Movie lens Analysis and Rating Prediction

Jingyi Zhu, NetID:jingyiz9 Xi Chen, NetID:xic6

1 Introduction

In this project, we use the movie lens dataset contains 1 million ratings from 6000 users on 4000 movies to build a movie recommender system to predict the rating of a movie. The ratings data has 1000209 observations and 4 columns: UserID, MovieID, Rating and Timestamp. We split the observations into train data that contains about 60% rows of the observations and test data that contains about 20% of the user-movie pairs from the ratings.dat from the MovieLens 1M dataset.

2 Method

In this report, we build the recommender system and use two models: Collaborative Filtering (CF) Method with User-based (UBCF) and with Item-based (IBCF). First, we set the seed 6404 at the beginning of the code to acquire the same result each time. We split the data into train data and test data. The train data contains just 60% of the original ratings, so it is possible some movies in movies.dat or users in users.dat do not appear in the training, but in test. Then, we apply two models to the train data we created.

load the data

```
setwd("~/Desktop/STAT542_project3/ml-1m")
set.seed(6404)

# read ratings data
# use colClasses = 'NULL' to skip columns
ratings = read.csv('ratings.dat', sep = ':',
    colClasses = c('integer', 'NULL'), header = FALSE)
colnames(ratings) = c('UserID', 'MovieID', 'Rating', 'Timestamp')
dim(ratings) # 1000209-by-4
```

```
## [1] 1000209       4
```

```
head(ratings)
```

```
##   UserID MovieID Rating Timestamp
## 1      1    1193      5 978300760
## 2      1     661      3 978302109
## 3      1     914      3 978301968
## 4      1    3408      4 978300275
## 5      1    2355      5 978824291
## 6      1    1197      3 978302268
```

```
# read movies data
# In movies.dat, some movie names contain single colon (:), so the above
# method does not work.
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)
head(movies)
```

```
##   MovieID                               Title                        Genres
## 1       1                    Toy Story (1995)  Animation|Children's|Comedy
## 2       2                      Jumanji (1995) Adventure|Children's|Fantasy
## 3       3             Grumpier Old Men (1995)                Comedy|Romance
## 4       4            Waiting to Exhale (1995)                  Comedy|Drama
## 5       5 Father of the Bride Part II (1995)                        Comedy
## 6       6                           Heat (1995)         Action|Crime|Thriller
```

```
#Explore the relationship between movie ratings and movie genres. First, simplify movie genres: multipl
movies$Genres = ifelse(grepl('\\|', movies$Genres), "Multiple",
                       movies$Genres)
rating_merged = merge(x = ratings, y = movies, by.x = "MovieID")
```

Prepare training and test data

```
ratings$Timestamp = NULL;
colnames(ratings) = c('user', 'movie', 'rating')

#train dataset
train.id = sample(nrow(ratings), floor(nrow(ratings)) * 0.6)
train = ratings[train.id, ]

#test dataset
test= ratings[-train.id, ]
test.id = sample(nrow(test), floor(nrow(test)) * 0.5)
test = test[test.id, ]
test2=test
test3=test
label = test[c('user', 'rating')]
test$rating = NULL
```

build R matrix

```
#First, create a utility matrix.
R = acast(train, user ~ movie)
```

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

```
R = as(R, 'realRatingMatrix')
#Normalize the utility matrix and visualize data:
R_m = normalize(R)
```

2.1 Model 1: Collaborative Filtering (CF) Method with User-based (UBCF)

Normalize the data

To explore the relationship between movie ratings and movie genres, we simplify movie genres: multiple genres to 'Multiple'. Then we merge ratings and movie datasets by the MovieID.
We train a recommender system and make prediction on the test data. First, we create a utility matrix and normalize the utility matrix with Z-score method. User rating zero mean each user's ratings is subtracted by its own mean to center the mean at zero. Z-scoring is that additionally divides each user's rating by its standard deviation. After normalizing the rating matrix R we acocunt for individual row bias of each user and make sure that all ratings are scaled similarly.

Apply the recommender system with UBCF

Then we apply the recommender system with UBCF method and obtain a short summary of the model. nn parameter sets the neighborhood of most similar users to consider for each user profile. We set nn = 25 so the ratings profiles of the 25 nearest neighbors will be the basis for making predictions on a users unrated items profile. We use the Cosine similarity metric due to the numeric ratings to calculate similarity between users real ratings profile. For per item, we calculate the average of ratings by each user's 25 most similar users. Weight the average ratings based on similarity score of each user whose rated the item and similarity score equals weight.

