

BA Practicum

NETFLIX PRIZE PROJECT

YU-MIN WANG | BUAN 6390 | Aug. 4, 2017

CONTENTS

[Data Description](#)

[Project Goal](#)

[Implementation Procedure](#)

[Challenges and Solutions](#)

[Codes and Results](#)

[Findings](#)

[Appendix](#)

Data Description

For training, the project offers 17770 text files which represent user's ratings for each movie from ID 01 to ID 17770. Below is the file format of mv_0000001 which contains user's ratings on movie 01.

Each file contains 4 kinds of information :

1. Movie ID: indicated by "1:" in the first row
2. User ID : "1488844" in the 2nd row
3. Rating : "3" in the 2nd row
4. Date : "2005-09-06" in the 2nd row

Name	Date modified	Type	
mv_0000001	9/15/2006 7:16 PM	Text Document	1: 1488844,3,2005-09-06 822109,5,2005-05-13 885013,4,2005-10-19 30878,4,2005-12-26 823519,3,2004-05-03 893988,3,2005-11-17 124105,4,2004-08-05 1248029,3,2004-04-22 1842128,4,2004-05-09 2238063,3,2005-05-11 1503895,4,2005-05-19 2207774,5,2005-06-06 2590061,3,2004-08-12 2442,3,2004-04-14 543865,4,2004-05-28 1209119,4,2004-03-23 804919,4,2004-06-10 1086807,3,2004-12-28 1711859,4,2005-05-08 372233,5,2005-11-23 1080361,3,2005-03-28 1245640,3,2005-12-19 558634,4,2004-12-14 2165002,4,2004-04-06 1181550,3,2004-02-01 1227322,4,2004-02-06 427928,4,2004-02-26 814701,5,2005-09-29 808731,4,2005-10-31 662870,5,2005-08-24 337541,5,2005-03-23 786312,3,2004-11-16 1133214,4,2004-03-07 1537427,4,2004-03-29 1209954,5,2005-05-09 2381599,3,2005-09-12 525356,2,2004-07-11 1910569,4,2004-04-12 2263586,4,2004-08-20
mv_0000002	9/15/2006 7:16 PM	Text Document	
mv_0000003	9/15/2006 7:16 PM	Text Document	
mv_0000004	9/15/2006 7:16 PM	Text Document	
mv_0000005	9/15/2006 7:16 PM	Text Document	
mv_0000006	9/15/2006 7:16 PM	Text Document	
mv_0000007	9/15/2006 7:16 PM	Text Document	
mv_0000008	9/15/2006 7:16 PM	Text Document	
mv_0000009	9/15/2006 7:16 PM	Text Document	
mv_0000010	9/15/2006 7:16 PM	Text Document	
mv_0000011	9/15/2006 7:16 PM	Text Document	
mv_0000012	9/15/2006 7:16 PM	Text Document	
mv_0000013	9/15/2006 7:16 PM	Text Document	
mv_0000014	9/15/2006 7:15 PM	Text Document	
mv_0000015	9/15/2006 7:16 PM	Text Document	
mv_0000016	9/15/2006 7:16 PM	Text Document	
mv_0000017	9/15/2006 7:16 PM	Text Document	
mv_0000018	9/15/2006 7:16 PM	Text Document	
mv_0000019	9/15/2006 7:16 PM	Text Document	
mv_0000020	9/15/2006 7:16 PM	Text Document	
mv_0000021	9/15/2006 7:16 PM	Text Document	
mv_0000022	9/15/2006 7:16 PM	Text Document	
mv_0000023	9/15/2006 7:16 PM	Text Document	
mv_0000024	9/15/2006 7:16 PM	Text Document	

Be noted that user IDs are not continuous integers. Unlike movie ID which is continuous integer from 01 to 17770, user IDs have gap between IDs because not every user made ratings.

Ratings are integers from 1 to 5.

For probe, the project offers a file “probe.txt” which is 10GB size with format like below. It contains 1.4 million lines in which “1:” means movieID 1 and the line with no colon is user ID.

