sm56_project4_analysis

Raja Mukherjee

11/28/2020

System 1

Primary objective of System 1 to provide a list of movies based of users selection of a particular genre.

Preprocessing steps for first movie

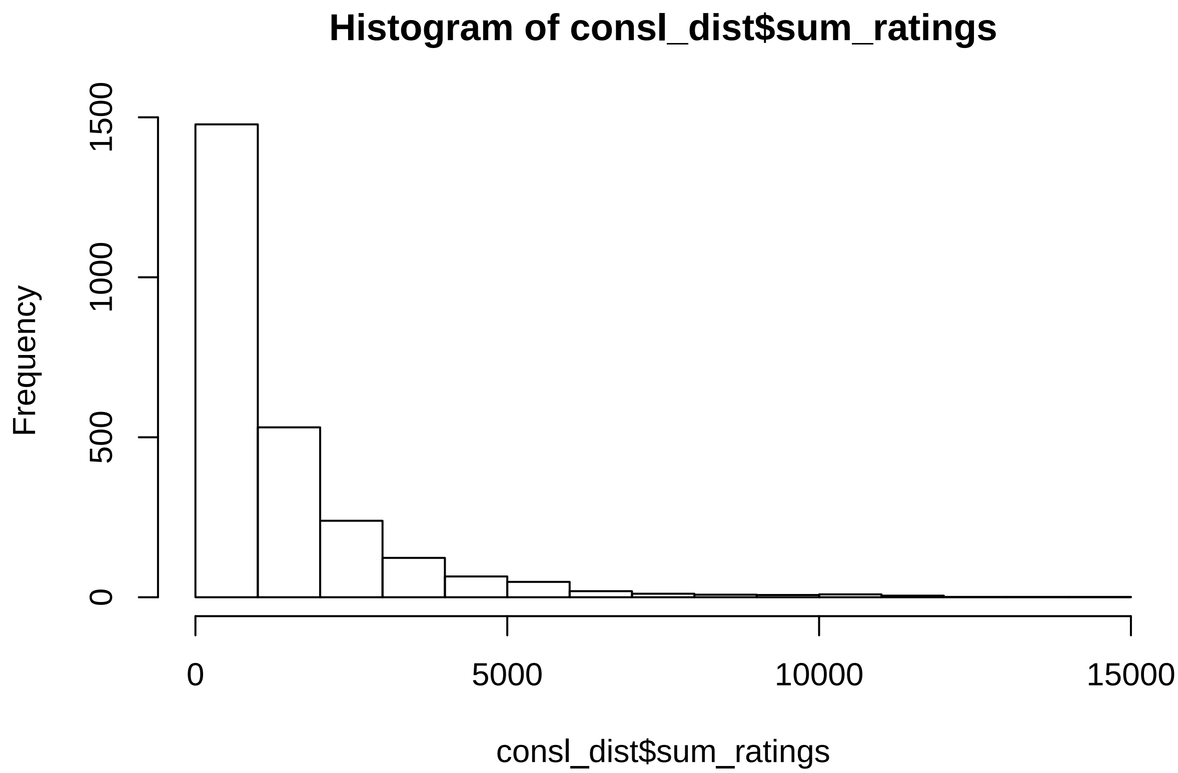
- Ingest movies file, ratings file and users file.

First recommendation scheme - All Time Most Popular - Highest Rating Among Most Reviewed

- Join these files using ratings and movies file using MovieID key
- Remove records (ratings) for movies which have recieved less than 47 rating.
- Rating were summed up by each movie, so it evaluates to net amount of stars received.
- This is based on the intuition that more successful/seen movies have received more reviews.
  And, also the fact that people are ratings a movie shows that they have formed an opinion about it.
- Distribution of sum of stars for each movie show that approx. 100 movies have received a lot of ratings over the years. While the rest of the movies and received a decent amount of ratings.

