# Matrix_Factorization

Mritunjay And Sunil

15/10/2019

# Recommended System Techniques on Airbnb (Amsterdam Hotel Recommendation)

# 1. Alternating Least Squares for Recommendation System

## Load all the User build functions

```
# Make recommendations for the target user using User-based CF
getrecommendations_UU <- function(targetuser, users, topN=5, simfun=pearsonsim) {
  sims = apply(users,1,function(user) simfun(user,targetuser))
  sims = sims[!is.na(sims) & sims >=0]
  wavrats = apply(users[names(sims),is.na(targetuser), drop=FALSE],2,function(rats) weighted.
mean(rats, sims, na.rm=TRUE))
  s = sort(wavrats[!is.na(wavrats)], decreasing = TRUE)
  if (topN == FALSE) s else s[1:min(topN,length(s))] # get topN items
}
#getrecommendations_UU = cmpfun(getrecommendations_UU)

# get recommedations for the target user using Item-based CF
getrecommendations_II <- function(targetuser, itemsims, topN=5) {
  targetuser = targetuser[colnames(itemsims)] # ensure the item order is the same as simmatri
x
  seenitems  = !is.na(targetuser)
  unseenitems = is.na(targetuser)
  seenrats = targetuser[seenitems]
  preds = apply(itemsims[unseenitems,seenitems, drop=FALSE], 1, function(simrow) my.weighted.
mean(seenrats, simrow))
  sp = sort(preds[!is.na(preds)] , decreasing = TRUE)
  sp[1:min(topN,length(sp))]  # get topN items
}
#getrecommendations_II = cmpfun(getrecommendations_II)

# compute the item-item similarity matrix (the matrix is symmetric so can compute half & then
copy)
# (setting dir=1 generates the user similarity matrix)
getitemsimsmatrix = function(users, simfun=cosinesim, dir=2) {
  rw <<- 1;
  itemsims = apply(users, dir, function(itemA) {
    rw <<- rw + 1 ; cl <<- 1;
    apply(users,dir,function(itemB) {cl<<-cl+1; if (cl<rw) NA else if (cl==rw) NA else simfun
(itemA,itemB)})
  })
  m = forceSymmetric(itemsims,uplo="L") # copy lower half to upper half
  as.matrix(m)
}
#getitemsimsmatrix = cmpfun(getitemsimsmatrix)

# similarity functions
euclidsim = function(x,y) { z=(y-x)^2; sz=sqrt(sum(z,na.rm=TRUE));
                            if (sz!=0) 1/(1+sz) else if (length(which(!is.na(z)))==0) NA else
1/(1+sz)}

euclidsimF= function(x,y) { z=(y-x)^2; sz=sum(z,na.rm=TRUE);
                            if (sz!=0) 1/(1+sz) else if (length(which(!is.na(z)))==0) NA else
1/(1+sz)}

cosinesim = function(x,y) { xy = x*y; sum(xy, na.rm=TRUE)/(sqrt(sum(x[!is.na(xy)]^2)*sum(y[!i
s.na(xy)]^2)))}

pearsonsim= function(x,y) { suppressWarnings(cor(unlist(x),unlist(y),use="pairwise.complete.o
bs")) }

mypearsim = function(x,y) { xy = x*y; x=x[!is.na(xy)]; y=y[!is.na(xy)];
                            mx=mean(x); my=mean(y);
```

```
                            sum((x-mx)*(y-my))/(sqrt(sum((x-mx)^2)*sum((y-my)^2)))}

pearsonRM = function(x,y) { mx=mean(x,na.rm=TRUE);my=mean(y,na.rm=TRUE);
                            xy=x*y;x=x[!is.na(xy)]; y=y[!is.na(xy)]
                            sum((x-mx)*(y-my))/(sqrt(sum((x-mx)^2)*sum((y-my)^2)))}

jacardsim = function(x,y) { validx= !is.na(x); validy= !is.na(y);
                            sum(as.integer(validx&validy))/sum(as.integer(validx|validy))}

##############################################################################
# For testing, we split the data by user, so test users are not in the trainset
# This is clean but does not test the situation where partial information
# is known about a user (as may be the case in User-based scenario).
# For item-based having partial info will make very little difference (since simmatrix is pre
computed)
##############################################################################

# make predicted ratings for a sample of items for each test user
# if trainusers is defined then do User-based CF else do Item-based CF
# Note: if Item-based CF is to be performed them the itemsimilarity matrix (itemsims) must be
defined
predictCF = function(testusers, trainusers=NULL, itemsims=NULL, numtestitems=10, random=FALSE
, simfun=cosinesim) {
  preds = sapply(1:nrow(testusers),function(i) {
    cat(".")
    predictuser(testusers[i,],trainusers=trainusers,itemsims=itemsims,numtestitems=numtestite
ms,random=random,simfun=simfun)})
  colnames(preds) = rownames(testusers)
  preds
}

predictuser <- function(testuser, trainusers=NULL, itemsims=NULL, numtestitems=10, random=FAL
SE, simfun=cosinesim) {
  seenitemnames   = names(testuser)[!is.na(testuser)]
  if (random) testitemnames = sample(seenitemnames,min(numtestitems,length(seenitemnames))) #
test a random N items
  else testitemnames = seenitemnames[1:min(numtestitems,length(seenitemnames))] # test first
 N items
  preds = list()
  for (testitemname in testitemnames) {
    truerating = testuser[testitemname]
    testuser[testitemname] = NA
    if (!is.null(trainusers)) {
      # do user-based CF
      usersims = apply(trainusers,1,function(trainuser) simfun(trainuser,testuser))
      usersims = usersims[!is.na(usersims) & usersims >=0]
      predictedrating = my.weighted.mean(trainusers[names(usersims),testitemname], usersims)
    }
    else {
      # do item-based CF
      predictedrating = my.weighted.mean(testuser[seenitemnames], itemsims[seenitemnames,test
itemname])
    }
    testuser[testitemname] = truerating # restore the actual rating
    preds = c(preds,predictedrating,truerating)
  }
  preds = unname(preds)
  m = as.matrix(preds)
```

```
  if (length(m) < numtestitems*2) for (i in (length(m)+1):(numtestitems*2)) { m = rbind(m,NA
)}
  return(m)
}
#predictuser= cmpfun(predictuser)

# a weighted mean that handles NA's in both arguments (ratings and similarities)
my.weighted.mean = function(x,y) {
    xy = x*y;
    z = sum(abs(y[!is.na(xy)]))
    if (z == 0) as.numeric(NA) else sum(xy,na.rm=TRUE)/z
}
#my.weighted.mean = cmpfun(my.weighted.mean)

# computes average, mean absolute error
# each row contains prediction, actual, prediction, actual etc, hence errors are just the dif
f between consecutive cells
avgMAE = function(preds) {
  plist = unlist(preds)
  errors = sapply(1:(length(plist)/2),function(i) abs(plist[i*2-1]-plist[i*2]))
  errors = errors[errors != Inf]
  mean(errors,na.rm=TRUE)
}

showCM = function(preds, like) {
  plist = unlist(preds)
  cnts = sapply(1:(length(plist)/2), function(i) {
    pred = plist[i*2-1] ; actual = plist[i*2]
    if (!is.na(pred) & !is.nan(actual)) {
      if (pred>=like) {if(actual>=like) c(1,0,0,0) else c(0,1,0,0)}
      else if(actual<like) c(0,0,1,0) else c(0,0,0,1)
    } else c(0,0,0,0)
  })
  s = rowSums(cnts)   #returns cnts for: TP, FP, TN, FN

  cat(sprintf("TN=%5d FP=%5d\n",s[3],s[2]))
  cat(sprintf("FN=%5d TP=%5d  (total=%d)\n",s[4],s[1], sum(s)))
  cat(sprintf("accuracy  = %0.1f%%\n",(s[1]+s[3])*100/sum(s)))
  cat(sprintf("precision = %3.1f%%\n",s[1]*100/(s[1]+s[2])))
  cat(sprintf("recall    = %3.1f%%\n",s[1]*100/(s[1]+s[4])))
}

########################
# miscellaneous aids
########################

maketraintest = function(users,numtestusers) {
  testnames  = sample(rownames(users), min(numtestusers,nrow(users))) # identify N users rand
omly for testing
  trainnames = setdiff(rownames(users),testnames) # take remaining users for training
  trainusers <<- users[trainnames,]
  testusers  <<- users[testnames,]
  list(trainusers,testusers)
}

# extract only prediction or only actual ratings from the output of predictCF()
listpreds= function(results) {unlist(results)[c(TRUE,FALSE)]}
listrats = function(results) {unlist(results)[c(FALSE,TRUE)]}
```

```
validcnt = function(x) length(which(is.finite(x)))

# How sparse is the data in a data frame? Compute % of non-blank entries
fillrate = function(df) {cat((length(which(!is.na(df)))*100)/(nrow(df)*ncol(df)),"%")}
#fillrate = cmpfun(fillrate)

# same as above but also works on vectors
fillratev = function(df) {t=unlist(df); cat((length(which(!is.na(t)))*100)/length(t),"%")}
#fillratev = cmpfun(fillratev)

# how many values are > 0? Compute % of entries > 0
fillrateG = function(df,thresh) {t=unlist(df); cat((length(which(!is.na(t) & t > thresh))*100
)/length(t),"%")}
fillrateL = function(df,thresh) {t=unlist(df); cat((length(which(!is.na(t) & t < thresh))*100
)/length(t),"%")}
fillrateE = function(df,thresh) {t=unlist(df); cat((length(which(!is.na(t) & t == thresh))*10
0)/length(t),"%")}
```

