MovieLens Capstone

Edy Susanto

2022-02-15

Background

This capstone project is an exploration of a MovieLens dataset that tries to help find movies for users based on ratings that others have left for movies. We experiment with creating a recommendation system, that will minimize the RMSE score. and then extracting new features from the data to try and get better predictions.

Preparation

The First Step is downloading and building the Movie Lens data set:

```
## -- Attaching packages --------------------------------------- tidyve
rse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.8
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_co
nflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

## Loading required package: lattice

##
## Attaching package: 'caret'

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

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

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

```
##
## 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

#  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-10M100
K/ratings.dat"))), col.names = c("userId", "movieId", "rating", "timest
amp"))

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(movieI
d), title = as.character(title), genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")
```

The Second Step, split the dataset 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)

## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
 "genres")

  edx <- rbind(edx, removed)

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

The Third Step , 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
```

The Four Step , preview characteristics of the training set:

```
## Classes 'data.table' and 'data.frame':   9000061 obs. of  6 variable
s:
##  $ 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 & Dumb
er (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy"
 "Action|Drama|Sci-Fi|Thriller" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

Exploration Dataset

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 calculation 69878 * 10677 = 746087406.. Not every user rates every movie.

Analysis

We will try to 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})", comment
s = 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:

```
## `summarise()` has grouped output by 'movieId', 'title'. You can over
ride using
## the `.groups` argument.

## # A tibble: 8 x 4
## # Groups:   movieId, title [8]
##   movieId title                                                releas
e_date     n
##     <dbl> <chr>
 <dbl> <int>
## 1    1422 Murder at 1600 (1997)
  1600  1552
## 2    4311 Bloody Angels (1732 HÃ¸tten: Marerittet Har et Pos~
  1732     8
## 3    5472 1776 (1972)
  1776   184
## 4    6290 House of 1000 Corpses (2003)
  1000   371
## 5    6645 THX 1138 (1971)
  1138   467
## 6    8198 1000 Eyes of Dr. Mabuse, The (Tausend Augen des Dr~
  1000    26
## 7    8905 1492: Conquest of Paradise (1992)
  1492   141
## 8   53953 1408 (2007)
  1408   465
```

```
# view and correct the incorrect release dates outside of the ranges
new_edx %>% filter(release_date < 1900) %>% group_by(movieId, title, re
lease_date) %>% summarize(n = n())

## `summarise()` has grouped output by 'movieId', 'title'. You can over
ride using
## the `.groups` argument.
## # A tibble: 8 x 4
## # Groups:   movieId, title [8]
##   movieId title                                               releas
e_date     n
##     <dbl> <chr>
 <dbl> <int>
## 1    1422 Murder at 1600 (1997)
  1600  1552
## 2    4311 Bloody Angels (1732 HÃ¸tten: Marerittet Har et Pos~
  1732     8
## 3    5472 1776 (1972)
  1776   184
## 4    6290 House of 1000 Corpses (2003)
  1000   371
## 5    6645 THX 1138 (1971)
  1138   467
## 6    8198 1000 Eyes of Dr. Mabuse, The (Tausend Augen des Dr~
  1000    26
## 7    8905 1492: Conquest of Paradise (1992)
  1492   141
## 8   53953 1408 (2007)
  1408   465
```

```r
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:

```
## `summarise()` has grouped output by 'movieId', 'title'. You can over
ride using
## the `.groups` argument.
```

```
## # A tibble: 6 x 4
## # Groups:   movieId, title [6]
##   movieId title                                          release_dat
e     n
##     <dbl> <chr>                                                 <db
l> <int>
## 1     671 Mystery Science Theater 3000: The Movie (1996)         300
0  3266
## 2    2308 Detroit 9000 (1973)                                    900
0    22
## 3    4159 3000 Miles to Graceland (2001)                         300
0   714
## 4    5310 Transylvania 6-5000 (1985)                             500
0   197
```

```
## 5    8864 Mr. 3000 (2004)                                           300
0   155
## 6   27266 2046 (2004)                                               204
6   422
```

```
# view and correct the incorrect release dates outside of the ranges
new_edx %>% filter(release_date > 2020) %>% group_by(movieId, title, re
lease_date) %>% summarize(n = n())
```

```
## `summarise()` has grouped output by 'movieId', 'title'. You can over
ride using
## the `.groups` argument.
```

```
## # A tibble: 6 x 4
## # Groups:   movieId, title [6]
##   movieId title                                              release_dat
e      n
##     <dbl> <chr>                                                      <db
l> <int>
## 1     671 Mystery Science Theater 3000: The Movie (1996)         300
0  3266
## 2    2308 Detroit 9000 (1973)                                      900
0    22
## 3    4159 3000 Miles to Graceland (2001)                           300
0   714
## 4    5310 Transylvania 6-5000 (1985)                               500
0   197
## 5    8864 Mr. 3000 (2004)                                           300
0   155
## 6   27266 2046 (2004)                                               204
6   422
```

```
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_a
ge = year_rated - release_date)
```

