DATA612-Project1-Baseline-RMSE

calculate statistics on a user-item ratings matrix

Michael Y.

2/17/2020

Contents

```r
library(tidyverse)
```

```
## -- Attaching packages ------------------------------------------------------------------------------

## <U+2713> ggplot2 3.2.1     <U+2713> purrr   0.3.3
## <U+2713> tibble  2.1.3     <U+2713> dplyr   0.8.3
## <U+2713> tidyr   1.0.0     <U+2713> stringr 1.4.0
## <U+2713> readr   1.3.1     <U+2713> forcats 0.4.0

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

```r
library(kableExtra)
```

```
##
## Attaching package: 'kableExtra'

## The following object is masked from 'package:dplyr':
##
##     group_rows
```

```r
library(RMySQL)
```

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

Project 1 - Global Baseline Predictors and RMSE

- **Briefly describe the recommender system that you're going to build out from a business perspective, e.g. "This system recommends data science books to readers."**

This system recommends recent movies to viewers.

- **Find a dataset, or build out your own toy dataset. As a minimum requirement for complexity, please include numeric ratings for at least five users, across at least five items, with some missing data.**

For our dataset, we will use the same ratings shown in the youtube playlist of videos that we were asked to watch. This will enable easy verification of the correctness of our figures at each step.

While the videos assumed that there were 6 users and 5 movies, here we will treat the data as 6 movies rated by 5 users.

I've loaded up the list of users, the list of movies, and their ratings into a MYSQL database.

The next step extracts the data into a "long" dataframe.

Connect to the MySql database and retrieve the data:

```r
# I created "stduser" as a read-only account in my database which only has "select" privilege
connstd <- dbConnect(MySQL(), user="stduser", password="password",
                     dbname="Proj1_Movies", host="localhost")

# Create a query which joins the 3 database tables,
# replacing the auto-generated ID codes with the movie names and the reviewers' names

query <- 'Select M.Movie_title, F.Friend_name, R.Rating
          From Movies as M, Friends as F, Ratings as R
          Where (M.Movie_id = R.Movie_ID AND F.Friend_id = R.Friend_ID);'

# Execute the query
results <- dbGetQuery(connstd, query)

# Close the database connection
discard <- dbDisconnect(connstd) # this function returns "TRUE", so assignment suppresses printing
```

- **Load your data into (for example) an R or pandas dataframe, a Python dictionary or list of lists, (or another data structure of your choosing).**

```
results %>% kable() %>%  kable_styling(c("striped", "bordered"))
```

Movie_title	Friend_name	Rating
Crazy Rich Asians	Alice	5
Crazy Rich Asians	Bob	NA
Crazy Rich Asians	Carol	4
Crazy Rich Asians	Dave	NA
Crazy Rich Asians	Eddie	4
Disney's Christopher Robin	Alice	4
Disney's Christopher Robin	Bob	3
Disney's Christopher Robin	Carol	5
Disney's Christopher Robin	Dave	3
Disney's Christopher Robin	Eddie	4
Mamma Mia! Here We Go Again	Alice	4
Mamma Mia! Here We Go Again	Bob	2
Mamma Mia! Here We Go Again	Carol	NA
Mamma Mia! Here We Go Again	Dave	NA
Mamma Mia! Here We Go Again	Eddie	3
Ocean's 8	Alice	2
Ocean's 8	Bob	2
Ocean's 8	Carol	3
Ocean's 8	Dave	1
Ocean's 8	Eddie	2
Peter Rabbit	Alice	4
Peter Rabbit	Bob	NA
Peter Rabbit	Carol	5
Peter Rabbit	Dave	4
Peter Rabbit	Eddie	5
Solo: A Star Wars Story	Alice	4
Solo: A Star Wars Story	Bob	2
Solo: A Star Wars Story	Carol	5
Solo: A Star Wars Story	Dave	4
Solo: A Star Wars Story	Eddie	4

The raw data:

The dimensions of the results dataframe are (30, 3) .

