Movie Recommendation

Overview

Movie recommendation system uses movie ratings data of different movies given by different users to make movie recommendations for the each user.

In this project, the goal is: Given a dataset of movies and ratings (not all movies are rated by all users and not all users rate all movies), we need to predict the rating that a user would give to a movie he hasn't yet rated.

We will be using the standard movielens 10M dataset for this project.

For each observation, we will be given userId, movieId, rating, timestamp, title and genres.

As an optimization metric, we will be using RMSE (root mean squared error) metric. We will report the best model (model with lower rmse) at the end.

We will start with analyzing the data, making a base model and improving its performance towards the end. We will be noting down the test rmse at every point of progress.

Loading libraries and downloading the data:

```r
#load the needed libraries
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## -- Conflicts ------------------------------------------------------------- tidyverse_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 from 'package:purrr':
##
##     lift
```

```r
library(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
library(Metrics)
```

```
##
## Attaching package: 'Metrics'

## The following objects are masked from 'package:caret':
##
##     precision, recall
```

```r
library(rafalib)
library(recosystem)
library(parallel)
library(furrr)
```

```
## Loading required package: future

##
## Attaching package: 'future'

## The following object is masked from 'package:caret':
##
##     cluster
```

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

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

# if using R 3.6 or earlier:
#movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
#                                            title = as.character(title),
#                                            genres = as.character(genres))
# if using R 4.0 or later:
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")
```

The data set has a total 10 million data points.
Divide the data into train(edx) and test(validation) sets with test set as 10% of the total data:

```r
# Create edx set, validation set (final hold-out test set)
# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
```

```
## 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,]

# Make sure userId and movieId in validation set are also in edx set
validation <- as.tibble(temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId"))
```

```
## Warning: `as.tibble()` is deprecated as of tibble 2.0.0.
## Please use `as_tibble()` instead.
## The signature and semantics have changed, see `?as_tibble`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```r
# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
```

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

```r
edx <- as.tibble(rbind(edx, removed))
```

Analyzing data (EDA)

```r
print(head(edx))
```

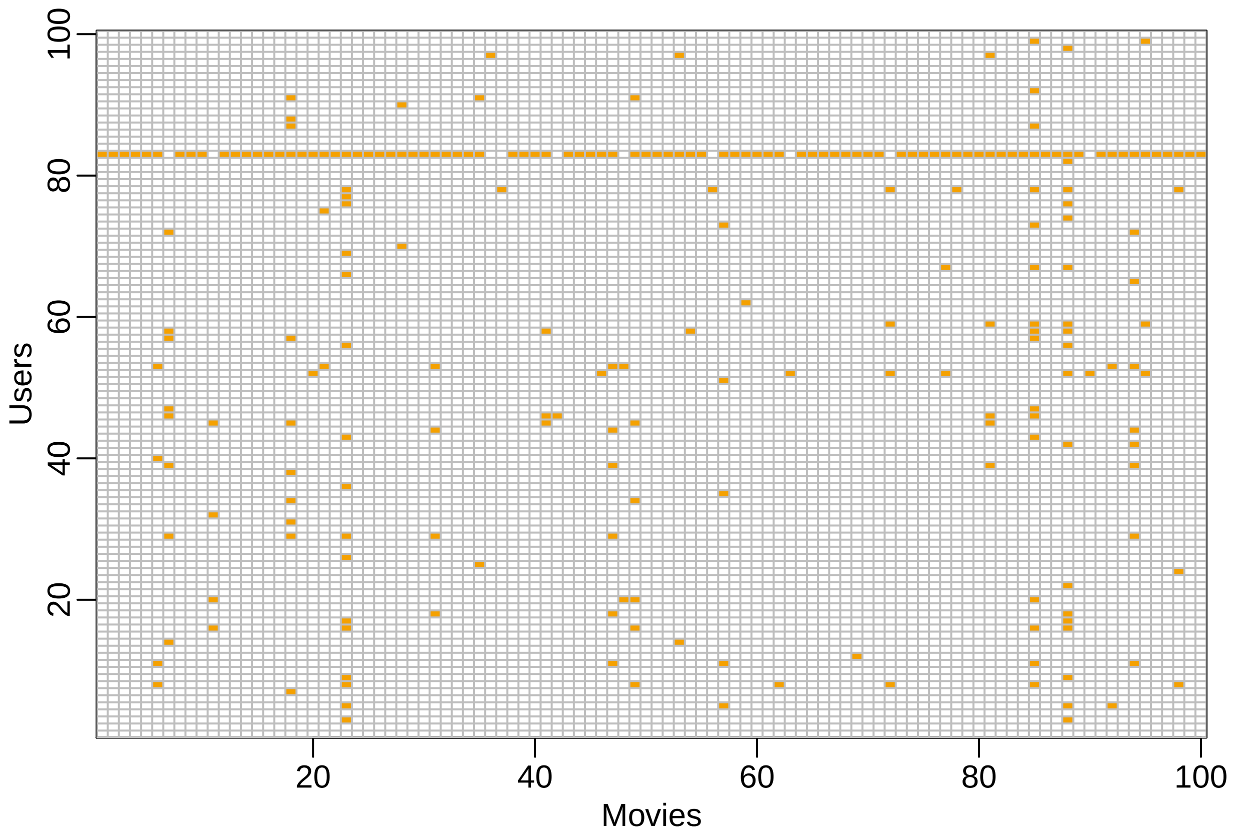
```
## # A tibble: 6 x 6
##   userId movieId rating timestamp title                   genres
##    <int>   <dbl>  <dbl>     <int> <chr>                   <chr>
## 1      1     122      5 838985046 Boomerang (1992)        Comedy|Romance
## 2      1     185      5 838983525 Net, The (1995)         Action|Crime|Thriller
## 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~
```

```
edx %>% summarise(
  n_users=n_distinct(userId),# unique users from train
  n_movies=n_distinct(movieId),# unique movies from train
  min_rating=min(rating),  # the lowest rating
  max_rating=max(rating) # the highest rating
)
```

```
## # A tibble: 1 x 4
##   n_users n_movies min_rating max_rating
##     <int>    <int>      <dbl>      <dbl>
## 1   69878    10677        0.5          5
```

We have 69878 unique users and 10677 unique movies in the edx (train) set. If we multiply n_users x n_movies, we get a number of almost 750 million. To visualize this is not possible because of the size constraints and it might crash r. Lets visualize some movies users. Matrix for a random sample of 100 movies and 100 users with yellow indicating a user/movie combination for which we have a rating.

