ADS Project4

Xiaoxiao Guo

4/17/2018

Memory-Based Algorithm

Load Data and make matrix

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 clicked. For dataset 2 (EachMovie), we assign NA to those items which users never rated.

```
source("../lib/memory.new.R")
source("../lib/memory.new.R")
```

```
# EachMovie
 movie_train <- read.csv("../data/data_sample/eachmovie_sample/data_train.csv", header = T)
 movie_train <- movie_train[, -1]
 movie_train$Movie <- factor(movie_train$Movie)
 movie_train$User <- factor(movie_train$User)
 movie_train <- spread(movie_train, Movie, Score)

 movie_test <- read.csv("../data/data_sample/eachmovie_sample/data_test.csv", header = T)
 movie_test <- movie_test[, -1]
 movie_test$Movie <- factor(movie_test$Movie)
 movie_test$User <- factor(movie_test$User)
 movie_test <- spread(movie_test, Movie, Score)

 row.names(movie_train) <- movie_train$User
 movie_train <- movie_train[ , -1]
 row.names(movie_test) <- movie_test$User
 movie_test <- movie_test[ , -1]

# Save EachMovie data
 save(movie_train, file = "../output/movie_train.RData")
 save(movie_test, file = "../output/movie_test.RData")
 write.csv(movie_train, file = "../output/movie_train.csv")
 write.csv(movie_test, file = "../output/movie_test.csv")

 rawtrain= read.csv("../data/data_sample/MS_sample/data_train.csv",as.is = TRUE)[,-1]
 rawtest = read.csv("../data/data_sample/MS_sample/data_test.csv",as.is = TRUE)[,-1]
 ms_train = reshape_ms(rawtrain)
 ms_test = reshape_ms(rawtest)
 save(ms_train, file = "ms_train.RData")
 save(ms_test, file = "ms_test.RData")
```

```
# EachMovie Dataset
load("../output/movie_train.RData")
load("../output/movie_test.RData")

# Sparse MS Matrix
```

```
load("../output/ms_train.RData")
load("../output/ms_test.RData")
```

Calculate similarity weights

Pearson Correlation & Vector Similarity & SimRank

```
# EachMovie
 pearson_weight <- sim_weights(movie_train,"pearson")
 pearson_weight[is.na(pearson_weight)] <- 0
 pearson_weight <- round(pearson_weight, 4)
 save(pearson_weight,file = "../output/em_pearson.RData")

 vector_weight <- sim_weights(movie_train, "vector")
 vector_weight[is.na(vector_weight)] <- 0
 save(vector_weight,file = "../output/em_vector.RData")

#MS
ms_pearson = sim_weights_ms(ms_train,"pearson")
save(ms_pearson ,file = "../output/ms_pearson.RData")
ms_vector = sim_weights_ms(ms_train, "vector")
save(ms_vector ,file = "../output/ms_vector.RData")
```

```
load("../output/em_pearson.RData")
load("../output/em_vector.RData")
load("../output/ms_pearson.RData")
load("../output/ms_vector.RData")
```

Variance Weighting Using Pearson Similarity

```
# EachMovie
 var_pearson <- sim_var(movie_train)
 var_pearson[is.na(var_pearson)] <- 0
 var_pearson <- round(var_pearson, 4)
 save(var_pearson, file = "../output/em_var_pearson.RData")


# MS
 ms_var_pearson = sim_var_ms(ms_train)
 ms_var_pearson <- round(ms_var_pearson, 4)
 save(ms_var_pearson,file = "../output/ms_var_pearson.RData")
```

```
load("../output/em_var_pearson.RData")
load("../output/ms_var_pearson.RData")
```

Simrank

```
# EachMovie
load("../output/w.RData")
```

```
s_user <- simrank.funcfunction(w)
```

```
load("../output/s_user.RData")
```

Neighbors Selection

Neighbors Selection Methods: Weight Threshold

```
# EachMovie
# Pearson
pearson.neighbor_em <- neighbors_select(pearson_weight)
# Vector
vector.neighbor_em <- neighbors_select(vector_weight)
# Pearson with Variance
pearson_var.neighbor_em <- neighbors_select(var_pearson)
# Simrank
simrank.neighbor_em <- neighbors_select(s_user, 0.0003)


#MS
pearson.neighbor_ms <- neighbors_select(ms_pearson)
vector.neighbor_ms <- neighbors_select(ms_vector)
pearson_var.neighbor_ms <- neighbors_select(ms_var_pearson)
```

Prediction

```
# EachMovie
pearson.pred_em <- pred.em(movie_test, movie_train, pearson_weight, pearson.neighbor_em)
vector.pred_em <- pred.em(movie_test, movie_train, vector_weight, vector.neighbor_em)
pearson_var.pred_em <- pred.em(movie_test, movie_train, var_pearson, pearson_var.neighbor_em)
sim.pred_em <- pred.em(movie_test, movie_train, s_user, simrank.neighbor_em)

#MS
pearson.pred_ms  <- pred.ms(ms_test, ms_train, ms_pearson,pearson.neighbor_ms)
vector.pred_ms  <- pred.ms(ms_test, ms_train, ms_vector,vector.neighbor_ms)
pearson_var.pred_ms  <- pred.ms(ms_test, ms_train, ms_var_pearson,pearson_var.neighbor_ms)
```

Evaluation - MAE

```
# EachMovie
pearson.mae_em <- mae(pearson.pred_em, as.matrix(movie_test))
vector.mae_em <- mae(vector.pred_em, as.matrix(movie_test))
pearson_var.mae_em <- mae(pearson_var.pred_em, as.matrix(movie_test))
simrnak.mae_em <- mae(sim.pred_em, as.matrix(movie_test))
pearson.mae_em
vector.mae_em
pearson_var.mae_em
simrnak.mae_em
```