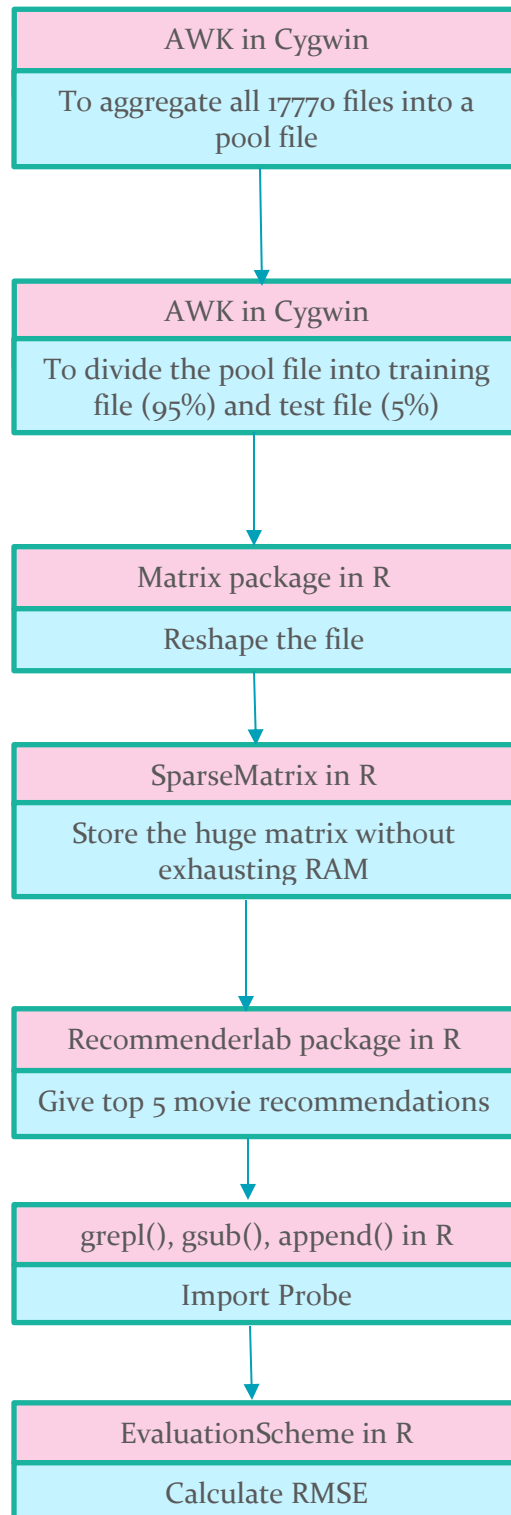
Name	Date modified	Type	Size	
				1:
				30878
				2647871
				1283744
				2488120
				317050
				1904905
				1989766
				14756
				1027056
				1149588
				1394012
				1406595
				2529547
				1682104
				2625019
				2603381
				1774623
				470861
				712610
				1772839
				1059319
				2380848
				548064
				10:
				1952305
				1531863
				1000:
				2326571
				977808
				1010534
				1861759
				79755
				98259
				1960212
				97460
				2623506
				2409123

The input methods for probe file and for training file are different.

Project Goal

Develop a recommendation model and make prediction based on these 17770 files.

Implementation Procedure



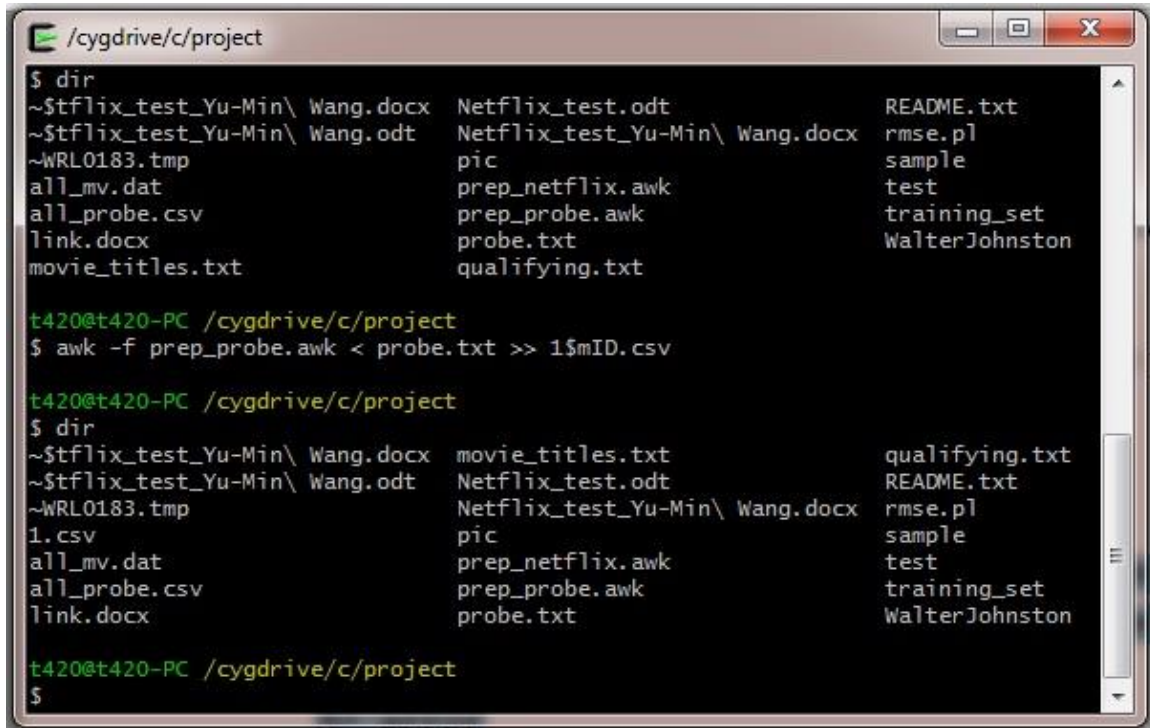
Challenges and Solutions

	CHALLENGES	SOLUTIONS
1	How to aggregate the 17770 files into a file in the format that R can import ?	Install Cygwin, run UNIX command, write a AWK script file as the input to UNIX command
2	How to reshape the aggregate file format from the original 4 columns by 100 million observations to a matrix with userID as rows and movieID as columns?	Two ways : one is to use library “reshape2” and function acast() to do reshape, but this method will overload RAM. The other way is to create a sparseMatrix
3	How to store such a huge matrix with 480114 rows of active users by 17770 columns of movies without exhausting RAM space?	Use library “Matrix” and function sparseMatrix() which will not physically store NA values
4	The maximum userID of the active users is 2649429 while the number of active users is 480189. Can we build only the active users in the rows while we can still trace the userID?	With sparseMatrix(), the used memory size of 2649429 rows is the same as 480189 rows because non rating users occupy zero space. Use the dimnames() function to assign user IDs for rownames then we can trace the userID.
5	How to make the memory usage as efficiently as possible?	Remove some useless matrix by rm(), and use gc() to do garbage collection. The following functions help know the usage of memory : memory.limit(), memory.size(), object.size()
6	How to import probe file?	Divide the 1.4 million records into 3 parts, and import the 3 parts separately in R
7	How to make a top 5 movie recommendations to each active user?	Use library “recommenderlab”, transform the matrix to realRatingMatrix, do function Recommender(), then use predict by indicating the user ID with topN =5
8	How to predict ratings and evaluate RMSE of each method?	Use function evaluationScheme(), then predict, then calcPredictionAccuracy() to calculate the RMSE