```
users <- as.vector(sample(unique(edx$userId), 100))
rafalib::mypar()
edx %>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>% select(sample(ncol(.), 100)) %>%
  as.matrix() %>% t(.) %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users") +
abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
```



```
## integer(0)
```

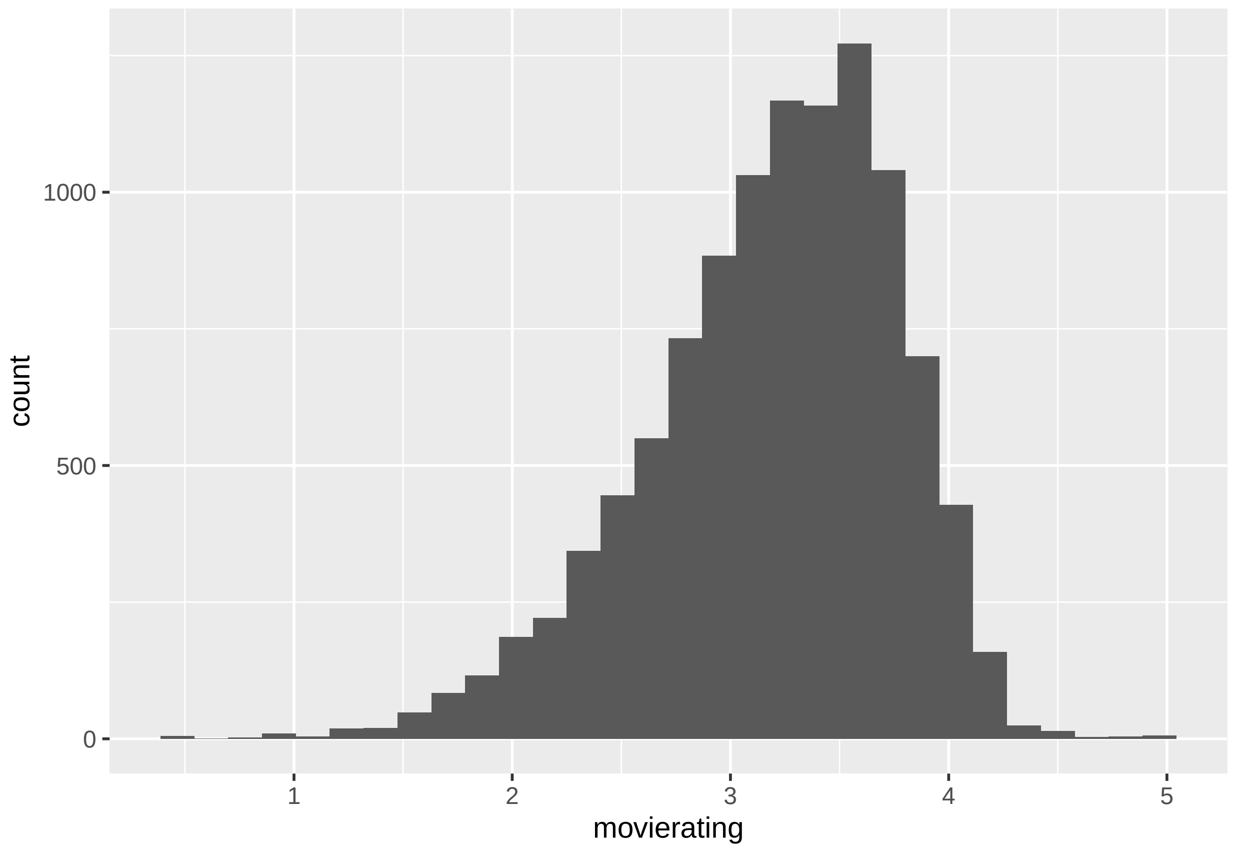
we see that the matrix is too sparse. There are too few filled ratings among the entire grid

Distribution of the avg rating of the movies

