MovieLens Project

Mahip Sharma

6/9/2019

Introduction:

The purpose of this project is to build a machine learning algorithm to predict movie ratings based on given "movielens" data. We will generate and tune different models on training set (i.e. "edx" set) and will then apply it on test set to predict movie ratings(i.e. "validation" set). True ratings and predicted ratings will then be compared using RMSE function (Root Mean Square Error). After we develop different models and compare their RMSE values, we will be in a position to select best model that fits defined standards.

Data Extration: Create edx (train) and validation (test) sets

We have used following link to extract movie related data:
"http://files.grouplens.org/datasets/movielens/ml-10m.zip".

Extracted data in split into 2 sets, edx and validation. "edx" set contains 10% of the data that will be used for training our models and "validation" set contains 90% of the data that will be used for testing our developed models and selecting the best model.

Following code will create edx and validation sets:

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

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

```
## ── Attaching packages ─────────────────────────────────────────────
──────────────────────────────────────────── tidyverse 1.2.1 ──
```

```
## ✓ ggplot2 3.1.1     ✓ purrr   0.3.2
## ✓ tibble  2.1.1     ✓ dplyr   0.8.1
## ✓ tidyr   0.8.3     ✓ stringr 1.4.0
## ✓ readr   1.3.1     ✓ forcats 0.4.0
```

```
## ── Conflicts ────────────────────────────────────────────────────────
──────────────────────────────────────── tidyverse_conflicts() ──
## ✕ dplyr::filter() masks stats::filter()
## ✕ dplyr::lag()    masks stats::lag()
```

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

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

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

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

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

```r
# 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 <- read.table(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(movieI
d))[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) # if using R 3.6.0: set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, l
ist = 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", "genr
es")

edx <- rbind(edx, removed)

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

edx and validation set Data Exploration

We can see that edx dataset has 9000055 rows and 6 columns. Validation set has 999999 rows and 6 columns

```
dim(edx)

## [1] 9000055       6

#Column Names
names(edx)

## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"

dim(validation)

## [1] 999999      6

#Column Names
names(validation)

## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

Following are the unique # of movies and unique # of users in edx set:

```
n_distinct(edx$movieId) #distinct movies

## [1] 10677

n_distinct(edx$userId) # distinct users

## [1] 69878
```

Following are unique # of movies and unique # of users in validation set:

```
n_distinct(validation$movieId) #distinct movies

## [1] 9809

n_distinct(validation$userId) # distinct users
```

```
## [1] 68534
```

Different Models for Movie Rating Prediction:

RMSE FUNCTION:

Following RMSE (Root Mean Square Error) function will take 2 inputs, true and predicted ratings, and will return RMSE value. This function will be used to calculate RMSE values for all the models. Note that lower the RMSE, better the prediction

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

GLOBAL VARIABLES: MU and RMSE Result dataframe

Following variables, mu and rmse_result, will be used in all the models

```
mu <- mean(edx$rating) # Mean of ratings of all the movies :  mu = 3.512465
  rmse_result <- data.frame(method = 'datframe initialization' , RMSE = 0) #
Initialization of rmse_result dataframe
  # Remove 1st entry rmse_result, as its dummy
  rmse_result <- rmse_result[-1,0]
```

Model 1 : Naive Average Model

Our 1st model considers just the mean value of all the movies to predict movie ratings. Note that any value other than mu will increase RMSE value as mean provides the lowest RMSE value. Enter RMSE value in “rmse_result” dataset.

```
naive_rmse <- RMSE(validation$rating , mu) # naive_rmse = 1.061202

#Add results into rmse_result variable
  rmse_result <- bind_rows(rmse_result , data.frame(method = 'Model 1 : Naive
Average Model' , RMSE = naive_rmse))
  rmse_result %>% knitr::kable()
```

