MovieLens

Sariq Sahazada

24/12/2020

INTRODUCTION:

This project is a part of HarvardX PH125.9x Data Science: Capstone course. For this project, I will be creating a movie recommendation system using the MovieLens dataset to demonstrate all the skills acquired throughout the courses in the course series. The task is to train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set. For this project the MovieLens Data set is collected by GroupLens Research and can be found in GroupLens web link (https://grouplens.org/datasets/movielens/latest/).

The data set is loaded using the code provided by course instructor in this link. (https://bit.ly/34ZU4PI) which split the data into edx set and 10% validation set. Validation set will be used to final evaluation. Below are is the Code:

```
##########################################################
# Create edx set, validation set (final hold-out test set)
##########################################################

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

```
## Loading required package: tidyverse

## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.4     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0

## -- 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
```

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

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

```r
library(tidyverse)
library(caret)
library(data.table)

# 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")

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") # if using R 3.5 or earlier, use `set.seed(1)`
```

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

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

```r
# 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")
```

```r
edx <- rbind(edx, removed)

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

Before we jump into the analysis of the data we will install neccasary packages. Below is the code for that:

```r
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
```

EXPLORING THE DATA:

To understand the data set properly, First we will examine the structure of the data set and see the summary statistics as well. Below is the code chunk for that:

```r
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

We can now confirm that the data set provided is in tidy format with 6 variables.To identify the number of unique users and unique movies we will use the below code:

```
dim(edx) # 9000055       6
```

```
## [1] 9000055       6
```

```
n_distinct(edx$movieId) # 10677
```

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

```
n_distinct(edx$title) # 10676: there might be movies of different IDs with the same title
```

```
## [1] 10676
```

```
n_distinct(edx$userId) # 69878
```

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

```
n_distinct(edx$movieId)*n_distinct(edx$userId) # 746087406
```

```
## [1] 746087406
```

```
n_distinct(edx$movieId)*n_distinct(edx$userId)/dim(edx)[1] # 83
```

```
## [1] 82.89809
```

Which shows the number of unique users as 69878 and movies as 10677. If we multiply both of them we will get 746087406 which is very much larger than the number of observations we have in the data set i.e 9000055. Which implies that not all the users are giving the ratings and also not all the movies are being rated as the same number of times.

Extracting age of Movies:

Every movie was released in a certain year, which is provided in the title of the movie. Every user rated a movie in a certain year, which is included in the timestamp information. we will define the difference between these two years, i.e., how old the movie was when it was watched/rated by a user, as the age of movies at rating. From the original dataset, We first extract the rating year (year_rated) from timestamp, and then extract the release year (year_released) of the movie from the title. age_at_rating was later calculated.

```
# convert timestamp to year
edx1 <- edx %>% mutate(year_rated = year(as_datetime(timestamp)))
# extract the release year of the movie
# edx1 has year_rated, year_released, age_at_rating, and titles without year information
edx1 <- edx1 %>% mutate(title = str_replace(title,"^(.+)\\s\\((\\d{4})\\)$","\\1__\\2" )) %>%
  separate(title,c("title","year_released"),"__") %>%
  select(-timestamp)
edx1 <- edx1 %>% mutate(age_at_rating= as.numeric(year_rated)-as.numeric(year_released))
head(edx1)
```

```
##    userId movieId rating                  title year_released
## 1:      1     122      5              Boomerang          1992
## 2:      1     185      5               Net, The          1995
## 3:      1     292      5               Outbreak          1995
## 4:      1     316      5               Stargate          1994
## 5:      1     329      5 Star Trek: Generations          1994
## 6:      1     355      5       Flintstones, The          1994
##                           genres year_rated age_at_rating
## 1:                Comedy|Romance       1996             4
## 2:         Action|Crime|Thriller       1996             1
## 3:  Action|Drama|Sci-Fi|Thriller       1996             1
## 4:       Action|Adventure|Sci-Fi       1996             2
## 5: Action|Adventure|Drama|Sci-Fi       1996             2
## 6:       Children|Comedy|Fantasy       1996             2
```

Extracting the Genres:

The genres information was provided in the original dataset as a combination of different classifications. For example, the movie "Boomerang" (movieId 122) was assigned "Comedy|Romance", and "Flintstones, The" (movieId 355) is "Children|Comedy|Fantasy". Both are combinations of different ones, while they actually share one genre (Comedy). It'll make more sense if we first split these combinations into single ones:

```r
memory.limit(size=56000) # To extend the RAM memory allocation size
```

```
## [1] 56000
```

```r
# edx2: the mixture of genres is split into different rows
edx2 <- edx1 %>% separate_rows(genres,sep = "\\|") %>% mutate(value=1)
n_distinct(edx2$genres)  # 20: there are 20 differnt types of genres
```

