Recommendation System

Haya Abdeh

29 June 2021

Abstract

This is the first project for the Harvard Data Science Professional Program by Prof. of Biostatistics Rafael Irizarry from Harvard University. In this first capstone project, we have to build a recommendation system for netflix company based on a 10M (millions) rows rating dataset named MovieLens created by the University of Minnesota, and going to analyze it and perform machine learning tasks in complete autonomy .

Summary

Recommender systems is a filtering tools for information that aims to predict ratings for users and items, mostly from big data to recommend their likes. Movie recommendation systems provide a mechanism to help users in classifying users with similar interests. This makes recommender systems basically a focal part of websites and e-commerce applications. we are going to apply some of machine learning principles to build a recommender system that would introduced in the next pages.

Introduction

Data science is the gate to introduce machine learning as a technique to describe big data and extract knowledge by applying algorithms that analyze and process data into helpful information and naturally intuitive solutions. and as machine learning improves the modern business making in recent years, we can see Netflix one of the biggest companies that took an active role as producer and distributor for both film and television series, worked on improving a recommendation system to attract more subscription users, and we all heard about the famous success story of the recommendation system competition at October 2006 by Netflix company. In which Netflix awarded a one-million-dollar prize to the team whom developed an algorithm that increased the accuracy of the company's recommendation engine by 10%. We are going to mimic a predict movie rating system for users in a large movie dataset, which would be a challenge to find out the movies with a high rate since users prefer movies over others.

The working mechanism is to apply the principles of machine learning to reduce the root mean squared error RMSE of the predicted ratings versus the actual ratings. This approach called linear regression for prediction as a baseline approach We are going to partition the data into train data and test data as a validation set, the first model that we will build assumes the same rating for all movies and all users, the Naive Baseline Model then we are going to start with calculating residual mean squared error RMSE with user bias since users prefer movies over others, then we are going to improve the RMSE within the next steps with Regularization which would permits us to penalize large estimates that come from small sample sizes, so we will be able to reduce the value of RMSE.and another approach called the matrix factorization will be build to calculate the RMSE value, then after that We will compare the linear regression approach with matrix factorization approach, to find out what approach gives us a better prediction with a lower value of RMSE. The dataset used is [MovieLens 10M dataset] (https://grouplens.org/datasets/movielens/10m/) (http://files.grouplens.org/datasets/movielens/ml-10m.zip)

```r
# Loading all needed libraries

library(dplyr)
library(tidyverse)
library(kableExtra)
library(tidyr)
library(ggplot2)
library(plotly)
library(gbm)
library(caret)
library(xgboost)
library(e1071)
library(class)
library(lightgbm)
library(ROCR)
library(randomForest)
library(PRROC)
library(reshape2)
library(data.table)
library(lubridate)
library(knitr)
library(recosystem)
library(tinytex)
webshot::install_phantomjs()
```

Executive Summary

We start with creating a recommendation system using the "prediction". We are going to train our algorithms using a (edx) set to predict movie ratings in the validation set, and we will evaluate how close our predictions are to the real true values by Measuring RMSE. we are going to use different Machine Learning techniques, like regression models, ensemble methods (random forest).

Exploratory Analysis for Data Set

Introduce The Dataset

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

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

after loading the data set edx, are going to do a quick check on the train set and the test set

```
#The Trainig Set
dim(edx)
```

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

Let us start looking at the data structure and type of the data set which we are going to work with

```
#class of the data set edx
class(edx)
```

```
## [1] "data.table" "data.frame"
```

```
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
# Find out Class of each Feature, along with internal structure
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
# Find out Names of the Columns (Features)
names(edx)
```

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

We can see the first six rows of the edx data set

```
#head of the data set edx
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
```

```
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                          genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

```
#The Test Set
dim(validation)
```

```
## [1] 999999      6
```

```
#The Test Set
class(validation)
```

```
## [1] "data.table" "data.frame"
```

```
# Find out Class of each Feature, along with internal structure
str(validation)
```

```
## Classes 'data.table' and 'data.frame':   999999 obs. of  6 variables:
##  $ userId   : int  1 1 1 2 2 2 3 3 4 4 ...
##  $ movieId  : num  231 480 586 151 858 ...
##  $ rating   : num  5 5 5 3 2 3 3.5 4.5 5 3 ...
##  $ timestamp: int  838983392 838983653 838984068 868246450 868245645 868245920 1136075494 1133571200
##  $ title    : chr  "Dumb & Dumber (1994)" "Jurassic Park (1993)" "Home Alone (1990)" "Rob Roy (1995)"
##  $ genres   : chr  "Comedy" "Action|Adventure|Sci-Fi|Thriller" "Children|Comedy" "Action|Drama|Roman
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
#The Test Set
names(validation)
```

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

SO far we find out that the edX dataset is made of 6 features and 9,000,055 observations.The validation set which represents 10% of the 10M Movielens dataset contains the same features , but with a total of 999,999 occurrences. we assured that the movieId and the userId in edx set are also in the validation set.

Data Type Analysis ##quantitative data

#userId : discrete, Unique user ID. #movieId: discrete, Unique movie ID. #timestamp : discrete , Date and time.

##qualitative features

#title: nominal , not unique - movie title #genres: nominal. #rating : continuous, a rating between 0 and 5 for the movie.

We can see the most rated Films arranged by descending order with the highest rate at the first.

```
# Most rated films
edx %>% group_by(title) %>%
  summarize(n_ratings = n()) %>%
  arrange(desc(n_ratings))
```

```
## # A tibble: 10,676 x 2
##    title                                                     n_ratings
##    <chr>                                                         <int>
##  1 Pulp Fiction (1994)                                           31362
##  2 Forrest Gump (1994)                                           31079
##  3 Silence of the Lambs, The (1991)                              30382
##  4 Jurassic Park (1993)                                          29360
##  5 Shawshank Redemption, The (1994)                              28015
##  6 Braveheart (1995)                                             26212
##  7 Fugitive, The (1993)                                          25998
##  8 Terminator 2: Judgment Day (1991)                             25984
##  9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)  25672
## 10 Apollo 13 (1995)                                              24284
## # ... with 10,666 more rows
```

#And The number of movies rated just once

```
# Number of movies rated once
edx %>% group_by(title) %>%
  summarize(n_ratings = n()) %>%
  filter(n_ratings==1) %>%
  count() %>% pull()
```