# Load all the relevant libraries and Get the working directory and Load the Amsterdam Hotel Airbn data set

```
pacman::p_load(tidyverse, purrr, stringr, data.table, modelr, readxl,caret, corrplot, broom,
ggpubr, tm, proxy, MASS,relaimpo, car,interplot, caTools, mice, gbm, reshape2, compiler, reco
mmenderlab, Matrix, knitr,tidyr, dplyr, animation, wordnet, RColorBrewer, wordcloud, Snowball
C, topicmodels, ggplot2, cluster, fpc, recosystem, dtplyr,softImpute)
getwd()
```

```
## [1] "C:/Users/Rapsy/Desktop/Recommender_Assignment/MJ/Model Based Recommendation System"
```

```
airbnb = read.csv("airbnb.csv", header=TRUE, sep=",") # transaction format!
names(airbnb) = c(colnames(airbnb))
head(airbnb,1)
```

```
##   Hotel_Id        Host_Name User_Id User_Name
## 1     2818 Erik And Mary Jo 2914515     Ivana
##                                 Hotel_name
## 1 Quiet Garden View Room & Super Fast WiFi
##                                    summary
## 1 Quiet Garden View Room & Super Fast WiFi
##
space
## 1 I'm renting a bedroom (room overlooking the garden) in my apartment in Amsterdam,  The r
oom is located to the east of the city centre in a quiet, typical Amsterdam neighbourhood the
"Indische Buurt". AmsterdamÃ¢\200\231s historic centre is less than 15 minutes away by bike o
r tram. The features of the room are: - Twin beds (80 x 200 cm, down quilts and pillows)  - 2
pure cotton towels for each guest  - reading lamps - bedside table - wardrobe - table with ch
airs - tea and coffee making facilities - mini bar - alarm clock - Hi-Fi system with cd playe
r, connection for mp3 player / phone - map of Amsterdam and public transport - Wi-Fi Internet
connection  Extra services: - Bike rental
##
description
## 1 Quiet Garden View Room & Super Fast WiFi I'm renting a bedroom (room overlooking the gar
den) in my apartment in Amsterdam,  The room is located to the east of the city centre in a q
uiet, typical Amsterdam neighbourhood the "Indische Buurt". AmsterdamÃ¢\200\231s historic cen
tre is less than 15 minutes away by bike or tram. The features of the room are: - Twin beds
(80 x 200 cm, down quilts and pillows)  - 2 pure cotton towels for each guest  - reading lamp
s - bedside table - wardrobe - table with chairs - tea and coffee making facilities - mini ba
r - alarm clock - Hi-Fi system with cd player, connection for mp3 player / phone - map of Ams
terdam and public transport - Wi-Fi Internet connection  Extra services: - Bike rental Indisc
he Buurt ("Indies Neighborhood") is a neighbourhood in the eastern portion of the city of Ams
terdam, in the Dutch province of Noord-Holland. The name dates from the early 20th century an
d is derived from the fact that the neighbourhood's streets are named after islands a
##   host_id host_name property_type    room_type accommodates
## 1 4070804    Daniel     Apartment Private room   Two Person
##             bathrooms    bedrooms    beds bed_type
## 1 One attach bathroom One bedroom One bed Real Bed
##
amenities
## 1 {Internet,Wifi,"Paid parking off premises","Buzzer/wireless intercom",Heating,Washer,"Sm
oke detector","Carbon monoxide detector","First aid kit","Safety card","Fire extinguisher",Es
sentials,Shampoo,"Lock on bedroom door","24-hour check-in",Hangers,"Hair dryer",Iron,"Laptop
friendly workspace","translation missing: en.hosting_amenity_49","translation missing: en.hos
ting_amenity_50","Private entrance","Hot water","Bed linens","Extra pillows and blankets","Si
ngle level home","Garden or backyard","No stairs or steps to enter","Flat path to guest entra
nce","Well-lit path to entrance","No stairs or steps to enter","Accessible-height bed","No st
airs or steps to enter","Host greets you","Handheld shower head","Paid parking on premises"}
##           cancellation_policy Ratings
## 1 strict_14_with_grace_period       3
```

