# Author: Yun-Ching Chen

#

# Copyright 2019 Yun-Ching Chen

#

# Permission is hereby granted, free of charge, to any person obtaining a copy of this software

# and associated documentation files (the "Software"), to deal in the Software without restriction,

# including without limitation the rights to use, copy, modify, merge, publish, distribute,

# sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is

# furnished to do so.

#

# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,

# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR

# PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE

# FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,

# ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN

# THE SOFTWARE.

#

library(Seurat)

library(ggplot2)

library(dplyr)

library(reshape2)

library(PRROC)

library(WriteXLS)

library(rpart)

library(stringr)

library(rpart.plot)

ExpectedLogW <- function(pc.ranges, one.clustering){

dist.expected <- sqrt(sum(apply(pc.ranges, 2, function(r){(r[1]-r[2])^2/6})))

dist.sum.expected <- 0.5\*sum(sapply(table(one.clustering), function(n){dist.expected\*(n-1)}))

return(log(dist.sum.expected))

}

ObservedLogW <- function(X, one.clustering){

if(is.data.frame(X)) X <- as.matrix(X)

dist.sum.observed <- log(0.5 \* sum(vapply(split(seq\_len(nrow(X)), one.clustering), function(I) {

xs <- X[I, , drop = FALSE]

sum(dist(xs)/nrow(xs))

}, 0)))

return(dist.sum.observed)

}

GapStatistic <- function(x, clusterings){

gap.stat <- data.frame(matrix(NA, ncol = 3, nrow = length(clusterings)))

colnames(gap.stat) <- c("E.log.W","log.W","gap")

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

gap.stat[i,"E.log.W"] <- ExpectedLogW(apply(x, 2, range), clusterings[[i]])

gap.stat[i,"log.W"] <- ObservedLogW(x, clusterings[[i]])

}

gap.stat[,"gap"] <- gap.stat[,"E.log.W"] - gap.stat[,"log.W"]

return(gap.stat)

}

BottomUpMerge <- function(sobj, k.max, npc, random.seed){

k.clustering <- 0

clusterings <- list()

clust.r <- 1.0

sobj <- FindNeighbors(sobj, reduction = "pca", dims = 1:npc, verbose = FALSE)

sobj <- FindClusters(sobj, resolution = clust.r, random.seed = random.seed, verbose = FALSE)

cat("Iteration for nPC =",npc,", r = 1.0")

while(length(unique(sobj@active.ident)) < k.max){

clust.r <- clust.r + 0.2

cat(",", clust.r)

sobj <- FindClusters(sobj, resolution = clust.r, random.seed = random.seed, verbose = FALSE)

}

cat("\n")

clusterings[[(k.clustering <- length(unique(sobj@active.ident)))]] <- as.character(sobj@active.ident)

# Merging clusters by nearest centers

while(k.clustering > 2){

merged <- NearestCluster(sobj@reductions$pca@cell.embeddings[,1:npc], clusterings[[k.clustering]])

clustering.merged <- clusterings[[k.clustering]]

clustering.merged[which(clustering.merged == merged[1])] <- merged[2]

clusterings[[k.clustering-1]] <- as.character(as.integer(as.factor(clustering.merged)))

k.clustering <- k.clustering - 1

}

clusterings[[1]] <- rep("1", nrow(sobj@reductions$pca@cell.embeddings))

return(clusterings[1:k.max])

}

NearestCluster <- function(x, clustering){

clust.center <- data.frame(matrix(NA, ncol = length(unique(clustering)), nrow = ncol(x)))

colnames(clust.center) <- unique(clustering)

for(clust in unique(clustering)){

clust.center[,clust] <- if(length(which(clustering == clust))==1) as.numeric(x[which(clustering == clust),]) else

as.numeric(apply(x[which(clustering == clust),], 2, mean))

}

dist.mat <- as.matrix(dist(t(clust.center)))

diag(dist.mat) <- Inf

cluster.nearest <- colnames(clust.center)[which(upper.tri(dist.mat)\*(dist.mat == min(dist.mat)) == 1, arr.ind = T)[1,]]

return(cluster.nearest)