```
## [1] 126
```

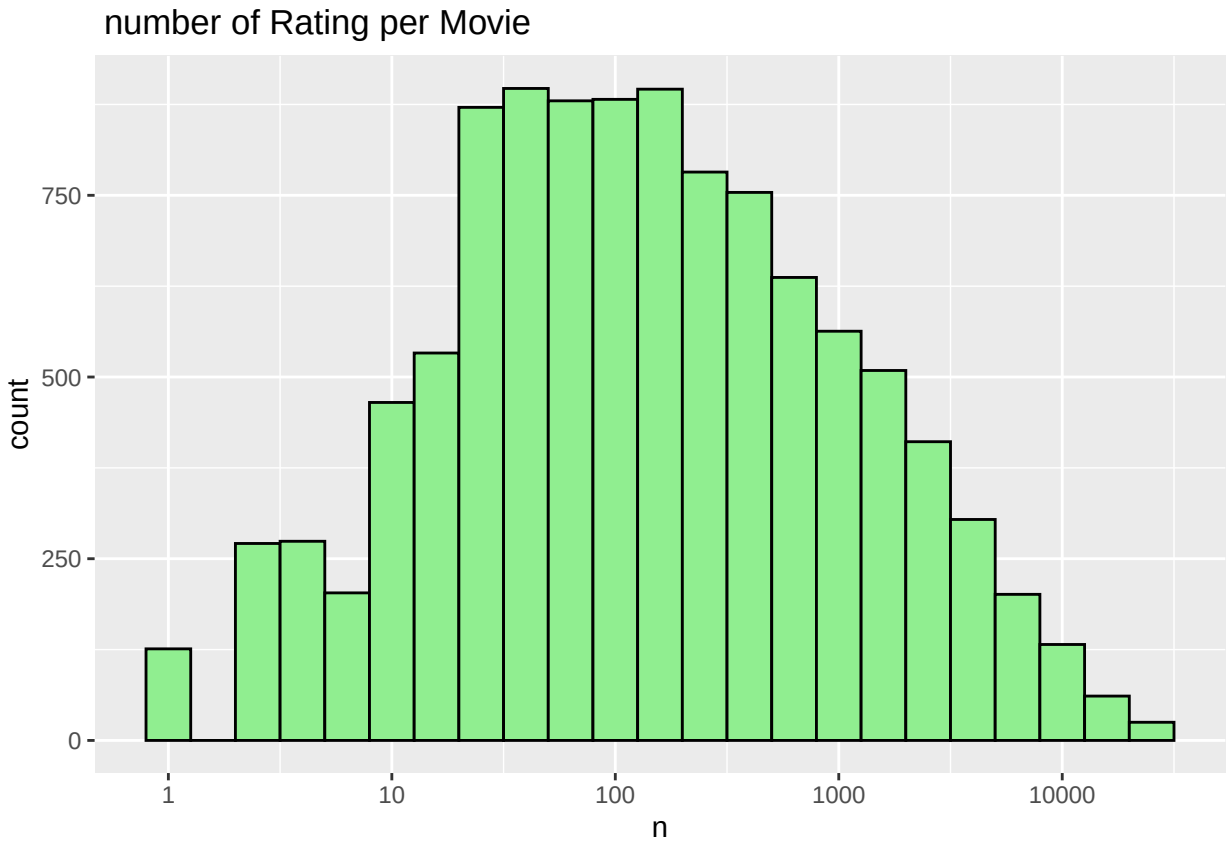
#The number of users on edx dataset is 69,878

```
#number of users in edx dataset
edx %>% group_by(userId) %>% summarize(count = n())
```

```
## # A tibble: 69,878 x 2
##    userId count
##     <int> <int>
##  1      1    19
##  2      2    17
##  3      3    31
##  4      4    35
##  5      5    74
##  6      6    39
##  7      7    96
##  8      8   727
##  9      9    21
## 10     10   112
## # ... with 69,868 more rows
```

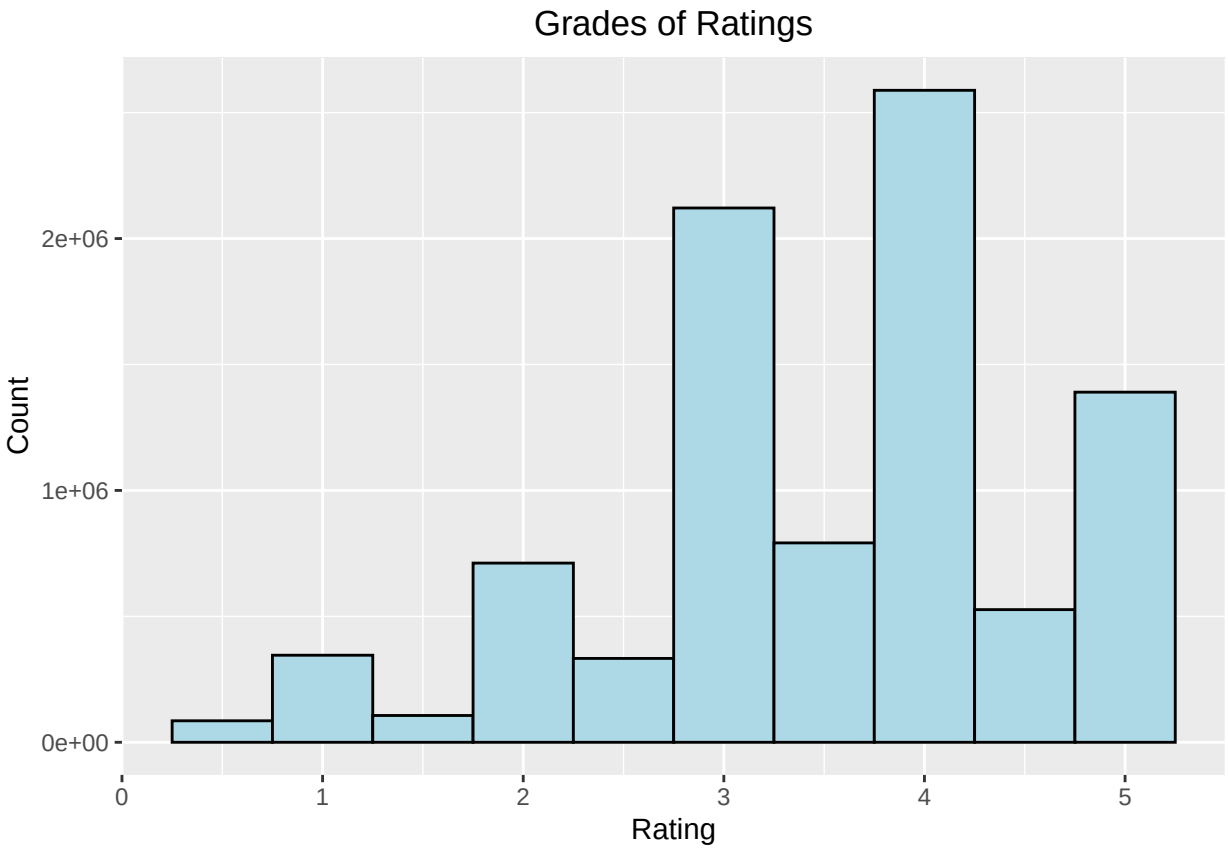
#The plot below shows the number of ratings for every movie

