**TODO**

* **Problem A: Database update function (last part of the question) - have to do merge regarding same person who is splitted in testSet and trainSet.**
* **Problem B: PGEC function (Is there a formula for this?)**
* **Problem B: Use ggplot to generate graph (Must finish the other 2 things first)**
* **Write-up in Latex - Wrote up draft on my computer (Charles)**

llibrary(rectools)

getInstEval()

# Setup functions

# Function for getting mode from a vector

vecmode <- function(x) {

vals <- unique(x)

freq <- tabulate(match(x, vals))

vals[which.max(freq)]

}

# Edited predict.usrData function with locMeasure parameter

predict.usrData <- function (origData, newData, newItem, k, wtcovs = NULL, wtcats = NULL, locMeasure = mean)

{

traincovs <- !is.null(origData$usrCovs)

newcovs <- !is.null(newData$cvrs)

if (!(traincovs + newcovs %in% c(0, 2)))

stop("mismatch in having/not having covars, orig and new data")

traincats <- !is.null(origData$usrCats)

newcats <- !is.null(newData$cats)

if (!(traincats + newcats %in% c(0, 2)))

stop("mismatch in having/not having cats, orig and new data")

checkNewItem <- function(oneUsr) {

whichOne <- which(oneUsr$itms == newItem)

if (length(whichOne) > 1) {

stop("same user/item pair encountered more than once")

}

if (length(whichOne) == 0) {

return(c(NA, NA))

}

else return(c(whichOne, oneUsr$ratings[whichOne]))

}

found <- as.matrix(sapply(origData, checkNewItem))

whoHasIt <- which(!is.na(found[1, ]))

if (is.null(whoHasIt) | length(whoHasIt) == 0)

return(NA)

origDataRatedNI <- origData[whoHasIt]

found <- found[, whoHasIt, drop = FALSE]

onecos <- function(y) cosDist(newData, y, wtcovs, wtcats)

cosines <- sapply(origDataRatedNI, onecos)

findKnghbourRtng <- function(ki) {

ki <- min(ki, length(cosines))

nearby <- order(cosines, decreasing = FALSE)[1:ki]

locMeasure(as.numeric(found[2, nearby]))

}

sapply(k, findKnghbourRtng)

}

# Update user data

updateUserData <- function(oldData, newData)

{

resultData <- oldData

newData <- formUserData(newData[,1:3])

for (i in 1:length(newData)) {

hasSameUserID = FALSE

for (j in 1:length(resultData)) {

if (resultData[[j]]$userID == newData[[i]]$userID) {

print("Has same user ID")

print(resultData[[j]]$itms)

resultData[[j]]$itms = append(resultData[[j]]$itms, newData[[i]]$itms)

resultData[[j]]$ratings = append(resultData[[j]]$ratings, newData[[i]]$ratings)

print(resultData[[j]]$itms)

hasSameUserID = TRUE

break

}

}

if (hasSameUserID == FALSE) {

print("Does not have same ID")

resultData = append(resultData, newData[[i]])

}

}

resultData

}

updatedFormedData <- updateUserData(oldData = formedTrain, newData = testset)

# Mean absolute prediction error (MAPE) method. Takes in two lists/vectors and calculates MAPE

calculateMAPE <- function(results) {

intermediateResult = sum((results$actualValue - results$forecastValue)/results$actualValue)

mape = (1/length(results))\*intermediateResult

}

# Probability of guessing exactly correctly (PGEC) method

# Setup data

set.seed(9999)

testidxs <- sample(1:nrow(ivl), 1000)

testset <- ivl[testidxs,]

trainset <- ivl[-testidxs,]

formedTestSetData <- formUserData(testset[,1:3])

formedTrainSetData <- formUserData(trainset[,1:3])

# Predict example test set data

predict.usrData(formedTrainSetData, formedTestSetData, 111, 10, locMeasure = mean)

# Predict for the entire test set Draft 1

predict\_func = function(studentID, instructorID, k\_neighbours, locMeasure = mean) {

predict.usrData(origData = formedTrainSetData, newData = formedTestSetData[[studentID]], newItem = instructorID, k = k\_neighbours, locMeasure = locMeasure)