}

SelectCandidate <- function(gap.gain){

gg.mean <- mean(as.matrix(gap.gain))

gg.sd <- sd(as.matrix(gap.gain))

gap.gain.candidates <- sort(apply(gap.gain\*(gap.gain > gg.mean + gg.sd), 2, max), decreasing = T)

gap.gain.candidates <- gap.gain.candidates[which(gap.gain.candidates > 0)]

k.candidates <- c()

pc.candidates <- c()

if(length(gap.gain.candidates) == 0){

max.ind <- which(gap.gain == max(gap.gain), arr.ind = T)

k.candidates <- c(max.ind[1,2]+1)

pc.candidates <- c(as.integer(rownames(gap.gain)[max.ind[1,1]]))

} else {

for(i in gap.gain.candidates){

ind <- which(gap.gain == i, arr.ind = T)

if(length(k.candidates) == 0){

k.candidates <- c(ind[1,2]+1)

pc.candidates <- c(as.integer(rownames(gap.gain)[ind[1,1]]))

} else {

k.temp <- ind[1,2]+1

pc.temp <- as.integer(rownames(gap.gain)[ind[1,1]])

if(all(k.temp > k.candidates) && all(pc.temp > pc.candidates)){

k.candidates <- c(k.candidates, k.temp)

pc.candidates <- c(pc.candidates, pc.temp)

}

}

}

}

return(list(k = k.candidates, pc = pc.candidates))

}

ComputeMarkers <- function(sobj, gap.gain, candidates, out.dir){

markers.all <- list()

out.xls <- list()

out.xls$gap.gain <- cbind(PC\_K = rownames(gap.gain), gap.gain)

for(i in 1:length(candidates$k)){

clustering.label <- paste0("PC",candidates$pc[i],"K",candidates$k[i])

Idents(sobj) <- sobj@meta.data[[clustering.label]]

sobj <- RunUMAP(sobj, dims = 1:candidates$pc[i], verbose = FALSE)

ggsave(DimPlot(sobj, reduction = "umap"), filename = paste0(out.dir,"/",clustering.label,"\_UMAP.pdf"))

sobj.markers <- FindAllMarkers(object= sobj, only.pos = TRUE, min.pct = 0.25, thresh.use = 0.25)

sobj.markers$AUROC <- NA

for(j in 1:nrow(sobj.markers)){

# May need to change to use another slot for analysis

sobj.markers$AUROC[j] <- roc.curve(scores.class0 = GetAssayData(sobj)[sobj.markers$gene[j],], weights.class0 = sobj@active.ident == sobj.markers$cluster[j])$auc

}

top.10 <- sobj.markers %>% group\_by(cluster) %>% top\_n(10, avg\_log2FC)

ggsave(DoHeatmap(object = sobj, features = top.10$gene) + NoLegend(),

filename = paste0(out.dir,"/",clustering.label,"\_DE\_genes\_LCF.png"), units = "in", width = 12, height = 8)

out.xls[[clustering.label]] <- sobj.markers[,c("gene","p\_val","avg\_log2FC","pct.1","pct.2","p\_val\_adj","cluster","AUROC")]

markers.all[[clustering.label]] <- sobj.markers

}

WriteXLS(out.xls, ExcelFileName = paste0(out.dir,"/data.xls"))

saveRDS(markers.all, file = paste0(out.dir,"/markers.all.rds"))

return(markers.all)

}

DecisionTree <- function(sobj, markers, out.dir, plot.decision.tree){

data.rpart <- list()

for(candidate in names(markers)){

data.rpart[[candidate]] <- list()

genes.candidate <- unique(markers[[candidate]]$gene[which(markers[[candidate]]$p\_val\_adj < 0.01)])

for(clust in as.character(unique(markers[[candidate]]$cluster))){

if(length(genes.candidate) == 0){

next

} else if(length(genes.candidate) == 1){

# May need to change to use another slot for analysis

data <- data.frame(as.factor(sobj@meta.data[[candidate]] == clust), as.numeric(GetAssayData(sobj)[genes.candidate,]))

colnames(data) <- c("label", genes.candidate)

