MovieLens

Magdalena Stefanowicz

2020-08-10

Introduction

This project was done for a course: HarvardX: PH125.9x Data Science: Capstone in HarvardX Professional Certificate Program in Data Science.

The goal of the project is to create a recommendation system by training a machine learning algorithm using the inputs in one subset to predict movie ratings in validation set. The recommendation system is made for dataset MovieLens 10M which contains 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users.

The following key steps are performed in the study: * edx set and validation set are created * edx set is further divided into train and test set to avoid overtraining * dataset is analyzed in order to find the most appropriate model * movie ratings are predicted using models with movie effect, user effect and regularization * the typical error loss (RMSE) is used to evaluate RMSE for different approaches * RMSE is used to evaluate how close predictions of the final model are to the true values.

Is it aimed to create a recommendation model with RMSE < 0.86490.

Analysis

Data setup

First, let's install needed packages and download MovieLens 10M dataset from the grouplens.org.

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse

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

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.1
## v tidyr   1.1.1     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

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

```r
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

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

```r
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

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

```r
if(!require(lubridate)) install.packages("data.table", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: lubridate

##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```r
library(tidyverse)
library(caret)
library(data.table)
library(lubridate)

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

Let's create edx set and validation set.

```r
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")
```

Let's create a validation set which corresponds to 10% of MovieLens data.

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

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

Let's 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")
```

Let's add rows removed from validation set back into edx set.

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

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```r
edx <- rbind(edx, removed)

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

Let's create an additional partition of train and test sets from the provided edx dataset to experiment with multiple parameters.

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

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
edx_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train <- edx[-edx_index,]
test <- edx[edx_index,]

test <- test %>%
  semi_join(train, by = "movieId") %>%
  semi_join(train, by = "userId")
```

Data analyzis

Edx dataset has 9 millon rows which corresponds to 90% of the total Movielens 10M dataset. Each row is a rating made by one user for a movie. Data

```r
edx %>% as_tibble() %>% head()
```

```
## # A tibble: 6 x 6
##   userId movieId rating timestamp title                   genres
##    <int>   <dbl>  <dbl>     <int> <chr>                   <chr>
## 1      1     122      5 838985046 Boomerang (1992)        Comedy|Romance
## 2      1     185      5 838983525 Net, The (1995)         Action|Crime|Thriller
## 3      1     292      5 838983421 Outbreak (1995)         Action|Drama|Sci-Fi|T~
## 4      1     316      5 838983392 Stargate (1994)         Action|Adventure|Sci-~
## 5      1     329      5 838983392 Star Trek: Generations~ Action|Adventure|Dram~
## 6      1     355      5 838984474 Flintstones, The (1994) Children|Comedy|Fanta~
```

```r
summary(edx)
```

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

Let's analyze user and movie dimensions. The number of unique users and unique movies in the edx dataset are:

```r
edx %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

If every user rated every movie we would have ca. 700M rows but we only have 10M which means that users rate only some movies.

The average number of rating per movie and the average number of rating per user are:

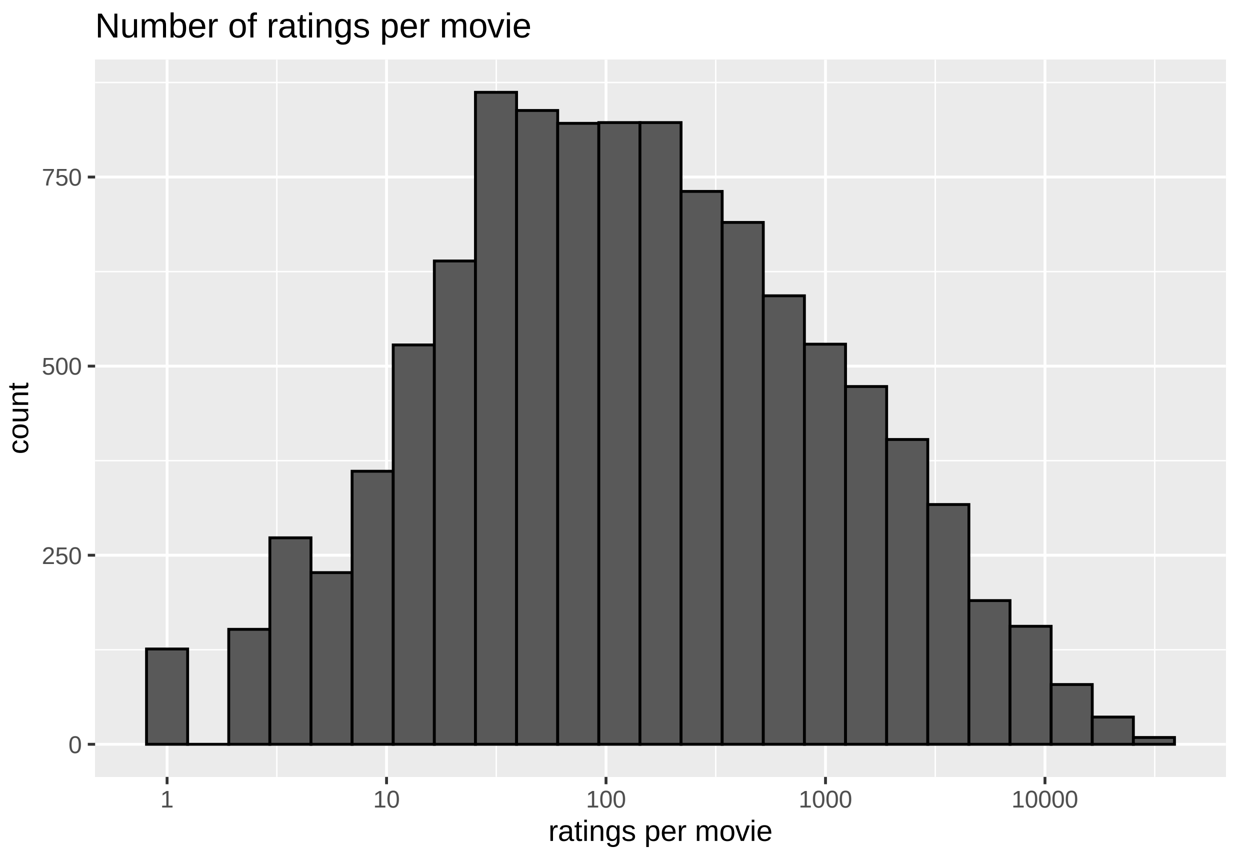
```r
edx %>%
  summarize(avg_no_rating_per_movie = round(nrow(edx)/n_distinct(movieId)), avg_no_rating_per_user = rou
```

```
##   avg_no_rating_per_movie avg_no_rating_per_user
## 1                     843                    129
```

However some users rate more movies then others and some movies get rated more than others. There are also a few movies which were rated only 1 time. Movies with both obscure ratings and very low number of ratings might effect the prediction.