Codes and Results

1. AWK and UNIX

Install Cygwin in Windows. Cygwin is a platform on which we can run UNIX commands.



```
/cygdrive/c/project
$ dir
~$tflix_test_Yu-Min\ Wang.docx  Netflix_test.odt      README.txt
~$tflix_test_Yu-Min\ Wang.odt   Netflix_test_Yu-Min\ Wang.docx rmse.pl
~WRL0183.tmp                   pic                   sample
all_mv.dat                    prep_netflix.awk      test
all_probe.csv                 prep_probe.awk        training_set
link.docx                     probe.txt              WalterJohnston
movie_titles.txt               qualifying.txt

t420@t420-PC /cygdrive/c/project
$ awk -f prep_probe.awk < probe.txt >> 1$mID.csv

t420@t420-PC /cygdrive/c/project
$ dir
~$tflix_test_Yu-Min\ Wang.docx  movie_titles.txt      qualifying.txt
~$tflix_test_Yu-Min\ Wang.odt   Netflix_test.odt      README.txt
~WRL0183.tmp                   Netflix_test_Yu-Min\ Wang.docx rmse.pl
1.csv                          pic                   sample
all_mv.dat                    prep_netflix.awk      test
all_probe.csv                 prep_probe.awk        training_set
link.docx                     probe.txt              WalterJohnston

t420@t420-PC /cygdrive/c/project
$
```

Write a AWK script file called `prep_netflix.awk`. This is a text file with the following content:

```
BEGIN { FS = ":" }
{ if (FNR == 1) mID = $1
  if (FNR != 1) print mID "," $0 }
```

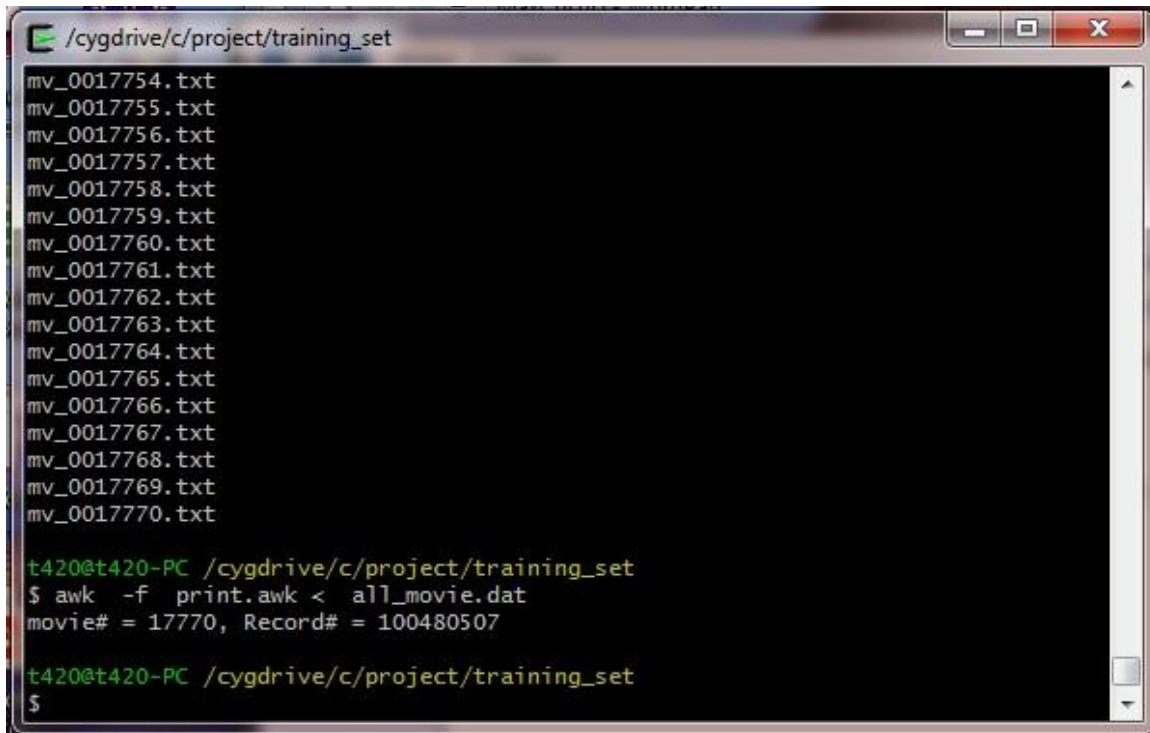
Where FS means file separator, FNR means the current count of the number of input records in the current input file. This script says for the first record, like 1;, just set variable `movieID`; for the other records, like 1488844, 3, 2005-09-06, insert the `movieID` in the first column following by a comma and the whole record.