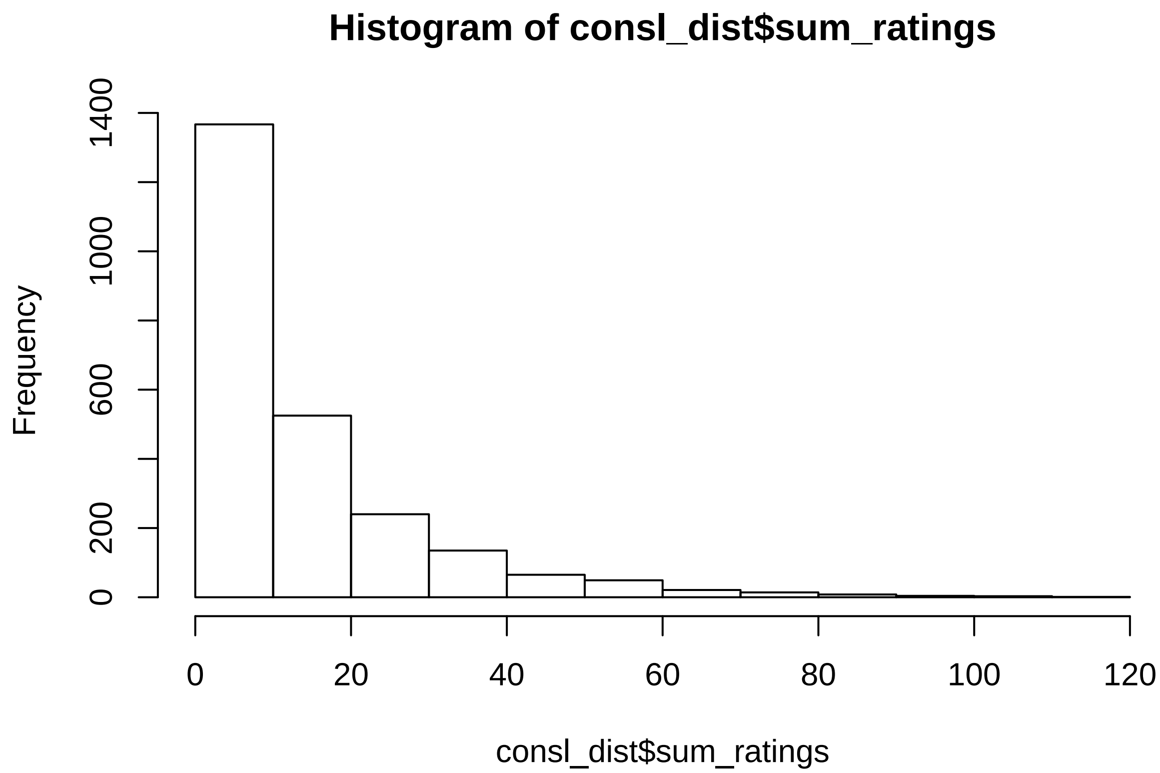
```
## [1] "Distribution of Sum of Ratings by MovieID"
```

```
##    0%     5%   10%     15% 20% 25% 30%     35% 40%    45% 50%   55%  60%     65%
## 1: 91 169.25 203.5 248.75 297 348 398 483.75 562 658.25 769 901.5 1051 1219.75
##       70%    75%  80%    85%  90%  95%  100%
## 1: 1432.5 1723.5 2091 2652.5 3342 4672 14800
```




```
## [1] "Distribution of Sum of Ratings by MovieID"
```

```
##    0% 5% 10% 15% 20% 25% 30% 35% 40% 45% 50% 55% 60% 65% 70% 75% 80% 85% 90%
## 1:  1  2   3   3   4   4   5   5   6   8   9  10  12  14  16  19  22  27  34
##    95% 100%
## 1:  48  113
```



Second recommendation scheme - Trending Movies (Recent reviews) - Highest Rating Among Most Reviewed

- Since I was continually hitting the barrier of out of memory, had to implement the model/scheme on a subset of data.
  Selected most recent rated 10000 rows by timestamp.

- Join these files using ratings and movies file using MovieID key
- Remove records (ratings) for movies which have recieved less than 47 rating.
- This is exactly same as previous recommendation, difference being that we only consider most recent 10000 reviews by Timestamp column.
- Here also we see similar distribution of number of movies.