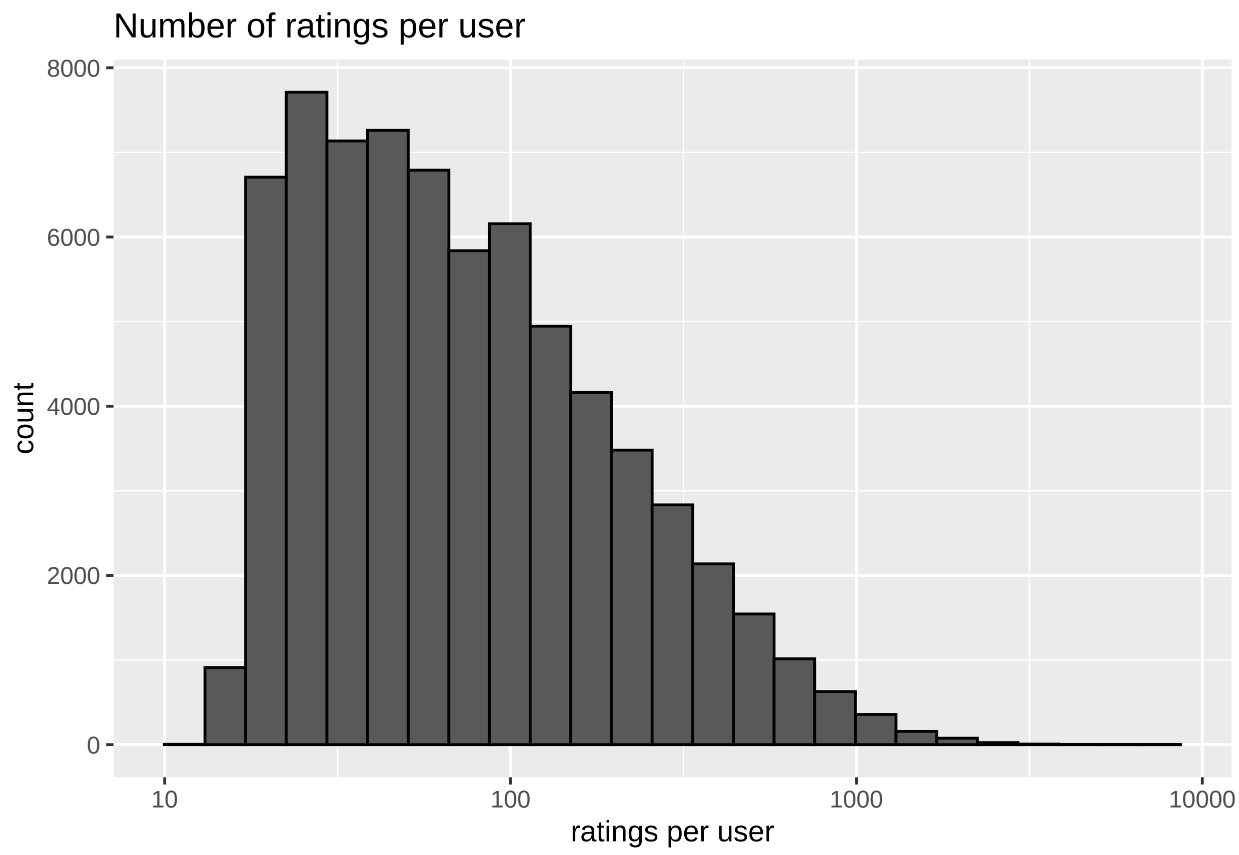
```r
  par(mfrow = c(1,2))
  edx %>%
    group_by(movieId) %>%
    summarize(count = n()) %>%
    ggplot(aes(count))+
    geom_histogram(bins = 25, color = "black")+
    scale_x_log10()+
    labs(x = "ratings per movie", y = "count", title = "Number of ratings per movie")
```

```
## 'summarise()' ungrouping output (override with '.groups' argument)
```



