Capstone Report

Simon Passek

2020-04-22

Contents

layout based on the package tint

1 Introduction

In one part of the edX course series "Data Science Professional Certificate" from Rafael Irizarry, the capstone task, named MovieLens Project, is to build a prediction algorithm for movie ratings. As a motivating example a recommendation challenge for the data science community is mentioned, in which Netflix offered one million dollars for the data science team, that would improves the movie recommendation system by 10 %. The Netflix data is not publicly available, so Professor Irizarry provides links to the GroupLens research lab database.

The dataset downloaded from there, comprises of 10000054 observations with 6 recorded features.

Table 1: A subset of the data with recorded features

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	231	5	838983392	Dumb & Dumber (1994)	Comedy
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

Each row represents a user rating a certain movie with values between 0 (the

worst possible rating) and 5 (the best possible rating).

Table 2: frequency distribution of ratings

edx$rating	n	percent
0.5	85374	0.0094859
1.0	345679	0.0384085
1.5	106426	0.0118250
2.0	711422	0.0790464
2.5	333010	0.0370009
3.0	2121240	0.2356919
3.5	791624	0.0879577
4.0	2588430	0.2876016
4.5	526736	0.0585259
5.0	1390114	0.1544562

In the dataset used in the MovieLens project 69878 distinct users rate 10677 distinct movies.

Table 3: distinct users and movies

n_distinct_users	n_distinct_movies
69878	10677

If every user rates every movie our dataset should comprise of 746087406 rows. But the actual data set has only 10000054 rows. This implies that not every user rated every movie. In the MovieLens Project one could implement information of a specific user, movie and related movies to predict the rating of that specific user on a new, unknown movie.

2 Analysis

2.1 Exploratory data analysis

In order to only use the train dataset for exploration/training of the prediction algorithm we will look at a data partition of the MovieLens data. This subset is termed edx and comprises of 9000055 observations.

```
edx %>% group_by(userId) %>%
  summarize(n = n()) %>%
  ggplot(aes(n),col = "black") +
```

```
  geom_histogram(color = "black") +
  xlim(0, 1000)+scale_x_log10()+
  labs(x = "number of ratings",
       y = "count of distinct users")
```

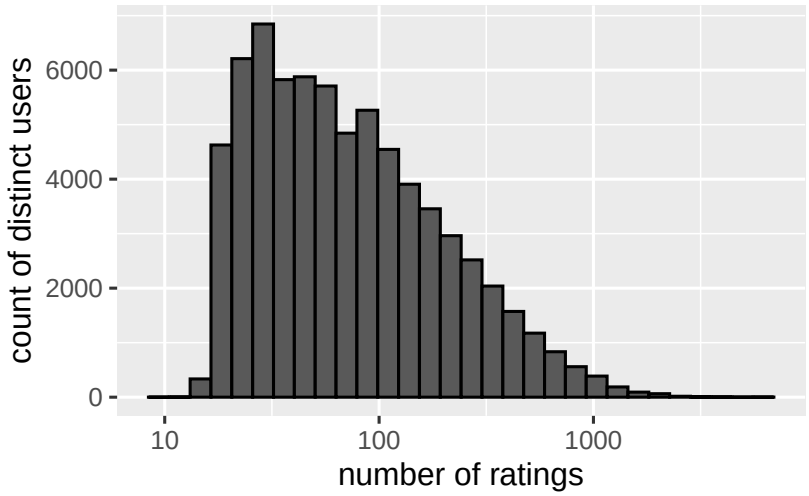


Figure 1: number of ratings per user

Some users seem to rate every movie they watch, whereas others rate more rarely.

