Project 4: Collaborative Filtering - Group 6

Shiqi Duan, Saaya Yasuda

2017/11/29

Contents

Project 4: Collaborative Filtering - Group 6

This file is currently a template for implementing collaborative filtering algorithms. We implement the methods on a two datasets, Microsoft Web data, and EachMovie, provided by the project instructors. The two datasets are splited into a training set and d test set by the intructors.

Step 0: Load the packages, specify directories

```
packages.used=c("pROC","matrixStats","dplyr","lsa")
packages.needed=setdiff(packages.used, intersect(installed.packages()[,1], packages.used))
if(length(packages.needed)>0){
  install.packages(packages.needed, dependencies = TRUE)
}
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(matrixStats)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:matrixStats':
##
##     count
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(lsa)
```

```
## Loading required package: SnowballC
```

```
setwd("~/Documents/GitHub/fall2017-project4-fall2017-proj4-grp6/doc")
# here replace it with your own path or manually set it in RStudio
# to where this rmd file is located
```

Step 1: Load and process the data

For each record in the dataset, there are some information we want to extract and store them in a regular matrix form: users id in rows, items id in columns, and each element in the matrix represents the rating score. As the two original datasets are in different forms, we use two processing method to preprocess the two datasets.

```
source("../lib/process.R")

process1 <- FALSE    # if TRUE, then process dataset1 in this main file
process2 <- FALSE    # if TRUE, then process dataset2 in this main file

# Prepocess dataset1
if(process1){
  train<-read.csv("../data/dataset1/data_train.csv", header = T)
  test<-read.csv("../data/dataset1/data_test.csv", header = T)
  train<-train[,-1]
  test<-test[,-1]
  ### convert data into matrix
  x<-train[train$V1=="V",]
  x<-x[order(x$V2),]
  attr<-unique(x$V2) # get all the attributes

  train_matrix<-process_dataset1(train)
  write.csv(train_matrix,"../output/dataset1_train.csv")
  test_matrix<-process_dataset1(test)
  write.csv(test_matrix,"../output/dataset1_test.csv")
}

# Prepocess dataset2
```

```
if(process2){
  train <- read.csv("../data/dataset2/data_train.csv",header=T)
  test<-read.csv("../data/dataset2/data_test.csv",header=T)
  movie <- unique(train$Movie)
  c <-length(movie)

  train_matrix <- process_dataset2(df = train)
  write.csv(train_matrix,"../output/dataset2_train.csv")
  test_matrix <- process_dataset2(df = test)
  write.csv(test_matrix,"../output/dataset2_test.csv")
}
# load processed dataset1
train1 <- read.csv("../output/dataset1_train.csv",header=T)
test1 <- read.csv("../output/dataset1_test.csv",header=T)
# load processed dataset2
train2 <- read.csv("../output/dataset2_train.csv",header=T)
test2 <- read.csv("../output/dataset2_test.csv",header=T)
```

Step 2: Algorithm design

Algorithm1: Cluster Model

In Cluster Model, we only use it on dataset2, EachMovie. We assume there are C clusters and each user belongs to one cluster c. We use EM algorithm on training set to find the probability (mu) that each user belongs to a certain cluster and the probability (gamma) that in a certain cluster a user will rate a movie with a certain score. Then use the parameters mu and gamma to predict the scores in the test set.
To determine the number of clusters to use in EM algorithm, we first run a 5-fold cross validation on the training set to get the cluster number with minimum cv-error.

```
# load processed dataset2
#train2 <- read.csv("../output/dataset2_train.csv",header=T)
#test2 <- read.csv("../output/dataset2_test.csv",header=T)

rownames(train2) <- train2[,1]
train <- train2[,-1]
rownames(test2) <- test2[,1]
test <- test2[,-1]

M <- ncol(train)
N <- nrow(train)
movie <- colnames(train)
user <- rownames(train)

## functions on EM algorithm and Score Estimation based on results from EM
# EM
cluster_model <- function(df, C, tau){
  #Input: dataframe to train, number of classes, threshold to determine convergence
  #Output: parameters for cluster models:
  #        mu: probability of belonging to class c
  #        gamma: probability of scores for a movie given the class

  ## Step1: initialize the parameters
```

```r
set.seed(2)
mu <- runif(C)
mu <- mu/sum(mu)
gamma <- array(NA,c(M,C,6)) #each list represents a class
#the ijTh element means the probability of rating jth movie with score i in that class
for(c in 1:C){
  for(m in 1:M){
    gamma[m,c,] <- runif(6)
    gamma[m,c,] <- gamma[m,c,]/sum(gamma[m,c,])
  }
}

w <- array(0, c(M,N,7))
for(k in 1:6){
  w[,,k] <- ifelse(t(df)==k, 1, 0)
  w[,,k] <- ifelse(is.na(w[,,k]),0,w[,,k])
  w[,,7] <- w[,,7] + w[,,k]
}

mu_new <- mu
gamma_new <- gamma
#expect <- matrix(0, N, C) # expectation with rows meaning classes and columns meaning users

## Iterations based on the stop criterion
threshold1 <- 9999
threshold2 <- 9999
threshold1_new <- 0
threshold2_new <- 0
count <- 0
while((threshold1 > tau | threshold2 >tau)&(abs(threshold1-threshold1_new) > tau | abs(threshold2-thr
  count <- count + 1
  print(paste0("iteration = ", count))

  threshold1_new <- threshold1
  threshold2_new <- threshold2

  mu <- mu_new
  gamma <- gamma_new

  ## Step2: Expectation
  phi <- matrix(0, C, N)
  for(k in 1:6){
    phi <- phi + t(log(gamma[,,k]))%*%w[,,k]
  }
  phi <- phi-rep(colMeans(phi),each=C)

  for(c in 1:C){
    phi[c,] <- mu[c]*exp(phi)[c,]
  }
  phi <- ifelse(phi == rep(colSums(phi),each=C), 1, phi/rep(colSums(phi),each=C))

  ## Step3: Maximization
  mu_new <- rowSums(phi)/N  #update mu
```