Deal with the missing value in the result

After running UBCF we obtain the test rating based on the UserID and MovieID to find ratings in the rec_list. If the rating is missing value in the matrix then we use 3.5 to replace the origin missing value because the common ratings are between 3 and 4. However, 3.5 is just a reletively appropriate value and it may be too arbitrary depends on different situations.
We use the test data we split above to make the prediction and the RMSE is 1.03389.

```
#Learn a recommender.
#recommenderRegistry$get_entries(dataType = "realRatingMatrix")
rec = Recommender(R, method = 'UBCF',
    parameter = list(normalize = 'Z-score', method = 'Cosine', nn = 25)
    )

#A short summary of the model:
print(rec)
```

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

```
names(getModel(rec))
```

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

```
# make prediction on the test data
recom = predict(rec, R, type = 'ratings')  # predict ratings. This may be slow.
rec_list = as(recom, 'list')  # each element are ratings of that user
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]
    # 2.5 may be too arbitrary
    test$rating[u] = ifelse(is.na(rating), 3.5, rating)
}

# write submission file
write.table(test, file = 'mysubmission1.csv', row.names = FALSE,
            col.names = TRUE, sep = ',')
#rmse
sqrt(mean((label$rating-test$rating)^2))
```

```
## [1] 1.03389
```

2.2 Model 2: Collaborative Filtering (CF) Method with Item-based (IBCF)

Item-based CF approach is very similar to user-based. But in this one, similarity is computed between items, not users. Assumption is that users will prefer items similar to other items they like. As with item-based CF, we have used center method to normalize the rating matrix and Cosine similarity metric. We set k = 350 so the ratings profiles of the 350 nearest neighbors will be the basis for making predictions on a item. IBCF doesn't need to access the initial data. For each item, the model stores the k-most similar, so the amount of information is small once the model is built.
After running IBCF we obtain the test rating based on the UserID and MovieID to find ratings in the rec_list. If the rating is missing value in the matrix then we use 3.5 to replace the origin missing value.
We apply the recommender system with most method and obtain a short summary of the model. After using predict() function we acquired the fitted values for the test data and the RMSE is 1.324066.

```
#Learn a recommender.
rec2 = Recommender(R, method = 'IBCF',
    parameter = list(normalize = 'center', method = 'Cosine', k = 350)
    )

#A short summary of the model:
print(rec2)
```

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

```
names(getModel(rec2))
```

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

```
#make prediction on the test data
#recommmend top 5 movies to each of the users
recom2 = predict(rec2, R, type = 'ratings', n=10)
rec_list2 = as(recom2, 'list')  # each element are ratings of that use
test2$rating = NA
# For all lines in test file, one by one
for (u in 1:nrow(test2)){
    # Read userid and movieid from columns 2 and 3 of test data
    userid = as.character(test2$user[u])
```

```
    movieid = as.character(test2$movie[u])
    rating = rec_list2[[userid]][movieid]
    # 2.5 may be too arbitrary
    test2$rating[u] = ifelse(is.na(rating), 3.5, rating)
}

# write submission file
write.table(test2, file = 'mysubmission2.csv', row.names = FALSE,
            col.names = TRUE, sep = ',')
#rmse
sqrt(mean((label$rating-test2$rating)^2))
```

```
## [1] 1.324066
```

3 Result

For Model 1: Collaborative Filtering (CF) Method with User-based (UBCF) the RMSE is 1.03389 on the test data.
For Model 2: Collaborative Filtering (CF) Method with Item-based (IBCF) the RMSE is 1.324066 on the test data.

4 Conclusion

Comparing the two models UBCF and IBCF we applied in the recommender system, UBCF needs to access the initial data and keep in the memory so it doesn't work well in a big rating matrix. Also, UBCF needs more computing power and time since building the similarity matrix. However, UBCF's accuracy is proven to be slightly more accurate than IBCF.

5 Other Information

- Computer system: MacBook Pro, 2.53 GHz, 4GB memory
- Running time of dataset: 28.27505 minutes
- Libraries used: dplyr recommenderlab reshape2