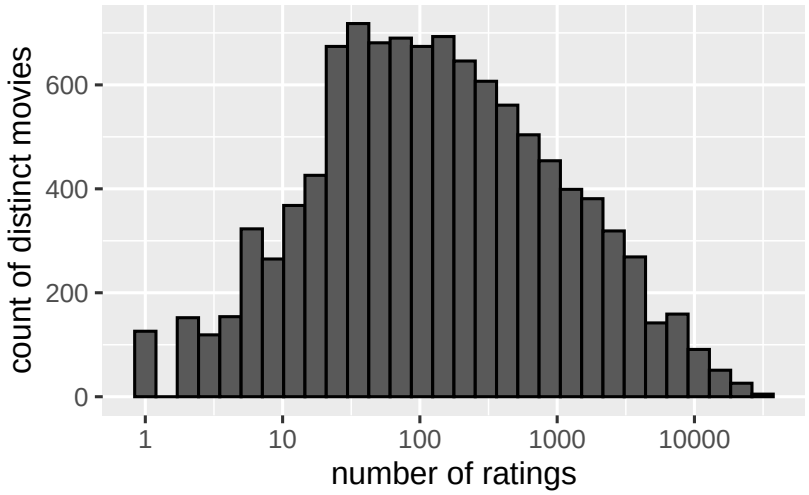


Figure 2: number of ratings per movie

The same trend can be seen in the number of ratings per movie. Some movies

seem to be pretty popular, whereas others are only rated by few users.

```
edx %>%
  group_by(movieId) %>%
  summarize(n_movies = n(), mean_rating = mean(rating)) %>%
  arrange(n_movies) %>%
  mutate(cut_movies = cut(n_movies,
                          breaks = quantile(
                            n_movies,
                            probs = seq(0, 1, 0.1)))) %>%
  filter(!is.na(cut_movies)) %>%
  ggplot(aes(mean_rating, fill = cut_movies))+
  geom_histogram()+scale_fill_discrete(
    name = "number of movie observations")
```

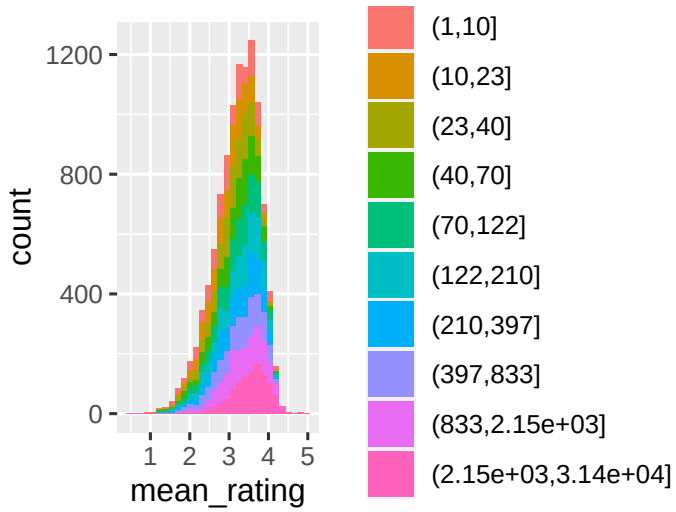


Figure 3: mean rating of individual movies

No real trend is observed. Independent of users or number of ratings, some movies tend to have higher mean_ratings than others. In the following analysis this observation is called “movie-effects”.

```
edx %>%
  group_by(userId) %>%
  summarize(n_user = n(), mean_rating = mean(rating)) %>%
  mutate(cut_users = cut(n_user,
                         breaks = quantile(
```

```
                          n_user,
                          probs = seq(0, 1, 0.1)))) %>%
  filter(!is.na(cut_users)) %>%
  ggplot(aes(mean_rating, fill = cut_users)) +
  geom_histogram()+
  labs(x = "mean_rating of users")+
  scale_fill_discrete("number of ratings per user")
```

