Project 4 - Collaborative Filtering Algorithm

Team 6

4/18/2017

Step 0: Load the packages, and get the directories

```
getwd()
```

```
## [1] "C:/StudyLife/Columbia/STAT 5243/Project 2/Spring2018-Project4-grp6/doc"
```

Step 1: Load and process the data

Here, we load in all the data we need for this project. There are four datasets, and they are respectively the testing and training sets of Microsoft Web data and Movie data.

```
movie_train<-read.csv("../data/data_sample/eachmovie_sample/data_train.csv")
movie_test<-read.csv("../data/data_sample/eachmovie_sample/data_test.csv")

MS_train<-read.csv("../data/data_sample/MS_sample/data_train.csv")
MS_test<-read.csv("../data/data_sample/MS_sample/data_test.csv")
```

After loading the datasets, we transform the datasets so that they can be manipulated and applied in various functions later.

Here is how we transform the movie dataset.

```
Transformer <- function(data.set){
  columns.data <- sort(unique(data.set$Movie))
  rows.data <- sort(unique(data.set$User))
  Table_ <- matrix(NA,nrow = length(rows.data), ncol = length(columns.data))
  for(i in 1:length(columns.data)){
    col.name <- columns.data[i]
    index <- which(data.set$Movie == col.name)
    scores <- data.set[index,4] #Scores
    users <- data.set[index,3] #Users
    index2 <- which(rows.data %in% users)
    Table_[index2,i] = scores
    Table_[!index2,i] = NA
  }
  colnames(Table_) <- columns.data
  rownames(Table_) <- rows.data
  return(Table_)
}

m_train<-Transformer(movie_train)
m_test<-Transformer(movie_test)

## add the missing columns to test data
miss_index<-numeric(length(train.index))
for(i in 1:length(train.index)){
  miss_index[i]<-train.index[i]%in%test.index
```

```
}
m_index<-which(miss_index == 0)
train.index[m_index]
m_matrix<-matrix(NA, ncol = 22, nrow = 5055)
colnames(m_matrix)<-m_index
m_test<-cbind(m_matrix, m_test)
m_test<-m_test[, order(as.numeric(colnames(m_test)))]


write.csv(m_train, "Movie_data_train.csv")
write.csv(m_test, "Movie_data_test.csv")
```

Here is how we transform the Microsoft dataset.

```
MS.Data.Transform.Function <- function(data){
  usernum <- which(data$V1 == 'C')
  userid <- data$V2[usernum]
  vrootid <- unique(data$V2[which(data$V1 == 'V')])
  table <- matrix(0, nrow=length(userid), ncol=length(vrootid))
  rownames(table) <- as.character(userid)
  colnames(table) <- as.character(vrootid)
  for (i in 1:length(usernum)){
    start.indx <- usernum[i]
    if (i != length(usernum)){
      end.indx <- usernum[i+1]
    }
    else {
      end.indx <- nrow(data)+1
    }

    userid_mat <- as.character(userid[i])

    for (j in (start.indx+1):(end.indx-1)){
      vrootid_mat <- as.character(data$V2[j])
      table[userid_mat, vrootid_mat] <- 1
    }
  }
  return(table)
}

transformed.traindata <- MS.Data.Transform.Function(MS_train)
transformed.testdata <- MS.Data.Transform.Function(MS_test)
write.csv(transformed.traindata,file = "MS_data_train.csv")
write.csv(transformed.testdata,file = "MS_data_test.csv")
```

Step 2: Create the required functions of Memory-based Algorithm and implemented them

First of all, We created three functions for similarity weighting.