# Structure of Datasets

```
str(airbnb)
```

```
## 'data.frame':    20677 obs. of  20 variables:
##  $ Hotel_Id           : int  2818 2818 2818 2818 2818 2818 2818 2818 2818 2818 ...
##  $ Host_Name          : Factor w/ 508 levels "Aafje","Adriana",..: 136 136 136 136 136 136
136 136 136 136 ...
##  $ User_Id            : int  2914515 5711109 2944771 4620679 373226 2200958 1348274 543307
6 2847616 857406 ...
##  $ User_Name          : Factor w/ 2932 levels "(Email hidden by Airbnb)",..: 1205 1153 287
5 1130 2021 2308 413 2823 569 1964 ...
##  $ Hotel_name         : Factor w/ 507 levels "'Westerpark Sanctuary', Office-Apartmen
t",..: 383 383 383 383 383 383 383 383 383 383 ...
##  $ summary            : Factor w/ 382 levels "","'LORE'S PLACE' A lovely, open writers hom
e in the fun 'Indische Buurt' in Amsterdam! We are offering a open pla"| __truncated__,..: 24
2 242 242 242 242 242 242 242 242 242 ...
##  $ space              : Factor w/ 504 levels "","- 100 m2 floor space - private garden of
45 m2  - living room with a '30s bar, 55 inch QLED TV and home cinema "| __truncated__,..: 15
8 158 158 158 158 158 158 158 158 158 ...
##  $ description        : Factor w/ 506 levels "'LORE'S PLACE' A lovely, open writers home i
n the fun 'Indische Buurt' in Amsterdam! We are offering a open pla"| __truncated__,..: 317 3
17 317 317 317 317 317 317 317 317 ...
##  $ host_id            : int  4070804 4070804 4070804 4070804 4070804 4070804 4070804 40708
04 4070804 4070804 ...
##  $ host_name          : Factor w/ 404 levels "Aafje","Adriana",..: 81 81 81 81 81 81 81 81
81 81 ...
##  $ property_type      : Factor w/ 15 levels "Apartment","Bed and breakfast",..: 1 1 1 1 1
1 1 1 1 1 ...
##  $ room_type          : Factor w/ 3 levels "Entire home/apt",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ accommodates       : Factor w/ 10 levels "Five Person",..: 10 10 10 10 10 10 10 10 10 1
0 ...
##  $ bathrooms          : Factor w/ 11 levels "Four attach bathroom",..: 4 3 3 3 3 3 3 3 3 3
...
##  $ bedrooms           : Factor w/ 7 levels "Five bedroom",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ beds               : Factor w/ 7 levels "Five bed","Four bed",..: 3 3 3 3 3 3 3 3 3 3
...
##  $ bed_type           : Factor w/ 4 levels "Couch","Futon",..: 4 4 4 4 4 4 4 4 4 4 ...
##  $ amenities          : Factor w/ 508 levels "{\"Cable TV\",Internet,Wifi,\"Paid parking o
ff premises\",\"Buzzer/wireless intercom\",Heating,\"Family/kid fri"| __truncated__,..: 16 16
16 16 16 16 16 16 16 16 ...
##  $ cancellation_policy: Factor w/ 3 levels "flexible","moderate",..: 3 3 3 3 3 3 3 3 3 3
...
##  $ Ratings            : int  3 2 5 3 3 3 3 3 2 3 ...
```