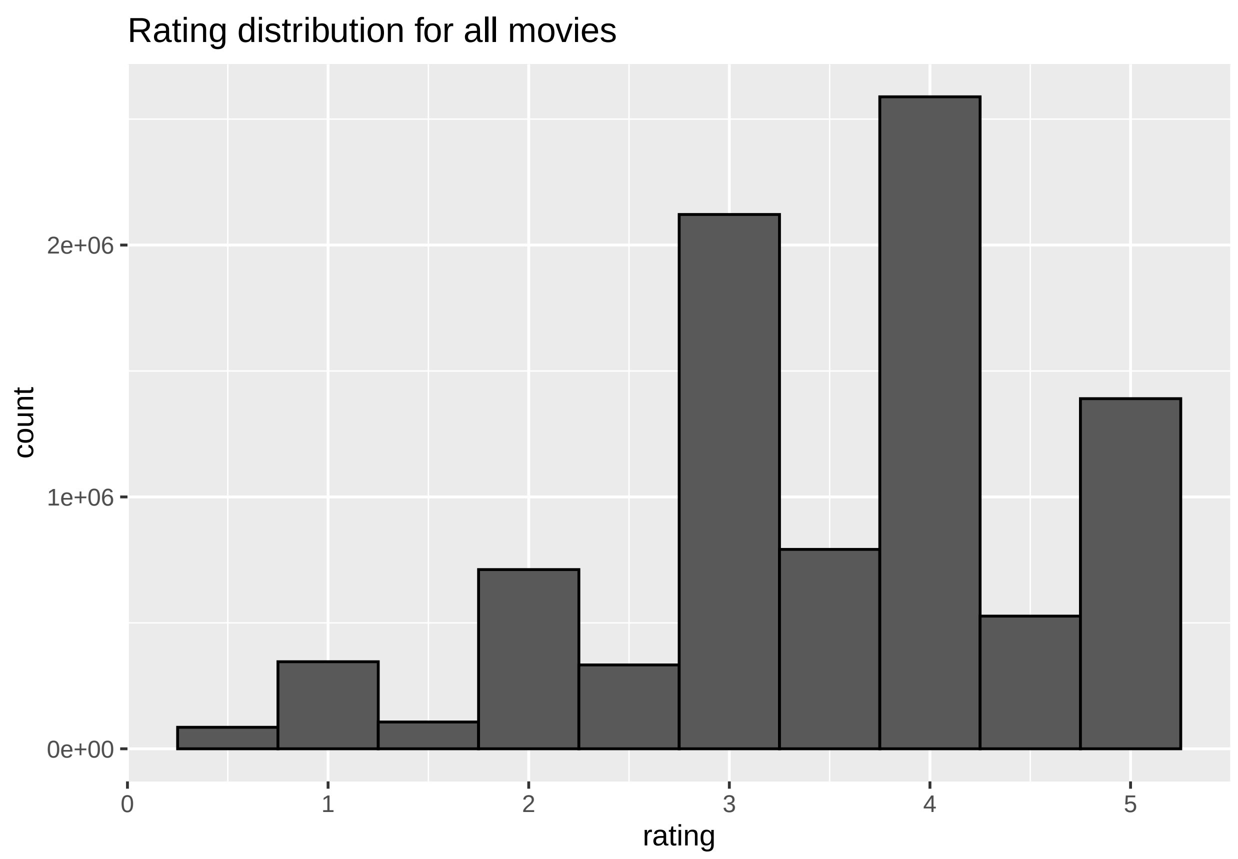
```
edx %>%
  group_by(userId) %>%
  summarize(count = n()) %>%
  ggplot(aes(count))+
  geom_histogram(bins = 25, color = "black")+
  scale_x_log10()+
  labs(x = "ratings per user", y = "count", title = "Number of ratings per user")
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```



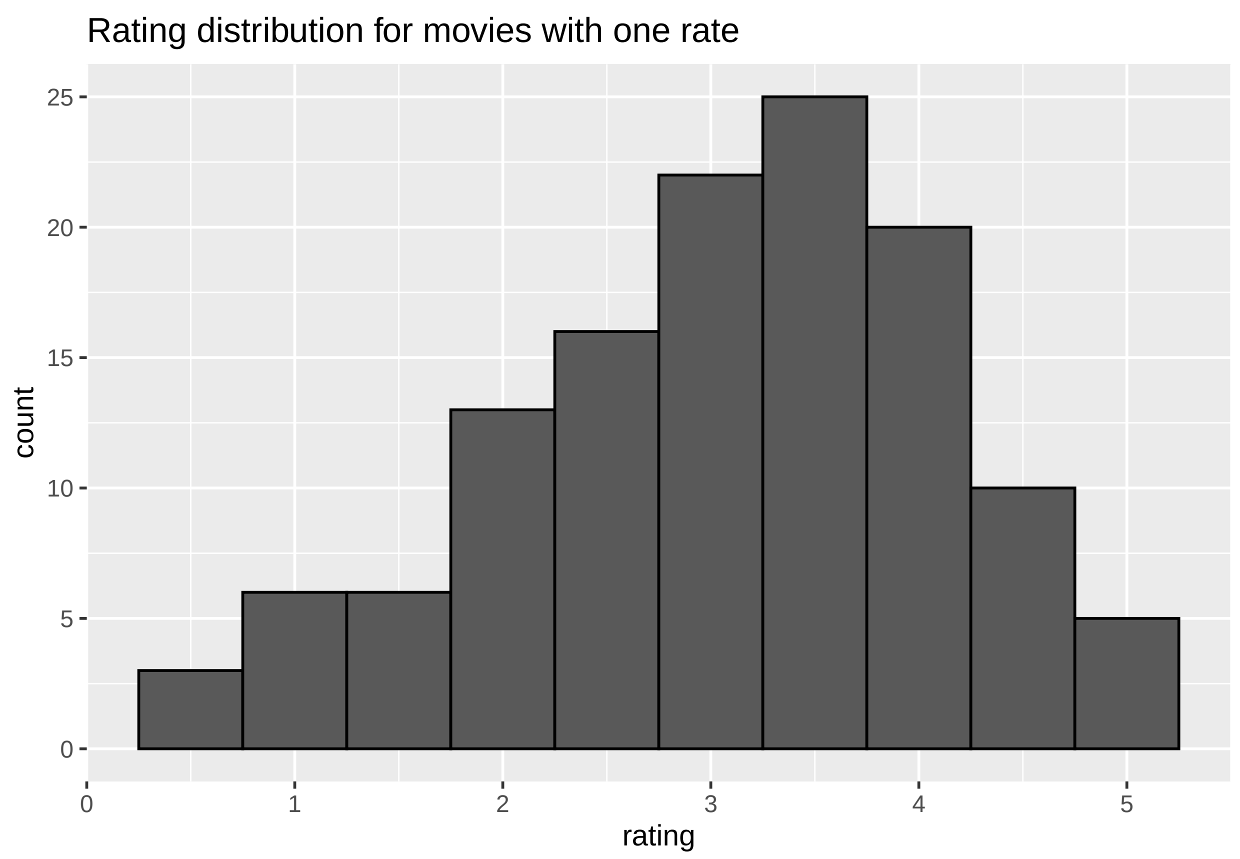
The next two plots show rating distribution for all movies and movies rated only once, correspondingly. Looking at the results for all movies, we can see that most movies have higher ratings (above 2.5). There is also a strong preference to rate movies with full scores rather then half scores. Distribution of ratings for movies with only one rating differs significantly from rating distribution for all movies. Movies with one rating are more likely to receive extremely low ratings and to receive half scores. It will be difficult to predict future rating for these movies.

