Report for recommendation system

Author: Reinhard Palaver

Date: 24 March 2020

Introduction

In following report we are talking about a recommendation system based on the idea from the Netflix challenge.

The principle idea from Netflix was: The company's challenge, begun in October 2006, was both geeky and formidable: come up with a recommendation software that could do a better job accurately predicting the movies customers would like than Netflix's in-house software, Cinematch. To qualify for the prize, entries had to be at least 10 percent better than Cinematch.

Read more at these sites:

https://bits.blogs.nytimes.com/2009/09/21/netflix-awards-1-million-prize-and-starts-a-new-contest/

https://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/

https://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf

In our "challenge" we'll take just a subset of the whole data and try to figure out a computed **result** defined as "root mean square error" (RMSE). It will be explained in detail later on.

The database we'll work on it, is generated by a given R-script from https://edx.org.

As a result we received two datasets called: **edx** and **validation**.

Dataset exploration

The edx dataset looks like:

```
edx %>% slice(1:10) %>% knitr::kable()
```

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy
1	356	5	838983653	Forrest Gump (1994)	Comedy|Drama|Romance|War
1	362	5	838984885	Jungle Book, The (1994)	Adventure|Children|Romance
1	364	5	838983707	Lion King, The (1994)	Adventure|Animation|Children|Dra
1	370	5	838984596	Naked Gun 33 1/3: The Final Insult (1994)	Action|Comedy

The dimensions (rows and columnes) of the dataset edx are:

Number of rows:

```
dim(edx)[1]
```

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

Number of columns:

```
dim(edx)[2]
```

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

The different variables at the dataset from edx and validation are:

```
names(edx)
```

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

Explanation of our different variables:

userId -> integer value for the different users

movieId -> numeric value for the different movies

rating -> numeric value for the different ratings as follows:

```
edx %>% group_by(rating) %>% summarize(number=n()) %>% knitr::kable()
```

rating	number
0.5	85374
1.0	345679
1.5	106426
2.0	711422
2.5	333010
3.0	2121240
3.5	791624
4.0	2588430
4.5	526736
5.0	1390114

timestamp -> integer value for date and time of rating (value shown are seconds starting from 1970-01-01)

title -> character value for title of the movie and year of release

genre -> character value/values (could be more than one) of the genre(s)

Each row represents a rating given by one user to one movie.

The number of unique movies at our dataset edx are:

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

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

And the number of unique users are:

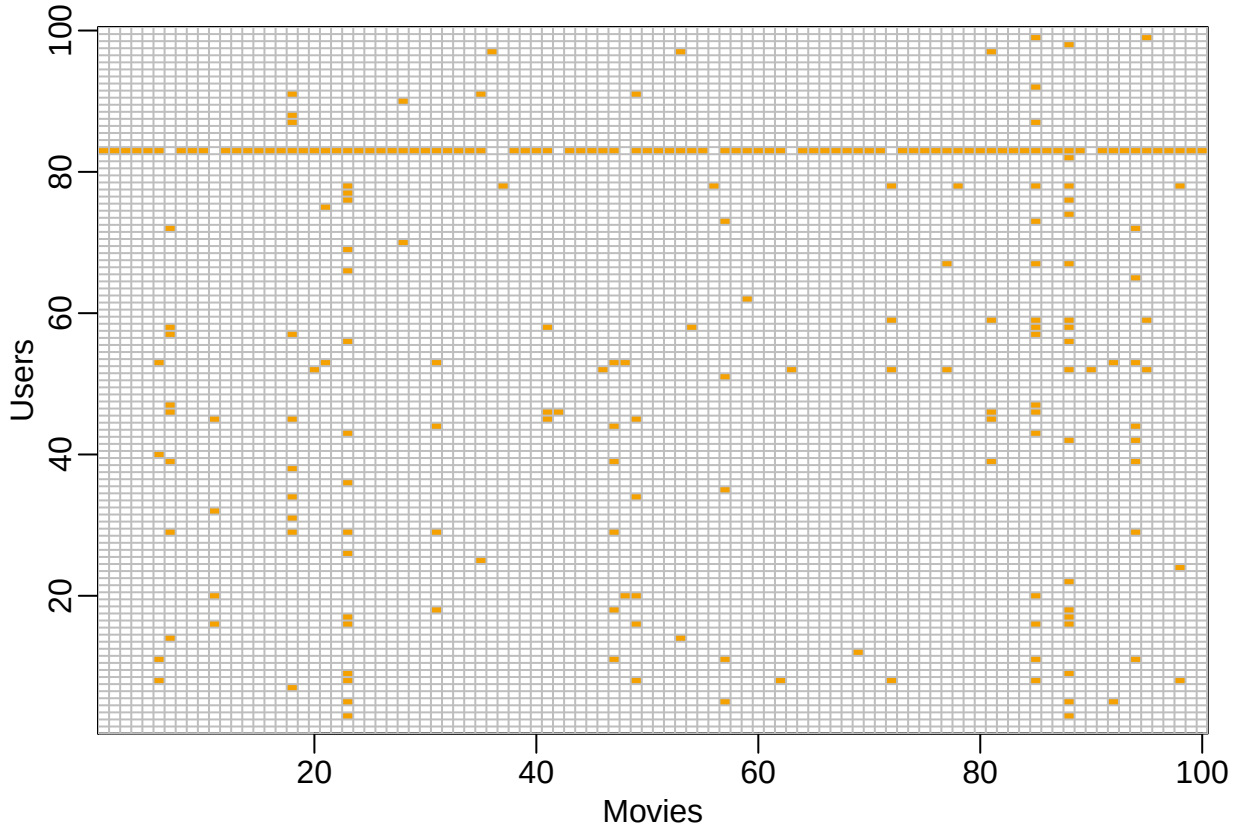
```
n_distinct(edx$userId)
```

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

If we multiply those two numbers, we get a number larger than 740 millions, yet our data table has about 9 million rows. This implies that not every user rated every movie.

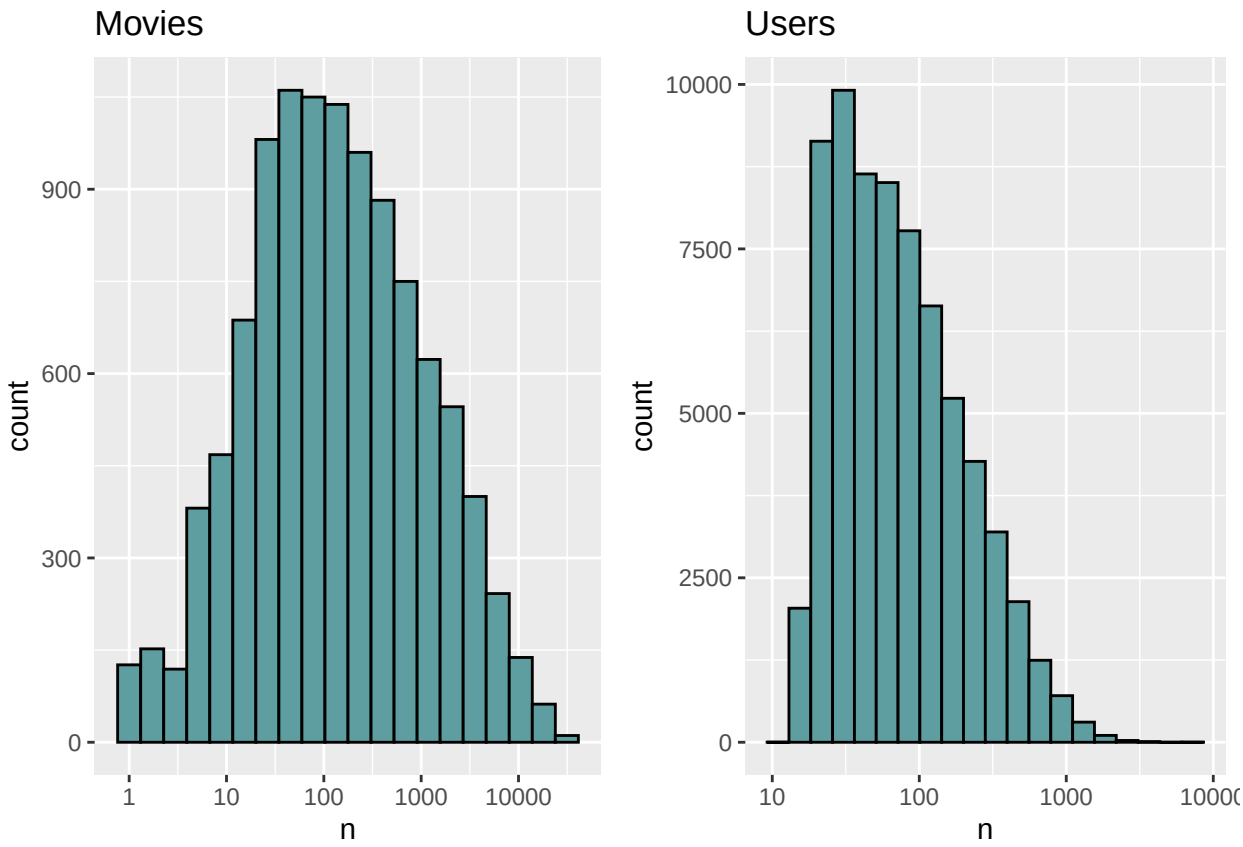
The dataset **edx** is used for training of different approaches and the dataset **validation** is used for the final check or coming up with the final result of RMSE.

This machine learning challenge is a bit complicate, because each outcome Y has a different set of predictors. To see this, note that if we are predicting the rating for movie i by user u, in principle, all other ratings related to movie i and by user u may be used as predictors, but different users rate different movies and a different number of movies. Furthermore, we may be able to use information from other movies that we have determined are similar to movie i or from users determined to be similar to user u. In essence, the entire matrix can be used as predictors for each cell. To see how sparse the entire matrix is, here the matrix for a random sample of 100 movies and 100 users is shown with yellow indicating a user movie combination for which we have a rating:



Let us look at some of the general properties of the data to better understand the challenge.

