Movielens~ Movie Ratings Prediction

Hrithik Muskan

04/01/2021

Index:

Introduction

MovieLens is a web-based recommender system and virtual community that recommends movies for its users to watch, based on their film preferences using collaborative filtering of members' movie ratings and movie reviews. It contains about 11 million ratings for about 8500 movies.[1] MovieLens was created in 1997 by GroupLens Research, a research lab in the Department of Computer Science and Engineering at the University of Minnesota,[2] in order to gather research data on personalized recommendations. This project is on a rating prediction system using the dataset from grouplens.

Dataset Generation

```
##########################################################
# Create edx set, validation set (final hold-out test 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 ---------------------------------------------------------------------------------

## 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 ------------------------------------------------------------------------------------ tid
## 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

## 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
```

```
##
## 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(tidyverse)
library(caret)
library(data.table)

# 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")

# 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")

# 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 <- 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)
```

Goal:

Goal of this project is to create a movie recommendation system using machine learning and data analytics and to achieve a RMSE < 0.86490 # Data Exploration:

Glimpse of the data:

```
glimpse(edx)
```

```
## Rows: 9,000,055
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 42...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ...
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83...
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)",...
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|...
```

Distinct Movies

```
n_distinct(edx$movieId)
```

```
## [1] 10677
```

Distinct Users

```
n_distinct(edx$userId)
```

```
## [1] 69878
```

Movie Genres and Ratings

```
genres = c("Drama", "Comedy", "Thriller", "Romance")
sapply(genres, function(g) {
    sum(str_detect(edx$genres, g))
})
```

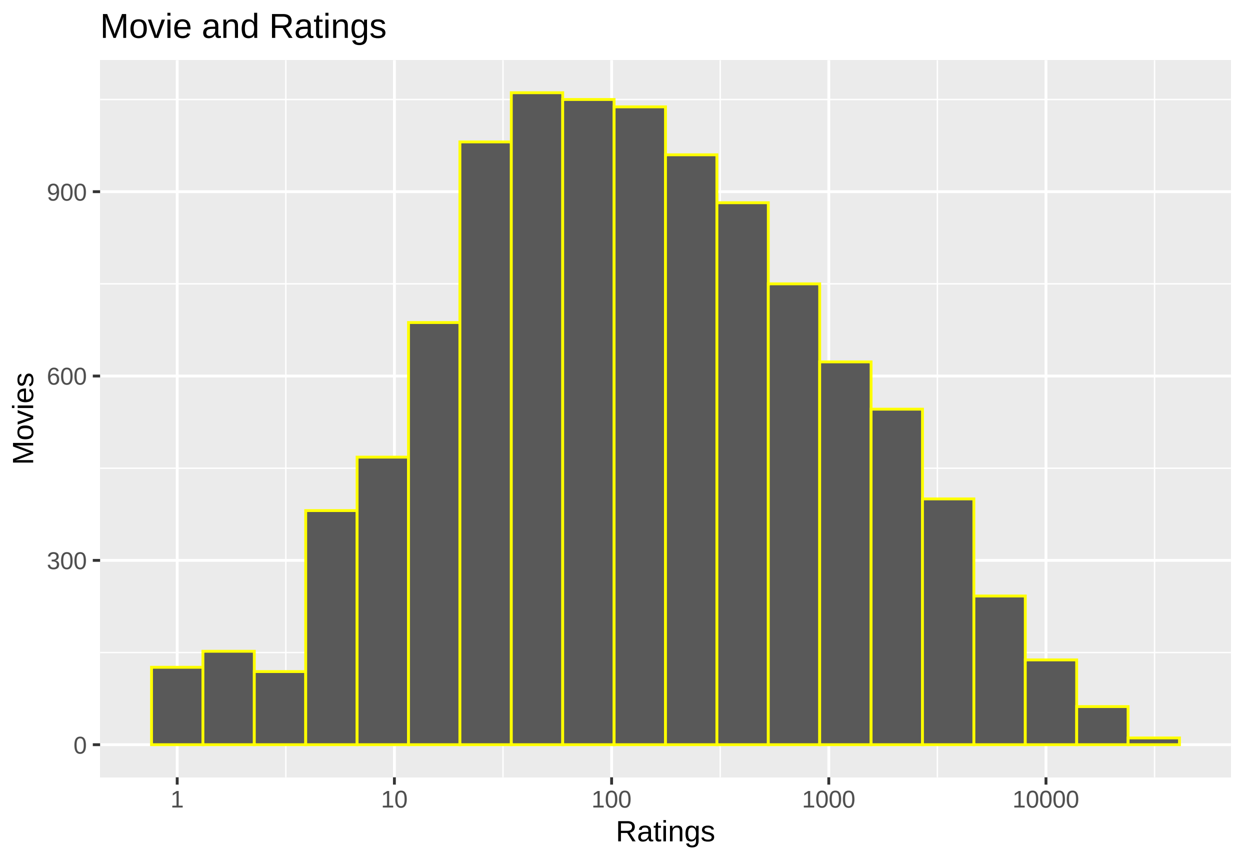
```
##    Drama   Comedy Thriller  Romance
##  3910127  3540930  2325899  1712100
```

