MovieLens

Lupita Sahu

17 June 2019

Project Overview

Recommendation systems use ratings that users have given items to make specific recommendations. Netflix uses a recommendation system to predict how many stars a user will give a specific movie. One star suggests it is not a good movie, whereas five stars suggests it is an excellent movie. Here, we will build a prediction algorithm that can reduce the RMSE to be below 87%.

Data loading

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 3.5.3
```

```
## -- Attaching packages --------------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 3.0.0       v purrr   0.3.1
## v tibble  2.0.1       v dplyr   0.8.0.1
## v tidyr   0.8.1       v stringr 1.3.1
## v readr   1.1.1       v forcats 0.3.0
```

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

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.5.3
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

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

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

ratings <- read.table(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))), col.names

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data
```

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

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

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

# 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 <- rbind(edx, removed)

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

Exploratory data analysis

Let's look at how the data looks

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

```
## # A tibble: 9,000,055 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|Thrill~
##  3      1     292      5 838983421 Outbreak (1995)     Action|Drama|Sci-Fi~
##  4      1     316      5 838983392 Stargate (1994)     Action|Adventure|Sc~
##  5      1     329      5 838983392 Star Trek: Generat~ Action|Adventure|Dr~
##  6      1     355      5 838984474 Flintstones, The (~ Children|Comedy|Fan~
##  7      1     356      5 838983653 Forrest Gump (1994) Comedy|Drama|Romanc~
##  8      1     362      5 838984885 Jungle Book, The (~ Adventure|Children|~
##  9      1     364      5 838983707 Lion King, The (19~ Adventure|Animation~
## 10      1     370      5 838984596 Naked Gun 33 1/3: ~ Action|Comedy
## # ... with 9,000,045 more rows
```

We can group into the total number of unique users and total number of unique movies.

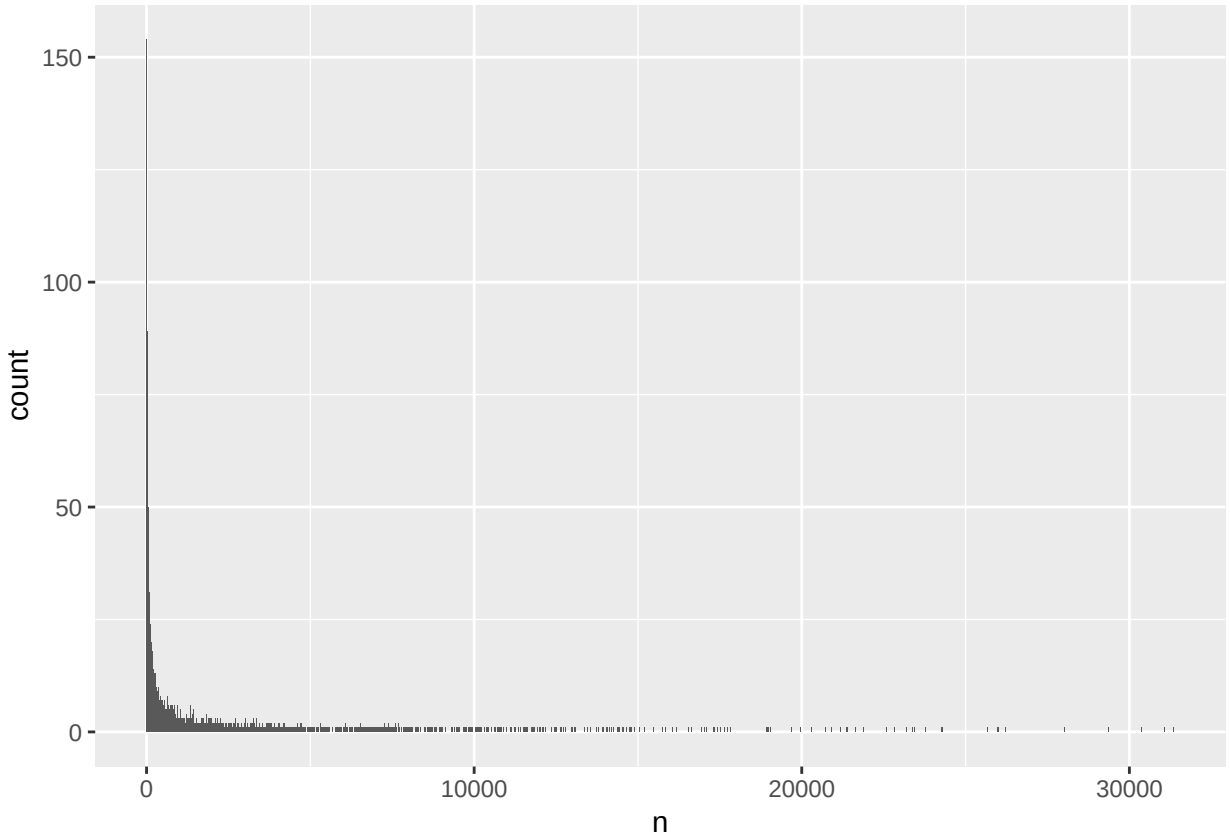
```r
edx %>% summarize(n_users = n_distinct(userId), n_movies = n_distinct(movieId))
```

