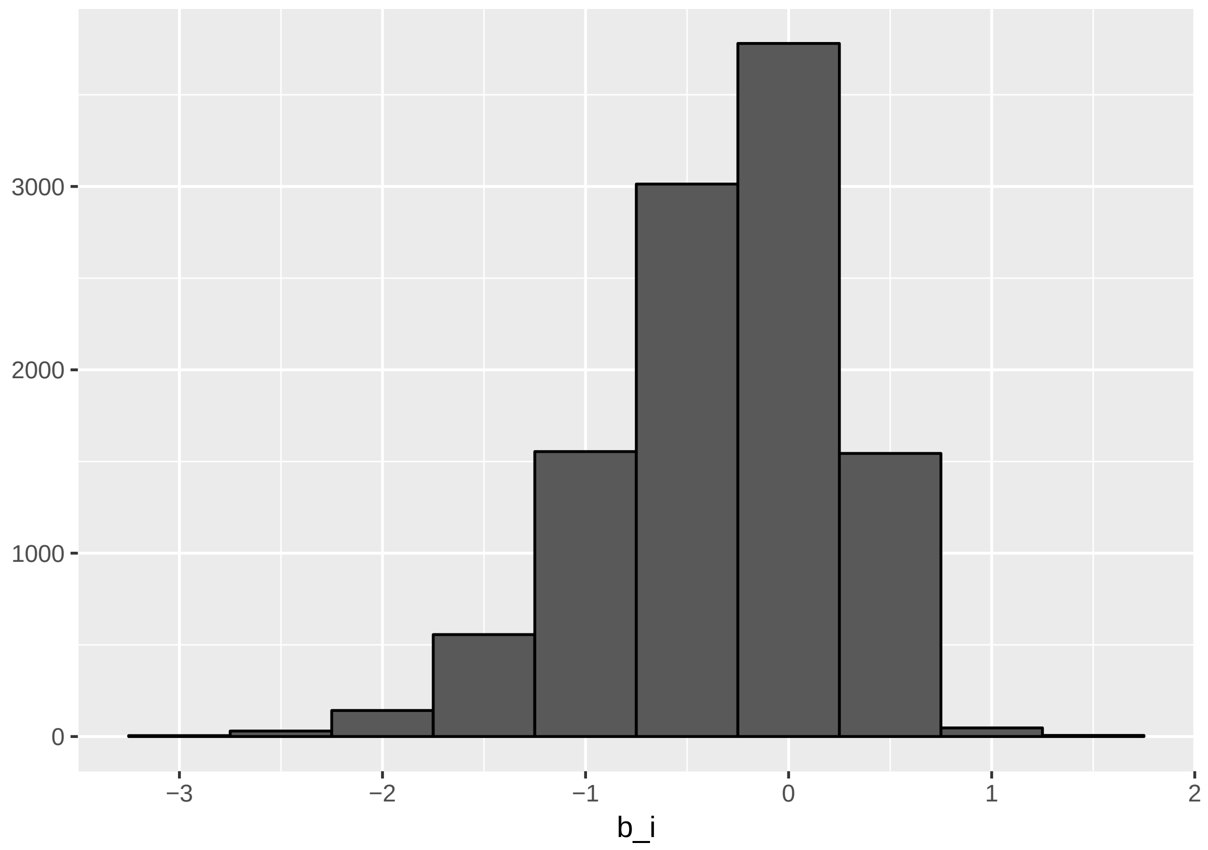
```r
# storing the result in a table to monitor the optimization process
rmse_results_0 <- tibble(method = "average only", RMSE = naive_rmse)
rmse_results_0
```

```
## # A tibble: 1 x 2
##   method        RMSE
##   <chr>        <dbl>
## 1 average only  1.06
```

Just using the average leads to a typical error of more than 1, which is unacceptable.

Let's introduce a movie bias factor (b_i) in our equation by calculating the average rating for each movie.

