HarvardX: Data Science
MovieLens Rating Prediction Project

Sabiny Chungath Abdul Jaleel

August 10, 2020

Introduction

This is the final project in Professional Certificate in Data Science course from Harvard University using R programming language. The objective of this project is to analyse the 'Movielens' dataset and predict the movie's rating based on the given dataset. This dataset contains user id,movie id,ratings,title of the movie and genres i.e movie type such as comic, action etc. The user information is stored in userId; the movie information is both in movieId and title columns. The rating date is available in timestamp. Each movie is tagged with one or more genre in the genres column.

The machine learning (ML) approach is to train an algorithm using this dataset to make a prediction when we do not know the outcome. As the size of a 'Movielens' dataset is large, we need to have a proper method to predict the movie ratings from the given dataset. Hence, to get best possible accuracy, we need to compare few machine learning algorithms to predict movie ratings.

Root Mean Square Error - RMSE has been used to evaluate the performance of the ML algorithms. This is one of the commonly used measure of the differences between predicted values by a model and the observed values. RMSE is a measure of accuracy to compare forecasting errors of different models for a dataset. Lower RMSE is better than a higher one. The effect of each error on RMSE is proportional to the size of the squared error, thus larger errors have a disproportionately large effect on RMSE. The evaluation criteria for this algorithm is a RMSE expected to be lesser than 0.86490.

Methods & Analysis

Due to the large dataset, an efficient method is needed to predict movie ratings based on user id and movie id. The penalized least squares approach is based on the mean movie rating. This average is adjusted for user-effects and movie-effect. For the result, a penalty - lambda has been taken to the model.

Regularisation and a penalty term will be applied to the models in this project. Regularizations are techniques used to reduce the error by fitting a function appropriately on the given training set and avoid overfitting. This is used for tuning the function by adding an additional penalty term in the error function.

Loading the Dataset

```
# Load the edx & validation data sets using the provided script

############################################################
# Create edx set, validation set, and submission file
############################################################
# Note: this process could take a couple of minutes

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

library(tidyverse)
library(caret)
library(lubridate)

# 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 3.6 or earlier
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(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")
```

To predict the most accurate movie rating, the MovieLens dataset will be splitted into 2 subsets that will be the "edx", a training subset to train the algorithm, and "validation" a subset to test the movie ratings.

```
# Validation set will be 10% of MovieLens data
set.seed(1)

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

# 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)
```

The above code gives the dataset for training and testing our dataset. It also removes the unnecessary files from the working directory, which is always a good coding practice.

Data Exploration

```
names(edx)
```

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

The edx data includes userId,movieId,rating,timestamp and genres.

```
class(edx)
```

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

```
#Finding the number of unique users that provided ratings and how many unique movies were rated
edx %>% summarize(n_users = n_distinct(userId), n_movies = n_distinct(movieId))
```