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

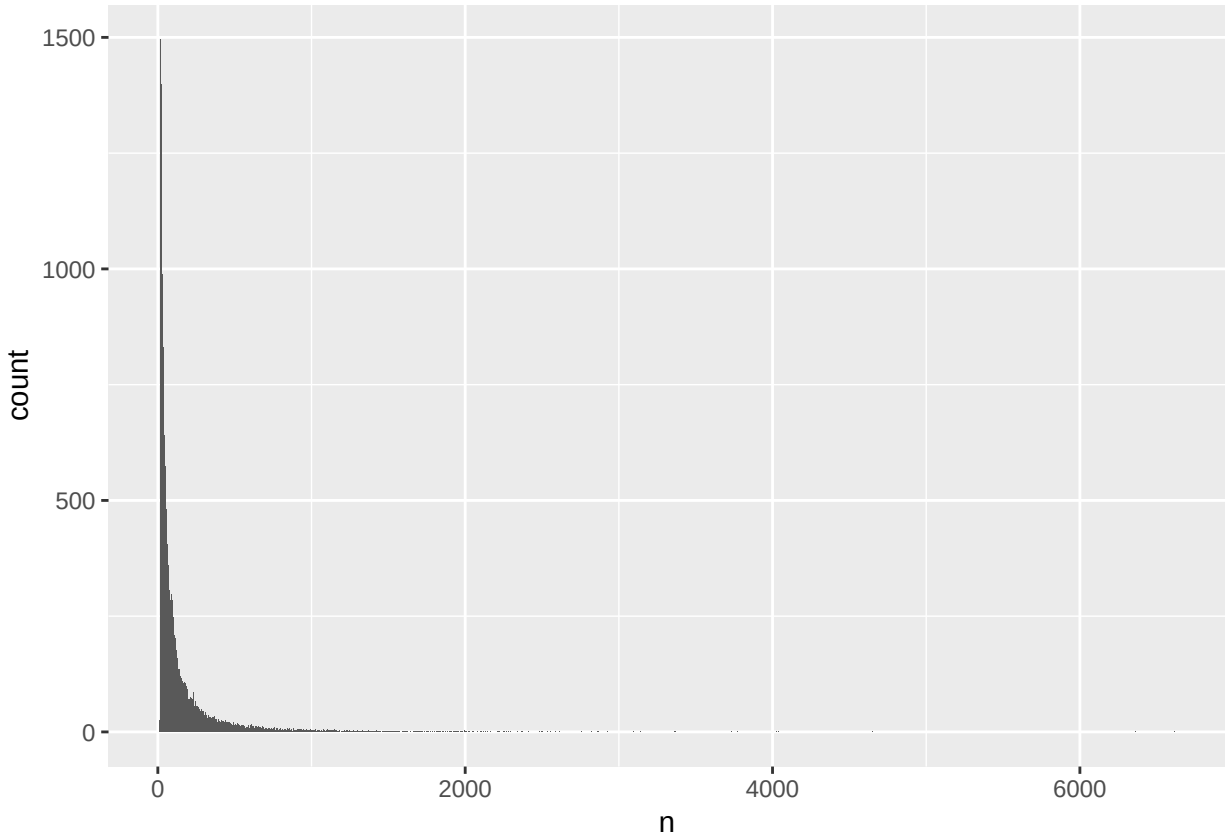
We can see that not every user reviewed every movie. The prediction algorithm in a way fills the ratings for each user for each movie.