Evaluation - ROC

```
# EachMovie
pearson.roc_em <- roc(4, pearson.pred_em, as.matrix(movie_test))
vector.roc_em <- roc(4, vector.pred_em, as.matrix(movie_test))
pearson_var.roc_em <- roc(4, pearson_var.pred_em, as.matrix(movie_test))
simrank.roc_em <- roc(4, sim.pred_em, as.matrix(movie_test))
pearson.roc_em
vector.roc_em
pearson_var.roc_em
simrank.roc_em
```

Evaluation - Rank Score

```
# MS
pearson.rs_ms <- rank_score(pearson.pred_ms,ms_test);pearson.rs_ms
vector.rs_ms<- rank_score(vector.pred_ms, ms_test);vector.rs_ms
pearson_var.rs_ms <- rank_score(pearson_var.pred_ms, ms_test);pearson_var.rs_ms
```

Model-Based Algorithm

Next, we do Model-Based Algorithm. We follow the steps which Chengling taught in class.

Install and import the needed packages.

```
train_df = read.csv('../output/MS_train_wide.csv')
packages.used=c('MCMCpack','plyr')

# check packages that need to be installed.
packages.needed=setdiff(packages.used,
                        intersect(installed.packages()[,1],
                                  packages.used))

# install additional packages
if(length(packages.needed)>0){
  install.packages(packages.needed, dependencies = TRUE)
}

# load packages
library('MCMCpack')
library('plyr')
```

Define the parameters.

we can change the number of classes here.

```
num_users = nrow(train_df)
num_web = ncol(train_df) - 1
```

```
num_class = 3
users = 1:num_users
classes = 1:num_class
```

First step of EM algorithm

Initilize the original distributions.

```
miu = rdirichlet(1, alpha = rep(1:num_class))
#miu nclass vector


gamma = array(rdirichlet(num_class*num_web, alpha = c(1,1)),
              dim = c(num_web, num_class, 2),
              dimnames = list(web = colnames(train_df)[2:ncol(train_df)],
                              class = 1:num_class,
                              click = c('click', 'not click')))
```

The E-step of EM algorithm.

```
expectation =  function(df, miu, gamma){
  phi = matrix(rep(NA, num_class*num_users), nrow = num_users,
               ncol = num_class)
  pi = matrix(rep(NA, num_class*num_users), nrow = num_users,
              ncol = num_class)

  for (user in users) {
    select1 = df[user, ]==1
    select0 = !(select1)
    for (class in classes) {
      phi[user, class] = prod(gamma[select1, class,"click"])*
        prod(gamma[select0, class,"not click"])
    }
    denom = miu %*% phi[user, ]

    for (class in classes) {
      pi[user, class] = miu[class]*phi[user, class]/denom
    }
  }
  return(pi)
}
```

The M-step of EM algorithm.

We update the distributions in the step.

```
update = function(pi, df, gamma){
  miu = aaply(pi, 2, mean)
  denominator = aaply(pi, 2, sum)
  denominator = matrix(rep(denominator, num_web), nrow = num_web, ncol = num_class, byrow = TRUE)
  choose_1 = df==1
```

```
  numerator_for_1 = t(as.matrix(choose_1)) %*% pi
  numerator_for_0 = t(as.matrix(!choose_1)) %*% pi
  gamma[ , , 'click'] = numerator_for_1/denominator
  gamma[ , , 'not click'] = numerator_for_0/denominator
  return(list(miu = miu, gamma = gamma))
}
```

Iterate step 2 and step 3 until it converges.

We use the sum of the absolute change of miu as the signal of convergence. If the difference of miu in this step and it in last step is small enough, we think it converges.

```
miu_old = miu + 10
iter = 0
df = train_df[, -1]
while (sum(abs(miu - miu_old)) > 0.03) {
  miu_old = miu
  pi = expectation(df, miu, gamma)
  list = update(pi, df, gamma)
  iter = iter + 1
  miu = list$miu
  gamma = list$gamma
  cat('\niteration', iter, ';difference between current miu and true miu', sum(abs(miu - miu_old)))
}
```

Prediction

```
prediction = function(user, df){
  select1 = df[user, ]==1
  prob_click = apply(gamma[select1, ,"click"], 2, prod)
  mat_click = matrix(rep(prob_click,num_web),
                     ncol=num_class,nrow=num_web,byrow = T)

  pred_click = (mat_click*gamma[ , ,"click"]) %*% miu
  denom = mat_click %*% miu
  prob = pred_click/as.numeric(denom)

  return(prob)
}

test_df = read.csv('../output/MS_test_wide.csv')
index = which(train_df$X %in% test_df$X)
predctions = t(sapply(index, prediction, df))
colnames(predctions) = colnames(df)
predctions = predctions[, (colnames(predctions) %in% colnames(test_df))]
```

Calculate the rank score

```
rank_score = function(predict, test){
  d = 0.03
```

```
  rank_pred = ncol(predict) + 1 - t(apply(predict, 1, rank,
                                          ties.method = "first"))
  rank_test = ncol(test) + 1 - t(apply(test, 1, rank,
                                       ties.method = "first"))
  logic = ifelse(test - d > 0, test - d, 0)
  ranka = apply(1 / (2 ^ ((rank_pred - 1)/4)) * logic, 1, sum)
  ranka_max = apply(1 / (2 ^ ((rank_test - 1) / 4)) * logic, 1, sum)
  r = 100 * sum(ranka) / sum(ranka_max)
  return(r)
}

rank_score(as.matrix(predctions),as.matrix(test_df[,2:ncol(test_df)]))
```