method	RMSE
Model 1 : Naive Average Model	1.061202

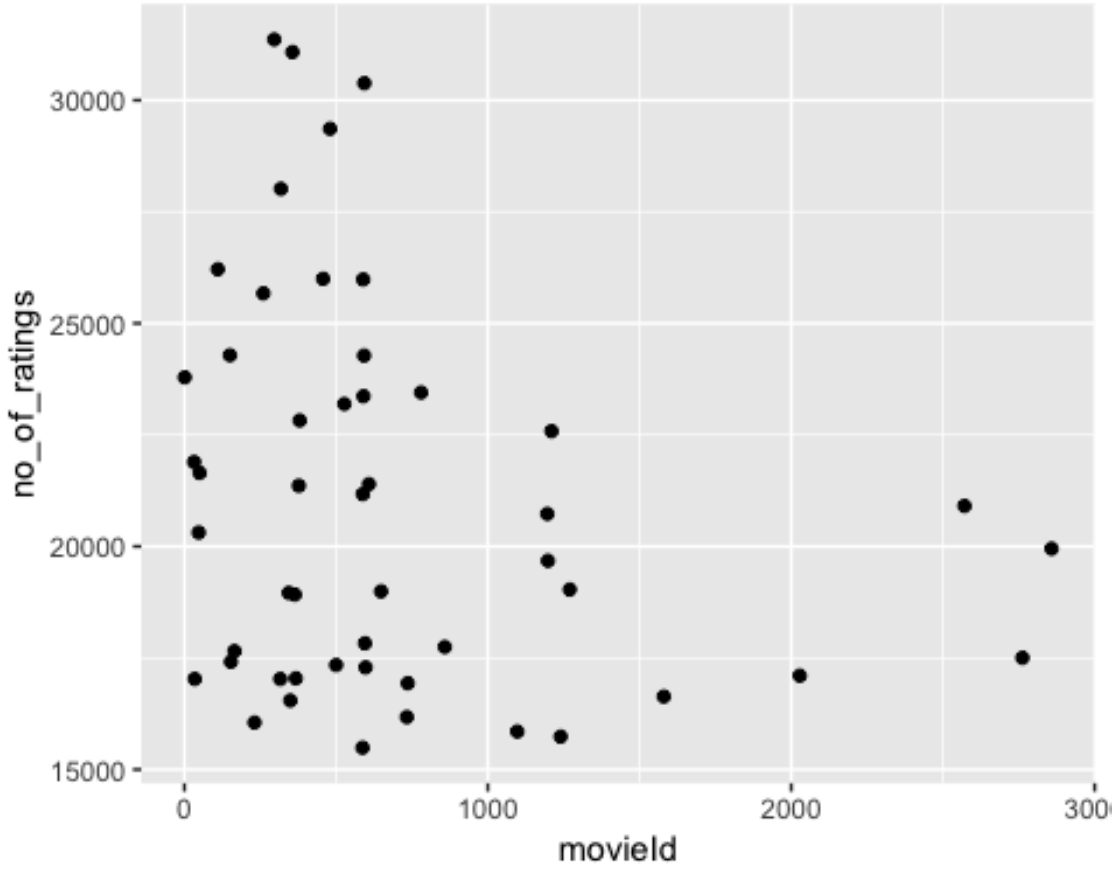
Model 1 has given RMSE value of 1.061202. Our RMSE value will reduce in following models

Model 2 : Movie Effect Model

Our 2nd model improves the 1st model by considering averages of individual movie ratings. Movie ratings vary across movies

Following plot will show that movie ratings may differ (values of only 50 rows are displayed):

```
edx %>% group_by(movieId) %>% summarize(no_of_ratings = n()) %>% top_n(.,50)
%>%  ggplot(aes(movieId,no_of_ratings)) + geom_point()

## Selecting by no_of_ratings
```