Then execute the following UNIX commands in Cygwin:

```
for j in mv*.txt; do echo $j; awk -f ../prep_netflix.awk < $j >>all_movie.dat;
done
```

These commands will sequentially read `mv_0000001.txt` to `mv_0017770.txt` and append the output to an aggregate file named `all_movie.dat` which is actually a CSV file contains 4 columns : `movieID`, `userID`, `Rate`, and `Date` with comma as separator.

The aggregate file occupies 2.5GByte. I wrote a AWK to run the file, knowing the aggregate file has about 100 million records.



```
/cygdrive/c/project/training_set
mv_0017754.txt
mv_0017755.txt
mv_0017756.txt
mv_0017757.txt
mv_0017758.txt
mv_0017759.txt
mv_0017760.txt
mv_0017761.txt
mv_0017762.txt
mv_0017763.txt
mv_0017764.txt
mv_0017765.txt
mv_0017766.txt
mv_0017767.txt
mv_0017768.txt
mv_0017769.txt
mv_0017770.txt

t420@t420-PC /cygdrive/c/project/training_set
$ awk -f print.awk < all_movie.dat
movie# = 17770, Record# = 100480507

t420@t420-PC /cygdrive/c/project/training_set
$
```

2. Divided into Train and Test

We can use AWK to divide the aggregate file into train and test, or we can use R instruction like `sample()` to do so, or we can use `evaluationScheme()` to indicate the train percentage, assuming train is 95% and test is 5%.

AWK script of train:

```
BEGIN { M = 18
        N = 20 }
{ r = (FNR - M) % N
  if ( r != 0 ) print $0 }
```

AWK script of test:

```
BEGIN { M = 18
        N = 20 }
{ r = (FNR - M) % N
  if ( r == 0 ) print $0 }
```

UNIX command:

```
awk -f test.awk < all_movie.dat >> test_movie.csv; done
```

```
awk -f train.awk < all_movie.dat >> train_movie.csv; done
```


R command of evaluationScheme():

```
> e <- evaluationScheme(r, method="split", train=0.95, k=1, given=-1)
```

3. Reshape the Aggregate File

To use recommenderlab library, we must transform the aggregate file into the format of ratingMatrix which looks like this:

	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8
u_1	?	4.0	4.0	2.0	1.0	2.0	?	?
u_2	3.0	?	?	?	5.0	1.0	?	?
u_3	3.0	?	?	3.0	2.0	2.0	?	3.0
u_4	4.0	?	?	2.0	1.0	1.0	2.0	4.0
u_5	1.0	1.0	?	?	?	?	?	1.0
u_6	?	1.0	?	?	1.0	1.0	?	1.0

The row is represented by u (user), column is represented by i (item, or movie). This form does not need date so we will delete the column of Date.

Initially I used the following R code to reshape the aggregate file into a ratingMatrix, however, the consequence of this huge matrix overloaded my 16GB RAM with an error.

```
> library("reshape2")

> g<-acast(mvTrain, CustID ~ mvID, value.var="Rate")

##Error: cannot allocate vector of size 63.6 Gb
```

Then I tried sparseMatrix function as below:

```
> library("Matrix")

> N <- sparseMatrix(i=mvTrain[[2]],j=mvTrain[[1]],x=mvTrain[[3]])
```

This successfully creates a matrix N and the size of N is only 1.3GB, with 2646429 rows (most of them are inactive users with no ratings) by 17770 columns, but only 100 million non NA.

```
> object.size(N)

1355202080 bytes

> nrow(N)
```



```
[1] 2649429

> ncol(N)

[1] 17770

> nnzero(N)

[1] 100480507
```

A sparse matrix is a matrix with more than 50% of fields as NA (missing value). I found the matrix N has 99% of fields as NA.

The function `sparseMatrix()` only stores non-NA values in memory, that means NA occupies zero space in the memory. Because of this, the non-active users will occupy zero space in memory because they make no ratings.

Because later we will delete the inactive users while keeping the index of user ID, we use the below instruction to give `rownames=CustID`, and `colnames=movieID`, in character.

```
> dimnames(N) <- list(user=as.character(c(1: nrow(N))),
movie=as.character(c(1:ncol(N))))
```

4.Delete the Inactive Users

In recommenderlab, there is an easy way to delete the inactive users. With `rowCounts()` function, we can delete the users with `rowCounts()=0`.

```
> library("recommenderlab")

> r<-as(N,"realRatingMatrix")

> r<-r[rowCounts(r)>0]    ## only keep the users which have ratings
```

Then we find the number of active users are 480189, only 18% of 2649429, the maximum ID of user.

```
> nrow(r)

[1] 480189
```

Although r has 480189 rows and N has 2649429 rows, they occupy almost the same memory size:

```
> object.size(N)

1355202080 bytes

> object.size(r)

1233725424 bytes
```

5. Get the Recommender Model

The function `Recommender` will give recommendation models with different methods. Here we use the methods of `POPULAR` and `UBCF` (User Based Collaborative Filtering).

```
> recP <- Recommender(r, method="POPULAR")
> recU <- Recommender(r, method="UBCF")
> names(getModel(recP))
[1] "topN"                "ratings"              "normalize"
[4] "aggregationRatings"  "aggregationPopularity" "verbose"
> names(getModel(recU))
[1] "description" "data"          "method"        "nn"            "sample"
[6] "normalize"    "verbose"
```

6. Give Top 5 Movie Recommendations

If we want to give a top 5 movie recommendations to a certain active user, e.g., `userID= 7` which is an active user, we can use the following command:

```
> recomP <- predict(recP, r["7"], n=5)
> as(recomP, "list")
$`7`
[1] "14621" "6974"  "14103" "7193"  "11812"
```

This tells for `userID=7`, the top 5 recommendations are movies with ID 14621, 6974, 14103, 7193, and 11812.