Let's look at how the movies are rated. It looks like some movies are rated more frequently than others. Likewise some users are more active at rating movies than others.

```r
edx %>% group_by(movieId) %>% summarise(n=n(), title=first(title)) %>% ggplot(aes(n)) + geom_bar(stat="
```

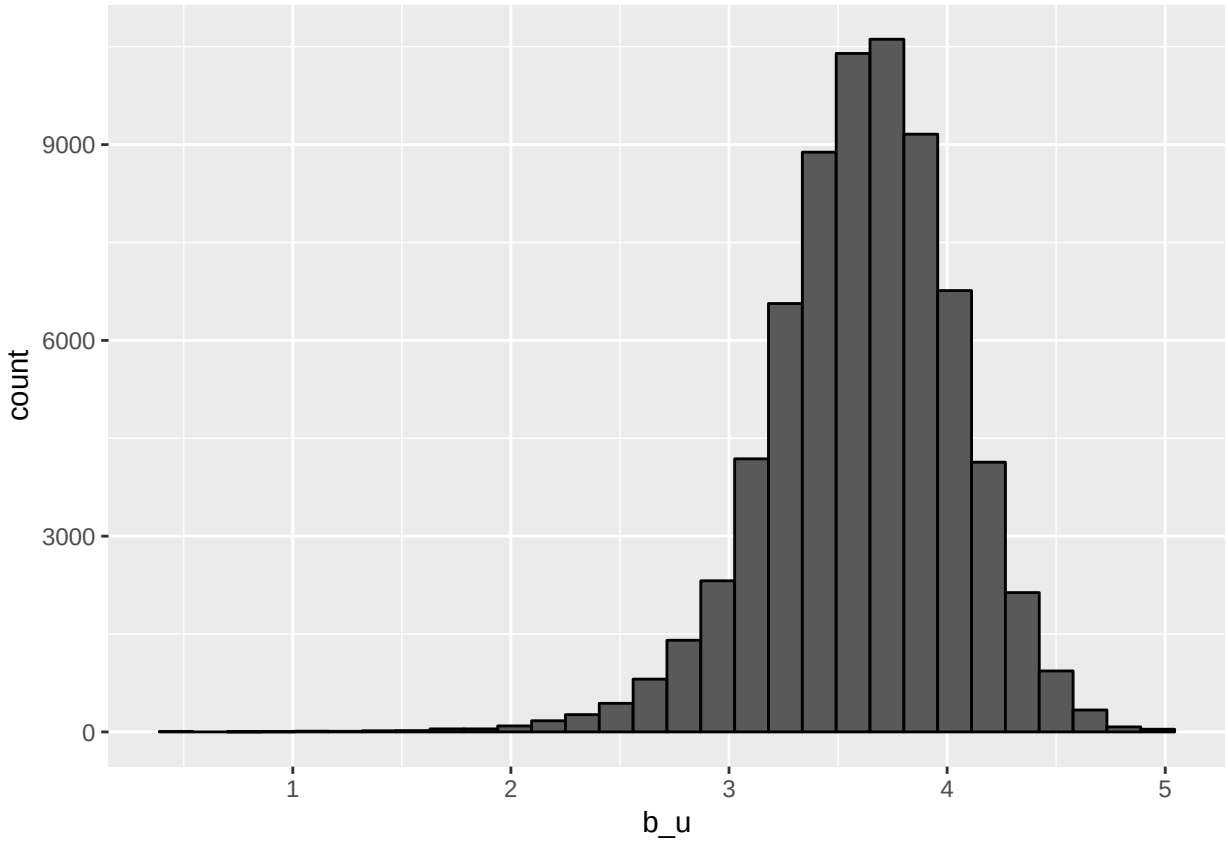



```
edx %>% group_by(userId) %>% summarise(n=n(), title=first(title)) %>% ggplot(aes(n)) + geom_bar(stat="c
```



Let's look at average rating given by users who have rated more than 100 movies. We see that some users are generous while other are not.

```
edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```



Preprocessing

We will partition the edx data into train and test data sets with 90%-10% split.