```
edx %>%
  group_by(movieId) %>%
  summarise(movierating = mean(rating)) %>%
  ggplot(aes(movierating)) +
  geom_histogram()
```

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

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

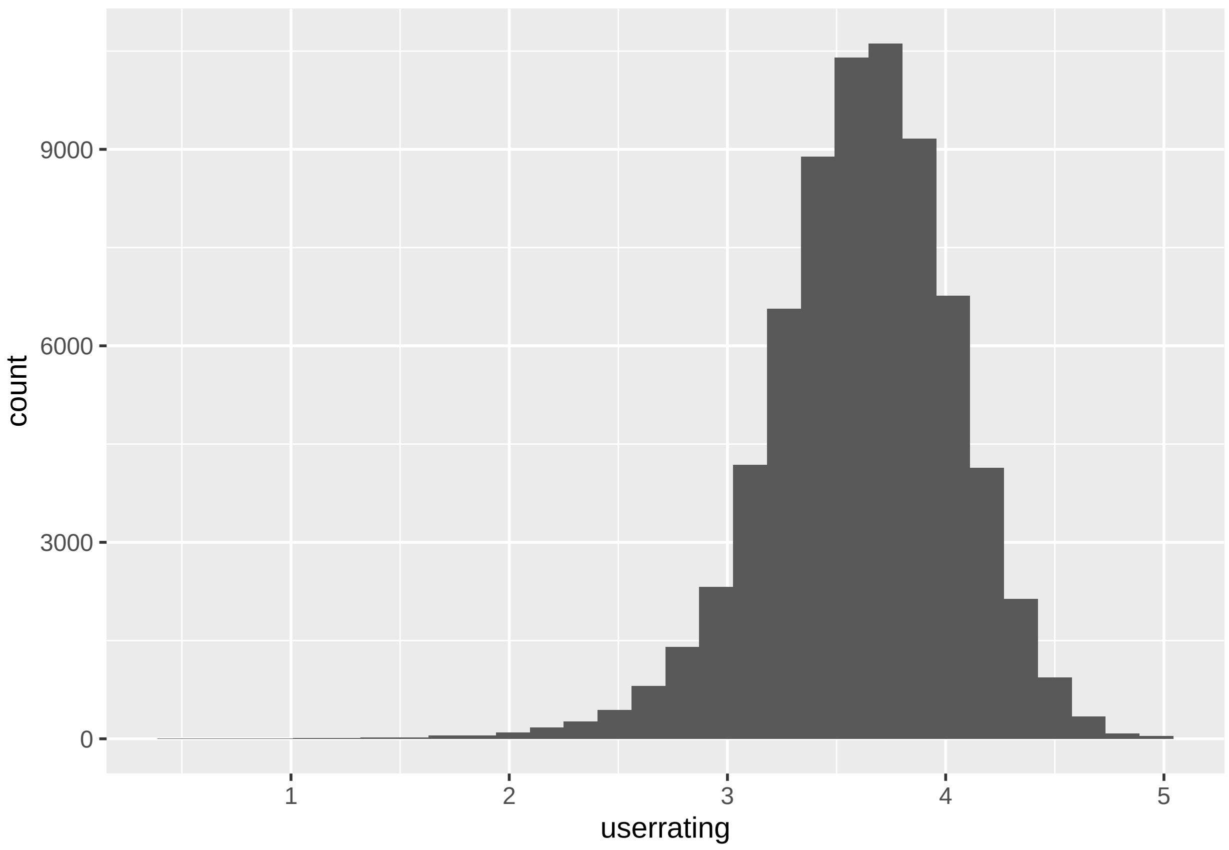


We see that all movies do not have a uniform average rating. Some movies have less than the total average rating, and some have more. We can say there is a bias for each movie which makes it's rating go away from the total average rating. Lets say it is movie bias(bm)

Distribution of the avg rating given by the users

```r
edx %>%
  group_by(userId) %>%
  summarise(userrating = mean(rating)) %>%
  ggplot(aes(userrating)) +
  geom_histogram()
```

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

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



We also notice that the users do not have a uniform average rating. Some users have less avg rating and some have more. User will be biased towards some particular movie and also in general each user may have the tendency to give more or less ratings. We therefore try to derive the user bias (bu) as well.

Methods and Analysis

Let's first try the simplest of all models, giving the average of all ratings as any prediction

```
mu <- mean(edx$rating)
cat('trainig rmse :', rmse(edx$rating, mu))
```

```
## trainig rmse : 1.060331
```

```
base_rmse <- rmse(validation$rating, mu)
rmse_results <- tibble(Model = "Average of ratings", RMSE = base_rmse)
cat('test rmse :', base_rmse)
```

```
## test rmse : 1.061202
```

We know from the data exploration that there exists a movie bias. Since there are so many rows.. fitting a linear model takes a lot of time We know that the least square estimate bi is just the average of yi - mu for each movie i. So we can compute them. y is the rating mu is the mean of all ratings

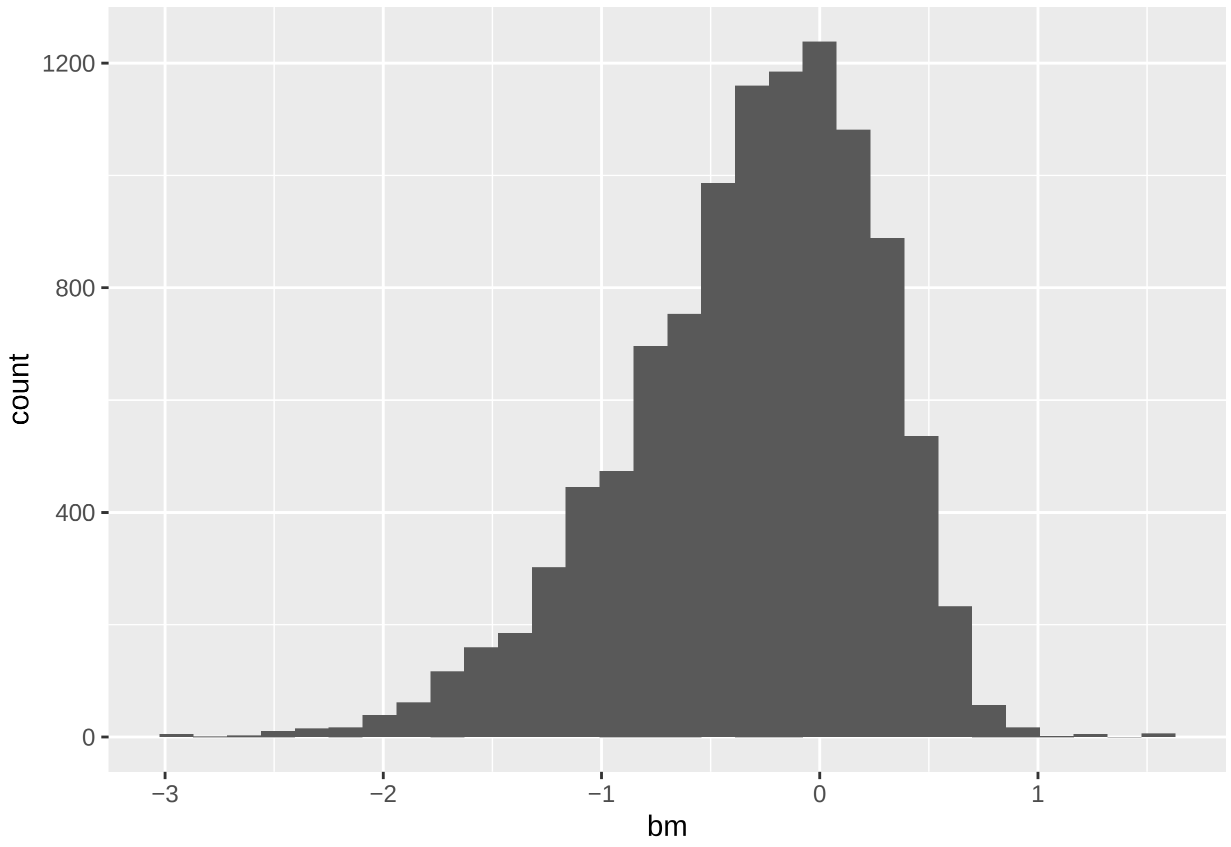
```
mbias <- edx %>%
  group_by(movieId) %>%
  summarise(bm = mean(rating - mu)) %>%
  ungroup()
```

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

Distribution of (movie bias) bm