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

The total of unique movies and users in the edx subset is about 69878 unique users and about 10677 different movies. A summary of the subset confirms that there are no missing values.

```
# edx summary statistics
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18122   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35743   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35869   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53602   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000061     Length:9000061
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

Data Cleaning

In this section we will take the first look at the loaded data frames. We will also perform necessary cleaning and some transformations so that the data better suits our needs.

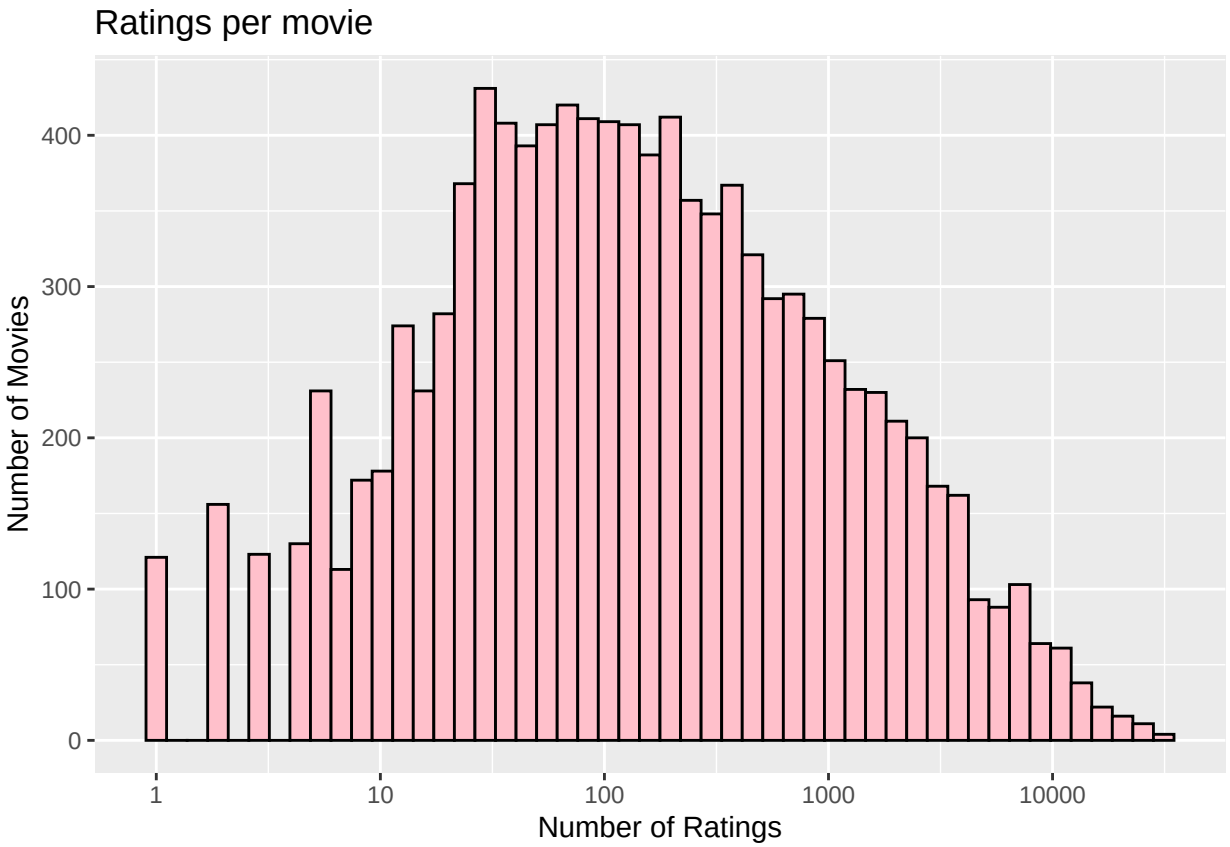
```
validation_set <- validation %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))

summary(validation_set)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18127   1st Qu.:  653   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35719   Median : 1835   Median :4.000   Median :1.036e+09
##  Mean   :35878   Mean   : 4121   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53649   3rd Qu.: 3633   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres                date
##  Length:999993      Length:999993      Min.   :1995-01-08 00:00:00
##  Class :character   Class :character   1st Qu.:2000-01-02 00:00:00
##  Mode  :character   Mode  :character   Median :2002-10-27 00:00:00
##                                        Mean   :2002-09-23 11:03:16
##                                        3rd Qu.:2005-09-18 00:00:00
##                                        Max.   :2009-01-04 00:00:00
```

Data Visualization