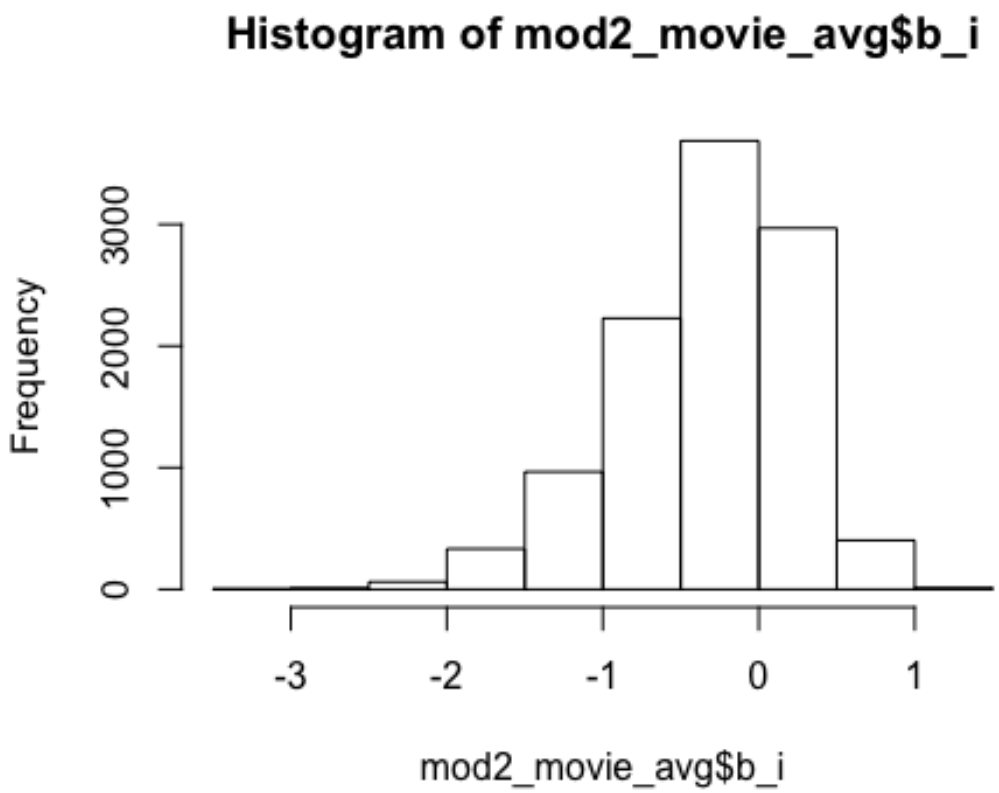


Following code will calculate movie averages on edx set. Let b_i denote movie average for a movie i. We need to subtract actual rating with mu (average of all movies), so that mu + b_i holds a maximum value of 5 and doesn't go beyond 5

```
mod2_movie_avg <- edx %>% group_by(movieId) %>% summarise(b_i = mean(rating -
mu))
```

Following graph displays distribution of b_i values:

```
hist(mod2_movie_avg$b_i)
```




```
# note that mu + b_i value can go upto 5
  max(mod2_movie_avg$b_i) + mu # 5

## [1] 5
```

Following code will predict movie ratings using validation set, using b_i that is calculated on edx set (i.e. training set)

```
mode2_predicted_ratings <- mu + validation %>% left_join(mod2_movie_avg, by =
'movieId') %>%
    .$b_i
```

Following code will calculate RMSE based on predicted ratings and will insert RMSE into rmse_result.

```
mod2_rmse <- RMSE(validation$rating , mode2_predicted_ratings) # model 2 rmse
= 0.9439087
  rmse_result <- bind_rows(rmse_result , data.frame(method = 'Model 2 : Movie
Effect Model' , RMSE = mod2_rmse))
```

```
## Warning in bind_rows_(x, .id): Unequal factor levels: coercing to characte
r

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector
```

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

method	RMSE
Model 1 : Naive Average Model	1.0612018
Model 2 : Movie Effect Model	0.9439087

We can see that Model 2 gave us better RMSE value (0.9439087) as compared to our 1st Model.