```
set.seed(135)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train <- edx[-test_index,]
temp <- edx[test_index,]

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

## take observations removed from test and add to train
removed <- anti_join(temp, test)
```

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

```
train <- rbind(train, removed)
rm(temp)
```

Modelling

Let's assume all movies have same ratings and calculate the RMSE

```
#Calculate the mean rating
mu <- mean(train$rating)
```

```
# calculate RMSE for naive model of predicting muHat
naiveRmse <- RMSE(test$rating, mu)
naiveRmse
```

```
## [1] 1.059463
```

RMSE of 1.06 can be improved after adding movie effect and user effect. After adding the movie bias RMSE can be calculated as follows.

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

predicted_ratings <- mu + test %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

model1_rmse <- RMSE(predicted_ratings, test$rating)
model1_rmse
```

```
## [1] 0.9428774
```

RMSE is better now at a value of 0.94. Let's calculate RMSE after consider user bias as follows:

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

predicted_ratings <- test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

model2_rmse <- RMSE(predicted_ratings, test$rating)
model2_rmse
```

```
## [1] 0.8642932
```

RMSE is now .87 which is much better. However we can still improve out model. Here are the movies with highest residuals:

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

```
##                                                      title  residual
## 1                                 From Justin to Kelly (2003)  4.159091
## 2                                 From Justin to Kelly (2003)  4.159091
## 3                                      PokÃ©mon Heroes (2003)  3.975410
## 4                           Shawshank Redemption, The (1994) -3.953996
## 5                           Shawshank Redemption, The (1994) -3.953996
## 6                           Shawshank Redemption, The (1994) -3.953996
## 7                           Shawshank Redemption, The (1994) -3.953996
## 8  Samurai Rebellion (JÃ´i-uchi: HairyÃ´ tsuma shimatsu) (1967) -3.944444
## 9                                    Godfather, The (1972) -3.912374
```

```
## 10                                          Godfather, The (1972) -3.912374
```

Here are the 10 best and 10 worst movies as per our estimate.

```
##Creating a dataset comtaining movie ID and titles only
movie_titles <- edx %>%
  select(movieId, title) %>%
  distinct()

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

```
## # A tibble: 10 x 2
##    title                                                 b_i
##    <chr>                                               <dbl>
##  1 Hellhounds on My Trail (1999)                        1.49
##  2 Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)                  1.49
##  3 Shadows of Forgotten Ancestors (1964)                1.49
##  4 Fighting Elegy (Kenka erejii) (1966)                 1.49
##  5 Sun Alley (Sonnenallee) (1999)                       1.49
##  6 Aerial, The (La Antena) (2007)                       1.49
##  7 Blue Light, The (Das Blaue Licht) (1932)             1.49
##  8 More (1998)                                          1.24
##  9 I'm Starting From Three (Ricomincio da Tre) (1981)   1.24
## 10 Human Condition II, The (Ningen no joken II) (1959)  1.24
```

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

```
## # A tibble: 10 x 2
##    title                                     b_i
##    <chr>                                   <dbl>
##  1 Besotted (2001)                         -3.01
##  2 Hi-Line, The (1999)                     -3.01
##  3 Under the Lighthouse Dancing (1997)     -3.01
##  4 Accused (Anklaget) (2005)               -3.01
##  5 Confessions of a Superhero (2007)       -3.01
##  6 War of the Worlds 2: The Next Wave (2008) -3.01
##  7 SuperBabies: Baby Geniuses 2 (2004)     -2.75
##  8 Disaster Movie (2008)                   -2.70
##  9 Hip Hop Witch, Da (2000)                -2.69
## 10 From Justin to Kelly (2003)             -2.67
```

If we see how often they were rated, almost all of them look biased with only few ratings.

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

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

```
## # A tibble: 10 x 3
##    title                                                 b_i     n
##    <chr>                                               <dbl> <int>
##  1 Hellhounds on My Trail (1999)                        1.49     1
##  2 Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)                  1.49     2
##  3 Shadows of Forgotten Ancestors (1964)                1.49     1
##  4 Fighting Elegy (Kenka erejii) (1966)                 1.49     1
##  5 Sun Alley (Sonnenallee) (1999)                       1.49     1
##  6 Aerial, The (La Antena) (2007)                       1.49     1
##  7 Blue Light, The (Das Blaue Licht) (1932)             1.49     1
##  8 More (1998)                                          1.24     6
##  9 I'm Starting From Three (Ricomincio da Tre) (1981)   1.24     2
## 10 Human Condition II, The (Ningen no joken II) (1959)  1.24     4
```