```
## histogram of number of ratings by movieId
edx %>%
  count(movieId) %>% ggplot(aes(n)) +
  geom_histogram(fill = "pink", bins = 50, color = "black") +
  scale_x_log10() +
  xlab("Number of Ratings") +
  ylab("Number of Movies") +
  ggtitle("Ratings per movie")
```

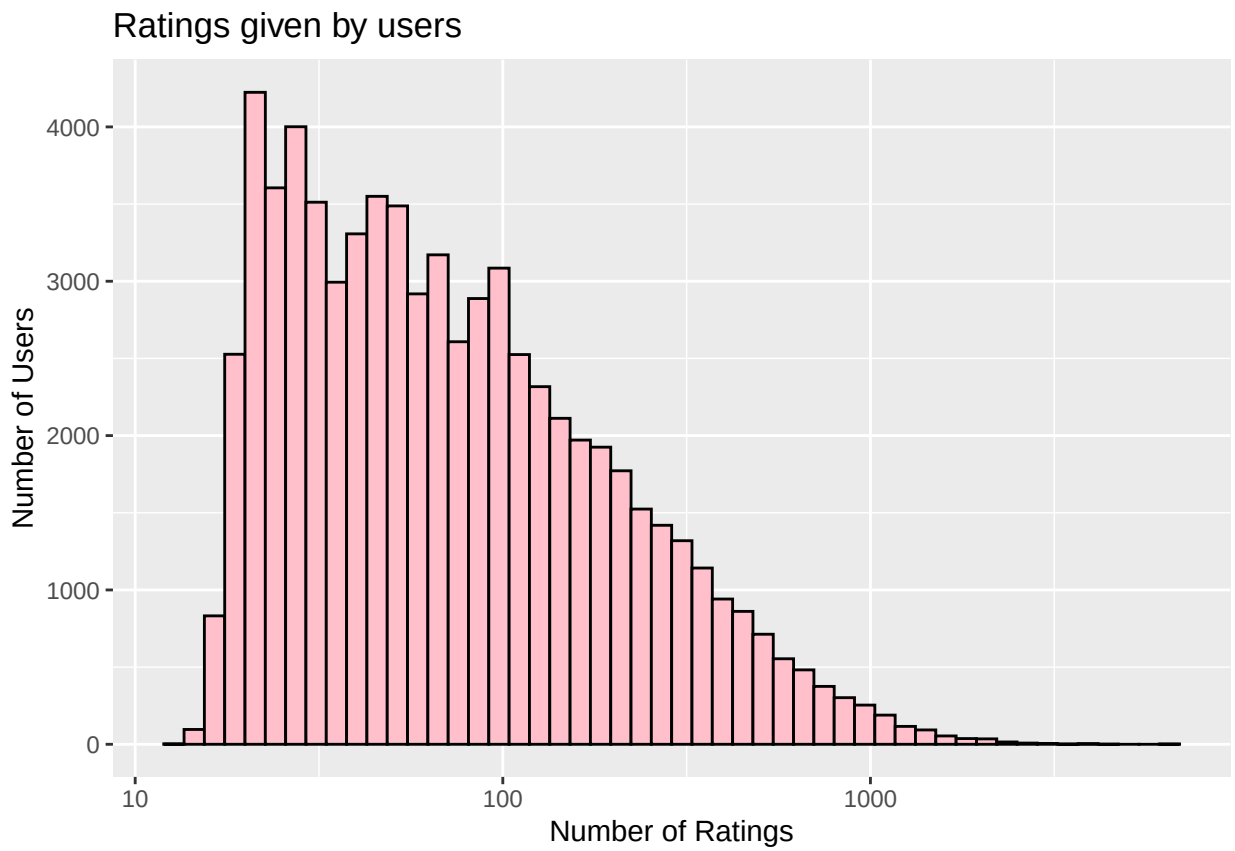


Visual exploration of the number of ratings by movieId:- As the above histagram shows, some movies get more rating than others. This indicates that the presence of movies effects.

There are 10677 different movies in the edx set. We know from intuition that some of them are rated more than others, since many movies are watched by few users and blockbusters tend to have more ratings.