```
## [1] 20
```

```r
genres_rating <- edx2 %>% group_by(genres) %>% summarize(n=n())
```

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

```r
genres_rating
```

```
## # A tibble: 20 x 2
##    genres                   n
##    <chr>                <int>
##  1 (no genres listed)       7
##  2 Action             2560545
##  3 Adventure          1908892
##  4 Animation           467168
##  5 Children            737994
##  6 Comedy             3540930
##  7 Crime              1327715
##  8 Documentary          93066
##  9 Drama              3910127
```

```
## 10 Fantasy            925637
## 11 Film-Noir          118541
## 12 Horror             691485
## 13 IMAX                 8181
## 14 Musical            433080
## 15 Mystery            568332
## 16 Romance           1712100
## 17 Sci-Fi            1341183
## 18 Thriller          2325899
## 19 War                511147
## 20 Western            189394
```

Splitting the genres information into multiple row can facilitate the exploration of genres. However, one thing to keep in mind is, if we consider one row as one record, the above transformation of the dataset actually duplicated each record into multiple ones, depending on the combination of the genres for each movie.

To avoid this problem and make more sense if we want to utilize the genres information in building the prediction model, genres of each movie should be split into multiple columns to indicate different combinations of the 19 basic genres. We can achieve this goal by spreading genres to the "wide" format:

```
# edx3 is the final version for exploration of the effects of movie year, age, rating year, and genres
edx3 <- edx2 %>% spread(genres, value, fill=0) %>% select(-"(no genres listed)")
dim(edx3) #  9000055      26
```

```
## [1] 9000055      26
```

By doing this, we can actually visualize the genres of each movie. For example, each row represents a movie, and each column respresents a genre. Each movie can have a unique combination of different genres.

MODELING STRATEGY:

The age of movie at rating seems to affect the rating, while genres does not add much information. Also, the effect of genres could also be included in the movie effect itself. Therefore, we will consider the effects of movie age at rating, movie effect, and user effect and build our model based on that. Regularization of movie effect and user effect will also be used to build a more robust model.I will evaluate these models and choose the best to go with. Residuals will be calculated and used as the input of matrix factorization technique.

RESULTS:

Define RMSE: Residual Mean Squared Error

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

First model: Use Average Ratings for all Movies regardless of User

In the Average ratings model, just based on the ratings itself, to minimize the RMSE, the best prediction of ratings for each movie will be the overall average of all ratings. The average rating is mu = 3.51247, and the naive RMSE is 1.0612.

```r
mu <- mean(edx$rating)
naive_rmse <- RMSE(validation$rating, mu)
rmse_results <- data_frame(Model = "Just the average", RMSE = naive_rmse)
```