```
mbias %>%
  ggplot(aes(bm)) + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
#train rmse
pred <- mu + edx %>%
  left_join(mbias, by = 'movieId') %>%
  .$bm
pred = ifelse(pred >= 0, pred, 0)
cat('train rmse :', rmse(edx$rating, pred))
```

```
## train rmse : 0.9423475
```

```r
#test rmse
pred <- mu + validation %>%
  left_join(mbias, by = 'movieId') %>%
  .$bm
pred = ifelse(pred >= 0, pred, 0)
bm_rmse = rmse(validation$rating, pred)
rmse_results <- bind_rows(rmse_results,
                          tibble(Model="movie_bias_effect",
                                 RMSE = bm_rmse))
cat('test rmse :', bm_rmse)
```

```
## test rmse : 0.9439087
```

We see that the rmse has reduced a little.. but is not up to the mark. We also know from the eda that there is a user bias. It is not practical to do a linear regression for userbias(bu) and movie bias(bm). We know that bu is the average of yiu - mu — bi (i is any movie i, u is any user u) y is the rating mu is the mean of all ratings b is the moviebias

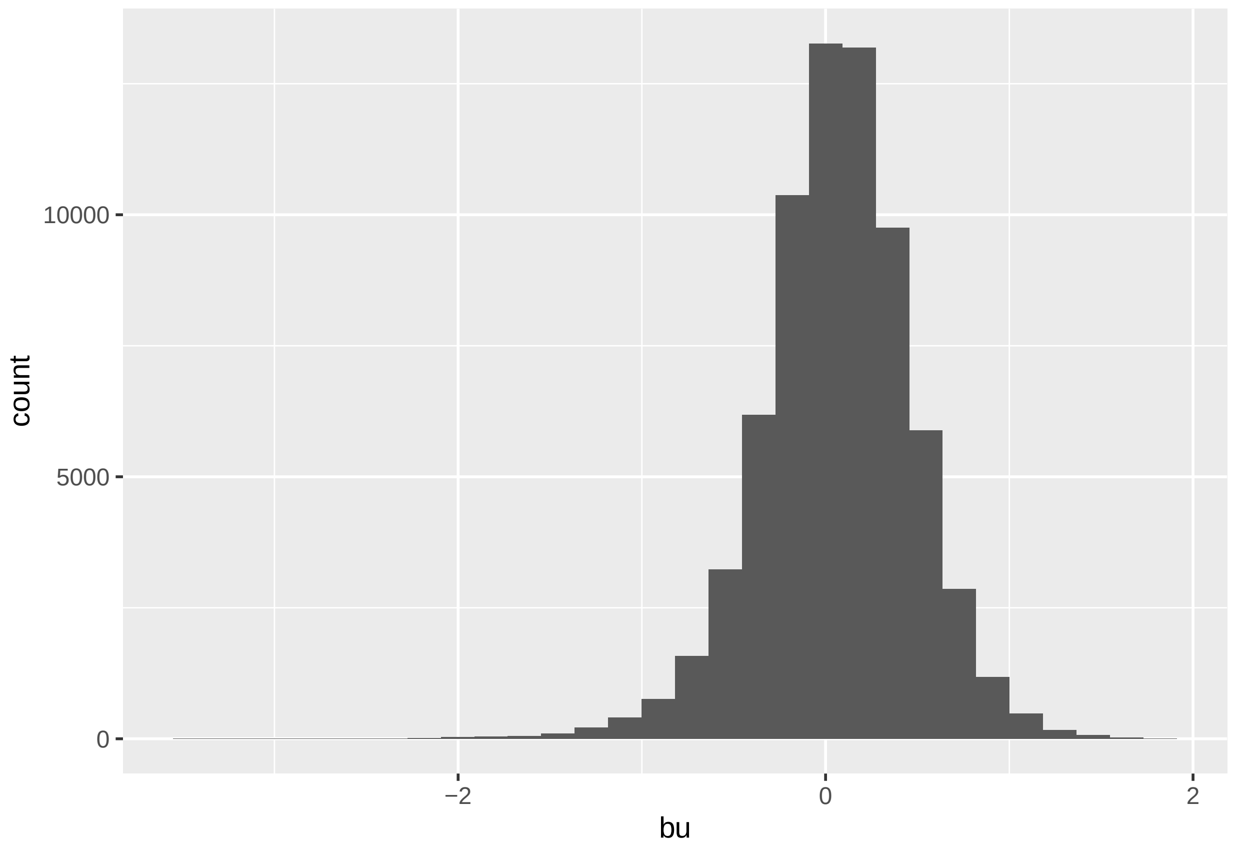
```r
ubias <- edx %>%
  left_join(mbias, by = 'movieId') %>%
  group_by(userId) %>%
  summarise(bu = mean(rating - mu - bm)) %>%
  ungroup()
```

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

Distribution of (user bias) bu