```
# histogram of number of ratings by userId

edx %>%
  count(userId) %>% ggplot(aes(n)) +
  geom_histogram(fill = "pink", bins = 50, color = "black") +
  scale_x_log10() +
  xlab("Number of Ratings")+
  ylab("Number of Users")+
  ggtitle("Ratings given by users")
```

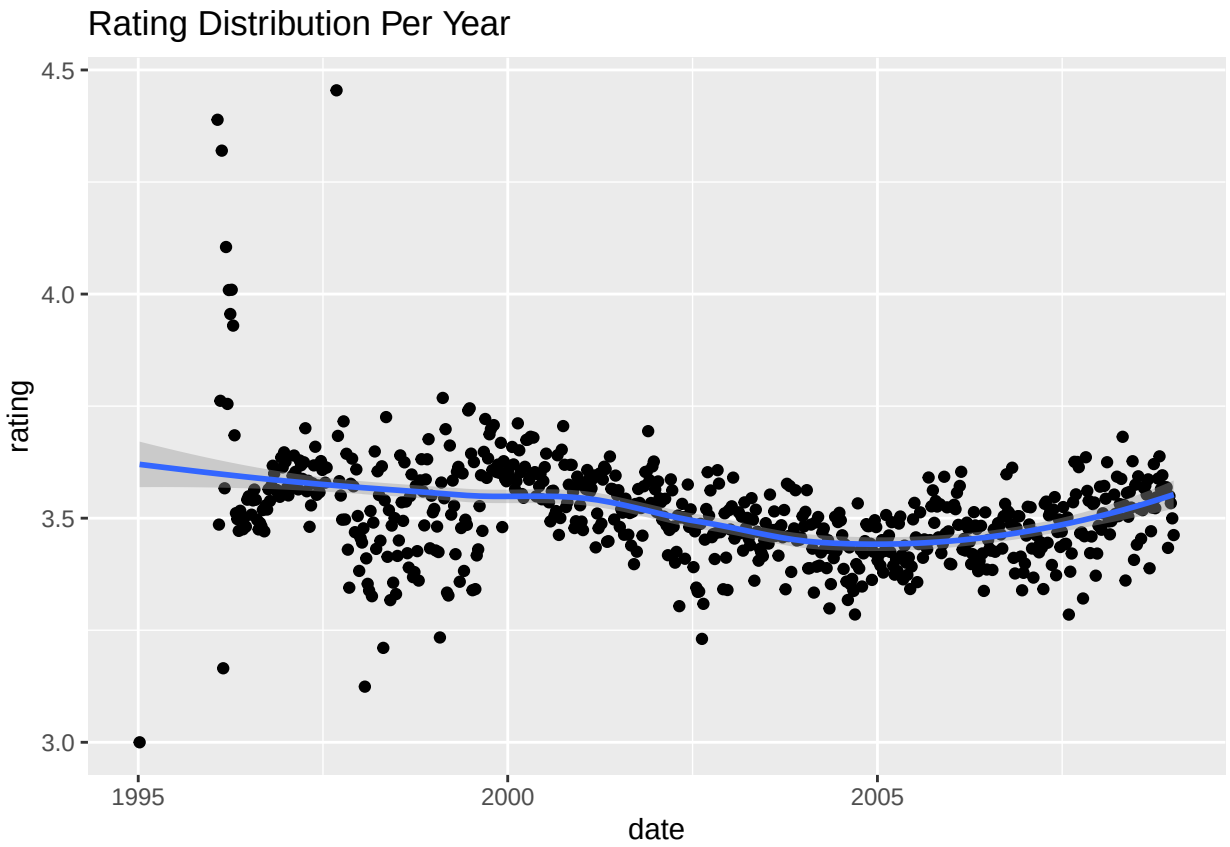


Visual exploration of the number of ratings by UserId:- As the above histogram shows, some users rate more than others based on thier preferences. This indicates that the presence of users effect model.

Above histograms show that not every user is equally active. Some users have rated very few movie and their opinion may bias the prediction model results.

There are 69878 different users are in the edx set.The majority of users rate few movies, while a few users rate more than a thousand movies.

```
edx %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating),.groups='drop') %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Rating Distribution Per Year")
```



We know that the edx set contains the timestamp variable which represents the time and data in which the rating was provided. By using as_datetime() function in the lubridate package , we can have each timestamp in the right format. Here used the geom_point() to create scatterplot.

Upon analysing the above visualization, we can see that there is some evidence of a time effect in the plot, but there is not a strong effect of time.

Model Development

The goal is to train a machine learning algorithm that predicts user ratings using the inputs of a provided subset (edx dataset provided by the staff) to predict movie ratings in a provided validation set.

The loss-function computes the RMSE, defined as follows:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

with N being the number of user/movie combinations and the sum occurring over all these combinations. The RMSE is our measure of model accuracy. We can interpret the RMSE similarly to a standard deviation: it is the typical error we make when predicting a movie rating. If its result is larger than 1, it means that our typical error is larger than one star, which is not a good result. The written function to compute the RMSE for vectors of ratings and their corresponding predictions is:

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

1.Mean Rating Model

Let's start by building the simplest possible recommendation system. We can use a model based approach as listed below:

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

with $\epsilon_{u,i}$ independent error sample from the same distribution centered at 0 and μ the "true" rating for all movies. We know that the estimate that minimize the RMSE is the least square estimate of $Y_{u,i}$, in this case, is the average of all ratings.

