Harvardx Capstone MovieLens Project

Apurba Das

3/29/2020

Contents

1 Introduction **2**

1.1 Overview . . . 2

1.2 Procedure . . . 2

1.3 Dataset and variables . . . 2

2 Methods/ Analysis **4**

2.1 Data Exploration . . . 4

2.2 Modeling . . . 7

2.2.1 A first model . . . 7

2.2.2 Modeling movie effects . . . 8

2.2.3 Movie + User Effects Model . . . 9

2.2.4 Regularization . . . 11

2.2.4.1 Regularized Movie Effect Model . . . 11

2.2.4.2 Regularized Movie + User Effect Model . . . 12

3 Results **14**

4 Conclusion **14**

5 Reference **14**

1 Introduction

This project is the a part of Harvardx's Data Science course and serves as the first Capstone project. This section describes the dataset and variables, and summarizes the goal of the project and key steps that were performed to achieve it.

1.1 Overview

The aim of this project is to create a movie recommendation system using the 10M version of the MovieLens dataset (to make the computation a little easier). Recommendation systems use ratings that users have given items to make specific recommendations and our goal is to use and find the best among the created models using the residual mean squared error (RMSE). The lower the RMSE, the better the model. Our target is to achieve a RMSE < 0.86490.

1.2 Procedure

Firstly, the MovieLens data is downloaded and datasets are created. Then, we develop algorithms using the edx set. For a final test of these algorithms, prediction of movie ratings is done in the validation set as if they were unknown. RMSE will be used to evaluate how close our predictions are to the true values in the validation set.

As we train multiple machine learning algorithms using the inputs in one subset to predict movie ratings in the validation set, we also analyze and understand the dataset along the way.

The final goal is to come up with a machine learning algorithm that can achieve a RMSE of < 0.86490.

1.3 Dataset and variables

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.The Netflix data is not publicly available, but the GroupLens research lab114 generated their own database with over 20 million ratings for over 27,000 movies by more than 138,000 users. This MovieLens dataset is automatically downloaded in our code from https://grouplens.org/datasets/movielens/10m/ and http://files.grouplens.org/datasets/movielens/ml-10m.zip.

```
################################
# Create edx set, validation set
################################

# Note: this process could take a couple of minutes

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

```
## Loading required package: tidyverse
```

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

```
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.4
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

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

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

```
## Loading required package: caret
```

```
## Warning: package 'caret' was built under R version 3.6.3
```

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

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

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

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

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

```
## Warning: package 'data.table' was built under R version 3.6.3
```

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

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

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

```
if(!require(ggplot2)) install.packages("ggplot2")
if(!require(dplyr)) install.packages("dplyr")

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

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

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

```
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

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

```
# 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")
```

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

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

Note that we split the Movielens data into separate training (edx) and test sets (Validation) to design and test our algorithm. The validation data is not used for training the algorithm and is used for evaluating the RMSE of the algorithms. The Validation set is chosen as 10% of MovieLens data so that our analysis can take place with the larger dataset of 90% of MovieLens data.

2 Methods/ Analysis

This section documents the analysis/ methods used and presents the findings, along with supporting statistics and figures.

2.1 Data Exploration

We start by exploring the data a little more to increase our understanding on the dataset.

First, let's see the dimensions of the dataset.

```
dim(edx)
```

```
## [1] 9000055       6
```

The initial contents of edx can be seen with the head() function.

```
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1                 Comedy|Romance
## 2           Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5         Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7         Children|Comedy|Fantasy
```

We can see this table is in tidy format with thousands of rows:

```
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|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: Generation~ Action|Adventure|Dram~
##  6      1     355      5 838984474 Flintstones, The (199~ Children|Comedy|Fanta~
##  7      1     356      5 838983653 Forrest Gump (1994)    Comedy|Drama|Romance|~
##  8      1     362      5 838984885 Jungle Book, The (199~ Adventure|Children|Ro~
##  9      1     364      5 838983707 Lion King, The (1994)  Adventure|Animation|C~
## 10      1     370      5 838984596 Naked Gun 33 1/3: The~ Action|Comedy
## # ... with 9,000,045 more rows
```

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

We can see the number of unique users that provided ratings and how many unique movies were rated:

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

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

Let's look at some of the general properties of the data to better understand the challenges.

