MovieLense Data Rating Prediction Project

Yogesh Kumar Malkoti

May 24, 2019

Introduction

The report is a capstone project of the EdX course HarvardX: PH125.9x Data Science. Its establishes the student's ability and his/her understanding of different algorithm and how they can use individual or ensembled approach to tackle and solve different data problems along with the help of programming language R which mimics these algorithms.

In this task we have to analyse a dataset called Movielans containing millions of movier ratings by individual users. The insights from these analysis are used to generate predictions of movies which will be then are compared with the actual ratings to check the quality of the prediction algorithm.

The data set has more than "ten million" ratings which make is very difficult for a normal laptop to calculate algorithms like glm. The R program probably will crash if we try to run these algorithm with this huge data set.

Summary

The report is split in three sections.

1. In first section we will load the dataset and preform some further analysis.
2. In Second section we will perform an exploratory data analysis to understand the data structure to gain some useful insights of it.
3. In third and final section, we will use a machine learning algorithm that will generate predictions which are then exported for a final test. Since this a large data set and so I have decided to use "Penalized Root Mean Squared Error" approach for this problem data set, as I am using this on my laptop. This algorithm achieved a RMSE of 0.86.

Regularization

Method Description

Due to the large dataset, an efficient method was needed to predict movie ratings based on an user id and a movie id. The regularization approach is based on the mean movie rating. This average is adjusted for user-effects and movie-effects. Analysis shows that ratings from users who rate just a few movies and movies with a small number of total ratings tend to have more volatile ratings than users who rate lots of movies and movies with lots of ratings. To adjust for these effects, a penalty - lambda - is taken into account. Once a prediction is made, it has to be translated from a continuous number into a number from 0.5 to 5.0 in 0.5 steps. The so derived predicted values get compared with the actual value to calculate an accuracy value.

Step 1) Download MovieLens Data

The dataset movielens gets split into a training-testset called edx and a set for validation purposes called validation.

My edits due to laptop security policy :(Since I am doing this project on my office laptop, the access to the data available on grouplense is restricted by policy server. Hance I have to download the data manually from other machine and send to this laptop. I have unzipped the data and placed the files at certain location on

my hard drive.. I am using two variable rating_file and movie_file which gives the absolute path of these files on my laptop.

```
####################################
# Create edx set and 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 1.2.1 --
```

```
## v ggplot2 3.1.1       v purrr   0.3.2
## v tibble  2.1.1       v dplyr   0.8.0.1
## v tidyr   0.8.3       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
```

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

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

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

```
# 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")



rating_file<- "C:\\Users\\yogesh.malkoti\\Documents\\capstone\\ml-10m\\ml-10M100K\\ratings.dat"

movie_file<-  "C:\\Users\\yogesh.malkoti\\Documents\\capstone\\ml-10m\\ml-10M100K\\movies.dat"

ratings <- read.table(text = gsub("::", "\t", readLines(rating_file)),
                      col.names = c("userId", "movieId", "rating", "timestamp"))
```

```
movies <- str_split_fixed(readLines(movie_file), "\\::", 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")

## My edits due to laptop security policy :) ####


# 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, 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(ratings, movies, test_index, temp, movielens, removed)
```

Step 2) Exploratory Data Analysis

This sections helps to understand the structure of the movilens dataset in order to use the insights for a better prediction of movie ratings. These insights are also part of the overall assesment section.

Let's find number of rows and columns in the edx dataset.

```
paste('The edx dataset has',nrow(edx),'rows and',ncol(edx),'columns.')
```

```
## [1] "The edx dataset has 9000055 rows and 6 columns."
```

This is a pretty big data set.

Let's find number of zeros and threes given in the edx dataset.

```
paste('There are', sum(edx$rating == 0), 'count for 0 rating and',
sum(edx$rating == 3),'count for 3 rating.')
```

```
## [1] "There are 0 count for 0 rating and 2121240 count for 3 rating."
```

Let's find the number of unique movies in the data set.

```
edx %>% summarize(unique_movies = n_distinct(movieId))
```

```
##   unique_movies
## 1         10677
```

Let's find number of unique users in the data set.

```
edx %>% summarize(unique_users = n_distinct(userId))
```

```
##   unique_users
## 1        69878
```