```
edx %>% count(movieId) %>% ggplot(aes(n))+
  geom_histogram(color = "black" , fill= "light green",bins = 10 , binwidth = 0.2)+
  scale_x_log10()+
  ggtitle(" number of Rating per Movie")+
  theme_gray()
```



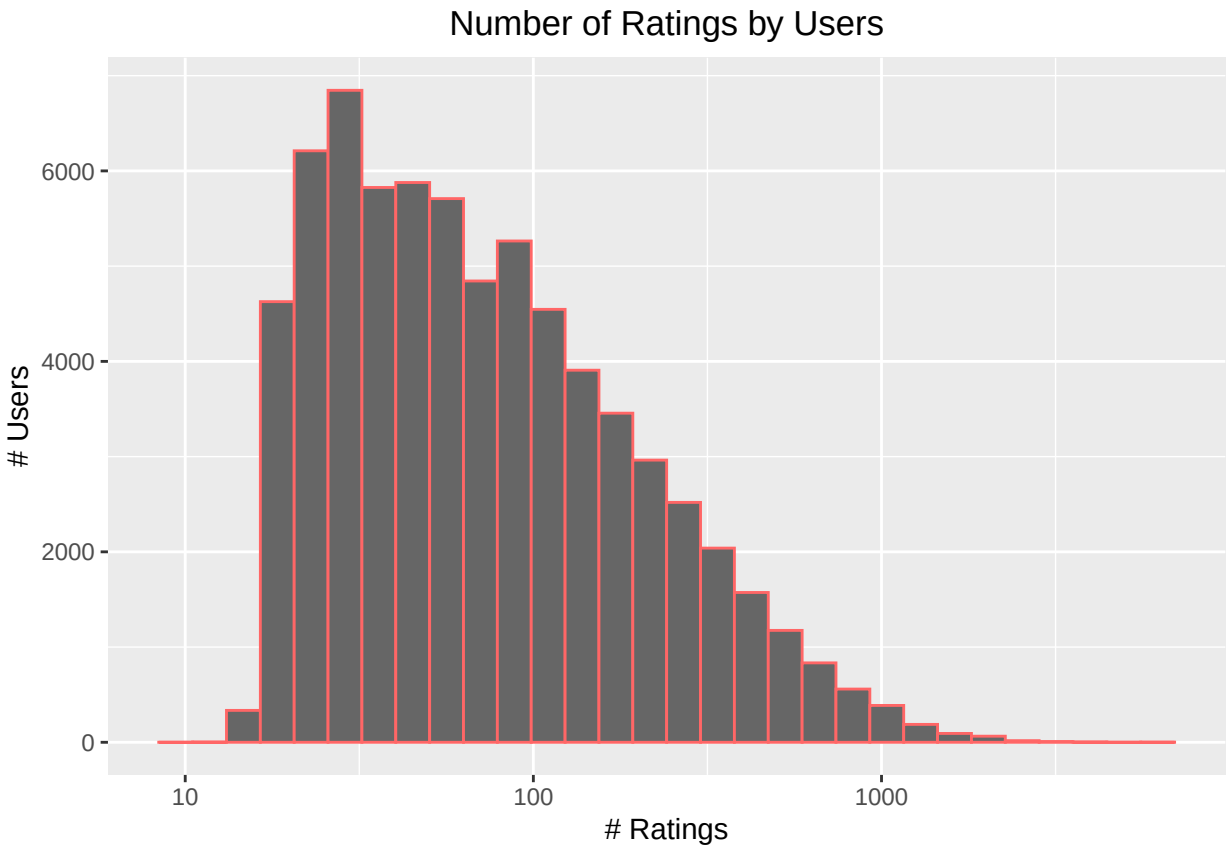
As there are users gives a full rate as (5,4,3,2,1) for the movies , there are other users gives a half rating (4.5 , 3.5 , 2.5 , 1.5 , 0.5) for movies , here are the most users gives ratings for movies

```
# Ratings Histogram
edx %>%
  ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.5, color = "black" , fill= "light blue") +
  xlab("Rating") +
  ylab("Count") +
  ggtitle("Grades of Ratings") +
  theme(plot.title = element_text(hjust = 0.5))
```



The majority of users rate between 10 and 100 movies, whilst some may rate over 1,000. Including a variable in the model to account for number of ratings should be discussed. Also we can see the number of users gives ratings for movies „ some users rate between 10 and 100 movies, whilst some may rate over 1,000 or even did not give any ratings

```
# Ratings Users - Number of Ratings
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(color = "#FF6666",fill= "#666666", bins=30) +
  scale_x_log10() +
  xlab("# Ratings") +
  ylab("# Users") +
  ggtitle("Number of Ratings by Users") +
  theme(plot.title = element_text(hjust = 0.5))
```


Number of Ratings by Users
Users
6000
4000
2000
0
10
100
1000
Ratings

Methods of Machine Learning

And as we do some data analysis on the edx DataSet we will run machine learning algorithms to make a prediction model that would predict the highest ratings on movies with reducing the value of residual mean squared error RMSE.

RMSE defined as the standard deviation of the residuals (prediction errors) where residuals are a measure of spread of data points from the regression line . and we calculate the RMSE to represent the error loss between the predicted ratings derived from applying the algorithm and actual ratings in the test set.

we are going to use the linear regression approach, the Naive Baseline Model then we are going to start with calculating residual mean squared error RMSE with user bias since users prefer movies over others, then we are going to build another model based on Regularization approach, and we are going to introduce the matrix factorization model,and after running these models we will compute the RMSE value and compare it with each model to find the optimal value of it

Let us start ...

#Naive Baseline Model

First, build a Naive model and calculate the average of the edx data set, assuming that all users give ratings for all movies

```
mu <- mean(edx$rating)
rmse <- RMSE(validation$rating, mu)

# We make a data frame to save the results of RMSE
results <- data.frame(model="Mean Of edx Model", RMSE=rmse)

#We can see that RMSE on the validation dataset is 1.06. It is very high and we need to get RMSE (below
```

#Add the movie bias term which calculate the average of the rankings for movies, we can use this formula:

$$Y_{u,i} = \hat{\mu} + b_i + \epsilon_{u,i}$$

we calculated mu_hat as mu at the previous step

```
mu <- mean(edx$rating)
movie_bias <- edx %>%
group_by(movieId) %>%
summarize(movie_bias = mean(rating - mu))
```

predict ratings with mu

```
  predict_rat <- validation %>%
  left_join(movie_bias, by='movieId') %>%
  mutate(pred = mu + movie_bias) %>%
  pull(pred)

RMSE(validation$rating, predict_rat)
```

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

```
rmse_result <- RMSE(validation$rating, predict_rat)

#Adding the result we had to the results dataset that we build before
results <- results %>% add_row(model="Movie Bias Model", RMSE=rmse_result)

