Hybrid Recommendation using Item Based CF and Model Based Technique called SVD (Singular Value Decomposition

Mritunjay And Sunil

17/10/2019

Hybrid Recommended System Techniques on Airbnb (Amsterdam Hotel Recommendation) by combining the best results of below two Techniques:-

1. Item Based CF using Cosine Similarity

2. Model Based Filtering using SVD (technique)Singular Value Decomposition)

Load All User Defined functions

```
# Make recommendations for the target user using User-based CF
getrecommendations_UUB <- function(targetuser, users, topN=5, simfun=jacardsim) {
  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(unseenrats) sum
(sims*unseenrats,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
}

# get recommedations for the target user using Item-based CF
getrecommendations_IIB <- function(targetuser, itemsims, topN=5) {
  targetuser = targetuser[colnames(itemsims)] # ensure the item order is the same
  seenitems  = !is.na(targetuser)
  unseenitems = is.na(targetuser)
  seenrats = targetuser[seenitems]
  #not much difference between below two options
  preds = apply(itemsims[unseenitems,seenitems, drop=FALSE], 1, function(sims) my.weighted.me
an(sims, seenrats))
  #preds = apply(itemsims[unseenitems,seenitems, drop=FALSE], 1, function(sims) sum(sims*seen
rats,na.rm=TRUE))
  s = sort(preds[!is.na(preds)] , decreasing = TRUE)
  s[1:min(topN,length(s))]  # get topN items
}

# evaluate recomendations (if trainusers != NULL then do User-based CF else do Item-based CF)
# computes #testitems in topN recommendations (hits) for each testuser across a set of hold-o
ut testitems
evalrecs = function(testusers, trainusers=NULL, itemsims=NULL, numtestitems=10, random=FALSE,
topN=3, simfun=jacardsim) {
  res = sapply(1:nrow(testusers),function(i) {
    cat(".")
    testuserI(testusers[i,],trainusers=trainusers,itemsims=itemsims,numtestitems=numtestitem
s,random=random,topN=topN,simfun=simfun)})
  colnames(res) = rownames(testusers)
  res
}

# may give inaccurate results if testuser is in trainusers (trainuser ratings on testitem are
not hidden)
testuserI <- function(testuser, trainusers=NULL, itemsims=NULL, numtestitems=10, random=FALSE
, topN=3, simfun=jacardsim) {
  seenitemnames   = names(testuser)[!is.na(testuser)]
  unseenitemnames = names(testuser)[is.na(testuser)]  # may be null
  if (random) testitemnames = sample(seenitemnames,min(numtestitems,length(seenitemnames))) #
test random N items
  else testitemnames = seenitemnames[1:min(numtestitems,length(seenitemnames))] # test first
 N items

  recs = ranks = list()
  rand = is.null(trainusers) & is.null(itemsims)
  for (testitemname in testitemnames) {
    truerating = testuser[testitemname]
    testuser[testitemname] = NA
    unseenitems = c(testitemname, unseenitemnames)
    if (!is.null(trainusers)) {
      # user-based CF
      usersims = apply(trainusers,1,function(trainuser) simfun(trainuser,testuser))
      usersims = usersims[!is.na(usersims) & usersims >=0]
      uitemsims = apply(trainusers[names(usersims),unseenitems,drop=FALSE],2,function(itemrat
s) sum(usersims*itemrats,na.rm=TRUE))
    }
    else if (!is.null(itemsims)) {
      # item-based CF
      seenitems = setdiff(seenitemnames, testitemname)
      seenrats = testuser[seenitems]
      uitemsims = apply(itemsims[unseenitems,seenitems,drop=FALSE],1,function(sims) my.weight
ed.mean(sims,seenrats))
    }
    else {
      # random prediction
      topNitems = sample(unseenitems,min(topN,length(unseenitems)))
      recs = c(recs,as.integer(is.element(testitemname,topNitems)))
    }
    if(!rand) {
      names(uitemsims) = unseenitems
      ssims = sort(uitemsims[!is.na(uitemsims)], decreasing = TRUE)
      ssims = ssims[1:min(topN,length(ssims))]
      res = as.integer(is.element(testitemname,names(ssims))) # test if the testitem is in th
e topN recommendations
      #res = paste(testitemname, paste(names(ssims[1:5]),collapse=",") ) # output item names
 only
      recs = c(recs,res)

      rk  = rank(uitemsims, na.last=NA)   # removes NA's
      rkpc = ((length(rk) - rk[testitemname] + 1)*100)/length(rk)
      ranks = c(ranks,rkpc)
    }
    testuser[testitemname] = truerating # restore the actual rating
  }
  # ensure outut is fixed length array
  if (length(recs)==0)  m1 = matrix(NA,numtestitems)
  else {
    m1 = as.matrix(recs)
    if (length(m1) < numtestitems) for (i in (length(m1)+1):(numtestitems)) {m1=rbind(m1,NA)}
  }
  if (length(ranks)==0) m2 = matrix(NA,numtestitems)
  else {
    m2 = as.matrix(ranks)
    if (length(m2) < numtestitems) for (i in (length(m2)+1):(numtestitems)) {m2=rbind(m2,NA)}
  }
  return(cbind(m1,m2))
}

meanHR = function(recs) {mean(unlist(recs[1:nrow(recs)/2,]),na.rm=TRUE)}
meanPR = function(recs) {mean(unlist(recs[(nrow(recs)/2+1):nrow(recs),]),na.rm=TRUE)}



# 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
}

# 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
}

# 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)
}

# 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)
}

# 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
}
```

```
# 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)),"%")}

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

# 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, MASS,relaimpo, car,interplot, caTools, mice, gbm, reshape2, compiler, recommenderlab,
Matrix, knitr,tidyr, dplyr, softImpute)
getwd()
```

```
## [1] "C:/Users/Rapsy/Desktop/Recommender_Assignment/MJ/Hybrid 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