1. Spearman Correlation:

```
MS_data <- read.csv("../data/MS_data_train.csv")
```

```r
new_movie_data <- read.csv("../data/Movie_data_train.csv")

weight_Spearman <- function(data1, dataset_name) {
  write_file_name <- paste0(dataset_name ,"_Spearman_weights" , ".csv")

  transformeddata <- t(data1)

  weights <-  cor(transformeddata, method = "spearman")

  write.csv(weights, write_file_name)
}

weight_Spearman(MS_data, "MS")

weight_Spearman(new_movie_data, "Movie")
```

2. Mean-square-difference:

```r
# 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.

m_train<-as.matrix(m_train)

mean_sq_diff <- function(matrix){

  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] <- (usermean[r]-usermean[c])^2
      }
      else if(r==c){
        w[r,c] <- 0
      }
      else {
        w[c,r]<-w[r,c]
      }
    }
  }
}
```

3. SimRank:

```r
MS.train.data <- read.csv("MS_data_train.csv",header=T)
MS.test.data <- read.csv("MS_data_test.csv",header=T)

# Substring the column name
```

```r
substr.colname <- function(dataset){
  names <- colnames(dataset)
  for(i in 1:ncol(dataset)){
    names[i] <- substr(names[i], start = 2, stop = nchar(names[i]))
  }
  return(names)
}

# SimRank Function
generate.simrank.mat <- function(c = 0.8, k = 5, traindata = MS.train.data,
                                 testdata = MS.test.data){
  MS.train.data = traindata
  MS.test.data = testdata

  # Pre process the train and test data
  rownames(MS.train.data) <- MS.train.data[,1]
  MS.train.data <- MS.train.data[,-1]  # variable name are exclude from the data area
  rownames(MS.test.data) <- MS.test.data[,1]
  MS.test.data <- MS.test.data[,-1]
  colnames(MS.train.data) <- substr.colname(MS.train.data)
  colnames(MS.test.data) <- substr.colname(MS.test.data)

  # Generate the bipartite graph
  bipartite.graph <- list()
  for(i in 1:nrow(MS.train.data)){
    bipartite.graph[[i]] <- colnames(MS.train.data)[which(MS.train.data[i,]==1)]
  }
  n <- length(bipartite.graph)
  t_MS.train.data <- as.data.frame(t(MS.train.data))
  for(i in 1:nrow(t_MS.train.data)){
    bipartite.graph[[n+i]] <- colnames(t_MS.train.data)[which(t_MS.train.data[i,]==1)]
  }
  tail(bipartite.graph)
  # Nodes name = user ID + vroots
  nodes.name <- as.numeric(c(rownames(MS.train.data),colnames(MS.train.data)))

  # Construction of Graph Matrix
  graph.matrix <- matrix(0,nrow = length(nodes.name),ncol = length(nodes.name))
  matrix1 <- matrix(0,nrow = nrow(MS.train.data),ncol = nrow(MS.train.data))
  matrix3 <- t(MS.train.data)
  matrix4 <- matrix(0,nrow = ncol(MS.train.data),ncol = ncol(MS.train.data))
  matrix5 <- cbind(matrix3,matrix4)
  graph.matrix <- cbind(matrix1,MS.train.data)
  colnames(graph.matrix) <- nodes.name
  colnames(matrix5) <- nodes.name
  graph.matrix <- rbind(graph.matrix,matrix5)
  dim(graph.matrix)

  # Column-normalized matrix
  colnum <- colSums(graph.matrix)
  for (i in 1:ncol(graph.matrix)){
    graph.matrix[,i] <- graph.matrix[,i]/colnum[i]
  }
```

```
  # Iteration Mtd of SimRank
  W <- as.matrix(graph.matrix)
  simrank.matrix.pre <- matrix(0, nrow = length(nodes.name), ncol = length(nodes.name))
  diag(simrank.matrix.pre) <- 1
  simrank.matrix <- simrank.matrix.pre

  for(i in 1:k){
    simrank.matrix.pre <- simrank.matrix
    simrank.matrix <- c * (t(W) %*% simrank.matrix.pre %*% W)
    diag(simrank.matrix) <- 1
    print(i)
  }
  dim(simrank.matrix)

  # Extraction of User Part
  simrank.fin.matrix <- simrank.matrix[1:nrow(MS.train.data),1:nrow(MS.train.data)]
  dim(simrank.fin.matrix)

  return(simrank.fin.matrix)
}


simrank.fin.matrix <- generate.simrank.mat(c = 0.8, k = 5, traindata = MS.train.data,
                                           testdata = MS.test.data)
write.csv(simrank.fin.matrix, file = "simrank.fin.matrix.csv")
```

Secondly, after having the similarity weights, we write out the functions which are able to find the neighbors and make prediction for those weights. Then, we find the neighbors and make the predictions for them.