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

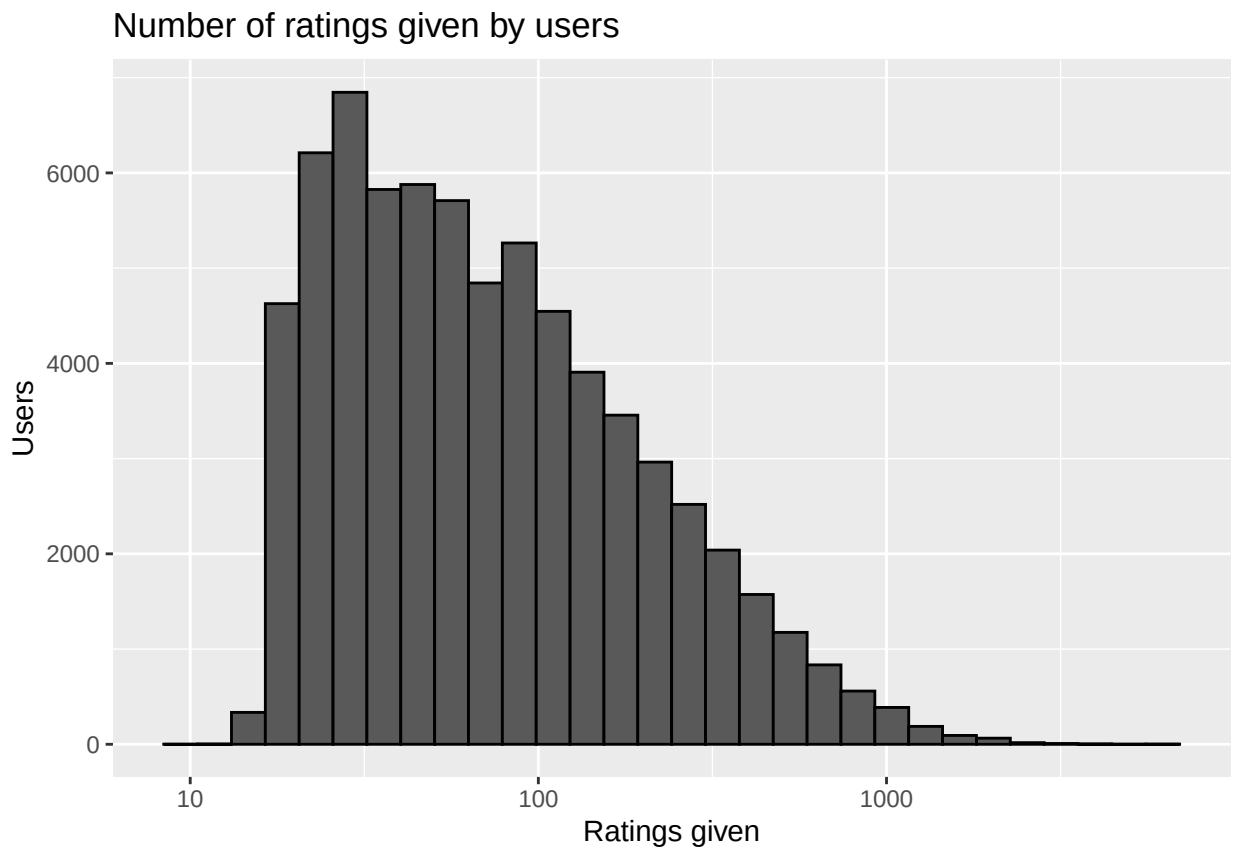
```
movie_ratings <- edx %>% group_by(movieId, title) %>%
  summarize(count = n()) %>%
  arrange(desc(count))

head(movie_ratings)
```

```
## # A tibble: 6 x 3
## # Groups:   movieId [6]
##   movieId title                                count
##     <dbl> <chr>                                <int>
## 1     296 Pulp Fiction (1994)                  31362
## 2     356 Forrest Gump (1994)                  31079
## 3     593 Silence of the Lambs, The (1991)     30382
## 4     480 Jurassic Park (1993)                 29360
## 5     318 Shawshank Redemption, The (1994)     28015
## 6     110 Braveheart (1995)                    26212
```

```
tail(movie_ratings)
```

```
## # A tibble: 6 x 3
## # Groups:   movieId [6]
##   movieId title                                        count
##     <dbl> <chr>                                        <int>
## 1   64953 Dirty Dozen, The: The Fatal Mission (1988)     1
## 2   64976 Hexed (1993)                                   1
## 3   65006 Impulse (2008)                                 1
## 4   65011 Zona Zamfirova (2002)                          1
## 5   65025 Double Dynamite (1951)                         1
## 6   65027 Death Kiss, The (1933)                         1
```

```
edx %>%
count(userId) %>%
ggplot(aes(n)) +
geom_histogram(bins = 30, color = "black") +
scale_x_log10() +
xlab("Ratings given") +
ylab("Users") +
ggtitle("Number of ratings given by users")
```



