myDir<-"C:/project/training\_set"

## myDir<-"C:/Users/yxw166230/Desktop/training\_set"

setwd(myDir)

#rm(list=ls())

#load("project.Rimage")

# myData<-"train\_movie.csv"

myData<-"all\_movie.dat"

library(data.table)

mvTrain <-fread(myData, col.names=c("mvID", "CustID", "Rate", "Date"))

# read 100480507 rows and 4 columns from 2.43 GB file in 0:04:08#mvTrain <-fread(myData, header = FALSE)

# Cut Date

mvTrain <-mvTrain[,-c(4)]

save.image("project.Rimage”)

# using acast in reshape2

library("reshape2")

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

##Error: cannot allocate vector of size 63.6 Gb

> object.size(mvTrain)

2009733896 bytes

library("Matrix")

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

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

> object.size(N)

1355202080 bytes

> nrow(mvTrain)

[1] 100480507

> nnzero(N)

[1] 100480507

> nrow(N)

[1] 2649429

> ncol(N)

[1] 17770

nnzero(N)/(17770\*2649429)

[1] 0.002134234

480189/2649429

[1] 0.1812424

nnzero(N)/(17770\*480189)

[1] 0.01177558

z<-rowSums(N)

nr <- nrow(N)

v <- c()

for (i in 1:nr) { if (z[i]==0) v<-append(v,i)}

N <- N[-v,]

library("recommenderlab")

r<-as(N,"realRatingMatrix")

## R <- r[rowCounts(r)!=0]

getRatingMatrix(r)

# delete inactive users

r<-r[rowCounts(r)>0]

## t[“59”] = t[7]

> recomP <- predict(recP, t["59"], n=5)

> as(recomP, "list")

$`59`

[1] "14550" "11521" "14240" "16377" "11283"

> recomP <- predict(recP, t[7], n=5)

> as(recomP, "list")

$`59`

[1] "14550" "11521" "14240" "16377" "11283"

> recomP <- predict(recP, t["2"], n=5)

Error in intI(i, n = d[1], dn[[1]], give.dn = FALSE) :

invalid character indexing

rnorm <- normalize(r)

image(R, main=”Raw Ratings”)

image(rnorm, main=”Normalized Ratings”)

##hist(colMeans(r), breaks=50)

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

hist(colMeans(r), breaks=50)

summary(colMeans(r))

Min. 1st Qu. Median Mean 3rd Qu. Max.

1.288 2.897 3.255 3.228 3.616 4.723

M <-(colMeans(r))

m <- max(M)

> m

[1] 4.72327

> which(M==m)

14961

14961

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

max(rowCounts(r))

[1] 16752

which(rowCounts(r)==16752)

305344

count<-rowCounts(r)

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

length(count[count>0 & count<1000]) / length(count[count>0])

[1] 0.9726337

> length(count[count>0 & count>500])

[1] 50107

> length(count[count>0 & count>1000])

[1] 11421

> length(count[count>0 & count>5000])

[1] 43

> length(count[count>0 & count>10000])

[1] 5

> length(count)

[1] 2649429

> length(count[count>0])

[1] 480189

hist(count[count>0 & count>5000], breaks=50)

tail(c1)

1461435 2118461 1664010 2439493 387418 305344

9332 14092 14993 15747 16578 16752

gc() # garbage collection

## evaluation of RMSE among different methods of recommendation

# delete inactive users

r<-r[rowCounts(r)>0]

> nrow(r)

[1] 480189

> object.size(N)

1355202080 bytes

> object.size(r)

1233725424 bytes

rownames(r[1,])

“6”

> r[rownames(r)==6,]

1 x 17770 rating matrix of class ‘realRatingMatrix’ with 597 ratings.

> rowCounts(r[1,])

6

597

> rowCounts(r[rownames(r)==6,])

6

597

> length(count[count==1])

[1] 1400

> length(count[count==2])

[1] 2309

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

e <- evaluationScheme(r, method="split", train=0.95, given=0)

e <- evaluationScheme(r[rowCounts(r) >10,], method="split", train=1, given=5)

e <- evaluationScheme(r[1:500,], method="split", train=0.9,k=1, given=15)

data("MSWeb")

MSWeb10 <- sample(MSWeb[rowCounts(MSWeb) >10,], 100)

rec <- Recommender(MSWeb10, method = "POPULAR")

# run rowCounts>1000

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")

> r1

Recommender of type ‘POPULAR’ for ‘realRatingMatrix’

learned using 10849 users.