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

# calculation of the movie averages
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# plot a histogram of the calculated movie bias b_i
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"))
```



```r
# predict ratings accounting for the movie bias b_i
predicted_ratings_1 <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

# calculate the RMSE for this model
model_1_rmse <- RMSE(predicted_ratings_1, validation$rating)

# add the result to the RMSE table
rmse_results_1 <- bind_rows(rmse_results_0,
                          tibble(method="movie bias",
                                 RMSE = model_1_rmse))
# show results
rmse_results_1
```

```
## # A tibble: 2 x 2
##   method        RMSE
##   <chr>        <dbl>
## 1 average only 1.06
## 2 movie bias   0.944
```

Considering the movie bias b_i improves the RMSE to 0.94, but this is still not really acceptable.

Let's calculate the user bias.

```r
# calculating the user averages
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# predicting the ratings taking the movie bias b_i and the user bias b_u into account
predicted_ratings_2 <- validation %>%
  left_join(user_avgs, by='userId') %>% # calculated above
  mutate(pred = mu + b_u) %>%
  pull(pred)

# obtaining the RMSE
model_2_rmse <- RMSE(predicted_ratings_2, validation$rating)

# adding the RMSE result to the results list
rmse_results_2 <- bind_rows(rmse_results_1,
                            tibble(method="user bias",
                                   RMSE = model_2_rmse))
# show results
rmse_results_2
```

```
## # A tibble: 3 x 2
##   method        RMSE
##   <chr>        <dbl>
## 1 average only 1.06
## 2 movie bias   0.944
## 3 user bias    0.995
```

The user bias b_u alone also improves the RMSE compared to the using the average only approach, but less than the movie bias b_i.

Let's use both factors and combine the movie bias b_i and the user bias b_u in the prediction model:

```
# predicting the ratings taking the movie bias b_i and the user bias b_u into account
predicted_ratings_3 <- validation %>%
  left_join(movie_avgs, by='movieId') %>% # were calculated in the previous part
  left_join(user_avgs, by='userId') %>% # were calculated in the previous part
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# obtaining the RMSE
model_3_rmse <- RMSE(predicted_ratings_3, validation$rating)

# adding the RMSE result to the results list
rmse_results_3 <- bind_rows(rmse_results_2,
                            tibble(method="movie bias + user bias",
                                   RMSE = model_3_rmse))
# show results
rmse_results_3
```

```
## # A tibble: 4 x 2
##   method                  RMSE
##   <chr>                  <dbl>
## 1 average only           1.06
## 2 movie bias             0.944
## 3 user bias              0.995
## 4 movie bias + user bias 0.865
```

Combining and taking into account the movie bias b_i and the user bias b_u in our prediction model the

RMSE can be reduced to less than 0.865, which is lower than the required RMSE of 0.87750.

Let's have a look at where we made the biggest mistakes estimating the movie bias.

```
# showing the biggest mistakes (top10)
edx %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(residual = rating - (mu + b_i)) %>%
  arrange(desc(abs(residual))) %>%
  select(title, residual) %>% slice(1:10)
```

```
##                                title  residual
## 1       From Justin to Kelly (2003)  4.097990
## 2       From Justin to Kelly (2003)  4.097990
## 3             Pokémon Heroes (2003)  3.970803
## 4             Pokémon Heroes (2003)  3.970803
## 5  Shawshank Redemption, The (1994) -3.955131
## 6  Shawshank Redemption, The (1994) -3.955131
## 7  Shawshank Redemption, The (1994) -3.955131
## 8  Shawshank Redemption, The (1994) -3.955131
## 9  Shawshank Redemption, The (1994) -3.955131
## 10 Shawshank Redemption, The (1994) -3.955131
```

Let's create a movieId title database to facilitate lookup.

```
# load edx dataset, select ohnly movieId and title, make them distinct (unique)
movie_titles <- edx %>%
  select(movieId, title) %>%
  distinct()
```

Movie bias and number of ratings of the top 10 and low 10 rated movies.

