MovieLens Capstone

Tareq Aghabi

2021-01-06

Introduction

The following capstone project explores the Movie Lens dataset in an attempt to create a movie recommendation system that best minimizes the RMSE score. Recommendation systems are common machine learning algorithms that are used in many different applications, and in this project, will be used to predict how a user rates a movie based on previous user ratings. The Movie Lens dataset holds millions of entries on a range of different movies and so is an ideal dataset machine learning data application. To be able to compute the code, a the 10 million subset will be used throughout the project.

Method

The first step is downloading and building the Movie Lens data set:

```
#  load data:
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")

# using R 4.0:
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")
```

Afterwards, the 10M dataset is split into a training and validation sets:

```
  set.seed(1)
  test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1,
list = FALSE)
  edx <- movielens[-test_index,]
  temp <- movielens[test_index,]

  validation <- temp %>%
```

```
    semi_join(edx, by = "movieId") %>%
    semi_join(edx, by = "userId")

  removed <- anti_join(temp, validation)

  edx <- rbind(edx, removed)

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

A preview of the training set "edx":

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     231      5 838983392          Dumb & Dumber (1994)
## 4:      1     292      5 838983421               Outbreak (1995)
## 5:      1     316      5 838983392               Stargate (1994)
## 6:      1     329      5 838983392 Star Trek: Generations (1994)
##                           genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:                         Comedy
## 4:  Action|Drama|Sci-Fi|Thriller
## 5:       Action|Adventure|Sci-Fi
## 6: Action|Adventure|Drama|Sci-Fi
```

Characteristics of the training set:

```
## Classes 'data.table' and 'data.frame':   9000061 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 838984885 838983707 ...
##  $ 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" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

Explore

The data set is comprised of 9000055 rows and 6 columns.

```
dim(edx)

## [1] 9000061       6
```

The data set is comprised of 10677 unique movies.

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

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

The data set is comprised of 69878 unique users

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

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

Total number of ratings can be calculated by multiplying 69878 unique users with 10677 unique movies, resulting in 746087406 possible ratings. However, the total possible number of ratings is 9000055. This limitation occurs because not every user rates every movie.

Analysis

To simplify this problem, we will extract the release date to calculate the age of every movie in the dataset. This new dataset will be used to analyze whether movie age affects ratings.

```
# create the new_edx data frame adn convert to timestamp format
library(lubridate)
edx <- mutate(edx, year_rated = year(as_datetime(timestamp)))
release <- stringi::stri_extract(edx$title, regex = "(\\d{4})", comments =
TRUE) %>% as.numeric()
new_edx <- edx %>% mutate(release_date = release) %>% select(-timestamp)
```

Eliminate the incorrect release dates before 1900 in the 10M Movie Lens data set:

```
new_edx[new_edx$movieId == "4311", "release_date"] <- 1998
new_edx[new_edx$movieId == "5472", "release_date"] <- 1972
new_edx[new_edx$movieId == "6290", "release_date"] <- 2003
new_edx[new_edx$movieId == "6645", "release_date"] <- 1971
new_edx[new_edx$movieId == "8198", "release_date"] <- 1960
new_edx[new_edx$movieId == "8905", "release_date"] <- 1992
new_edx[new_edx$movieId == "53953", "release_date"] <- 2007
```

Eliminate the incorrect release dates after 2000 in the 10M Movie Lens data set:

```
new_edx[new_edx$movieId == "27266", "release_date"] <- 2004
new_edx[new_edx$movieId == "671", "release_date"] <- 1996
new_edx[new_edx$movieId == "2308", "release_date"] <- 1973
new_edx[new_edx$movieId == "4159", "release_date"] <- 2001
new_edx[new_edx$movieId == "5310", "release_date"] <- 1985
new_edx[new_edx$movieId == "8864", "release_date"] <- 2004
new_edx[new_edx$movieId == "1422", "release_date"] <- 1997
```

Calculate the true age of the move:

```
new_edx <- new_edx %>% mutate(age_movie = 2020 - release_date, rating_age =
year_rated - release_date)
```

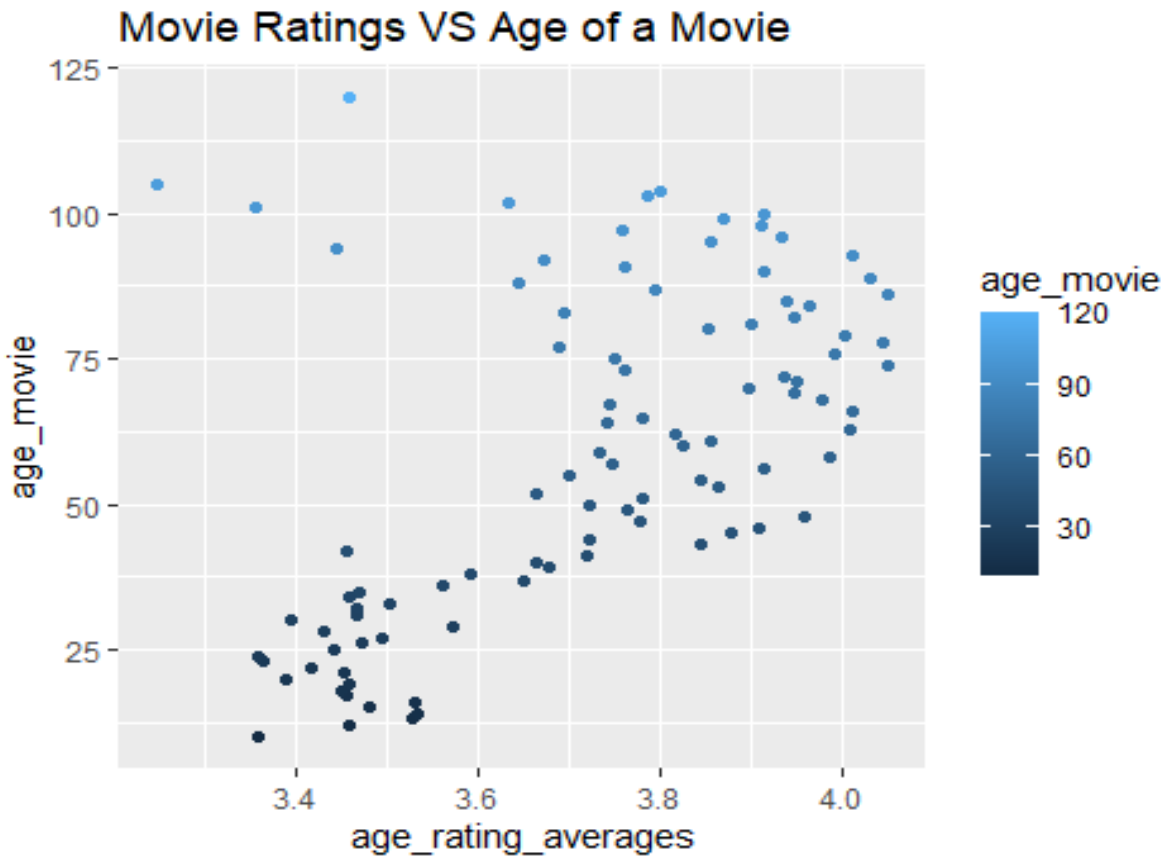
Preview of the updated training set:

```
##    userId movieId rating                         title
## 1:      1     122      5              Boomerang (1992)
## 2:      1     185      5               Net, The (1995)
## 3:      1     231      5          Dumb & Dumber (1994)
## 4:      1     292      5               Outbreak (1995)
## 5:      1     316      5               Stargate (1994)
## 6:      1     329      5 Star Trek: Generations (1994)
##                           genres year_rated release_date age_movie
rating_age
## 1:                Comedy|Romance       1996         1992        28
4
## 2:         Action|Crime|Thriller       1996         1995        25
1
## 3:                         Comedy       1996         1994        26
2
## 4:  Action|Drama|Sci-Fi|Thriller       1996         1995        25
1
## 5:       Action|Adventure|Sci-Fi       1996         1994        26
2
## 6: Action|Adventure|Drama|Sci-Fi       1996         1994        26
2
```

Visulization

To further visualize the data, the relationship between movie rating and movie age averages will be calculated and plotted:

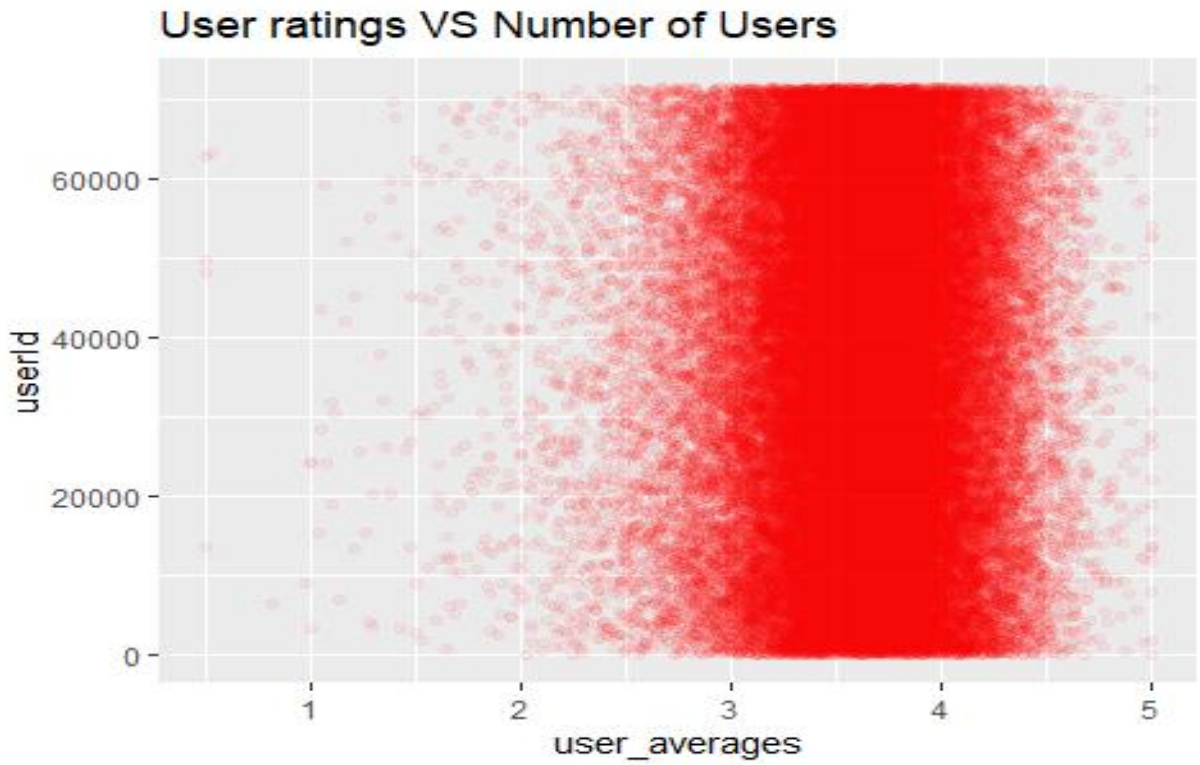
```
movie_avg <- new_edx %>% group_by(movieId) %>%
summarize(movie_rating_averages = mean(rating))

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

age_avg <- new_edx %>% group_by(age_movie) %>% summarize(age_rating_averages
= mean(rating))

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

age_avg %>%
  ggplot(aes(age_rating_averages, age_movie)) +
  geom_point(aes(color=age_movie)) +
  ggtitle("Movie Ratings VS Age of a Movie")
```



As shown in the plot, the average movie rating increases as the age of a movie increases, with a few outliers for movies over a 100 years old.

We will also explore the relationship between the user and the average age of the user:

```
user_avg <- new_edx %>% group_by(userId) %>% summarize(user_averages =
mean(rating))