Data Visualisation:

Let's see a few important visualizations:

Number of Movies and Number of Ratings

```
edx %>%
count(movieId) %>%
ggplot(aes(n)) +
geom_histogram(bins = 20, color = "yellow") +
scale_x_log10() +
xlab("Ratings") +
  ylab("Movies") +
ggtitle("Movie and Ratings")
```

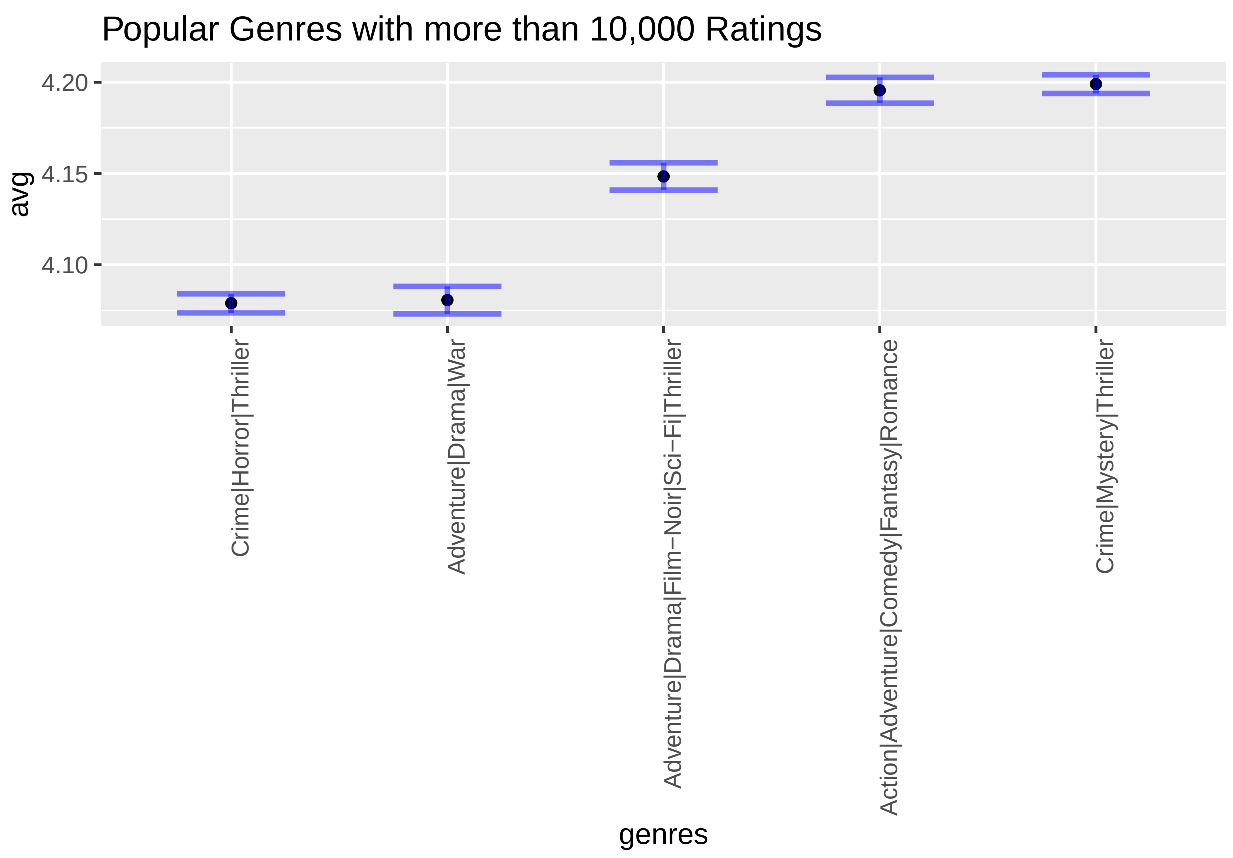


Genres and Ratings

```
popular_genres <- edx %>%group_by(genres) %>% summarize(n = n(),sd = sd(rating) ,se  = sd/sqrt(n),avg =
```