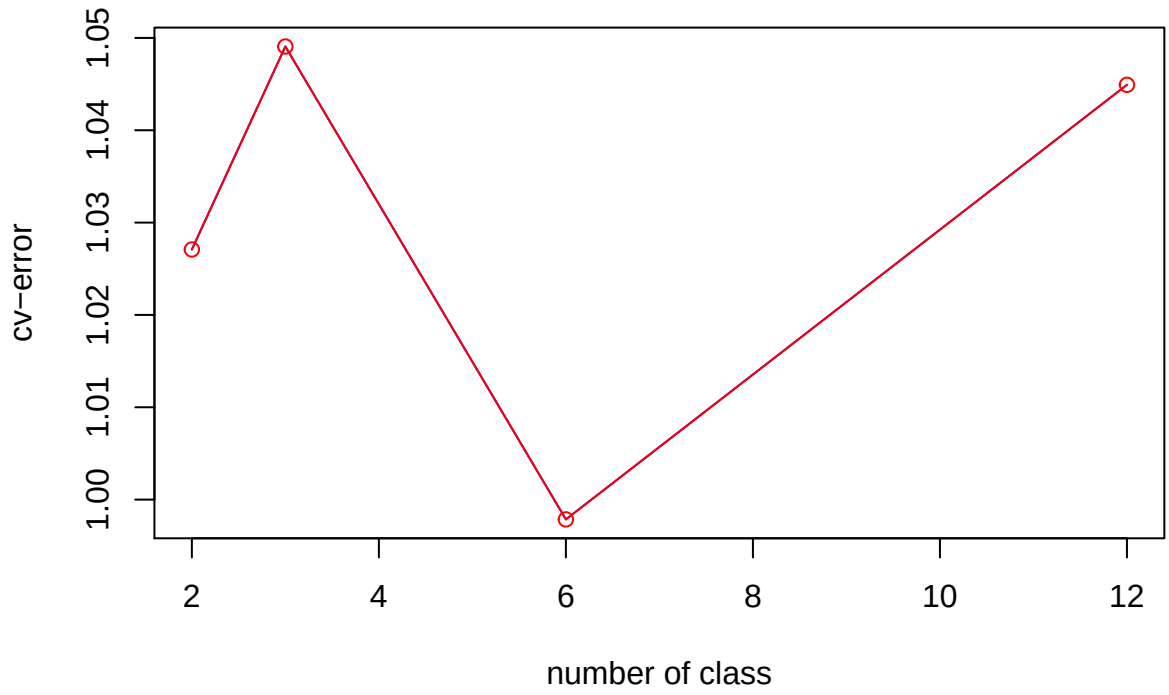
```
    for(k in 1:6){
      gamma_new[,,k] <- w[,,k]%*%t(phi)/w[,,7]%*%t(phi) #update gamma
    }
    gamma_new[gamma_new == 0] <- 10^(-100)
    if(sum(is.na(gamma_new)) != 0){
      is_zero <- which(is.na(gamma_new))
      gamma_new[is_zero] <- rep(1/6, length(is_zero))
    }

    ## Check convergence
    threshold1 <- mean(abs(mu_new - mu)) #mean absolute difference of mu
    threshold2 <- 0
    for(c in 1:C){
      threshold2 <- max(threshold2,norm(as.matrix(gamma_new[,c,] - gamma[,c,]), "O"))# matrix 1-norm of
    }
    print(paste0("threshold1 = ", threshold1, " threshold2 = ", threshold2))
  }
  return(list(mu = mu, gamma = gamma))
}

# Score Estimation
score_estimation_CM <- function(train_df, test_df, par){

  ###Input: dataframe of training set, test set, parameter list
  ###Output: estimated score
  set.seed(2)
  mu <- par$mu
  gamma <- par$gamma
  C <- length(mu)

  w <- array(0, c(M,N,7))
  for(k in 1:6){
    w[,,k] <- ifelse(t(train_df)==k, 1, 0)
    w[,,k] <- ifelse(is.na(w[,,k]),0,w[,,k])
  }
  w[,,7] <- ifelse(!is.na(t(test_df)), 1, 0)

  ##probability by Naive Bayes formula
  prob <- array(0,c(N,M,7))
  prob_mu <- matrix(mu, N, C, byrow = TRUE)
  phi <- matrix(0, C, N)
  for(k in 1:6){
    phi <- phi + t(log(gamma[,,k]))%*%w[,,k]
  }

  phi <- exp(phi)

  den <- matrix(diag(prob_mu%*%phi), N, M, byrow=FALSE)   #denominater in equation (2) of cluster model


  for(k in 1:6){
    print(paste0("k = ", k))
```

```
    num <- (t(phi)*prob_mu)%*%t(gamma[,,k]) #numerator in equation (2) of cluster model notes
    prob[,,k] <- ifelse(num==den & num == 0, runif(1)/6, num/den)
    prob[,,7] <- prob[,,7] + k*prob[,,k]
  }
  return(prob[,,7]*t(w[,,7]))
}

## 5-fold cross-validation
run.cm = FALSE #if TRUE, then run 5-fold cross-validation on training set of EachMovie
c_list <- c(2,3,6,12) #cluster numebrs we would like to try

if(run.cm){
  set.seed(2)
  K <- 5
  n <- ncol(train)
  n1 <- nrow(train)
  n.fold <- floor(n/K)
  n1.fold <- floor(n1/K)
  s <- sample(rep(1:K, c(rep(n.fold, K-1), n-(K-1)*n.fold)))