Be noted we must indicate `r["7"]` instead of `r[7]`. The former indicates the row name = "7" (as `dimnames()` has done, row name = "userID"), while the latter indicates the 7th active user.

7. Give Predictions

We can predict `userID="6"` by the following code. Please be noted that the `predict` function only predicts for unknown data, the prediction for known data is NA.

```
predP <- predict(recP, t["6"], type="ratings")
as(predP, "matrix")
```

8. Import Probe

There is no suitable AWK script to identify the lines with colon and without colon, so we use R commands like `gsub()`, `grepl()`, and `append()` to import the probe file. The challenge is the import can not finish after spending a whole night. So I divide the 1.4 million lines into 3 parts, and import the 3 parts separately. The import of each part spends about 20 minutes. Then appending the 3 parts into a vector.

```
dat = readLines("probe.txt")
> str(dat)
chr [1:1425333] "1:" "30878" "2647871" "1283744" "2488120" ...
mID<- 1
mvID1<-c()
CustID1<-c()
## import part 1
for (str in dat[1:499999]) if(grepl(":",str)) {mID=
as.numeric(gsub("\\:", "", str))} else {mvID1<-append(mvID1,
mID); CustID1 <- append(CustID1, as.numeric(str))}
## import part 2
mvID2<-c()
CustID2<-c()
for (str in dat[500000:999999]) if(grepl(":",str)) {mID=
as.numeric(gsub("\\:", "", str))} else {mvID2<-append(mvID2,
mID); CustID2 <- append(CustID2, as.numeric(str))}
## import part 3
mvID3<-c()
CustID3<-c()
for (str in dat[1000000:length(dat)]) if(grepl(":",str)) {mID=
as.numeric(gsub("\\:", "", str))} else {mvID3 <-append(mvID3,
mID); CustID3 <- append(CustID3, as.numeric(str))}

## merge the 3 parts to get the vectors CustID and mvID
mvID<-c()
CustID<-c()
mvID<-append(mvID1, mvID2)
mvID<-append(mvID, mvID3)
CustID<-append(CustID1, CustID2)
CustID<-append(CustID, CustID3)
```

9. Calculate RMSE for Rating Predictions

We can use `evaluationScheme()`, `Recommender()`, `predict()`, `calcPredictionAccuracy()` to do rating predictions and get the RMSE. Below is an example of method POPULAR:

```
> e <- evaluationScheme(r[rowCounts(r)>1000,], method="split",
train=0.95,k=1, given=-1)

> e

Evaluation scheme using all-but-1 items

Method: 'split' with 1 run(s).

Training set proportion: 0.950

Good ratings: NA

Data set: 11421 x 17770 rating matrix of class 'realRatingMatrix' with
16170843 ratings.

> r1 <- Recommender(getData(e, "train"), "POPULAR")

> p1 <- predict(r1, getData(e, "known"), type="ratings")

> error1 <- calcPredictionAccuracy(p1, getData(e, "unknown"))

> error1

      RMSE      MSE      MAE
0.8741432 0.7641263 0.6914844
```

By another method UBCF, we got a better RMSE:

```
> r2 <- Recommender(getData(e, "train"), "UBCF")

> p2 <- predict(r2, getData(e, "known"), type="ratings")

> error2 <- calcPredictionAccuracy(p2, getData(e, "unknown"))

> error2

      RMSE      MSE      MAE
0.8602512 0.7400322 0.6720614
```