Notes: Since the datasets for the similarity weights we calculated are too large to upload onto the github, we just load all the datasets from our local directory. Those datasets, however, have been already been uploaded onto the google drive we created. You can download the data from here if you want:

https://drive.google.com/drive/u/1/folders/1AEJbWGCD507uqtJ5hWNZD9nv8hRBv44v

Here are the find neighbors function and the prediction function:

```
find_neighbours<-function(sim_mat,method='combined',threshold=NA,n=NA){

  #####Input: similarity matrix
  #####       method: weighthres, bestn, combined
  #####       threshold
  #####Output: index of neighbours

  stopifnot(method=='weighthres'|method=='combined'|method=='bestn')

  neighbours<-list()

  if (method=='weighthres'){
    for (i in 1:nrow(sim_mat)){
      sel<-ifelse(abs(sim_mat[i,])>threshold,1,0)
      ind<-which(sel==1)
      neighbours[[i]]<-ind[which(ind!=i)]
    }
  }else{
```

```
    if(method=='bestn'){
      for (i in 1:nrow(sim_mat)){
        ord<-order(abs(sim_mat[i,]),decreasing = T)
        neighbours[[i]]<-ord[which(ord!=i)][1:n]
      }
    }else{
      for(i in 1:nrow(sim_mat)){
        ind.w<-which(ifelse(abs(sim_mat[i,])>threshold,1,0)==1)
        sel.w<-ind.w[which(ind.w!=i)]
        ord.b<-order(abs(sim_mat[i,]),decreasing = T)
        sel.b<-ord.b[which(ord.b!=i)][1:n]
        neighbours[[i]]<-intersect(sel.w,sel.b)
      }
    }
  }

  null <- (1:length(neighbours))[lapply(neighbours,length) == 0]
  if(length(null)!=0){
    for (i in 1:length(null)){
      index <- null[i]
      neighbours[[index]] <- null[i]
    }
  }

  return(neighbours)
}


prediction<-function(df,sim_mat,nbor_list){

  df.new<-df   #copy for calculating the mean by control NA.
  for (i in 1:nrow(df.new)){
    df.new[i,]<-ifelse(df.new[i,]==0,NA,df.new[i,])
  }
  df.mean<-matrix(rep(rowMeans(df.new,na.rm=T),each=ncol(df)),nrow=nrow(df),byrow = T)

  pred<-matrix(NA,ncol=ncol(df),nrow=nrow(df))

  for (i in 1:nrow(df)){
    ind<-nbor_list[[i]]
    nbors_mat<-df[ind,]
    w<-as.matrix(sim_mat[i,ind]/sum(sim_mat[i,ind]))
    nei.demean <- as.matrix(nbors_mat-df.mean[ind,])
    pred.a<- df.mean[i,1] + (w %*% (nei.demean))
    rm.ind<-which(df[i,]==1)
    pred.a[rm.ind]<-0
    pred[i,]<-pred.a
  }

  for (i in 1:nrow(df)){
    pred[i,]<-ifelse(pred[i,]<0,0,pred[i,])
  }
```

```
  rownames(pred)<-rownames(df)
  colnames(pred)<-colnames(df)

  return(pred)
}
```