```
# showing the best 10
edx %>% count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:10)
```

```
## Joining, by = "movieId"

## # A tibble: 10 x 3
##    title                                                          b_i     n
##    <chr>                                                        <dbl> <int>
##  1 Hellhounds on My Trail (1999)                                 1.49     1
##  2 Satan's Tango (Sátántangó) (1994)                             1.49     2
##  3 Shadows of Forgotten Ancestors (1964)                         1.49     1
##  4 Fighting Elegy (Kenka erejii) (1966)                          1.49     1
##  5 Sun Alley (Sonnenallee) (1999)                                1.49     1
##  6 Blue Light, The (Das Blaue Licht) (1932)                      1.49     1
##  7 Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko~  1.24     4
##  8 Human Condition II, The (Ningen no joken II) (1959)           1.24     4
##  9 Human Condition III, The (Ningen no joken III) (1961)         1.24     4
## 10 Constantine's Sword (2007)                                    1.24     2
```

```
# showing the worst 10
edx %>% count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(title, b_i, n) %>%
  slice(1:10)
```

```
## Joining, by = "movieId"
```

```
## # A tibble: 10 x 3
##    title                                        b_i     n
##    <chr>                                      <dbl> <int>
##  1 Besotted (2001)                            -3.01     2
##  2 Hi-Line, The (1999)                        -3.01     1
##  3 Accused (Anklaget) (2005)                  -3.01     1
##  4 Confessions of a Superhero (2007)          -3.01     1
##  5 War of the Worlds 2: The Next Wave (2008) -3.01     2
##  6 SuperBabies: Baby Geniuses 2 (2004)        -2.72    56
##  7 Hip Hop Witch, Da (2000)                   -2.69    14
##  8 Disaster Movie (2008)                      -2.65    32
##  9 From Justin to Kelly (2003)                -2.61   199
## 10 Criminals (1996)                           -2.51     2
```

Since many movies either in the top 10 or low 10 have only very few total ratings we want to correct for this by introducing regularization, which penalizes seldomly rated movies:

```
# at the moment a randomly chosen regularization value called lambda
lambda <- 2.5

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

# regularized movie bias using lambda and number of ratings
movie_reg_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
```

if we now look at the top 10 and the 10 worst movies after regularization

```
# showing the top 10 after regularization
edx %>%
  count(movieId) %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  left_join(movie_titles, by = "movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:10)
```

```
## # A tibble: 10 x 3
##    title                                              b_i     n
##    <chr>                                            <dbl> <int>
##  1 Shawshank Redemption, The (1994)                 0.943 28015
##  2 Godfather, The (1972)                            0.903 17747
##  3 More (1998)                                      0.886     7
##  4 Usual Suspects, The (1995)                       0.853 21648
##  5 Schindler's List (1993)                          0.851 23193
##  6 Casablanca (1942)                                0.808 11232
##  7 Rear Window (1954)                              0.806  7935
##  8 Sunset Blvd. (a.k.a. Sunset Boulevard) (1950) 0.803  2922
##  9 Third Man, The (1949)                           0.798  2967
## 10 Double Indemnity (1944)                         0.797  2154
```

```
edx %>%
  count(movieId) %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  left_join(movie_titles, by = "movieId") %>%
  arrange(b_i) %>%
  select(title, b_i, n) %>%
  slice(1:10)
```