# Removing all those users corresponding to missing ratings and

# Extract only the explicit ratings and visualize the histogram of Ratings

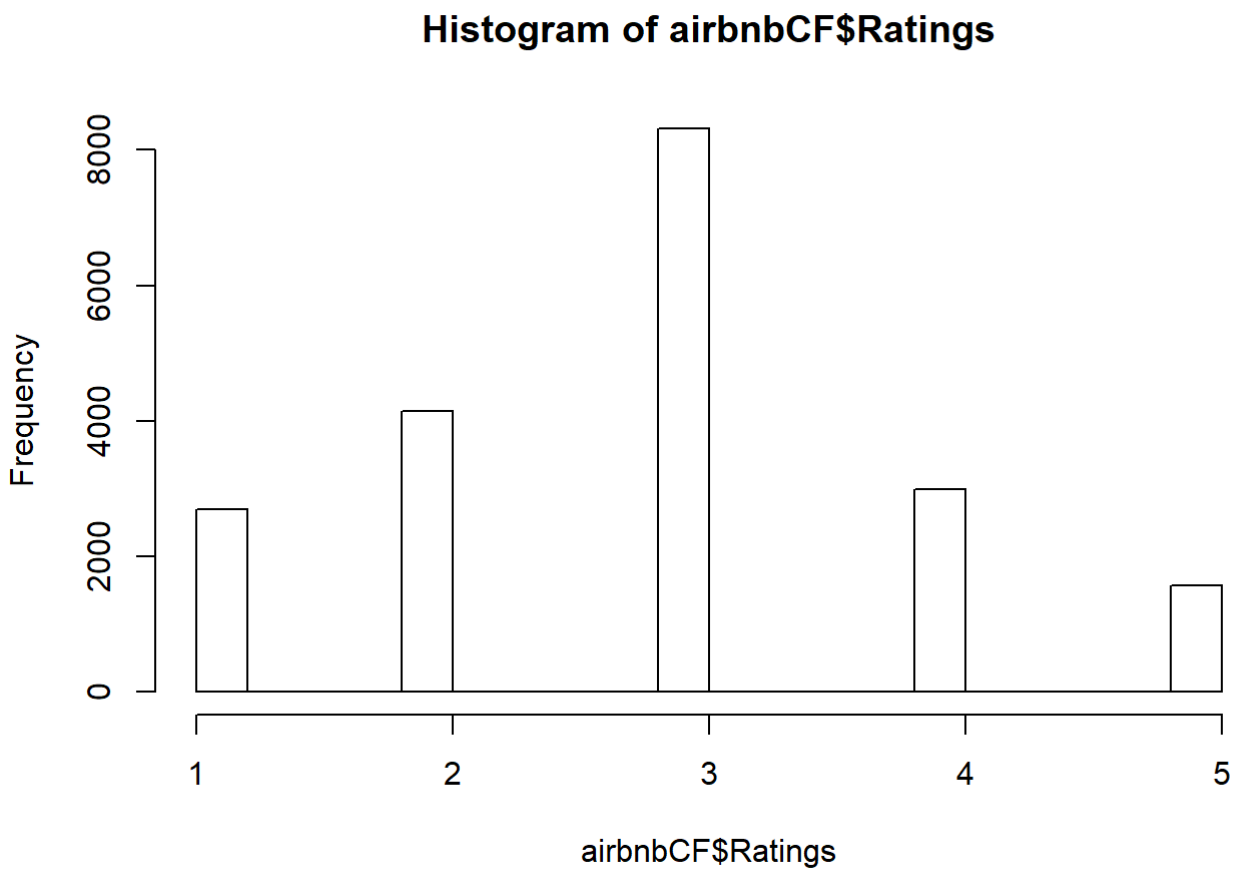
```
airbnbCF = airbnb[,c("Hotel_Id","User_Id", "Ratings")]
sapply(airbnbCF, function(x){sum(is.na(x))})
```

```
## Hotel_Id  User_Id  Ratings
##        0        0      967
```

```
airbnbCF$Ratings[is.na(airbnbCF$Ratings)] = 0
airbnbCF = airbnbCF[airbnbCF$Ratings > 0,]
sapply(airbnbCF, function(x){sum(is.na(x))})
```

```
## Hotel_Id  User_Id  Ratings
##        0        0        0
```

```
hist(airbnbCF$Ratings)
```




# Eliminate users with too few ratings and Consider Activer users who had rated hotels more than and equal to 10 hotels

```
cnts = aggregate(Hotel_Id ~ User_Id, data = airbnbCF, FUN = length)
colnames(cnts) = c("user","numitems")
activeusers = cnts$user[cnts$numitems >= 10] ; length(activeusers)
```

```
## [1] 422
```

```
evCF = airbnbCF[airbnbCF$User_Id %in% activeusers,]
dim(evCF)
```

```
## [1] 4672    3
```

## Eliminate Hotels with too few ratings and Consider Active Hotels who had been rated more than and equal to 10 users

```
cnts = aggregate(User_Id ~ Hotel_Id, data = airbnbCF, FUN=length)
colnames(cnts) = c("item","numusers")
popularhotels = cnts$item[cnts$numusers >= 10] ; length(popularhotels)
```

```
## [1] 508
```

```
ev = evCF[evCF$Hotel_Id %in% popularhotels,]
dim(ev)
```

```
## [1] 4672    3
```

```
str(ev)
```

```
## 'data.frame':    4672 obs. of  3 variables:
##  $ Hotel_Id: int  2818 2818 2818 2818 2818 2818 2818 20168 20168 20168 ...
##  $ User_Id : int  2944771 2847616 2807294 4489932 5461945 4380449 4644013 913549 5039682 4
017740 ...
##  $ Ratings : num  5 2 4 3 3 2 5 2 4 2 ...
```

## Remove duplicate records from the datasets

```
ev_Final = ev %>% distinct(User_Id,Hotel_Id,.keep_all = TRUE)
dim(ev_Final)
```

```
## [1] 4621    3
```

```
str(ev_Final)
```

```
## 'data.frame':    4621 obs. of  3 variables:
##  $ Hotel_Id: int  2818 2818 2818 2818 2818 2818 2818 20168 20168 20168 ...
##  $ User_Id : int  2944771 2847616 2807294 4489932 5461945 4380449 4644013 913549 5039682 4
017740 ...
##  $ Ratings : num  5 2 4 3 3 2 5 2 4 2 ...
```

