Project_3_0523_panz2

PanZhang & JiangongLi

April 27, 2019

Two methods: **UBCF, IBCF**

Collaborative filtering is a technique used by recommender systems, that making prediction about a user by gathering preference or taste from many other users. This approach has advantages such as no needs to define features and can recommend outside the user's profile. However, the technique needs enough data to start recommendation and cannot not recommend new. There are two types of Collaborative filtering: **user-based (UBCF) and Item-based (IBCF). User-based collaborative filtering** calculate the similarity between users, then predict the rating or index from the same group. **Item- based collaborative filtering** calculate the similarity between items, then predict the rating or index from the same group. In general, the IBCF have more computational cost than UBCF.

We chose to normalize the rating matrix using "Z-score" normalization option. The raw ratings are discrete, including 1,2,3,4,5. After Z-score normalization, the values of rating become continuous, the distribution of rating is closer to a normal distribution with mean=0, and the range of rating is between -4 and 4.

nn = 25 (same with the default) is used in Recommender when method is UBCF. Here, nn = 25 means that the number of a nearest neighborhood of similar users is 25. And then missing ratings for a user can be predicted based on a neighborhood of similar users.

k =30 (default) is used in Recommender when method is IBCF. Here, k = 25

means that only a list of the k most similar items for each item and their similarity values are stored for prediction.

We used Cosine method to do similarity measure for both UBCF and IBCF. Cosine similarity measures the similarity between two vectors by calculating the cosine of the angle between them. And the similarity matrix was not normalized.

For the missing values after running UBCF and IBCF, we replaced them with the mean rating value of train data.

The RMSE of UBCF is 1.032978, and the RMSE of IBCF is 1.169354.

```
set.seed(0523)
setwd("/Users/panzhang/Desktop/STAT542/project3")
library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

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

library(recommenderlab)

## Loading required package: Matrix

## Loading required package: arules

##
## Attaching package: 'arules'

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

```
## The following objects are masked from 'package:base':
##
##     abbreviate, write

## Loading required package: proxy

##
## Attaching package: 'proxy'

## The following object is masked from 'package:Matrix':
##
##     as.matrix

## The following objects are masked from 'package:stats':
##
##     as.dist, dist

## The following object is masked from 'package:base':
##
##     as.matrix

## Loading required package: registry

## Registered S3 methods overwritten by 'registry':
##   method               from
##   print.registry_field proxy
##   print.registry_entry proxy

library(reshape2)
```

Read file

```
ratings = read.csv('ratings.dat', sep = ':',
    colClasses = c('integer', 'NULL'), header = FALSE)
colnames(ratings) = c('UserID', 'MovieID', 'Rating', 'Timestamp')
dim(ratings)

## [1] 1000209       4

movies = readLines('movies.dat')
movies = strsplit(movies, split = "::", fixed = TRUE, useBytes = TRUE)
movies = matrix(unlist(movies), ncol = 3, byrow = TRUE)
movies = data.frame(movies, stringsAsFactors = FALSE)
colnames(movies) = c('MovieID', 'Title', 'Genres')
movies$MovieID = as.integer(movies$MovieID)
movies$Genres = ifelse(grepl('\\|', movies$Genres), "Multiple",
                       movies$Genres)
rating_merged = merge(x = ratings, y = movies, by.x = "MovieID")
users = read.csv('users.dat', sep = ':', header = FALSE)
users = users[,-c(2,4,6,8)]
colnames(users) = c('UserID', 'Gender', 'Age', 'Occupation',"Zip")
```

Prepare training and test data

```
ratings$Timestamp = NULL;
colnames(ratings) = c('user', 'movie', 'rating')
train.id = sample(nrow(ratings), floor(nrow(ratings)) * 0.6)
train = ratings[train.id, ]
head(train)
```

```
##        user movie rating
## 718577 4305  2148      3
## 515369 3182  3438      1
## 469017 2888  3527      5
## 91012   602   541      5
## 528517 3266  2858      5
## 175915 1113  2064      5
```

```
test = ratings[-train.id, ]
test.id = sample(nrow(test), floor(nrow(test)) * 0.5)
test = test[test.id, ]
head(test)
```