  s1 <- sample(rep(1:K, c(rep(n1.fold, K-1), n1-(K-1)*n1.fold)))

  validation_error <-  matrix(NA, K, length(c_list))

  train_df <- data.frame(matrix(NA, N, M))
  colnames(train_df) <- movie
  rownames(train_df) <- user

  test_df <- data.frame(matrix(NA, N, M))
  colnames(test_df) <- movie
  rownames(test_df) <- user

  for(i in 1:K){
    train_df[s1 != i, ] <- train[s1 != i, ]
    train_df[s1 == i, s != i] <- train[s1==i, s != i]
    test_df[s1 == i,s == i] <- train[s1 == i ,s == i]
    estimate_df <- test_df

    for(c in 1:length(c_list)){

      cm_par <- cluster_model(df = train_df, C = c_list[c], tau = 0.05)
      estimate_df <- score_estimation_CM(train_df = train_df, test_df = test_df, par = cm_par)
      validation_error[i,c] <- sum(abs(estimate_df-test_df),na.rm = T)/sum(!is.na(estimate_df-test_df))
  }
}
  save(validation_error, file=paste0("../output/validation_err.RData"))
  validation_error
}
if(!run.cm){
  load("../output/validation_err.RData")
  validation_error
}
```

```
##          [,1]     [,2]      [,3]     [,4]
## [1,] 1.027688 1.060630 0.9934067 1.049292
```

```
## [2,] 1.024357 1.046646 0.9970300 1.044445
## [3,] 1.032995 1.052055 1.0041185 1.049380
## [4,] 1.031121 1.049346 1.0030223 1.046118
## [5,] 1.019260 1.036678 0.9917086 1.035373
```

```r
# plot mean cv-error
cv_error<-colMeans(validation_error)
plot(c_list,cv_error,xlab="number of class",ylab="cv-error",col="blue",type="l")
points(c_list,cv_error,col="red",type="o")
```




```r
# find the best cluster number
class = c_list[which.min(cv_error)]
print(paste("Best class number is", class))
```

```
## [1] "Best class number is 6"
```

```r
# use best class number to find parameters in EM and predict the rates of movies in test set
run.best_par = FALSE #if TRUE, the rum the EM algorithm to find best parameters
if(run.best_par){
  best_par <- cluster_model(df = train, C = class, tau = 0.01)
  save(best_par, file=paste0("../output/best_par.RData"))
}else{
  load("../output/best_par.RData")
  best_par = best_par
}
estimate<-score_estimation_CM(train_df = train, test_df = test, par = best_par)
```