# we notice that RMSE is 0.94 ,, still not the result that we need
```

#Now calculate user bias term. to minimizes the effect of extreme ratings made by users and we will use the formula

$$Y_{u,i} = \hat{\mu} + b_i + b_u + \epsilon_{u,i}$$

```
# user bias
# again the average of all movies
# and use the movie_bias from the previous step
mu <- mean(edx$rating)
user_bias <- edx %>%
left_join(movie_bias, by='movieId') %>%
group_by(userId) %>%
summarize(user_bias = mean(rating - mu - movie_bias))
```

predict ratings with movie and user bias

```
mu <- mean(edx$rating)
predict_rat_user <- validation %>%
left_join(movie_bias, by='movieId') %>%
left_join(user_bias, by='userId') %>%
mutate(pred = mu + movie_bias + user_bias) %>%
pull(pred)

RMSE(predict_rat_user, validation$rating)
```

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

```
# Let us Add the result to the results dataset
result_user <- RMSE(predict_rat_user, validation$rating)
results <- results %>% add_row(model="bias Movie and User  Model", RMSE= result_user)

# We got a RMSE = 0.8653488 this is a good result so far, letus continue to see if we can get better re
```

Calculating the rating as the Genres of the movies „, we use this formula

$$Y_{u,i} = \hat{\mu} + b_i + b_u + b_{u,g} + \epsilon_{u,i}$$

```
mu <- mean(edx$rating)
# Calculate average by The movie
movie_avg <- edx %>%
   group_by(movieId) %>%
   summarize(movie_bias = mean(rating - mu))

# Calculate average by The user
user_avg <- edx %>%
```

```
   left_join(movie_avg, by='movieId') %>%
   group_by(userId) %>%
   summarize(user_bias = mean(rating - mu - movie_bias))

#Calculate the genres as the user prefer more
  prefered_genre <- edx %>%
  left_join(movie_avg, by='movieId') %>%
  left_join(user_avg, by='userId') %>%
  group_by(genres) %>%
  summarize(movie_user_genre = mean(rating - mu - movie_bias - user_bias))

# Compute the predicted ratings on validation dataset

rmse_m_u_g <- validation %>%
   left_join(movie_avg, by='movieId') %>%
   left_join(user_avg, by='userId') %>%
   left_join(prefered_genre, by='genres') %>%
   mutate(pred = mu + movie_bias + user_bias + movie_user_genre) %>%
   pull(pred)

RMSE(validation$rating, rmse_m_u_g)
```

```
## [1] 0.8649469
```

```
result_m_u_g <- RMSE(validation$rating, rmse_m_u_g)

#  Add the result to the results dataset

results <- results %>% add_row(model="Movie and User and genres Model", RMSE= result_m_u_g)

# We got a RMSE = 0.8649469 this is a very satisfying result but it did not make a very significant imp
```

Since every predictors will reduce the RMSE value, lets see if we can improve the value of RMSE with another approach.

Regularization approach

Regularization technique permits us to penalize large estimates that come from small sample sizes. We use regularization to reduce the effect of bias because of extreme rating by the users on movies as users prefer movies over others.

we use lambda to denote the prediction obtained when we use a parameter, we use this formula

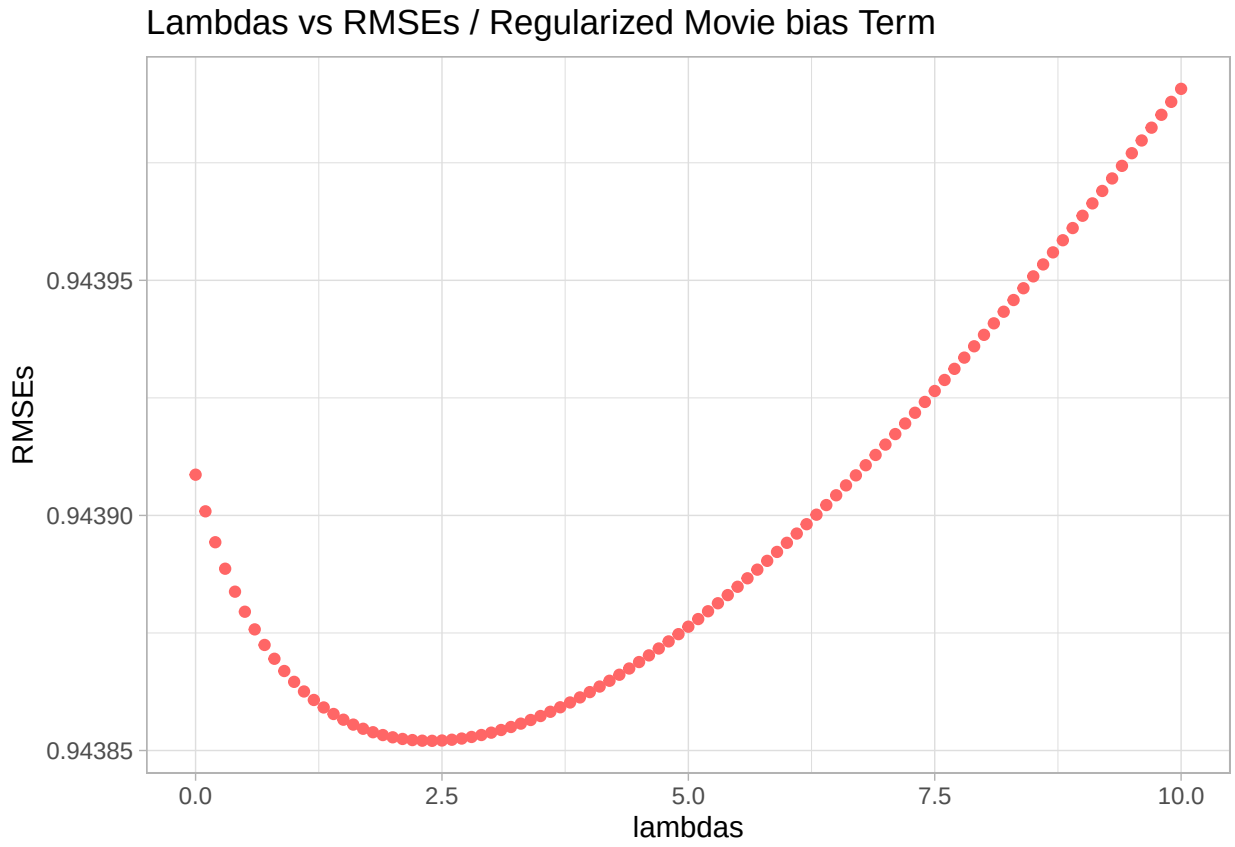
$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu})$$

###Introducing the next working steps

#compute regularized movie bias term #compute regularize movie and user bias term #compute regularize movie and user and genres bias term #compute predictions on validation set #return RMSE of the predictions

