main

Group4

4/9/2018

Step 0: Load the packages

```
#if (!require("")) install.packages("")

#default the wd to the fold this rmd file exists
```

Step 1: Load and process the data

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

Step 2 : Transformation

Convert the original dataset to a matrix which rows represents users and columns represents items For dataset 1 (MS), we assign 0 to those items which users never visited. For dataset 2 (Movie), we assign NA to those items which users never rated.

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

MS_train <- Transform_ms(MS_train)
MS_test <- Transform_ms(MS_test)
# save(MS_train, file = "../output/MS_train.RData")
# save(MS_test, file = "../output/MS_test.RData")

movie_train <- Transform_m(movie_train)
movie_test  <- Transform_m(movie_test)
# save(movie_train, file = "../output/movie_train.RData")
# save(movie_test, file = "../output/movie_test.RData")
```

Memory-based Algorithm

Step 3 : Similarity Weight

Pearson Correlation & Mean-square-difference & SimRank

```
load("../output/MS_train.RData")
load("../output/MS_test.RData")
load("../output/movie_train.RData")
load("../output/movie_test.RData")

#For dataset 1 (MS)

#Pearson Correlation
#ms_pc <- pearson_corr(MS_train)
#save(ms_pc, file = "ms_pc.RData")

#Mean-square-difference
#ms_msd <- MSD_Weight(MS_train)
#save(ms_msd, file = "../output/ms_msd.RData")

#SimRank
#ms_sr <- simrank(MS_train)
#save(ms_sr, file = "../output/simrank_MS_train.RData")


#For dataset 2 (Movie)

#Pearson Correlation
#movie_pc <- pearson_corr(movie_train)
#save(movie_pc, file = "movie_pc.RData")

#Mean-square-difference
#movie_msd <- MSD_Weight(movie_train)
#save(movie_msd, file = "../output/movie_msd.RData")
```

No Variance Weighting

Step 4: Selecting Neighbors

```
# Implementation on Dataset 1

# MSD + WT
# ms_msd_wt <- corr_thresh(ms_msd, 0.5)
# save(ms_msd_wt, file = "../output/ms_msd_wt.RData")


# MSD + BNN
# ms_msd_bnn <- Select_BNN(ms_msd, 20)
# save(ms_msd_bnn, file = "../output/ms_msd_bnn.RData")

# MSD + combine
# ms_msd_combine <- corr_thresh(ms_msd, 0.5)
# save(ms_msd_combine, file = "../output/ms_msd_combine.RData")


# PC + WT
# ms_pc_wt <- corr_thresh(ms_pc, 0.5)
# save(ms_pc_wt, file = "../output/ms_pc_wt.RData")

# PC + BNN
# ms_pc_bnn <- Select_BNN(ms_pc, 20)
# save(ms_pc_bnn, file = "../output/ms_pc_bnn.RData")

# PC + combine
# ms_pc_combine <- combine(ms_pc, 0.005, 20)
# save(ms_pc_combine, file = "../output/ms_pc_combine.RData")

# Simrank + WT
# ms_sr_wt <- corr_thresh(ms_sr, 0.5)
# save(ms_sr_wt, file = "../output/ms_pc_wt.RData")

# Simrank + BNN
# ms_sr_bnn <- Select_BNN(ms_sr, 20)
# save(ms_sr_bnn, file = "../output/ms_sr_bnn.RData")

# Simrank + combine
# ms_sr_combine <- combine(ms_sr, 0.005, 20)
# save(ms_sr_combine, file = "../output/ms_sr_combine.RData")



# Implementation on Dataset 2

# MSD + WT
# movie_msd_wt <- corr_thresh(movie_msd, 0.5)
# save(movie_msd_wt, file = "../output/movie_msd_wt.RData")


# MSD + BNN
# movie_msd_bnn <- Select_BNN(movie_msd, 20)
# save(movie_msd_bnn, file = "../output/movie_msd_bnn.RData")

# MSD + combine
# movie_msd_combine <- combine(movie_msd, 0.005, 20)
# save(movie_msd_combine, file = "../output/movie_msd_combine.RData")