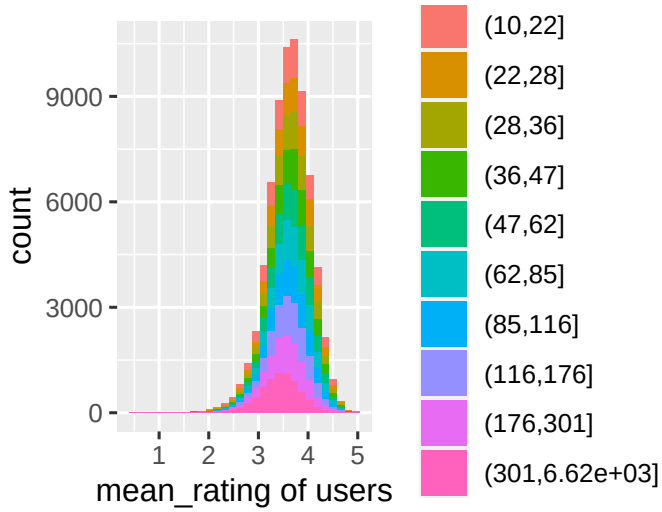


Figure 4: mean rating of individual users 1

Some users seem to give better mean_ratings then other users, independent of the number of ratings per user and the acutal movie. In the following analysis this observation is called “user-effects”.

```
edx %>%
  group_by(genres) %>%
  summarize(n_genres = n(), mean_rating = mean(rating)) %>%
  filter(n_genres > 50000) %>%
  ggplot(aes(x = reorder(genres, mean_rating),
             y = mean_rating))+
  geom_point()+
  coord_flip()
```

