MovieLens Capstone Project

Gayathri Mamidibatthula

07/12/2021

Introduction and Overview:

Recommender systems are the systems that are designed to recommend things to the user based on many different factors. These systems predict the most likely product that the users are most likely to purchase and are of interest to. Companies like Netflix, Amazon, etc. use recommender systems to help their users to identify the correct product or movies for them.

The recommender system deals with a large volume of information present by filtering the most important information based on the data provided by a user and other factors that take care of the user's preference and interest. It finds out the match between user and item and imputes the similarities between users and items for recommendation.

Dataset used:

GroupLens Research has collected and made available rating data sets from the MovieLens web site (https://movielens.org). The data sets were collected over various periods of time, depending on the size of the set.

The data given is the movielens dataset, containing the rating given by users on movies of various genres in the range of 0-5.

Executive Summary:

The goal of the project is the predict the rating given by users on movies in the validation data. Each user can rate more than once, and each movies can have multiple ratings from different users.

Key Steps performed:

1. Loading the Dataset
2. Exploratory Data Analysis(EDA)
3. Data Wrangling
4. Building the ML Models
5. Applying the Final Model to the Validation Set

LOADING THE DATASET

Create edx set, validation set, and submission file :

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

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

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.2     v dplyr   1.0.6
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

## -- 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")
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")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
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)
```

```
##Loading libraries

library(tidyverse)
library(ggplot2)
library(dplyr)
library(markdown)
library(knitr)
library(caret)
library(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
```

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
#Checking for any null values in the dataset:
anyNA(edx)
```

```
## [1] FALSE
```

2. EXPLORATORY DATA ANALYSIS(EDA)

```r
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1                 Comedy|Romance
## 2          Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5        Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7        Children|Comedy|Fantasy
```

```r
head(validation)
```

```
##   userId movieId rating timestamp
## 1      1     231      5 838983392
## 2      1     480      5 838983653
## 3      1     586      5 838984068
## 4      2     151      3 868246450
## 5      2     858      2 868245645
## 6      2    1544      3 868245920
##                                                      title
## 1                                     Dumb & Dumber (1994)
## 2                                     Jurassic Park (1993)
## 3                                        Home Alone (1990)
## 4                                           Rob Roy (1995)
## 5                                    Godfather, The (1972)
## 6 Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                   genres
## 1                                 Comedy
## 2        Action|Adventure|Sci-Fi|Thriller
## 3                        Children|Comedy
## 4                Action|Drama|Romance|War
## 5                            Crime|Drama
## 6 Action|Adventure|Horror|Sci-Fi|Thriller
```

```r
## No of rows and columns in the dataset:

rows<- dim(edx)[1]
columns<- dim(edx)[2]
summary(rows)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   9e+06   9e+06   9e+06   9e+06   9e+06   9e+06
```

```
summary(columns)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       6       6       6       6       6       6
```

```
## No of unique movies and users:

uniqueMovies <- length(unique(edx$movieId))
uniqueUsers<- length(unique(edx$userId))
head(uniqueMovies)
```