Then, we implement the function into all the weights we got.

```
############### Movie data set
######## Spearman Correlation
Movie_weights <- read.csv("C:/StudyLife/Columbia/STAT 5243/Local Project 4/Each Movie Case/Movie_Spearma

data1 <- read.csv("C:/StudyLife/Columbia/STAT 5243/Local Project 4/Each Movie Case/Movie_data_train.csv

movietest <- read.csv("C:/StudyLife/Columbia/STAT 5243/Local Project 4/Each Movie Case/Movie_data_test.
movietest <- movietest[,-1]

numeric_movietest <- NULL
for(i in 1:ncol(movietest)) {
  numeric_movietest <- cbind(numeric_movietest, as.numeric(movietest[,i]))
}

# neighbors of method bestn
Movie_neighbors <- find_neighbours(Movie_weights[,-1], method = "bestn" , n = 60)

movie_prediction <- prediction(data1[, -1], Movie_weights[, -1], Movie_neighbors)

# neighbors of method weighthres
movie_neighbors_weighthres <- find_neighbours(Movie_weights[,-1], method = "weighthres" , threshold = 0

movie_prediction_weighthres <- prediction(data1[, -1], Movie_weights[, -1], movie_neighbors_weighthres)

# neighbors of method combined
movie_neighbors_combined <- find_neighbours(Movie_weights[,-1], method = "combined" , threshold = 0.3, r

movie_prediction_combined <- prediction(data1[, -1], Movie_weights[, -1], movie_neighbors_combined)


######### MSD
Movie_MSD_weights <- read.csv("C:/StudyLife/Columbia/STAT 5243/Local Project 4/Each Movie Case/Movie_msd

# neighbors of method bestn
Movie_MSD_neighbors <- find_neighbours(Movie_MSD_weights[,-1], method = "bestn" , n = 60)

movie_prediction_MSD <- prediction(data1[, -1], Movie_MSD_weights[, -1], Movie_MSD_neighbors)


# neighbors of method weighthres
movie_MSD_neighbors_weighthres <- find_neighbours(Movie_MSD_weights[,-1], method = "weighthres" , thresh

movie_prediction_MSD_weighthres <- prediction(data1[, -1], Movie_MSD_weights[, -1], movie_MSD_neighbors_

# neighbors of method combined
movie_MSD_neighbors_combined <- find_neighbours(Movie_MSD_weights[,-1], method = "combined" , threshold
```

```
movie_prediction_MSD_combined <- prediction(data1[, -1], Movie_MSD_weights[, -1], movie_MSD_neighbors_c


####### MS_data #########

#######Spearman+weighthreshold
df<-read.csv('MS_data_train.csv',header=T,sep=',')[,-1]
df_test<-read.csv('MS_data_test.csv',header=T,sep=',')[,-1]
##import similarity
sim_mat<-read.csv('MS_spearman_weights.csv',header=T,sep=',')[,-1]
##find neighbourhoods
tm<-system.time(nb<-find_neighbours(sim_mat,method='weighthres',threshold=0.03)
##prediction
df<-as.matrix.data.frame(df)
tm2<-system.time(pred<-prediction(df,sim_mat,nb))
r<-rank_score(pred, df_test, d=0,alpha=134)


#######Spearman+bestn
df<-read.csv('MS_data_train.csv',header=T,sep=',')[,-1]
df_test<-read.csv('MS_data_test.csv',header=T,sep=',')[,-1]
##import similarity
sim_mat<-read.csv('MS_spearman_weights.csv',header=T,sep=',')[,-1]
##find neighbourhoods
tm<-system.time(nb<-find_neighbours(sim_mat,method='bestn',n=40)
##prediction
df<-as.matrix.data.frame(df)
tm2<-system.time(pred<-prediction(df,sim_mat,nb))
r<-rank_score(pred, df_test, d=0,alpha=134)



#######Spearman+combined
df<-read.csv('MS_data_train.csv',header=T,sep=',')[,-1]
df_test<-read.csv('MS_data_test.csv',header=T,sep=',')[,-1]
##import similarity
sim_mat<-read.csv('MS_spearman_weights.csv',header=T,sep=',')[,-1]
##find neighbourhoods
tm<-system.time(nb<-find_neighbours(sim_mat,method='combined',n=40,threshold=0.03)
##prediction
df<-as.matrix.data.frame(df)
tm2<-system.time(pred<-prediction(df,sim_mat,nb))
r<-rank_score(pred, df_test, d=0,alpha=134)




#######MSD+weighthreshold
df<-read.csv('MS_data_train.csv',header=T,sep=',')[,-1]
df_test<-read.csv('MS_data_test.csv',header=T,sep=',')[,-1]
##import similarity
sim_mat<-read.csv('MS_msd_weights.csv',header=T,sep=',')[,-1]
##find neighbourhoods
```

```
tm<-system.time(nb<-find_neighbours(sim_mat,method='weighthres',threshold=0.03)
##prediction
df<-as.matrix.data.frame(df)
tm2<-system.time(pred<-prediction(df,sim_mat,nb))
r<-rank_score(pred, df_test, d=0,alpha=134)


#######MSD+bestn
df<-read.csv('MS_data_train.csv',header=T,sep=',')[,-1]
df_test<-read.csv('MS_data_test.csv',header=T,sep=',')[,-1]
##import similarity
sim_mat<-read.csv('MS_msd_weights.csv',header=T,sep=',')[,-1]
##find neighbourhoods
tm<-system.time(nb<-find_neighbours(sim_mat,method='bestn',n=40)
##prediction
df<-as.matrix.data.frame(df)
tm2<-system.time(pred<-prediction(df,sim_mat,nb))
r<-rank_score(pred, df_test, d=0,alpha=134)



#######MSD+combined
df<-read.csv('MS_data_train.csv',header=T,sep=',')[,-1]
df_test<-read.csv('MS_data_test.csv',header=T,sep=',')[,-1]
##import similarity
sim_mat<-read.csv('MS_msd_weights.csv',header=T,sep=',')[,-1]
##find neighbourhoods
tm<-system.time(nb<-find_neighbours(sim_mat,method='combined',n=40,threshold=0.03)
##prediction
df<-as.matrix.data.frame(df)
tm2<-system.time(pred<-prediction(df,sim_mat,nb))
r<-rank_score(pred, df_test, d=0,alpha=134)
```

