# Recommender System Report

*Chenzi Zhang, NetID chenziz2; Zhaoxu Qiu, NetID zhaoxuq2*

*4/28/2019*

## Contents

**Code** **1**
Step1: set seed with UID . . . . . 1
Step2: create train data & test data . . . . . 1
Step3: Build two models to predict the movie rating for each user_movie pair in test data . . . . . 2
Step 4: Report the RMSE of two models on the test data . . . . . 3

**Introduction** **3**

**Method** **4**
Model 1: SVDF . . . . . 4
Model 2: UBCF . . . . . 5

**Results** **6**
Evalution on Prediction Accuracy . . . . . 6
Evalution on Efficiency . . . . . 6
What's also interesting . . . . . 6

**Conclusion** **7**

## Code

### Step1: set seed with UID

```
if (!require("pacman")) install.packages("pacman")
```

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

```
pacman::p_load(
  "recommenderlab",
  "dplyr",
  "reshape2",
  "ggplot2"
)
set.seed(8837)
```

### Step2: create train data & test data

```
#setwd("")
ratings = read.csv('ratings.dat', sep = ':',
                   colClasses = c('integer', 'NULL'), header = FALSE) #1000209       4
colnames(ratings) = c('UserID', 'MovieID', 'Rating', 'Timestamp')
trainId = sample(nrow(ratings), floor(nrow(ratings)*0.6), replace = FALSE)
```

```r
train = ratings[trainId, ]
train$Timestamp = NULL

test = ratings[-trainId, ]
testId = sample(nrow(test), floor(nrow(test)*0.5), replace = FALSE)
test = test[testId, ]
test$Rating = NULL
test$Timestamp = NULL
```

## Step3: Build two models to predict the movie rating for each user_movie pair in test data

### Recommender System

```r
#create a utility matrix.
R = acast(train, UserID ~ MovieID) #row: UserID 6040, col: MovieID 3628
```

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

```r
R = as(R, 'realRatingMatrix')
```

### Model 1: SVDF

```r
SVDF.k = 7
SVDF.normalize = "center"

rec = Recommender(R, method = "SVDF",
                  parameter = list(normalize = SVDF.normalize, k = SVDF.k)
)
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$UserID[u])
  movieid = as.character(test$MovieID[u])

  rating = rec_list[[userid]][movieid]
  # set benchmark
  test$rating[u] = ifelse(is.na(rating), 3, rating)
}
SVDFtest = test
```

**Model 2: UBCF**

```
UBCF.normalize = "center"
UBCF.method = "Cosine"
UBCF.nn = 25

rec = Recommender(R, method = "UBCF",
                  parameter = list(normalize = UBCF.normalize, method = UBCF.method, nn = UBCF.nn)
)
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$UserID[u])
  movieid = as.character(test$MovieID[u])

  rating = rec_list[[userid]][movieid]
  # set benchmark
  test$rating[u] = ifelse(is.na(rating), 3, rating)
}
UBCFtest = test
```

## Step 4: Report the RMSE of two models on the test data

```
test.y = ratings[-trainId, ][testId, ]$Rating
SVDF.result = sqrt(mean((test.y - SVDFtest$rating)^2))
UBCF.result = sqrt(mean((test.y - UBCFtest$rating)^2))
print(SVDF.result)
```

```
## [1] 0.8820901
```

```
print(UBCF.result)
```

```
## [1] 1.035013
```

# Introduction

In our project, we are interested in analyzing the preference for MovieLens 1M users. The goal is to build models to predict the rating for movies which users will give based on Movielens 1M dataset with 1000209 rows and 4 variables. The train data set we use to build models contains 600125 rows of the ratings.dat from the MovieLens 1M dataset with 4 variables: UserID, MovieID, Rating, Timestamp. We drop the last variable for we do nothing with it. The test data set we use to examine accuracy have 200042 observations with same variables.

In this report, we use two models. The SVDF model has more parameters to be tunned but the best performace among all recommended models. The UBCF model is more efficient with smaller running time but has less accuracy. These two models have RMSE below or slightly larger than 1 and efficient running process.