```
## # A tibble: 10 x 3
##    title                                                 b_i     n
##    <chr>                                               <dbl> <int>
##  1 SuperBabies: Baby Geniuses 2 (2004)                 -2.60    56
##  2 From Justin to Kelly (2003)                         -2.58   199
##  3 Disaster Movie (2008)                               -2.46    32
##  4 Pokémon Heroes (2003)                               -2.44   137
##  5 Carnosaur 3: Primal Species (1996)                  -2.34    68
##  6 Glitter (2001)                                      -2.32   339
##  7 Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie) (2002) -2.31   202
##  8 Gigli (2003)                                        -2.30   313
##  9 Barney's Great Adventure (1998)                     -2.30   208
## 10 Hip Hop Witch, Da (2000)                            -2.28    14
```

We can see that after regularization that the top 10 contain many well known and therefore expected movies. Apart from one movie "More (1998)" all others have more than 2,000 ratings, the top half even more than 11,000 ratings, which provides confidence in the results.

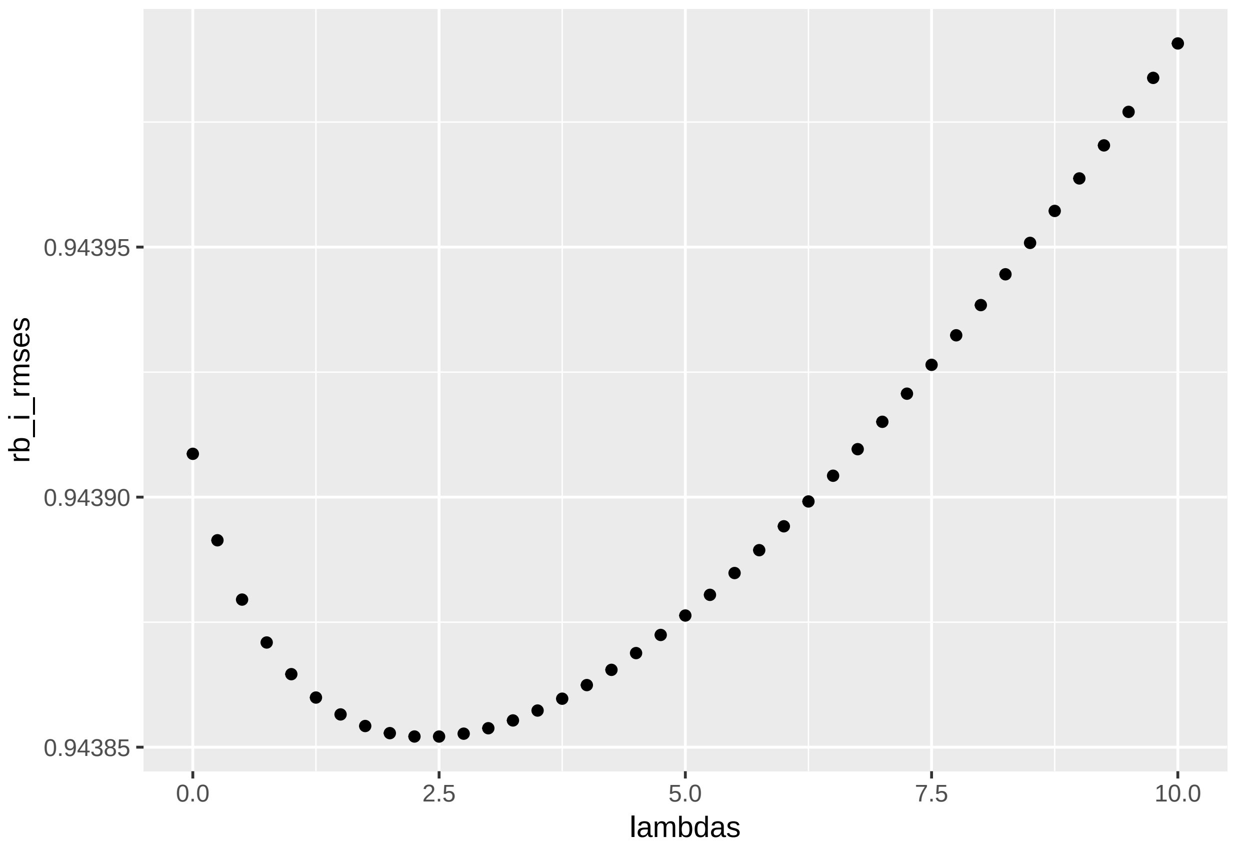
Lets calculate the RMSE.

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

# predicting ratings using a regularized movie bias b_i
###### this gives an error, movie_reg_avgs and edx do not have the same length, not all movieIds presen
predicted_ratings_4 <- validation %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)

# obtaining the RMSE
model_4_rmse <- RMSE(predicted_ratings_4, validation$rating)

# adding the RMSE result to the results list
rmse_results_4 <- bind_rows(rmse_results_3,
                          tibble(method="regularized movie bias",
                          RMSE = model_4_rmse))
# show results
rmse_results_4
```

```
## # A tibble: 5 x 2
##   method                  RMSE
##   <chr>                  <dbl>
## 1 average only            1.06
## 2 movie bias              0.944
## 3 user bias               0.995
## 4 movie bias + user bias  0.865
## 5 regularized movie bias  0.944
```