Further exploration will be done as we proceed with the different kinds of modeling.

2.2 Modeling

This section explains the process and techniques used and our modeling approach. The models will be evaluated using the RMSE function that is defined as:

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

2.2.1 A first model

We start by building the simplest possible recommendation system: we predict the same rating for all movies regardless of user. We know that the estimate that minimizes the RMSE is the least squares estimate of mu and, in this case, is the average of all ratings:

```
mu_hat <- mean(edx$rating)
mu_hat
```

```
## [1] 3.512465
```

If we predict all unknown ratings with mu_hat, we obtain the following RMSE:

```
naive_rmse <- RMSE(validation$rating, mu_hat)
naive_rmse
```

```
## [1] 1.061202
```

From looking at the distribution of ratings, we can visualize that this is the standard deviation of that distribution. We get a RMSE of about 1. The target is to achieve a RMSE < 0.86490. So we'll have to proceed to build a better model.

As we go along, there will be a need to compare different approaches. Hence, starting by creating a results table with this naive approach:

```
rmse_results <- tibble(Method = "Just the average", RMSE = naive_rmse)
```

Viewing the results obtained so far:

```
rmse_results %>% knitr::kable()
```

Method	RMSE
Just the average	1.061202

2.2.2 Modeling movie effects

We know from experience that some movies are just generally rated higher than others. This intuition, that different movies are rated differently, is confirmed by data. We can again use least squares to estimate the bias in the following way:

```
#fit <- lm(rating ~ as.factor(movieId), data = edx)
```

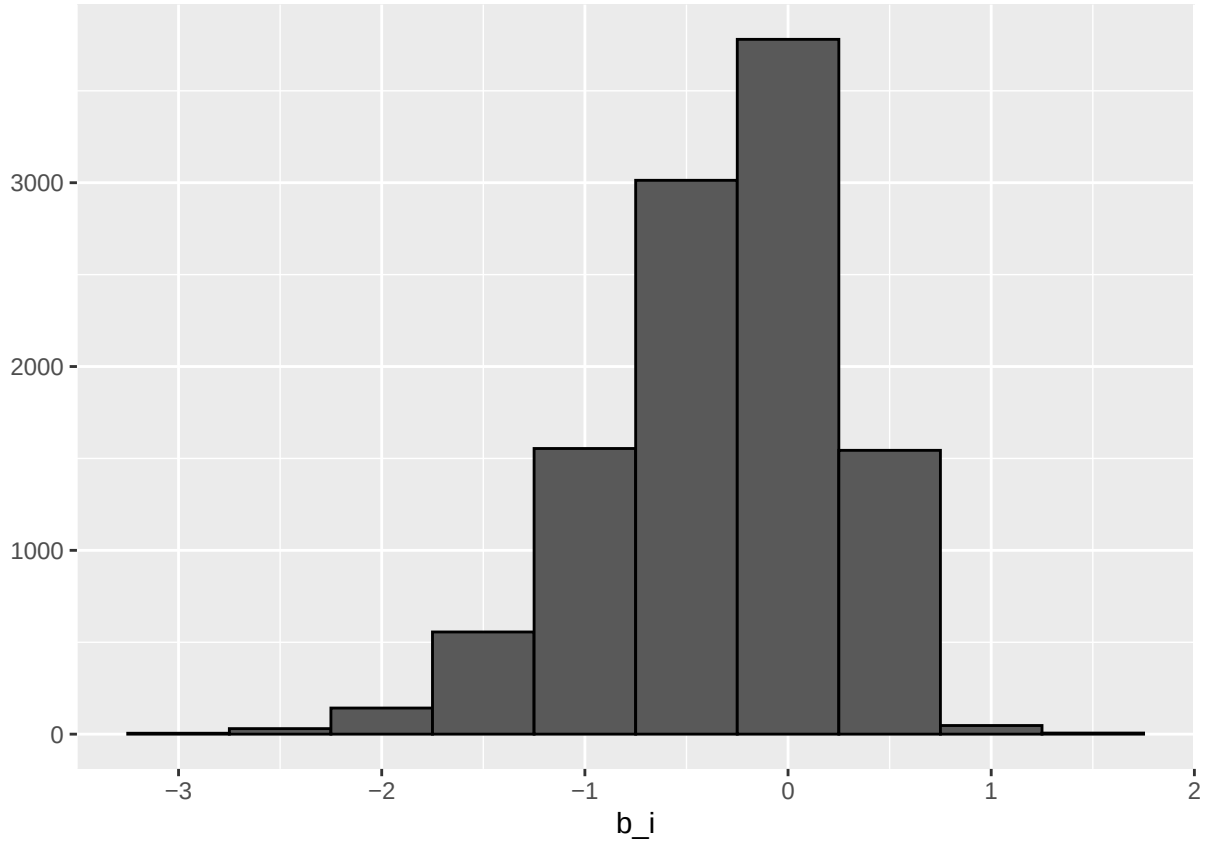
The lm() function will be very slow given the thousands of bias, hence the above is commented.