Let's find the movie which has greatest rating.

```
edx %>% group_by(title) %>% summarise(number = n()) %>%
  arrange(desc(number))
```

```
## # A tibble: 10,676 x 2
##    title                                                     number
##    <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
```

Lets find five most given ratings in order from most to least

```
head(sort(-table(edx$rating)),5)
```

```
##
##        4        3        5      3.5        2
## -2588430 -2121240 -1390114  -791624  -711422
```

In general, half star ratings are less common than whole star ratings (e.g., there are fewer ratings of 3.5 than there are ratings of 3 or 4, etc.).

```
table(edx$rating)
```

```
##
##     0.5       1     1.5       2     2.5       3     3.5       4     4.5
##   85374  345679  106426  711422  333010 2121240  791624 2588430  526736
##       5
## 1390114
```

Data Analysis

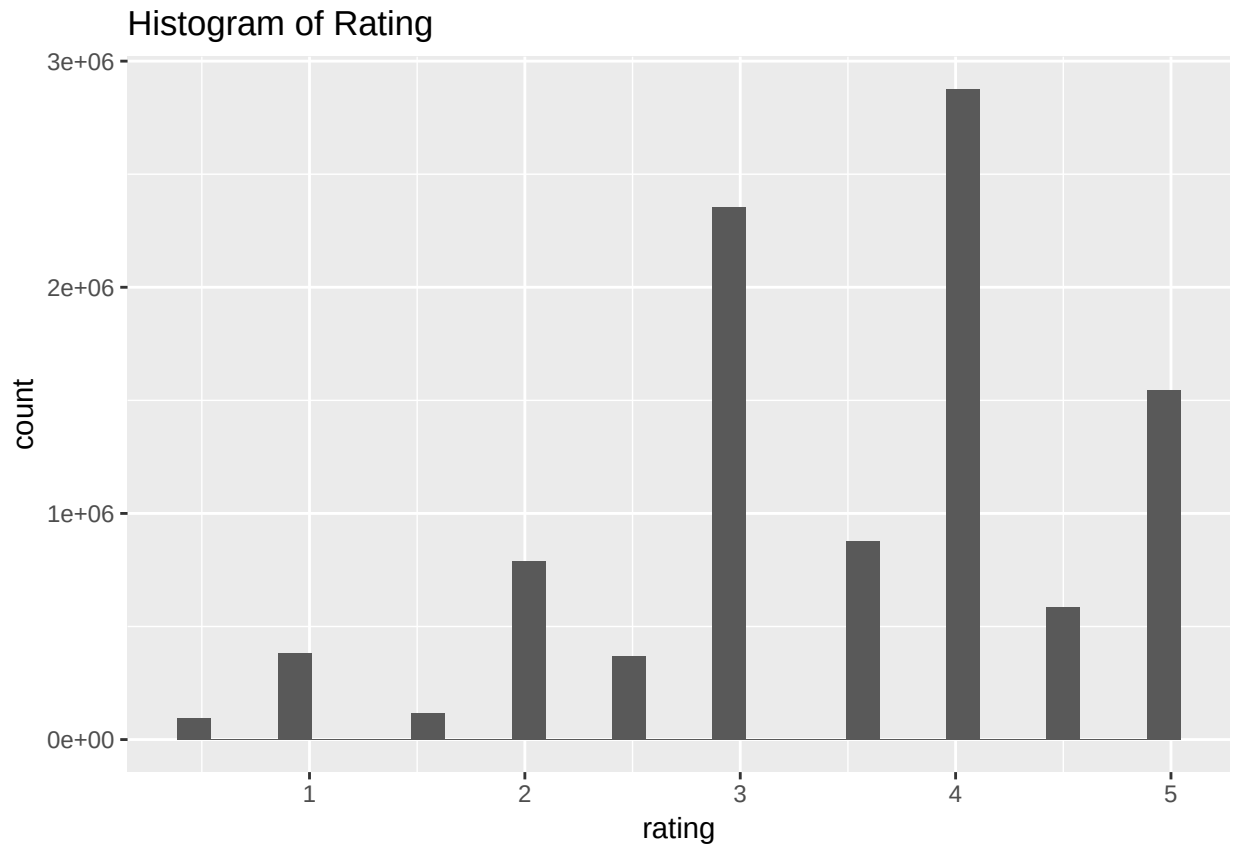
```
str(movielens)
```

```
## 'data.frame':    10000054 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983653 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller"
```

