**5.1**

*(a)*

The silhouette index for each datapoint is shown below.

A close up of a piece of paper

Description automatically generated

A screenshot of a cell phone

Description automatically generated

*(b)*

The average silhouette index for different clustering choices is plotted below:

A picture containing man

Description automatically generated

Here we see that the silhouette index peaks at . We know this is the optimal choice because “*simdatxy*” was generated from four Gaussian distributions centered at , , and with the same standard deviation of 2. At , we may approximate and . Hence .

*(c)*

400 random datapoints, whose x and y components follow uniform distributions in the range , are generated using the “*runif()*” function.

A picture containing drawing

Description automatically generated

By varying the number of clusters, we see that the silhouette index fluctuates around 0.375 and its profile lacks an obvious peak. As a result, we cannot infer what number of clusters is optimal.

A screenshot of a cell phone

Description automatically generated

*R Code:*

#5.1

#a

library("dplyr")

simdat = lapply(c(0, 8), function(mx) {

lapply(c(0,8), function(my) {

tibble(x = rnorm(100, mean = mx, sd = 2),

y = rnorm(100, mean = my, sd = 2),

class = paste(mx, my, sep = ":"))

}) %>% bind\_rows

}) %>% bind\_rows

simdatxy = simdat[, c("x", "y")]

library("cluster")

pam4 <- pam(simdatxy, 4)

sil <- silhouette(pam4, 4)

index <- order(as.array(as.numeric(row.names(sil))))

sil[index,]

plot(sil, col=c("red","green","blue","purple"), main="Silhouette")

#b

sil\_wid <- function(k){

p <- pam(simdatxy,k)

s <- silhouette(p,k)

return(mean(s[,"sil\_width"]))

}

k\_vector <- c(2:11)

width\_vector <- c()

for (val in k\_vector){

val

w <- sil\_wid(val)

width\_vector <- c(width\_vector,w)

}

plot(k\_vector,width\_vector,xlab = "K", ylab = "silhouette index")

#c

library(ggplot2)

simdatxy2 <- tibble(x = runif(400,min=-10,max=10), y = runif(400,min=-10,max=10))

ggplot(simdatxy2, aes(x = x, y = y)) + geom\_point() +

coord\_fixed()

sil\_wid2 <- function(k){

p <- pam(simdatxy2,k)

s <- silhouette(p,k)

return(mean(s[,"sil\_width"]))

}

k\_vector <- c(2:20)

width\_vector <- c()

for (val in k\_vector){

val

w <- sil\_wid2(val)

width\_vector <- c(width\_vector,w)

}

plot(k\_vector,width\_vector,xlab = "K", ylab = "silhouette index")

**5.3**

*(a)*

*“kmeans()”* was used to separate the *spirals* data into 2 clusters. It is evident that the result is far from satisfactory.

A close up of a mans face

Description automatically generated

*(b)*

Using *“dbscan::dbscan()”* has proven to provide us with better results, except a small amount of misclassifications at the tail. The *eps* and *minPts* parameters were 0.15 and 3 respectively.

The density-based clustering method starts from a randomly selected seed. Branching out from that seed, all data points within *eps* are labeled as the next frontier where breadth-first search is executed. When all nodes of the outermost layer no longer have enough close-by neighbours to hop to, a new class is created along with a new seed and the algorithm repeats [1].

A close up of a mans face

Description automatically generated

*(c)*

As soon as the *eps* and *minPts* parameters leaves their ranges of tolerance and , the clustering algorithm fails and produces too many groups.

A close up of a mans face

Description automatically generated

Setting *minPts* constant at 3, sweeping *eps* from 0.01 to 1 with increments of 0.01 changes the number of clustering produced as shown in the Figure below:

A screenshot of a cell phone

Description automatically generated

Setting *eps* constant at 0.15, sweeping *minPts* from 1 to 20 with increments of 1 changes the number of clustering produced as shown in the Figure below:

A screenshot of a cell phone

Description automatically generated

It is thus evident that the clustering algorithm used by *dbscan* is not robust for this non-convex set. The more efficient way would be to use kernel clustering to transform the datapoints into a linearly separable space.

*R Code:*

#5.3

#a

library(kernlab)

data(spirals)

s\_data <- tibble(x = spirals[,1], y = spirals[,2],cluster = as.factor(kmeans(spirals, centers=2)$cluster))

ggplot(s\_data, aes(x = x, y = y, col = cluster)) + geom\_point() + coord\_fixed()

#b

library(dbscan)

s\_data2 <- tibble(x = spirals[,1], y = spirals[,2],cluster = as.factor(dbscan::dbscan(spirals, eps = 0.15, minPts = 3)$cluster))

ggplot(s\_data2, aes(x = x, y = y, col = cluster)) + geom\_point() + coord\_fixed()

#c

s\_data3 <- tibble(x = spirals[,1], y = spirals[,2],cluster = as.factor(dbscan::dbscan(spirals, eps = 0.15, minPts = 3)$cluster))

ggplot(s\_data3, aes(x = x, y = y, col = cluster)) + geom\_point() + coord\_fixed()

e\_vector <- seq(0.01,1,0.01)

ncluster\_vector <- vapply(e\_vector, function(x){length(unique(as.factor(dbscan::dbscan(spirals, eps = x, minPts = 3)$cluster)))},numeric(1))

plot(e\_vector,ncluster\_vector, xlab="Parameter eps", ylab="Number of Clusters")

P\_vector <- seq(1,20,1)

ncluster\_vector2 <- vapply(P\_vector, function(x){length(unique(as.factor(dbscan::dbscan(spirals, eps = 0.15, minPts = x)$cluster)))},numeric(1))

plot(P\_vector,ncluster\_vector2, xlab="Parameter minPts", ylab="Number of Clusters")

Reference

[1] “DBSCAN Clustering Tutorial - Nearist.ai - Medium.” [Online]. Available: https://medium.com/nearist-ai/dbscan-clustering-tutorial-dd6a9b637a4b. [Accessed: 27-Jun-2020].