```
#airbnb$Hotel_Id = as.factor(airbnb$Hotel_Id)
# airbnb$User_Id = as.factor(airbnb$User_Id)
#airbnb$Hotel_Id = as.character(airbnb$Hotel_Id)
#length(unique(airbnb$Hotel_Id))
#airbnb$Hotel_Id = factor(airbnb$Hotel_Id,levels=c(unique(airbnb$Hotel_Id)), ordered = FALSE)
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 ...
```

Summary of Dataset

```
#summary(airbnb)
```

Create a dataset for CF from main airbnb dataset (User_ID, Hotel_ID, Ratings)

```
colnames(airbnb)
```

```
##  [1] "Hotel_Id"            "Host_Name"           "User_Id"
##  [4] "User_Name"           "Hotel_name"          "summary"
##  [7] "space"               "description"         "host_id"
## [10] "host_name"           "property_type"       "room_type"
## [13] "accommodates"        "bathrooms"           "bedrooms"
## [16] "beds"                "bed_type"            "amenities"
## [19] "cancellation_policy" "Ratings"
```

```
airbnbCF = airbnb[,c("User_Id","Hotel_Id","Ratings")]
head(airbnbCF,4)
```

```
##   User_Id Hotel_Id Ratings
## 1 2914515     2818       3
## 2 5711109     2818       2
## 3 2944771     2818       5
## 4 4620679     2818       3
```

Unique User and Hotel

```
length(unique(airbnbCF$User_Id))
```

```
## [1] 2932
```

```
length(unique(airbnbCF$Hotel_Id))
```

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