## Method

We choose to use `recommenderlab` package to build two models: SVDF and UBCF. By inputing train data into these two models with tunned parameters, we will get a Recommender System for each model. Inputing test data into these two models, we will get the prediction result for `Rating` variable. In this way, we can conduct comperation between prediction and original `Rating` value by RMSE. Based on these result, we can compare the performances of two models.
Utility matrix: Before building models by `Recommender()`, we create utility matrix format data to be successfully inputed into `Recommender()` function. By using `acast()`, we convert train data into a matrix whose rows are `UserIDs` and colnums are `MovieIDs` with `Rating` as value filling the matrix. Then, using `as()` to convert the matrix into a `realRatingMatrix` , we succesfully deal with the missing value problem.

Problem: It is possible some movies in movies.dat or users in users.dat do not appear in the training, but in test. Solution: Since ratings data range from 1 to 5 and the mean is 3, we set missing user-movies pairs with value 3.

### Model 1: SVDF

By comparing the RMSE among all models in `recommenderlab` package, we choose SVDF who has the best performance as our first model. This model has a recommender based on Funk SVD with gradient descend. The original algorithm of Funk SVD was posted on blog by Simon Funk. This algorithm factorized our UserID~MovieID rating matrix as the product of two lower dimensional matrics instead of a Singular Value Decomposition. The first matrix has number of rows as UserID and the second matrix has number of columns as each MovieID. The row corresponding to a specific user and the column corresponding to a specific movie is referred to as latent factors, meaning the number of columns from first matrix and the number of rows from the second matrix are the number of latent factors. The rating matrix can be written as:

$$\tilde{R} = HW, \tilde{R} \in \mathbb{R}^{Users \times Movies}$$
$$H \in \mathbb{R}^{Users \times LatentFactors}$$
$$W \in \mathbb{R}^{Latentfactors \times Movies}$$

As a result, we can tune the number of latent factors. Increasing the number of latent factors will improve recommendation quality. However, when it reaches a point where the model start to earn overfitting, the recommendation quality will decrease. So, here in this model, we have a pernalization parameter. The objective function we want to minimize is:

$$\underset{H,W}{\arg\min} \|R - \tilde{R}\|_F + \alpha\|H\| + \beta\|W\|$$

where $\|.\|$ is defined to be the frobenius norm.

Then, we start to tune parameters `k` which is the number of latent factors and `normalize` which is the method for normalization of our rating matrix. We find that `normalize = NULL` provide us bad RMSEs (larger than 1.2), so `normalize = "center" or "Z-score"` as tried. Here are the results:

Table 1: RMSE (normalize = 'center', RunTime: mins)

| k | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| RMSE | 0.919190 | 0.902014 | 0.890724 | 0.885933 | 0.885054 | 0.883663 | 0.8820901 | 0.882772 | 0.881557 | 0.879274 |
| RunTime | 7.55478 | 10.57286 | 14.61774 | 19.30369 | 23.01872 | 27.69215 | 31.11930 | 37.50001 | 40.20423 | 46.85852 |

Table 2: RMSE (normalize = 'Z-score', RunTime: mins)

| k | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| RMSE | 0.918927 | 0.902006 | 0.891422 | 0.886963 | 0.886672 | 0.885382 | 0.884094 | 0.882666 | 0.881574 | 0.882191 |
| RunTime | 27.32548 | 26.92438 | 31.28472 | 32.53115 | 37.23498 | 42.32669 | 40.19847 | 40.67366 | 46.32476 | 44.35590 |

Therefore, we choose `k`=7, normalize = "center"`holds the small RMSE with short running time as paramters into`Recommender()'. There are three steps for modeling and predicting:

Step 1: Build SVDF recommender. By calling `Rcommender()`, we input utility matrix and parameters with `method = 'SVDF'`, getting a SVDF recommender.

Step 2: Predict `Ratings`. By calling `predict()`, we input SVDF recommender and utility matrix to full fill the user-movie matrix with predited rating results, obtaining the full result. Then, we convert the result matrix to list in order to output the test result by format with `UserID`, `MovieID`, 'Rating' variables. Each element are ratings of that user.