The first thing we notice is that some movies get rated more than others. Below you will see the distribution. This should not surprise us given that there are blockbuster movies watched by millions and artsy, independent movies watched by just a few. Our second observation is that some users are more active than others at rating movies:



Data preparation for modeling

First of all we have to create two datasets for training and testing, based on our edx dataset. The dataset for testing during modeling will be 20% from edx dataset.

```
# create test and train sets (test set will be 20% of edx data)
set.seed(755, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[test_index,]
test_set <- edx[-test_index,]
rm(test_index)
```

To make sure we don't include users and movies in the test set that do not appear in the training set, we will remove these.

```
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

Modeling

Now we are going to compute the different models and therefore we are working with the loss function RMSE and as a baseline the goal which we have to achieve in our project.

Our goal is to be better than **0.8649**.

The root mean squared error (RMSE) is the square root of the MSE. And the MSE is the average squared error of the predictions. Larger errors are weighted more than smaller one, thus of the error doubles, the MSE increases four times. One the other hand, errors smaller than 1 also decrease faster, for an example an error of 0.5 results in MSE of 0.25.

Therefore we create a simple function like this, which we are using at the further statistical computing.

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

First model

The first model is the simplest one, where we predict the same rating for all movies, regardless of the user and movie. Therefore we compute the average of all ratings like this:

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

```
## [1] 3.512219
```

And then we compute the RMSE on the test set data.

```
naive_rmse <- RMSE(test_set$rating, mu)
naive_rmse
```

```
## [1] 1.060172
```

We get a RMSE of about 1.06. That's pretty big.

For having a history of the different results from the different models we are creating a table to store the results we obtain. We will call this table **rmse_results**.

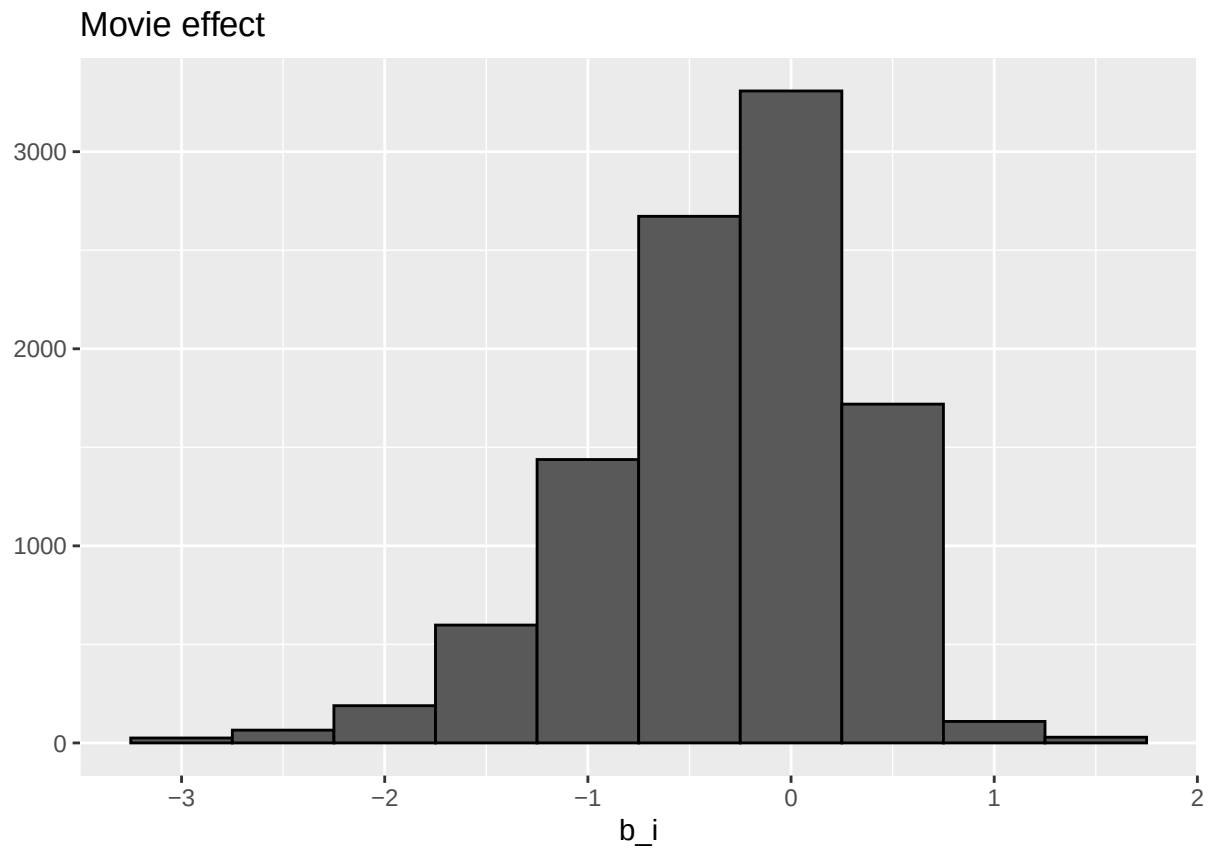
method	RMSE
Just the average	1.060172

Second model (movie effect)

Now we will have a look for the different ratings for movies, because we know that for instance some movies are generally rated higher than others. Therefore we are computing the average rating per movie (movie_avgs), subtract the overall average (mu) and plot it for analysis reason to show the variability (b_i).