```
dim(airbnbCF)
```

```
## [1] 20677     3
```

Removing all those users corresponding to missing ratings and

Extract only the explicit ratings and visualize the histogram of Ratings

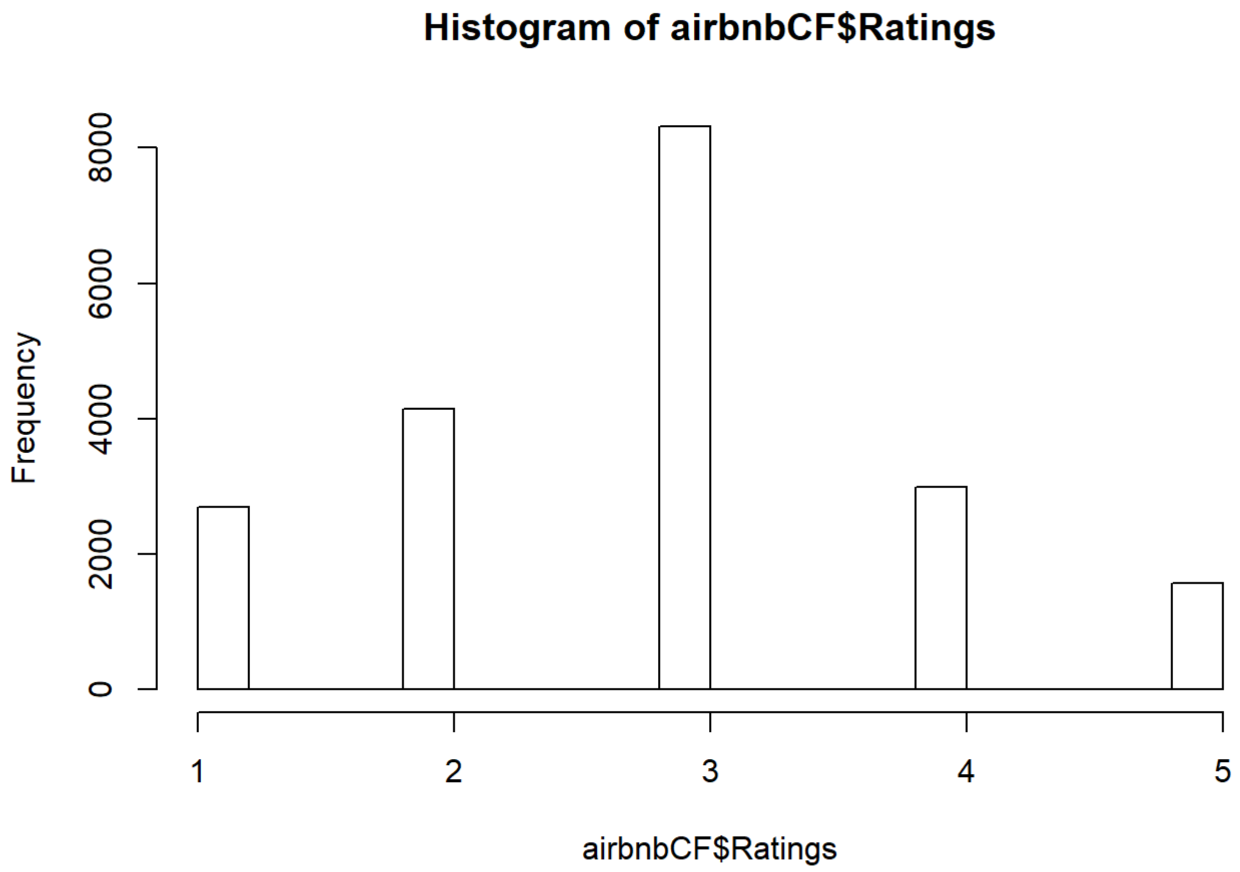
```
sapply(airbnbCF, function(x){sum(is.na(x))})
```

```
##  User_Id Hotel_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))})
```