```r
### Structure
str(results)
```

Structure and Summary of the results dataframe:

```
## 'data.frame':    30 obs. of  3 variables:
##  $ Movie_title: chr  "Crazy Rich Asians" "Crazy Rich Asians" "Crazy Rich Asians" "Crazy Rich Asians"
##  $ Friend_name: chr  "Alice" "Bob" "Carol" "Dave" ...
##  $ Rating     : int  5 NA 4 NA 4 4 3 5 3 4 ...
```

```r
### Summary
summary(results)
```

```
##  Movie_title        Friend_name            Rating
##  Length:30          Length:30          Min.   :1.00
##  Class :character   Class :character   1st Qu.:3.00
##  Mode  :character   Mode  :character   Median :4.00
##                                        Mean   :3.52
##                                        3rd Qu.:4.00
##                                        Max.   :5.00
##                                        NA's   :5
```

- **From there, create a user-item matrix.**

We use `pivot_wider` to convert the above format into a table with 6 rows and 5 columns.

```
#### use pivot_wider to make a User-Item (UI) matrix
results %>% pivot_wider(names_from = Friend_name, values_from = Rating) -> UI
UI
```

```
## # A tibble: 6 x 6
##   Movie_title                 Alice   Bob Carol  Dave Eddie
##   <chr>                       <int> <int> <int> <int> <int>
## 1 Crazy Rich Asians               5    NA     4    NA     4
## 2 Disney's Christopher Robin      4     3     5     3     4
## 3 Mamma Mia! Here We Go Again     4     2    NA    NA     3
## 4 Ocean's 8                       2     2     3     1     2
## 5 Peter Rabbit                    4    NA     5     4     5
## 6 Solo: A Star Wars Story         4     2     5     4     4
```

```
#### movie_title is still a column -- make it rownames instead
UI  %>% column_to_rownames("Movie_title") -> UI
UI
```

```
##                             Alice Bob Carol Dave Eddie
## Crazy Rich Asians               5  NA     4   NA     4
## Disney's Christopher Robin      4   3     5    3     4
## Mamma Mia! Here We Go Again     4   2    NA   NA     3
## Ocean's 8                       2   2     3    1     2
## Peter Rabbit                    4  NA     5    4     5
## Solo: A Star Wars Story         4   2     5    4     4
```

```
### corresponds to the ratings in video J
```

- **If you choose to work with a large dataset, you're encouraged to also create a small, relatively dense "user-item" matrix as a subset so that you can hand-verify your calculations.**

```r
### check the rowMeans (note- this is the entire dataset - we will exclude test next)
rowMeans(UI,na.rm = T) %>% t %>% t
```

```
##                                 [,1]
## Crazy Rich Asians           4.333333
## Disney's Christopher Robin  3.800000
## Mamma Mia! Here We Go Again 3.000000
## Ocean's 8                   2.000000
## Peter Rabbit                4.500000
## Solo: A Star Wars Story     3.800000
```

```r
### check the colMeans (note- this is the entire dataset - we will exclude test next)
colMeans(UI,na.rm=T)
```

```
##    Alice      Bob    Carol     Dave    Eddie
## 3.833333 2.250000 4.400000 3.000000 3.666667
```

- **Break your ratings into separate training and test datasets.**

Because in this example we are trying to match the values displayed in the video playlist, we have to select the same cases for the training and test sets as the selection made in the online videos.

Accordingly, we will make a matrix of ones and zeros which will facilitate extracting the desired elements from the overall matrix.
The original dataset contained 30 possible values, but 5 of those were NA.

The train/test split is 80/20, with 20 elements remaining in the training dataset and 5 elements removed to the test dataset.

Going forward, we would use a random number generator to make random selections of these elements.

Test dataset:

```
test_extractor = matrix(
  data = c(
  0,0,0,0,0,
  0,0,0,1,0,
  1,0,0,0,0,
  0,1,0,0,0,
  0,0,1,0,0,
  0,0,0,0,1
  ),
  nrow=6,
  ncol=5,
  byrow = T)

test_extractor
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0    0
## [2,]    0    0    0    1    0
## [3,]    1    0    0    0    0
## [4,]    0    1    0    0    0
## [5,]    0    0    1    0    0
## [6,]    0    0    0    0    1
```