```
# calculating the average of all ratings of the edx set
mu <- mean(edx$rating)
#calculating the RMSE for mean
neive_RMSE <- RMSE(edx$rating, mu)
neive_RMSE
```

```
## [1] 1.060393
```

```
#Here, we represent results table with the first RMSE:
#summarizing the rmse
RMSE_results <- data.frame(Method = "Mean", RMSE = neive_RMSE)
```

This give us our baseline RMSE to compare with next modelling approaches.

In order to do better than simply predicting the average rating, we incorporate some of insights we gained during the data analysis.

2.Movie, User and time Effect Model

Since we know that some movies are generally rated higher than others. We can augment our previous model by the movie average to determine movie bias b_i,and on each user's average to determine user bias b_u. For current purposes, no attempt was made to split these into individual effects, and each combo was treated as its own genre. The resulting variable is called "b" (as bias) for each movie "i" b_i, that represents average ranking for movie i:

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

```
#Movie effect model
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu),.groups='drop')
#predicted ratings of movie for RMSE
predicted_ratings_bi <- validation_set %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(pred = mu + b_i)
#calculating the RMSE for movie effect
movie_RMSE <- RMSE(validation_set$rating, predicted_ratings_bi$pred)
movie_RMSE
```

```
## [1] 0.9437046
```

```
#Movie_user effect model
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i),.groups='drop')
#predicted ratings of movie and user for RMSE
predicted_ratings_bu <- validation_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u)
#calculating the RMSE for movies and users effects
movie_user_RMSE <- RMSE(validation_set$rating, predicted_ratings_bu$pred)
movie_user_RMSE
```

```
## [1] 0.8655329
```

```
#Movie+user+time effect model
time_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(b_t = mean(rating - mu - b_i - b_u),.groups='drop')
#predicted ratings of movie, user and time for RMSE
  predicted_ratings_bt <- validation_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(time_avgs, by='date') %>%
  mutate(pred = mu + b_i + b_u + b_t)
  #calculating the RMSE for movies, users and time effects
movie_user_time_RMSE <- RMSE(validation_set$rating, predicted_ratings_bt$pred)
movie_user_time_RMSE
```

```
## [1] 0.865457
```

When making predictions, we need one number, one prediction, not an interval. For this we introduce the concept of regularization, that permits to penalize large estimates that come from small sample sizes. The general idea is to add a penalty for large values of b_i to the sum of squares equation that we minimize. So having many large b_i, make it harder to minimize. Regularization is a method used to reduce the effect of overfitting.

3.Regularization Model

Regularization penalizes records which stray far from the mean but have few associated ratings. The general idea behind regularization is to constrain the total variability of the effect sizes.

Movie & User Regularization Model

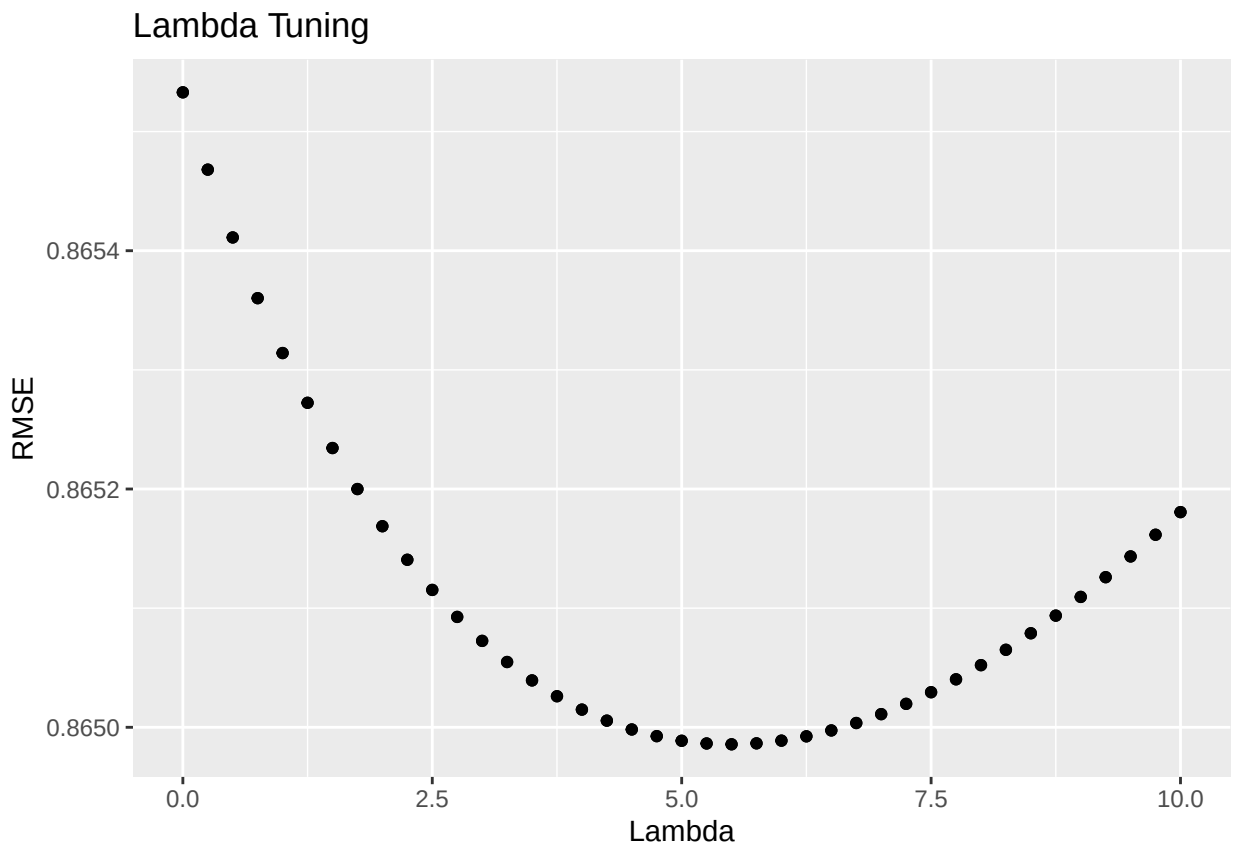
So estimates of b_i and b_u are caused by movies with very few ratings and in some users that only rated a very small number of movies. Hence this can strongly influence the prediction. The use of the regularization permits to penalize these aspects. We should find the value of lambda (that is a tuning parameter) that will minimize the RMSE. This shrinks the b_i and b_u in case of small number of ratings.