```r
ubias %>%
  ggplot(aes(bu)) + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```r
#train rmse
pred <- mu + edx %>%
  left_join(mbias, by = 'movieId') %>%
  left_join(ubias, by = 'userId') %>%
  mutate(req = bm + bu) %>%
  .$req
pred = ifelse(pred >= 0, pred, 0)
cat('train rmse:', rmse(edx$rating, pred))
```

```
## train rmse: 0.8567
```

```r
#test rmse
pred <- mu + validation %>%
  left_join(mbias, by = 'movieId') %>%
  left_join(ubias, by = 'userId') %>%
  mutate(req = bm + bu) %>%
  .$req
pred = ifelse(pred >= 0, pred, 0)
bmu_rmse <- rmse(validation$rating, pred)
rmse_results <- bind_rows(rmse_results,
                          tibble(Model="movie_user_bias_effect",
                                      RMSE = bmu_rmse))
cat('test rmse:', bmu_rmse)
```

```
## test rmse: 0.8653462
```

The rmse has further reduced, though not so significantly. Let us first look at the samples where the model made huge error.

```
analysis <- edx %>%
  left_join(mbias, by = 'movieId') %>%
  left_join(ubias, by = 'userId') %>%
  mutate(predr = (bm + bu + mu), diff = (predr - rating))  %>%
  group_by(title) %>%
  summarise(n = n(), rating =  mean(rating), predr = mean(predr), diff = mean(diff), diffsq = diff^2) %
  arrange(-diffsq)
```

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

```
print(head(analysis, 20))
```

```
## # A tibble: 20 x 6
##    title                                          n rating   predr   diff diffsq
##    <chr>                                      <int>  <dbl>   <dbl>  <dbl>  <dbl>
##  1 Accused (Anklaget) (2005)                      1   0.5  -1.86   -2.36   5.55
##  2 Stacy's Knights (1982)                         1   1    -0.865  -1.86   3.48
##  3 Mountain Eagle, The (1926)                     2   1    -0.0916 -1.09   1.19
##  4 Confessions of a Superhero (2007)              1   0.5  -0.508  -1.01   1.02
##  5 Condo Painting (2000)                          1   3     2.04   -0.959  0.920
##  6 Neil Young: Human Highway (1982)               1   1.5   0.545  -0.955  0.913
##  7 Criminals (1996)                               2   1     0.0578 -0.942  0.888
##  8 Elstree Calling (1930)                         3   1.5   0.588  -0.912  0.832
##  9 Jack-O (1995)                                  4   2.5   1.65   -0.852  0.726
## 10 Goodbye, 20th Century (Zbogum na dvadeset~     3   1.83  1.00   -0.832  0.692
## 11 Song of Freedom (1936)                         3   3.67  2.84   -0.830  0.690
## 12 Deux mondes, Les (2007)                        1   2.5   1.69   -0.808  0.653
## 13 In Bed (En la cama) (2005)                     1   2.5   1.69   -0.808  0.653
## 14 Dogwalker, The (2002)                          1   2     1.21   -0.786  0.617
## 15 Strange Planet (1999)                          1   2     1.22   -0.777  0.603
## 16 Last Minute, The (2001)                        2   2     1.27   -0.734  0.539
## 17 Last Mistress, The (Une vieille maîtresse~     8   3     2.29   -0.712  0.507
## 18 Free Radicals (Böse Zellen) (2003)             3   2.83  2.12   -0.710  0.505
## 19 Stone Angel, The (2007)                        1   2.5   1.80   -0.702  0.493
## 20 Emerald Cowboy (2002)                          1   3     2.31   -0.686  0.471
```

```
cat('avg number of ratings per movie', mean(analysis$n))
```

```
## avg number of ratings per movie 843.0175
```

We have an avg of 843 ratings per movie, lets see how many odd ones are present. say the odd ones are the ones having less than 50 ratings

```
analysis %>%
  filter(n < 50)
```

```
## # A tibble: 3,595 x 6
##    title                                          n rating   predr   diff diffsq
##    <chr>                                     <int>  <dbl>   <dbl>  <dbl>  <dbl>
##  1 Accused (Anklaget) (2005)                     1   0.5  -1.86   -2.36   5.55
##  2 Stacy's Knights (1982)                        1   1    -0.865  -1.86   3.48
##  3 Mountain Eagle, The (1926)                    2   1    -0.0916 -1.09   1.19
##  4 Confessions of a Superhero (2007)             1   0.5  -0.508  -1.01   1.02
##  5 Condo Painting (2000)                         1   3     2.04   -0.959  0.920
##  6 Neil Young: Human Highway (1982)              1   1.5   0.545  -0.955  0.913
##  7 Criminals (1996)                              2   1     0.0578 -0.942  0.888
##  8 Elstree Calling (1930)                        3   1.5   0.588  -0.912  0.832
##  9 Jack-O (1995)                                 4   2.5   1.65   -0.852  0.726
## 10 Goodbye, 20th Century (Zbogum na dvadeset~    3   1.83  1.00   -0.832  0.692
## # ... with 3,585 more rows
```

There are roughly 3500 odd movies with less than 50 ratings.. this will drastically effect our analysis

Cases where we underestimated