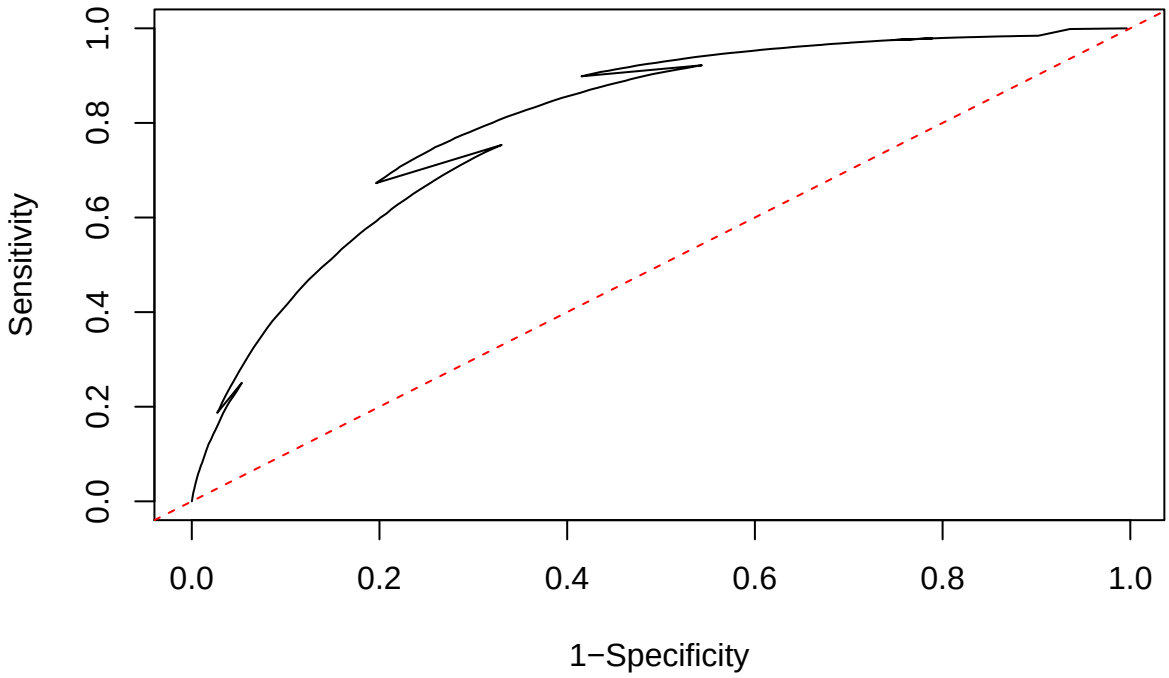
```
## [1] "k = 1"
```

```
## [1] "k = 2"
## [1] "k = 3"
## [1] "k = 4"
## [1] "k = 5"
## [1] "k = 6"
```

```
# MAE of EM algorithm
error_em<- sum(abs(estimate-test),na.rm = T)/sum(!is.na(estimate-test))
print(paste0("THE MAE of cluster model is ", error_em))
```

```
## [1] "THE MAE of cluster model is 1.0087992212154"
```

```
# ROC of EM algorithm
run.emroc = FALSE #if TRUE, run specificity and sensitivity calculation for EM
if(run.emroc){
  threshold <- seq(1,6,by=0.02)
  sensitivity <- rep(0, length(threshold))
  specificity <- rep(0,length(threshold))
  for(i in 1:length(threshold)){
    print(paste("threshold = ", threshold[i]))
    t <- threshold[i]
    test_t <- ifelse(test>=t,1,0)
    estimate_t <- ifelse(estimate>=t,1,0)
    sensitivity[i] <- sum(test_t == 1 & estimate_t == 1, na.rm=T)/sum(test_t == 1, na.rm=T)
    specificity[i] <- sum(test_t == 0 & estimate_t == 0, na.rm=T)/sum(test_t == 0, na.rm=T)
  }
  save(sensitivity,file=paste("../output/em_sensitivity.RData"))
  save(specificity,file=paste("../output/em_specificity.RData"))
}else{
  load("../output/em_sensitivity.RData")
  load("../output/em_specificity.RData")
}
plot(1-specificity,sensitivity,xlab="1-Specificity",ylab="Sensitivity", type="l")
abline(0,1, col="red", lty=2)
```




```
test_roc <- ifelse(test>=4,1,0)
estimate_roc <- ifelse(estimate>=4,1,0)
roc(as.vector(test_roc), as.vector(estimate_roc), na.rm = T)
```

```
##
## Call:
## roc.default(response = as.vector(test_roc), predictor = as.vector(estimate_roc),     na.rm = T)
##
## Data: as.vector(estimate_roc) in 68012 controls (as.vector(test_roc) 0) < 131864 cases (as.vector(tes
## Area under the curve: 0.7382
```