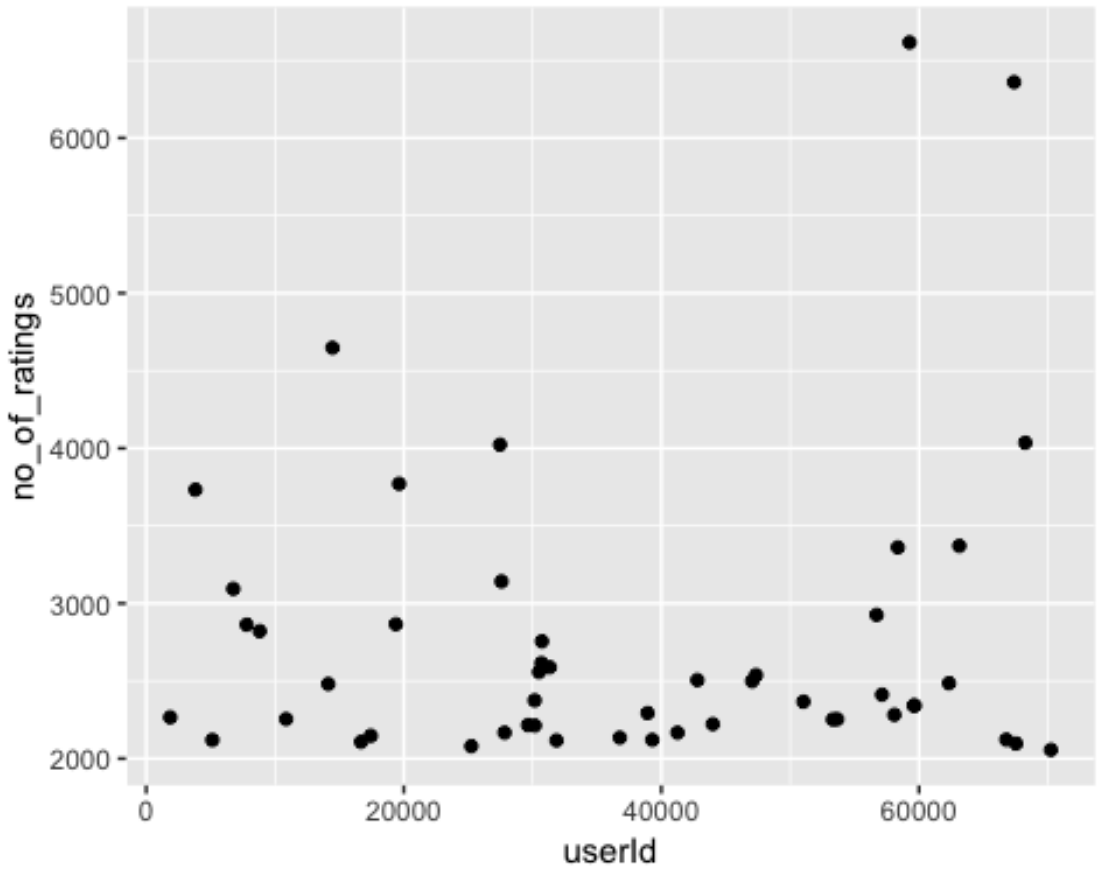
Model 3 : Movie + User Effect Model

We can see that there is scope of improvement in Model 2. Like different movies have different ratings, different users also give different ratings to movies. If we consider this factor in our equation, then RMSE reduces further

Plot to show that different users can give different movies ratings (values of only 50 rows are displayed):

```
edx %>% group_by(userId) %>% summarize(no_of_ratings = n()) %>% top_n(.,50) %
>%
    ggplot(aes(userId,no_of_ratings)) + geom_point()
```

```
## Selecting by no_of_ratings
```

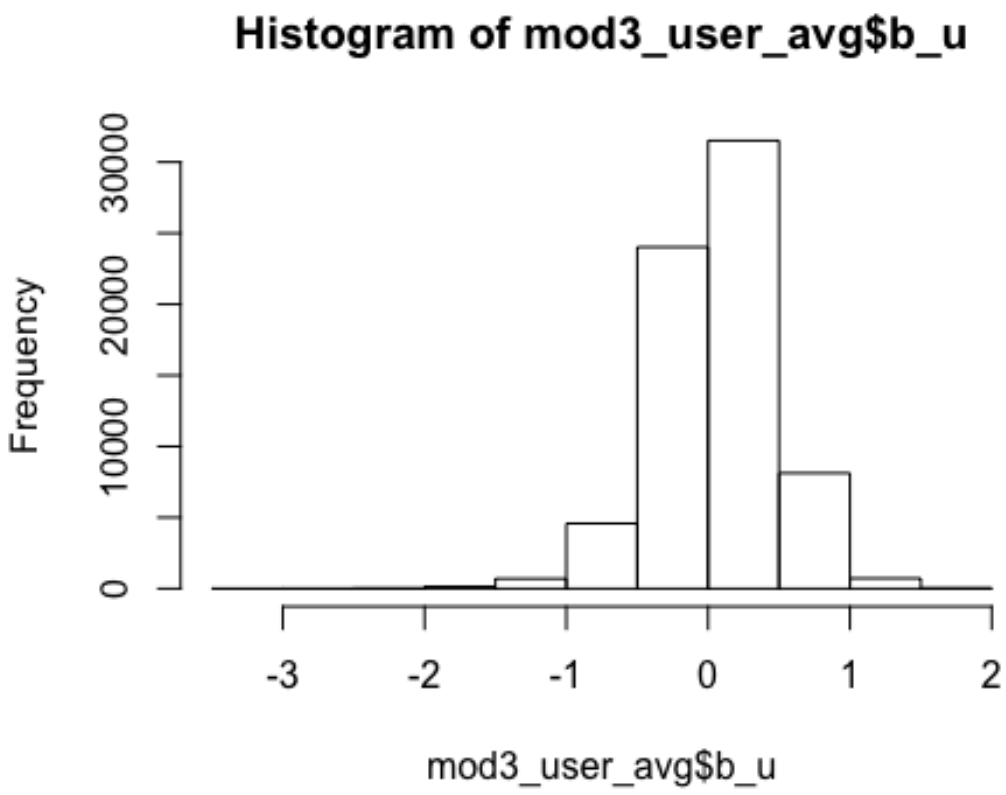


Following code will calculate movie averages on edx set. Let b_i denote movie average for a movie i. We need to subtract actual rating with mu (average of all movies), so that mu + b_i holds a maximum value of 5 and doesn't go beyond 5.

```
mod3_movie_avg <- edx %>% group_by(movieId) %>% summarise(b_i = mean(rating -
mu))
```