```
movie_avgs <- train_set %>%
  group_by(movieId) %>%
```

```
  summarize(b_i = mean(rating - mu))
qplot(b_i, data = movie_avgs, bins = 10, color = I("black")) + ggtitle("Movie effect")
```



We see that the movie effect is normally left skewed distributed. We have to know, that the overall average is about 3.5 and we are able to see, that some movies are good and other movies are bad rated.

And now we will do a prediction with the values of b_i and store it in our table **rmse_results**.

```
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
model_1_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                  data_frame(method="Movie Effect Model",
                  RMSE = model_1_rmse))
options(digits = 2)
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.06
Movie Effect Model	0.95

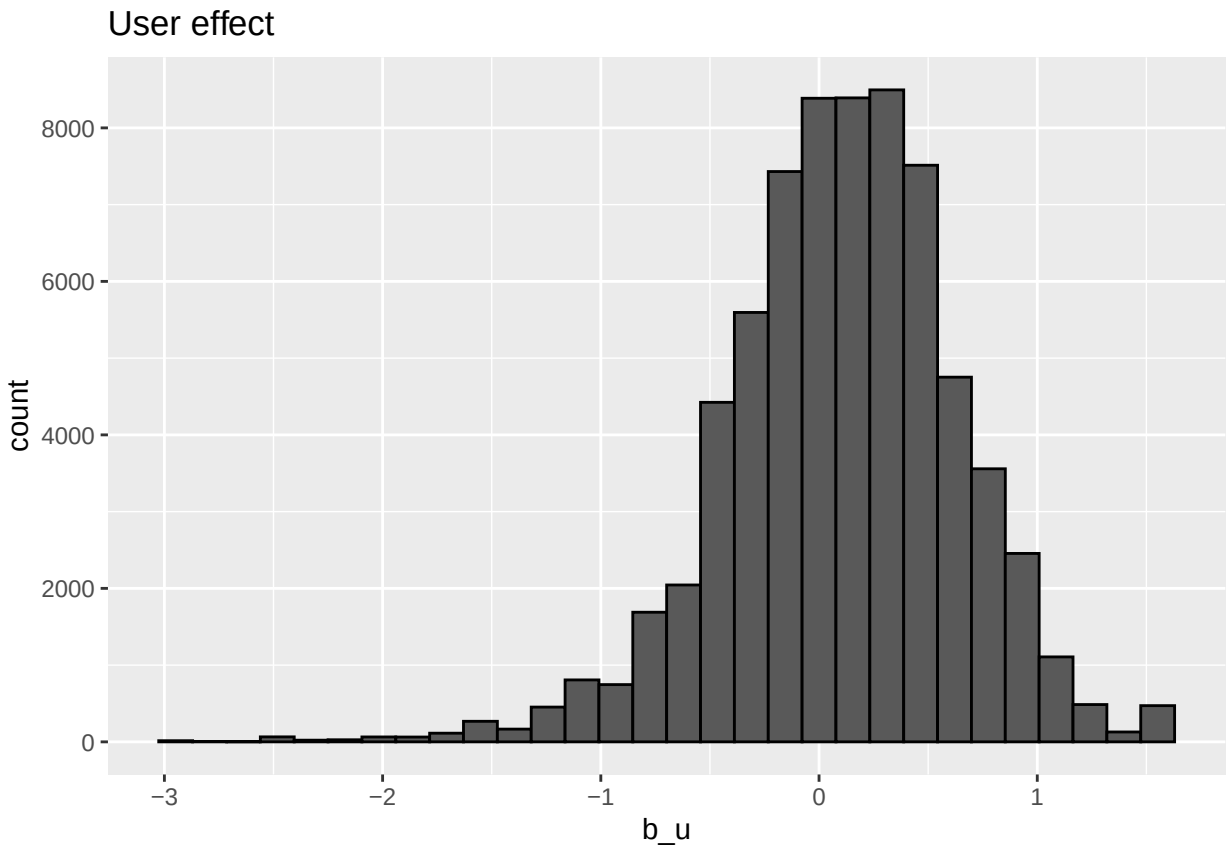
We can see that our RMSE improves round about 0.11 points.

Third model (user effect)

Now let us have a look for the different users and the effect based on the user ratings.

We are also computing the average rating per user (user_avgs) and subtract the overall rating (mu) and show with a histogram the variability for those users that have rated over 100 movies.

```
train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = I("black")) + ggtitle("User effect")
```



Compared to the movie effect we can see that there are more users somewhere in the middle and the user effect is normally distributed.

Now we compute the user averages considering the movie effect before we are doing our prediction.

```
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

And now we will do a prediction with the values of b_u and b_i and store it in our table **rmse_results**.

```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="User/movie Effect Model",
                                     RMSE = model_2_rmse))
options(digits = 3)
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.060
Movie Effect Model	0.946
User/movie Effect Model	0.884

And once more we can see that our improvement of RMSE becomes round about 0.06 points better and therefore we extend the shown digits to three digits because we are coming closer to our final goal.