```
par(mfrow = c(1,2))
edx %>%
  ggplot(aes(rating)) +
  geom_histogram(bins = 10, color = "black") +
  ggtitle("Rating distribution for all movies")
```



```
edx %>%
  group_by(movieId) %>%
  summarize(count = n()) %>%
  filter(count == 1) %>%
  left_join(edx, by = "movieId") %>%
  ggplot(aes(rating)) +
  geom_histogram(bins = 10, color = "black") +
  ggtitle("Rating distribution for movies with one rate")
```

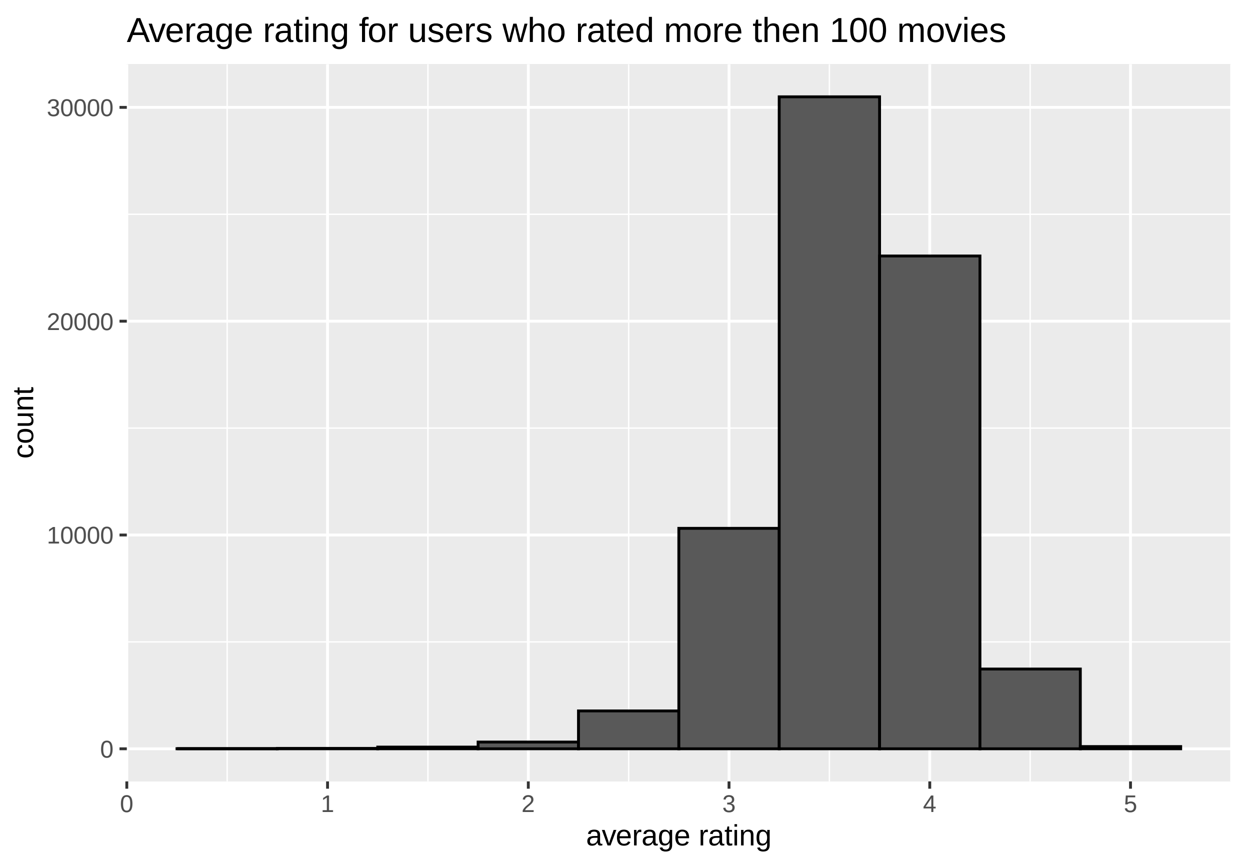
```
## `summarise()` ungrouping output (override with `.groups` argument)
```



The next plot shows the average rating for users who have rated over 100 movies. User who rate many movies tend to give non-extreme ratings. Also they are more likely to rate higher then lower (above 2.5).