The movielens dataset has more than 10 million ratings. Each rating comes with a userId, a movieId, the rating, a timestamp and information about the movie like title and genre.

```
movielens %>% ggplot(aes(rating)) +
              geom_histogram() +
              ggtitle("Histogram of Rating")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

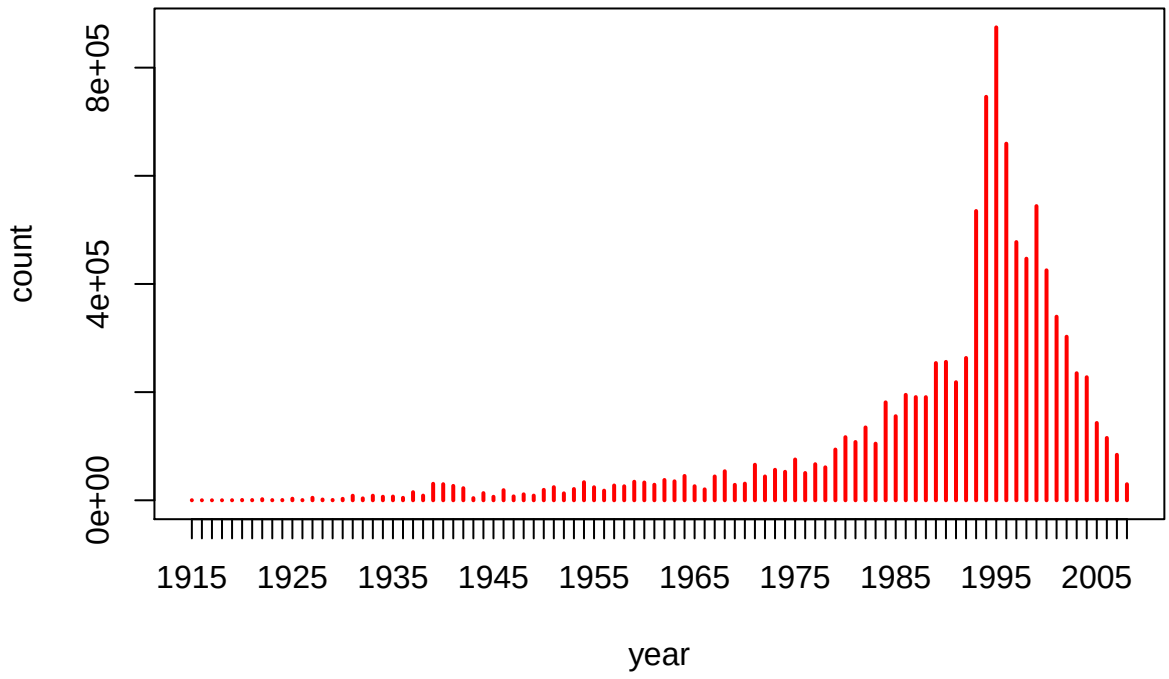