```
##  User_Id Hotel_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:
##  $ User_Id : int  2944771 2847616 2807294 4489932 5461945 4380449 4644013 913549 5039682 4
017740 ...
##  $ Hotel_Id: int  2818 2818 2818 2818 2818 2818 2818 20168 20168 20168 ...
##  $ 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:
##  $ User_Id : int  2944771 2847616 2807294 4489932 5461945 4380449 4644013 913549 5039682 4
017740 ...
##  $ Hotel_Id: int  2818 2818 2818 2818 2818 2818 2818 20168 20168 20168 ...
##  $ Ratings : num  5 2 4 3 3 2 5 2 4 2 ...
```

Item Based Collaborative Filtering

Convert the dataframe from long to wide format

```
users_IBCF = acast(ev_Final, User_Id ~ Hotel_Id, value.var = "Ratings")
users_IBCF = sweep(users_IBCF, 1, rowMeans(users_IBCF, na.rm=TRUE) )  # normalise the data
dim(users_IBCF)
```

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

Check the sparsity and fill rate of the matrix

```
fillrate(users_IBCF)
```

```
## 2.155558 %
```

setup the train/test scheme

```
numtestusers = 84
test  = sample(rownames(users_IBCF), min(numtestusers,nrow(users_IBCF)))
train = setdiff(rownames(users_IBCF),test)
```

compute the item similarity matrix

Cosine Similarity

```
st=Sys.time(); item_cosine_sims = getitemsimsmatrix(users_IBCF[train,], simfun=cosinesim); Sy
s.time()-st
```

```
## Time difference of 3.85266 secs
```

```
cat("Fill rate for cosine similarity : "); fillrate(item_cosine_sims); cat("\n\n");
```

```
## Fill rate for cosine similarity :
```

```
## 12.60308 %
```

test IBCF Using Cosine similarity

```
preds = predictCF(users_IBCF[test,], itemsims=item_cosine_sims, numtestitems=10, random=FALSE
)
```

```
## ....................................................................................
```

```
cat("avg MAE =",avgMAE(preds), "from", validcnt(listpreds(preds)),"tests")
```

```
## avg MAE = 1.064312 from 592 tests
```

Recommendation for a user - 57920, using Item - Based

Recommendation using Cosine similarity for Item Based

```
target = users_IBCF[rownames(users_IBCF)[1],]
getrecommendations_II(target, item_cosine_sims)
```

```
## 107195 171631 182839 543930 675673
##   2.25   2.25   2.25   2.25   2.25
```

```
Top1_Hotel = integer(nrow(users_IBCF))
Top2_Hotel = integer(nrow(users_IBCF))
Top3_Hotel = integer(nrow(users_IBCF))
Top4_Hotel = integer(nrow(users_IBCF))
Top5_Hotel = integer(nrow(users_IBCF))

for (i in 1:nrow(users_IBCF)) {
  target = users_IBCF[rownames(users_IBCF)[i],]
  cfib = getrecommendations_II(target, item_cosine_sims)
  Top1_Hotel[i] = names(cfib[order(cfib,decreasing=TRUE)[1]])
  Top2_Hotel[i] = names(cfib[order(cfib,decreasing=TRUE)[2]])
  Top3_Hotel[i] = names(cfib[order(cfib,decreasing=TRUE)[3]])
  Top4_Hotel[i] = names(cfib[order(cfib,decreasing=TRUE)[4]])
  Top5_Hotel[i] = names(cfib[order(cfib,decreasing=TRUE)[5]])
}