Fourth model (Regularization)

To improve our results, we will use regularization. Because this is one of the techniques that was used by the winners of the Netflix challenge.

Regularization constrains the total variability of the effect sizes by penalizing large estimates that come from small sample sizes.

For a better understanding let us show the largest mistakes that we made when only using the movie effects. Here are 10 of the largest mistakes:

```
test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(residual = rating - (mu + b_i)) %>%
  arrange(desc(abs(residual))) %>%
  select(title,  residual) %>% slice(1:10) %>% knitr::kable()
```

title	residual
Amati Girls, The (2000)	-4.50
Pom Poko (a.k.a. Raccoon War, The) (Heisei tanuki gassen pompoko) (1994)	4.50
Safe Conduct (Laissez-Passer) (2002)	-4.50
War and Peace (Jang Aur Aman) (2001)	-4.50
Pom Poko (a.k.a. Raccoon War, The) (Heisei tanuki gassen pompoko) (1994)	4.50
Safe Conduct (Laissez-Passer) (2002)	-4.50
Adventures of Don Juan (1948)	-4.50
Carnosaur 3: Primal Species (1996)	4.19
From Justin to Kelly (2003)	4.12
From Justin to Kelly (2003)	4.12

These seem to be obscure movies and for better understanding we create a database that includes the movieID and titles using this code:

```
movie_titles <- train_set %>%
  select(movieId, title) %>%
  distinct()
```

After that we will have a look for the best 10 and worst 10 movies with their estimates of b_i.

best 10 movies:

```
# show best 10 movies
movie_avgs %>% left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	b_i
Homage (1995)	1.49
Somebody is Waiting (1996)	1.49
Unforgotten: Twenty-Five Years After Willowbrook (1996)	1.49
Amati Girls, The (2000)	1.49
More (1998)	1.49
Strait-Jacket (1964)	1.49
Safe Conduct (Laissez-Passer) (2002)	1.49
House with Laughing Windows, The (La Casa dalle Finestre che Ridono) (1976)	1.49
War and Peace (Jang Aur Aman) (2001)	1.49
Tokyo Joe (1949)	1.49

worst 10 movies:

```
movie_avgs %>% left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(title, b_i) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	b_i
Autopsy (Macchie Solari) (1975)	-3.01
Way We Laughed, The (CosÃ¬ Ridevano) (1998)	-3.01
Tattoo (1981)	-3.01
I Love You Too (Ik ook Van Jou) (2001)	-3.01
Attack Force Z (a.k.a. The Z Men) (Z-tzu te kung tui) (1982)	-3.01
Mother Lode (1982)	-3.01
Time Walker (a.k.a. Being From Another Planet) (1982)	-3.01
Camera Obscura (2000)	-3.01
Hip Hop Witch, Da (2000)	-3.01
Hi-Line, The (1999)	-3.01

And now let us have a look how often (column n) they were rated in our training set.

best 10 movies:

```
train_set %>% dplyr::count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

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

title	b_i	n
Homage (1995)	1.49	1
Somebody is Waiting (1996)	1.49	4
Unforgotten: Twenty-Five Years After Willowbrook (1996)	1.49	1
Amati Girls, The (2000)	1.49	1
More (1998)	1.49	2
Strait-Jacket (1964)	1.49	1
Safe Conduct (Laissez-Passer) (2002)	1.49	1
House with Laughing Windows, The (La Casa dalle Finestre che Ridono) (1976)	1.49	1
War and Peace (Jang Aur Aman) (2001)	1.49	1
Tokyo Joe (1949)	1.49	1

worst 10 movies:

```
train_set %>% dplyr::count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

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

title	b_i	n
Autopsy (Macchie Solari) (1975)	-3.01	1
Way We Laughed, The (CosÃ¬ Ridevano) (1998)	-3.01	1
Tattoo (1981)	-3.01	1
I Love You Too (Ik ook Van Jou) (2001)	-3.01	1
Attack Force Z (a.k.a. The Z Men) (Z-tzu te kung tui) (1982)	-3.01	1
Mother Lode (1982)	-3.01	1
Time Walker (a.k.a. Being From Another Planet) (1982)	-3.01	1
Camera Obscura (2000)	-3.01	1
Hip Hop Witch, Da (2000)	-3.01	3
Hi-Line, The (1999)	-3.01	1

What we can see, is, that most of these are just rated once and those large errors effects an increase of our residual error. Because of this situation we follow the concept of regularization.

Regularization permits to penalize large estimates that com from small sample sizes. The general idea is to add a penalty for large values of b to sum of squares equitations that we minimize. What we are doing is just to divide each sum of b with lambda plus n (= sample size).