Choosing the penalty term for the movie bias:

```r
# penalty value (lambda) sequence
lambdas <- seq(0, 10, 0.25)

# average movie rating
mu <- mean(edx$rating)

# calculating sum of rating - mu, and number of movies
just_the_sum_rb_i <- edx %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())

# predict ratings with the sequence of lambdas and obtain the RMSEs
rb_i_rmses <- sapply(lambdas, function(l){
  predicted_ratings <- validation %>%
    left_join(just_the_sum_rb_i, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, validation$rating))
})

# qplot lambas vs RMSES
qplot(lambdas, rb_i_rmses)
```



```
# determine the lambda with the smallest RMSE
lambdas[which.min(rb_i_rmses)]
```

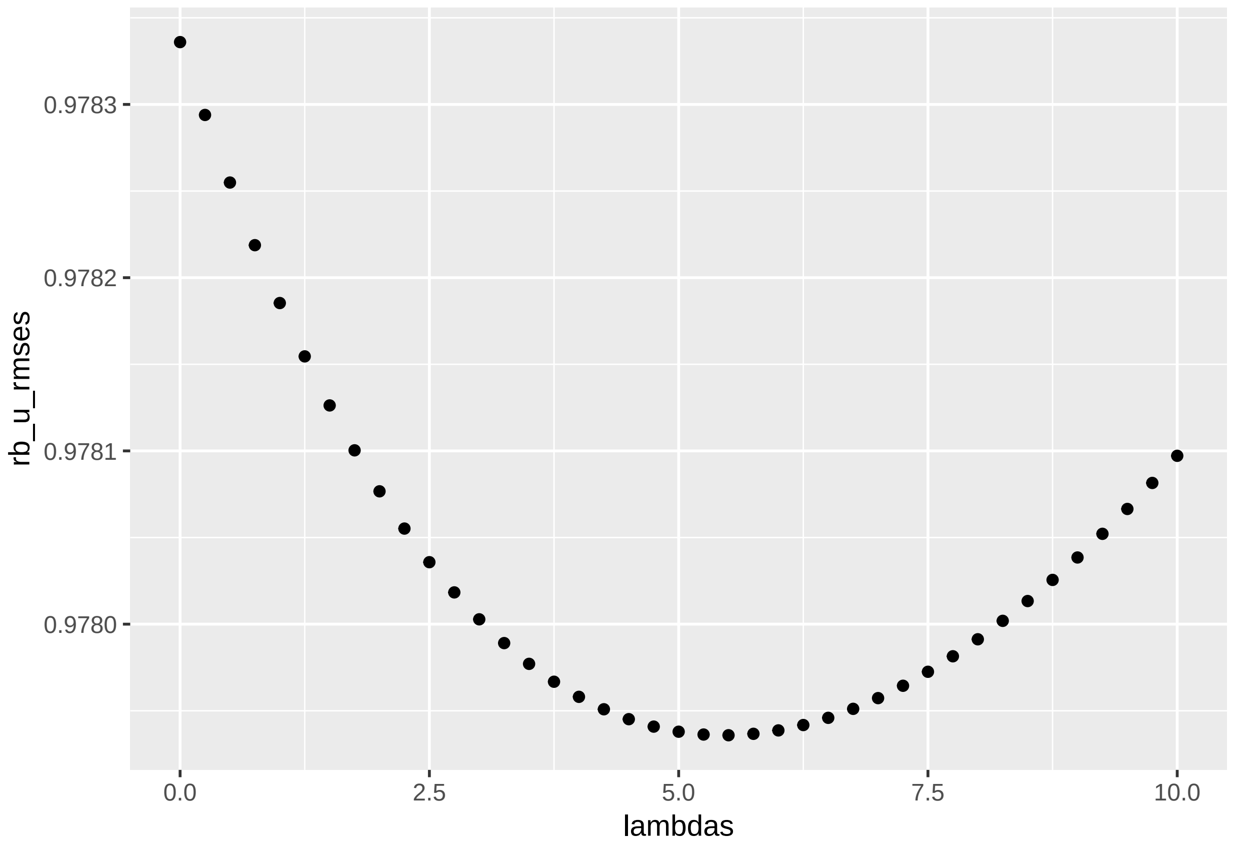
```
## [1] 2.5
```

The best lambda value for movie bias b_i is 2.5.