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

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

> error

RMSE MSE MAE

0.8741432 0.7641263 0.6914844

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

r2

#Recommender of type ‘UBCF’ for ‘realRatingMatrix’

#learned using 10849 users.

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

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

error

# RMSE MSE MAE

#0.8602512 0.7400322 0.6720614

## 5000 OK

> e <- evaluationScheme(r[1:5000,], 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: 5000 x 17770 rating matrix of class ‘realRatingMatrix’ with 1013816 ratings.

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

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

> r1

Recommender of type ‘POPULAR’ for ‘realRatingMatrix’

learned using 4750 users.

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

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

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

> error

RMSE MSE MAE

0.9696380 0.9401978 0.7643784

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

> error

RMSE MSE MAE

1.008576 1.017226 0.799719

## try 10000 ok

e <- evaluationScheme(r[1:10000,], method="split", train=0.95,k=1, given=-1)

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

r1

#Recommender of type ‘POPULAR’ for ‘realRatingMatrix’

#learned using 9500 users.

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

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

error

# RMSE MSE MAE

#0.9646331 0.9305170 0.7710089

## try 20000 ok, but worse

e <- evaluationScheme(r[1:20000,], method="split", train=0.95,k=1, given=-1)

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

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

error

# RMSE MSE MAE

#0.9867694 0.9737138 0.7756189

> object.size(e)

100920080 bytes

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

r2 <- Recommender(getData(e, "train"), "POPULAR")

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

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

p2 <- predict(r2, t, type="ratings")

error <- rbind(UBCF = calcPredictionAccuracy(p1, getData(e, "uknown")), POPULAR = calcPredictionAccuracy(p2, getData(e, "uknown")))

error

RMSE(true, predicted, na.rm = TRUE)

recP <- Recommender(r, method="POPULAR")

recU <- Recommender(r, method="UBCF")

recALS <- Recommender(r, method="ALS")

recRAN <- Recommender(r, method="RANDOM")

> names(getModel(recP))

[1] "topN" "ratings" "normalize"

[4] "aggregationRatings" "aggregationPopularity" "verbose"

> names(getModel(recU))

[1] "description" "data" "method" "nn" "sample"

[6] "normalize" "verbose"

##Error in validObject(.Object) :

# invalid class “topNList” object: invalid object for slot "itemLabels" in class "topNList": got class # "NULL", should #be or extend class "character"

recS <- Recommender(r, method="SVD")

recI <- Recommender(r, method="IBCF")

recSVDF <- Recommender(r, method="SVDF")

# Error in asMethod(object) :

# Cholmod error 'problem too large' at file ../Core/cholmod\_dense.c, line 105

### 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]

### predict : top 5 recommendations for users with ID 6,7,8,10,25,33

head(rownames(r))

[1] "6" "7" "8" "10" "25" "33"

recomP <- predict(recP, r[c("6","7","8", "10","25","33")], n=5)

as(recomP, "list")

$`6`

[1] "8904" "14103" "7193" "4640" "16384"

$`7`

[1] "14621" "6974" "14103" "7193" "11812"

$`8`

[1] "14550" "11521" "14240" "2452" "11283"

$`10`

[1] "14550" "11521" "14240" "2452" "16954"

$`25`

[1] "14550" "11521" "14240" "2452" "16377"

$`33`

[1] "14550" "11521" "14240" "2452" "16377"

predP <-predict(recP, t, type="ratings")

##Error in subCsp\_ij(x, i, j, drop = drop) :

##Cholmod error 'problem too large' at file ../Core/cholmod\_sparse.c, line 92

as(predP, "matrix")[,1:10]

1 2 3 4 5 6 7 8

3.635393 3.505309 3.655344 2.638300 3.704698 3.063812 2.261109 3.049017

9 10

2.705629 3.301235

## predict NA for all known values, only unknowns have predictions

as(predP, "matrix")[,which(T["6",]!=0)]