```
# calculate average
mu <- mean(edx$rating)

# determine a sequence (lambda)
lambda <- seq(0, 10, 0.1)

# output RMSE of each lambda
rmses <- sapply(lambda, function(l){


# compute regularized movie bias part
  movie_bias <- edx %>%
  group_by(movieId) %>%
  summarize(movie_bias = sum(rating - mu)/(n()+l))

# Compute the predicted ratings on validation dataset
  predict_rate <- validation %>%
  left_join(movie_bias, by='movieId') %>%
  mutate(pred = mu + movie_bias) %>%
  pull(pred)

  return(RMSE(predict_rate, validation$rating))
})
```

Including Plots of lambda's results



Find the minimum value of lambda that would minimize the value of RMSE

```
# find lambda value that minimize the RMSE
min_lambda <- lambda[which.min(rmses)]

min_lambda
```

```
## [1] 2.4
```

```
# Predict the RMSE on the validation set

rmse_regularizedmovieterm <- min(rmses)
rmse_regularizedmovieterm
```

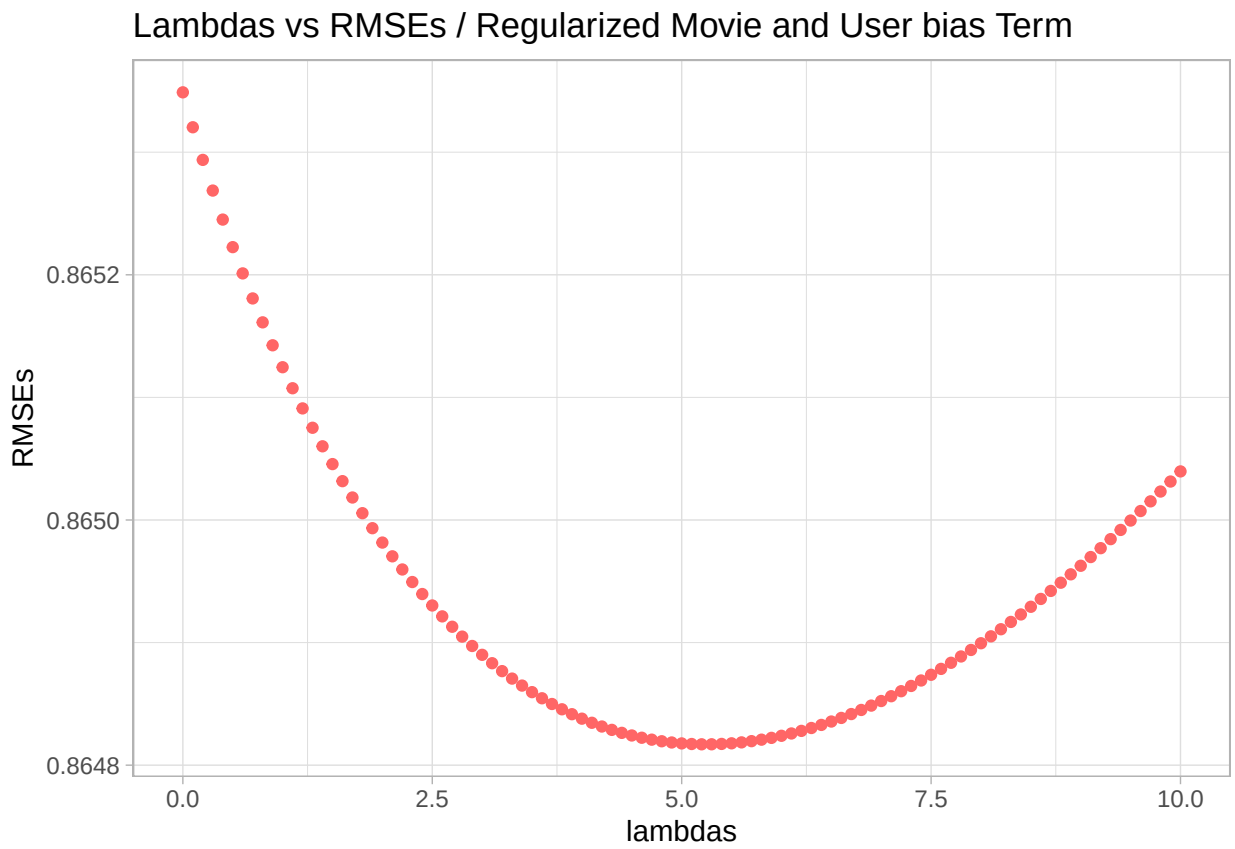
```
## [1] 0.9438521
```

```
# Adding the results to the results dataset
results <- results %>% add_row(model="Regularized Movie Term", RMSE=rmse_regularizedmovieterm)
```

#Regularization on movie and user term

```r
# Since we see before that the movie and user bias model had the acceptable RMSE ,, we will ses if the

# Determine a sequence (lambda)
lambda <- seq(from=0, to=10, by=0.1)


# output RMSE of each lambda
rmses <- sapply(lambda, function(l){

# calculate average
mu <- mean(edx$rating)

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

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

predict_rat <- validation %>%
  left_join(movie_bias, by = "movieId") %>%
  left_join(user_bias, by = "userId") %>%
  mutate(pred = mu + movie_bias + user_bias) %>%
  pull(pred)

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

Now we plot the Lambdas vs RMSEs for the Regularized Movie and user bias Term



find lambda value that minimize the RMSE in Regularized Movie and User bias Term

```
# find lambda value that minimize the RMSE in Regularized Movie and User bias Term
min_lambda <- lambda[which.min(rmses)]

min_lambda
```

```
## [1] 5.2
```

```
# Predict the RMSE on the validation set

rmse_regularizedmovieuserterm <- min(rmses)
rmse_regularizedmovieuserterm
```