But in this case, we know that the least squares estimate is just the average of rating - mu for each movie. So it can be computed in the following way:

```
# Dropping the hat notation in the code going forward
mu <- mean(edx$rating)
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

We can see that these estimates vary substantially:

```
#Understanding these estimates
qplot(b_i, data = movie_avgs, bins = 10, color = I("black"))
```



Let's see how much our prediction improves:

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

# Calculating the RMSE using this model
rmse_me <- RMSE(predicted_ratings, validation$rating)
rmse_results <- add_row(rmse_results, Method = 'Movie Effect Model', RMSE = rmse_me)

# Viewing the results obtained so far
rmse_results %>% knitr::kable()
```

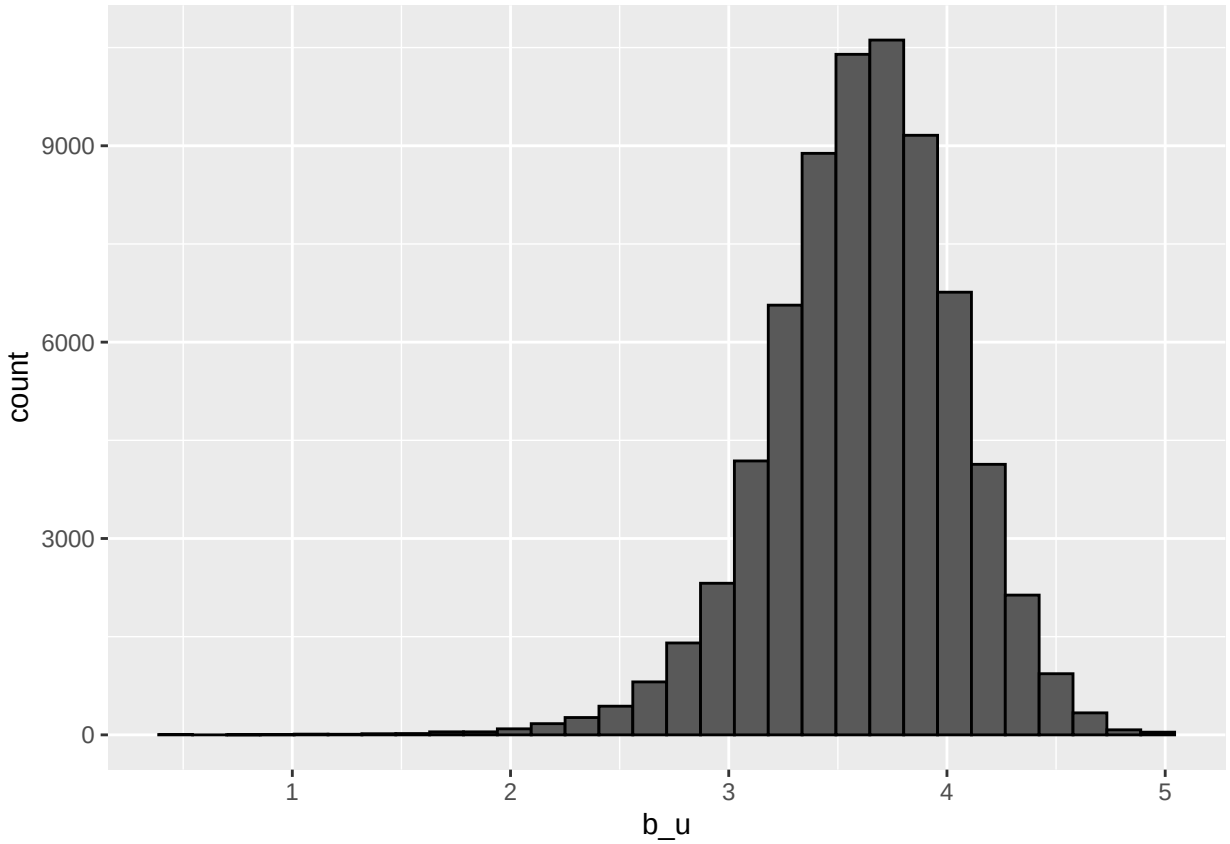
Method	RMSE
Just the average	1.0612018
Movie Effect Model	0.9439087