```
analysis %>%
  filter(diff < 0, n < 50) %>%
  arrange(-diffsq)
```

```
## # A tibble: 3,010 x 6
##    title                                         n rating   predr   diff diffsq
##    <chr>                                     <int>  <dbl>   <dbl>  <dbl>  <dbl>
##  1 Accused (Anklaget) (2005)                     1   0.5  -1.86   -2.36   5.55
##  2 Stacy's Knights (1982)                        1   1    -0.865  -1.86   3.48
##  3 Mountain Eagle, The (1926)                    2   1    -0.0916 -1.09   1.19
##  4 Confessions of a Superhero (2007)             1   0.5  -0.508  -1.01   1.02
##  5 Condo Painting (2000)                         1   3     2.04   -0.959  0.920
##  6 Neil Young: Human Highway (1982)              1   1.5   0.545  -0.955  0.913
##  7 Criminals (1996)                              2   1     0.0578 -0.942  0.888
##  8 Elstree Calling (1930)                        3   1.5   0.588  -0.912  0.832
##  9 Jack-O (1995)                                 4   2.5   1.65   -0.852  0.726
## 10 Goodbye, 20th Century (Zbogum na dvadeset~    3   1.83  1.00   -0.832  0.692
## # ... with 3,000 more rows
```

Cases where we overestimated

```
analysis %>%
  filter(diff > 0, n < 50) %>%
  arrange(-diffsq)
```

```
## # A tibble: 585 x 6
##    title                                            n rating predr  diff diffsq
##    <chr>                                        <int>  <dbl> <dbl> <dbl>  <dbl>
##  1 Mickey (2003)                                    1   4.5  5.16  0.663  0.440
##  2 Fists in the Pocket (I Pugni in tasca) (1965)    1   4    4.66  0.658  0.433
##  3 War of the Worlds 2: The Next Wave (2008)        2   0.5  0.966 0.466  0.217
##  4 Down and Derby (2005)                            1   3.5  3.96  0.456  0.208
##  5 Jimmy Carter Man from Plains (2007)              1   4    4.45  0.451  0.204
##  6 Small Cuts (Petites coupures) (2003)             1   3    3.45  0.451  0.204
##  7 If You Only Knew (2000)                          5   3.6  4.04  0.440  0.193
##  8 Tashunga (1995)                                 19   3.53 3.96  0.430  0.185
##  9 Ring of Darkness (2004)                          1   3.5  3.92  0.418  0.175
## 10 Entertaining Angels: The Dorothy Day Story (~    19   3.53 3.92  0.395  0.156
## # ... with 575 more rows
```

since we have extremes in the movies which are rated less.. in most cases by one or 2 users its is good to regularize the moviebias (bm) and user bias (bu) in order to penalize the estimates that come from small sample sizes.

lambda be the regularization parameter for the movies with less data. We will follow the same procedure as used so far.. but with regularization parameter.

To choose the appropriate regularization parameter lambda using 4 fold cross validation. lets divide the edx set to 4 sets.

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

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

```
tmp <- createDataPartition(y = edx$rating, times = 1, p = 0.5, list = F)
t <- edx[tmp, ]
```

```
## Warning: The `i` argument of ``[`()` can't be a matrix as of tibble 3.0.0.
## Convert to a vector.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
p <- createDataPartition(y = t$rating, times = 1, p = 0.5, list = F)
p1 <- t[p, ]
p2 <- t[-p, ]
t <- edx[-tmp, ]
p <- createDataPartition(y = t$rating, times = 1, p = 0.5, list = F)
p3 <- t[p, ]
p4 <- t[-p, ]

v <- list(p1, p2, p3, p4)
```

We will try running the steps parallely to optimize time

```
#lambda is the regularization parameter which we will optimize over the regltn set
possible_lambdas <- seq(4, 5.5, by = 0.25)

plan(multicore) #we will be using multiprocessing to run in parallel

options(future.globals.maxSize= 891289600)

getrmses <- function(l) {
  r <- future_map_dbl(v, getmeancvrmse, l = l)
  mean(r)
}

getmeancvrmse <- function(temp, l = 0) {
  #we make train and regltn sets appropriately for all the 4 cross validation folds
  train <- anti_join(edx, temp)
  # Make sure userId and movieId in regltn set are also in train set
  regltn <- temp %>%
    semi_join(train, by = "movieId") %>%
    semi_join(train, by = "userId")

  # Add rows removed from regltn set back into train set
  removed <- anti_join(temp, regltn)
  train <- rbind(train, removed)

  mbias <- train %>%
    group_by(movieId) %>%
    summarise(bm = sum(rating - mu)/(n() + l)) %>%
    ungroup()
  ubias <- train %>%
    left_join(mbias, by = 'movieId') %>%
    group_by(userId) %>%
    summarise(bu = sum(rating - mu - bm)/(n() + l)) %>%
    ungroup()
  pred <- mu + regltn %>%
    left_join(mbias, by = 'movieId') %>%
    left_join(ubias, by = 'userId') %>%
    mutate(req = bm + bu) %>%
    .$req
  pred = ifelse(pred >= 0, pred, 0)
  rmse(regltn$rating, pred)
}
rmses <- future_map_dbl(possible_lambdas, getrmses)
```

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

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

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

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

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

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

## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
```