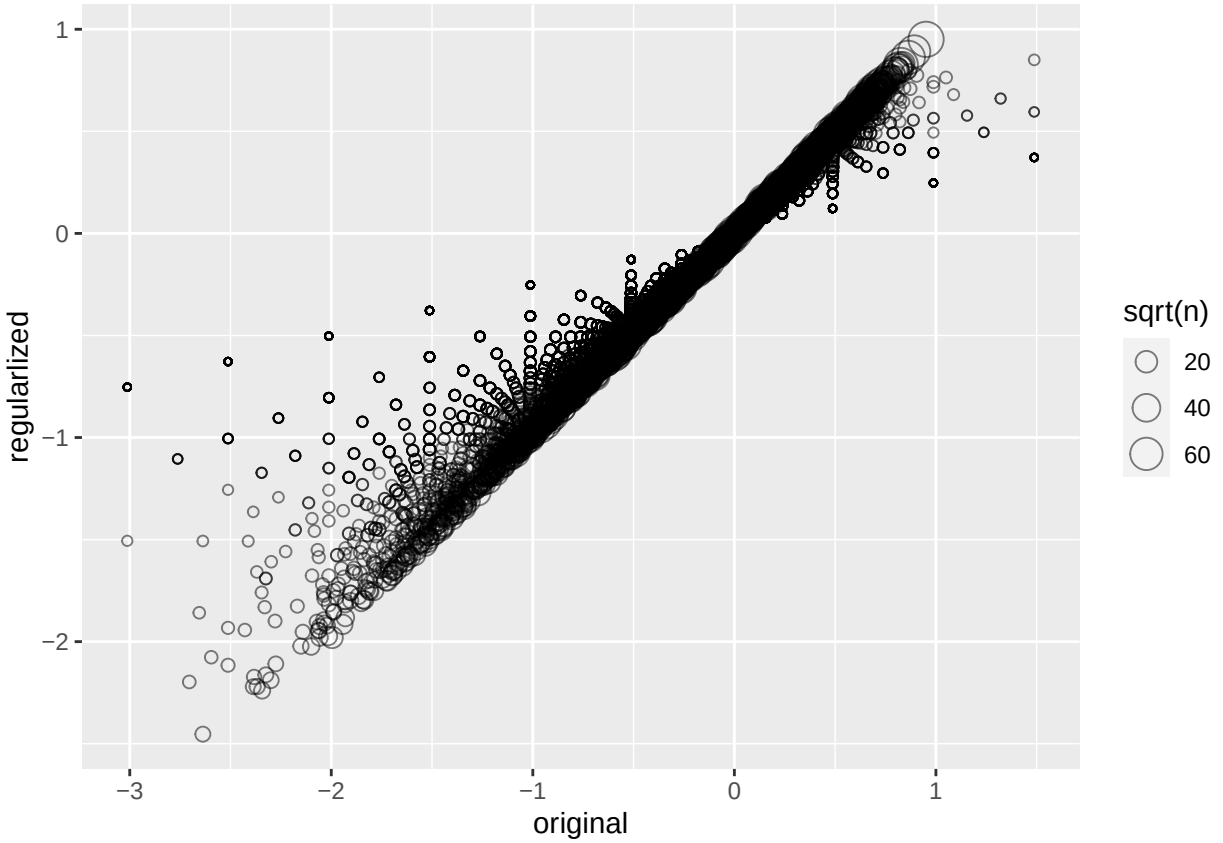
This approach will give us the desired effect. When n from i is very large, which give us a stable estimate, then lambda is effectively ignored because n from i plus lambda is about equal to n from i (e.g.: 1 (n) plus 3 (lambda) = 4 effects more than 100 (n) plus 3 (lambda) = 103).

However, when n from i is small, the estimates of b is shrunken towards zero. The larger lambda, the more we shrink. Lambda is a numeric value. Just to show this effect we will try lambda equals to 3 and do it only together with the movie effects. Later we are looking for an optimized value of lambda. First we compute b_i with lambda 3.

```
lambda <- 3
mu <- mean(train_set$rating)
movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
```

Then we plot these regularlized averages against the original averages from b_i. The size of the circle tells us how large n from i is.

```
data_frame(original = movie_avgs$b_i,
           regularlized = movie_reg_avgs$b_i,
           n = movie_reg_avgs$n_i) %>%
  ggplot(aes(original, regularlized, size=sqrt(n))) +
  geom_point(shape=1, alpha=0.5)
```



Here you can see that when n is small, the values are shrinking more towards zero.

And now we'll have a look at our top 10 movies based on the estimates we got when using regularization.

```
train_set %>%
     dplyr::count(movieId) %>%
     left_join(movie_reg_avgs) %>%
     left_join(movie_titles, by="movieId") %>%
     arrange(desc(b_i)) %>%
     select(title, b_i, n) %>%
     slice(1:10) %>%
     knitr::kable()
```

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

title	b_i	n
Shawshank Redemption, The (1994)	0.951	5583
Godfather, The (1972)	0.891	3517
Usual Suspects, The (1995)	0.862	4318
Somebody is Waiting (1996)	0.850	4
Third Man, The (1949)	0.833	599
Schindler's List (1993)	0.825	4707
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	0.821	641
General, The (1927)	0.820	220
Casablanca (1942)	0.817	2207
One Flew Over the Cuckoo's Nest (1975)	0.807	2561

Note that these are other ones. And we will also have a look for the 10 worst movies.