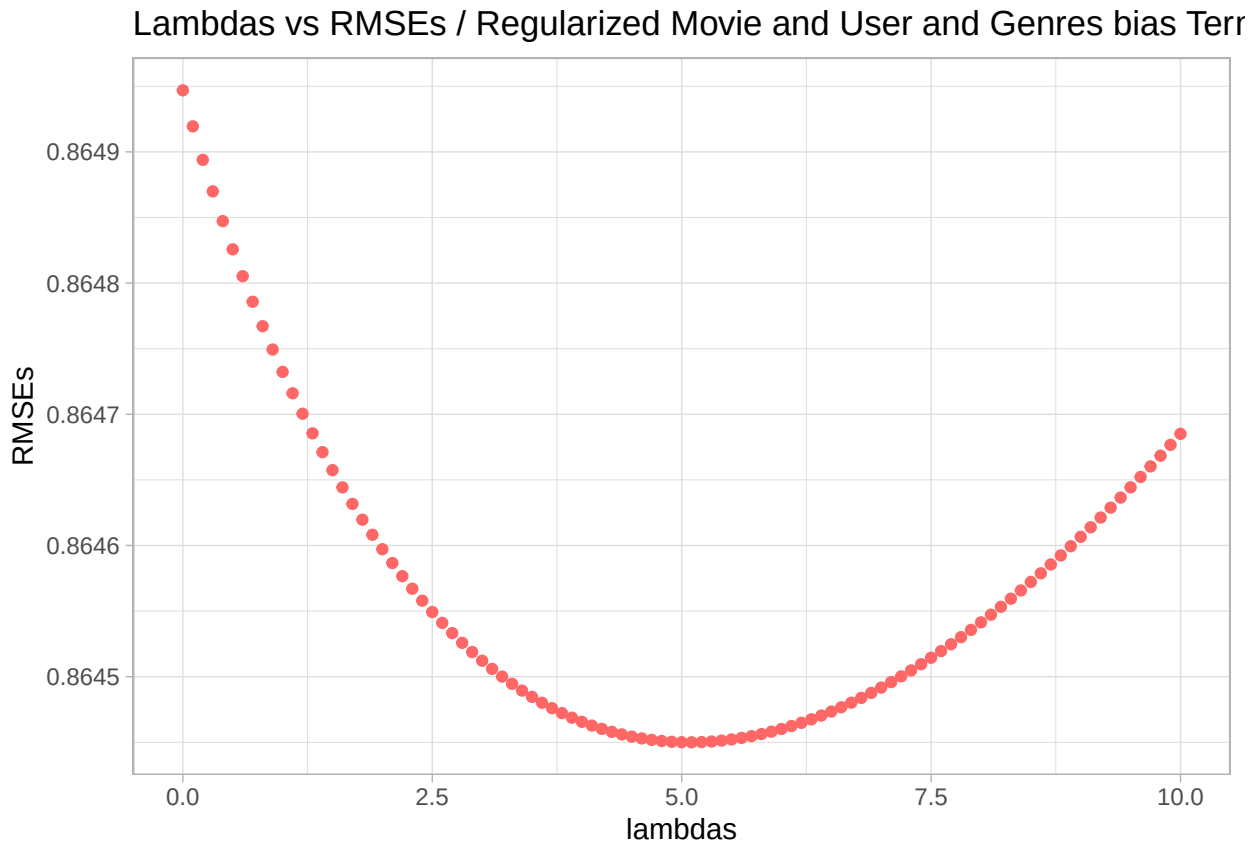
```
## [1] 0.864817
```

```
# Adding the results to the results dataset
results <- results %>% add_row(model="Regularized Movie and User Term", RMSE=rmse_regularizedmovieuserte
```

#We can see that the RMSE from regularized movies and user term is Slightly lower than the non regularized movies and user bias term „ let us see if can had a better performance
#Let us compute the Regularized movie and user and genres Term and see if it would make RMSE is less than before

```
# Determine a sequence (lambda)
lambda <- seq(from=0, to=10, by=0.1)


# output RMSE of each lambda
rmses <- sapply(lambda, function(l){
# calculate average
mu <- mean(edx$rating)

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

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

prefered_genre <- edx %>%
      left_join(movie_bias, by='movieId') %>%
      left_join(user_bias, by='userId') %>%
      group_by(genres) %>%
      summarize(mov_user_gen= sum(rating - mu - movie_bias - user_bias) / (n() + l))


predict_rat <- validation %>%
  left_join(movie_bias, by = "movieId") %>%
  left_join(user_bias, by = "userId") %>%
  left_join( prefered_genre, by="genres") %>%
  mutate(pred = mu + movie_bias + user_bias + mov_user_gen) %>%
  pull(pred)

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

We plot the Lambdas vs RMSEs for the Regularized Movie and user and genres bias Term



Find lambda value that minimize the RMSE in Regularized Movie and User and Genres bias Term, and record the value of RMSE notice the value of RMSE

```
# find lambda value that minimize the RMSE in Regularized Movie and User and Genres bias Term
min_lambda <- lambda[which.min(rmses)]

min_lambda
```

```
## [1] 5.1
```

```
# Predict the RMSE on the validation set

rmse_regularizedmovieuserterm <- min(rmses)
rmse_regularizedmovieuserterm
```

```
## [1] 0.86445
```

```
# Adding the results to the results dataset
results <- results %>% add_row(model="Regularized Movie and User and Genres Term", RMSE=rmse_regularize
```

#Matrix Factorization Approach

There are 2 types of recommender systems: Content filtering (meta data or side information – which is based on the description of the item) And collaborative Filtering: which calculate the similarity measures of the target ITEMS then find the minimum (Euclidean distance, or Cosine distance, or other metric, depending on the algorithm used).

we will work with the the collaborative filtering type of recommender systems using Matrix factorization approach.

Matrix factorization algorithms work by splitting the user-item interaction matrix into the product of two lower dimensional rectangular matrices.and the recommender system will predict unknown entries in the rating matrix based on observed values.

Let us start building the model

```
#define RMSE function

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

We create Two new matrices (train and validation set) with three features (movieId, userId, rating)

```
#create Two new matrices
edx_m_f <- edx %>% select(movieId, userId, rating)
validation_m_f <- validation %>% select(movieId, userId, rating)

edx_m_f <- as.matrix(edx_m_f)
validation_m_f<- as.matrix(validation_m_f)



# save the files as tables
write.table(edx_m_f, file = "training_set.txt", sep = " ", row.names = FALSE,
    col.names = FALSE)

write.table(validation_m_f, file = "validation_set.txt", sep = " ",
    row.names = FALSE, col.names = FALSE)



set.seed(1)

training_ds <- data_file("training_set.txt")

validation_ds <- data_file("validation_set.txt")


#calling the function Reco()
r = Reco()

#perform many different settings until you will reach the optimal

opts = r$tune(training_ds, opts = list(dim = c(10, 20, 30), lrate = c(0.1,
    0.2), costp_l1 = 0, costq_l1 = 0, nthread = 1, niter = 10))
```

Let us train the model with calling (train) function as the result of the previous step (tune)

```
r$train(training_ds, opts = c(opts$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9725   1.2028e+07
##    1       0.8718   9.8796e+06
##    2       0.8390   9.1796e+06
##    3       0.8174   8.7601e+06
##    4       0.8016   8.4792e+06
##    5       0.7898   8.2775e+06
##    6       0.7803   8.1272e+06
##    7       0.7724   8.0078e+06
##    8       0.7657   7.9131e+06
##    9       0.7598   7.8339e+06
##   10       0.7546   7.7669e+06
##   11       0.7500   7.7090e+06
##   12       0.7459   7.6589e+06
##   13       0.7422   7.6139e+06
##   14       0.7388   7.5766e+06
##   15       0.7357   7.5416e+06
##   16       0.7328   7.5112e+06
##   17       0.7302   7.4824e+06
##   18       0.7277   7.4575e+06
##   19       0.7254   7.4354e+06
```

```
#We write predictions to a tempfile

save_prediction = tempfile()
```