Following code will calculate user averages on edx set. Let b_u denote user average for a user u. We will subtract actual rating with mu (average of all movies) and b_i (average of individual movies) to calculate b_u

```
mod3_user_avg <-  edx %>% left_join(mod3_movie_avg, by = 'movieId') %>%
    group_by(userId) %>% summarize(b_u = mean(rating - mu - b_i))

# Folloing graph will display distribution of b_u
  hist(mod3_user_avg$b_u)
```



Following code will predict movie ratings on validation set, using b_i and b_u that is calculated on edx set (i.e. training set)

```
mod3_predicted_ratings <- validation %>% left_join(mod3_movie_avg, by = 'movi
eId') %>%
    left_join(mod3_user_avg, by = 'userId') %>%
    mutate(pred = mu + b_i + b_u) %>% .$pred
```

Following code will calculate RMSE based on predicted ratings and will insert RMSE into rmse_result.

```
mod3_rmse <- RMSE(validation$rating , mod3_predicted_ratings) # model 3 rmse
= 0.8653488
  rmse_result <- bind_rows(rmse_result , data.frame(method = 'Model 3 : Movie
+ User Effect Model' , RMSE = mod3_rmse))

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

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

method	RMSE
Model 1 : Naive Average Model	1.0612018

Model 2 : Movie Effect Model	0.9439087
Model 3 : Movie + User Effect Model	0.8653488

We can see that after considering user specific ratings in our model, RMSE has reduced further (0.8653488).

Note that this value (0.8653488) is better than the value stated in “MovieLens Grading Rubric” (0.87750).

We will try to improve our model to reduce RMSE value even futher.

Model 4 : Regularized Movie and User Effect Model

We will use regularization in this model. This model will penalize larger estimates that come from small sample size. Penalty will be applied to both b_u and b_i. Lambda is the variable that will be used to penalize small sample size.

Following code displays 10 movies with highest b_i value. Note that these movies were rated very few times that made them appear in the top 10 positions

```
mod4_movie_avg <- edx %>% group_by(movieId) %>% summarise(b_i = mean(rating -
mu))
    movie_title <- edx %>% select(movieId,title) %>% distinct()
temp1 <- edx %>% left_join(mod4_movie_avg , by = 'movieId') %>%
    left_join(movie_title, by='movieId') %>%
    arrange(desc(b_i)) %>%
    distinct(title.x,b_i) %>%
    select(title.x,b_i) %>%
    slice(1:10) %>%
    rename(title = title.x)
edx %>% group_by(title) %>% mutate(n = n()) %>% inner_join(temp1, by = 'title
') %>%
    arrange(desc(b_i)) %>%
    distinct(title,b_i,n)
```

```
## # A tibble: 10 x 3
## # Groups:   title [10]
##    title                                                    b_i
n
##    <chr>                                                  <dbl> <int
>
##  1 Sun Alley (Sonnenallee) (1999)                          1.49
1
##  2 Fighting Elegy (Kenka erejii) (1966)                    1.49
1
##  3 Satan's Tango (Sátántangó) (1994)                       1.49
2
##  4 Shadows of Forgotten Ancestors (1964)                   1.49
```

```
1
##  5 Hellhounds on My Trail (1999)                                    1.49
1
##  6 Blue Light, The (Das Blaue Licht) (1932)                         1.49
1
##  7 Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko…  1.24
4
##  8 Constantine's Sword (2007)                                       1.24
2
##  9 Human Condition II, The (Ningen no joken II) (1959)              1.24
4
## 10 Human Condition III, The (Ningen no joken III) (1961)            1.24
4
```

Small number of movie rating sample size has bumped up rating of these movies.

Following code displays 10 movies with lowest b_i value, again these movies were rated very few times that made them appear in the last 10 positions

```
temp2 <- edx %>% left_join(mod4_movie_avg , by = 'movieId') %>%
left_join(movie_title, by='movieId') %>%
    arrange((b_i)) %>%
    distinct(title.x,b_i) %>%
    select(title.x,b_i) %>%
    slice(1:10) %>%
    rename(title = title.x)
edx %>% group_by(title) %>% mutate(n = n()) %>% inner_join(temp2, by = 'title
') %>%
    arrange((b_i)) %>%
    distinct(title,b_i,n)
```