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

```
head(uniqueUsers)
```

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

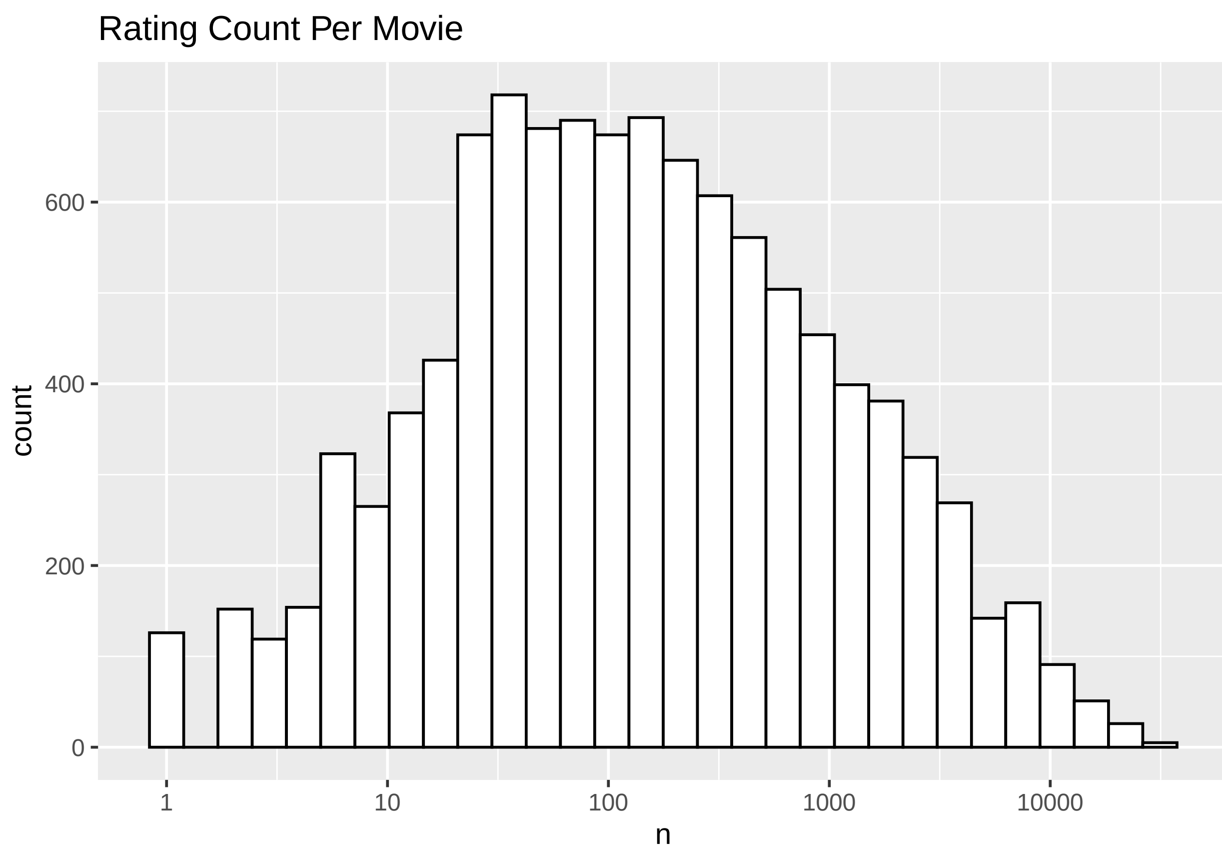
Movie with greatest number of ratings:

```
numRatings <- edx %>% group_by(movieId) %>%
  summarize(numRatings = n(), movieTitle = first(title)) %>%
  arrange(desc(numRatings)) %>%
  top_n(10, numRatings)
  head(numRatings)
```

```
## # A tibble: 6 x 3
##   movieId numRatings movieTitle
##     <dbl>      <int> <chr>
## 1     296      31362 Pulp Fiction (1994)
## 2     356      31079 Forrest Gump (1994)
## 3     593      30382 Silence of the Lambs, The (1991)
## 4     480      29360 Jurassic Park (1993)
## 5     318      28015 Shawshank Redemption, The (1994)
## 6     110      26212 Braveheart (1995)
```

