Homewok5

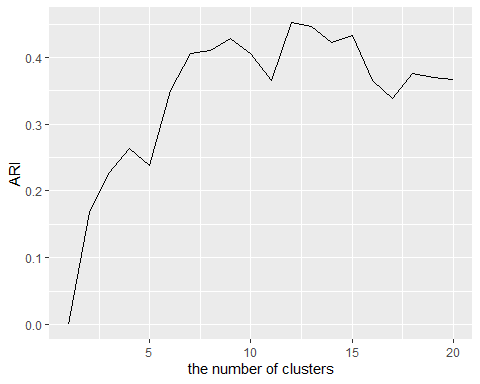
Jun Rao

9/8/2020

library(stats)  
library(ggplot2)  
library(data.table)  
library(kernlab)  
  
# use data.table::fread to read the compressed CSV data file into R as a data table.  
dt<- fread("zip.test.gz")  
  
#To exclude the label column (that is the first column)  
df<-as.matrix(dt[, -1])

Spectral clustering involves computing an N x N affinity/kernel matrix (quadratic time/space in N), then an eigen-decomposition (cubic time in N), which results in a matrix of eigenvectors, which are new features/column that can be clustered with a standard algorithm like kmeans. Spectral clustering is useful for detecting clusters that are non-linear/non-spherical in shape. 1.Use kernlab::specc in R to compute a spectral clustering on the first 300 rows of the zip.test data, from 1 to 20 clusters. Compute ARI of each clustering and plot y=ARI as a function of x=number of clusters using default parameters of specc. What number of clusters is the most accurate?

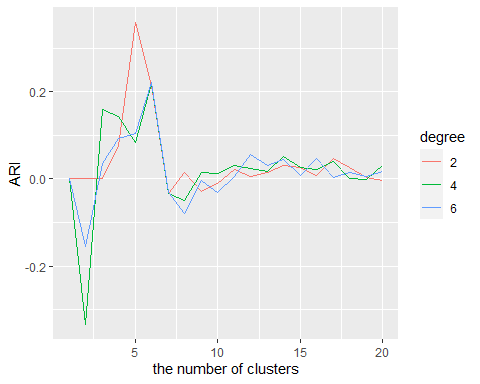
metrics.dt.list <- list()  
  
 for(n.clusters in 1: 20){  
 if(n.clusters == 1){  
 result = data.table(n.clusters,ARI=0)  
 }  
 else{  
 fit <- specc(df[1:300,],n.clusters)  
 result <- data.table(n.clusters,ARI=pdfCluster::adj.rand.index(fit@.Data, dt[1:300,][["V1"]]))  
   
 }  
 metrics.dt.list[[paste(n.clusters)]] <- result  
   
  
 }  
 metrics.dt <- do.call(rbind, metrics.dt.list)  
  
   
 ggplot()+geom\_line(aes(n.clusters, ARI),data=metrics.dt) + xlab("the number of clusters") + ylab("ARI")



*From the picture, we can find that when the cluster is 12, it is the accurate.*

2.Again compute spectral clusterings but using a polynomial kernel (kernlab:polydot) instead of the default RBF kernel. Compute clusterings using three different polynomial degree values (2, 4, 6) and 1 to 20 clusters. Compute ARI of each clustering and plot y=ARI as a function of x=number of clusters, using different colors for the different polynomial degrees. What number of clusters and polynomial degree is most accurate? accurate?

dt <- data.table(matrix(numeric(), 60, 3))  
 index = 1  
 for(n.clusters in 1: 20){  
 for (degree in seq(2,6,2)) {  
 if(n.clusters == 1){  
 temp <- c(n.clusters,degree,0)  
 dt[index,names(dt) := as.list(temp)]  
 }  
 else{  
 fit <- specc(df[1:300,],n.clusters,polydot(degree=degree))  
 ARI = pdfCluster::adj.rand.index(fit@.Data, dt[1:300,][["V1"]])  
 temp <- c(n.clusters,degree,ARI)  
 dt[index,names(dt) := as.list(temp)]  
 }  
   
  
 index = index + 1  
 }  
   
   
 }  
  
  
# g <- ggplot()+geom\_line(aes(V1, V3),data=dt) + xlab("the number of clusters") + ylab("ARI")  
#   
# g + facet\_grid(rows = vars(V2))  
ggplot( aes(x=V1, y=V3, group=V2, color=factor(V2)),data=dt) +  
 geom\_line() +   
 labs(x = "the number of clusters", y="ARI", color = "degree")