```
## # A tibble: 10 x 3
## # Groups:   title [10]
##    title                                          b_i     n
##    <chr>                                        <dbl> <int>
##  1 Besotted (2001)                              -3.01     2
##  2 Accused (Anklaget) (2005)                    -3.01     1
##  3 War of the Worlds 2: The Next Wave (2008)    -3.01     2
##  4 Confessions of a Superhero (2007)            -3.01     1
##  5 Hi-Line, The (1999)                          -3.01     1
##  6 SuperBabies: Baby Geniuses 2 (2004)          -2.72    56
##  7 Hip Hop Witch, Da (2000)                     -2.69    14
##  8 Disaster Movie (2008)                        -2.65    32
##  9 From Justin to Kelly (2003)                  -2.61   199
## 10 Stacy's Knights (1982)                       -2.51     1
```

Movie regularization

To remove above displayed noise that gives false impression of the movie rankings, we will use following code that penalizes this behaviour.

Following code will calculate movie averages using lambda as 2.5 (to start with) on edx set (i.e. training set).

```
lambda <- 2.5
  mod4_movie_avg_reg <- edx %>% group_by(movieId) %>%
    summarize(b_i_lambda = sum(rating - mu) / (n() + lambda), n_i = n())
```

Note that if movie rating sample size is large, than lambda will not have much effect on the equation/b_i, but if movie rating sample size is small, than lambda will impact the equation by reducing b_i value i.e. penalizing b_i if movie received lesser number of ratings. This will ensure that b_i now predicts movies correctly. Following code displays top 10 movies and last 10 movies after using regularization on b_i (or movies):

```
# Top 10 movies based on regularized movie ratings

Temp3 <- edx %>% left_join(mod4_movie_avg_reg , by = 'movieId') %>%
    left_join(movie_title, by='movieId') %>%
    arrange(desc(b_i_lambda)) %>%
    distinct(title.x,b_i_lambda) %>%
    select(title.x,b_i_lambda) %>%
    slice(1:10) %>%
    rename(title = title.x)
edx %>% group_by(title) %>% mutate(n = n()) %>% inner_join(Temp3, by = 'title
') %>%
    arrange(desc(b_i_lambda)) %>%
    distinct(title,b_i_lambda,n)

## # A tibble: 10 x 3
## # Groups:   title [10]
##    title                                           b_i_lambda     n
##    <chr>                                                <dbl> <int>
##  1 Shawshank Redemption, The (1994)                     0.943 28015
##  2 Godfather, The (1972)                                0.903 17747
##  3 More (1998)                                          0.886     7
##  4 Usual Suspects, The (1995)                           0.853 21648
##  5 Schindler's List (1993)                              0.851 23193
##  6 Casablanca (1942)                                    0.808 11232
##  7 Rear Window (1954)                                   0.806  7935
##  8 Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)        0.803  2922
##  9 Third Man, The (1949)                                0.798  2967
## 10 Double Indemnity (1944)                              0.797  2154

# Last 10 movies based on regularized movie ratings

Temp4 <- edx %>% left_join(mod4_movie_avg_reg , by = 'movieId') %>%
```

```
    left_join(movie_title, by='movieId') %>%
    arrange(b_i_lambda) %>%
    distinct(title.x,b_i_lambda) %>%
    select(title.x,b_i_lambda) %>%
    slice(1:10) %>%
    rename(title = title.x)
edx %>% group_by(title) %>% mutate(n = n()) %>% inner_join(Temp4, by = 'title
') %>%
    arrange(desc(b_i_lambda)) %>%
    distinct(title,b_i_lambda,n)

## # A tibble: 10 x 3
## # Groups:   title [10]
##    title                                                b_i_lambda     n
##    <chr>                                                     <dbl> <int>
##  1 Hip Hop Witch, Da (2000)                                  -2.28    14
##  2 Barney's Great Adventure (1998)                           -2.30   208
##  3 Gigli (2003)                                              -2.30   313
##  4 Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie) (2002)       -2.31   202
##  5 Glitter (2001)                                            -2.32   339
##  6 Carnosaur 3: Primal Species (1996)                        -2.34    68
##  7 Pokémon Heroes (2003)                                     -2.44   137
##  8 Disaster Movie (2008)                                     -2.46    32
##  9 From Justin to Kelly (2003)                               -2.58   199
## 10 SuperBabies: Baby Geniuses 2 (2004)                       -2.60    56
```

Following code will predict movie ratings for validation set (i.e. test set) using movie averages calculated in above step.

```
mod4_predicted_ratings <- validation %>% left_join(mod4_movie_avg_reg, by = '
movieId') %>%
    mutate(pred = mu + b_i_lambda) %>% .$pred
```

Following code will give us RMSE value based on our predicted movie ratings.