```
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```r
rmse_results
```

```
## # A tibble: 1 x 2
##   Model             RMSE
##   <chr>            <dbl>
## 1 Just the average  1.06
```

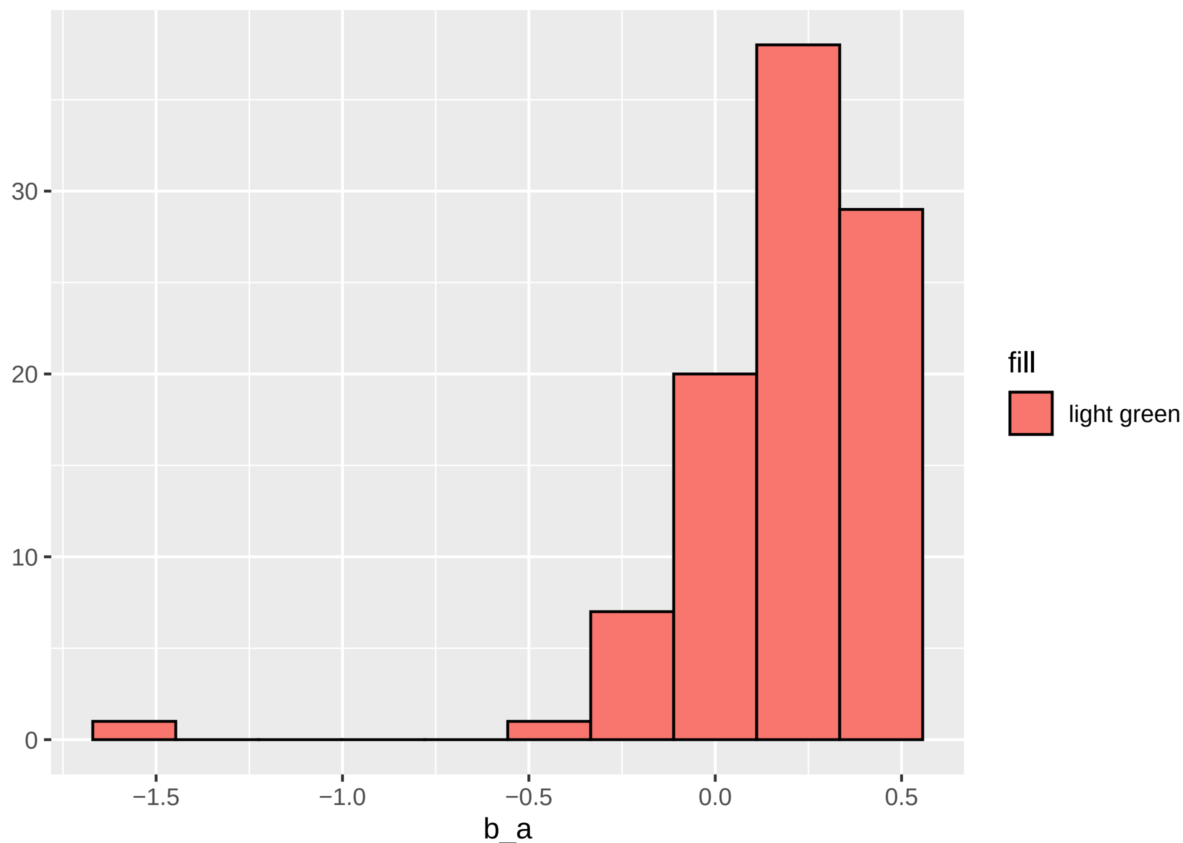
Modeling Age Effects: adding b_a to represent ratings on movies with certain age

Because earlier we saw that the age of movies at the time of rating seems to affect ratings, We try to see if add a bias of age (b_a) to the model could better predict the ratings. First let's calculate the age bias and take a look at its distribution. Then we will make predictions and evaluate the RMSE using the validation set.

```r
age_effect<- edx1 %>%
  group_by(age_at_rating) %>%
  summarize(b_a = mean(rating)-mu)
```

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

```r
age_effect %>% qplot(b_a, geom ="histogram", bins = 10, data = ., color = I("black"), fill = "light gre
```




```
validation1 <- validation %>%
  mutate(year_rated = year(as_datetime(timestamp)))%>%
  mutate(title = str_replace(title,"^(.+)\\s\\((\\d{4})\\)$","\\1__\\2" )) %>%
  separate(title,c("title","year_released"),"__") %>%
  select(-timestamp) %>%
  mutate(age_at_rating= as.numeric(year_rated)-as.numeric(year_released))

predicted_ratings_2 <- mu + validation1 %>%
  left_join(age_effect, by='age_at_rating') %>%
  pull(b_a)
model_2_rmse <- RMSE(validation$rating,predicted_ratings_2) # 1.05239
rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="Age Effect Model",
                                     RMSE = model_2_rmse))
rmse_results
```