System 2

Shiny Hosted App - https://rajamukherjee.shinyapps.io/movie_recommendation/?_ga=2.267501194.523982317.1607448657-1652788879.1607109954

Preprocessing steps

```
- Steps to normalize and improve skewness
- Movies which had received rating from less than 47 users were removed, as they might skew inferences/r
- Rating were normalized using z-score. This helped reduce skewness.
```

Considered collaborative systems

```
- Models considered were UBCF with Cosine similarity, IBCF with Cosine similarity, IBCF with Pearson si
  - UBCF with Cosine similarity: This algorithm classified users together by grouping together which us
  - IBCF with Cosine similarity: Items which were classified by same users with similar rating were gro
- Details regarding training, evaluation and parameters -
  - Recommenderlab package (evaluationScheme and evaluate) was used to evaluate between various models.
  - nn (neighbours) = 50 was considered to avoid NA issues while recommending
  - k = 4 (cross validation 4 fold)
  - Since Ratings are skewed towards higher scores, goodRating is set to 4.
  - "given" parameter is set to 15 (as this is minimum of items per user) so these items per user for t
```

Minimum Number of movies per user (The "given" parameter cannot be greater than this number)

```
## [1] 15
```

```r
evlt <- evaluationScheme(data = rating_matrix,
                         method = "split",
                         k = 4,
                         given = 15,
                         goodRating = 4)
evlt
```

Training different recommender systems (using train data)

```r
# eval=FALSE, include=FALSE
models_to_evaluate = list(
  `UBCF Pearson` = list(name = "UBCF",
                        param = list(normalize = "Z-score", method = "pearson", nn = 50)),
  `UBCF Cosine` = list(name = "UBCF",
                       param = list(normalize = "Z-score", method = "cosine", nn = 50)),
  `IBCF Pearson` = list(name = "IBCF",
                        param = list(normalize = "Z-score", method = "pearson", normalize_sim_matrix = 
  `IBCF Cosine` = list(name = "IBCF",
                       param = list(normalize = "Z-score", method = "cosine", normalize_sim_matrix = TR
  `Random` = list(name = "RANDOM",
                  param=NULL),
  `Popular` = list(name = "POPULAR",
                   param = NULL)
)

n_recommendations = c(5, 10, 15)

list_results = evaluate(x = evlt,
                        method = models_to_evaluate,
                        n = n_recommendations
                        )

# Save the model to a file
save(list_results, file = 'movie_recommendation/model/run_results.rda')
```

Loading saved models, and performing evaluation

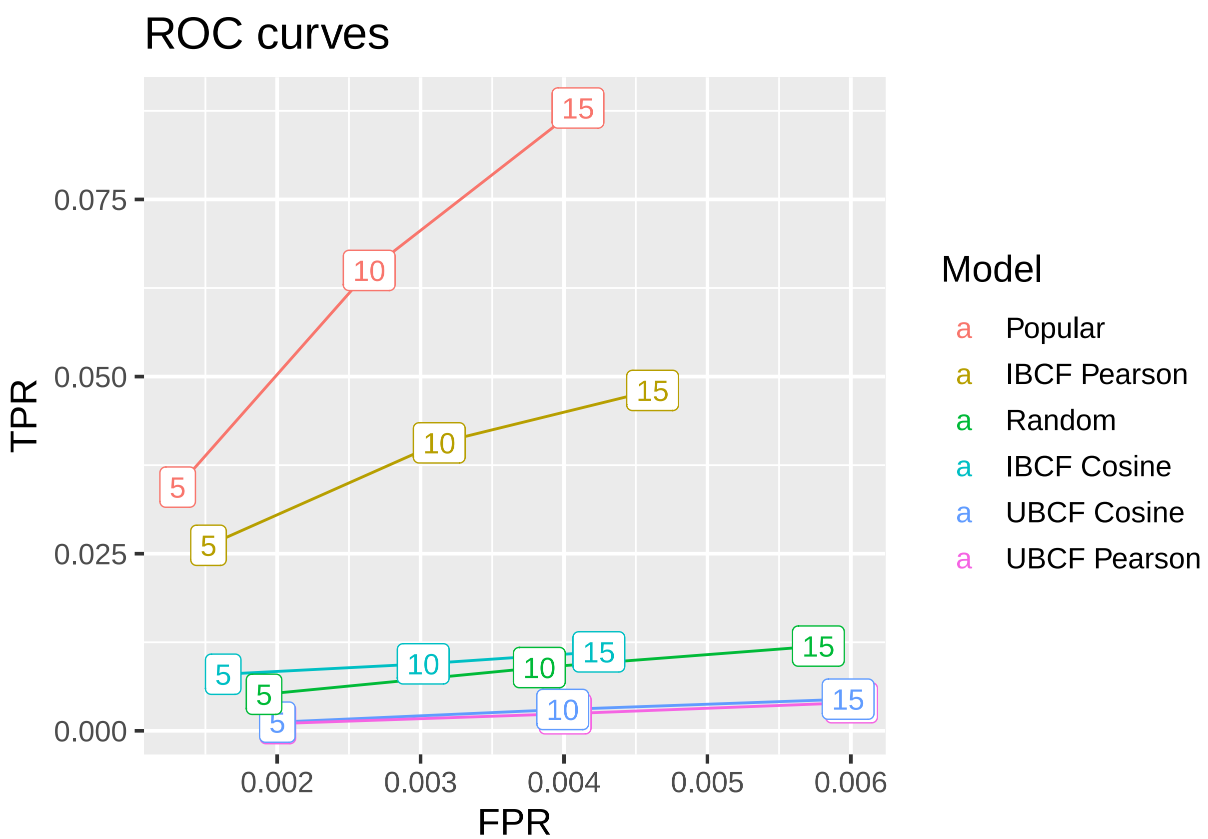
```r
# save(list_results, file = 'model/run_results.rda')
# print(getwd())
n_recommendations = c(5, 10, 15)
load(file = 'movie_recommendation/model/run_results.rda')
```