```
##        user movie rating
## 203493 1248  1721      3
## 881392 5323  3717      2
## 515687 3182   344      5
## 856590 5142  2373      3
## 654078 3942  1885      1
## 89837   591  3370      4
```

```
label = test[c('user', 'rating')]
test$rating = NULL
head(label)
```

```
##        user rating
## 203493 1248      3
## 881392 5323      2
## 515687 3182      5
## 856590 5142      3
## 654078 3942      1
## 89837   591      4
```

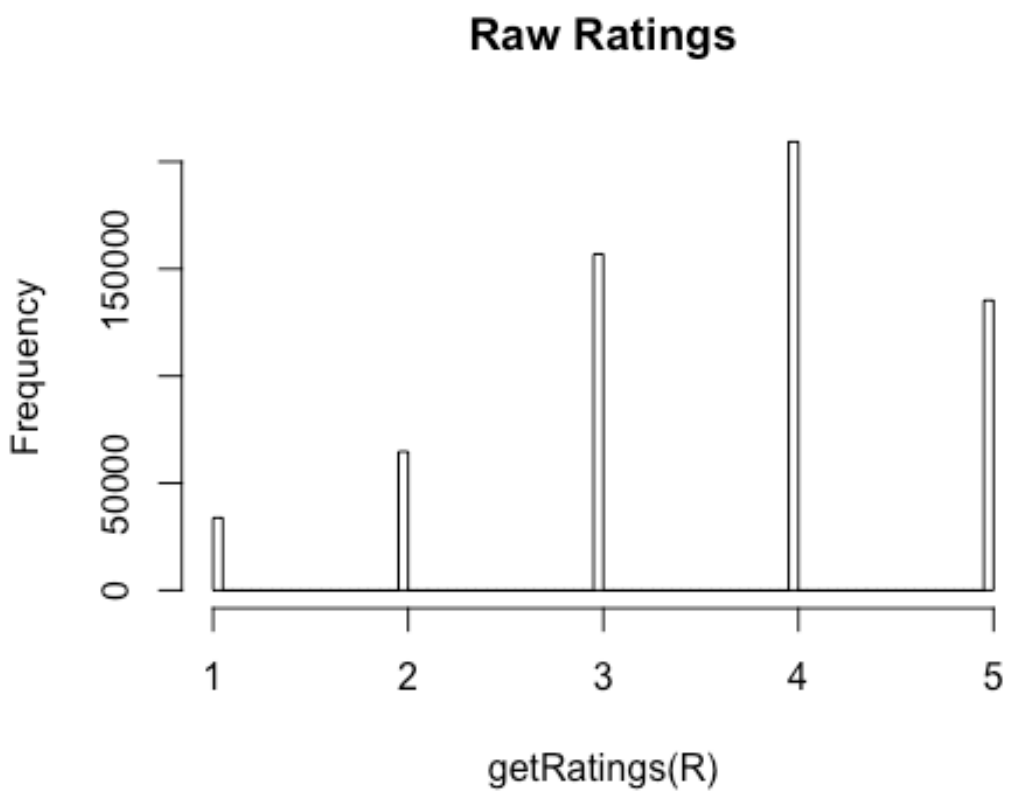
```
head(test)
```