```
# Optimize lambda by minimizing RMSE
# lambda is a tuning parameter.
lambdas <- seq(0, 10, 0.25)
l_RMSE <- sapply(lambdas, function(l){
   mu <- mean(edx$rating)
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l),.groups='drop')

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

  predicted_ratings <- validation_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u)
    return(RMSE(validation_set$rating, predicted_ratings$pred))
})
```

```
#We plot RMSE vs lambdas to select the optimal lambda
qplot(lambdas, l_RMSE)+ geom_point() +
  xlab('Lambda') + ylab("RMSE") + ggtitle("Lambda Tuning")
```




```
#For the full model, the optimal  lambda is:
lambda <- lambdas[which.min(l_RMSE)]
lambda
```

```
## [1] 5.5
```

```
# now calculate the regularized accuracy with the best lambda
Reg_user_RMSE <- min(l_RMSE)
Reg_user_RMSE
```

```
## [1] 0.8649857
```

Movie, User and Time Regularization Model

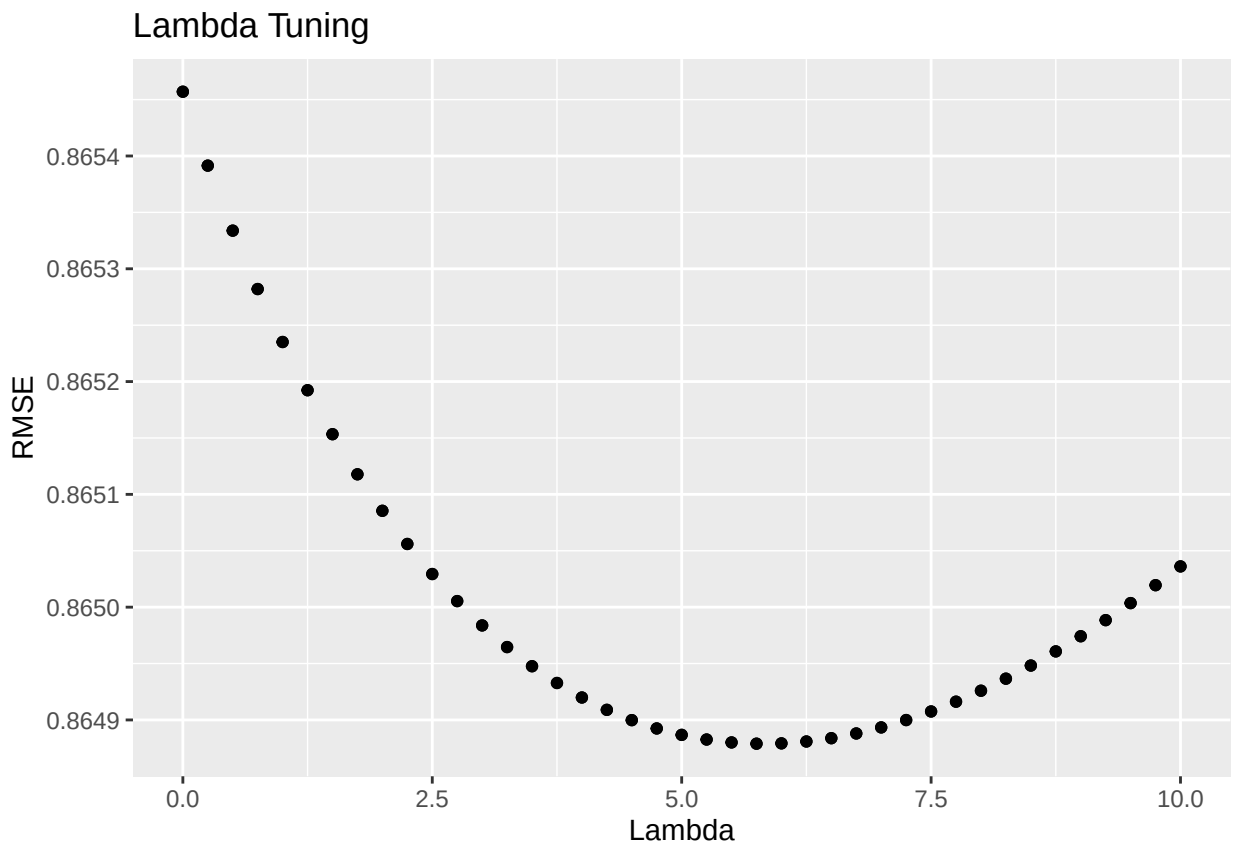
```
# Optimize lambda by minimizing RMSE
# lambda is a tuning parameter.
lambdas <- seq(0, 10, 0.25)
mur_RMSE <- sapply(lambdas, function(l){
   mu <- mean(edx$rating)
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l),.groups='drop')
```

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

  b_t  <- edx %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(b_t = mean(rating - mu - b_i - b_u),.groups='drop')

   predicted_ratings <- validation_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_t, by='date') %>%
    mutate(pred = mu + b_i + b_u + b_t)
    return(RMSE(validation_set$rating, predicted_ratings$pred))
})
```

```
#We plot RMSE vs lambdas to select the optimal lambda
qplot(lambdas, mur_RMSE)+ geom_point() +
  xlab('Lambda') + ylab("RMSE") + ggtitle("Lambda Tuning")
```



```r
#For the full model, the optimal  lambda is:
lambda <- lambdas[which.min(mur_RMSE)]
lambda