df_cfib <- data.frame(Top1_Hotel, Top2_Hotel, Top3_Hotel, Top4_Hotel, Top5_Hotel, stringsAsFa
ctors = TRUE)
rownames(df_cfib) = rownames(users_IBCF)
write.csv(df_cfib, file = "Item Based Collaborative Recommended Hotel For each user using cos
ine.csv")
```

Singular Value Decomposition

Here We are using SVD as its giving lowest MEA value

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/ 508 levels "2818","20168",..: 1 1 1 1 1 1 1 2 2 2 ...
##  $ item  : Factor w/ 422 levels "57920","142145",..: 194 183 177 327 400 316 341 44 372 28
1 ...
##  $ 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]
```

```
##       57920 142145 186729 187580 195580 195859 201541 216385 241336 262799
## 2818     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
## 20168    NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
## 25428    NA     NA     NA      2     NA     NA     NA     NA     NA     NA
## 27886    NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
## 28871    NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
## 29051    NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
## 31080    NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
## 38266    NA     NA     NA     NA     NA      4     NA     NA     NA     NA
## 42970    NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
## 43109    NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
##       270282 294984 306166 312863 358394
## 2818      NA     NA     NA     NA     NA
## 20168     NA     NA     NA     NA     NA
## 25428     NA     NA     NA     NA     NA
## 27886     NA     NA     NA     NA     NA
## 28871     NA     NA     NA     NA     NA
## 29051     NA     NA     NA     NA     NA
## 31080     NA     NA     NA     NA     NA
## 38266     NA     NA     NA     NA     NA
## 42970     NA     NA     NA     NA     NA
## 43109     NA     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="svd") # for comparison
```

take a look at the factorised matrixes

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

```
## [1] 508  30
##              [,1]        [,2]         [,3]         [,4]         [,5]
##  [1,] -0.04475873  0.12017246 -0.007039607 -0.010651696 -0.035567491
##  [2,] -0.02150215  0.01154478 -0.016469515 -0.046507252  0.024907211
##  [3,] -0.02355918  0.02432668 -0.089597401 -0.012115129  0.023582060
##  [4,] -0.04594398 -0.07285134  0.043939168 -0.045887321 -0.045629093
##  [5,] -0.05920806 -0.02551149  0.014819637  0.004292923  0.040589427
##  [6,] -0.04466338  0.06175403 -0.065064862 -0.072596013 -0.002302032
##  [7,] -0.04560249 -0.01524386  0.018896121 -0.036924806  0.011044849
##  [8,] -0.04307068  0.03174469  0.036559393 -0.029751956  0.060542290
##  [9,] -0.04370861  0.06263092 -0.077062706 -0.075726859  0.063145883
## [10,] -0.04452110 -0.09703427 -0.047906426  0.020369450 -0.069777818
```

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

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

```
##              [,1]         [,2]        [,3]         [,4]         [,5]
##  [1,] -0.03725447  0.067141652  0.01111318  0.011045977 -0.023309932
##  [2,] -0.04347560  0.009987232  0.04190138  0.037372082  0.003017587
##  [3,] -0.03422910 -0.039366190 -0.05890214  0.054348281  0.052738516
##  [4,] -0.02987382  0.009827514 -0.01613537 -0.063466462  0.016819616
##  [5,] -0.04539351 -0.052523659 -0.04701724 -0.001900315 -0.033409965
##  [6,] -0.03422805  0.056553156 -0.02716743 -0.009804949 -0.008442109
##  [7,] -0.02527500 -0.015766107 -0.05478224  0.059622299  0.023929926
##  [8,] -0.04273353  0.029403376  0.02725478  0.019536186  0.060411121
##  [9,] -0.06140685 -0.099977330  0.05445051  0.009623139  0.042287817
## [10,] -0.05408868 -0.009945638 -0.01282545 -0.044388372 -0.029709999
```

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

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

```
## [1] 127.85612  85.70825  84.94799  84.04881  83.18348  82.72060
```

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

```
trainuserscompleted1 = complete(trainusers, fit1)
dim(trainuserscompleted1)
```

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

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

Combining the result of Item Based Collaborative filtering and Model Based Filtering

Recommendating the first 3 top hotels of each models to the respective users.