```
summary(movielens$rating)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.000   4.000   3.512   4.000   5.000
```

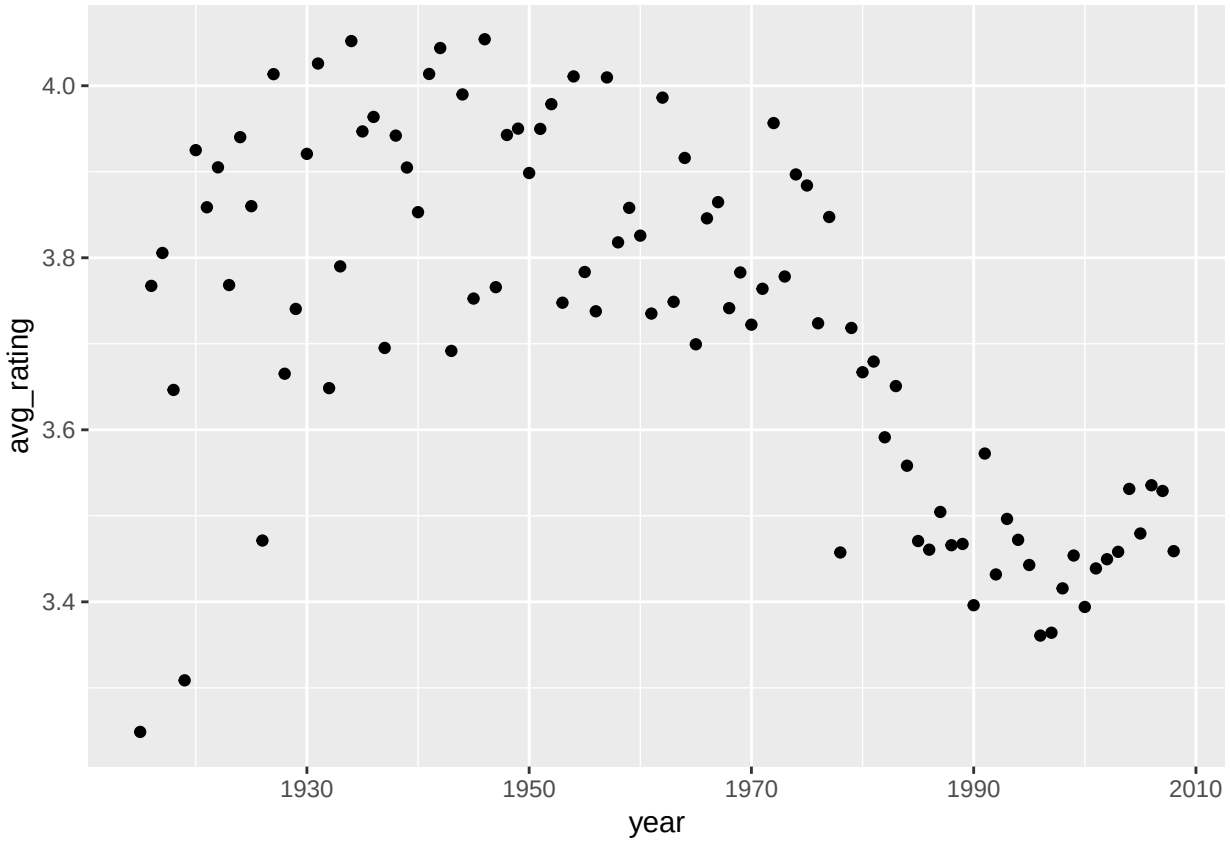
Ratings range from 0.5 to 5.0. The difference in meadian an mean shows that the distribution is skewed towards higher ratings. The chart shows that whole-number ratings are more common that 0.5 ratings.

```
movielens$year <- as.numeric(substr(as.character(movielens$title),nchar(as.character(movielens$title))-4

plot(table(movielens$year),
     col = I("red"),
     xlab="year", ylab="count")
```



More recent movies get more user ratings. Movies earlier than 1930 get few ratings, whereas newer movies, especially in the 90s get far more ratings.