Step 3: Clustering

Here, we write out the Model-based Algorithm (Cluster Models) and apply it to the movie dataset.

Here is all the procedure we have for this algorithm.

```
m_train_full<-read.csv(choose.files())
m_test<-read.csv(choose.files())
m_train<-m_train_full[, -1]
m_test<-m_test[, -1]
m_train<-m_train[1:100, 1:200]
################ EM Algorithm#################

clusterEM<- function (m_train, tau, C){
  J <- ncol(m_train)
  # C<-5 # choice of Groups
  I<-nrow(m_train)
  ## initialize the mu and and gamma
  mu_int<-numeric(C)
  gamma<-array(0, dim = c(7, J, C)) #7 vote, J movie, C cluster
```

```
for (c in 1:(C-1)) {
  mu_int[c]<-runif(1, 0, 1-sum(mu_int))
}

mu_int[C]<-1-sum(mu_int)


## Problem's here
for( c in 1:C){
  for(j in 1:J){
  gamma[1, j, c]<-runif(1, 0, 1)
  gamma[2, j, c]<-runif(1, 0, 1-gamma[1, j, c])
  gamma[3, j, c]<-runif(1, 0, 1-gamma[1, j, c]-gamma[2, j, c])
  gamma[4, j, c]<-runif(1, 0, 1-gamma[1, j, c]-gamma[2, j, c]-gamma[3, j, c])
  gamma[5, j, c]<-runif(1, 0, 1-gamma[1, j, c]-gamma[2, j, c]-gamma[3, j, c]-gamma[4, j, c])
  gamma[6, j, c]<-runif(1, 0, 1-gamma[1, j, c]-gamma[2, j, c]-gamma[3, j, c]-gamma[4, j, c]-gamma[5, j
  gamma[7, j, c]<-1-gamma[1, j, c]-gamma[2, j, c]-gamma[3, j, c]-gamma[4, j, c]-gamma[5, j, c]-gamma[
  }
}


pi<-matrix(NA, ncol = C, nrow= nrow(m_train)) # r = user, C = Group

phi<-matrix(NA, ncol = C, nrow =I)
pi_pre <- matrix(0, nrow = I, ncol = C)



iter <- 1 #iteration
ITER<-200
check_conv <- Inf

while(check_conv > tau & iter < ITER){
  for(i in 1:I){ #get phi
    sub_mu_phi <- array(0,C)
    for(j in 1:20) {
      #i<-1; j <- 3
      sub_mu_phi <- sub_mu_phi + log(gamma[m_train[i,j]+1, j,])
      #print(sub_mu_phi)
    }
    phi[i, ] <- exp(sub_mu_phi)
  }
  tmp_1 <- mu_int * phi
  pi <- tmp_1/apply(tmp_1, 1, sum)


  mu_int <- apply(pi, 2, sum)/I

  for(c in 1:C){
    for(j in 1:J){

      gamma[1, j, c]<-sum(pi[, c]*(m_train[, j] == 0))/sum(pi[, c])
      gamma[2, j, c]<-sum(pi[, c]*(m_train[, j] == 1))/sum(pi[, c])
```

```
        gamma[3, j, c]<-sum(pi[, c]*(m_train[, j] == 2))/sum(pi[, c])
        gamma[4, j, c]<-sum(pi[, c]*(m_train[, j] == 3))/sum(pi[, c])
        gamma[5, j, c]<-sum(pi[, c]*(m_train[, j] == 4))/sum(pi[, c])
        gamma[6, j, c]<-sum(pi[, c]*(m_train[, j] == 5))/sum(pi[, c])
        gamma[7, j, c]<-sum(pi[, c]*(m_train[, j] == 6))/sum(pi[, c])
      }
    }

    check_conv <- norm(pi - pi_pre)
    print(check_conv)
    print(iter)

    pi_pre <- pi
    iter = iter + 1
  }

  return(list(mu = mu_int, gamma= gamma, pi = pi))
}

cl<-clusterEM(m_train, 5, 3)
```