```

```
## [1] 5.75
```

```r
# now calculate the regularized accuracy with the best lambda
final_RMSE <- min(mur_RMSE)
final_RMSE
```

```
## [1] 0.8648789
```

we observe that regularization gets down the RMSE's value to 0.8648789.

Results

The RMSE values of all the represented models are the following:

Method	RMSE
Mean	1.0603926
Movie Effect Model	0.9437046
Movie+User Effect Model	0.8655329
Movie+User+Time Effect Model	0.8654570
Regularized Movie+User Effect Model	0.8649857
Regularized Movie+User+Time Effect Model	0.8648789

We therefore found the lowest value of RMSE that is 0.8648789. So we can confirm that the final model for our project is the following:

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

This model work well if the average user doesn't rate a particularly good movie with a large positive b_i, by disliking a particular movie.

Conclusion

The goal of this project was to develop a best movie rating recommendation system to predict movie rating for the 10M version of 'Movielens' dataset, therefor machine learning algorithm has been built to predict movie ratings with MovieLens dataset.

The optimal model characterised by the lowest RMSE value (0.8648789) lower than the initial evaluation criteria (0.86490) given by the goal of the this project. The RMSE table shows an improvement of the model over different assumptions. The simplest model 'Using mean only' calculates the RMSE more than 1, then incorporating 'Movie effect', 'User effect' and 'Time effect' for better accuracy. This is substantial improvement given the simplicity of the model. A deeper insight into the data revealed some data point in the features have large effect on errors. Therefore, a regularization model was used to penalize this data points. The final RMSE is 0.8648789. This implies we can trust our prediction for movie rating system.