# PC + WT
# movie_pc_wt <- corr_thresh(movie_pc, 0.5)
# save(movie_pc_wt, file = "../output/movie_pc_wt.RData")

# PC + BNN
# movie_pc_bnn <- corr_thresh(movie_pc, 0.5)
# save(movie_pc_bnn, file = "../output/movie_pc_bnn.RData")

# PC + combine
# movie_pc_combine <- combine(movie_pc, 0.005, 20)
# save(movie_pc_combine, file = "../output/movie_pc_combine.RData")
```

Step 5 : Prediction

```
# Implementation on Dataset 1

# MSD + WT
# pred_ms_msd_wt <- avg_dev_pred(MS_train,MS_test,ms_msd, ms_msd_wt)

# MSD + BNN
# ZScore_ms_msd_bnn<- ZScore_Mat(ms_msd, ms_msd_bnn, MS_train, MS_test)

# MSD + combine
# pred_ms_msd_combine <- avg_dev_pred(MS_train,MS_test,ms_msd, ms_msd_combine)


# PC + WT
# pred_ms_pc_wt <- avg_dev_pred(MS_train,MS_test,ms_pc, ms_pc_wt)

# PC + BNN
# ZScore_ms_pc_bnn<- ZScore_Mat(ms_pc, ms_pc_bnn, MS_train, MS_test)

# PC + combine
# pred_ms_pc_combine <- avg_dev_pred(MS_train,MS_test,ms_pc, ms_pc_combine)

# Simrank + WT
# pred_ms_sr_wt <- avg_dev_pred(MS_train,MS_test,ms_sr, ms_sr_wt)

# Simrank + BNN
# ZScore_ms_sr_bnn<- ZScore_Mat(ms_sr, ms_sr_bnn, MS_train, MS_test)

# Simrank + combine
# pred_ms_sr_combine <- avg_dev_pred(MS_train,MS_test, ms_sr, ms_sr_combine)



# Implementation on Dataset 2

# MSD + WT
# pred_movie_msd_wt <- avg_dev_pred(movie_train,movie_test,movie_msd, movie_msd_wt)

# MSD + BNN
# ZScore_movie_msd_bnn<- ZScore_Mat(movie_msd, movie_msd_bnn, movie_train, movie_test
)
```

```
# MSD + combine
# pred_movie_msd_combine <- avg_dev_pred(movie_train,movie_test,movie_msd, movie_msd_
combine)


# PC + WT
# pred_movie_pc_wt <- avg_dev_pred(movie_train,movie_test,movie_pc, movie_pc_wt)

# PC + BNN
# ZScore_movie_pc_bnn<- ZScore_Mat(movie_pc, movie_pc_bnn, movie_train, movie_test)

# PC + combine
# pred_movie_pc_combine <- avg_dev_pred(movie_train,movie_test,movie_pc, movie_pc_com
bine)
```

Step 6 : Valuation

```
# Implementation on Dataset 1: ranked scoring

# MSD + WT
# ms_msd_wt_RS <- Rank_Score(pred_ms_msd_wt, MS_test)
# save(ms_msd_wt_RS, file = "../output/ms_msd_wt_RS.RData")

# MSD + BNN
# ms_msd_bnn_RS <- Rank_Score(ZScore_ms_msd_bnn, MS_test)
# save(ms_msd_bnn_RS, file = "../output/ms_msd_bnn_RS.RData")


# MSD + combine
# ms_msd_combine_RS <- Rank_Score(pred_ms_msd_combine, MS_test)
# save(ms_msd_combine_RS, file = "../output/ms_msd_combine_RS.RData")



# PC + WT
# ms_pc_wt_RS <- Rank_Score(pred_ms_pc_wt, MS_test)
# save(ms_pc_wt_RS, file = "../output/ms_pc_wt_RS.RData")


# PC + BNN
# ms_pc_bnn_RS <- Rank_Score(ZScore_ms_pc_bnn, MS_test)
# save(ms_pc_bnn_RS, file = "../output/ms_pc_bnn_RS.RData")