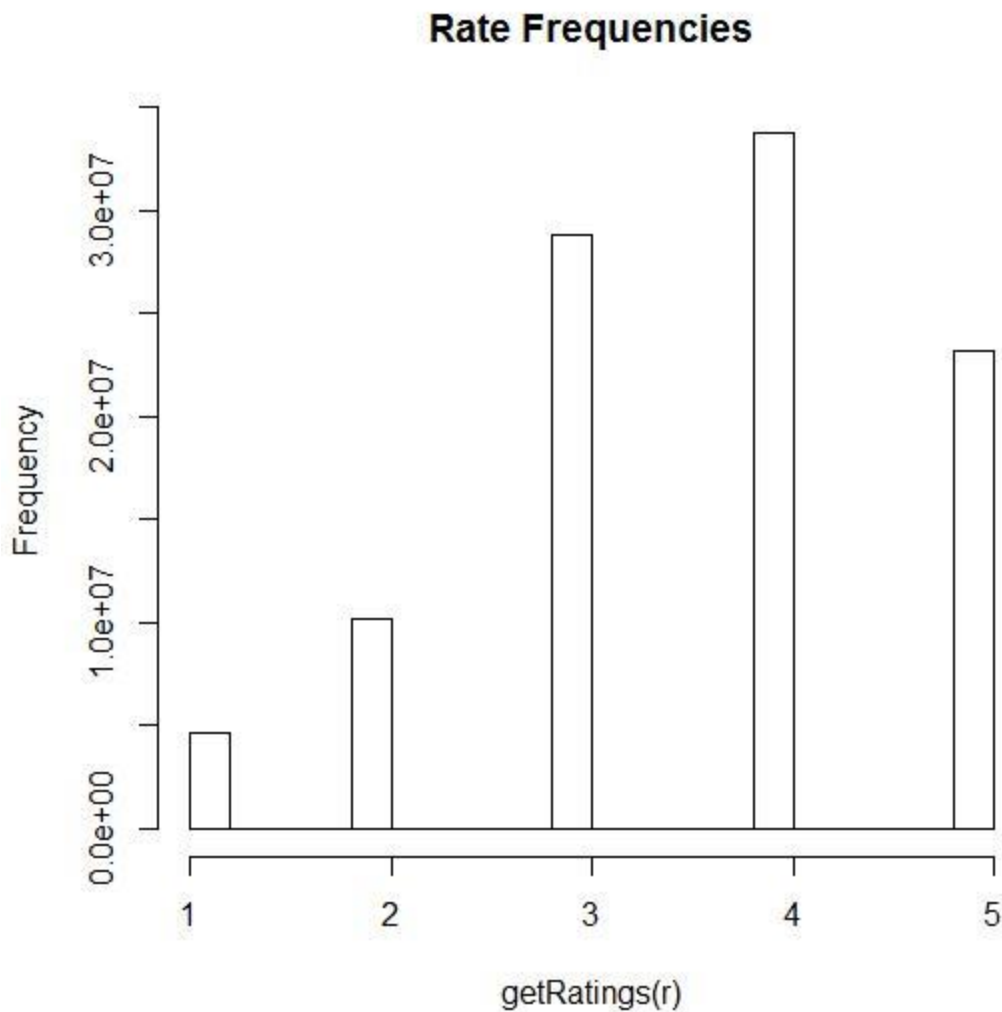
It seems UBCF is a little better than POPULAR because UBCF has lower RMSE.

Findings

1) Rate “4” happens the most often, “1” the least

Of all the 100 million ratings, we find that “4” appears about 35 million times, “3” happens 30 times, and “1” only 5 million times.

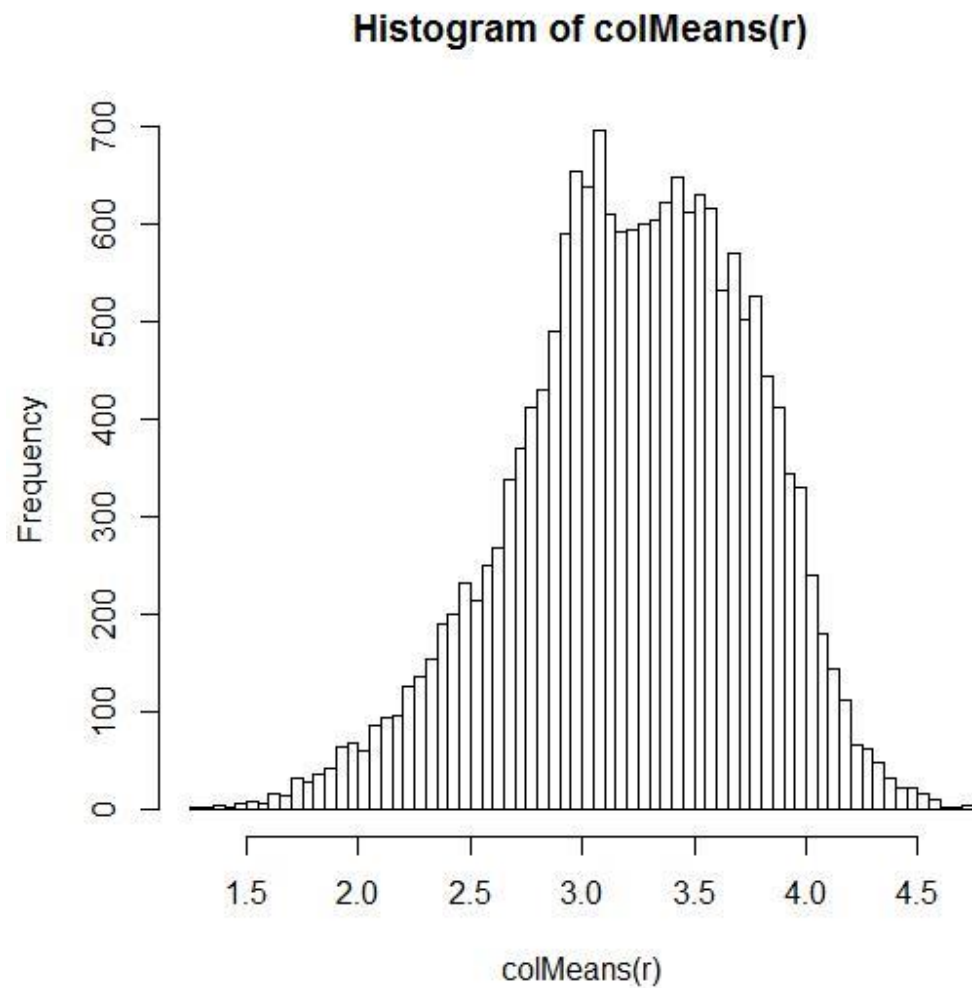
```
> hist(getRatings(r), breaks=20, main="Rate Frequencies")
```



2) The mode of average movie rating is about 3.0

Of all the 17770 movies, the distribution of average ratings for all movies is shown as below histogram. We find the mode is around 3.0.

```
> hist(colMeans(r), breaks=50)
```



3) The movie with the highest rating is ...