# (2) using the softImpute library

## reread the data ensuring users and items are read as factors

```
events = ev_Final[,c(2,1,3)]
ctypes = c("factor","factor","numeric")
colnames(events) = c("user","item","rating")
events$user= factor(events$user)
events$item= factor(events$item)
str(events)
```

```
## 'data.frame':    4621 obs. of  3 variables:
##  $ user  : Factor w/ 422 levels "57920","142145",..: 194 183 177 327 400 316 341 44 372 28
1 ...
##  $ item  : Factor w/ 508 levels "2818","20168",..: 1 1 1 1 1 1 1 2 2 2 ...
##  $ rating: num  5 2 4 3 3 2 5 2 4 2 ...
```

# Create a wide format of dataset

```
users = acast(events, user ~ item, value.var = "rating")
#colnames(users) = sort(unique(events$item))
#rownames(users) = sort(unique(events$user))
users[1:10,1:15]
```

```
##        2818 20168 25428 27886 28871 29051 31080 38266 42970 43109 43980
## 57920    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## 142145   NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## 186729   NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## 187580   NA    NA     2    NA    NA    NA    NA    NA    NA    NA    NA
## 195580   NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## 195859   NA    NA    NA    NA    NA    NA    NA     4    NA    NA    NA
## 201541   NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## 216385   NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## 241336   NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## 262799   NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
##        44129 44391 46386 47061
## 57920     NA    NA    NA    NA
## 142145    NA    NA    NA    NA
## 186729    NA    NA    NA    NA
## 187580     1    NA    NA     3
## 195580    NA    NA    NA    NA
## 195859    NA    NA    NA    NA
## 201541    NA    NA    NA    NA
## 216385    NA     3    NA    NA
## 241336    NA    NA    NA    NA
## 262799    NA    NA    NA    NA
```

# split the events using the same split (train_ind & test_ind) as used earlier

```
set.seed(123)
smp_size <- floor(0.8 * nrow(events))
train_indexes <- sample(1: nrow(events), size = smp_size)
trainevents <- events[train_indexes, ]; dim(trainevents)
```

```
## [1] 3696    3
```

```
testevents  <- events[-train_indexes, ]; dim(testevents)
```

```
## [1] 925   3
```

```
write.csv(trainevents, "trainevents.csv")
write.csv(testevents, "testevents.csv")
```

# make a copy and then blank out the test events (ie set test ratings for the test (user,item) pairs to NA)

```
trainusers = users
cat("Fill rate whole wide matrix : ");
```

```
## Fill rate whole wide matrix :
```

```
fillrate(trainusers)
```

```
## 2.155558 %
```

```
cat("\n")
```

```
cat("Fill rate Testset matrix : ");
```

```
## Fill rate Testset matrix :
```

```
x = apply(testevents,1,function(row) trainusers[row[1],row[2]] <<- NA) # row[1] ~ user, row
[2] ~ item
fillrate(trainusers)
```

```
## 1.724074 %
```

# factorize into U * D * V using 30 latent features

```
trainusers=as(trainusers,"Incomplete") # coerce into correct matrix format with missing entri
es
```

# do one of the below

```
fit1=softImpute(trainusers, rank.max=30, type="als") # als is the default
fit2=softImpute(trainusers, rank.max=30, type="svd") # for comparison
```

# take a look at the factorised matrixes

```
dim(fit1$u) ; fit1$u[1:10,1:5] # the user latent features
```

```
## [1] 422  30
```

```
##               [,1]         [,2]         [,3]         [,4]        [,5]
##  [1,] -0.0102665120 -0.039022002 -0.039916900  0.017307879 -0.05132815
##  [2,] -0.0485639745 -0.014048117  0.091880517 -0.003925695  0.03365736
##  [3,] -0.0435257243  0.008191458 -0.046380231  0.030249175 -0.03265346
##  [4,] -0.0441161024  0.046941331 -0.044996848 -0.054321034  0.07684595
##  [5,]  0.0004133337  0.002748377  0.009720409 -0.039670830 -0.09524968
##  [6,]  0.0269234434 -0.039275965  0.011325124 -0.071261375  0.09622555
##  [7,] -0.0547167417 -0.003534646  0.113800828  0.055150135  0.02090859
##  [8,] -0.0324761649 -0.018744270 -0.022901537  0.038781958 -0.07593921
##  [9,] -0.0382489783 -0.012074396 -0.042280848  0.036422480  0.06002570
## [10,] -0.0881630665 -0.084178802 -0.046947466  0.002412226 -0.02267661
```

```
dim(fit1$v) ; fit1$v[1:10,1:5] # the item latent features
```

```
## [1] 508  30
```

```
##               [,1]        [,2]         [,3]         [,4]        [,5]
##  [1,] -0.064264593 -0.08652862 -0.029733556 -0.018568499 -0.04809533
##  [2,] -0.006459371  0.02884584  0.007924384  0.001926542 -0.06942288
##  [3,] -0.051640991 -0.01899587 -0.027683398 -0.018424423  0.01479871
##  [4,] -0.059439289 -0.04315917 -0.002858857  0.044273551 -0.04693955
##  [5,] -0.051473735  0.06148844 -0.037114580 -0.023259303  0.04861710
##  [6,] -0.015081718  0.01600833  0.022450516  0.006833347  0.03346169
##  [7,] -0.060154176 -0.05729578  0.054448279 -0.006753314  0.01315366
##  [8,] -0.004018383  0.02697307  0.105513347  0.022276481  0.02623606
##  [9,] -0.037741161 -0.04860415 -0.058899188  0.022065958 -0.04939723
## [10,] -0.033744133  0.01806630  0.018701708 -0.019520701  0.05652530
```

```
length(fit1$d); head(fit1$d)   # the singular values
```

```
## [1] 30
```

```
## [1] 136.5130 107.4458 104.6133 102.5861 101.7587 101.2385
```

# make predictions for all of the empty (user,item) pairs (the test pairs + those missing in orginal dataset)

```
trainuserscompleted1 = complete(trainusers, fit1)
trainuserscompleted2 = complete(trainusers, fit2)
```