} else {

# May need to change to use another slot for analysis

data <- as.data.frame(t(as.matrix(GetAssayData(sobj)[genes.candidate,])))

data$label <- as.factor(sobj@meta.data[[candidate]] == clust)

}

data.rpart[[candidate]][[clust]] <- rpart(label ~., data = data)

}

}

summary.rpart <- data.frame(matrix(NA, nrow = length(markers), ncol = 20))

rownames(summary.rpart) <- names(markers)

colnames(summary.rpart) <- paste0("s",1:20)

for(candidate in names(data.rpart)){

for(nsplit in 1:20){

err.rate <- 0

for(clust in unique(as.character(sobj@meta.data[[candidate]]))){

if(is.null(data.rpart[[candidate]][[clust]])){

err.rate <- err.rate + 1.0

} else {

err.rate <- err.rate +

data.rpart[[candidate]][[clust]]$cptable[max(which(data.rpart[[candidate]][[clust]]$cptable[,"nsplit"] <= nsplit)),"rel error"]\*

length(which(sobj@meta.data[[candidate]] == clust))/nrow(sobj@meta.data)

}

}

summary.rpart[candidate, nsplit] <- err.rate

}

}

# plot decision tree

pdf(paste0(out.dir,"/DT\_plot.pdf"))

if(plot.decision.tree){

for(candidate in names(data.rpart)){

clust.sorted <- unique(as.character(sobj@meta.data[[candidate]]))

if(!any(is.na(suppressWarnings(as.integer(clust.sorted))))) clust.sorted <- as.character(sort(as.integer(clust.sorted)))

for(clust in clust.sorted){

if(!is.null(data.rpart[[candidate]][[clust]])){

rpart.plot(data.rpart[[candidate]][[clust]], roundint = F, main = paste0(candidate,":",clust))

}

}

}

}

dev.off()

saveRDS(data.rpart, file = paste0(out.dir,"/DT.rds"))

saveRDS(summary.rpart, file = paste0(out.dir,"/DT\_summary.rds"))

return(summary.rpart)

}

PlotSummary <- function(gap.gain, summary.rpart, markers.all, out.dir){

data.plot <- melt(as.matrix(gap.gain))

colnames(data.plot) <- c("PC","k","gap.gain")

data.plot$PC <- factor(data.plot$PC, levels = rev(rownames(gap.gain)))

data.plot$k <- factor(data.plot$k, levels = colnames(gap.gain))

temp <- str\_split\_fixed(gsub("^PC","",rownames(summary.rpart)),"K",2)

best.ind <- if(nrow(summary.rpart) > 1) which.min(apply(summary.rpart[,5:15], 1, mean)) else 1

data.plot$label <- ""

for(i in 1:nrow(temp)) data.plot$label[which(data.plot$PC == temp[i,1] & data.plot$k == temp[i,2])] <- "X"

data.plot$label[which(data.plot$PC == temp[best.ind,1] & data.plot$k == temp[best.ind,2])] <- "B"

p <- ggplot(data = data.plot, aes(x = k, y=PC, fill = gap.gain, label = label)) + geom\_tile() + scale\_x\_discrete(position = "top") + geom\_text() +

scale\_fill\_gradient(low = "darkgrey", high = "darkred") + labs(x="# of clusters (k)",y="# of top PCs")

ggsave(p, filename = paste0(out.dir,"/PC\_K.pdf"))

}

IKAP <- function(sobj, pcs = NA, pc.range = 20, k.max = NA, r.kmax.est = 1.5, out.dir = "./IKAP", scale.data = TRUE, find.var.features = TRUE,

confounders = c('percent.mt','nFeature\_RNA'), plot.decision.tree = TRUE, random.seed = 0){

dir.create(out.dir, recursive = T)

if(DefaultAssay(sobj) == "SCT" || (DefaultAssay(sobj) == "integrated" && sobj@commands$FindIntegrationAnchors$normalization.method == "SCT")){

cat("SCT or SCT-integrated data is used. Skip data scaling.\n")