```
##        user movie
## 203493 1248  1721
## 881392 5323  3717
## 515687 3182   344
## 856590 5142  2373
## 654078 3942  1885
## 89837   591  3370
```

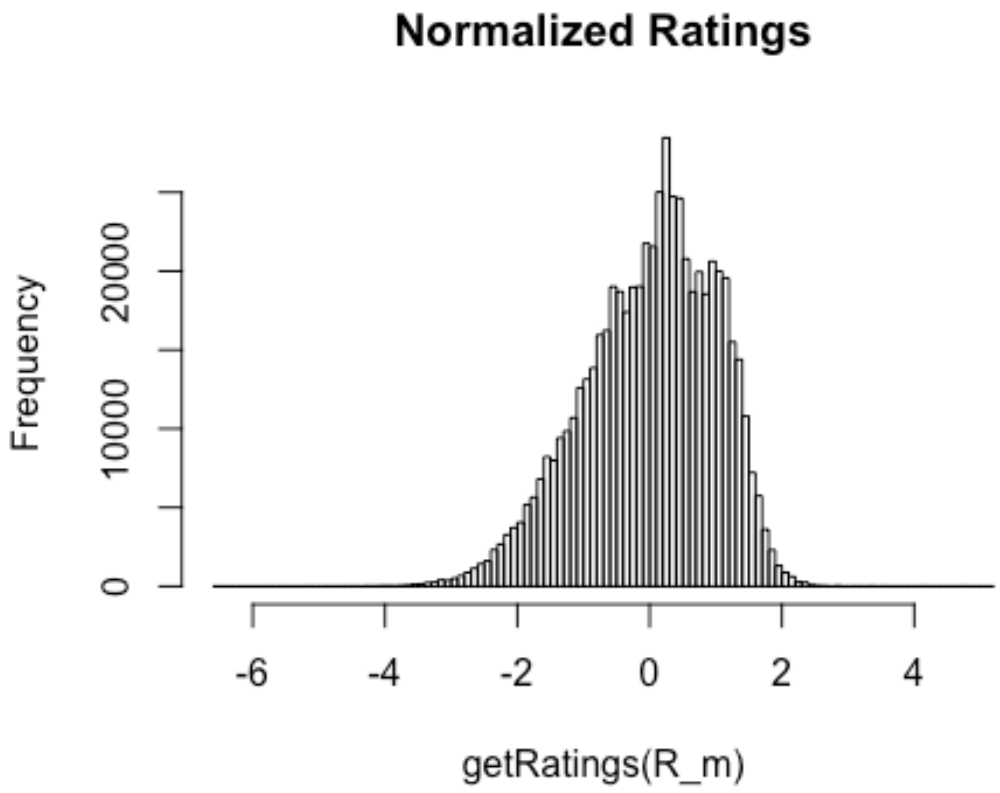
```r
# function for RSEM calculation
rmse <- function(actuals, predicts){
  return (sqrt(mean((actuals - predicts)^2, na.rm = T)))
}

# function for predicting test and output RSEM
pred_test_evaluate <- function(test, rec_list){
  test$rating = NA
  # For all lines in test file, one by one
  for (u in 1:nrow(test)){
      # Read userid and movieid from columns 2 and 3 of test data
      userid = as.character(test$user[u])
      movieid = as.character(test$movie[u])
      rating = rec_list[[userid]][movieid]
      test$rating[u] = ifelse(is.na(rating), mean(train$rating), rating)
  }
  return(rmse(label$rating,test$rating))
}
```

Recommender System

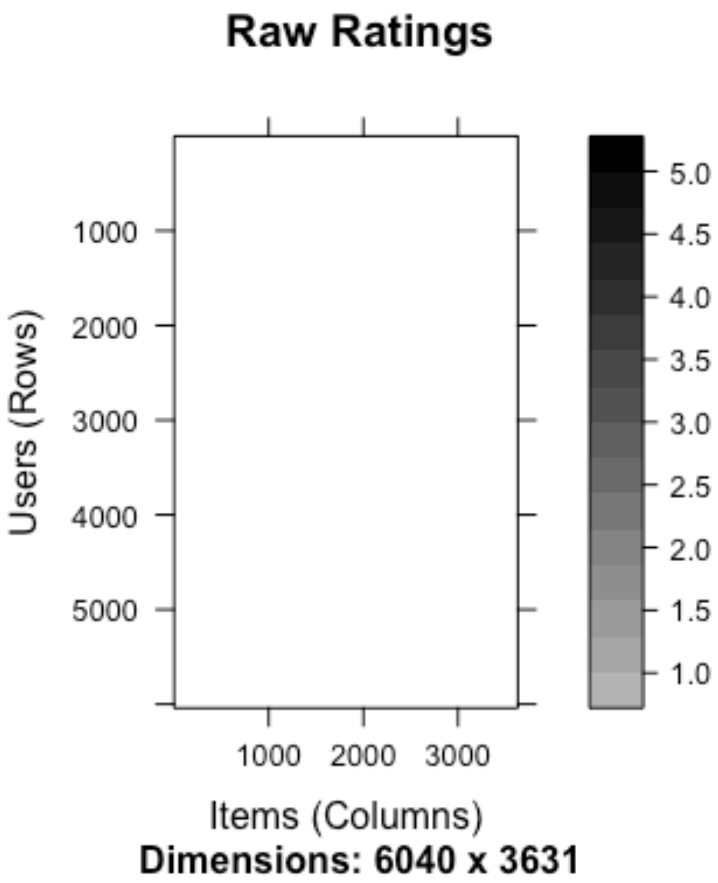
```r
R = acast(train, user ~ movie)