```
## # A tibble: 2 x 2
##   Model              RMSE
##   <chr>             <dbl>
## 1 Just the average   1.06
## 2 Age Effect Model   1.05
```

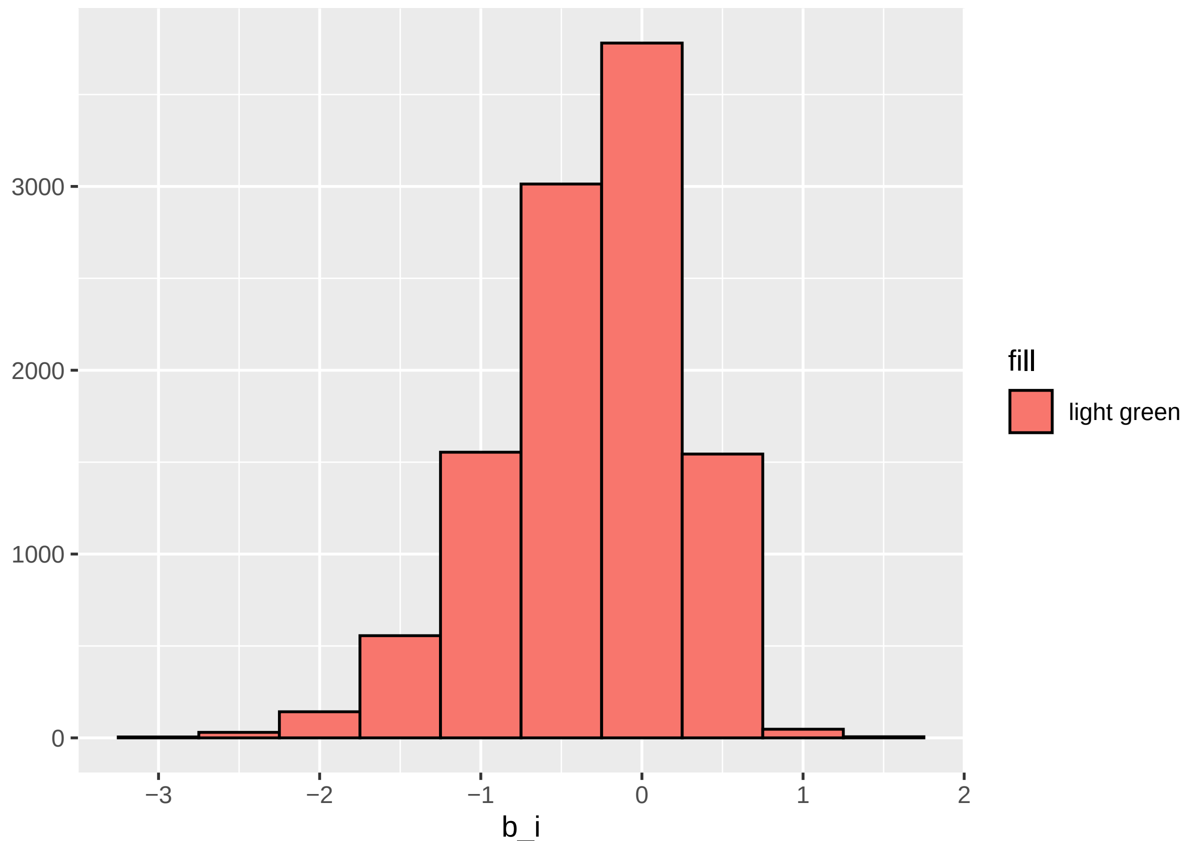
Modeling Movie Effects: Adding b_i to represent Average Ranking for movie_i

Since the intrinsic features of a movie could obviously affect the ratings of a movie, we add the bias of movie/item (b_i) to the model, i.e., for each movie, the average of the ratings on that specific movie will have a difference from the overall average rating of all movies. We can plot the distribution of the bias and calculate the RMSE of this model.

```
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

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

```
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"), fill = "light gre
```




```
predicted_ratings_3 <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
model_3_rmse <- RMSE(validation$rating,predicted_ratings_3)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="Movie Effect Model",
                                     RMSE = model_3_rmse))
rmse_results
```

```
## # A tibble: 3 x 2
##   Model               RMSE
##   <chr>              <dbl>
## 1 Just the average   1.06
## 2 Age Effect Model   1.05
## 3 Movie Effect Model 0.944
```

User Effects: Adding b_u to represent Average Ranking for user_u

Similar to the movie effect, intrinsic features of a given user could also affect the ratings of a movie. For example, a stricter user could give lower scores for all movies he/she watched than rated by other users. We now further add the bias of user (b_u) to the movie effect model.

```
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

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

```
predicted_ratings_4 <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
model_4_rmse <- RMSE(validation$rating,predicted_ratings_4)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="Movie + User Effects Model",
                                     RMSE = model_4_rmse))
rmse_results
```

```
## # A tibble: 4 x 2
##   Model                       RMSE
##   <chr>                      <dbl>
## 1 Just the average           1.06
## 2 Age Effect Model           1.05
## 3 Movie Effect Model         0.944
## 4 Movie + User Effects Model 0.865
```

CONCLUSION:

From the summarized RMSEs of different models, we can see that Movie + User Effect Model largely improved the accuracy of the prediction.

```
## # A tibble: 4 x 2
##   Model                       RMSE
##   <chr>                      <dbl>
## 1 Just the average           1.06
## 2 Age Effect Model           1.05
## 3 Movie Effect Model         0.944
## 4 Movie + User Effects Model 0.865
```

MovieLens is a classical dataset for recommendation system and represents a challenge for development of better machine learning algorithm. In this project, the “Just the average” model only gives a RMSE of 1.0612, and the best model (Movie + User Effect Model) could largely improved it to 0.0.8653.. In conclusion, Movie + User Effect Model appears to be a very powerful technique for recommendation system, which usually contains large and sparse dataset making it hard to make prediction using other machine learning strategies. The effects of age and genres could be further explored to improve the performance of the model. The Ensemble method should also be considered in the future to apply on the MovieLens dataset, in order to combine the advantages of various models and enhance the overall performance of prediction.