# compute the MAE for the predictions made for the test events fir model 1 - fit1 (Using ALS)

```
rownames(trainuserscompleted1) = rownames(users) # copy across the user names
colnames(trainuserscompleted1) = colnames(users) # copy across the item names
# Output recommendation using ALS.
trainuserscompleted1[1:10,1:10]
```

```
##               2818      20168       25428      27886      28871
## 57920   1.49656560 -0.1432019 -0.24430582  1.0739055 -0.5314558
## 142145 -0.44328097 -0.7048120  1.06300311  1.5819880  0.3333384
## 186729  0.09959493 -0.1377113  0.89264230  1.4541477  1.7345458
## 187580 -0.04135468 -0.4857795  2.00000000 -3.1014433  0.4174980
## 195580  0.48677202  0.4724423  1.22155216 -1.0072422 -2.2254499
## 195859  0.22559863 -0.3402947 -0.53471554 -1.9558357  0.6289133
## 201541  0.52344801 -0.9103698  0.06991817  0.7483926  1.0288884
## 216385  1.08106566  1.1276565 -0.16031961  0.3074504 -0.6153120
## 241336 -0.97940174 -0.6633407  0.04455350 -0.3976856  1.7096291
## 262799  0.02042747 -0.4486453  2.25595400  1.1410179 -0.5987040
##              29051       31080      38266       42970      43109
## 57920  -0.24856111 -0.35155931 -1.1371221  0.96959425  0.3705530
## 142145  2.15969410  0.72742235  1.1970255 -1.33872357  0.7322183
## 186729  0.29045616  1.04272823  1.0054205  0.30050192  0.7711399
## 187580 -0.61475475 -0.43023071 -0.0946785 -1.84449828 -0.3311865
## 195580 -0.68246239 -1.14384327 -1.1408844 -1.62165847 -1.6309054
## 195859 -0.30923819  0.64381606  0.3746261  0.03597052  0.2982090
## 201541  1.89601894  1.48553943  2.1973977 -0.06174800  1.9030811
## 216385  0.53315554  0.05180157 -0.5238580  1.91018788 -1.0936715
## 241336 -0.36994894  0.46422255 -0.7485257  1.20720836  1.2185064
## 262799 -0.06865498  0.55479734 -0.5918073 -0.61246149 -0.1697243
```

```
dim(trainuserscompleted1) # 422 508
```

```
## [1] 422 508
```

```
outcome = as.data.frame(trainuserscompleted1)
#outcome = outcome[,-1]

Top1_Hotel = integer(nrow(outcome))
Top2_Hotel = integer(nrow(outcome))
Top3_Hotel = integer(nrow(outcome))
Top4_Hotel = integer(nrow(outcome))
Top5_Hotel = integer(nrow(outcome))

for (i in 1:nrow(outcome)) {
  a = as.matrix(outcome[i,])[1,]
  Top1_Hotel[i] = names(a[order(a,decreasing=TRUE)[1]])
  Top2_Hotel[i] = names(a[order(a,decreasing=TRUE)[2]])
  Top3_Hotel[i] = names(a[order(a,decreasing=TRUE)[3]])
  Top4_Hotel[i] = names(a[order(a,decreasing=TRUE)[4]])
  Top5_Hotel[i] = names(a[order(a,decreasing=TRUE)[5]])
}

df1 <- data.frame(Top1_Hotel, Top2_Hotel, Top3_Hotel, Top4_Hotel, Top5_Hotel, stringsAsFactor
s = TRUE)
rownames(df1) = rownames(users)
write.csv(df1, file = "Recommended Hotel For each user using ALS.csv")
abserrs = apply(testevents, 1, function(row) abs(trainuserscompleted1[row[1],row[2]] - users
[row[1],row[2]])) # row[1] ~ user, row[2] ~ item
mean(t(abserrs), na.rm=TRUE) # show the MAE
```

```
## [1] 2.662412
```

# compute the MAE for the predictions made for the test events fir model 2 - fit2 (Using SVD)

```
rownames(trainuserscompleted2) = rownames(users) # copy across the user names
colnames(trainuserscompleted2) = colnames(users) # copy across the item names
# Output recommendation using ALS.
trainuserscompleted2[1:10,1:10]
```

```
##              2818       20168       25428       27886      28871
## 57920   1.6229976 -0.73719264 -0.66822467 -0.09787137  0.7993859
## 142145  0.4762894  0.08311856 -0.06130967 -0.51905792  0.6893669
## 186729  0.2714230  0.03178666  0.01392250  0.46083892  1.3745222
## 187580  0.3586904  0.97810811  2.00000000 -0.50077113  0.1635062
## 195580 -0.3508718 -0.05309896  0.07706351  0.23999472 -0.6767932
## 195859 -0.3379241 -0.16059954 -0.06382876 -1.22144524 -0.4601641
## 201541  2.1777938  0.04281306  0.69269621 -0.97911890  0.8348631
## 216385  0.1349993  1.41917835 -0.24047881  0.73530561 -0.5436771
## 241336 -1.7513406 -0.82290619 -0.72880671 -0.02108603  1.6754001
## 262799  0.3443658  0.04423292  0.24964975  0.87842163 -0.1676665
##             29051       31080         38266       42970      43109
## 57920   1.1800501  0.17313151  0.0007116486  1.24990267 -0.2466749
## 142145  0.9842375  0.82525185  0.2276855547  0.01258011 -0.4363097
## 186729 -0.3376164  0.83137245 -0.1335478067  2.14062517  0.3925549
## 187580  0.2514803 -0.26988401  0.2947752897  1.05435622  0.8702850
## 195580  0.7191313 -0.30691169 -0.0669920311 -0.70409017  0.6630953
## 195859  0.9759271 -0.07602706 -0.2401142291 -0.86556146 -0.5946764
## 201541 -0.7614856  1.28705867  0.0515483596  1.17352870 -0.9709091
## 216385 -0.1632830  0.98337638  1.5727420204  0.59676554  0.3410050
## 241336 -0.5135105  0.59255318 -0.6502973895 -0.62030195  0.6515390
## 262799  0.8335425  0.34661632 -0.0171279312  1.32068453  0.1541693
```

```
dim(trainuserscompleted2) # 422 508
```

```
## [1] 422 508
```

```
outcome = as.data.frame(trainuserscompleted2)
#outcome = outcome[,-1]

Top1_Hotel = integer(nrow(outcome))
Top2_Hotel = integer(nrow(outcome))
Top3_Hotel = integer(nrow(outcome))
Top4_Hotel = integer(nrow(outcome))
Top5_Hotel = integer(nrow(outcome))

for (i in 1:nrow(outcome)) {
  a = as.matrix(outcome[i,])[1,]
  Top1_Hotel[i] = names(a[order(a,decreasing=TRUE)[1]])
  Top2_Hotel[i] = names(a[order(a,decreasing=TRUE)[2]])
  Top3_Hotel[i] = names(a[order(a,decreasing=TRUE)[3]])
  Top4_Hotel[i] = names(a[order(a,decreasing=TRUE)[4]])
  Top5_Hotel[i] = names(a[order(a,decreasing=TRUE)[5]])
}

df2 <- data.frame(Top1_Hotel, Top2_Hotel, Top3_Hotel, Top4_Hotel, Top5_Hotel, stringsAsFactor
s = TRUE)
rownames(df2) = rownames(users)
write.csv(df2, file = "Recommended Hotel For each user using SVD.csv")
abserrs = apply(testevents, 1, function(row) abs(trainuserscompleted2[row[1],row[2]] - users
[row[1],row[2]])) # row[1] ~ user, row[2] ~ item
mean(t(abserrs), na.rm=TRUE) # show the MAE
```

```
## [1] 2.57295
```