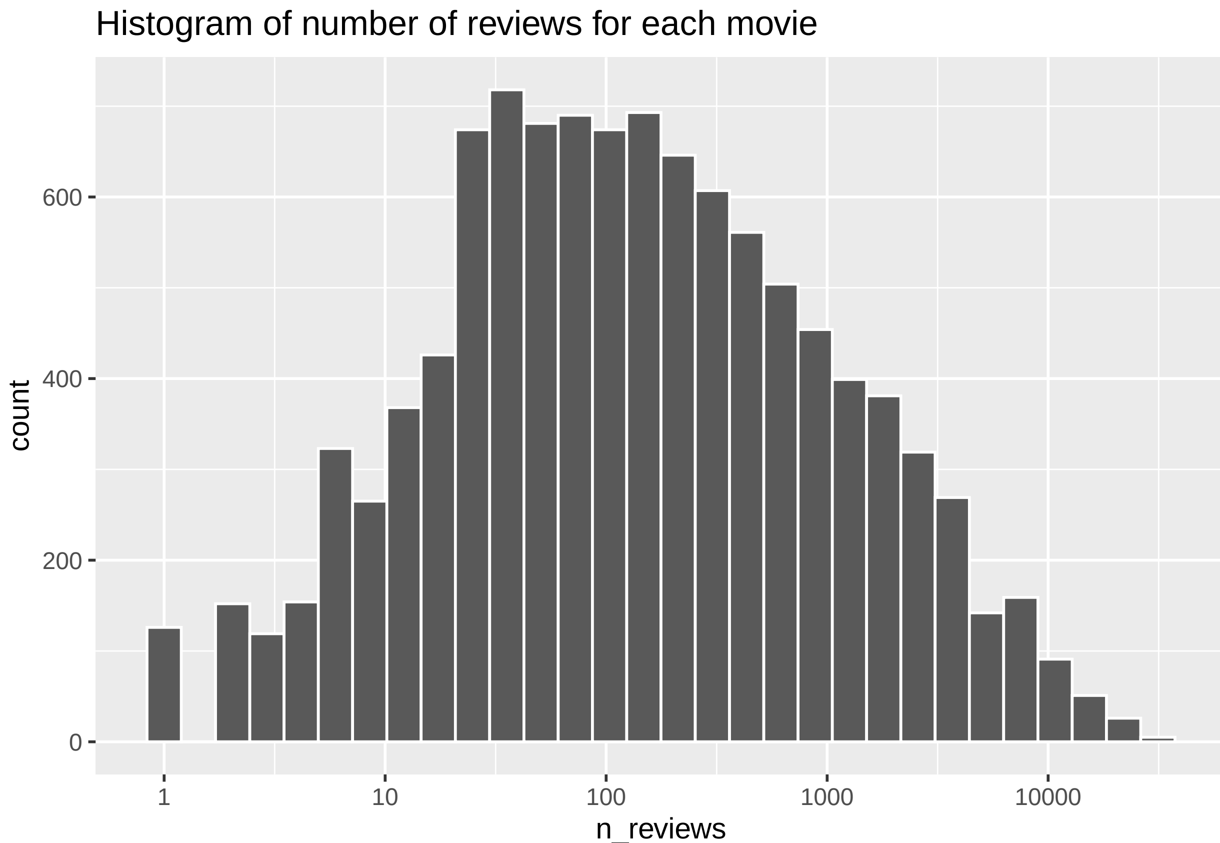
Histogram representation of no of ratings based on movies:

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



Histogram of number of reviews per movie id:

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



Most movies have below 1000 number of reviews, while some have more than 10000.