```
mod4_rmse <- RMSE(validation$rating,mod4_predicted_ratings) # 0.9438521
```

Tuning lambda:

To tune lambda i.e. to find a value of lambda that will give us the least value of RMSE, we will use following code

lambdas will have a range of values from 0 to 10 with an interval of 0.25.

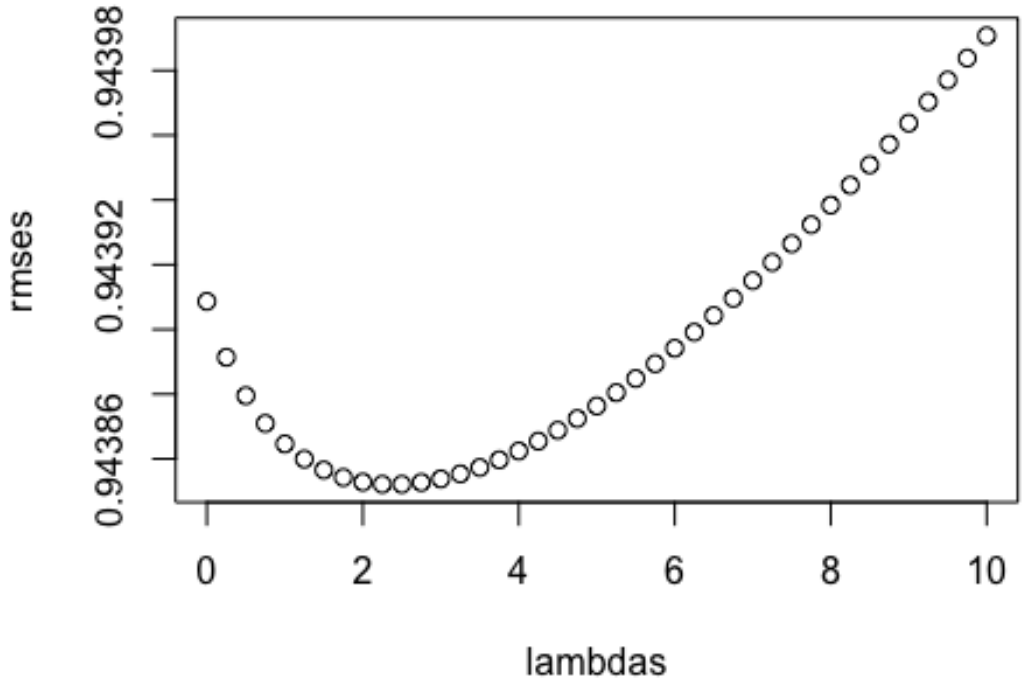
```
lambdas <- seq(0,10,0.25)

    sum_of_numerator_lambda <- edx %>% group_by(movieId) %>%
      summarize(s = sum(rating - mu) , n_i = n())
```

Following code will give RMSE value for different values of lambda.

```
rmses <- sapply(lambdas, function(l){
      predicted_ratings <- validation %>% left_join(sum_of_numerator_lambda,
by = 'movieId') %>%
        mutate(b_i_lambda = s / (n_i + l)) %>%
        mutate(pred = mu + b_i_lambda) %>% .$pred

      RMSE(validation$rating , predicted_ratings)
    })
```

Following plot will give us details on which lambda value gives us the least RMSE.

```
plot(lambdas,rmses)
```




```
    # Following code will give us the best lambda value
    lambdas[which.min(rmses)] # 2.5
```

```
## [1] 2.5
```

```
    # Following code will provide use with least RMSE value when lambda is 2.
5
    min(rmses) # 0.9438521
```