Though below graphs show that Popular seems to be performing the best as shown in ROC curve below, I could only understand how IBCF and UBCF works and so have used these two in final recommendations.

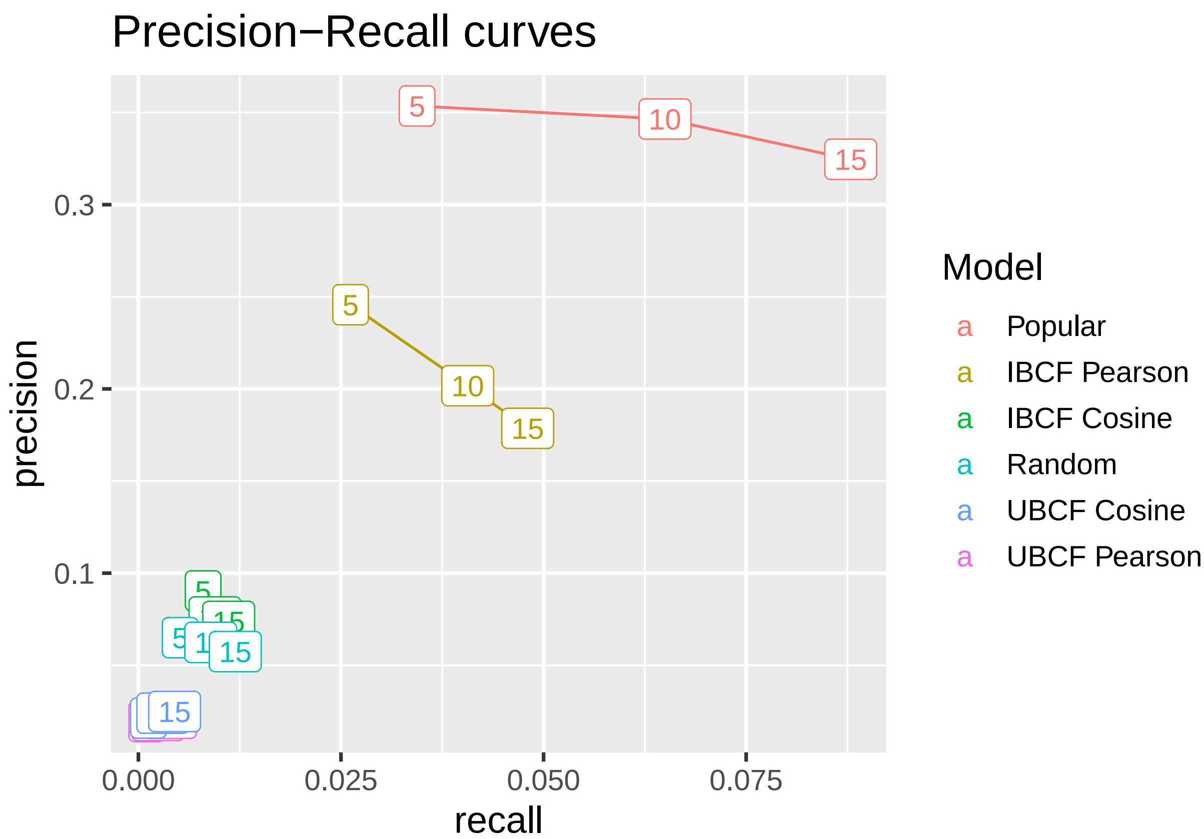
```r
avg_conf_matr <- function(results) {

  tmp <- results %>%
    # Pull into a list of all confusion matrix information for one model
    getConfusionMatrix()  %>%
    as.list()
    # Calculate average value of 4 cross-validation rounds
    as.data.frame(Reduce("+",tmp) / length(tmp)) %>%
    mutate(n = n_recommendations) %>%
    # select only columns needed and sorting out order
    select('n', 'precision', 'recall', 'TPR', 'FPR')
}

# Using map() to iterate function across all models
results_tbl <- list_results %>%
  map(avg_conf_matr) %>%
# Turning into an unnested tibble
  enframe() %>%
# Unnesting to have all variables on same level
  unnest()
results_tbl
```

```
## # A tibble: 18 x 6
##    name             n precision  recall     TPR     FPR
##    <chr>        <dbl>     <dbl>   <dbl>   <dbl>   <dbl>
##  1 UBCF Pearson     5    0.0195 0.00102 0.00102 0.00201
##  2 UBCF Pearson    10    0.0200 0.00241 0.00241 0.00401
##  3 UBCF Pearson    15    0.0212 0.00397 0.00397 0.00600
##  4 UBCF Cosine      5    0.0214 0.00122 0.00122 0.00200
##  5 UBCF Cosine     10    0.0240 0.00301 0.00301 0.00399
##  6 UBCF Cosine     15    0.0249 0.00447 0.00447 0.00598
##  7 IBCF Pearson     