We already see an improvement, but we need to make it better.

2.2.3 Movie + User Effects Model

Let's compute the average rating for user u for those that have rated over 100 movies.

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



It is noticed that there is substantial variability across users as well: some users are very cranky and others love every movie. This implies that a further improvement to the model is possible, which we achieve using the following method:

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

We now construct predictors and see how much the RMSE improves:

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

# Calculating the RMSE using this model
```

```
rmse_ue <- RMSE(predicted_ratings, validation$rating)
rmse_results <- add_row(rmse_results, Method = 'Movie + User Effects Model', RMSE = rmse_ue)

# Viewing the results obtained so far
rmse_results %>% knitr::kable()
```

Method	RMSE
Just the average	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488

There is certainly an improvement.

2.2.4 Regularization

We have already seen that there are some movies that were rated by jusy 1 users and others by many more users. So, these movies are the obscure ones which have would very high probability of being the best or the worst movie. This is because with just a few users, we have more uncertainty. Therefore, larger estimates of bias, negative or positive, are more likely.

These are noisy estimates that we should not trust, especially when it comes to prediction. Large errors can increase our RMSE, so we would rather be conservative when unsure.

Thus, a penalty term needs to be introduced to regularize this effect. In short, regularization permits us to penalize large estimates that are formed using small sample sizes.

2.2.4.1 Regularized Movie Effect Model

The general idea behind regularization is to constrain the total variability of the effect sizes. This helps because in certain cases we have ratings of just 1 user and in others we have say, 100. When our sample size is very large, a case which will give us a stable estimate, then the penalty lambda is effectively ignored since the addition of the lambda has almost no effect on the sample size. However, when the sample size is small,then we lose the stability.

Let's compute these regularized estimates using lambda = 3

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

Now, we check if we have improved our results.

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

# Calculating the RMSE using this model
rmse_lambda3 <- RMSE(predicted_ratings, validation$rating)
rmse_results <- add_row(rmse_results, Method = 'Regularized Movie Effect Model', RMSE = rmse_lambda3)
```

```
# Viewing the results obtained so far
rmse_results %>% knitr::kable()
```