# PC + combine
# ms_pc_combine_RS <- Rank_Score(pred_ms_pc_combine, MS_test)
# save(ms_pc_combine_RS, file = "../output/ms_pc_combine_RS.RData")

# Simrank + WT
# ms_sr_wt_RS <- Rank_Score(pred_ms_sr_wt, MS_test)
# save(ms_sr_wt_RS, file = "../output/ms_sr_wt_RS.RData")

# Simrank + BNN
# ms_sr_bnn_RS <- Rank_Score(ZScore_ms_sr_bnn, MS_test)
# save(ms_sr_bnn_RS, file = "../output/ms_sr_bnn_RS.RData")

# Simrank + combine
# ms_sr_combine_RS <- Rank_Score(pred_ms_sr_combine, MS_test)
# save(ms_sr_combine_RS, file = "../output/ms_sr_combine_RS.RData")


# Implementation on Dataset 2: MAE

# MSD + WT
# movie_msd_wt_MAE <- MAE(pred_movie_msd_wt, movie_test)
# save(movie_msd_wt_MAE, file = "../output/movie_msd_wt_MAE.RData")



# MSD + BNN
# movie_msd_bnn_MAE <- MAE(ZScore_movie_msd_bnn, movie_test)
# save(movie_msd_bnn_MAE, file = "../output/movie_msd_bnn_MAE.RData")

# MSD + combine
# movie_msd_combine_MAE <- MAE(pred_movie_msd_combine, movie_test)
# save(movie_msd_combine_MAE, file = "../output/movie_msd_combine_MAE.RData")




# PC + WT
# movie_pc_wt_MAE <- MAE(pred_movie_pc_wt, movie_test)
# save(movie_pc_wt_MAE, file = "../output/movie_pc_wt_MAE.RData")

# PC + BNN
# movie_msd_pc_MAE <- MAE(ZScore_movie_pc_bnn, movie_test)
# save(movie_msd_pc_MAE, file = "../output/movie_msd_pc_MAE.RData")