“2003, Lord of the Rings : The Return of the King” with average rating 4.72327

```
> M <- (colMeans(r))
```

```
> m=max(M)
```

```
> m
```

```
[1] 4.72327
```

```
> which(M==m)
```

```
14961
```

```
14961,2003, Lord of the Rings: The Return of the King: Extended Edition
```

4) There are 5 users rating more than 10000 movies, but 1269 users rating 1 movie

```
> length(count[count>10000])
```

```
[1] 5
```

```
> length(count[count==1])
```

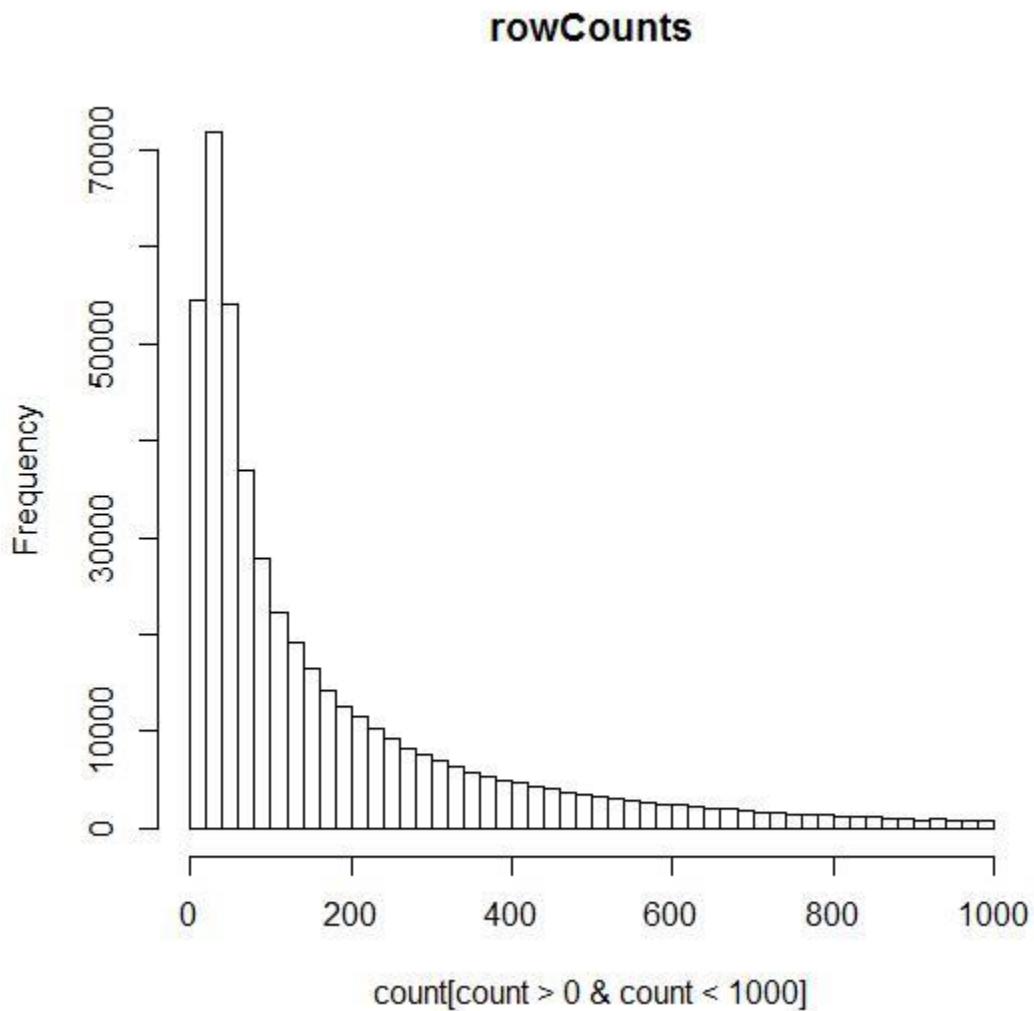
```
[1] 1269
```

5) 97% of active users rate less than 1000 movies, most users rate 21-40 movies

```
> length(count[count<1000]) / length(count)
```

```
[1] 0.9726337
```

```
> hist(count[count>0 & count<1000], breaks=50, main="rowCounts ")
```



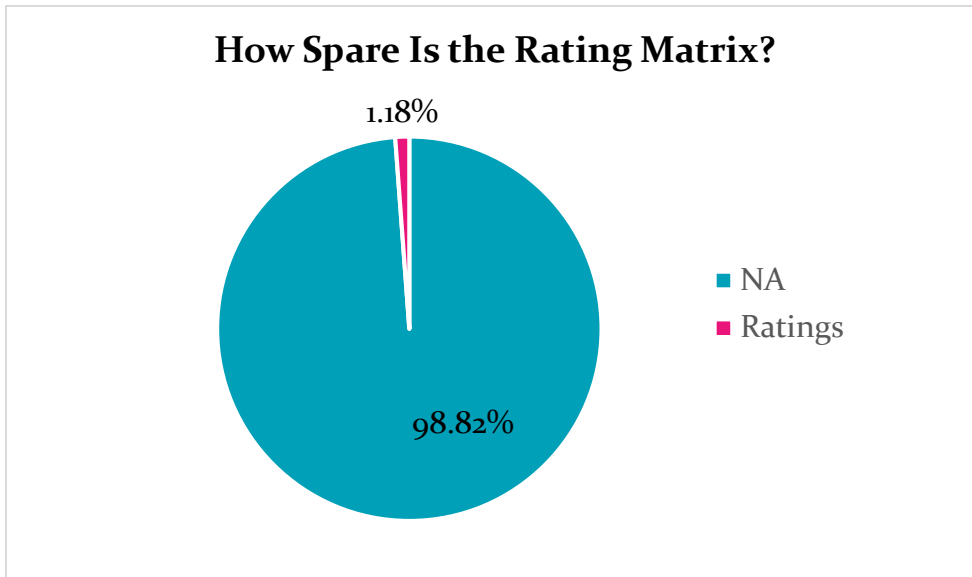
6) Almost 99% of the fields in the rating matrix is NA