3. DATA WRANGLING

```
#Splitting edx sample to train and test set with ratio 80:20
train_index <- createDataPartition(y=edx$rating, times=1, p=0.8,list=FALSE)

train <- edx[train_index,]
test <- edx[-train_index,]

## Making sure we only include the users and movies in test set, which are also in training set.
## Extra entries are removed from test set using semi-join function

test <- test %>%
  semi_join(train, by = "movieId") %>%
  semi_join(train, by = "userId")
```

```
##Data Wrangling

class(train$timestamp)
```

```
## [1] "integer"
```

```
##Timestamp returns an integer,hence changing timestamp to datetime, and using only the year
train$timestamp <- year(as_datetime(train$timestamp))
```

```
#extracting release year from title
pattern <- "(?<=\\()\\d{4}(?=\\))"
train$release_year <- train$title %>% str_extract(pattern) %>% as.integer()


#one-hot encode the genres column
train$genres <- str_split(train$genres, pattern="\\|")
one_hot_genres <- enframe(train$genres) %>%
  unnest(value) %>%
  mutate(temp = 1) %>%
  pivot_wider(names_from = value, values_from = temp, values_fill = list(temp = 0))
train <- cbind(train, one_hot_genres) %>% select(-name)
train$genres <- NULL

#adding the average rating  for each movie minus the total average rating
avg_rating <- mean(train$rating)
movie_score <- train %>% group_by(movieId) %>%
  summarise(movie_score = mean(rating-avg_rating))

#adding the average rating for each user minus the total average rating and movie score
user_score <- train %>% left_join(movie_score, by="movieId") %>%
  mutate(movie_score = ifelse(is.na(movie_score), 0, movie_score)) %>%
  group_by(userId) %>%
  summarise(user_score = mean(rating-avg_rating-movie_score))

train <- train %>% left_join(user_score) %>% left_join(movie_score)
```

```
## Joining, by = "userId"
```

```
## Joining, by = "movieId"
```

```
head(train)
```

```
##   userId movieId rating timestamp                         title release_year
## 1      1     122      5      1996               Boomerang (1992)         1992
## 2      1     292      5      1996                Outbreak (1995)         1995
## 3      1     316      5      1996                Stargate (1994)         1994
## 4      1     329      5      1996 Star Trek: Generations (1994)         1994
## 5      1     355      5      1996       Flintstones, The (1994)         1994
## 6      1     356      5      1996            Forrest Gump (1994)         1994
##   Comedy Romance Action Drama Sci-Fi Thriller Adventure Children Fantasy War
## 1      1       1      0     0      0        0         0        0       0   0
## 2      0       0      1     1      1        1         0        0       0   0
## 3      0       0      1     0      1        0         1        0       0   0
## 4      0       0      1     1      1        0         1        0       0   0
## 5      1       0      0     0      0        0         0        1       1   0
## 6      1       1      0     1      0        0         0        0       0   1
##   Crime Animation Musical Mystery Western Horror Film-Noir Documentary IMAX
## 1     0         0       0       0       0      0         0           0    0
## 2     0         0       0       0       0      0         0           0    0
## 3     0         0       0       0       0      0         0           0    0
## 4     0         0       0       0       0      0         0           0    0
```

```
## 5     0          0       0       0       0      0          0            0    0
## 6     0          0       0       0       0      0          0            0    0
##   (no genres listed) user_score movie_score
## 1                  0   1.713607 -0.64424834
## 2                  0   1.713607 -0.09807398
## 3                  0   1.713607 -0.16571682
## 4                  0   1.713607 -0.18230219
## 5                  0   1.713607 -1.03010228
## 6                  0   1.713607  0.50112574
```

```
##Same wrangling process applied to test set:
class(test$timestamp) ##returns an integer
```

```
## [1] "integer"
```

```
#changing timestamp to datetime, and using only the year
test$timestamp <- year(as_datetime(test$timestamp))

#extract release year from title
pattern <- "(?<=\\()\\d{4}(?=\\))"
test$release_year <- test$title %>% str_extract(pattern) %>% as.integer()

#one-hot encode the genres column
test$genres <- str_split(test$genres, pattern="\\|")
one_hot_genres <- enframe(test$genres) %>%
  unnest(value) %>%
  mutate(temp = 1) %>%
  pivot_wider(names_from = value, values_from = temp, values_fill = list(temp = 0))
test <- cbind(test, one_hot_genres) %>% select(-name)
train$genres <- NULL

#adding columns of genres that are not present in test set, and removing those that are not in the trai
for(col in names(train)){
  if(!col %in% names(test)){
    test$newcol <- 0
    names(test)[names(test)=="newcol"] <- col
  }
}
for(col in names(test)){
  if(!col %in% names(train)){
    test[,col] <- NULL
  }
}

#adding the average scores on the train set of each user and movie
test$user_score <- NULL
test$movie_score <- NULL
test <- test %>% left_join(user_score, by="userId") %>% left_join(movie_score, by="movieId")

#if there are users or movies in the test set that are not in the train set, assigning the score of the
test <- test %>% mutate(user_score = ifelse(is.na(user_score), 0, user_score)) %>% mutate(movie_score =

#reorder the columns to follow the train set
```

```
test <- test %>% select(names(train))
head(test)
```

```
##   userId movieId rating timestamp                                  title
## 1      1     185      5      1996                        Net, The (1995)
## 2      1     364      5      1996                   Lion King, The (1994)
## 3      1     594      5      1996 Snow White and the Seven Dwarfs (1937)
## 4      2     539      3      1997             Sleepless in Seattle (1993)
## 5      2     590      5      1997               Dances with Wolves (1990)
## 6      2     736      3      1997                          Twister (1996)
##   release_year Comedy Romance Action Drama Sci-Fi Thriller Adventure Children
## 1         1995      0       0      1     0      0        1         0        0
## 2         1994      0       0      0     1      0        0         1        1
## 3         1937      0       0      0     1      0        0         0        1
## 4         1993      1       1      0     1      0        0         0        0
## 5         1990      0       0      0     1      0        0         1        0
## 6         1996      0       1      1     0      0        1         1        0
##   Fantasy War Crime Animation Musical Mystery Western Horror Film-Noir
## 1       0   0     1         0       0       0       0      0         0
## 2       0   0     0         1       1       0       0      0         0
## 3       1   0     0         1       1       0       0      0         0
## 4       0   0     0         0       0       0       0      0         0
## 5       0   0     0         0       0       0       1      0         0
## 6       0   0     0         0       0       0       0      0         0
##   Documentary IMAX (no genres listed) user_score movie_score
## 1           0    0                  0  1.7136069 -0.38321653
## 2           0    0                  0  1.7136069  0.24622865
## 3           0    0                  0  1.7136069  0.11031502
## 4           0    0                  0 -0.3097544  0.03528644
## 5           0    0                  0 -0.3097544  0.22990079
## 6           0    0                  0 -0.3097544 -0.28875489
```

4. BUILDING ML MODELS

The evaluation method used is the RMSE (Root Mean Square Error) function defined below. Lower RMSE means the ratings predicted is closer to the actual ratings, which means better results.