# PC + combine
# movie_pc_combine_MAE <- MAE(pred_movie_pc_combine, movie_test)
# save(movie_pc_combine_MAE, file = "../output/movie_pc_combine_MAE.RData")
```

Model-based Algorithm

Step 3: Cluster Model

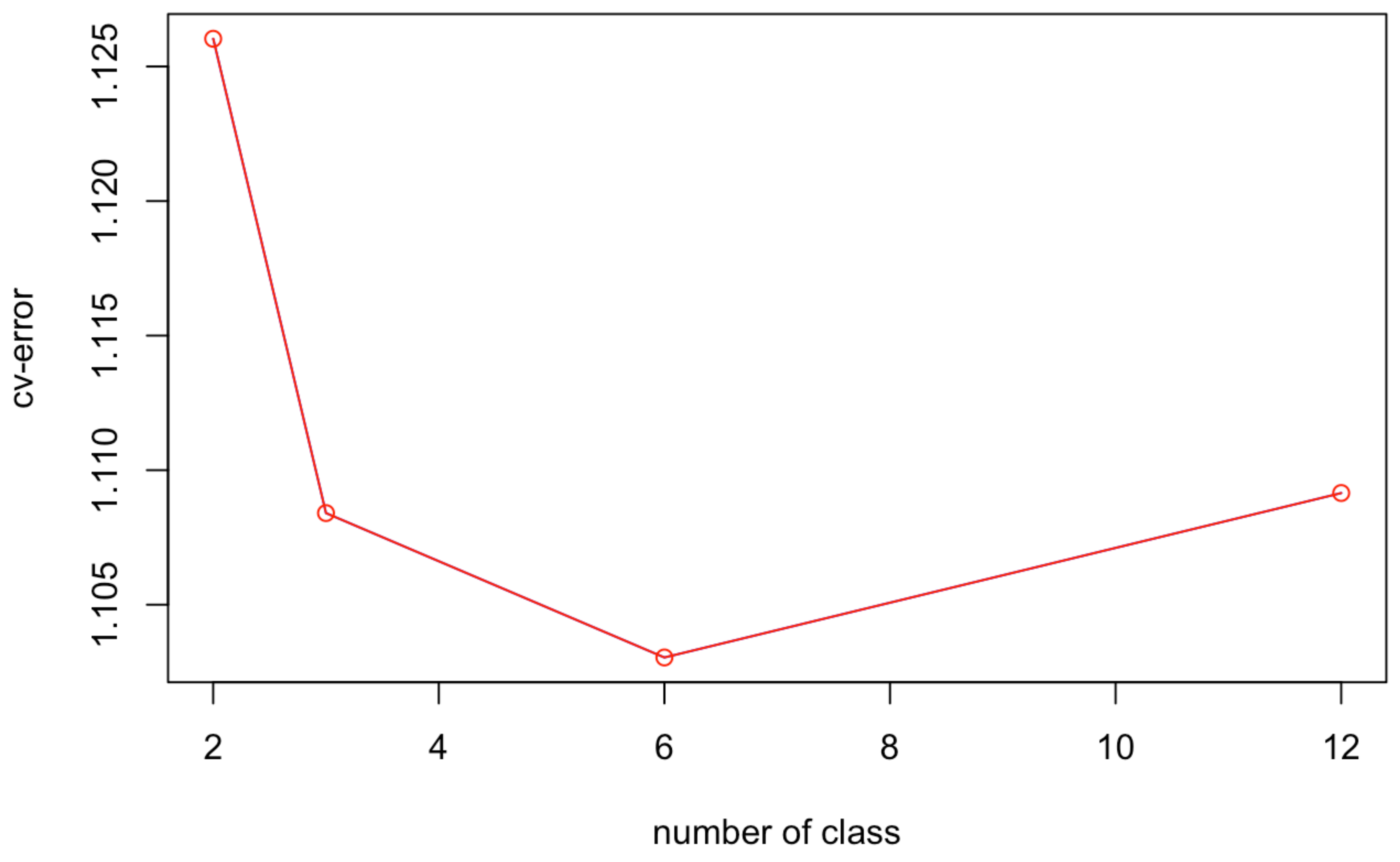
```
load("../output/movie_train.RData")
load("../output/movie_test.RData")
train <- movie_train
```

```
test <- movie_test


N <- nrow(train)
M <- ncol(train)

user <- rownames(train)
movie <- colnames(train)


### cluster model
em_fun <- function(data, C, thres){
  #Input: train_data, number of classes, threshold to determine convergence
  #Output: parameters for cluster models:
  #    mu: probability of belonging to class c, vector
  # gamma: probability of scores for a movie given the class, 3 dimentions


  #========================
  # Step 1 - initialization
  #========================
  set.seed(2)
  mu <- runif(C)
  mu <- mu/sum(mu)
  gamma <- array(NA,c(M,C,6)) #each matrix represents a class
  #the i,j-th element means the probability of rating jth movie with score i in the c
lass
  for(m in 1:M){
    for(c in 1:C){
      gamma[m,c,] <- runif(6)
      gamma[m,c,] <- gamma[m,c,]/sum(gamma[m,c,])
    }
  }

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

  mu_new <- mu
  gamma_new <- gamma

  ## Iterations based on the stop criterion
  thres1 <- 1000
  thres2 <- 1000
  thres1_new <- 0
  thres2_new <- 0
  count <- 0
```

```r
  while((thres1>thres|thres2>thres)&(abs(thres1-thres1_new)>thres|abs(thres2-thres2_n
ew)>thres))
  {
    count <- count + 1
    print(paste0("iteration = ", count))

    thres1_new <- thres1
    thres2_new <- thres2

    mu <- mu_new
    gamma <- gamma_new

    #========================
    # Step 2 - Expectation
    #========================
    #expectation pi with rows meaning classes and columns meaning users
    phi <- matrix(0, C, N)

    for(k in 1:6){
      phi <- phi + t(log(gamma[,,k]))%*%v[,,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))