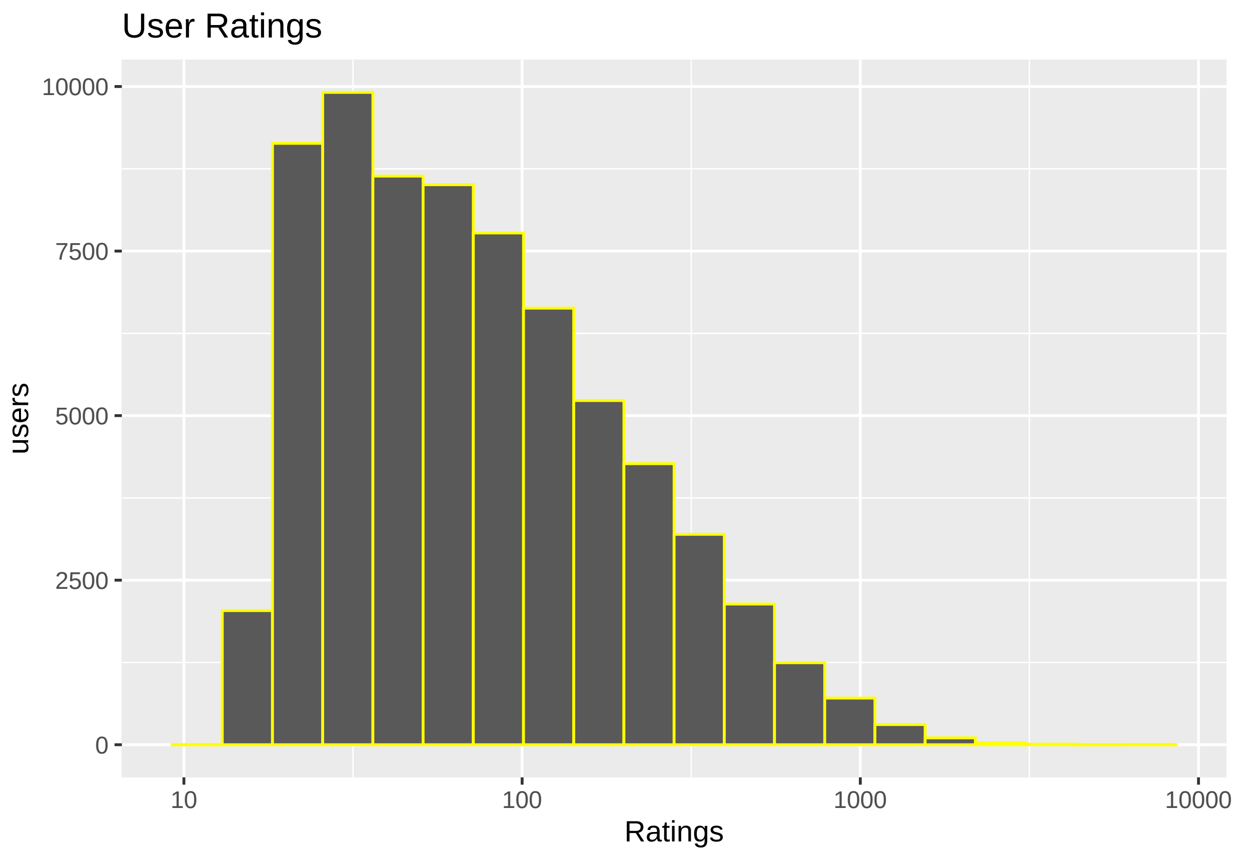
```
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
```

```
plot1 <- popular_genres %>% ggplot (aes(x=genres, y=avg)) +
    geom_point() +
    geom_errorbar(aes(ymin=avg-se, ymax=avg+se), width=0.5, colour="blue", alpha=0.5, size=1) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
    ggtitle("Popular Genres with more than 10,000 Ratings")
plot1
```



# User Ratings

```
edx %>%count(userId) %>%ggplot(aes(n)) +
geom_histogram(bins = 20, color = "yellow") +scale_x_log10() +
xlab("Ratings") + ylab("users") +ggtitle("User Ratings")
```



# Insights: 1. The movie genre has a good effect o ratings, some genres are very popular and thus get ratings more than 3.5 2. Users also have great effect, more users have given less number of ratings and obviously we should set a band for considering a user's rating. (For example: Users who have given more than 15 ratings should be considered only to account for prediction)(bias)

Modeling

Accuracy Check

This will be done by using the RMSE method , where we calcluate the root of mean of the squared error that in prediction with respect to actual value. Let's define our accuracy check function using RMSE

```
Accuracy <- function(actual_ratings, predicted_ratings){
  sqrt(mean((actual_ratings - predicted_ratings)^2))
}
```

Possible Bias:

1. User Bias = mean(user)-mean(allusers)
2. Movie Bias= mean(movie)-mean(allmovie)

These bias terms will help us achieve better results.