```
## multiply the test_extractor by the User-Item-Matrix (element by element -- not matrix multplication)
UI_test <- test_extractor * UI
#UI_test
## set the zeroes to "NA" for the test matrix
UI_test[UI_test==0]<-NA

### convert from dataframe to matrix
UI_test <- as.matrix(UI_test)

### display UI_test
UI_test %>% kable(caption = "*USER-ITEM TEST MATRIX*") %>%   kable_styling(c("bordered","striped"))
```

Table 1: *USER-ITEM TEST MATRIX*

	Alice	Bob	Carol	Dave	Eddie
Crazy Rich Asians	NA	NA	NA	NA	NA
Disney's Christopher Robin	NA	NA	NA	3	NA
Mamma Mia! Here We Go Again	4	NA	NA	NA	NA
Ocean's 8	NA	2	NA	NA	NA
Peter Rabbit	NA	NA	5	NA	NA
Solo: A Star Wars Story	NA	NA	NA	NA	4

```
### corresponds to the "blue" values in video J
```

Table 2: *USER-ITEM TRAINING MATRIX*

	Alice	Bob	Carol	Dave	Eddie
Crazy Rich Asians	5	NA	4	NA	4
Disney's Christopher Robin	4	3	5	NA	4
Mamma Mia! Here We Go Again	NA	2	NA	NA	3
Ocean's 8	2	NA	3	1	2
Peter Rabbit	4	NA	NA	4	5
Solo: A Star Wars Story	4	2	5	4	NA

Training dataset:

```
### Define the Training extractor as 1 minus the test extractor
train_extractor = 1 - test_extractor
train_extractor
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    1    1    1    1
## [2,]    1    1    1    0    1
## [3,]    0    1    1    1    1
## [4,]    1    0    1    1    1
## [5,]    1    1    0    1    1
## [6,]    1    1    1    1    0
```

```
## multiply the train_extractor by the User-Item-Matrix (element by element -- not matrix multplication
UI_train <- train_extractor * UI
## set the zeroes to "NA" for the train matrix
UI_train[UI_train==0]<-NA

### convert from dataframe to matrix
UI_train <- as.matrix(UI_train)

### UI_train
UI_train %>% kable(caption = "*USER-ITEM TRAINING MATRIX*") %>%   kable_styling(c("bordered","striped")
```

```
### corresponds to the "black" values in video K
```

Table 3: *MEAN-RATING MATRIX*

	Alice	Bob	Carol	Dave	Eddie
Crazy Rich Asians	3.5	3.5	3.5	3.5	3.5
Disney's Christopher Robin	3.5	3.5	3.5	3.5	3.5
Mamma Mia! Here We Go Again	3.5	3.5	3.5	3.5	3.5
Ocean's 8	3.5	3.5	3.5	3.5	3.5
Peter Rabbit	3.5	3.5	3.5	3.5	3.5
Solo: A Star Wars Story	3.5	3.5	3.5	3.5	3.5

- **Using your training data, calculate the raw average (mean) rating for every user-item combination.**

```
mean_value <- mean(UI_train, na.rm = T)
mean_value
```

```
## [1] 3.5
```

```
## value 3.5 corresponds to the result in video K

### make a matrix with same rownames and colnames as UI_train, but replace the values
mean_rating <- UI_train
mean_rating[T] <- mean_value
mean_rating  %>% kable(caption = "*MEAN-RATING MATRIX*") %>%   kable_styling(c("bordered","striped"))
```

- **Calculate the RMSE for raw average for both your training data and your test data.**

```
### Training RMSE
train_RMSE_raw <- sqrt( mean ( (UI_train - mean_rating)^2, na.rm=T ) )
train_RMSE_raw
```