    #========================
    # Step 3 - Maximization
    #========================
    mu_new <- rowSums(phi)/N  #update mu vector

    for(k in 1:6){
      gamma_new[,,k] <- v[,,k]%*%t(phi)/v[,,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
    thres1 <- mean(abs(mu_new - mu)) #mean absolute difference of mu
    thres2 <- 0
    for(c in 1:C){
      thres2 <- max(thres2,norm(as.matrix(gamma_new[,c,] - gamma[,c,]), "O"))
    }
    print(paste0("threshold1 = ", thres1, " threshold2 = ", thres2))
  }
  return(list(mu = mu, gamma = gamma))
}

#predict score estimate function
cm_predict <- function(train_df, test_df, par){
  set.seed(2)
  mu <- par$mu
  gamma <- par$gamma
  C <- length(mu)

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

  ##using 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]))%*%v[,,k]
  }

  phi <- exp(phi)

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


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

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


### 5-fold cross validation to find best class number C among c_list(2,3,6,12)

set.seed(2)
K <- 5
n <- ncol(train)
```

```r
m <- nrow(train)
n.fold <- floor(n/K)
m.fold <- floor(m/K)
s <- sample(rep(1:K, c(rep(n.fold, K-1), n-(K-1)*n.fold)))
s1 <- sample(rep(1:K, c(rep(m.fold, K-1), m-(K-1)*m.fold)))

c_list <- c(2,3,6,12)
validation_error <-  matrix(NA, K, length(c_list))

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

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


#cv 5 folds
#calculate cv error
cv_fun <- function(train_data,test_data){
 for(i in 1:K){
  train_data[s1 != i, ] <- train[s1 != i, ]
  train_data[s1 == i, s != i] <- train[s1==i, s != i]
  test_data[s1 == i,s == i] <- train[s1 == i ,s == i]
  #write.csv(train_data,paste0("../output/cluster_model_subtrain.csv"))
  #write.csv(test_data,paste0("../output/cluster_model_validation.csv"))

  estimate_data <- test_data

  for(c in 1:length(c_list)){
    cm_par <- em_fun(data = train_data, C = c_list[c], thres = 0.05)
    estimate_data <- cm_predict(train_df = train_data, test_df = test_data, par = cm_
par)
    validation_error[i,c] <- sum(abs(estimate_data-test_data),na.rm = T)/sum(!is.na(e
stimate_data-test_data))

  }}
  return(validation_error)
  }

#validation_error <- cv_fun(train_data,test_data)
#save(validation_error, file=paste0("../output/validation_err.RData"))

# Cluster number comparism
load("../output/validation_err.RData")
cv_error<-colMeans(validation_error)

# setwd("../figs/")
# jpeg(file=paste("cv_err",".jpg") )
```

```
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")
```



```
#dev.off()


class = c_list[which.min(cv_error)]
print(paste("Best class number is", class))
```

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

```r
class <- 6

#best_par <- em_fun(data = train, C = class, thres = 0.01)
#save(best_par, file = "../output/best_par.RData")

load("../output/best_par.RData")


###estimate scores

#estimate <- cm_predict(train, test, best_par)

#write.csv(estimate, paste0("../output/cluster_model_estimate.csv"))


estimate <- read.csv("../output/cluster_model_estimate.csv")
estimate <- estimate[,-1]

# MAE of EM algorithm
coltest <- colnames(test)
colnames(estimate) <- movie

coltest <- which(is.element(movie, coltest))
estimate <- estimate[,coltest]

MAE <- function(pred, true){
  mae <- sum(abs(pred-test),na.rm = T)/sum(!is.na(abs(pred-test)))
  return(mae)
}

error_em<- MAE(estimate,test)
error_em
```

```
## [1] 2.803069
```