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

```
##              2818       20168        25428       27886      28871
## 57920   1.4548782 -0.66372371 -0.736279166 -0.08819728  0.8754661
## 142145  0.2149920  0.19745554 -0.035277509 -0.48685324  0.7127859
## 186729  0.2777827 -0.01881564  0.004037607  0.31368199  1.4327740
## 187580  0.2383450  0.67232480  2.000000000 -0.48181071  0.2975830
## 195580 -0.4957390 -0.02402794  0.057984810  0.22486722 -0.6963341
## 195859 -0.4912095 -0.53667394 -0.085172336 -1.27942264 -0.4866281
## 201541  2.1714546 -0.79960993  0.818631727 -0.88032213  1.1247541
## 216385  0.2471816  1.13972111 -0.138140302  0.95821431 -0.4430417
## 241336 -1.5523491 -0.54548742 -0.788635015 -0.08978671  1.6039974
## 262799  0.1686813 -0.12327020  0.220087698  0.96264460 -0.1153244
##             29051      31080       38266      42970      43109
## 57920   1.3150124  0.1706867 -0.06158830  1.3105778 -0.2048360
## 142145  0.9898728  1.0300185  0.12249664 -0.1860413 -0.3440962
## 186729 -0.4830289  0.7386340 -0.11514379  2.2093327  0.5211194
## 187580  0.3300409 -0.4256023  0.65598179  1.3287113  1.0070207
## 195580  0.7255458 -0.3135304 -0.03185019 -0.8608198  0.7025585
## 195859  1.0528820 -0.3978179  0.15655065 -0.7479730 -0.3562201
## 201541 -0.4922477  0.7635531  0.87737190  1.6387083 -0.5178038
## 216385 -0.1002685  0.7469134  1.93119293  0.6020303  0.5394158
## 241336 -0.7996207  0.6623985 -0.87321625 -0.7734142  0.6793636
## 262799  0.9762305  0.2689816  0.19483020  1.2717994  0.2603851
```

```
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]])
}

df_svd <- data.frame(Top1_Hotel, Top2_Hotel, Top3_Hotel, Top4_Hotel, Top5_Hotel, stringsAsFactors = TRUE)
rownames(df_svd) = colnames(users)
write.csv(df_svd, file = "Model Based Recommended Hotel For each user using SVD.csv")

Hybrid_Top1_Hotel = integer(nrow(outcome))
Hybrid_Top2_Hotel = integer(nrow(outcome))
Hybrid_Top3_Hotel = integer(nrow(outcome))
Hybrid_Top4_Hotel = integer(nrow(outcome))
Hybrid_Top5_Hotel = integer(nrow(outcome))
Hybrid_Top6_Hotel = integer(nrow(outcome))

for (i in 1:nrow(outcome)) {
  a = as.matrix(outcome[i,])[1,]
  Hybrid_Top1_Hotel[i] = names(a[order(a,decreasing=TRUE)[1]])
  Hybrid_Top2_Hotel[i] = names(a[order(a,decreasing=TRUE)[2]])
  Hybrid_Top3_Hotel[i] = names(a[order(a,decreasing=TRUE)[3]])
  target = users_IBCF[rownames(users_IBCF)[i],]
  cfib = getrecommendations_II(target, item_cosine_sims)
  Hybrid_Top4_Hotel[i] = names(cfib[order(cfib,decreasing=TRUE)[1]])
  Hybrid_Top5_Hotel[i] = names(cfib[order(cfib,decreasing=TRUE)[2]])
  Hybrid_Top6_Hotel[i] = names(cfib[order(cfib,decreasing=TRUE)[3]])
}

df_final = data.frame(Hybrid_Top1_Hotel,
                      Hybrid_Top2_Hotel,
                      Hybrid_Top3_Hotel,
                      Hybrid_Top4_Hotel,
                      Hybrid_Top5_Hotel,
                      Hybrid_Top6_Hotel, stringsAsFactors = TRUE)
rownames(df_final) = colnames(users)
write.csv(df_final, file = "Hybrid Recommended Hotel For each user using SVD and Item based Collaborative.csv")
trainuserscompleted1 = t(trainuserscompleted1)
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] 2.56936
```

The End