Approach-1:

Naive Average

```
avg<-mean(edx$rating)
results1<- Accuracy(validation$rating,avg)
results1
```

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

Approach-2:

Deviation from mean rating, bias due to movies.

```
movie_dev <- edx %>%
  group_by(movieId) %>%
  summarize(movie_bias = mean(rating - avg))
```

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

```
movie_dev
```

```
## # A tibble: 10,677 x 2
##    movieId movie_bias
##      <dbl>      <dbl>
##  1       1     0.415
##  2       2    -0.307
##  3       3    -0.365
##  4       4    -0.648
##  5       5    -0.444
##  6       6     0.303
##  7       7    -0.154
##  8       8    -0.378
##  9       9    -0.515
## 10      10    -0.0866
## # ... with 10,667 more rows
```

Applying the bias factor:

```
approach_2 <- avg +  validation %>%
  left_join(movie_dev, by='movieId') %>%
  pull(movie_bias)
results2 <- Accuracy(approach_2, validation$rating)
results2
```

```
## [1] 0.9439087
```

Approach-3

Considering the user deviation or bias:

```
user_dev<- edx %>%
  left_join(movie_dev, by='movieId') %>%
  group_by(userId) %>%
  summarize(user_bias = mean(rating - avg - movie_bias))
```

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

```
predicted_ratings <- validation%>%
  left_join(movie_dev, by='movieId') %>%
  left_join(user_dev, by='userId') %>%
  mutate(pred = avg + user_bias) %>%
  pull(pred)

results3<- Accuracy(predicted_ratings, validation$rating)
results3
```

```
## [1] 0.9947953
```

Approach-4

Considering both User as well as movie bias

```
predicted_ratings <- validation%>%
  left_join(movie_dev, by='movieId') %>%
  left_join(user_dev, by='userId') %>%
  mutate(pred = avg + user_bias+movie_bias) %>%
  pull(pred)

results4<- Accuracy(predicted_ratings, validation$rating)
results4
```

```
## [1] 0.8653488
```

Regularization

Now we considered the user and movie bias and are very close to achieve the goal of a RMSE<0.86490, for that purpose now Regularization will be useful as it will put some penalty on the user bias as well as the movie bias resulting due to a movie which has very less number of ratings and also users who didn't give a lot of ratings. We need to find out a tuning parameter which will lower the RMSE in such cases. Let's find out and get to the fine tuning of our approach:

Approach-5

```
alpha <- seq(1, 10, 0.25)


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

  avg <- mean(edx$rating)
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - avg)/(n()+l))
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - avg)/(n()+l))
  predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = avg + b_i + b_u) %>%
    pull(pred)

  return(Accuracy(predicted_ratings, validation$rating))
})
```

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

Now let's choose the best possible alpha

```
alpha <- alpha[which.min(rmses)]
alpha
```

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

Accuracy of this model is:

```
min(rmses)
```

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

```
result5=min(rmses)
```

Conclusion:

```
Models<-c("Naive","Movie Bias","User Bias","Movie and user Bias"," Regulaization and User+Movie bias")
Results<-c(results1,results2,results3,results4,result5)
Prediction_Table= data.frame(Models,Results)
Prediction_Table
```

```
##                              Models   Results
## 1                             Naive 1.0612018
## 2                        Movie Bias 0.9439087
## 3                         User Bias 0.9947953
## 4               Movie and user Bias 0.8653488
## 5  Regulaization and User+Movie bias 0.8648170
```

We do notice that the 5th mode has a Rmse < 0.86490 and thus we achieve the goal.

Hence we got the desired results with the help of a few techniques. Moreover we can go for feature engineering to get even better results and we can also try: 1. Matrix Factorization 2. COnsidering Genres

References

1. https://en.wikipedia.org/wiki/MovieLens
2. https://stackoverflow.com/questions/62140483/how-to-interpret-dplyr-message-summarise-regrouping-output-by-x-override