Calculating RMSE function:

```
##Calculating RMSE function
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2, na.rm = TRUE))
}
```

Naive RMSE - predict ratings as the mean of training set ratings

```
##Naive RMSE

##calculate mean of the training set
train_rating_1<- mean(train$rating)
train_rating_1
```

```
## [1] 3.512549
```

```r
##predict RMSE for Naive model
naive_rmse <- RMSE( train_rating_1,test$rating)
naive_rmse
```

```
## [1] 1.060492
```

The baseline prediction is to naively predict all ratings as the average rating on the train set. The result RMSE is 1.060492, which means on average the prediction is off by about 1, which is not very good.

Predict RMSE for Linear Model

```r
#Linear Model with timestamp, release_year, user_score, and movie_score
control <- trainControl(method = "none")
fit_linear <- train(rating~user_score+movie_score+timestamp+release_year, data=train, method="lm", trCon
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

```r
print(fit_linear$finalModel)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Coefficients:
##  (Intercept)    user_score   movie_score     timestamp  release_year
##    4.4546646     0.9999583     1.0094569    -0.0002259    -0.0002461
```

```r
y_hat <- predict(fit_linear, test)
linear_model <- RMSE(test$rating, y_hat)
cat("RMSE :", linear_model)
```

```
## RMSE : 0.8666739
```

Linear model gives RMSE of 0.8666739 which is better than baseline RMSE, but can be improved.

Linear Model with regularisation(with user and movie score)

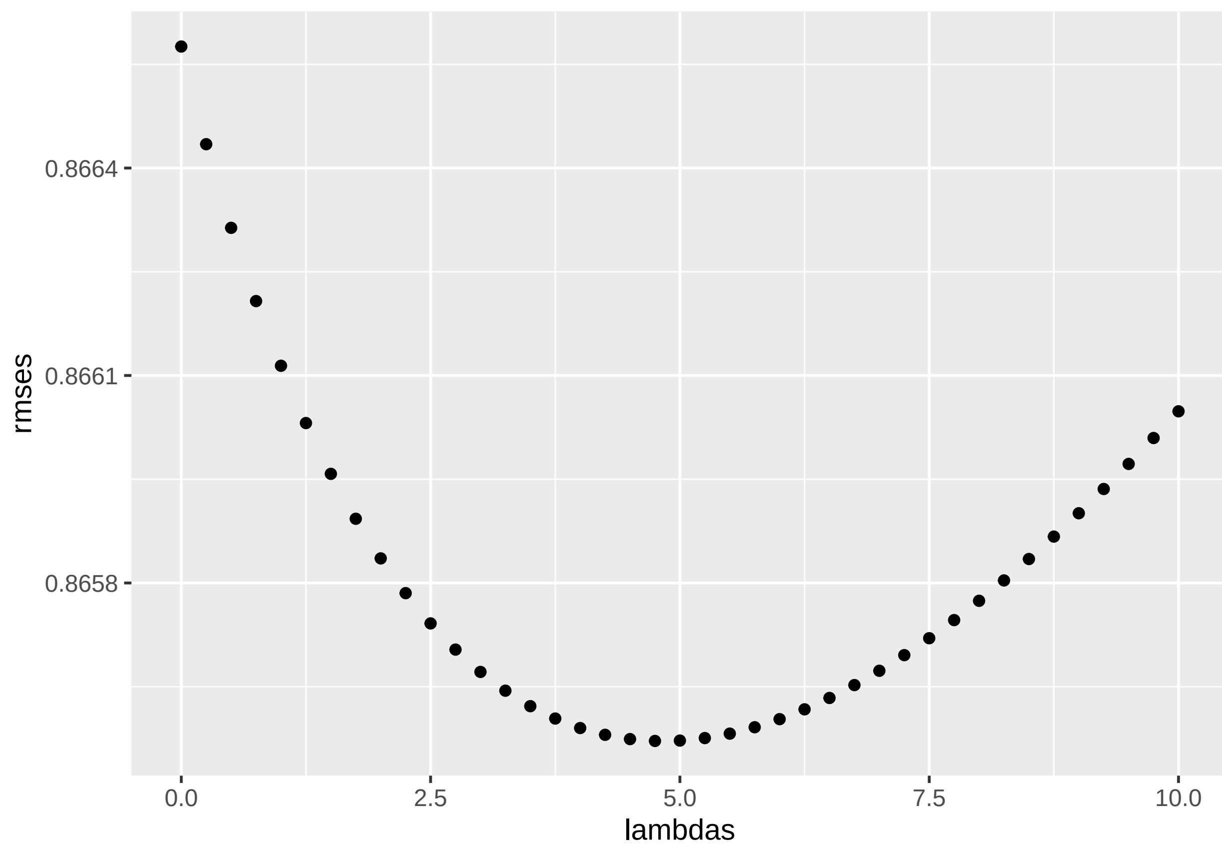
```r
#Linear Model with regularisation, with only user_score and movie_score