```r
edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 10, color = "black")+
  labs(x = "average rating", title = "Average rating for users who rated more then 100 movies" )
```

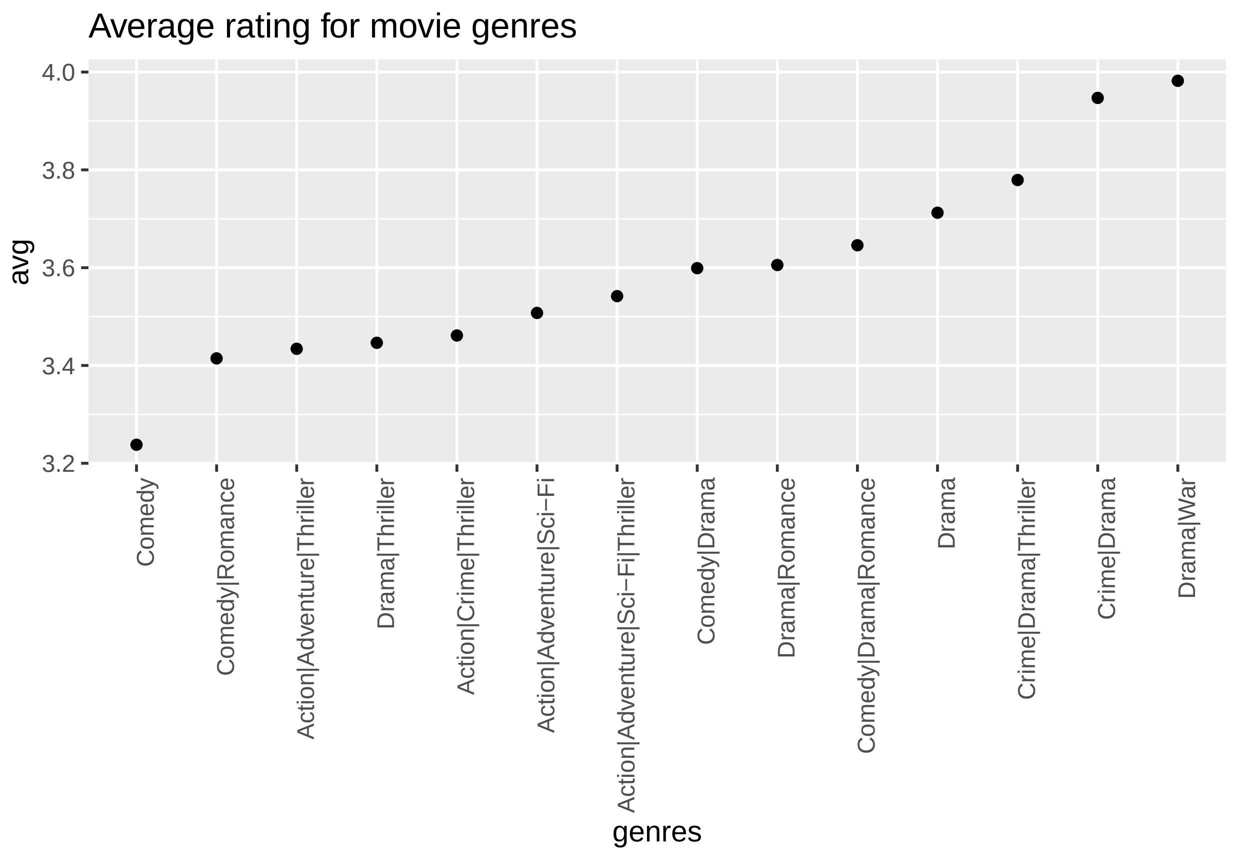
```
## `summarise()` ungrouping output (override with `.groups` argument)
```



Let's analyze movie ratings by genres. Comedy has the lowest average rating and Drama/War has the highest average rating among all the genres that received more than 100,000 ratings.

```
edx  %>%
  group_by(genres) %>%
    summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
    filter(n >= 100000) %>%
    mutate(genres = reorder(genres, avg)) %>%
    ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
    geom_point() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  ggtitle("Average rating for movie genres")
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