## Using rating as value column: use value.var to override.

R = as(R, 'realRatingMatrix')
R_m = normalize(R, method="Z-score")
hist(getRatings(R), main = "Raw Ratings",breaks = 100)
```



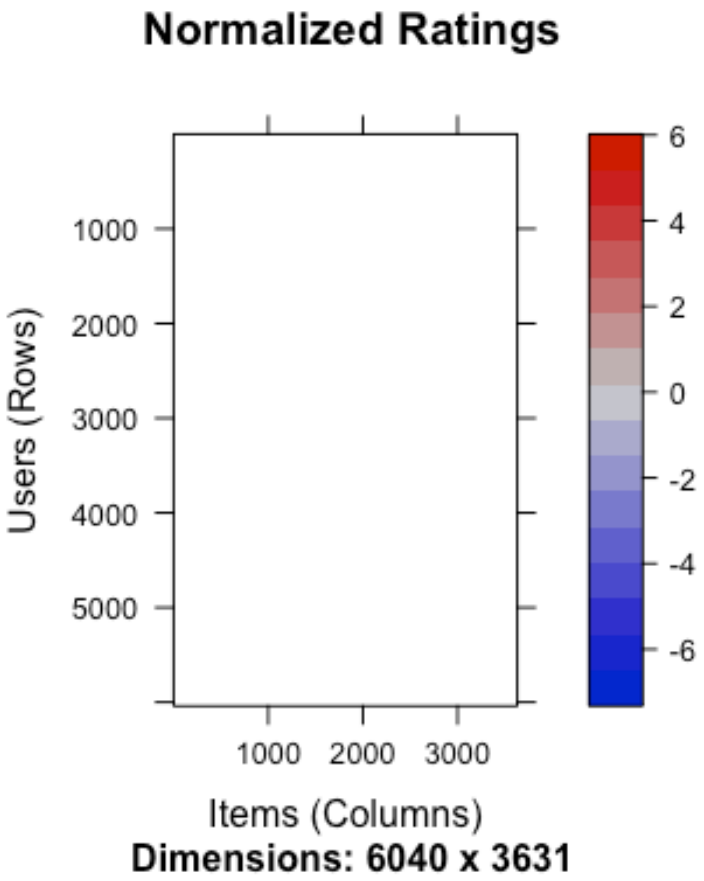

```
hist(getRatings(R_m), main = "Normalized Ratings",breaks = 100)
```




```r
image(R, main = "Raw Ratings")
```




```r
image(R_m, main = "Normalized Ratings")
```




```
#head(getRatingMatrix(R_m))

recommenderRegistry$get_entries(dataType = "realRatingMatrix")
```

```
## $ALS_realRatingMatrix
## Recommender method: ALS for realRatingMatrix
## Description: Recommender for explicit ratings based on latent factors, cal
culated by alternating least squares algorithm.
## Reference: Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, Rong Pan (200
8). Large-Scale Parallel Collaborative Filtering for the Netflix Prize, 4th I
nt'l Conf. Algorithmic Aspects in Information and Management, LNCS 5034.
## Parameters:
##   normalize lambda n_factors n_iterations min_item_nr seed
## 1      NULL    0.1        10           10           1 NULL
##
## $ALS_implicit_realRatingMatrix
## Recommender method: ALS_implicit for realRatingMatrix
## Description: Recommender for implicit data based on latent factors, calcul
ated by alternating least squares algorithm.
## Reference: Yifan Hu, Yehuda Koren, Chris Volinsky (2008). Collaborative Fi
ltering for Implicit Feedback Datasets, ICDM '08 Proceedings of the 2008 Eigh
th IEEE International Conference on Data Mining, pages 263-272.
## Parameters:
```