*From the picture, we can find that the most accurate at*

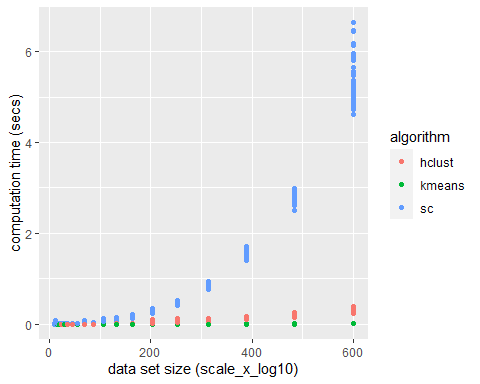
*when degree = 2 and cluster = 5*

*when degree = 4 and cluster = 6*

*when degree = 6 and cluster = 6*

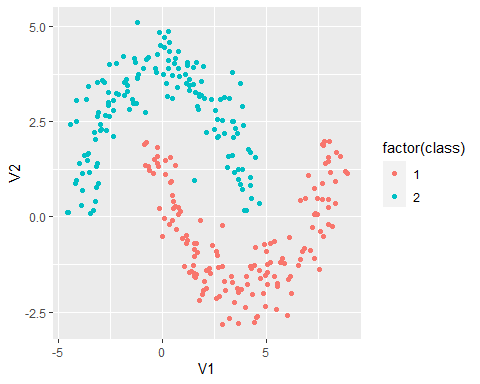
3.Use microbenchmark to compute timings of kmeans, hclust, and specc for several subset sizes N. Plot y=time in seconds versus x=subset size, with each algorithm in a different color. Use log/log scales so that different time complexity classes appear as lines with different slopes. Use a large enough range of N so that the complexity class of each algorithm is clear (kmeans linear, hclust quadratic, specc cubic).

Myplot <- function(data.set){  
 time.dt.list <- list()  
 (log.scale.seq <- as.integer(c(10^seq(1, log10(600), l=20))))  
 # seq(10, 1000, by =100)  
 for (N in log.scale.seq) {  
 X <- data.set[1:N,]  
 timing.df <- microbenchmark::microbenchmark(hclust={  
 d.mat <- stats::dist(X, method="manhattan")  
 as.matrix(d.mat)  
 cl.tree <- stats::hclust(d.mat, method="single")  
 stats::cutree(cl.tree, k=3)  
 },   
 kmeans={  
 stats::kmeans(X, 3)  
 },  
 sc ={  
 specc(X,3)  
 })  
 time.dt.list[[paste(N)]] <- data.table(N, timing.df)  
 }  
 time.dt <- do.call(rbind, time.dt.list)  
 time.dt[, data.size := N]  
 time.dt[, time.seconds := time/1e9]  
 time.dt[, algorithm := expr]  
   
 g<-ggplot()+  
 geom\_point(aes(x=data.size,y=time.seconds,color=algorithm),data=time.dt)+  
 xlab("data set size (scale\_x\_log10)") + ylab("computation time (secs)")  
   
   
 return(g)  
}  
  
  
Myplot(df[1:1000,])



Code a simple spectral clustering algorithm from scratch (without using any special packages like kernlab), using the descriptions in the textbooks. Compute a kernal/similarity matrix, then either the un-normalized or normalized graph Laplacian, then compute its eigenvalues/vectors, then consider the eigenvectors corresponding to the smallest eigenvalues, then run stats::kmeans. Run the algorithm on the halfcircle/moons data we saw in class, and draw the two different clusters in different colors.

# spectral clustering  
SPC <- function(data.matrix, K){  
   
 #we total have two parameters in the function  
 if(is.na(data.matrix) || is.na(K)){  
 stop("You need to input valid parameters!!")  
 }  
   
 nodes <- dim(data.matrix)[1]  
   
 #initialize Similarity matrix or adjacency matrix  
 similarity.matrix <- matrix(NA, nrow = nodes, ncol=nodes)  
   
 # Using Gaussian kernel (sigma = 0.09) to cluster the data  
 #if two points have the small similarity value,  
 #these two points will have a stronger link  
   
 sigma <- 0.09  
   
 for (i in 1:nodes){  
 for (j in 1: nodes){  
 similarity.matrix[i,j] <- exp(-sum((data.matrix[i,] - data.matrix[j,])^2)/(2\*sigma^2))  
 }  
 }  
   
 #calculate the degree matrix  
 #it means how many connection between  
 #the first data point to the rest N-1 data points  
 S\_degree <- rowSums(similarity.matrix)  
   
 #calculate Laplacian   
 Laplacian <- diag(S\_degree) - similarity.matrix  
 normalize.Laplacian <- diag(S\_degree^(-1/2)) %\*% similarity.matrix %\*% diag(S\_degree^(-1/2))  
   
 #error = sum()  
 # ev <- eigen(Laplacian)  
 #   
 # e <- kmeans(ev$vectors[,which.min(ev$values)], K, iter.max=50)$cluster  
   
 ev <-eigen(normalize.Laplacian)  
 ev.vector <- ev$vectors[,1:K]  
  
 for (index in 1:nodes) {  
 ev.vector[index,] <- ev.vector[i,] / sqrt(sum(ev.vector[index,]^2))  
 }  
  
 e <- kmeans(ev.vector, K, iter.max=50)$cluster  
   
 return(e)  
}  
  
  
  
  
set.seed(1)  
halfcircle <- function(r, center = c(0, 0), class, sign, N=150, noise=0.5) {  
 angle <- runif(N, 0, pi)  
 rad <- rnorm(N, r, noise)  
 data.table(  
 V1 = rad \* cos(angle) + center[1],  
 V2 = sign \* rad \* sin(angle) + center[2],  
 class = factor(class))  
}  
  
  
X.dt <- rbind(  
 halfcircle(4, c(0, 0), 1, 1),  
 halfcircle(4, c(4, 2), 2, -1))  
  
X.mat <- as.matrix(X.dt[, 1:2])  
  
K <- 2  
class = SPC(X.mat,K)  
  
  
DF <- as.data.frame(cbind(X.mat,class))  
  
ggplot()+  
 geom\_point(aes(  
 V1, V2, color=factor(class)),  
 data=DF)



*I code my own function by using Gaussian kernel (sigma = 0.09) to build similarity matrix. I find something very interesting, when sigma = 0.09, the answer is most accurate.*