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

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

title	b_i	n
From Justin to Kelly (2003)	-2.45	40
Glitter (2001)	-2.24	65
Son of the Mask (2005)	-2.22	40
Gigli (2003)	-2.22	45
Carnosaur 3: Primal Species (1996)	-2.20	13
3 Ninjas: High Noon On Mega Mountain (1998)	-2.19	59
PokÃ©mon Heroes (2003)	-2.17	31
House of the Dead, The (2003)	-2.16	40
Carnosaur 2 (1995)	-2.12	16
Barney's Great Adventure (1998)	-2.11	38

And now we will do a prediction and will see the improvement. Remember the RMSE with movie effects has been 0.946.

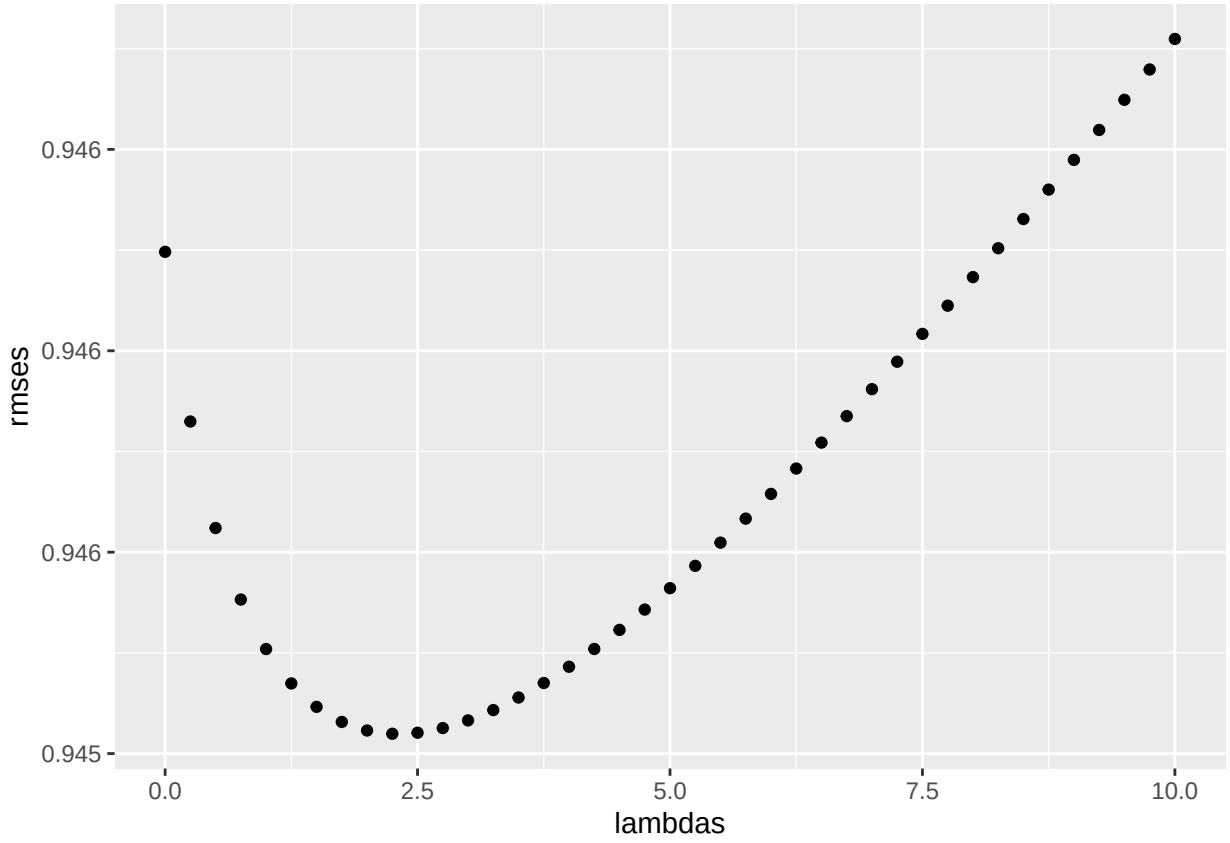
```
predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  .$pred

model_3_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie Effect Model",
                                     RMSE = model_3_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.060
Movie Effect Model	0.946
User/movie Effect Model	0.884
Regularized Movie Effect Model	0.945

So this provides a small improvement of 0.001 points. But anyway! Now we are looking for an optimized lambda because lambda is a tuning parameter. We are using cross validation to choose it. So we work with lambda starting from zero towards to 10 stepwise with 0.25 and make a plot with the different RMSE's regarding lambda.