Algorithm2: Memory-based Algorithm

In this Algorithm, there are four components we need to consider: Similarity Weight, Variance Weighting, Selecting Neighbors, and Rating Normalization.

Functions for Algorithm2

Similarity Weight:

We use this to find the similarity correlation of different users.

These functions assume setwd("~/Documents/GitHub/fall2017-project4-fall2017-proj4-grp6/").

```
# SW1: Spearman
spearman_weight <- function(matrix){
  file=deparse(substitute(matrix))
```

```
  file_name= paste0("./output/spearman_",file,".csv")

  matrix[is.na(matrix)] = 0
  matrix = t(matrix)
  w = cor(matrix,use="everything",method="spearman")

  write.csv(w,file=file_name)
}

# SW2: Vector Similarity
vector_similarity <- function(matrix){
  library(lsa)

  file=deparse(substitute(matrix))
  file_name= paste0("./output/vectorsimilarity_",file,".csv")

  matrix[is.na(matrix)] = 0
  matrix = t(matrix)
  w = cosine(matrix)
  write.csv(w,file=file_name)
}

# SW3: Mean Squared Difference
# Reduced the computational time significantly by assigning arguments outside
# the loops, initiating the output matrix r with NA first, and using apply
# to mean() outside the loops.
mean_sq_diff <- function(matrix){

  file=deparse(substitute(matrix))
  file_name= paste0("./output/meansqdiff_",file,".csv")

  matrix[is.na(matrix)] = 0
  usermean = apply(matrix,1,mean)

  ncolrow = nrow(matrix)
  w <- matrix(rep(NA), ncolrow, ncolrow)
  rownames(w) = rownames(matrix)
  colnames(w) = rownames(matrix)

  for (r in 1:ncolrow){
    for (c in 1:ncolrow){
      if (r>c){
        w[r,c] = w[c,r]
      }
      else if(r==c){
        w[r,c] = 0
      }
      else {
        w[r,c] = (usermean[r]-usermean[c])^2
      }
    }
  }
  write.csv(w,file=file_name)
```

```
}

## Calculate SW on datasets
rownames(train1) = train1[,1]
train1 = train1[,-1]
rownames(test1) = test1[,1]
test1 = test1[,-1]

rownames(train2) = train2[,1]
train2 = train2[,-1]
rownames(test2) = test2[,1]
test2 = test2[,-1]

run.sw1 = FALSE #If TURE run Spearman function on datasets
run.sw2 = FALSE #If TURE run vector similarity function on datasets
run.sw3 = FALSE #If TURE run mean squared difference function on datasets
if(run.sw1){
  #w11 =
  spearman_weight(train1)
  #w12 =
  spearman_weight(train2)
  #write.cvs(w11,"../output/spearman_train1.csv")
  #write.cvs(w12,"../output/spearman_train1.csv")
}
if(run.sw2){
  #w21 =
  vector_similarity(train1)
  #w22 =
  vector_similarity(train2)
  #write.cvs(w21,"../output/vectorsimilarity_train1.csv")
  #write.cvs(w22,"../output/vectorsimilarity_train2.csv")
}
if(run.sw3){
  #w31 =
  mean_sq_diff(train1)
  #w32 =
  mean_sq_diff(train2)
  #write.csv(w31,"../output/meansqdiff_train1.csv")
  #write.csv(w32,"../output/meansqdiff_train2.csv")
}

# SW4: SimRank, we only nedd this method for dataset1, Microsoft Web data
run.sw4 = FALSE #if TRUE, we are going to run the whole SimRank process for dataset1
if(run.sw4){
  # convert web name
  convert_colname <- function(dataset){
    names <- colnames(dataset)
    for(i in 1:ncol(dataset)){
      names[i] <- sub("X","",names[i])
    }
    return(names)
  }
```

```
  train_sm <- train1
  test_sm <- test1
  colnames(train_sm1) <- convert_colname(train_sm)
  colnames(test_sm1) <- convert_colname(test_sm)
  linkgraph_bip <- list()
  for(i in 1:nrow(train_sm)){
    linkgraph_bip[[i]] <- colnames(train_sm[i,])[which(train_sm[i,]==1)]
  }

  n = length(linkgraph_bip)

  t_train1 <- t(train_sm)
  t_train1 <- as.data.frame(t_train1)
  for(i in 1:nrow(t_train1)){
    linkgraph_bip[[n+i]] <- colnames(t_train1[i,])[which(t_train1[i,]==1)]
  }

  length(linkgraph_bip)
  nods_name <- as.numeric(c(rownames(train_sm),colnames(train_sm)))

  length <- c()
  for(i in 1:length(linkgraph_bip)){
    length[i] = length(linkgraph_bip[[i]])
  }

  max_len <- max(length)

  # Get graph matrix
  graph_matrix <- matrix(nrow = length(linkgraph_bip), ncol = max_len+1)
  graph_matrix[,1] <- nods_name

  for(i in 1:length(linkgraph_bip)){
    graph_matrix[i,2:(1+length(linkgraph_bip[[i]]))]<-as.numeric(linkgraph_bip[[i]])
  }

  dim(graph_matrix)

###################################### Simrank Algorithm ######################################

  graph <- graph_matrix

  nods<-nods_name
  nnods <- length(nods)

  trans_matrix <- function(graph){
    trans_matrix <- matrix(0, nrow = nnods, ncol = nnods)
    rownames(trans_matrix) <- nods
    colnames(trans_matrix) <- nods
    for(i in 1:nnods){
      n = sum(is.na(graph[i,-1])==F)
      for(m in 1:n){
        loc <- which(nods == graph[i,m+1])
        trans_matrix[i,loc] <- 1/n
```

```
      }
    }
    return(trans_matrix)
  }

  trans_matrix <- trans_matrix(graph)
  #write.csv(trans_matrix, file = "D:/Github/fall2017-project4-fall2017-proj4-grp6/output/trans.csv")

  #### naive method

  sim_matrix <- matrix(0, nrow = nnods, ncol = nnods)
  rownames(sim_matrix)<-nods
  colnames(sim_matrix)<-nods
  sim_matrix_old <- sim_matrix
  diag(sim_matrix) <- 1 # initial simrank matrix with diagnal = 1
  In <- sim_matrix

  c = 0.8
  niter = 5
  sim_list <- list()

  for(i in 1:niter){
    print(c("start",i))
    ptm <- proc.time()[1]
    sim_matrix_old <- sim_matrix
    sim_matrix <- c * (t(trans_matrix) %*% sim_matrix_old %*% trans_matrix) + (1-c)*In
    sim_list[[i]] <- sim_matrix
    end <- proc.time()[1]
    print(c("end", i, end-ptm))
  }


  simrank <- sim_list[[5]]
  simrank_matrix <- simrank[1:nrow(train1),1:nrow(train1)]
  # for easier upcoming calculation
  diag(simrank_matrix) <- 1
}
```