```
library(dplyr)
avg_ratings <- movielens %>% group_by(year) %>% summarise(avg_rating = mean(rating))
avg_ratings %>% ggplot(aes(x=year, y=avg_rating)) + geom_point()
```



Movies from earlier decades have more volatile ratings, which can be explained by the lower frequence of movie ratings. However, since the project is measured by the accuarcy, this volatility has to be taken into account.

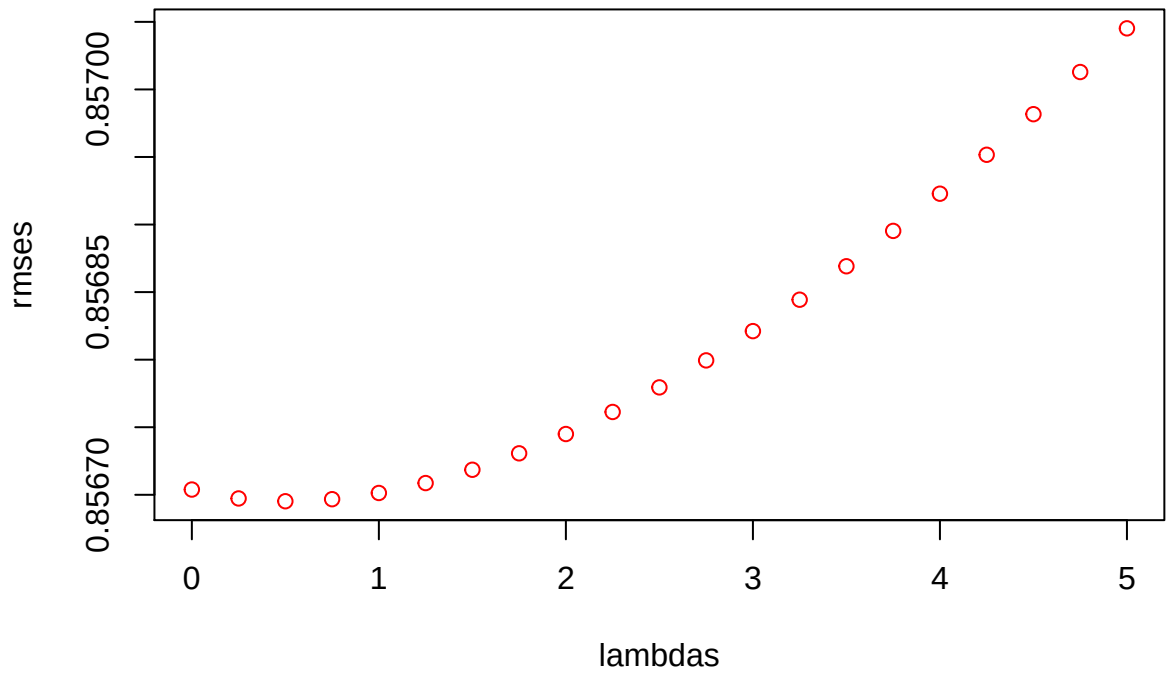
Results

The challenge was to get the highest accuracy, which is measured as the number of exact matches of predicted ratings vs ratings of the validation set. Since the predictions based on earlier algorithms depicts that the best and worst movies were rated by very few users as these were mostly obscure movies.therefore larger estimates of movie, negative or positive, are more likely when fewer users rate the movies.These are basically noisy estimates that we should not trust, especially when it comes to prediction. The most promising algorithm in this problem is regularization. Regularization permits us to penalize large estimates that come from small sample sizes and can be run on small laptop.

Choose Optimal Penalty Rate Lambda.

```
#Root Mean Square Error Loss Function
RMSE <- function(true_ratings, predicted_ratings){
        sqrt(mean((true_ratings - predicted_ratings)^2))
      }

lambdas <- seq(0, 5, 0.25)

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

```
  #Calculate the mean of ratings from the edx training set
  mu <- mean(edx$rating)

  #Adjust mean by movie effect and penalize low number on ratings
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  #ajdust mean by user and movie effect and penalize low number of ratings
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - (b_i + mu))/(n()+l))


  #predict ratings in the training set to derive optimal penalty value 'lambda'
  predicted_ratings <-
    edx %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

  return(RMSE(predicted_ratings, edx$rating))

})

plot(lambdas, rmses,
    col = I("red"))
```




```
lambda <- lambdas[which.min(rmses)]
paste('Optimal RMSE of',min(rmses),'is achieved with Lambda',lambda)
```

```
## [1] "Optimal RMSE of 0.856695227644159 is achieved with Lambda 0.5"
```

Predictions will be done using this value.

Apply Lamda on Validation set for Data-Export

```
lambda <- lambda

pred_y_lse <- sapply(lambda,function(l){

  #Derive the mearn from the training set
  mu <- mean(edx$rating)

  #Calculate movie effect with optimal lambda
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  #Calculate user effect with optimal lambda
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - (b_i + mu))/(n()+l))
```

```
  #Predict ratings 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 #validation


  return(predicted_ratings)

})
```

RMSE of Validation data.

```
paste('RMSE for validation data achieved', RMSE(pred_y_lse, validation$rating))
```

```
## [1] "RMSE for validation data achieved 0.86522255159723"
```

Generated predictions for validation dataset.

Export Predictions

```
# Ratings will go into the CSV submission file below:

write.csv(validation %>% select(userId, movieId, rating) %>% mutate(predcit_rating = pred_y_lse),
          "predicted_ratings.csv", na = "", row.names=FALSE)
```

Conclusion

Project objective was to predict movie ratings from a huge record of movies. The size of the dataset restricted the machin learning algorithms my laptop was able to perform on the dataset. The regularization approach was able to come up with ratings that are near the true ratings. However, accuracy is measured as absolute difference between the predicted value and the acutal value. The transformation from a continuous number to the actual rating did not result in a high overall accuarcy, although the prediction in terms of real numbers makes sense.