Step 3: Input predicted `Ratings` into user-movie paired test data, dealing with missing values. By conducting loop for each row of test data, we choose the corresponding rating from our prediction list. It is possible some movies in `movies.dat` or users in `users.dat` do not appear in the training, but in test, so we still have missing values after running svdf and replace missing value with value 3 as mentioned.

Then, we do comparison between our prediction rating and original rating, obtaining the RMSE values.

## Model 2: UBCF

We use User-based Collaborative Filtering method to build the model. There are three steps to build model and make prediction:

Step 1: build UBCF recommender. There are several parameters in this model. The parameter `normalize` sets how to normalize real ratings provided by different users. As mentioned above in the SVDF model, `normalize = "center"` performs best among these three options, so we use `normalize = "center"` and each user's vector of ratings is subtracted by its own mean to center the mean at zero. The parameter `method` sets the type of similarity metric to calculate similarity between users real ratings profile. We choose to use Cosine similarity, which is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. $similarity = cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$ The parameter `nn` sets the neighborhood of most similar users to consider for each user profile and the neighbors' ratings will be the basis for making predictions. After tuning, we find `nn = 25` outperform among these values, whose running time is the second shortest and performance seems good enough. Thus for each user we identify the 25 most similar users.

Table 3: RMSE, RunTime (mins)

| nn | 10 | 15 | 20 | 25 | 30 | 35 |
|---|---|---|---|---|---|---|
| RMSE | 1.035488 | 1.035184 | 1.035057 | 1.035013 | 1.034952 | 1.034903 |
| RunTime | 20.97347 | 21.57644 | 21.51089 | 21.34166 | 21.56512 | 21.54354 |

And then use these parameters into the model by function `Recommender()`.

Step 2: predict ratings. By using function `predict()`, we input the recommender model and utility matrix to obtain the user-movie matrix with predicted ratings and then convert the matrix into list to get the output with variables `UserID`, `MovieID` and `Rating`. Each element are average of the ratings by each user's 25 most similar users and we weight the average ratings based on similarity score of each user whose rated the item. Similarity score equals weight.

Step 3: Input predicted `Rating` into user-movie paired test data, deal with missing value. By reading `userid` and `movieid` from all lines of test data, we assign predicted ratings to corresponding user-movie pairs. It is possible some movies in movies.dat or users in users.dat do not appear in the training, but in test, so we still have missing value after running UBCF. We assign 3 to the missing value as rating ranges from 1 to 5. Finally, we compare the prediction results with `Rating` in test data by calculatinf RMSE.

## Results

### Evalution on Prediction Accuracy

RMSE:

```
print(SVDF.result)
```

```
## [1] 0.8820901
```

```
print(UBCF.result)
```

```
## [1] 1.035013
```

In general, recommender based on Funk SVD with gradient descend has better prediction performance than recommender based on user-based collaborative filtering. However, the maximum rating of original data is 5 and the minimum is 1. Compared with the scale of orignial rating data, RMSEs for both two method seems a little large and prediction seems a little inaccurate.

### Evalution on Efficiency

Table 4: Running Time (Unit: Mins)

| model | SVDF | UBCF |
|---|---:|---:|
| runningtime | 31.11930 | 21.34166 |

My computer system is iMac which has 3 GHz Intel Core i5 Processor, 16 GB 2400 MHz DDR4 Memory. After comparing running time of these two models, we find SVDF has longer running time than UBCF. But the prediction performance of SVDF is much better than that of UBCF considering the range of ratings. In spite of the longer running time, we still think SVDF method is more efficient.

### What's also interesting

By comparing the RMSE from SVDF and UBCF, we find that SVDF has obvious better accuracy than UBCF. Refering to the algorithm of both models, we find that SVDF not only considers the Collaborative Filtering or Latent Factor model on the interactin term, but also takes overall, user, movie effect into consideration, resulting in better accuracy.

## Conclusion

For SVDF, there are more parameters that need to be tuned manually. For example, we have to tune the value of k in SVDF method many times, which is time-consuming. Besides that, it has longer running time but much better prediction performance. For UBCF, there are less parameter options to choose. But comparing the range of rating from original data and RMSE, the improvement of accuray is more important and it's reasonable to sacrifice running time to obtain a better recommendation. In our case, it's more suitable to use SVDF model