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

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

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

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

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

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

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

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

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

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

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

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

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

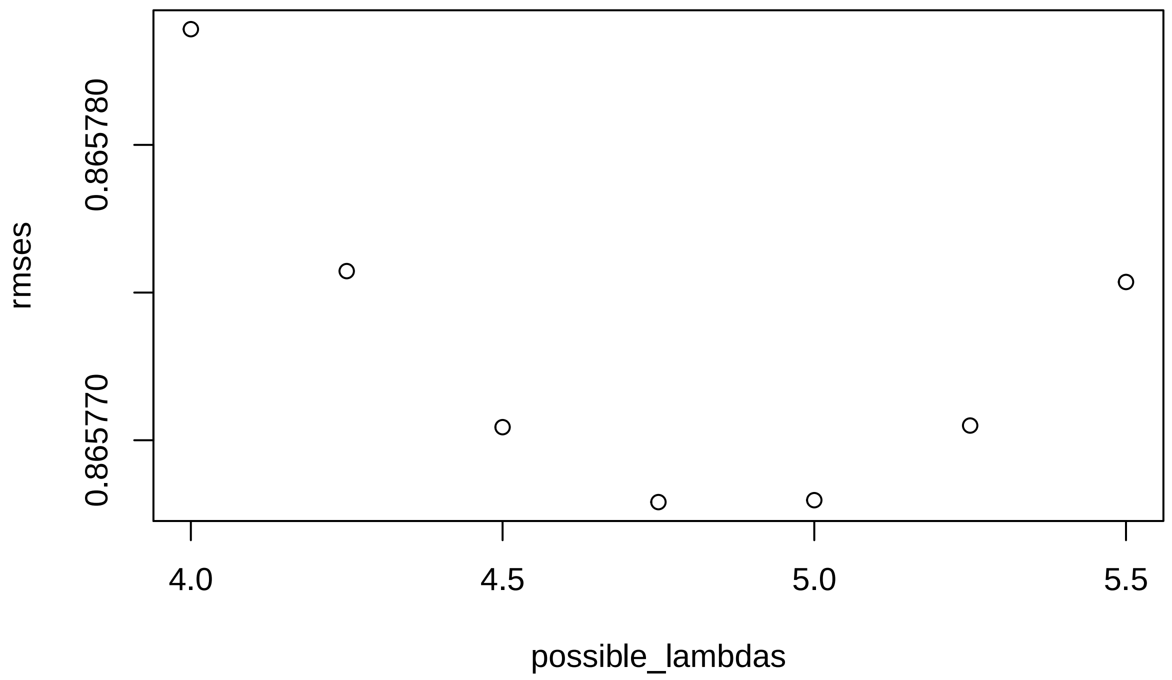
```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
```

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

```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Plotting rmses vs lambdas

```r
plot(possible_lambdas, rmses)
```



```
rmses
```

```
## [1] 0.8657839 0.8657757 0.8657704 0.8657679 0.8657680 0.8657705 0.8657754
```

Best lambda

```
l <- possible_lambdas[min(rmses) == rmses]
print(l)
```

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

We got the optimal value of lambda(l). We can now train the whole edx set upon the validation parameter. Working : If n() for a movie/user is large, it doesn't make much difference in moviebias(bm) / userbias(bu). But if n() for a movie/user is very small, then the moviebias(bm) / userbias(bu) will be less than what they otherwise would be.. helping in getting a less inaccurate estimate for smaller samples.

Training the whole set on l.

```
mbias <- edx %>%
  group_by(movieId) %>%
  summarise(bm = sum(rating - mu)/(n() + l)) %>%
  ungroup()
```

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

```
ubias <- edx %>%
  left_join(mbias, by = 'movieId') %>%
  group_by(userId) %>%
  summarise(bu = sum(rating - mu - bm)/(n() + l)) %>%
  ungroup()
```

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

Predictions and results

```r
#predictions for train and test sets
predtrain <- mu + edx %>%
  left_join(mbias, by = 'movieId') %>%
  left_join(ubias, by = 'userId') %>%
  mutate(req = bm + bu) %>%
  .$req
predtrain = ifelse(predtrain >= 0, predtrain, 0)

predtest <- mu + validation %>%
  left_join(mbias, by = 'movieId') %>%
  left_join(ubias, by = 'userId') %>%
  mutate(req = bm + bu) %>%
  .$req
predtest = ifelse(predtest >= 0, predtest, 0)
#rmse computations
cat('train rmse :', rmse(edx$rating, predtrain))
```

```
## train rmse : 0.857011
```

```r
mbreg_bias <- rmse(validation$rating, predtest)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="regularised_movie_user_bias_effect",
                                     RMSE = mbreg_bias))
```

```
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```r
cat('test rmse :', mbreg_bias)
```

```
## test rmse : 0.8648188
```

Let us get the residuals of the predictions and then perform matrix factorization on the residuals.

```r
edx_res <- edx %>%
  left_join(mbias, by = "movieId") %>%
  left_join(ubias, by = "userId") %>%
  mutate(res = rating - mu - bm - bu) %>%
  select(userId, movieId, res)
head(edx_res)
```

```
## # A tibble: 6 x 3
##   userId movieId   res
##    <int>   <dbl> <dbl>
## 1      1     122 0.797
## 2      1     185 0.527
## 3      1      292 0.239
## 4      1      316 0.307
## 5      1      329 0.319
## 6      1      355 1.17
```

We will be using recosystem library to perform matrix factorization on the residuals. For this both training and test set needs to be arranged in 3 columns userId, movieId, rating which will be transformed to a matrix format.

Note: We will be predicting residuals using this model, and the residuals will be added to output of the best model obtained so far. Then rmse is computed and compared with other models.

```r
# as matrix
train_fact <- as.matrix(edx_res)
test_fact <- validation %>%
  select(userId, movieId, rating)
test_fact <- as.matrix(test_fact)

# build a recommender object
r <-Reco()

# write train_fact and test_fact tables on disk
write.table(train_fact , file = "trainset.txt" , sep = " " , row.names = FALSE, col.names = FALSE)
write.table(test_fact, file = "validset.txt" , sep = " " , row.names = FALSE, col.names = FALSE)

# use data_file() to specify a data set from a file in the hard disk.
# r$tune needs training data to be of type DataSource.
set.seed(2000)
train_fact <- data_file("trainset.txt")
test_fact <- data_file("validset.txt")
```

We will tune the training data with r$tune to get the best optimal parameters for training.