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

user_avg %>%
  ggplot(aes(user_averages, userId)) +
  geom_point(alpha=0.05, color="red") +
  ggtitle("User ratings VS Number of Users")
```



As shown in the plot, the average user rating across all different users is saturated around a rating between of 3 and 4.

Results

Calculating the RMSE function:

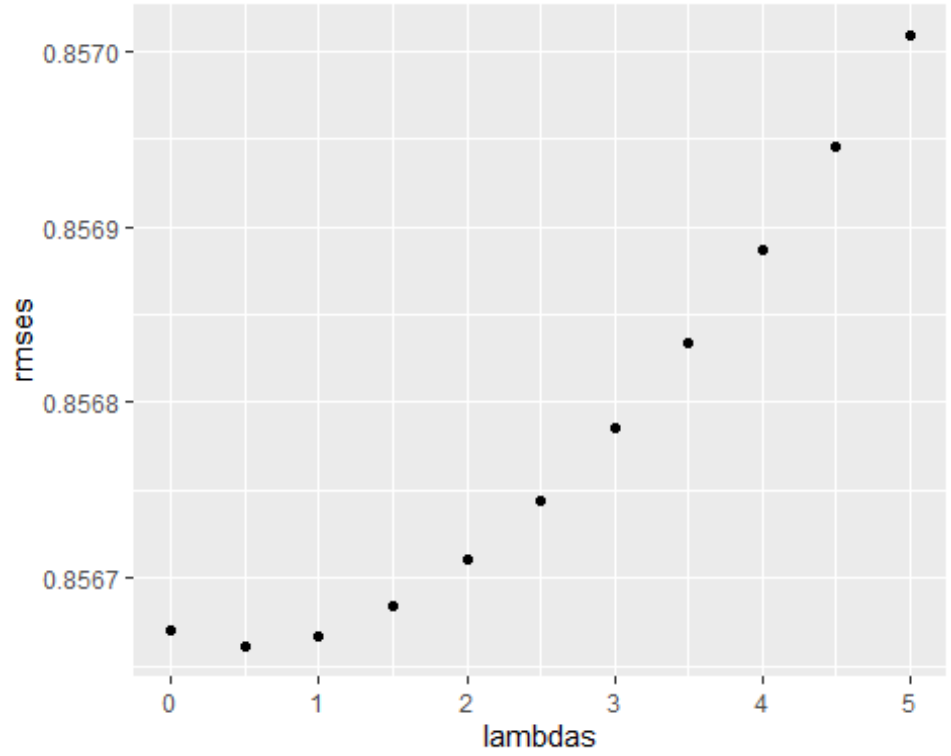
```
rmse_function <- function(true, predicted){
  sqrt(mean((true - predicted)^2))
}
```

Determining lambda:

```
lambdas <- seq(0,5,.5)
rmses <- sapply(lambdas, function(l){
  mu <- mean(new_edx$rating)

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

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

  predicted <- new_edx %>%
```

```
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i +  b_u) %>% .$pred

  return(RMSE(predicted, new_edx$rating))
})
```

Plot Lambda against the rmse values:



As seen in the plot, the lambda that minimizes the RMSE is lambda = 0.5. The test on the validation set is as follows:

```
mu <- mean(validation$rating)
l <- 0.15
b_i <- validation %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n() + l))

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

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

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

predicted <- validation %>%
  left_join(b_i, by = "movieId") %>%
```

```
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i +  b_u) %>% .$pred

rmse_function(predicted, validation$rating)

## [1] 0.8253432
```

Using the Movie ID and User ID as bases for the predictors, and by minimizing lambda, the final RMSE is calculated to be 0.8253432.

Conclusion

In conclusion, this machine learning algorithm successfully minimized the RMSE from a list of possible lambdas. The RMSE was calculated to be 0.8253432 using the Movie ID and User ID. When calculating the RMSE, the accuracy of was measured as the difference between the true value and the predicted value. The biggest limitation to this project was the computing power of my laptop. As such, data manipulation on the Movie Lens data set was restricted.