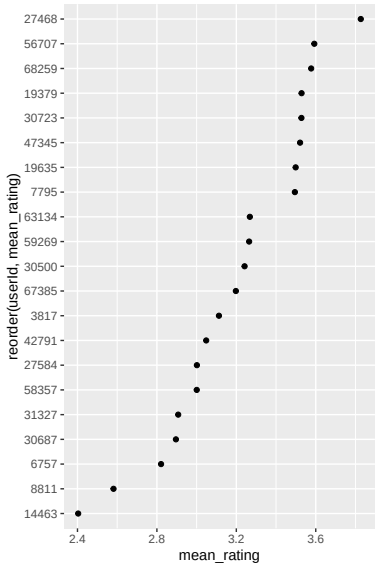


Figure 5: mean rating of individual users 2

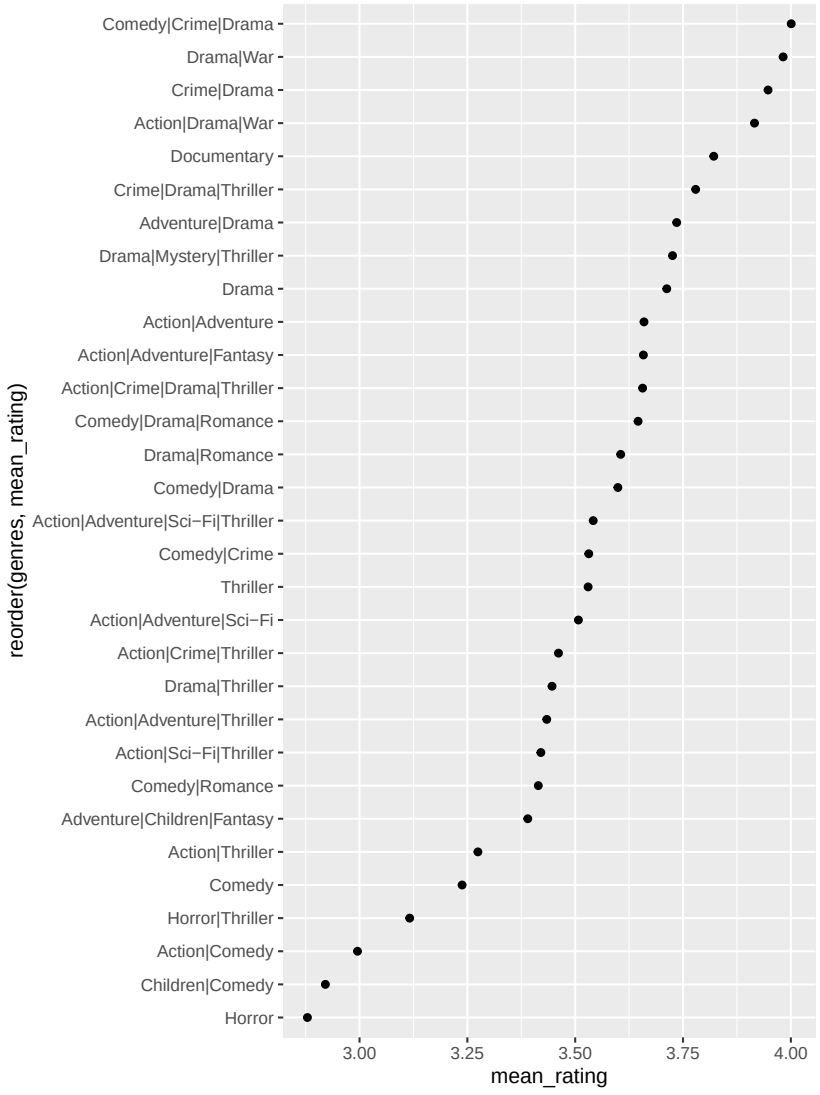


Figure 6: mean rating per genre 1

Independent of users, different genres have different mean_ratings. In the following analysis this observation is termed “genres-effects”.

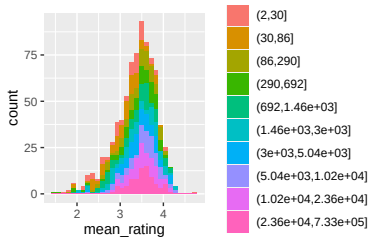


Figure 7: mean rating per genre 2

```
edx %>%
  mutate(date_time = lubridate::as_datetime(timestamp)) %>%
  mutate(week =
           lubridate::round_date(date_time,
                                 unit = "week")) %>%
  group_by(week) %>%
  summarize(n_week = n(), mean_rating = mean(rating)) %>%
  ggplot(aes(week, mean_rating))+
  geom_point()+
  geom_smooth(color = "darkred")
```

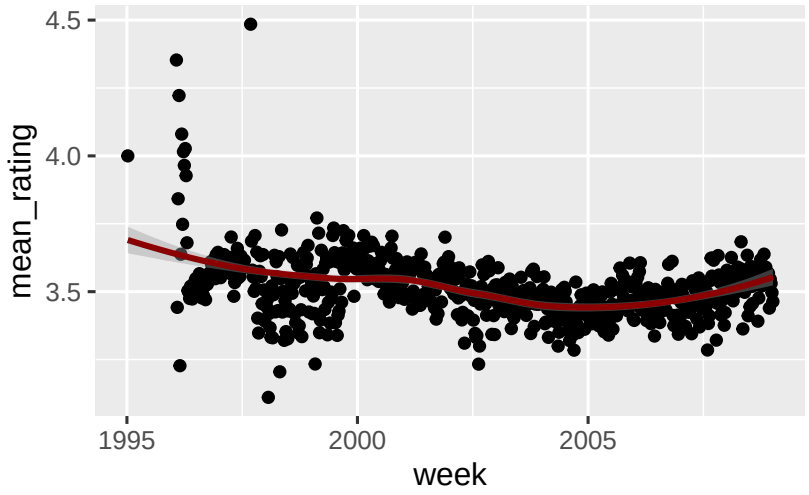


Figure 8: time dependency of movie ratings

The mean_rating across movies, genres, users also seems to be time dependent. In the following analysis this observation is termed “week-effects”.