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

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

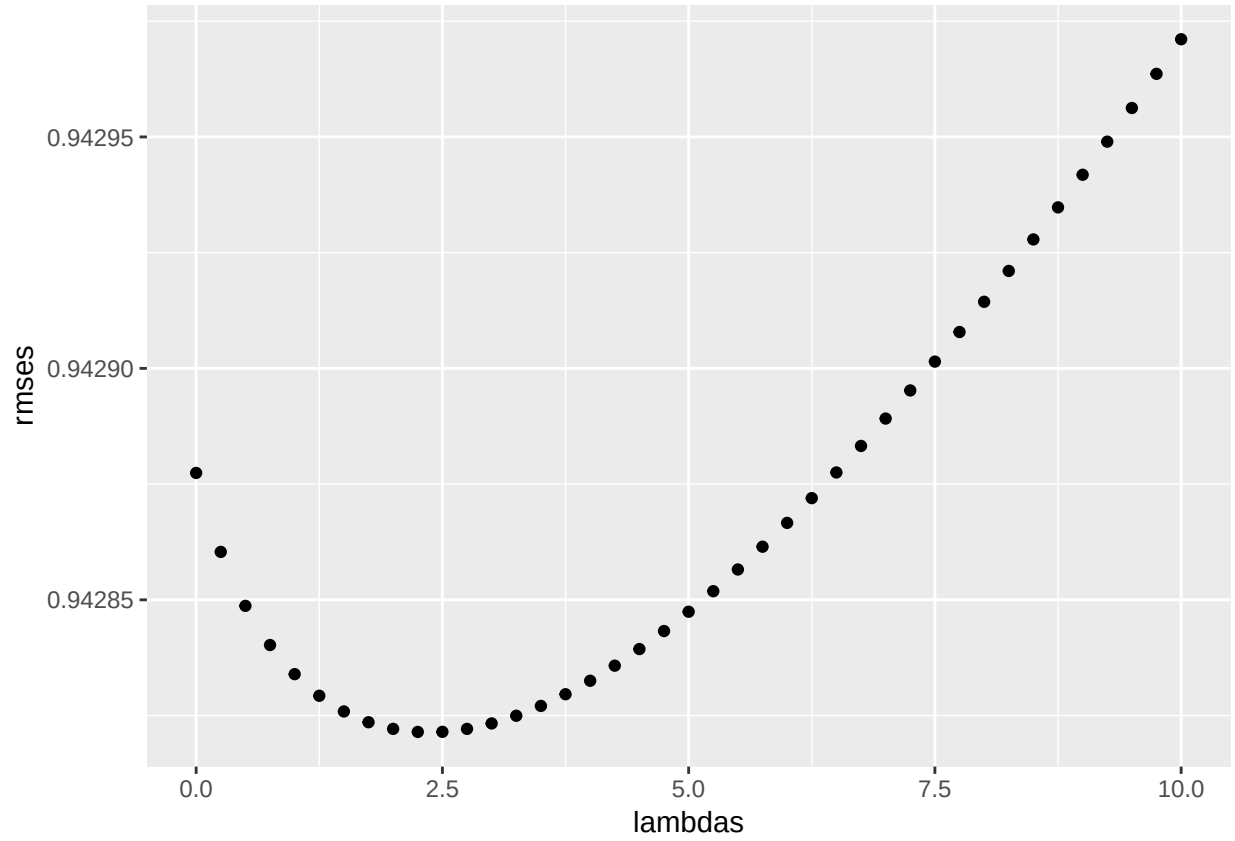
```
## # A tibble: 10 x 3
##    title                                       b_i     n
##    <chr>                                     <dbl> <int>
##  1 Besotted (2001)                           -3.01     2
##  2 Hi-Line, The (1999)                       -3.01     1
##  3 Under the Lighthouse Dancing (1997)       -3.01     1
##  4 Accused (Anklaget) (2005)                 -3.01     1
##  5 Confessions of a Superhero (2007)         -3.01     1
##  6 War of the Worlds 2: The Next Wave (2008) -3.01     2
##  7 SuperBabies: Baby Geniuses 2 (2004)       -2.75    47
##  8 Disaster Movie (2008)                     -2.70    24
##  9 Hip Hop Witch, Da (2000)                  -2.69    14
## 10 From Justin to Kelly (2003)               -2.67   176
```

This is why we will look at regularization, where we penalize large estimates that are formed using small sample sizes

penealized least squares approach

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

just_the_sum <- train %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())

rmses <- sapply(lambdas, function(l){
  predicted_ratings <- test %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
```

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

qplot(lambdas, rmses)
```




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

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

lambda is a tuning parameter which is set to be 1.75. We'll calculate RMSE using penalized regularization using this value.