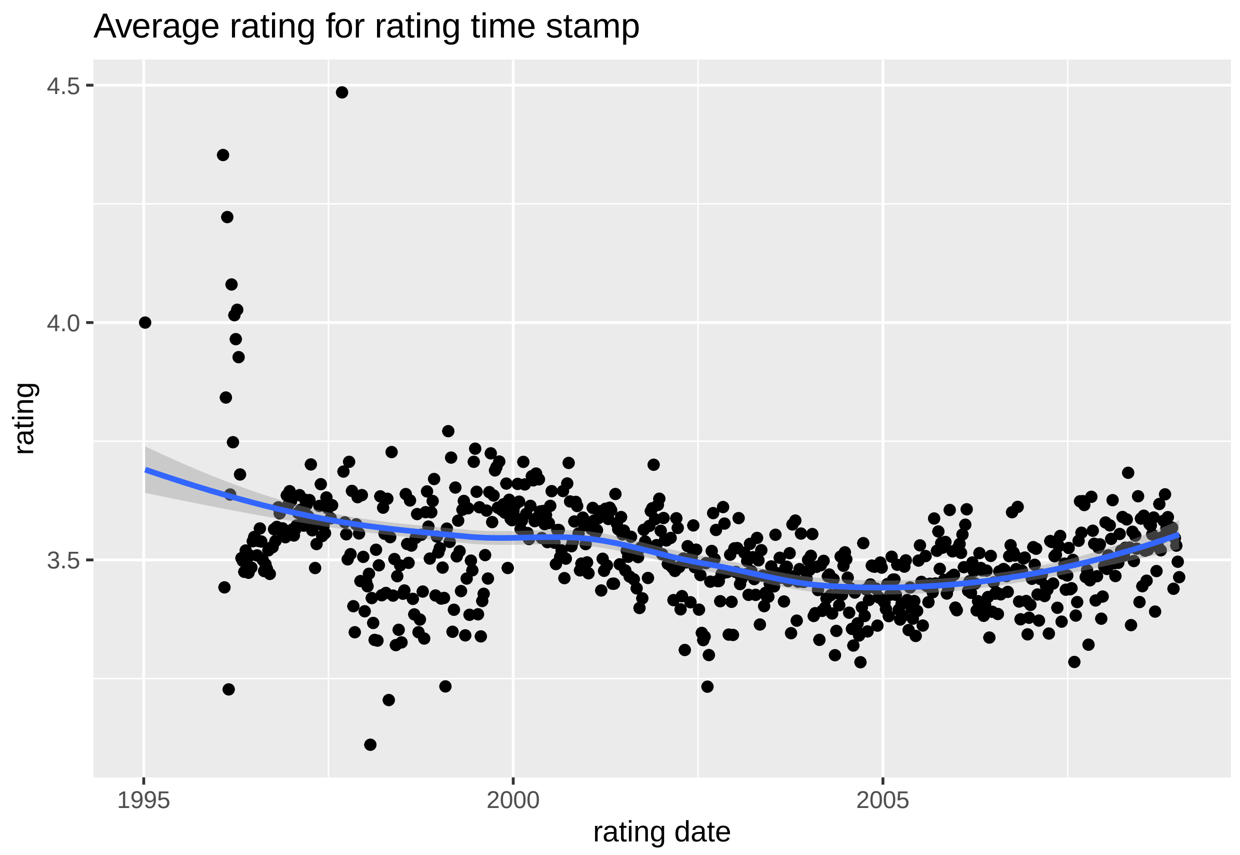


Let's analyze movie ratings by date. The plot below shows that movie ratings provided around 2005 are somewhat lower then other ratings. It also shows that ratings provided before 1997 are the highest.

```
edx <- mutate(edx, date = as_datetime(timestamp))

edx %>% mutate(date = round_date(date, unit = "week")) %>%
    group_by(date) %>%
    summarize(rating = mean(rating)) %>%
    ggplot(aes(date, rating)) +
    geom_point() +
    geom_smooth()+
  labs(x = "rating date", title = "Average rating for rating time stamp")
```

```
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
```

```
## ‘geom_smooth()‘ using method = ’loess’ and formula ’y ~ x’
```



Modelling approach

Considering the analyzis above, it can be concluded that the following parameters have an impact on the rating predictions: - movie bias - user bias - obscure ratings - movie genres - rating time stamp

Consequently, movie effect, user effect and regularization will be applied to the recommendation model.

Results

Naive prediction

First, let's predict the same rating for all movies regardless of user which is the average of all ratings.

```
movie_mean <- (mean(train$rating))
movie_mean
```

```
## [1] 3.512482
```

Let's check RMSE for this basic model.

```
RMSE(test$rating, movie_mean)
```

```
## [1] 1.059904
```

Movie effect

Secondly, let's add movie effects to our model as some movies are rated higher then others. We can use least squares to estimate the movie effect (b_i), but instead we estimate b_i by the average of difference between predicted rating and average rating for each movie .

```
movie_avgs <- train %>%
  group_by(movieId) %>%
  summarize(b_i = (mean(rating - movie_mean)))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

Let's check how RMSE has improved.

```
predicted_ratings <- movie_mean + test %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i
RMSE(predicted_ratings, test$rating)
```

```
## [1] 0.9437429
```

User effect

As some users tend to give low ratings while others are more eager to rate movies high. Let's call it user effect (b_u) and add it to the prediction. B_u can be computed as the average of the difference between estimated prediction, average rating and the movie effect for each movie.

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

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

The prediction with both movie effect and user effect can be computed.

```
predicted_ratings <- test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = movie_mean + b_i + b_u) %>%
  .$pred
```

Let's check RMSE for the model so far. Including movie effect and user effect.