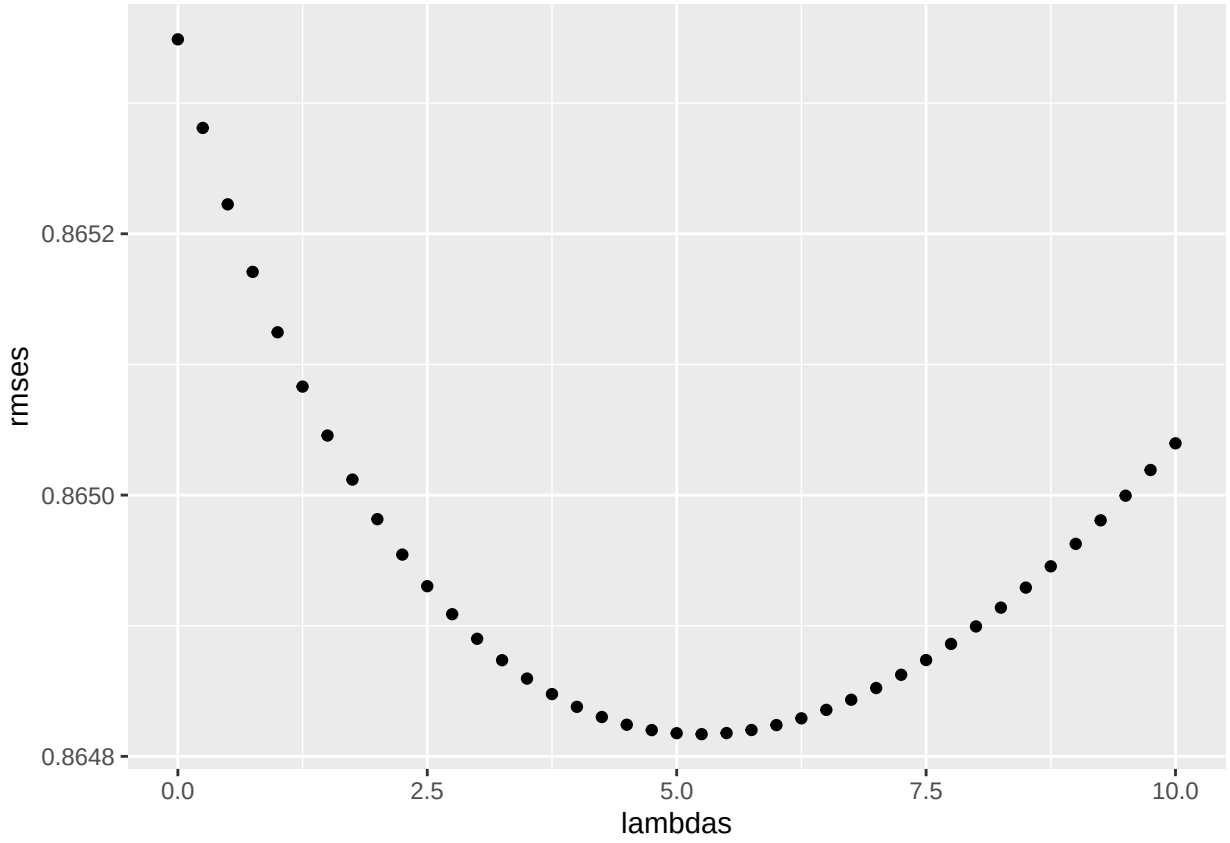
Method	RMSE
Just the average	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488
Regularized Movie Effect Model	0.9438538

There is a slight improvement over just the least squares estimates.

2.2.4.2 Regularized Movie + User Effect Model

Note that lambda or the penalty term is a tuning parameter. We can use cross-validation to choose it.

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

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

  mu <- mean(edx$rating)

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

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

  predicted_ratings <-
    validation %>%
    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, validation$rating))
})

qplot(lambdas, rmses)
```



For the final model, the optimal lambda is:

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

```
## [1] 5.25
```

The RMSE for the final model with optimised lambda is:

```
rmse_lambda_opt  <- min(rmses)

# Calculating the RMSE using this model
rmse_results <- add_row(rmse_results, Method = 'Regularized Movie + User Effect Model', RMSE = rmse_lamb

# Viewing the final comparison results:

rmse_results %>% knitr::kable()
```

Method	RMSE
Just the average	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488
Regularized Movie Effect Model	0.9438538
Regularized Movie + User Effect Model	0.8648170

3 Results

These are the final RMSE values of all the models constructed:

Method	RMSE
Just the average	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488
Regularized Movie Effect Model	0.9438538
Regularized Movie + User Effect Model	0.8648170

4 Conclusion

We have successfully constructed multiple models for our recommendation system. Hence, among the models designed, the Regularized Movie + User Effect Model gives the best RMSE of 0.8648170 and meets our expectation of delivering a RMSE < 0.86490. There can be other models built that take into account the week, genre or even by using matrix factorization, etc. These are bound to have a positive impact of reducing the RMSE and further improve our model.

5 Reference

We used https://rafalab.github.io/dsbook as a guide throughout the course of this project.