```
lambdas <- seq(0, 10, 0.25)
mu <- mean(train_set$rating)
just_the_sum <- train_set %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())
rmses <- sapply(lambdas, function(l){
  predicted_ratings <- test_set %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
qplot(lambdas, rmses)
```



As a result we receive an optimal lambda of:

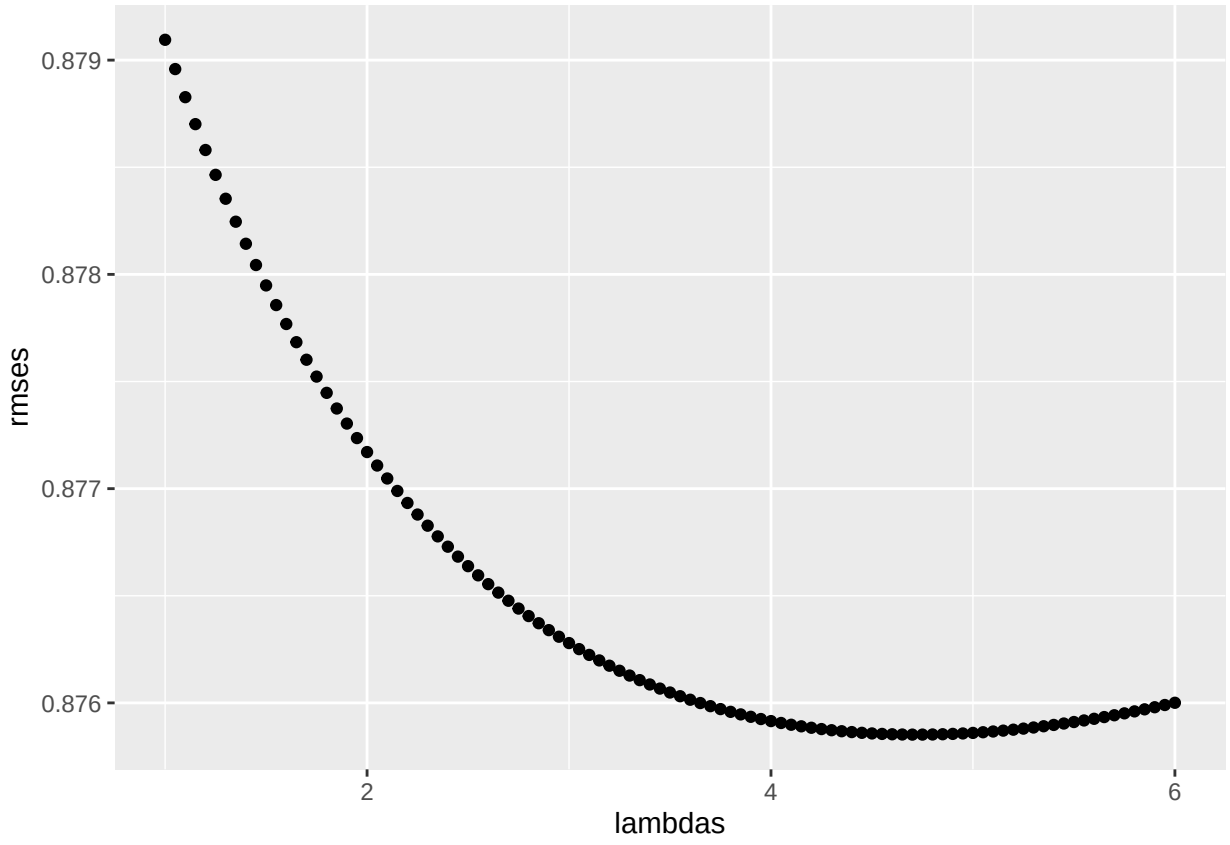
```
## [1] 2.25
```

We can also use regularization to estimate the user effect in addition to the movie effect and look for the optimal lambda. But now we take a range for lambda from 1 to 6 stepwise with 0.05.

```
lambdas <- seq(1, 6, 0.05)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
```

```
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})

qplot(lambdas, rmses)
```



As a result we receive an optimal lambda of:

```
## [1] 4.7
```

At the end we will do a prediction and will see the improvement. Remember the RMSE with movie and user effects has been 0.884.

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie + User Effect Model",
                                     RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.060
Movie Effect Model	0.946
User/movie Effect Model	0.884
Regularized Movie Effect Model	0.945
Regularized Movie + User Effect Model	0.876

As a result we got an improvement of 0.008 points.

Final model with datasets edx and validation

Now we will repeat a predcition based at the edx set versus the validation set using the optimal lambda from the training models and compute the final RMSE.

```
lambda <- lambdas[which.min(rmses)]

b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))
b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
predicted_ratings <-
  validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
final_rmse <-RMSE(predicted_ratings, validation$rating)


rmse_results <- bind_rows(rmse_results,
                          data_frame(method="final model",
                                     RMSE = final_rmse))
options(digits = 6)
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.060172
Movie Effect Model	0.945873
User/movie Effect Model	0.883886
Regularized Movie Effect Model	0.945291
Regularized Movie + User Effect Model	0.875852
final model	0.864821

After all we reached the goal with an RMSE of 0.864821. The perfomance of this methods seems to be quite good.

Conclusion

In our "challenge" we have just used two methods (linear model and regularization), but it was quite enough to reach the goal. Independent from that, there are other possibilities and methods to improve the result. One thing are methods and the other things are predictors. As additional predictors we could use genres and/or year of release (or age of movie).

The Netflix challenge winners implemented two general classes of models. One was similar to k-nearest neighbors, where you found movies that were similar to each other and users that were similar to each other. The other one was based on a approach called matrix factorization. In addition they also used the concept of regularization. And in general we can say, that there are more than one method as written before to do work on it and improve results.

On the other hand we have to say, that here we are just talking about to reach a goal. Not considering whats happens, if there is a new user or a new movie or a user which is never rating and so on.