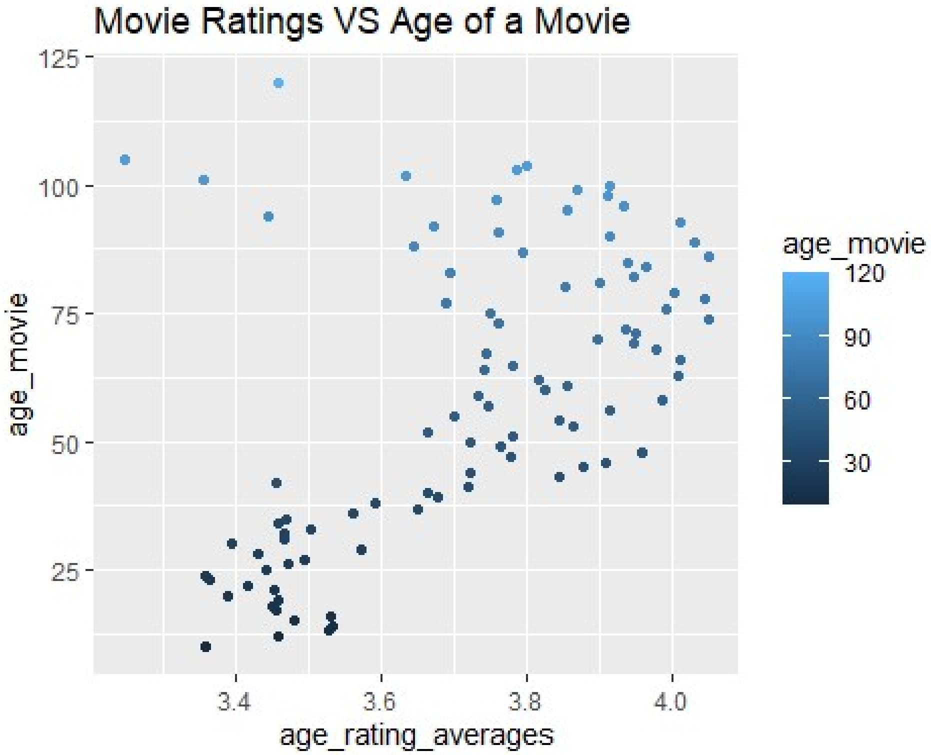
Preview of the updated training set:

```
##    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 r
ating_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
```

Visualization

Plot relationship between movie rating and movie age averages:

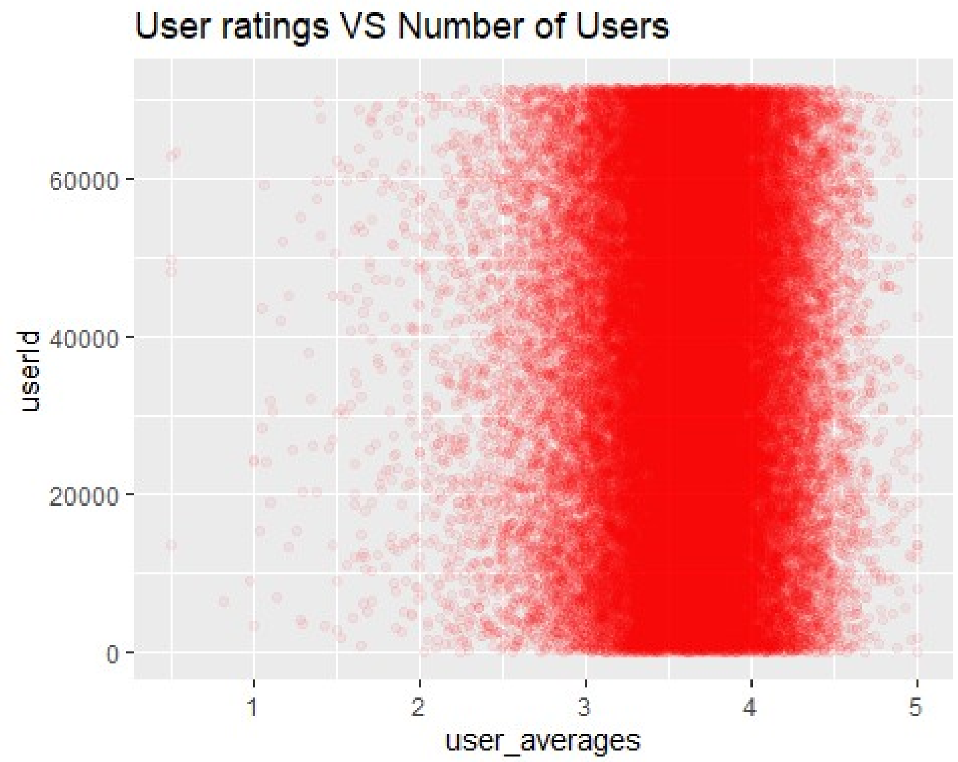
```
movie_avg <- new_edx %>% group_by(movieId) %>% summarize(movie_rating_a
verages = mean(rating))
age_avg <- new_edx %>% group_by(age_movie) %>% summarize(age_rating_ave
rages = mean(rating))

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



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))

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.

Outcomes

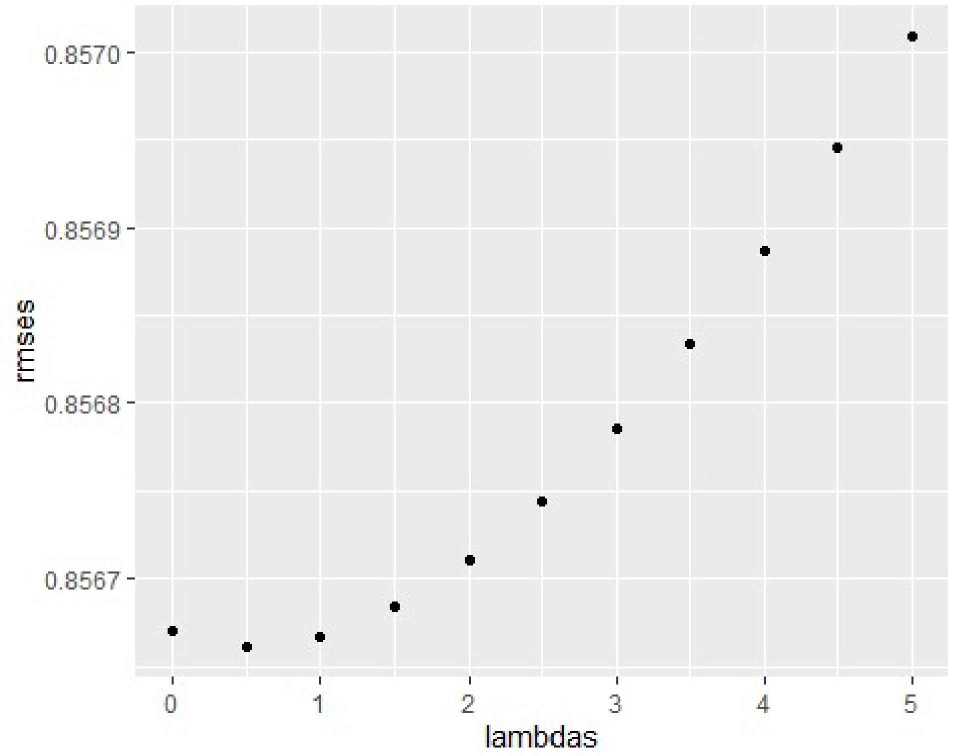
RMSE function:

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

Lambda Function:

```
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 VS 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))

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

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
```

Final RMSE is calculated to be 0.8253432.

Finding

Finding in this project that 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.