With the predict function let us make predictions on validation set and compute RMSE:

```
r$predict(validation_ds, out_file(save_prediction))


real_rates <- read.table("validation_set.txt", header = FALSE, sep = " ")$V3


pred_rates <- scan(save_prediction)
```

RMSE as we mentioned earlier is the amount by which the values predicted by an estimator differ from the quantities being estimated.

```
RMSE_mf <- RMSE(real_rates, pred_rates)
```

Root Mean Squared Error RMSE of Matrix Factorization model

```
# print out the Root Mean Squared Error
RMSE_mf
```

```
## [1] 0.7830654
```

```
#add the RMSE result to the table of results
results <- results %>% add_row(model="Matrix Factorization Model", RMSE=RMSE_mf)
```

So we can see how lower is the value of RMSE that generated from matrix factorization method.

Let us compare the first 100 predictions of the matrix factorization model with the real ratings. we will make rounding on predictions to be convenient

```
p_r_rounded <- pred_rates

p_r_rounded <- round(p_r_rounded/0.5) * 0.5

fst100_pred <- data.frame(real_rates[1:100], p_r_rounded[1:100])

names(fst100_pred) <- c("real_rates", "predicted_rates")

correct_predictions <- 1:100

 x <- ifelse(fst100_pred$real_rates==fst100_pred$predicted_rates,1,0)
```

```
fst100_pred %>% mutate(correct_predictions)
```

```
##     real_rates predicted_rates correct_predictions
## 1          5.0             4.0                   1
## 2          5.0             5.0                   2
## 3          5.0             4.5                   3
## 4          3.0             3.5                   4
## 5          2.0             4.5                   5
## 6          3.0             3.0                   6
## 7          3.5             4.0                   7
## 8          4.5             4.5                   8
## 9          5.0             4.5                   9
## 10         3.0             3.5                  10
## 11         3.0             3.5                  11
## 12         3.0             3.5                  12
## 13         3.0             3.5                  13
## 14         3.0             4.0                  14
## 15         3.0             3.5                  15
## 16         3.0             4.5                  16
## 17         3.0             4.0                  17
## 18         3.0             3.0                  18
## 19         4.0             4.0                  19
## 20         5.0             3.5                  20
## 21         3.0             3.5                  21
## 22         4.0             4.5                  22
## 23         4.0             4.5                  23
## 24         4.0             4.5                  24
## 25         5.0             5.0                  25
## 26         4.0             3.0                  26
## 27         2.0             2.5                  27
## 28         5.0             4.5                  28
## 29         5.0             4.5                  29
## 30         5.0             4.5                  30
```

```
## 31        4.0             4.5             31
## 32        3.0             3.0             32
## 33        4.0             4.5             33
## 34        4.0             3.5             34
## 35        5.0             4.5             35
## 36        5.0             4.0             36
## 37        5.0             4.5             37
## 38        4.0             3.0             38
## 39        4.0             4.0             39
## 40        4.0             4.5             40
## 41        4.0             4.0             41
## 42        5.0             4.0             42
## 43        3.5             3.5             43
## 44        5.0             3.5             44
## 45        4.0             3.5             45
## 46        4.5             4.0             46
## 47        2.5             3.0             47
## 48        4.5             3.5             48
## 49        3.5             4.0             49
## 50        4.0             3.5             50
## 51        2.5             3.0             51
## 52        3.5             4.0             52
## 53        3.5             4.0             53
## 54        3.5             3.5             54
## 55        3.5             4.0             55
## 56        2.5             3.0             56
## 57        3.0             3.5             57
## 58        3.5             3.0             58
## 59        3.0             3.5             59
## 60        3.5             3.0             60
## 61        2.5             3.5             61
## 62        4.0             3.5             62
## 63        3.5             3.5             63
## 64        4.0             3.5             64
## 65        3.5             3.5             65
## 66        4.0             3.5             66
## 67        3.5             4.0             67
## 68        3.0             3.5             68
## 69        4.5             3.0             69
## 70        2.0             2.5             70
## 71        3.0             2.5             71
## 72        3.0             3.5             72
## 73        4.0             4.0             73
## 74        3.5             3.0             74
## 75        3.5             3.5             75
## 76        3.5             3.0             76
## 77        4.0             4.0             77
## 78        3.0             3.5             78
## 79        3.5             3.5             79
## 80        3.5             4.0             80
## 81        4.0             3.5             81
## 82        3.5             3.0             82
## 83        3.0             3.5             83
## 84        3.5             3.5             84
```

```
## 85         4.0             3.5                  85
## 86         3.5             4.0                  86
## 87         4.0             4.0                  87
## 88         3.5             3.5                  88
## 89         3.0             3.0                  89
## 90         3.0             3.0                  90
## 91         3.0             2.5                  91
## 92         1.0             2.5                  92
## 93         4.0             3.5                  93
## 94         4.0             3.5                  94
## 95         3.0             3.5                  95
## 96         3.0             2.0                  96
## 97         3.0             2.5                  97
## 98         4.5             3.5                  98
## 99         3.5             3.5                  99
## 100        3.5             3.5                 100
```