Variance Weighting

We use this to distinguish the items with high variance of rating. As variance weighting has to take use of the results of similarity weights, so we have different functions for each similarity weighting method.

```
train2_vw <- train2
train2_vw[is.na(train2_vw)]<-0

run.vw1 = FALSE #if TRUE, run vector weighting algorithm on the datasets with Spearman
run.vw2 = FALSE #if TRUE, run vector weighting algorithm on the datasets with vector similarity
run.vw3 = FALSE #if TRUE, run vector weighting algorithm on the datasets with mean squared difference

# spearman
var_weighting_spearman <- function(matrix){
  var <- apply(matrix, 2, var)
  v <- (var-min(var))/max(var)
```

```
  matrix <- matrix * sqrt(v)
  spearman_vw <- cor(t(matrix), method = "spearman")
  return(spearman_vw)
}
if(run.vw1){
  spearman_vm_matrix1 <- var_weighting_spearman(train1)
  spearman_vm_matrix2 <- var_weighting_spearman(train2_vw)
  write.csv(vs_vw_matrix1,"../output/spearman_vm_train1.csv")
  write.csv(vs_vw_matrix2,"../output/spearman_vm_train2.csv")
}

# Vector Similarity
var_weighting_vecsim <- function(matrix){
  var <- apply(matrix, 2, var)
  v <- (var-min(var))/max(var)
  matrix <- matrix * sqrt(v)
  vs_vw <- cosine(t(matrix))
  return(vs_vw)
}
if(run.vw2){
  vs_vw_matrix1 <- var_weighting_vecsim(train1)
  vs_vw_matrix2 <- var_weighting_vecsim(train2_vw)
  write.csv(vs_vw_matrix1,"../output/vectorsimilarity_vm_train1.csv")
  write.csv(vs_vw_matrix2,"../output/vectorsimilarity_vm_train2.csv")
}

# mean squared difference
var_weighting_msd <- function(matrix){
  var <- apply(matrix, 2, var)
  v <- (var-min(var))/max(var)
  matrix <- matrix * sqrt(v)
  msd_vw <- mean_sq_diff(matrix)
  return(msd_vw)
}
if(run.vw3){
  msd_vm_matrix1 <- var_weighting_msd(train1)
  msd_vm_matrix2 <- var_weighting_msd(train2_vw)
}
```

Selecting Neighbors:

We use this to select the users that are more similar to a certain user than others.

```
train2_sn <- train2
train2_sn[is.na(train2_sn)]<-0
test2_sn <- test2
test2_sn[is.na(test2_sn)]<-0

run.sn <- FALSE #if TRUE, the run selecting neighbors algorithm on the results from above parts
if(run.sn){
  #### weight
  sp_w1 = read.csv("./output/spearman_train1.csv",header=T)
  sp_w2 = read.csv("./output/spearman_train2.csv",header=T)
  sp_vm_w1 = read.csv("./output/spearman_vm_train1.csv",header=T)
```

```
sp_vm_w2 = read.csv("./output/spearman_vm_train1.csv",header=T)
vs_w1 = read.csv("./output/vectorsimilarity_train1.csv",header=T)
vs_w2 = read.csv("./output/vectorsimilarity_train1.csv",header=T)
vs_vm_w1 = read.csv("./output/vectorsimilarity_vm_train1.csv",header=T)
vs_vm_w2 = read.csv("./output/vectorsimilarity_vm_train2.csv",header=T)

msd_w1 = read.csv("./output/meansqdiff_train1.csv",header=T)
msd_w2 = read.csv("./output/meansqdiff_train2.csv",header=T)

rownames(sp_w1) <- sp_w1[,1]
sp_w1 <- sp_w1[,-1]

rownames(sp_w2) <- sp_w2[,1]
sp_w2 <- sp_w2[,-1]

rownames(sp_vm_w1) <- sp_vm_w1[,1]
sp_vm_w1 <- sp_vm_w1[,-1]

rownames(sp_vm_w2) <- sp_vm_w2[,1]
sp_vm_w2 <- sp_vm_w2[,-1]

rownames(vs_w1) <- vs_w1[,1]
vs_w1 <- vs_w1[,-1]

rownames(vs_w2) <- vs_w2[,1]
vs_w2 <- vs_w2[,-1]

rownames(vs_vm_w1) <- vs_vm_w1[,1]
vs_vm_w1 <- vs_vm_w1[,-1]

rownames(vs_vm_w2) <- vs_vm_w2[,1]
vs_vm_w2 <- vs_vm_w2[,-1]

rownames(msd_w1) <- msd_w1[,1]
msd_w1 <- msd_w1[,-1]

rownames(msd_w2) <- msd_w2[,1]
msd_w2 <- msd_w2[,-1]

#### correlation-thresholding
cor_threshold <- function(matrix, n){
  matrix_for_cal <- matrix
  matrix_for_cal[abs(matrix_for_cal) < n] <- NA
  diag(matrix_for_cal) <- NA
  cor_thre <- list()
  for(i in 1:nrow(matrix_for_cal)){
    x <- matrix_for_cal[i,]
    cor_thre[[i]] <- which(is.na(x)==F)
    print(i) # To see the process
  }
  return(cor_thre)
}
#### best-n-neighbors
```

```
  bestnn <- function(matrix, nnbors){
    best <- list()
    for(i in 1:nrow(matrix)){
      best[[i]] <- order(matrix[i,], decreasing = T)[2:(nnbors+1)]
      print(i)
    }
    return(best)
  }
  # you can try different files into the function here, this is just an example
  bnnbors <- bestnn(spearman_matrix1, 30) # nnbors = 30

  #### combined
  list1 = bestnn(matrix, nnbors)
  list2 = cor_threshold(matrix, n)

  combined <- function(matrix){
    bnn = list1
    thresh = list2
    combine <- list()
    for(i in 1:nrow(matrix)){
      combine[[i]] <- unique(c(bnn[[i]],thresh[[i]]))
    }
    return(combine)
  }
}
```

Rating Normalization:

We use Z-score to predict scores in this component.

```
z_score<-function(df, w, neighbor_matrix){
  #Input: Rating dataset with users in row and items in columns, Weight matrix between users
  #Output: Prediction matrix with users in row and items in columns
  N <- nrow(df)
  M <- ncol(df)

  df_mean <- matrix(rep(rowMeans(df), M), N, M)
  df_sd <- matrix(rep(rowSds(as.matrix(df)), M), N, M)
  w_new <- neighbor_matrix * w
  w_sum <- matrix(rep(rowSums(w_new), M), N, M)
  df_scale <- (df - df_mean)/df_sd
  p <- df_sd * t((t(df_scale) %*% as.matrix(w_new)))/w_sum
  return(p + df_mean)
}
```

Step 3: Evaluation

There are three evaluation methods we used. We used Ranked Scoring for evaluate dataset1, MAE and ROC for evaluate dataset2. Here are the algorithm functions for these three evaluation methods. For ROC, we used the function roc in package pROC. #### MAE for dataset 2

```
# MAE
MAE <- function(pred, test){
```

```
  out = sum(abs(pred - test))/sum(test != 0)
  return(out)
}
```

Ranked Scoring functions for dataset 1

```
# Ranked Scoring - 2 functions

########################################
# Rank matrix function (helper function)
# Input: predicted matrix & test set matrix.
#        These matrices need to have user names in rownames.
# Output: return the ranked test set matrix, based on predicted vote values.
########################################

rank_matrix <- function(pred_matrix, observed_matrix){
  nrow = nrow(observed_matrix)
  ncol = ncol(observed_matrix)
  ranked_mat = matrix(NA, nrow, ncol)

  for (r in 1:nrow){
    # get username of the row
    user_name = rownames(observed_matrix)[r]

    # sort pred values
    sorted_pred = sort(pred_matrix[user_name,], decreasing=TRUE)

    # sort observed values based on pred values.
    sorted_obs = unlist( observed_matrix[user_name,][names(sorted_pred)] )

    # save the ranked row in the new matrix.
    ranked_mat[r,] = unname(sorted_obs)
  }
  rownames(ranked_mat) = rownames(observed_matrix)
  return(ranked_mat)
}

########################################
# Ranked scroing function
# Input: predicted matrix & test set matrix. Also alpha value.
#        These matrices need to have user names in rownames.
# Output: return the ranked score for the test set matrix
########################################

randked_scoring <- function(pred_matrix, observed_matrix, alpha){
  # Get ranked version of the observed_matrix
  ranked_mat = rank_matrix(pred_matrix, observed_matrix)

  nrow = nrow(ranked_mat)
  ncol = ncol(ranked_mat)
  a_minus_one = alpha-1

  # Assuming d=0, this is the numerator for r_a.
```

```
  # There's no negative value in dataset 1, but just in case.
  ranked_mat[ranked_mat<0] = 0

  # denominator of r_a & r_a_max
  denom_vec = 2^(0:(ncol-1)/a_minus_one)
  denom_mat = matrix(rep(denom_vec, nrow), nrow, ncol, byrow=T)

  # Get a vector of r_a
  utility_matrix = ranked_mat/denom_mat
  r_a_vector = rowSums(utility_matrix)

  # Rank the observed_matrix for r_a max in order to have the maximum achievable utility.
  max_numerator_matrix = t(apply(observed_matrix, 1, sort,decreasing=T))

  # Get a vector of r_a max
  max_utility_matrix = max_numerator_matrix/denom_mat
  max_r_a_vector = rowSums(max_utility_matrix)

  # Get the r_a / r_a_max score
  r = 100 * sum(r_a_vector)/sum(max_r_a_vector)

  return(r)

  }
```

For ranked score, we initially had a large alpha value which returned really high scores with little variance. We added an alpha as a function input so that we can play with different alpha values, as the paper 1 suggested (paper 1 used 5 and 10).

For the dataset 1, dataset1_prediction_evaluation.Rmd runs the prediction and evaluation. The PDF version with the output is also in the doc file.

Additional helper functions are defined in the file. Namely:

```
source('../lib/evaluation_ranked_scoring.r')
source('../lib/z_score.R')

########################################
# Helper 1: assign_rownames(dataset)
# Most datasets contain usernames in the 1st col. Assigning them as rownames.
########################################
assign_rownames = function(dataset){
  rownames(dataset) = dataset[,1]
  dataset = dataset[,-1]
  return(dataset)
}
########################################
# Helper 2: evaluate(train, test, weight, neighbor)
# Input: 4 matrices: train data, test data, weights, and neighbor matrix
# Output: ranked score
########################################
evaluate = function(train, test, weight, neighbor){
  weight = assign_rownames(weight)
  neighbor = assign_rownames(neighbor)
```

```
  #prediction
  pred = z_score(train,weight,neighbor)
  row.names(pred) = row.names(train)

  #for alpha values in ranked scoring formula:
  avg_half_life = mean(rowSums(test)) # average number of sites visited by user.
  avg_half_life = round(avg_half_life, 3)
  alpha = c(avg_half_life, 5, 10) #paper 1 used 5 & 10 for alpha in ranked scoring.

  score_vec = vector()
  for (a in alpha){
    score_vec = c(score_vec, randked_scoring(pred, test, a))
  }
  names(score_vec) = alpha
  return (score_vec)
}
```

For the alpha values, we used the average number of sites the users visited (3.087) in addition to the suggested values, 5 and 10. We were able to compare and find some differences based on the alpha values.