#splitting the train set into 2 to calculate the best lambda
idx <- createDataPartition(train$rating, times=1, p=0.8, list=FALSE)
train_part_1 <- train[idx, ]
train_part_2 <- train[-idx, ]

#calculating the best lambda
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  avg_rating <- mean(train_part_1$rating)
  movie_score <- train_part_1 %>%
    group_by(movieId) %>%
```

```
    summarize(b_m = sum(rating - avg_rating)/(n()+l))
  user_score <- train_part_1 %>%
    left_join(movie_score, by="movieId") %>%
    mutate(b_m = ifelse(is.na(b_m), 0, b_m)) %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_m - avg_rating)/(n()+l))
  predicted_ratings <-
    train_part_2 %>%
    left_join(movie_score, by = "movieId") %>%
    left_join(user_score, by = "userId") %>%
    mutate(b_m = ifelse(is.na(b_m), 0, b_m)) %>%
    mutate(b_u = ifelse(is.na(b_u), 0, b_u)) %>%
    mutate(pred = avg_rating + b_m + b_u) %>%
    .$pred
  return(RMSE(predicted_ratings, train_part_2$rating))
})

lambda <- lambdas[which.min(rmses)]
qplot(lambdas, rmses)
```




```
print(lambda)
```

```
## [1] 4.75
```

The lambda which minimises the RMSE is 4.75, so it is used to train the model and predict the test set

Training the final model

```
##Lambda = 4.75. It is used to train the model and predict the test set

#prediction on test set
lambda <- 4.75
avg_rating <- mean(train$rating)
movie_score <- train %>%
  group_by(movieId) %>%
  summarize(b_m = sum(rating - avg_rating)/(n()+lambda))
user_score <- train %>%
  left_join(movie_score, by="movieId") %>%
  mutate(b_m = ifelse(is.na(b_m), 0, b_m)) %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_m - avg_rating)/(n()+lambda))
predicted_ratings <-
  test %>%
  left_join(movie_score, by = "movieId") %>%
  left_join(user_score, by = "userId") %>%
  mutate(b_m = ifelse(is.na(b_m), 0, b_m)) %>%
  mutate(b_u = ifelse(is.na(b_u), 0, b_u)) %>%
  mutate(pred = avg_rating + b_m + b_u) %>%
  .$pred

linear_reg <- RMSE(test$rating, predicted_ratings)
cat("RMSE :", linear_reg)
```

```
## RMSE : 0.865945
```

RMSE in this case is 0.865945, which is better than previous models.

But the genres columns hasnt been used to help with the predictions. First we will see the effect of genre on the ratings.

```
##Using genres column to help with predictions
not_genres <- c("userId", "movieId", "rating", "timestamp", "title", "release_year", "user_score", "movi
genres <- colnames(train)[!colnames(train) %in% not_genres]
genres
```

```
##  [1] "Comedy"             "Romance"            "Action"
##  [4] "Drama"              "Sci-Fi"             "Thriller"
##  [7] "Adventure"          "Children"           "Fantasy"
## [10] "War"                "Crime"              "Animation"
## [13] "Musical"            "Mystery"            "Western"
## [16] "Horror"             "Film-Noir"          "Documentary"
## [19] "IMAX"               "(no genres listed)"
```

```
#calculating the average ratings for each genre
genre_scores <- data.frame(genre="",m=0, sd=0)
for(genre in genres){
  results <- train %>% filter(train[colnames(train)==genre]==1) %>%
    summarise(m=mean(rating), sd=sd(rating))
  genre_scores <- genre_scores %>% add_row(genre=genre, m=results$m, sd=results$sd)
```