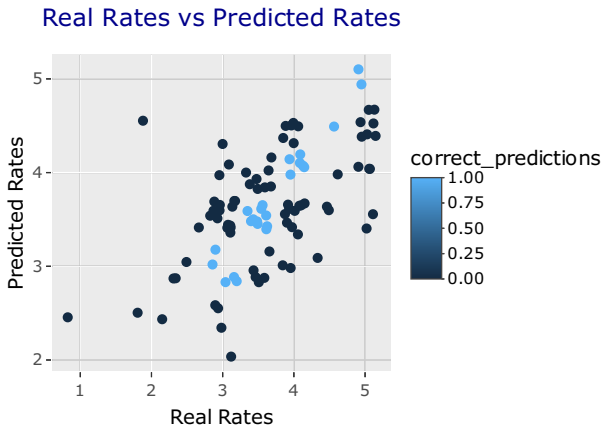
```
fst100_pred$correct_predictions <- x
names(fst100_pred) <- c("real_rates", "predicted_rates" , "correct_predictions")
fst100_pred
```

```
##     real_rates predicted_rates correct_predictions
## 1          5.0             4.0                   0
## 2          5.0             5.0                   1
## 3          5.0             4.5                   0
## 4          3.0             3.5                   0
## 5          2.0             4.5                   0
## 6          3.0             3.0                   1
## 7          3.5             4.0                   0
## 8          4.5             4.5                   1
## 9          5.0             4.5                   0
## 10         3.0             3.5                   0
## 11         3.0             3.5                   0
## 12         3.0             3.5                   0
## 13         3.0             3.5                   0
## 14         3.0             4.0                   0
## 15         3.0             3.5                   0
## 16         3.0             4.5                   0
## 17         3.0             4.0                   0
## 18         3.0             3.0                   1
## 19         4.0             4.0                   1
## 20         5.0             3.5                   0
## 21         3.0             3.5                   0
## 22         4.0             4.5                   0
## 23         4.0             4.5                   0
## 24         4.0             4.5                   0
## 25         5.0             5.0                   1
## 26         4.0             3.0                   0
## 27         2.0             2.5                   0
## 28         5.0             4.5                   0
## 29         5.0             4.5                   0
## 30         5.0             4.5                   0
## 31         4.0             4.5                   0
## 32         3.0             3.0                   1
```

```
## 33        4.0             4.5             0
## 34        4.0             3.5             0
## 35        5.0             4.5             0
## 36        5.0             4.0             0
## 37        5.0             4.5             0
## 38        4.0             3.0             0
## 39        4.0             4.0             1
## 40        4.0             4.5             0
## 41        4.0             4.0             1
## 42        5.0             4.0             0
## 43        3.5             3.5             1
## 44        5.0             3.5             0
## 45        4.0             3.5             0
## 46        4.5             4.0             0
## 47        2.5             3.0             0
## 48        4.5             3.5             0
## 49        3.5             4.0             0
## 50        4.0             3.5             0
## 51        2.5             3.0             0
## 52        3.5             4.0             0
## 53        3.5             4.0             0
## 54        3.5             3.5             1
## 55        3.5             4.0             0
## 56        2.5             3.0             0
## 57        3.0             3.5             0
## 58        3.5             3.0             0
## 59        3.0             3.5             0
## 60        3.5             3.0             0
## 61        2.5             3.5             0
## 62        4.0             3.5             0
## 63        3.5             3.5             1
## 64        4.0             3.5             0
## 65        3.5             3.5             1
## 66        4.0             3.5             0
## 67        3.5             4.0             0
## 68        3.0             3.5             0
## 69        4.5             3.0             0
## 70        2.0             2.5             0
## 71        3.0             2.5             0
## 72        3.0             3.5             0
## 73        4.0             4.0             1
## 74        3.5             3.0             0
## 75        3.5             3.5             1
## 76        3.5             3.0             0
## 77        4.0             4.0             1
## 78        3.0             3.5             0
## 79        3.5             3.5             1
## 80        3.5             4.0             0
## 81        4.0             3.5             0
## 82        3.5             3.0             0
## 83        3.0             3.5             0
## 84        3.5             3.5             1
## 85        4.0             3.5             0
## 86        3.5             4.0             0
```

```
## 87          4.0             4.0                 1
## 88          3.5             3.5                 1
## 89          3.0             3.0                 1
## 90          3.0             3.0                 1
## 91          3.0             2.5                 0
## 92          1.0             2.5                 0
## 93          4.0             3.5                 0
## 94          4.0             3.5                 0
## 95          3.0             3.5                 0
## 96          3.0             2.0                 0
## 97          3.0             2.5                 0
## 98          4.5             3.5                 0
## 99          3.5             3.5                 1
## 100         3.5             3.5                 1
```

```
pr_mf <- ggplot(data = fst100_pred, aes(x = real_rates, y = predicted_rates ,col = correct_predictions )

    ggtitle("Real Rates vs Predicted Rates ") + theme(plot.title = element_text(color = "dark blue", hj
ggplotly(pr_mf)
```



Results of RMSE:

As a result we find that the minimum value of RMSE obtained from Matrix Factorization model which is (0.7830654).

Let us print the results table to show and compare the RMSE values for each approach , so we can evaluate our models performances.

```
# printing the results table to show the RMSE values
print(results)
```

```
##                                       model      RMSE
## 1                          Mean Of edx Model 1.0612018
## 2                           Movie Bias Model 0.9439087
## 3                bias Movie and User  Model 0.8653488
## 4            Movie and User and genres Model 0.8649469
## 5                     Regularized Movie Term 0.9438521
## 6            Regularized Movie and User Term 0.8648170
## 7 Regularized Movie and User and Genres Term 0.8644500
## 8                 Matrix Factorization Model 0.7830654
```

Conclusion

we find that Matrix Factorization approach is effective way to make prediction on movie ratings as well RMSE had been minimized less than the Naive Baseline Model nor the regularized biased model„

So after training different models, we find out that movie_id and user_id gives a desirable value of RMSE Without regularization, but when applying regularization and adding the genres predictor, it make possible to reach a lower value of RMSE „ but Matrix Factorization model gives us a better result for RMSE so this approach is effective way to make prediction on movie ratings as well RMSE had been minimized less than the Naive Baseline Model nor the regularized biased model,

Future Work

The effects of genres and age could be further explored to make improvements on the performance in regularized model and to check if it would give a better results. Also trying to build more machine learning models to find if we can get a value of RMSE less than what we tried in the previews models, so we could make the prediction process is more effective and more accurate. also the ensemble method should also be considered to apply on the MovieLens dataset, to combine the advantages of various models and enhance the overall performance of prediction.

References:

https://www.sciencedirect.com/science/article/pii/S1110866516300470

https://www.sciencedirect.com/science/article/pii/S1045926X14000901

https://rafalab.github.io/dsbook/clustering.html

https://learning.edx.org/course/course-v1:HarvardX+PH125.9x+1T2021/block-v1:HarvardX+PH125.9x+1T2021+type@sequential+block@e8800e37aa444297a3a2f35bf84ce452/block-v1:HarvardX+PH125.9x+1T2021+type@vertical+block@e9abcdd945b1416098a15fc95807b5db

https://cran.r-project.org/web/packages/tune/tune.pdf

https://www.rdocumentation.org/packages/e1071/versions/1.7-7/topics/tune

https://ggplot2.tidyverse.org/reference/aes_group_order.html

https://www.r-bloggers.com/2020/03/how-to-standardize-group-colors-in-data-visualizations-in-r/

https://ggplot2.tidyverse.org/reference/geom_jitter.html

https://www.journaldev.com/45290/predict-function-in-r

https://rpubs.com/Jango/486734

http://rstudio-pubs-static.s3.amazonaws.com/493427_68e66956f18044ea8d21ee64c0337a1e.html#abstract

https://rstudio-pubs-static.s3.amazonaws.com/288836_388ef70ec6374e348e32fde56f4b8f0e.html#creating_models

https://mono33.github.io/MovieLensProject/#abstract

https://www.rpubs.com/rezapci/Data_Science_Machine_Learning_HarvardX

https://rpubs.com/delongmeng/557151

https://rpubs.com/Papacosmas/harvard

https://rstudio-pubs-static.s3.amazonaws.com/593016_55f82043d74f401cbafe347e5e025d90.html