Step 4: Evaluation

We firstly write out the functions for the two simulation methods, which are respectively Rank Score and MAE.

```
## MAE
evaluation_mae <- function(pred_mat, Movie_test){
  ## function to calculate mean absolute error of predicted value
  ## Input: pred_mat - predicted value
  ##        Movie_test - test data matrix
  ## Output: MAE
  for (i in 1:nrow(pred_mat)){
    pred_mat[i,]<-ifelse(pred_mat[i,]==0,NA,pred_mat[i,])
  }
  mae <- mean(abs(pred_mat - Movie_test), na.rm = T)
  return(mae)
}

## Rank Score
rank_score<-function(pred,test,d,alpha){
  R_a<-rep(NA,nrow(test))
  R_a_max<-rep(NA,nrow(test))

  for(i in 1:nrow(test)){
    j<-order(pred[i,],decreasing = T)
    pred.s<-sort(pred[i,])
    numer<-ifelse(pred.s-d>0,pred.s-d,0)
    R_a[i] <- sum( numer / 2^((j-1)/(alpha-1)))
    pred.s<-ifelse(test[i,]==1,1,pred.s)
    j<-order(pred.s,decreasing = T)
    pred.s<-sort(pred.s)
    denom<-ifelse(pred.s-d>0,pred.s-d,0)
```

```
    R_a_max[i] <- sum( denom / 2^((j-1)/(alpha-1)))
  }
  R <- 100 * (sum(R_a) / sum(R_a_max))
  return(R)
}
```

Then, we calculate the evaluations for the required the datasets.

```
######## Spearman
dataframe_movie_prediction <- data.frame(movie_prediction)

eval_mae_Spearman <- evaluation_mae(as.matrix(dataframe_movie_prediction), as.matrix(numeric_movietest))
eval_mae_Spearman

# neighbors of method weighthres
dataframe_movie_prediction_weighthres <- data.frame(movie_prediction_weighthres)

eval_mae_Spearman_weighthres <- evaluation_mae(as.matrix(dataframe_movie_prediction_weighthres), as.mat
eval_mae_Spearman_weighthres

# neighbors of method combined
dataframe_movie_prediction_combined <- data.frame(movie_prediction_combined)

eval_mae_Spearman_combined <- evaluation_mae(as.matrix(dataframe_movie_prediction_combined), as.matrix(
eval_mae_Spearman_combined

######### MSD
# neighbors of method bestn
dataframe_movie_MSD_prediction <- data.frame(movie_prediction_MSD)

eval_mae_MSD <- evaluation_mae(as.matrix(dataframe_movie_MSD_prediction), as.matrix(numeric_movietest))
eval_mae_MSD

# neighbors of method weighthres
dataframe_movie_prediction_MSD_weighthres <- data.frame(movie_prediction_MSD_weighthres)

eval_mae_MSD_weighthres <- evaluation_mae(as.matrix(dataframe_movie_prediction_MSD_weighthres), as.matr
eval_mae_MSD_weighthres

# neighbors of method combined
dataframe_movie_prediction_MSD_combined <- data.frame(movie_prediction_MSD_combined)

eval_mae_MSD_combined <- evaluation_mae(as.matrix(dataframe_movie_prediction_MSD_combined), as.matrix(nu
eval_mae_MSD_combined
```