Please see the dataset1_prediction_evaluation.pdf for more details on how the functions were used.

When doing the evaluation, to compare different variants in each component, we read the recommended papers first. From the papers, we learn that Spearman should be better than vector similarity weighting and mean squared difference weighting, taking variance weighting will not make more significant contribution to the results, and the combination of weight-threshold and best-b-estimator will not be better than using only one selecting neighbor method from these two.

However, we do not find evidence from papers whether SimRank is better than Spearman or not, nor which one of weight-threshold and best_n_neighbor is better than the other. So for dataset2 we just used Spearman and no variance weighting when comparing selecting neighbor methods(by ROC results). Then we get the best selecting neighbor methods, best_n_neighbor, and use it to compare variants in other components. For dataset1, we used no variance weighting and best_n_neighbor(based on paper and our results from dataset2) to compare similarity weighting methods and get the best similarity weighting method. Then use it while comparing variants in other components.

Step 4: Results and Conclusions

Dataset 1 Conclusion

Best score when alpha = avg number of site visits per user: 3.087

```
alpha3 <-read.csv("../output/dataset1_result_alpha3.csv",header=T)
alpha3
```

```
##                                X   X3.087       X5      X10
## 1   VecSim with NS thresh & no VW 34.46454 42.45547 53.89660
## 2 Spearman with NS thresh & No VW 33.57993 41.65699 53.24078
## 3  Spearman with NS combo & No VW 33.57338 41.65260 53.23769
## 4    Spearman with NS thresh & VW 33.26044 41.37642 53.01921
## 5      VecSim with NS thresh & VW 32.82186 41.32847 53.27723
## 6      VecSim with NS bnn & No VW 23.58450 30.01127 40.69014
## 7    Spearman with NS bnn & No VW 23.39347 29.85521 40.55418
```

Best score when alpha = 5

```
alpha5 <-read.csv("../output/dataset1_result_alpha5.csv",header=T)
alpha5
```

```
##                                 X   X3.087       X5      X10
## 1   VecSim with NS thresh & no VW 34.46454 42.45547 53.89660
## 2 Spearman with NS thresh & No VW 33.57993 41.65699 53.24078
## 3  Spearman with NS combo & No VW 33.57338 41.65260 53.23769
## 4    Spearman with NS thresh & VW 33.26044 41.37642 53.01921
## 5      VecSim with NS thresh & VW 32.82186 41.32847 53.27723
## 6      VecSim with NS bnn & No VW 23.58450 30.01127 40.69014
## 7    Spearman with NS bnn & No VW 23.39347 29.85521 40.55418
```

Best score when alpha = 10

```
alpha10 <-read.csv("../output/dataset1_result_alpha10.csv",header=T)
alpha10
```

```
##                                 X   X3.087       X5      X10
## 1   VecSim with NS thresh & no VW 34.46454 42.45547 53.89660
## 2      VecSim with NS thresh & VW 32.82186 41.32847 53.27723
## 3 Spearman with NS thresh & No VW 33.57993 41.65699 53.24078
## 4  Spearman with NS combo & No VW 33.57338 41.65260 53.23769
## 5    Spearman with NS thresh & VW 33.26044 41.37642 53.01921
## 6      VecSim with NS bnn & No VW 23.58450 30.01127 40.69014
## 7    Spearman with NS bnn & No VW 23.39347 29.85521 40.55418
```

Based on the results we obtained, for the dataset1, Vector Similarity + No Variance Weighting + Weight Threshold was our best model to the ranked score evaluation method, for all of the alpha values. The rest of the models performed in the identical order for alpha = 3.087 and alpha = 5. However, for alpha = 10, Vector Similarity + Variance Weighting + Weight Threshold performed better than the Spearman algorithims.

In general, we found that the models with variance weighting didn't perform as well as no variance weighting. For this dataset, best_n_neighbor (bnn) particularly didn't perform well.

See the dataset1_prediction_evaluation.pdf for more information.

Dataset 2 Conclusion

```
results<-read.table("../output/results.txt",header=T)
results
```

```
##   Dataset       Similarity_Weight Variance_Weighting Selecting_Neighbor
## 1       2                Spearman                 no   weight_threshold
## 2       2                Spearman                 no    best_n_neighbor
## 3       2                Spearman                 no           combined
## 4       2                Spearman                yes    best_n_neighbor
## 5       2       vector_similarity                 no    best_n_neighbor
## 6       2 mean_squared_difference                 no    best_n_neighbor
## 7       2           cluster_model               <NA>               <NA>
##      ROC     MAE
## 1 0.7164 1.35276
## 2 0.7477 1.26598
## 3 0.7156 1.36644
```

```
## 4 0.7236 1.26230
## 5 0.7236 1.23412
## 6 0.0000 0.00000
## 7 0.7382 1.01366
```

Based on the results we obtained, we conlcude that for dataset2, Spearman + No Variance Weighting + Best_n_neighbors is the best model with respect to ROC, and Cluster Model is the best model with respect to MAE. Overall we recommend Cluster model as our best one as it has a higher ROC than other models and we consider people are more interested in whether the algorithm could predict the score of an item correctly with repect to direction(visited or not, high score 4-6 or low score 1-3). If people are more interested in whether the algorithm could predict the scores with respect to how close it is to the actual score, they should chose Spearman + No Variance Weighting + Best_n_neighbors.