The rating matrix has 480189 rows (of active users) and 17770 columns (of movies), which creates a huge matrix with $480189 \times 17770 = 8.53$ billion fields. However, there are only 100 million ratings. So the sparseness of the rating matrix is $100480507 / (480189 \times 17770) = 1.18\%$. Almost 99% of the fields in the matrix is missing value.

```
> ncol(r)
[1] 17770

> nrow(r)
[1] 480189

> nnzero(N)
[1] 100480507
```



Appendix

```
myDir<-"C:/project/training_set"
setwd(myDir)
myData<-"all_movie.dat"
library(data.table)
# read 100480507 rows and 4 columns from 2.43 GB file in 0:04:08
mvTrain <-fread(myData, col.names=c("mvID", "CustID", "Rate", "Date"))
# Cut Date
mvTrain <-mvTrain[,-c(4)]
# Use sparseMatrix to transform into rating matrix and save RAM
library("Matrix")
N <- sparseMatrix(i=mvTrain[[2]],j=mvTrain[[1]],x=mvTrain[[3]])
# Give rowname=userID and columnname=movieID, both in character
dimnames(N) <- list(user=as.character(c(1:nrow(N))),
movie=as.character(c(1:ncol(N))))
# import library recommenderlab and transform N into rating matrix r
library("recommenderlab")
r<-as(N,"realRatingMatrix")
# delete inactive users
r<-r[rowCounts(r)>0]
count<-rowCounts(r)
# draw histograms
hist(getRatings(r), breaks=20, main="Rate Frequencies")
hist(colMeans(r), breaks=50)
hist(count[count>0 & count<1000], breaks=50, main="rowCounts ")
summary(colMeans(r))
# compare object size of N and r
object.size(N)
object.size(r)
## make recommender models
recP <- Recommender(r, method="POPULAR")
recU <- Recommender(r, method="UBCF")
recALS <- Recommender(r, method="ALS")
recRAN <- Recommender(r, method="RANDOM")
```

```

### import test Data
testData<-"test_movie.csv"
mvTest <-fread(testData, col.names=c("mvID", "CustID", "Rate", "Date"))
mvTest <-mvTest[,-c(4)]
T <- sparseMatrix(i=mvTest[[2]],j=mvTest[[1]],x=mvTest[[3]])
dimnames(T) <- list(user=as.character(c(1: nrow(T))),
movie=as.character(c(1:ncol(T))))
t<-as(T,"realRatingMatrix")
t<-t[rowCounts(t)>0]
## Give top 5 recommendations to users with ID 6,7,8,10,25,33
recomP <- predict(recP, r[c("6","7","8","10","25","33")], n=5)
as(recomP, "list")
## Predict ratings for userID="6" in test sample t. NA for known data.
predP <-predict(recP, t["6"], type="ratings")
as(predP, "matrix")

### import Probe
dat = readLines("probe.txt")

mID<- 1
mvID1<-c()
CustID1<-c()

## import part 1
for (str in dat[1:499999]) if(grepl(":",str)) {mID=
as.numeric(gsub("\\:","", str))} else {mvID1<-append(mvID1, mID);
CustID1 <- append(CustID1, as.numeric(str))}

## import part 2
mvID2<-c()
CustID2<-c()

for (str in dat[500000:999999]) if(grepl(":",str)) {mID=
as.numeric(gsub("\\:","", str))} else {mvID2<-append(mvID2, mID);
CustID2 <- append(CustID2, as.numeric(str))}

## import part 3
mvID3<-c()
CustID3<-c()

```

```

for (str in dat[1000000:length(dat)]) if(grepl(":",str)) {mID=
as.numeric(gsub("\\:", "", str))} else {mvID3 <-append(mvID3, mID);
CustID3 <- append(CustID3, as.numeric(str))}

## merge the 3 parts to get the vectors CustID and mvID

mvID<-c()

CustID<-c()

mvID<-append(mvID1, mvID2)

mvID<-append(mvID, mvID3)

CustID<-append(CustID1, CustID2)

CustID<-append(CustID, CustID3)


## Use evaluationScheme to calculate RMSE of predictions. Compare
## POPULAR and UBCF. We only run the observations with rowCounts>1000

e <- evaluationScheme(r[rowCounts(r)>1000,], method="split",
train=0.95,k=1, given=-1)

e

r1 <- Recommender(getData(e, "train"), "POPULAR")

p1 <- predict(r1, getData(e, "known"), type="ratings")

error1 <- calcPredictionAccuracy(p1, getData(e, "unknown"))

error1

p2 <- predict(r2, getData(e, "known"), type="ratings")

error2 <- calcPredictionAccuracy(p2, getData(e, "unknown"))

error2

```