}

test\_prediction = predict\_func(21, 122, 10, locMeasure = mean)

predicted\_values\_for\_testset = lapply(formedTestSetData, predict\_func, w)

Mx:

set.seed(9999) # so we are all using the same random numbers

testidxs <- sample(1:nrow(ivl),1000)

testset <- ivl[testidxs,]

trainset <- ivl[-testidxs,]

formedTrain <- formUserData(trainset[,1:3])

formedSet <- formUserData(testset[,1:3])

predict(formedTrain,formedSet[[27]],122,10,locMeasure=mean)

|  |  |  |  |
| --- | --- | --- | --- |
| User ID | Actual | Predict | Difference |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Code for analyzing:

usr\_id <- vector()

actual <- vector()

predict <- vector()

difference <- vector()

#for each k value, do the following process once

#for each user in testset, predict each item and put the data in frame

lapply(1:25, function(k) {

lapply(formedTest, function(usr) {

itmNumList <- 1:length(usr$ratings)

lapply(itmNumList, function(num) {

#remove the record from the user to predict

holdRating = usr$ratings[[num]]

tmpUsr <- usr

tmpUsr$itms <- usr$itms[-num]

tmpUsr$ratings <- usr$ratings[-num]

result <- predict(formedTrain,tmpUsr,usr$itms[num],k,locMeasure=median)

diff = result - holdRating

#append(usr\_id, usr$userID)

usr\_id <<- c(usr\_id, usr$userID)

actual <<- c(actual, holdRating)

predict <<- c(predict, result)

difference <<- c(difference, diff)

})

})

})

kNNResult <- data.frame(

usr\_id,

actual,

predict,

difference

)

#kNNResult

write.csv(kNNResult, file = "meanResult.csv")

#updated on tues, 16th Oct. by MX

############ Update by Chloe on Tuesday 1:14PM

## calculate the MAPE error

MAPE = function(df\_list) {

error\_percentage = sum(abs(df\_list['difference'])/abs(df\_list['actual']))/nrow(df\_list)

return (round(error\_percentage, 3))

}

## calculate the PGEC error

PGEC = function(df\_list) {

accuracy = length(which((df\_list['difference'] == 0.0) == TRUE))/ nrow(df\_list)

return (accuracy)

}

file\_list = c("meanResult.csv", 'vecmodeResult.csv', 'medianResult.csv')

final\_df\_make = function(input\_file){

data = read.csv(file = input\_file, header = TRUE, sep = ",")

k\_value = rep(1:25, each = 1000)

new\_data = cbind(k\_value, data)

data\_split = split(new\_data, new\_data$k\_value)

mape\_df = data.frame(sapply(data\_split, MAPE))

names(mape\_df) = c('MAPE\_Error')

pgec\_df = data.frame(sapply(data\_split, PGEC))

names(pgec\_df) = c('PGEC\_Accuracy')

final\_error\_df = cbind(mape\_df, pgec\_df)

graph\_df = cbind(k\_value = rownames(final\_error\_df), final\_error\_df)

return (graph\_df)

}

library(ggplot2)

plot\_graph = function(dataframe, data\_col, colname){

ggplot(dataframe, aes(x = factor(k\_value, levels = dataframe$k\_value),

y = data\_col)) +

geom\_bar(stat = 'identity', fill = 'steelblue') +

geom\_text(aes(label = data\_col), vjust = -0.5, color = "black") +

xlab('K\_Value') +

ylab(colname)

}

create\_graph = function(input\_file){

graph\_df = final\_df\_make(input\_file)

plot\_graph(graph\_df, graph\_df$MAPE\_Error, 'MAPE\_Error')

plot\_graph(graph\_df, graph\_df$PGEC\_Accuracy, 'PGEC\_Accuracy')

}

## Use lapply to run the create\_graph function for other csv files (median and mode)

# lapply(file\_list, create\_graph)

input\_file = "meanResult.csv"

graph\_df = final\_df\_make(input\_file)

plot\_graph(graph\_df, graph\_df$MAPE\_Error, 'MAPE\_Error')

plot\_graph(graph\_df, graph\_df$PGEC\_Accuracy, 'PGEC\_Accuracy')