Choosing the penalty term for the user bias:

```
# penalty value (lambda) sequence
lambdas <- seq(0, 10, 0.25)

# average movie rating
mu <- mean(edx$rating)

# calculating sum of rating - mu, and number of users
just_the_sum_rb_u <- edx %>%
  group_by(userId) %>%
  summarize(s_u = sum(rating - mu), n_u = n())

# predict ratings with the sequence of lambdas and obtain the RMSEs
rb_u_rmses <- sapply(lambdas, function(l){
  predicted_ratings_u <- validation %>%
    left_join(just_the_sum_rb_u, by='userId') %>%
    mutate(b_u = s_u/(n_u+l)) %>%
    mutate(pred_u = mu + b_u) %>%
    pull(pred_u)
  return(RMSE(predicted_ratings_u, validation$rating))
})
```

```r
# qplot lambas vs RMSES
qplot(lambdas, rb_u_rmses)
```



```r
# determine the lambda with the smallest RMSE
lambdas[which.min(rb_u_rmses)]
```

```
## [1] 5.5
```

The best lambda value for user bias b_u is 5.5.

Lets determine the RMSE with these values:

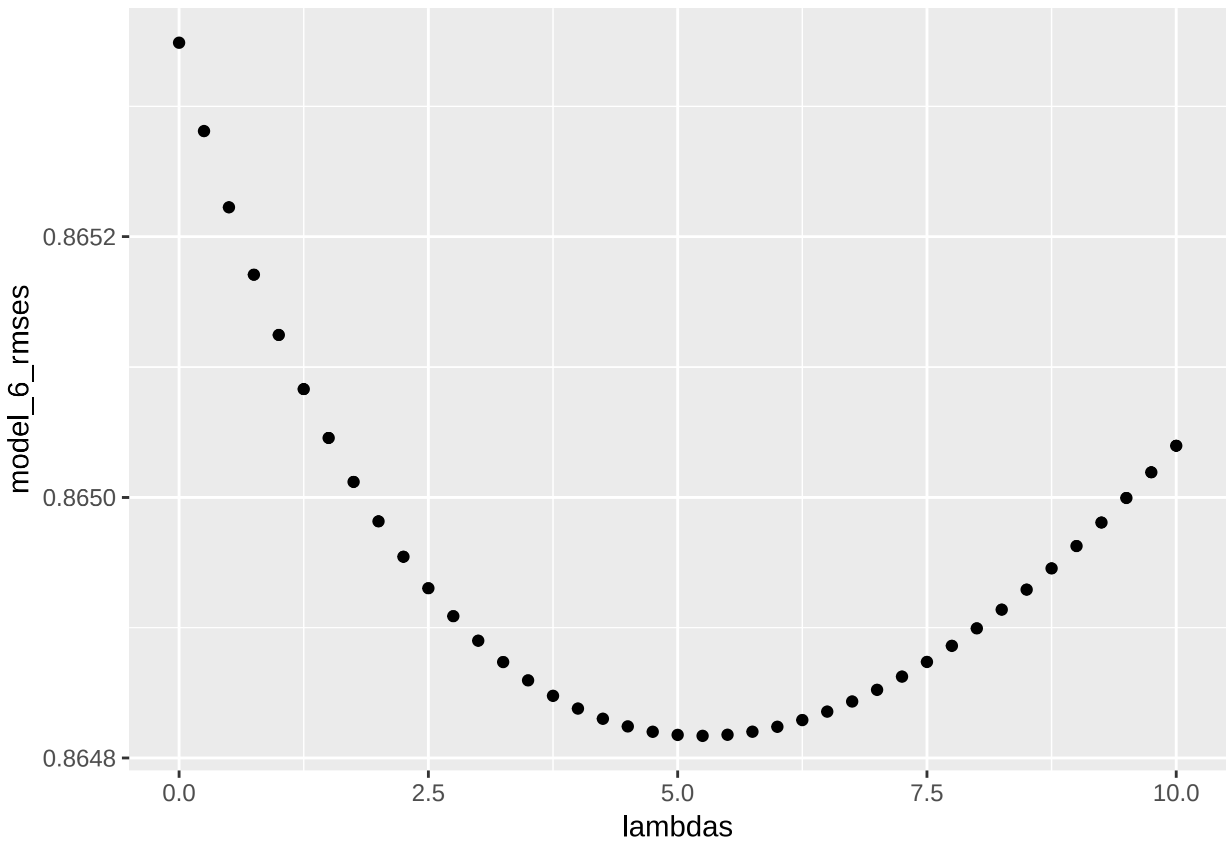
```r
# movie lambda
l_i <- 2.5
# user lambda
l_u <- 5.5

# average user rating
mu <- mean(edx$rating)

# movie bias using regularization
b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l_i))

# user bias using regularization
b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l_u))

# predict ratings with regularized movie and user biases
predicted_ratings_5 <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

# obtain the RMSE
model_5_rmse <- RMSE(predicted_ratings_5, validation$rating)

# add the RMSE
rmse_results_5 <- bind_rows(rmse_results_4,
                          tibble(method="indiv. regul. movie bias + user bias",
                                     RMSE = model_5_rmse))
# show results
rmse_results_5
```

```
## # A tibble: 6 x 2
##   method                                RMSE
##   <chr>                                <dbl>
## 1 average only                         1.06
## 2 movie bias                           0.944
## 3 user bias                            0.995
## 4 movie bias + user bias               0.865
## 5 regularized movie bias               0.944
## 6 indiv. regul. movie bias + user bias 0.865
```