```
## [1] 1.161895
```

```
## 1.1619 cooresponds to the result in video L

### Test RMSE
test_RMSE_raw <- sqrt( mean ( (UI_test - mean_rating)^2, na.rm=T ) )
test_RMSE_raw
```

```
## [1] 1.024695
```

```
## 1.0247 cooresponds to the result in video L
```

Table 4: USER BIAS

Alice	0.300000
Bob	-1.166667
Carol	0.750000
Dave	-0.500000
Eddie	0.100000

Table 5: MOVIE (item) BIAS

Crazy Rich Asians	0.8333333
Disney's Christopher Robin	0.5000000
Mamma Mia! Here We Go Again	-1.0000000
Ocean's 8	-1.5000000
Peter Rabbit	0.8333333
Solo: A Star Wars Story	0.2500000

- **Using your training data, calculate the bias for each user and each item.**

```
### User Bias
user_bias <- colMeans(UI_train,na.rm = T) - mean_value
user_bias %>% t %>% t  %>% kable(caption = "USER BIAS") %>%   kable_styling(c("bordered","striped"))
```

```
# results correspond to the columns in video N


### Item (movie) bias
movie_bias <- rowMeans(UI_train,na.rm = T) - mean_value
movie_bias %>% t %>% t  %>% kable(caption = "MOVIE (item) BIAS") %>%   kable_styling(c("bordered","stri
```

```
# results correspond to the rows in video N
```

- **From the raw average, and the appropriate user and item biases, calculate the baseline predictors for every user-item combination.**

```r
### start from the matrix of the mean_rating
baseline_predictor <- mean_rating

minrating = 1
maxrating = 5
for (r in 1:nrow(baseline_predictor))
  for (c in 1:ncol(baseline_predictor))
    baseline_predictor[r,c] <-
  ### We have to ensure that the results are in the range [minrating,maxrating]
  ### which is why we have the min(max()) wrapper
  min(
    max(
      baseline_predictor[r,c] + movie_bias[r] + user_bias[c],
      1),
    5)

baseline_predictor
```

```
##                                Alice      Bob Carol     Dave    Eddie
## Crazy Rich Asians           4.633333 3.166667  5.00 3.833333 4.433333
## Disney's Christopher Robin  4.300000 2.833333  4.75 3.500000 4.100000
## Mamma Mia! Here We Go Again 2.800000 1.333333  3.25 2.000000 2.600000
## Ocean's 8                   2.300000 1.000000  2.75 1.500000 2.100000
## Peter Rabbit                4.633333 3.166667  5.00 3.833333 4.433333
## Solo: A Star Wars Story     4.050000 2.583333  4.50 3.250000 3.850000
```

```r
## matches the table in video "O"
```

- **Calculate the RMSE for the baseline predictors for both your training data and your test data.**

```
### Training RMSE
train_RMSE_baseline <- sqrt( mean ( (UI_train-baseline_predictor)^2, na.rm=T ) )
train_RMSE_baseline
```

```
## [1] 0.4708886
```

```
## 0.4709 matches the TRAINING RMSE value in video "P"

### Test RMSE
test_RMSE_baseline <- sqrt( mean ( (UI_test-baseline_predictor)^2, na.rm=T ) )
test_RMSE_baseline
```

```
## [1] 0.736546
```

```
## 0.7365 matches the TEST RMSE value in video "P"
```

- **Summarize your results.**

```
### improvement in TRAIN RMSE when moving from raw average to baseline predictor
train_RMSE_improvement = 1 - train_RMSE_baseline/ train_RMSE_raw
train_RMSE_improvement
```

```
## [1] 0.5947236
```

```
### improvement in TEST RMSE when moving from raw average to baseline predictor
test_RMSE_improvement  = 1 - test_RMSE_baseline / test_RMSE_raw
test_RMSE_improvement
```

```
## [1] 0.2812047
```

```
### Improvements match the results given in video "P"
```

The training RMSE declined from 1.162 to 0.471, which is an improvement of 59.472 percent.

The testing RMSE declined from 1.025 to 0.737, which is an improvement of 28.12 percent.