2.2 Modelling approaches

In order to not overtrain or overfit a model, the train datapartition edx also has to be split in train- and test-set. To build the algorithm only the test set is going to be used.

```
# create train and test data set
set.seed(1, sample.kind = "Rounding")

test_index <- createDataPartition(edx$rating, times = 1, p = 0.2, list = FALSE)

train_set_incomplete <- edx[-test_index,]
test_set_incomplete <- edx[test_index,]

# to make sure to have the same movieIds and userIds in train- and test_set
test_set <- test_set_incomplete %>%
  semi_join(train_set_incomplete, by = "movieId") %>%
  semi_join(train_set_incomplete, by = "userId")

#row_bind the dropped rows to the train_set
removed <- anti_join(test_set_incomplete, test_set)
train_set <- rbind(train_set_incomplete, removed)
saveRDS(train_set, "train_set")
saveRDS(test_set, "test_set")
```

```
# load data set
train_set <- readRDS("train_set")
test_set <- readRDS("test_set")

# using library(lubridate) to transform the timestamp into date format

train_set <- train_set %>%
  mutate(date = lubridate::as_datetime(timestamp))
train_set  <- train_set %>%
  mutate(week = lubridate::round_date(date, "week"))

test_set <- test_set %>%
  mutate(date = lubridate::as_datetime(timestamp))
test_set <- test_set %>%
  mutate(week = lubridate::round_date(date, "week"))
```

To evaluate the performance of different models, a loss function is needed. In the MovieLens project, as in the Netflix challenge, the residual mean squared error (RMSE) is used.

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

Taking together the four effects(user-effects(u), movie-effects(i), genres-effects(g), week-effects(w)) observed in the exploratory data analysis, the first

model could look like this:

$$Y_{u,i,g,w} = \mu + b_u + b_i + b_g + b_w + \varepsilon_{u,i,g,w}$$

with the bs as "effects" and $\varepsilon_{u,i,g,w}$ as independend errors by random variation.
As descriped in the corresponding chapter of Rafael Irizarry´s book "Introduction to Data Science" one can use the least squares to estimate the effects b_u, b_i, b_g, b_w:

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

movie_effects_reg <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

user_effects_reg <- train_set %>%
  left_join(movie_effects_reg, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

genre_effects_reg <- train_set %>%
  left_join(movie_effects_reg, by = "movieId") %>%
  left_join(user_effects_reg, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating- mu - b_i - b_u))

week_effects_reg <- train_set %>%
  left_join(movie_effects_reg, by = "movieId") %>%
  left_join(user_effects_reg, by = "userId") %>%
  left_join(genre_effects_reg, by = "genres") %>%
  group_by(week) %>%
  summarize(b_w = mean(rating - mu - b_i - b_u - b_g))

prediction <- test_set %>%
  left_join(movie_effects_reg, by = "movieId") %>%
  left_join(user_effects_reg, by = "userId") %>%
  left_join(genre_effects_reg, by = "genres") %>%
  left_join(week_effects_reg, by = "week") %>%
  mutate(pred = mu+b_i+b_u+b_g+b_w) %>%
  pull(pred)


rmse <- caret::RMSE(test_set$rating, prediction)
```

```
rmse
```

```
## [1] 0.8654875
```

Let´s explore the largest mistakes of the prediction:

```
errors <- test_set %>%
    left_join(movie_effects_reg, by = "movieId") %>%
    left_join(user_effects_reg, by = "userId") %>%
    left_join(genre_effects_reg, by = "genres") %>%
    left_join(week_effects_reg, by = "week") %>%
    mutate(residual = rating - (mu+b_i+b_u+b_g+b_w),
           prediction = mu+b_i+b_u+b_g+b_w) %>%
  arrange(desc(abs(residual)))

errors[1:5, c("movieId",  "residual", "rating")] %>%
  knitr::kable(caption =
                 "the 5 largest residuals in the training set")
```

Table 4: the 5 largest residuals in the training set

movieId	residual	rating
4973	-5.045448	0.5
737	5.000694	5.0
193	4.869428	5.0
737	4.796328	5.0
904	-4.751406	0.5

The largest mistakes seem to be made when predicting rather obscure movies, with very large or very small ratings.

Lets look at the training data, based on which the predictions were made:

```
# subset errors
errors_subset <- errors %>%
  group_by(movieId) %>%
  summarize(mean_residual = mean(abs(residual))) %>%
  arrange(desc(abs(mean_residual)))


# look at the number of observations with the biggest prediction errors
```

```
train_set %>%
  right_join(errors_subset, by = "movieId") %>%
  group_by(movieId) %>%
  summarize(n_movie = n(), mean_residual = mean(abs(mean_residual)),
            mean_rating = mean(rating)) %>%
  arrange(desc(abs(mean_residual))) %>%
  slice(1:10) %>%
  knitr::kable(caption = "the 10 worst predictions based on the training data")
```

Table 5: the 10 worst predictions based on the training data

movieId	n_movie	mean_residual	mean_rating
59655	2	3.656439	3.750000
8904	3	3.324838	1.666667
62390	3	3.306294	3.500000
8876	10	3.205870	3.400000
6420	5	3.150402	3.400000
6026	6	2.956402	3.666667
26059	1	2.930300	1.500000
8519	4	2.823403	3.250000
5831	3	2.778778	1.500000
6766	2	2.689278	1.000000

The biggest absolute errors are observed, when predicting ratings with just a view observed movies. Even with the model taking into account all four observed effects, the residuals are largest in the movies with a low number of observations.

```
train_set %>%
  right_join(errors_subset, by = "movieId") %>%
  group_by(movieId) %>%
  summarize(n_movie = n(), mean_residual = mean(abs(mean_residual)),
            mean_rating = mean(rating)) %>%
  arrange(desc(abs(mean_residual))) %>%
  slice(1:100) %>%
  summarize(mean_n = mean(n_movie), mean_mean_residual = mean(abs(mean_residual)))
```

mean_n	mean_mean_residual
8.2	2.070898

The biggest 100 mean_errors stratified by movieId have an average number of observations in the train_set of 8.2.

When looking at the numbers of ratings from each recoreded user, a similar observation can be made.

```
errors_subset <- errors %>%
  group_by(userId) %>%
  summarize(mean_residual = mean(abs(residual))) %>%
  arrange(desc(abs(mean_residual)))


train_set %>%
  right_join(errors_subset, by = "userId") %>%
  group_by(userId) %>%
  summarize(n_user = n(),
            mean_residual = mean(mean_residual),
            mean_rating = mean(rating)) %>%
  arrange(desc(abs(mean_residual))) %>%
  slice(1:100) %>%
  summarize(mean_n = mean(n_user),
            mean_mean_residual = mean(abs(mean_residual)))
```

mean_n	mean_mean_residual
25.38	2.348835

The largest 100 mean_errors stratified by userId have an average number of observations in the train_set of 25.

With only a view users rating a movie in our train_set sample, we have much uncertainty about the true movie rating in the population. The idea of **regularization** is to penalize the least squares estimates, which have few numbers of observations. To formulate this mathematically for a penalized effect b_i:

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

where λ is the penalty term and n_i is the number of observations. If n_i is large, λ has almost no impact on $\hat{b}_i(\lambda)$. But if n_i is small, the penalty induced by λ is large.[1]

[1] Introduction to Data Science by Rafael Irizarry

2.3 Regularization approach

The λ can be seen as a typical machine learning tuning parameter. To select the best tuning parameter only the train_set data should be used. Therefore 5 bootstrap cross validations are used to select the optimal value for λ.

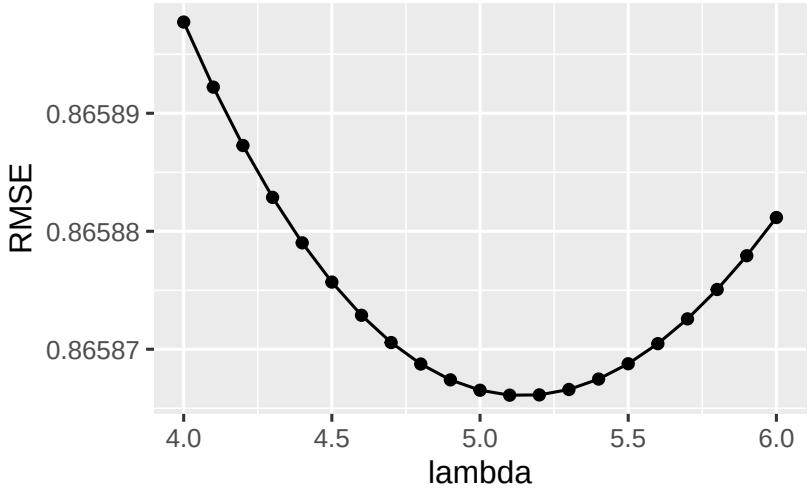
```
lambdas = seq(4, 6, 0.1)