```r
# tuning training set using the default 5 fold cross validation
cores <- detectCores()
opts <- r$tune(train_fact, opts = list(dim = c(40, 45), lrate = c(0.05, 0.1, 0.2),
                                      costp_l1 = 0, costq_l1 = 0,
                                      nthread = (cores - 1), niter = 10))
opts
```

```
## $min
## $min$dim
## [1] 40
##
## $min$costp_l1
## [1] 0
##
## $min$costp_l2
## [1] 0.01
##
## $min$costq_l1
## [1] 0
##
```

```
## $min$costq_l2
## [1] 0.1
##
## $min$lrate
## [1] 0.1
##
## $min$loss_fun
## [1] 0.791104
##
##
## $res
##    dim costp_l1 costp_l2 costq_l1 costq_l2 lrate  loss_fun
## 1   40        0     0.01        0     0.01  0.05 0.8066172
## 2   45        0     0.01        0     0.01  0.05 0.8060118
## 3   40        0     0.10        0     0.01  0.05 0.7956834
## 4   45        0     0.10        0     0.01  0.05 0.7948895
## 5   40        0     0.01        0     0.10  0.05 0.8005457
## 6   45        0     0.01        0     0.10  0.05 0.7978886
## 7   40        0     0.10        0     0.10  0.05 0.8277587
## 8   45        0     0.10        0     0.10  0.05 0.8268016
## 9   40        0     0.01        0     0.01  0.10 0.8265211
## 10  45        0     0.01        0     0.01  0.10 0.8313061
## 11  40        0     0.10        0     0.01  0.10 0.8013637
## 12  45        0     0.10        0     0.01  0.10 0.8029507
## 13  40        0     0.01        0     0.10  0.10 0.7911040
## 14  45        0     0.01        0     0.10  0.10 0.7916930
## 15  40        0     0.10        0     0.10  0.10 0.8235568
## 16  45        0     0.10        0     0.10  0.10 0.8232341
## 17  40        0     0.01        0     0.01  0.20 0.8510833
## 18  45        0     0.01        0     0.01  0.20 0.8540502
## 19  40        0     0.10        0     0.01  0.20 0.8103681
## 20  45        0     0.10        0     0.01  0.20 0.8113348
## 21  40        0     0.01        0     0.10  0.20 0.7996436
## 22  45        0     0.01        0     0.10  0.20 0.8014471
## 23  40        0     0.10        0     0.10  0.20 0.8209091
## 24  45        0     0.10        0     0.10  0.20 0.8215474
```

Training the model and making predictions

```r
# training the recommender model
r$train(train_fact, opts = c(opts$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.8602   7.0079e+06
##    1       0.8356   6.4661e+06
##    2       0.8154   6.2720e+06
##    3       0.7960   6.0809e+06
##    4       0.7799   5.9312e+06
##    5       0.7668   5.8153e+06
##    6       0.7559   5.7213e+06
##    7       0.7467   5.6460e+06
##    8       0.7386   5.5835e+06
##    9       0.7315   5.5294e+06
##   10       0.7251   5.4813e+06
##   11       0.7193   5.4397e+06
##   12       0.7141   5.4023e+06
##   13       0.7094   5.3692e+06
##   14       0.7050   5.3383e+06
##   15       0.7012   5.3120e+06
##   16       0.6976   5.2883e+06
##   17       0.6944   5.2668e+06
##   18       0.6914   5.2465e+06
##   19       0.6887   5.2289e+06
```

```
# Making prediction on validation set and calculating RMSE:
pred_file <- tempfile()
r$predict(test_fact, out_file(pred_file))
```

```
## prediction output generated at /var/folders/fc/clqxht0d7vbbvtphcc8vykbw0000gn/T//RtmpP4914Q/file1d14
```

```
y_pred_resid <- scan(pred_file)
```

Adding the predictions to our previous model test output predtest.

```
y_pred <- predtest + y_pred_resid
rmse_mf <- RMSE(y_pred,validation$rating)
cat('final Rmse:', rmse_mf)
```

```
## final Rmse: 0.7852316
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="factorization + regularised_movie_user_bias",
                                     RMSE = rmse_mf))
```

Results and Conclusions / Future Work

We see how rmse reduced over the models and we get a final rmse of < 0.79 on the test(validation) set.

```
print(rmse_results)
```

```
## # A tibble: 5 x 2
##   Model                                        RMSE
##   <chr>                                       <dbl>
## 1 Average of ratings                          1.06
## 2 movie_bias_effect                           0.944
## 3 movie_user_bias_effect                      0.865
## 4 regularised_movie_user_bias_effect          0.865
## 5 factorization + regularised_movie_user_bias 0.785
```

We still have some issues in the model to be addressed:

1) Sometimes the model predicts a rating of less than 5 or greater than 5. Though this doesn't effect the recommendation system as the above 5 rating are surely recommended and lower ones are not recommended provided the system is equipped with getting < 0 or > 5 ratings.. else the system would not function as expected. Reason: According to our model, some users can have more(+/-) user bias and some movies can have more(+/-) movie bias. We finally output y = mu + moviebias + userbias + residualpredictions. So overall the y can be cross the limits of 0 and 5.

2) If a user is absolutely new, we cannot know any of his preferences to recommend which is called coldstart problem.

Future work: 1) As a solution for coldstart problem the in normal scenario is to recommend user higher rated movies. But the problem of user may not like some depending on their gender/age/employment_status. To resolve that we should be provided with user profile details. Using the details of the profile, to know which category they belong to, recommend them the best recommendations given for a person of their category on average. That might enhance the user experience more than just giving mean of all ratings.

3) Use timestamp and genres info to extract further useful information to reduce the rmse.

4) Use title for finding movies like starwars - part 1, 2, 3, 4. If the user likes starwars, they are likely to like the other movie series too.