# for comparison we can make the predictions for the test set events manually from the fractorised matrices

## add user and item names to U and V so we can index them by user name and item name for ALS

```
rownames(fit1$u) = sort(unique(events$user))
rownames(fit1$v) = sort(unique(events$item))
fit1$u[1:10,1:5]; fit1$v[1:10,1:5]
```

```
##                [,1]         [,2]         [,3]         [,4]        [,5]
## 57920  -0.0102665120 -0.039022002 -0.039916900  0.017307879 -0.05132815
## 142145 -0.0485639745 -0.014048117  0.091880517 -0.003925695  0.03365736
## 186729 -0.0435257243  0.008191458 -0.046380231  0.030249175 -0.03265346
## 187580 -0.0441161024  0.046941331 -0.044996848 -0.054321034  0.07684595
## 195580  0.0004133337  0.002748377  0.009720409 -0.039670830 -0.09524968
## 195859  0.0269234434 -0.039275965  0.011325124 -0.071261375  0.09622555
## 201541 -0.0547167417 -0.003534646  0.113800828  0.055150135  0.02090859
## 216385 -0.0324761649 -0.018744270 -0.022901537  0.038781958 -0.07593921
## 241336 -0.0382489783 -0.012074396 -0.042280848  0.036422480  0.06002570
## 262799 -0.0881630665 -0.084178802 -0.046947466  0.002412226 -0.02267661
```

```
##              [,1]        [,2]        [,3]        [,4]        [,5]
## 2818  -0.064264593 -0.08652862 -0.029733556 -0.018568499 -0.04809533
## 20168 -0.006459371  0.02884584  0.007924384  0.001926542 -0.06942288
## 25428 -0.051640991 -0.01899587 -0.027683398 -0.018424423  0.01479871
## 27886 -0.059439289 -0.04315917 -0.002858857  0.044273551 -0.04693955
## 28871 -0.051473735  0.06148844 -0.037114580 -0.023259303  0.04861710
## 29051 -0.015081718  0.01600833  0.022450516  0.006833347  0.03346169
## 31080 -0.060154176 -0.05729578  0.054448279 -0.006753314  0.01315366
## 38266 -0.004018383  0.02697307  0.105513347  0.022276481  0.02623606
## 42970 -0.037741161 -0.04860415 -0.058899188  0.022065958 -0.04939723
## 43109 -0.033744133  0.01806630  0.018701708 -0.019520701  0.05652530
```

# add user and item names to U and V so we can index them by user name and item name for SVD

```
rownames(fit2$u) = sort(unique(events$user))
rownames(fit2$v) = sort(unique(events$item))
fit2$u[1:10,1:5]; fit2$v[1:10,1:5]
```

```
##              [,1]        [,2]        [,3]        [,4]        [,5]
## 57920  -0.03675581  0.003098126 -0.076972886  0.023992969 -0.01148598
## 142145 -0.04204361 -0.018484234 -0.009909967  0.048101908  0.02025055
## 186729 -0.03576007  0.049186948  0.040928790  0.026538277  0.05153958
## 187580 -0.03201761 -0.005949048 -0.021339361 -0.073479685  0.01486073
## 195580 -0.04420493  0.040354793  0.046911348 -0.021913682 -0.03098403
## 195859 -0.03363530  0.025762329 -0.055905538 -0.018142743  0.01708473
## 201541 -0.03107184  0.042517255 -0.006959076  0.014680940  0.02933230
## 216385 -0.04004934 -0.057458236 -0.050776943 -0.002810722  0.02717538
## 241336 -0.05845218 -0.037322022  0.112420133  0.039589213  0.04954880
## 262799 -0.05605130  0.012730095  0.007986937 -0.031732097 -0.03745082
```

```
##             [,1]        [,2]        [,3]        [,4]        [,5]
## 2818  -0.04182050  0.001336952 -0.122066661 -0.022233943 -0.03634110
## 20168 -0.02017147 -0.010469273 -0.009055421 -0.050733396 -0.01787569
## 25428 -0.02302950  0.072861976 -0.020961462 -0.046093886  0.02364071
## 27886 -0.04946250 -0.051574066  0.068246680 -0.020391422 -0.06901869
## 28871 -0.05900196 -0.019960125  0.031245502  0.010523218  0.02967231
## 29051 -0.04260082  0.052894460 -0.048273480 -0.070066585 -0.01390032
## 31080 -0.04303905 -0.029322134  0.016852554 -0.017518054 -0.04281556
## 38266 -0.04623462 -0.039631574 -0.025655100 -0.038893290  0.07614499
## 42970 -0.04574691  0.047919074 -0.055075670 -0.126881787  0.05180360
## 43109 -0.04651126  0.051359106  0.075709091  0.005820344 -0.07483597
```