set.seed(1, sample.kind = "Rounding")

B <- 5 # number of bootstrap iterations

rmse_bootstrapping <- replicate(B, {

  #create bootstrap sample without using test_set
  boots_index <- sample(1:nrow(train_set),replace = TRUE, size = 1440018)
  train_set_boots <- train_set[-boots_index,]
  test_sub <- train_set[boots_index,] # 20 % of the test_set as bootstrap validation set

  # make sure users and movies of train set are in the test set
  test_set_boots <- test_sub %>%
    semi_join(train_set_boots, by = "movieId") %>%
    semi_join(train_set_boots, by = "userId")

  lost <- anti_join(test_sub, test_set_boots)

  train_set_boots <- rbind(train_set_boots, lost)


  rmse <- map_df(lambdas, function(x){

    mu <- mean(train_set_boots$rating)

   # movie effects with regularization
    movie_effects_reg <- train_set_boots %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating - mu)/(n()+x))

   # user effects with regularization
    user_effects_reg <- train_set_boots %>%
      left_join(movie_effects_reg, by = "movieId") %>%
      group_by(userId) %>%
      summarize(b_u = sum(rating - mu - b_i)/(n()+x))
```

```
   # genres effects with regularization
    genres_effects_reg <- train_set_boots %>%
      left_join(movie_effects_reg, by = "movieId") %>%
      left_join(user_effects_reg, by = "userId") %>%
      group_by(genres) %>%
      summarize(b_g = sum(rating - mu - b_i - b_u)/(n()+x))

   # week effects with regularization
    week_effects_reg <- train_set_boots %>%
      left_join(movie_effects_reg, by = "movieId") %>%
      left_join(user_effects_reg, by = "userId") %>%
      left_join(genres_effects_reg, by = "genres") %>%
      group_by(week) %>%
      summarize(b_w = sum(rating - mu - b_i - b_u - b_g)/(n()+x))

   # making the prediction on the test set
    prediction <- test_set_boots %>%
      left_join(movie_effects_reg, by = "movieId") %>%
      left_join(user_effects_reg, by = "userId") %>%
      left_join(genres_effects_reg, by = "genres") %>%
      left_join(week_effects_reg, by = "week") %>%
      mutate(pred = mu + b_i + b_u + b_g + b_w) %>%
      pull(pred)

    # calculating RMSE
    rmse <- RMSE(prediction[which(!is.na(prediction))],
                 test_set_boots$rating[which(!is.na(prediction))])

    tibble(rmse = rmse, lambda = x)
  })

  rmse
})


# tibble with results
rmse_bootstrapping_df <- bind_cols(lapply(rmse_bootstrapping, function(x)as_tibble(x)))

# defining colnames of tibble
colnames(rmse_bootstrapping_df) <- sapply(seq_len(B), function(x){
  c(paste("RMSE", x, sep = "_"), paste("lambda", x, sep = "_"))
})
```