30 501 1542 2965 4996 5762 5787 5924 6099 6134 6166 6475 6775

NA NA NA NA NA NA NA NA NA NA NA NA NA

7779 8292 8333 8682 8764 10123 10255 10773 12812 12870 14911 15234 16564

NA NA NA NA NA NA NA NA NA NA NA NA NA

16948 17088 17467

NA NA NA

> which(T["6",]!=0)

30 501 1542 2965 4996 5762 5787 5924 6099 6134 6166 6475 6775

30 501 1542 2965 4996 5762 5787 5924 6099 6134 6166 6475 6775

7779 8292 8333 8682 8764 10123 10255 10773 12812 12870 14911 15234 16564

7779 8292 8333 8682 8764 10123 10255 10773 12812 12870 14911 15234 16564

16948 17088 17467

16948 17088 17467

recomRAN <-predict(recRAN, t[7], type="ratings")

as(recomRAN, "matrix")[,1:10]

1 2 3 4 5 6 7 8 9 10

4.899987 4.097082 4.247193 3.729882 5.000000 5.000000 3.233906 3.784761 3.952470 5.000000

recomRAN <-predict(recRAN, t[7], n=5)

as(recomRAN, "list")

[[1]]

[1] "7" "32" "67" "70" "79"

##recomALS <-predict(recALS, t[7], type="ratings")

##Error in combine\_data(model$data@data, newdata@data) :

## Certain users are both in the training data and newdata, but with other items

##recomU <-predict(recU, t[7], type="ratings")

##Error in asMethod(object) :

## Cholmod error 'problem too large' at file ../Core/cholmod\_dense.c, line 105

# recI <- Recommender(r, method="IBCF")

##Error in asMethod(object) :

## Cholmod error 'problem too large' at file ../Core/cholmod\_dense.c, line 105

nr <- nrow(N)

v <- c()

for (i in 1:nr) { if (z(i)==0) v<-append(v,i)}

N <- N[-v,]

#create a matrix in R

## row<- max(mvTrain$V2) ## user

u <- sort(unique(mvTrain$CustID))

row <- length(u) ## no. of unique user (proper way)

v <- sort(unique(mvTrain$mvID))

col<- length(v) ## movie

N<- Matrix(0,nrow=row, ncol=col, dimnames=list(u,v), sparse=TRUE)

library(bigmemory)

big.matrix(row, col)

matTrain <- matrix(0, nrow=row,ncol=col)

## ERROR: cannot allocate matrix of size 63Gb

## for (i in 1:nrow(mv24)) {mat[mv24[i,2], mv24[i,1]] <- mv24[i,3]}

## Using Which to indicate the position of a value in a vector

for (i in 1:nrow(mvTrain)) {matTrain[which(u==mvTrain[i,2], mvTrain[i,1]] <- mvTrain[i,3]}

################################ PROBE ##############################

dat = readLines("probe.txt")

> str(dat)

chr [1:1425333] "1:" "30878" "2647871" "1283744" "2488120" ...

library(“foreach”)

foreach (a in dat) { print(a) }

while (cnt < 7) {

print(v)

cnt = cnt + 1}

mID<- 1

mvID1<-c()

CustID1<-c()

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))}

> str(CustID)

num [1:492581] 30878 2647871 1283744 2488120 317050 ...

> mID

[1] 17031

mID= 17031

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))}

> proc.time()

user system elapsed

13296.58 322.12 41166.01

> mID

[1] 5055

mID=5055

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))}

> mID

[1] 9999

## combine 3 into 1 file

mvID<-c()

CustID<-c()

mvID<-append(mvID1, mvID2)

mvID<-append(mvID, mvID3)

CustID<-append(CustID1, CustID2)

CustID<-append(CustID, CustID3)

> str(mvID)

num [1:1408395] 1 1 1 1 1 1 1 1 1 1 ...

> str(CustID)

num [1:1408395] 30878 2647871 1283744 2488120 317050 ...