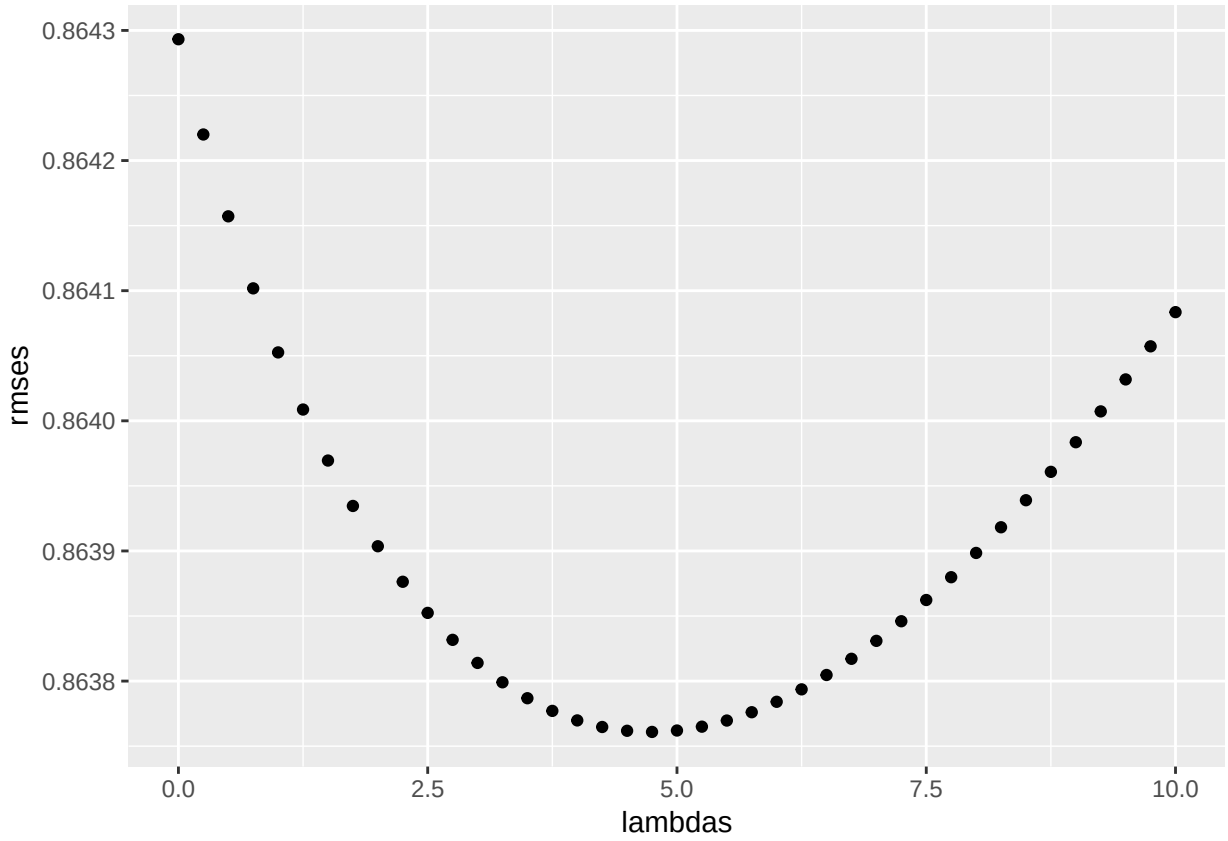
```
movie_reg_avgs <- train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())

predicted_ratings <- test %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)

model3_rmse <- RMSE(predicted_ratings, test$rating)
model3_rmse
```

```
## [1] 0.9428215
```

After tuning the value of lambda on train data set we will use cross-validation on test set to find the final

value of lambda

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

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

  mu <- mean(train$rating)

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

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

  predicted_ratings <-
    test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

    return(RMSE(predicted_ratings, test$rating))
})

qplot(lambdas, rmses)
```




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

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

```
min(rmses)
```

```
## [1] 0.8637609
```

RMSE is now 0.86 with a lambda value of 4.75.

Results

We will now use the final model to calculate RMSE on the validation data set.

```
b_i <- train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))

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

final_predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
```

```
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

final_rmse <- RMSE(final_predicted_ratings, validation$rating)
```

Conclusion

The final model accounting for regularized movie bias and regularized user bias yielded an RMSE of 0.8652203.