}

if(scale.data && DefaultAssay(sobj) != "SCT" &&

!(DefaultAssay(sobj) == "integrated" && sobj@commands$FindIntegrationAnchors$normalization.method == "SCT")){

if(!all(confounders %in% colnames(sobj@meta.data))){

warning(confounders[which(!confounders %in% colnames(sobj@meta.data))],"not in Seurat metadata: skipped for regression.\n")

}

cat("Scaling and removing confounders.\n")

confounders <- intersect(confounders, colnames(sobj@meta.data))

if(length(confounders) > 0) sobj <- ScaleData(sobj, vars.to.regress = confounders)

else sobj <- ScaleData(sobj)

}

if(DefaultAssay(sobj) == "SCT" || DefaultAssay(sobj) == "integrated"){

cat("SCT or integrated data is used. Skip data variable feature finding.\n")

}

if(find.var.features && DefaultAssay(sobj) != "SCT" && DefaultAssay(sobj) != "integrated"){

sobj <- FindVariableFeatures(sobj, selection.method = "vst", nfeatures = 2000)

}

cat("Running PCA ... \n")

if(is.na(pcs)){

sobj <- RunPCA(sobj, npcs = 50)

pc.change <- which(abs(diff(sobj@reductions$pca@stdev)/sobj@reductions$pca@stdev[2:length(sobj@reductions$pca@stdev)]) > 0.1)

while(length(pc.change) > 0 && max(pc.change)+pc.range+2 > length(sobj@reductions$pca@stdev)){

sobj <- RunPCA(sobj, pcs.compute = max(pc.change)+pc.range+2+10, do.print = F)

pc.change <- which(abs(diff(sobj@reductions$pca@stdev)/sobj@reductions$pca@stdev[2:length(sobj@reductions$pca@stdev)]) > 0.1)

}

pcs <- if(length(pc.change) == 0) 2:(pc.range+2) else (max(pc.change)+2):(max(pc.change)+pc.range+2)

} else {

sobj <- RunPCA(sobj, pcs.compute = max(pcs))

}

if(is.na(k.max)){

cat("Determine k.max.\n")

sobj <- FindNeighbors(sobj, reduction = "pca", dims = 1:min(pcs), verbose = FALSE)

sobj <- FindClusters(sobj, resolution = r.kmax.est, random.seed = random.seed, verbose = FALSE)

k.min.pc <- length(unique(sobj@active.ident))

sobj <- FindNeighbors(sobj, reduction = "pca", dims = 1:max(pcs), verbose = FALSE)

sobj <- FindClusters(sobj, resolution = r.kmax.est, random.seed = random.seed, verbose = FALSE)

k.max.pc <- length(unique(sobj@active.ident))

k.max <- as.integer((k.min.pc + k.max.pc)/2)

cat("k.max =", k.max, "\n")

}

gap.gain <- data.frame(matrix(NA, ncol = k.max - 1, nrow = length(pcs)))

colnames(gap.gain) <- as.character(2:k.max)

rownames(gap.gain) <- paste0(pcs)

cat("Perform clustering for every nPC:\n")

for(npc in pcs){

clusterings <- BottomUpMerge(sobj, k.max, npc, random.seed)

gap.stat <- GapStatistic(sobj@reductions$pca@cell.embeddings[,1:npc], clusterings)

names(clusterings) <- paste0("PC",npc,"K",1:k.max)

sobj@meta.data <- cbind(sobj@meta.data, as.data.frame(clusterings)[,2:k.max])

gap.gain[as.character(npc),] <- diff(gap.stat$gap)

}

candidates <- SelectCandidate(gap.gain)

cat("Compute marker gene lists ... \n")

markers.all <- ComputeMarkers(sobj, gap.gain, candidates, out.dir)

cat("Build decision tree ... \n")

summary.rpart <- DecisionTree(sobj, markers.all, out.dir, plot.decision.tree)

cat("Plotting summary ... \n")

PlotSummary(gap.gain, summary.rpart, markers.all, out.dir)

return(sobj)

}