# compute a predicted rating for each (user,item) in testevents forn ALS

# prediction = sum( userfeatures (from u matrix) * singularvalues (d matrix) * itemfeatures (from v matrix) )

```
prats1 = apply(testevents,1,function(row) c(sum(fit1$u[row[1],] * fit1$d * fit1$v[row[2],]),
 row[3])) # row[1] ~ user, row[2] ~ item
head(t(prats1))
```

```
##                         rating
## 6  "1.03839616364428"   "2"
## 8  "-0.111377317000089" "2"
## 16 "0.686165973101163"  "3"
## 23 "2.70828463316113"   "3"
## 24 "0.584123981844604"  "2"
## 28 "0.515211685354921"  "2"
```

```
length(prats1)
```

```
## [1] 1850
```

```
df = data.frame(prats1)
dim(df)
```

```
## [1]   2 925
```

```
df = t(df)
str(testevents)
```

```
## 'data.frame':    925 obs. of  3 variables:
##  $ user  : Factor w/ 422 levels "57920","142145",..: 316 44 327 96 161 177 46 103 225 222
...
##  $ item  : Factor w/ 508 levels "2818","20168",..: 1 2 3 4 4 4 4 5 6 6 ...
##  $ rating: num  2 2 3 3 2 2 5 3 3 3 ...
```

```
testevents$prediction = df[,1]
testevents$MAE = abs(testevents$rating - as.numeric(testevents$prediction))
sum(testevents$MAE)/925
```

```
## [1] 2.662412
```

```
write.csv(testevents,"testevents_ALS.csv")
```

# compute a predicted rating for each (user,item) in testevents forn SVD

# prediction = sum( userfeatures (from u matrix) * singularvalues (d matrix) * itemfeatures (from v matrix) )

```
prats2 = apply(testevents,1,function(row) c(sum(fit2$u[row[1],] * fit2$d * fit2$v[row[2],]),
 row[3])) # row[1] ~ user, row[2] ~ item
head(t(prats2))
```

```
##                         rating
## 6  "1.08407994889929"   "2"
## 8  "0.130874412057766"  "2"
## 16 "0.346852303155659"  "3"
## 23 "-0.915493159623353" "3"
## 24 "0.264919949273675"  "2"
## 28 "0.516082881905233"  "2"
```

```
length(prats2)
```

```
## [1] 1850
```

```
df = data.frame(prats2)
dim(df)
```

```
## [1]   2 925
```

```
df = t(df)
str(testevents)
```

```
## 'data.frame':    925 obs. of  5 variables:
##  $ user      : Factor w/ 422 levels "57920","142145",..: 316 44 327 96 161 177 46 103 225
222 ...
##  $ item      : Factor w/ 508 levels "2818","20168",..: 1 2 3 4 4 4 4 5 6 6 ...
##  $ rating    : num  2 2 3 3 2 2 5 3 3 3 ...
##  $ prediction: chr  "1.03839616364428" "-0.111377317000089" "0.686165973101163" "2.7082846
3316113" ...
##  $ MAE       : num  0.962 2.111 2.314 0.292 1.416 ...
```

```
testevents$prediction = df[,1]
testevents$MAE = abs(testevents$rating - as.numeric(testevents$prediction))
sum(testevents$MAE)/925
```

```
## [1] 2.57295
```

```
write.csv(testevents,"testevents_SVD.csv")
```

# Average Mean Absolute error for ALS ans SVD along with its confusion Matrix

```
cat("Average Mean Absolute error and Confusion Matrix Using ALS \n\n")
```

```
## Average Mean Absolute error and Confusion Matrix Using ALS
```

```
preds = as.numeric(unlist(prats1))
cat("avg MAE =",avgMAE(preds))
```

```
## avg MAE = 2.662412
```

```
cat("Confusion Matrix with Threshold Like as 3 \n \n")
```

```
## Confusion Matrix with Threshold Like as 3
##
```

```
showCM(preds, like=3)
```

```
## TN=  320 FP=    0
## FN=  605 TP=    0  (total=925)
## accuracy  = 34.6%
## precision = NaN%
## recall    = 0.0%
```

```
cat("Confusion Matrix with Threshold Like as 2 \n \n \n")
```

```
## Confusion Matrix with Threshold Like as 2
##
##
```

```
showCM(preds, like=2)
```

```
## TN=  125 FP=    3
## FN=  769 TP=   28  (total=925)
## accuracy  = 16.5%
## precision = 90.3%
## recall    = 3.5%
```

```
cat("\n \nAverage Mean Absolute error and Confusion Matrix Using SVD \n\n")
```

```
##
##
## Average Mean Absolute error and Confusion Matrix Using SVD
```

```
preds = as.numeric(unlist(prats2))
cat("avg MAE =",avgMAE(preds))
```

```
## avg MAE = 2.57295
```

```
cat("Confusion Matrix with Threshold Like as 3 \n \n")
```

```
## Confusion Matrix with Threshold Like as 3
##
```

```
showCM(preds, like=3)
```

```
## TN=  320 FP=    0
## FN=  605 TP=    0  (total=925)
## accuracy  = 34.6%
## precision = NaN%
## recall    = 0.0%
```

```
cat("Confusion Matrix with Threshold Like as 2 \n \n")
```

```
## Confusion Matrix with Threshold Like as 2
##
```

```
showCM(preds, like=2)
```

```
## TN=  126 FP=    2
## FN=  780 TP=   17  (total=925)
## accuracy  = 15.5%
## precision = 89.5%
## recall    = 2.1%
```