```
##   lambda alpha n_factors n_iterations min_item_nr seed
## 1    0.1    10        10           10           1 NULL
##
## $IBCF_realRatingMatrix
## Recommender method: IBCF for realRatingMatrix
## Description: Recommender based on item-based collaborative filtering.
## Reference: NA
## Parameters:
##    k   method normalize normalize_sim_matrix alpha na_as_zero
## 1 30 "Cosine"  "center"                FALSE   0.5      FALSE
##
## $POPULAR_realRatingMatrix
## Recommender method: POPULAR for realRatingMatrix
## Description: Recommender based on item popularity.
## Reference: NA
## Parameters:
##   normalize
## 1  "center"
##                                                    aggregationRatings
## 1 new("standardGeneric", .Data = function (x, na.rm = FALSE, dims = 1,
##                                                  aggregationPopularity
## 1 new("standardGeneric", .Data = function (x, na.rm = FALSE, dims = 1,
##
## $RANDOM_realRatingMatrix
## Recommender method: RANDOM for realRatingMatrix
## Description: Produce random recommendations (real ratings).
## Reference: NA
## Parameters: None
##
## $RERECOMMEND_realRatingMatrix
## Recommender method: RERECOMMEND for realRatingMatrix
## Description: Re-recommends highly rated items (real ratings).
## Reference: NA
## Parameters:
##   randomize minRating
## 1         1        NA
##
## $SVD_realRatingMatrix
## Recommender method: SVD for realRatingMatrix
## Description: Recommender based on SVD approximation with column-mean imput
ation.
## Reference: NA
## Parameters:
##    k maxiter normalize
## 1 10     100  "center"
##
## $SVDF_realRatingMatrix
## Recommender method: SVDF for realRatingMatrix
## Description: Recommender based on Funk SVD with gradient descend.
## Reference: NA
```

```
## Parameters:
##    k gamma lambda min_epochs max_epochs min_improvement normalize verbose
## 1 10 0.015  0.001         50        200           1e-06  "center"   FALSE
##
## $UBCF_realRatingMatrix
## Recommender method: UBCF for realRatingMatrix
## Description: Recommender based on user-based collaborative filtering.
## Reference: NA
## Parameters:
##     method nn sample normalize
## 1 "cosine" 25  FALSE  "center"
```

```
#reconmmender for UBCF
rec_ubcf = Recommender(R, method = "UBCF",
    parameter = list(normalize = 'Z-score', method = 'Cosine', nn=25)
    )
print(rec_ubcf)
```

```
## Recommender of type 'UBCF' for 'realRatingMatrix'
## learned using 6040 users.
```

```
names(getModel(rec_ubcf))
```

```
## [1] "description" "data"        "method"      "nn"          "sample"
## [6] "normalize"   "verbose"
```

```
recom_ubcf = predict(rec_ubcf, R, type = 'ratings')  # predict ratings. This
may be slow.
rec_list_ubcf = as(recom_ubcf, 'list')  # each element are ratings of that us
er
pred_test_evaluate(test, rec_list_ubcf)
```

```
## [1] 1.032978
```

```
#reconmmender for IBCF
rec_ibcf = Recommender(R, method = "IBCF",
    parameter = list(normalize = 'Z-score', method = 'Cosine')
    )
print(rec_ibcf)
```

```
## Recommender of type 'IBCF' for 'realRatingMatrix'
## learned using 6040 users.
```

```
names(getModel(rec_ibcf))
```

```
## [1] "description"         "sim"                 "k"
## [4] "method"              "normalize"           "normalize_sim_matrix"
## [7] "alpha"               "na_as_zero"          "verbose"
```

```
recom_ibcf = predict(rec_ibcf, R, type = 'ratings')  # predict ratings. This
may be slow.
rec_list_ibcf = as(recom_ibcf, 'list')  # each element are ratings of that us
```

```r
er
pred_test_evaluate(test, rec_list_ibcf)
```

```
## [1] 1.169354
```