Let’s use cross validation to chose lambda.

```r
# penalty value (lambda) sequence
lambdas <- seq(0, 10, 0.25)

# cross validate the lambda sequence
model_6_rmses <- sapply(lambdas, function(l){

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

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

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

  # predict ratings
  predicted_ratings_6 <-
```

```
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  # return RMSE
  return(RMSE(predicted_ratings_6, validation$rating))
})

# qplot of lambdas vs. RMSES
qplot(lambdas, model_6_rmses)
```



```
# determine lambda with the smallest RMSE
lambda <- lambdas[which.min(model_6_rmses)]
lambda
```

```
## [1] 5.25
```

The calculated value that minimizes the RMSE is 5.25.

RMSE result with the final model.

```
# obtaining the rmse
rmse_results_6 <- bind_rows(rmse_results_5,
                            data_frame(method="regularized movie bias + user bias",
                                       RMSE = min(model_6_rmses)))
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

```
# final results table
rmse_results_6
```

```
## # A tibble: 7 x 2
##   method                               RMSE
##   <chr>                               <dbl>
## 1 average only                        1.06
## 2 movie bias                          0.944
## 3 user bias                           0.995
## 4 movie bias + user bias              0.865
## 5 regularized movie bias              0.944
## 6 indiv. regul. movie bias + user bias 0.865
## 7 regularized movie bias + user bias   0.865
```

After cross validation with an optimized lambda of 5.25 a RMSE of 0.8648 is achieved, which is a slight improvement over the unregularized movie bias and user bias model as well as the individually regularized lambdas.

5. Conclusion

Predicting movie preferences is not only a really interesting example to show what ML algorithms can do but at least since Netflix was found it has a real life use case. The most basic approach would be to use just the average movie rating as a prediction value. With such a large dataset it might seem like a safe bet, but the calculated RMSE yields an unacceptable error greater than 1. Using a naive Bayes approach and defining a user and a movie bias term the RMSE can be reduced to less than 0.865 and with regularization the RMSE was further improved to 0.8648170. Compared to using just the native average movie rating as a prediction value this is an impressive improvement of 18.5%. Other predictors that could be taken into account using this dataset are the total time a movie was available as older movies aggregate more votes over time, and genre effects as viewer groups tend to have similar voting patterns. Furthermore, if the total dataset from the provider was available the user age and the change of the user age over time could be taken into account by defining age groups or the IP adresses could be used to define countries and regions and discern a cultural background. In general using machine learning algorithms to build a reliable prediction model enables movie streaming providers, like Netflix, to bring the movie recommendation system to a personal level.