```r
# gather tibble

rmse_bootstrapping_df_long <- rmse_bootstrapping_df %>%
  pivot_longer(everything(),
               names_to = c(".value", "bootstrap_iteration"),
               names_pattern = "(.*)_(.*)" )

# filtering for best lambdas in all bootstrap iterations

rmse_bootstrapping_df_long_best <- rmse_bootstrapping_df_long  %>%
  group_by(bootstrap_iteration) %>%
  filter(RMSE == min(RMSE)) %>%
  arrange(RMSE)

rmse_bootstrapping_df_long_best %>%
  knitr::kable(caption = "best lambdas per bootstrap iteration")
```

Table 8: best lambdas per bootstrap iteration

bootstrap_iteration	RMSE	lambda
4	0.8649495	4.9
1	0.8650271	5.3
3	0.8658661	5.1
5	0.8663770	5.1
2	0.8665013	4.9



3 Results

As computed above, the optimal value for λ seems to be 5.1. Choosing this value, a model is trained using all the edX data to predict on the validation data.

```
validation <- validation %>% mutate(
  date = lubridate::as_datetime(timestamp)) %>%
  mutate(week = lubridate::round_date(
    date, unit = "week"))

edx <- edx %>% mutate(
  date = lubridate::as_datetime(timestamp)) %>%
  mutate(week = lubridate::round_date(
    date, unit = "week"))

# defining train_set and test_set as edx and validation

train_set <- edx
test_set <- validation

lambdas <- 5.1

rmse_lambdas_all <- map_df(lambdas, function(x){

  mu <- mean(train_set$rating)

  movie_effects_reg <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+x))

  user_effects_reg <- train_set %>%
    left_join(movie_effects_reg, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n()+x))

  genre_effects_reg <- train_set %>%
    left_join(movie_effects_reg, by = "movieId") %>%
    left_join(user_effects_reg, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating- mu - b_i - b_u)/(n()+x))

  week_effects_reg <- train_set %>%
    left_join(movie_effects_reg, by = "movieId") %>%
```

```
    left_join(user_effects_reg, by = "userId") %>%
    left_join(genre_effects_reg, by = "genres") %>%
    group_by(week) %>%
    summarize(b_w = sum(rating - mu - b_i - b_u - b_g)/(n()+x))

  prediction <- test_set %>%
    left_join(movie_effects_reg, by = "movieId") %>%
    left_join(user_effects_reg, by = "userId") %>%
    left_join(genre_effects_reg, by = "genres") %>%
    left_join(week_effects_reg, by = "week") %>%
    mutate(pred = mu+b_i+b_u+b_g+b_w) %>%
    pull(pred)


  rmse <- RMSE(test_set$rating, prediction)

  tibble(rmse)
})


rmse_lambdas_all_view <- rmse_lambdas_all %>%
  mutate(lambdas) %>% arrange(rmse)
rmse_lambdas_all_view %>%
  knitr::kable(caption = "RMSE of the final model")
```

Table 9: RMSE of the final model

rmse	lambdas
0.8643053	5.1

4 Conclusion

The model reaching an RMSE of 0.8643053 is still a rather simple model but it fulfills the expectation of the edX capstone.

By taking into account the four effects(user-effects(u), movie-effects(i), genres-effects(g), week-effects(w)) observed in the exploratory data analysis, the prediction can be improved compared to a model, that only takes into account the mean of each movie. The usage of regularization leads to a further, significant improvement of the prediciton, by penalizing the influence of small number of observations.
In order to select the tuning parameter λ without danger of overfitting, the

training dataset edx is again partitioned into a test- and train-set. A five fold bootstrap cross validation is used to select the best lambda.

A variety of different approaches, extensions and their combination in machine learning ensembles are possible to further improve the models performance. For example one could use factor analysis, principal component analysis or singular value decomposition to improve prediction accuracy of the “genres-effects” or the “movie-effects”. For recommendation challenges specific packages and algorithms were developed. For example the recommenderlab package on CRAN.