```
}
genre_scores <- genre_scores[-1,]
genre_scores[is.na(genre_scores)] <- 0
genre_scores
```

```
##                 genre        m        sd
## 2              Comedy 3.436869 1.0746596
## 3             Romance 3.553544 1.0306635
## 4              Action 3.421419 1.0666085
## 5               Drama 3.673207 0.9952509
## 6              Sci-Fi 3.395830 1.0931448
## 7            Thriller 3.507736 1.0309997
## 8           Adventure 3.493570 1.0530043
## 9            Children 3.418654 1.0926682
## 10            Fantasy 3.502073 1.0657977
## 11                War 3.781151 1.0114896
## 12              Crime 3.665973 1.0124422
## 13          Animation 3.601127 1.0193228
## 14            Musical 3.563654 1.0565897
## 15            Mystery 3.677465 0.9998703
## 16            Western 3.556714 1.0235136
## 17             Horror 3.269039 1.1500854
## 18          Film-Noir 4.012402 0.8887853
## 19        Documentary 3.784756 1.0037628
## 20               IMAX 3.775502 1.0328820
## 21 (no genres listed) 4.000000 0.5773503
```

Linear regularised model with genre feature

```
lambda <- 4.75
avg_rating <- mean(train$rating)
movie_score <- train %>%
  group_by(movieId) %>%
  summarize(b_m = sum(rating - avg_rating)/(n()+lambda))
user_score <- train %>%
  left_join(movie_score, by="movieId") %>%
  mutate(b_m = ifelse(is.na(b_m), 0, b_m)) %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_m - avg_rating)/(n()+lambda))
genre_score <- as.matrix(test[, genres]) %*% genre_scores$m
n_genres <- rowSums(test[,genres])
genre_score <- genre_score / n_genres

#using the genre_scores if the user and movie is unknown

predicted_ratings <-
  test %>%
  left_join(movie_score, by = "movieId") %>%
  left_join(user_score, by = "userId") %>%
  cbind(genre_score) %>%
  mutate(pred = genre_score) %>%
  mutate(pred = ifelse(!is.na(b_m)|!is.na(b_u),
                       avg_rating + replace_na(b_m,0) + replace_na(b_u,0),
```

```
                        pred))

linear_reg_2 <- RMSE(test$rating, predicted_ratings$pred)
cat("RMSE :", linear_reg_2)
```

```
## RMSE : 0.865945
```

We get almost similar result with genres, which shows that genres does not improve predictions by a large margin.

Final RMSE Results:

```
data.frame(
  method=c("Naive Prediction", "Linear Model", "Linear Model with Regularisation(only using movie and us
  rmse=c(naive_rmse, linear_model, linear_reg, linear_reg_2))
```

```
##                                                                 method      rmse
## 1                                                     Naive Prediction 1.0604925
## 2                                                         Linear Model 0.8666739
## 3 Linear Model with Regularisation(only using movie and user scores) 0.8659450
## 4    Linear Model with Regularisation(movie, user, and genre scores) 0.8659450
```

5. APPLYING THE FINAL MODEL TO VALIDATION SET

For the final model, we use the best performing model in the previous section, which is the regularised model.

```
#Train the final model
lambda <- 4.75
avg_rating <- mean(edx$rating)
movie_score <- edx %>%
  group_by(movieId) %>%
  summarize(b_m = sum(rating - avg_rating)/(n()+lambda))
user_score <- edx %>%
  left_join(movie_score, by="movieId") %>%
  mutate(b_m = ifelse(is.na(b_m), 0, b_m)) %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_m - avg_rating)/(n()+lambda))
predicted_ratings <-
  validation %>%
  left_join(movie_score, by = "movieId") %>%
  left_join(user_score, by = "userId") %>%
  mutate(b_m = ifelse(is.na(b_m), 0, b_m)) %>%
  mutate(b_u = ifelse(is.na(b_u), 0, b_u)) %>%
  mutate(pred = avg_rating + b_m + b_u) %>%
  .$pred

final_result <- RMSE(validation$rating, predicted_ratings)
cat("RMSE :", final_result)
```

```
## RMSE : 0.8648201
```

Conclusion:

The final RMSE is 0.8648201, which is achieved by using regularisation model. Improvements can be made by stratifying the user_scores by genres.