```
RMSE(predicted_ratings, test$rating)
```

```
## [1] 0.865932
```

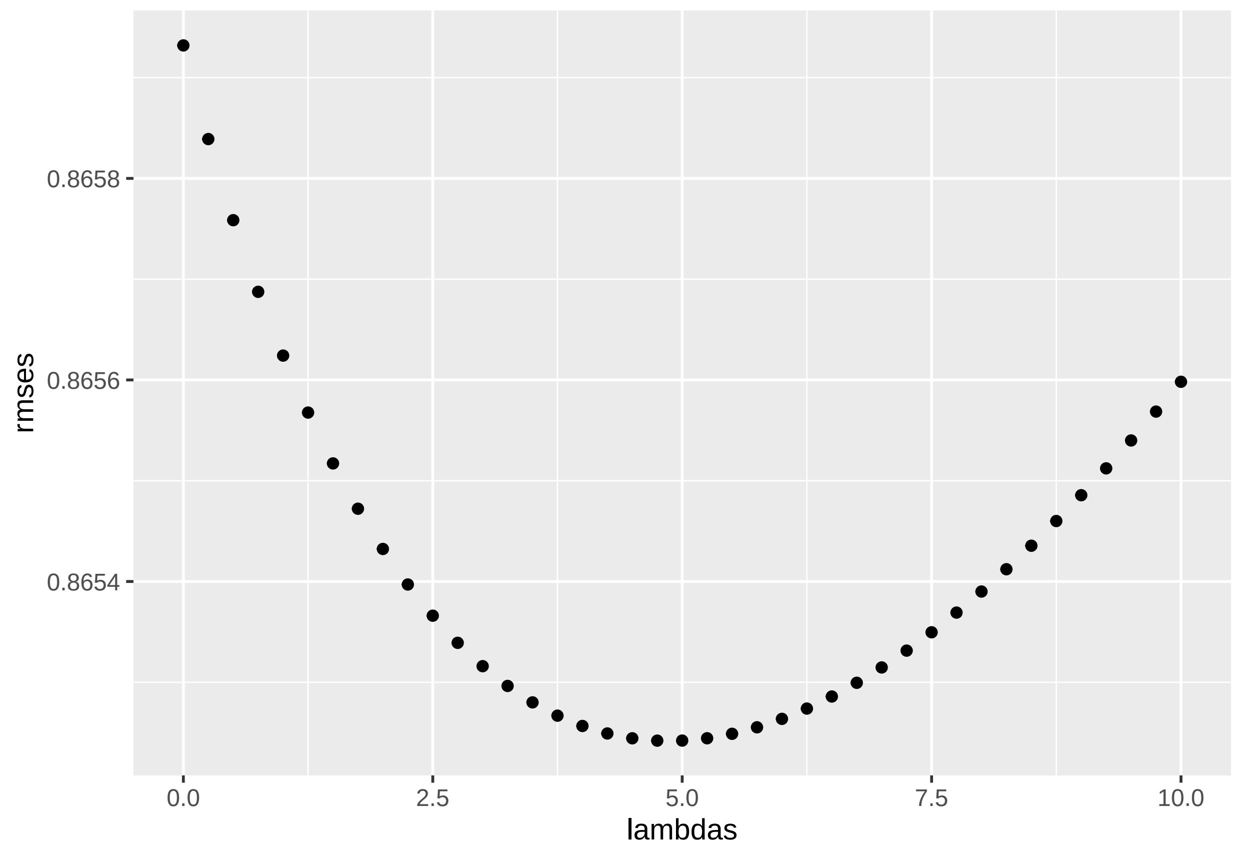
Regularization

Further improvements of the recommendation can be made. Let's try regularization. We're going to penalize large estimates made on small sample sizes. First let's use cross validation to choose the best lambda.

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

rmses <- sapply(lambdas, function(l){

  movie_mean <- mean(train$rating)

  b_i <- train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - movie_mean)/(n()+l))

  b_u <- train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - movie_mean)/(n()+l))

  predicted_ratings <- test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = movie_mean + b_i + b_u) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test$rating))
})
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
qplot(lambdas, rmses)
```



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

```
## [1] 4.75
```

The plot above shows that lambda 4.75 is the optimal penalty. RMSE for the regularized model with movie and user effects is:

```
rmses %>% min(rmses)
```

```
## [1] 0.8652421
```

Final RMSE based on the validation set.

```
movie_mean <- mean(edx$rating)
l = 4.75
```

```r
b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - movie_mean)/(n()+1))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - movie_mean)/(n()+1))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
predicted_ratings <- validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = movie_mean + b_i + b_u) %>%
    pull(pred)

RMSE(predicted_ratings, validation$rating)
```

```
## [1] 0.8648201
```

Conclusion

A machine learning algorithm was built in order to predict movie ratings with MovieLens 10M dataset. The regularized effects of unique users and movies were applied to the model.

The final RMSE result for the recommendation model is 0.8648201 which is better then the target RMSE of 0.86490.

Limitation of the recommendation system

Due to computational limitations and a large amount of data methods such as regression could not be used.

Future work

RMSE could be further improved by adding genre effect and rating time stamp effect.