```
## [1] 0.9438521
```

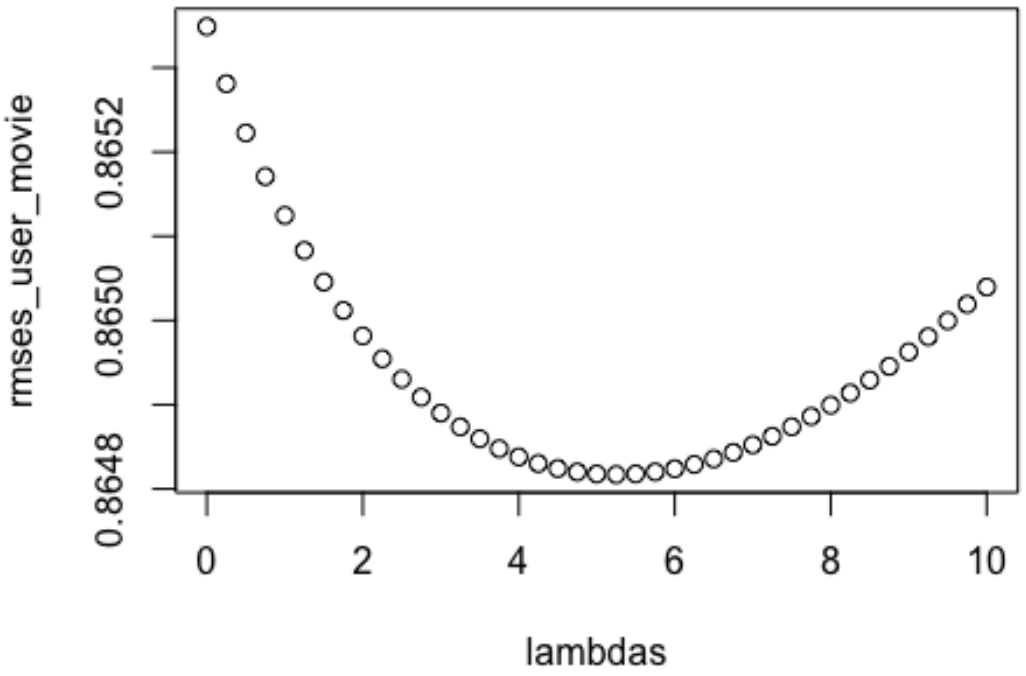
Movie and User regularization:

Above method didnt give us RMSE lesser than Model 3, but we have not removed user noise in our model. Following code will remove user noise as well. We will tune lambdas in below code

```
# lambdas will have a range of values from 0 to 10 with interval of 0.25
    lambdas <- seq(0,10,0.25)

# Following code will apply all the values of lambda and will calculate RMSE
based on each
    # each lambda value
      rmses_user_movie <- sapply(lambdas, function(l){

        # Following code calculates b_i by penalizing movies with lower ratin
gs using edx set
        b_i <- edx %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu)
/ (n() + l))

        # Following code calculates b_u by penalizing users that have rated l
ess number of
        # movies using edx set
        b_u <- edx %>% left_join(b_i, by='movieId') %>% group_by(userId) %>%
          summarize(b_u = sum(rating - b_i - mu) / (n() + l) )

        # Following code will predit movie ratings using b_i and b_u calculat
ed in above code
        # on validation set
        predicted_ratings <- validation %>% left_join(b_i, by = 'movieId') %>
%
          left_join(b_u , by = 'userId') %>%
          mutate(pred = mu + b_i + b_u) %>% .$pred

        # Following code returns RMSE value
        return(RMSE(validation$rating , predicted_ratings))

      })
```

Following plot will give us details on which lambda value gives us the least RMSE.

```
plot(lambdas,rmses_user_movie)
```



Following code will give us the best lambda value.

```
lambdas[which.min(rmses_user_movie)] # 5.25
```

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

Following code will give us the RMSE value when lambda is 5.25 (i.e. lowest RMSE).

```
min(rmses_user_movie) # 0.864817
```

```
## [1] 0.864817
```

Add results into rmse_result:

```
rmse_result <- bind_rows(rmse_result , data.frame(method = 'Model 4 : Regular
ized Movie and User Effect Model' , RMSE = min(rmses_user_movie)))
```

```
## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector
```

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

method	RMSE
Model 1 : Naive Average Model	1.0612018

Model 2 : Movie Effect Model	0.9439087
Model 3 : Movie + User Effect Model	0.8653488
Model 4 : Regularized Movie and User Effect Model	0.8648170

Result:

Following code shows RMSE values of all the 4 models

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

method	RMSE
Model 1 : Naive Average Model	1.0612018
Model 2 : Movie Effect Model	0.9439087
Model 3 : Movie + User Effect Model	0.8653488
Model 4 : Regularized Movie and User Effect Model	0.8648170

Following code shows that the best mode with least RMSE value (0.864817) is Mode 4 (Regularization on users and movies)

```
rmse_result[which.min(rmse_result$RMSE),]
```

```
##                                                 method     RMSE
## 4 Model 4 : Regularized Movie and User Effect Model 0.864817
```

Conclusion:

We can see that the Regularization model (considering Movie and User) gave us the least RMSE value and thus proves to be the best model that satisfies MovieLens Grading Rubric criteria. Final result set also reflects that Model 3 (Movie and User Effect Model) is also within acceptable limits.

So, I recommend to use Model 4 or Model 3 to predict movie ratings.