5    0.246  0.0262  0.0262  0.00152
##  8 IBCF Pearson    10    0.202  0.0406  0.0406  0.00313
##  9 IBCF Pearson    15    0.179  0.0480  0.0480  0.00462
## 10 IBCF Cosine      5    0.0903 0.00799 0.00799 0.00162
## 11 IBCF Cosine     10    0.0762 0.00945 0.00945 0.00302
## 12 IBCF Cosine     15    0.0738 0.0111  0.0111  0.00424
## 13 Random           5    0.0650 0.00516 0.00516 0.00191
## 14 Random          10    0.0624 0.00893 0.00893 0.00383
## 15 Random          15    0.0574 0.0120  0.0120  0.00577
## 16 Popular          5    0.354  0.0344  0.0344  0.00131
## 17 Popular         10    0.346  0.0650  0.0650  0.00264
## 18 Popular         15    0.325  0.0879  0.0879  0.00410
```

```r
results_tbl %>%
  ggplot(aes(FPR, TPR,
             colour = fct_reorder2(as.factor(name),
                                   FPR, TPR))) +
  geom_line() +
  geom_label(aes(label = n))  +
  labs(title = "ROC curves", colour = "Model") +
  theme_grey(base_size = 14)
```




```r
results_tbl %>%
  ggplot(aes(recall, precision,
             colour = fct_reorder2(as.factor(name),
                                   precision, recall))) +
  geom_line() +
  geom_label(aes(label = n))  +
  labs(title = "Precision-Recall curves", colour = "Model") +
  theme_grey(base_size = 14)
```


Precision−Recall curves
precision
0.3
0.2
0.1
recall
0.000
0.025
0.050
0.075
5
10
15
Model
Popular
IBCF Pearson
IBCF Cosine
Random
UBCF Cosine
UBCF Pearson