For the cluster, the predictions and evaluations are shown here:

```
#Generating prediction
mu <- cl$mu
gamma<-cl$gamma
pi<-cl$pi


pred_func <- function(m_train, gamma, mu, pi){
  item_test <- colnames(m_train)
```

```
  # order <- match(item_test, items)
  I <- nrow(m_train)
  J <- ncol(m_train)
  predictions <- matrix(numeric(I*J), nrow = I, ncol = J)
  #colnames(predictions) <- item_test

  user_cluster <- apply(pi, 1, which.max)
  error <- 0

  for(i in 1:I){
    for(j in 1:J){
      cluster <- user_cluster[i]
      predictions[i,j] <- sum(c(gamma[1, j, cluster], gamma[2, j, cluster], gamma[3, j, cluster], gamma
                              gamma[5, j, cluster], gamma[6, j, cluster], gamma[7, j, cluster])*c(0, 1, 2
    }
  }
  predictions<-as.data.frame(predictions)
  return(predictions)
}
NN<-apply(predictions, 1, is.na)
MM<-apply(m_test, 1, is.na)
pre<-pred_func(m_train, gamma, mu, pi)

clusterMAE<-function(m_test, prediction){
  J <- ncol(m_test)
  # C<-5 # choice of Groups
  I<-nrow(m_test)
  MAE<-abs(m_test-pre)
  return(sum(MAE)/(I*J))
}

evaluation_mae <- function(pred_mat, Movie_test){
  ## function to calculate mean absolute error of predicted value
  ## Input: pred_mat - predicted value
  ##        Movie_test - test data matrix
  ## Output: MAE
  pred_mat<-pre
  Movie_test<-m_test
  for (i in 1:nrow(pred_mat)){
    Movie_test[i,]<-ifelse(Movie_test[i,]==0,NA,Movie_test)
  }
  mae <- as.matrix(abs(pred_mat - Movie_test))
  mae<-mean(mae, na.rm = TRUE)
  return(mae)
}

evaluation_mae2 <- function(pred_mat, Movie_test, Movie_train){
  ## function to calculate mean absolute error of predicted value
  ## Input: pred_mat - predicted value
  ##        Movie_test - test data matrix
  ## Output: MAE
  for (i in 1:nrow(pred_mat)){
    Movie_test[i,]<-ifelse(Movie_train[i,]!=0,NA,Movie_test)
```

```
  }
  mae <- as.matrix(abs(pred_mat - Movie_test))
  return(mae)
}


pre<-clusterEM(m_train, 5, 5)
pre<-pred_func(m_train, pre$gamma, pre$mu, pre$pi)
A<-evaluation_mae2(pre, m_test, m_train)

### CV

C_CV<-seq(20,30, 2)
MAE<-numeric(10)
for (i in 1:10){
  pre<-clusterEM(m_train, 5, C_CV[i])
  pre<-pred_func(m_train, pre$gamma, pre$mu, pre$pi)
  MAE[i]<-evaluation_mae(pre, m_test)
}

pre<-clusterEM(m_train, 5, 60)
pre<-pred_func(m_train, pre$gamma, pre$mu, pre$pi)
MAEEEE<-evaluation_mae(pre, m_test)
```

Step5: Results

Here are the plots we created for showing our results for the Movie dataset and Microsoft Web dataset respectively.