> proc.time()

user system elapsed

10368.96 317.82 37334.00

#for all dat, run all night without stopping

> str(dat)

chr [1:1425333] "1:" "30878" "2647871" "1283744" "2488120" "317050" "1904905" "1989766" "14756" "1027056"

> dat[1425328:length(dat)]

[1] "9998:" "1288730" "2536567" "1107317" "9999:" "1473765"

> object.size(dat)

34432912 bytes

> gsub("\\.", "\_", "a.b")

[1] "a\_b"

gsub("\\:", "", "1000b:")

[1] "1000b"

chars <- "test"

value <- "es"

grepl(value, chars)

# TRUE

hist <- as.numeric(pichman$WS)

##

L= length(CustID)

Rate01<- c()

Rate1<- c()

Rate2<- c()

Rate3<- c()

Rate<- c()

prob <- cbind(CustID, mvID)

prob<- prob[order(prob[,1], prob[,2]),]

for (k in 1:499999){ Rate1[k] <- N[CustID1[k],mvID1[k]]; print(k)}

##1hr only compute 10000 records

for (k in 8756:499999) Rate1[k] <- N[CustID1[k],mvID1[k]]

for (k in 500000:999999) Rate2[k] <- N[CustID2[k],mvID2[k]]

for (k in 1000000:L) Rate3[k] <- N[CustID3[k],mvID3[k]]

prob <- cbind(CustID, mvID)

prob<- prob[order(prob[,1], prob[,2]),]

CustUID <-sort(unique(CustID))

length(CustUID)

#[1] 462858

For(j in

Rate<- c()

probe<-cbind(prob,Rate)

P <- sparseMatrix(i=prob[[1]],j=prob[[2]])

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

p<-as(P,"realRatingMatrix")

probe<-read.csv("probe.txt", header=FALSE)

n=nrow(probe)

n

[1] 1425333

library(stringr)

probe<-probe %>% str\_replace(":","")

###################

for (i in 1:n) {if

## Using acast to reshape

tr<-tr[,-c(1)]

library("reshape2")

y then looks like this:

id variable value

1 1 var1 0.1560812

2 2 var1 1.0343844

3 3 var1 -1.4157728

4 1 var2 0.8808935

5 2 var2 0.1719239

6 3 var2 0.6723758

7 1 var3 -0.7589631

8 2 var3 1.1325995

9 3 var3 -1.5744876

now I can cast it back via acast:

> acast(y,y[,1] ~ y[,2])

var1 var2 var3

1 0.1560812 0.8808935 -0.7589631

2 1.0343844 0.1719239 1.1325995

3 -1.4157728 0.6723758 -1.5744876

g<-acast(mvTrain, V2 ~ V1, value.var="V3")

## max value of a vector

max(mv24$V2)

## number of distinct values in a vector

length(unique(mv24$V2))

[1] 37858

## sort a vector

sort(mv24$V2)

##

a <- sort(unique(mv24$V2)

## Order vs. Rank

a <- c(4.1, 3.2, 6.1, 3.1)

order(a)

[1] 4 2 1 3

rank(a)

[1] 3 2 4 1

so rank tells you what order the numbers are in, order tells you how to get them in ascending order.

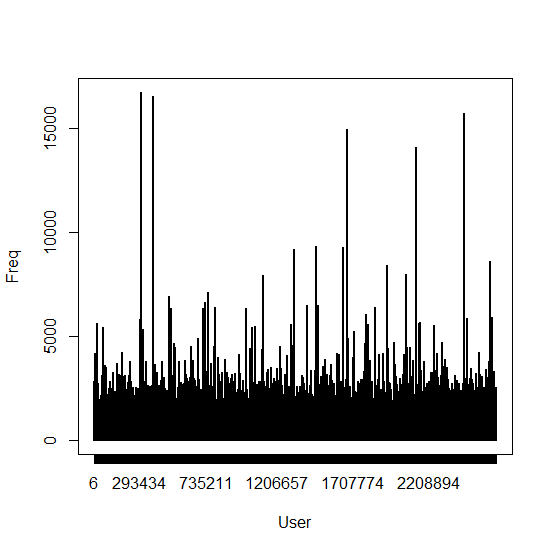
## load .Rhistory

loadhistory(file = ".Rhistory")

savehistory(file = ".Rhistory")

f<-table(mvTrain$CustID)

sort(f, xlab=”User”, ylab=”Freq”)



AWS: <https://www.slideshare.net/ianmcook/running-r-on-the-amazon-cloud-2013-0620>

<http://www.louisaslett.com/RStudio_AMI/video_guide.html>