Week 7 Code Examples, CSC 400, Spring 2024

1. DBSCAN
2. BIRCH
3. SOM
4. Fuzzy C-Means
5. Mean Shift Clustering
6. OPTICS
7. Visualization

DBSCAN

We'll start by loading packages and a practice dataset. Then we'll visualize it.

```
library(factoextra)
data("multishapes")
df <- multishapes[, 1:2]
set.seed(123)
km.res <- kmeans(df, 5, nstart = 25)
fviz_cluster(km.res, df, frame = FALSE, geom = "point")
```

We can use the dbscan function in the fpc package, but since there is also one in the dbscan package with the same name, we specify which package version we mean.

```
library(fpc)
library(dbscan)
data("multishapes", package = "factoextra")
df <- multishapes[, 1:2]
set.seed(123)
db <- fpc::dbscan(df, eps = 0.15, MinPts = 5)
plot(db, df, main = "DBSCAN", frame = FALSE)

fviz_cluster(db, df, stand = FALSE, frame = FALSE, geom = "point")
```

We can print the results, including which cluster each point belongs to. If it's not in a cluster, it's coded as 0, which is either noise or an outlier.

```
print(db)
db$cluster[sample(1:1089, 50)]
```

We can also try to optimize parameters for the best fit.

```
dbscan::kNNdistplot(df, k =  5)
abline(h = 0.15, lty = 2)
```

Reference [2] includes an additional example using the iris dataset.

BIRCH

We will look at a simple example with simulated data.

```
library(stream)
stream <- DSD_Gaussians(k = 3, d = 2)

BIRCH <- DSC_BIRCH(threshold = .1, branching = 8, maxLeaf = 20)
update(BIRCH, stream, n = 500)
BIRCH

plot(BIRCH, stream)
```

SOM

We'll look at a SOM model using some simulated data. First, we create our data.

```
sample.size <- 10000
sample.rgb <- as.data.frame(matrix(nrow = sample.size, ncol = 3))
colnames(sample.rgb) <- c('R', 'G', 'B')

sample.rgb$R <- sample(0:255, sample.size, replace = T)
sample.rgb$G <- sample(0:255, sample.size, replace = T)
sample.rgb$B <- sample(0:255, sample.size, replace = T)
```

Now, we load our packages and create the SOM model.

```
library(kohonen)

grid.size <- ceiling(sample.size ^ (1/2.5))
som.grid <- somgrid(xdim = grid.size, ydim = grid.size, topo = 'hexagonal', toroidal = T)
som.model <- som(data.matrix(sample.rgb), grid = som.grid)

som.events <- som.model$codes[[1]]
som.events.colors <- rgb(som.events[,1], som.events[,2], som.events[,3], maxColorValue = 255)
som.dist <- as.matrix(dist(som.events))
```

We can compare the results of the data before and after the model.

```
plot(som.model,
      type = 'mapping',
      bg = som.events.colors[sample.int(length(som.events.colors),
                                         size = length(som.events.colors))],
      keepMargins = F,
      col = NA,
      main = '')

plot(som.model,
      type = 'mapping',
      bg = som.events.colors,
      keepMargins = F,
      col = NA,
      main = '')
```

Next question is how many clusters should we have?

Here's another example on a wine dataset.

```
data(wines)
head(wines)
scale(wines)
head(scale(wines))
grid <- somgrid(xdim = 5, ydim = 5, topo = "hexagonal")
som.wines <- som(scale(wines), grid = somgrid(xdim = 5, ydim = 5, "hexagonal"))
str(som.wines)
plot(som.wines, type = "mapping")
som.wines$grid$pts
som.wines$unit.classif
som.wines$codes[[1]]
dist(som.wines$codes[[1]])
hclust(dist(som.wines$codes[[1]]))
peta <- cutree(hclust(dist(som.wines$codes[[1]])), 5)
plot(peta)
plot(som.wines, type = "codes", bgcol = rainbow(5)[peta])
add.cluster.boundaries(som.wines,peta)
```

Fuzzy C-Means

We need to load packages and the data. We'll examine the data before we begin.

```
library(ppclust)
library(factoextra)
library(dplyr)
library(cluster)
library(fclust)
data(iris)
x=iris[,-5]
x[1:5,]
pairs(x, col=iris[,5])
cor(iris[,1:4])
library(psych)
pairs.panels(iris[,-5], method = "pearson")
```

We can create our model one time, or several times for comparison: because it's fuzzy, it might produce slightly different results each time.

```
res.fcm <- fcm(x, centers=3)
as.data.frame(res.fcm$u)[1:6,]
res.fcm$v0
res.fcm$v
summary(res.fcm)
res.fcm <- fcm(x, centers=3, nstart=5)
res.fcm$func.val
res.fcm$iter
res.fcm$best.start
summary(res.fcm)
plotcluster(res.fcm, cp=1, trans=TRUE)
```

We have several options for visualizing the results.

```
res.fcm2 <- ppclust2(res.fcm, "kmeans")
fviz_cluster(res.fcm2, data = x,
             ellipse.type = "convex",
             palette = "jco",
             repel = TRUE)

res.fcm3 <- ppclust2(res.fcm, "fanny")

cluster::clusplot(scale(x), res.fcm3$cluster,
                  main = "Cluster plot of Iris data set",
                  color=TRUE, labels = 2, lines = 2, cex=1)
```

We can analyze and validate the model with several tests.

```
res.fcm4 <- ppclust2(res.fcm, "fclust")
idxsf <- SIL.F(res.fcm4$Xca, res.fcm4$U, alpha=1)
paste("Fuzzy Silhouette Index: ",idxsf)
idxsf <- PE(res.fcm4$U)
paste("Partition Entropy: ",idxsf)
idxpc <- PC(res.fcm4$U)
paste("Partition Coefficient : ",idxpc)
idxmpc <- MPC(res.fcm4$U)
paste("Modified Partition Coefficient :",idxmpc)
```

We can also look at properties like the gap index, Davies-Bouldin’s index and the Calinski-Harabasz pseudo F-statistic.

```
library(clusterSim)
cl1<-pam(iris[,1:4],4)
cl2<-pam(iris[,1:4],5)
clall<-cbind(cl1$clustering,cl2$clustering)
g<-index.Gap(iris[,1:4], clall, reference.distribution="unif", B=10,method="pam")

print(g)

cl2 <- pam(iris[,1:4], 5)
print(index.DB(iris[,1:4], cl2$clustering, centrotypes="centroids"))

c<- pam(iris[,1:4],10)
index.G1(iris[,1:4],c$clustering)
```

Mean Shift Clustering

Start by loading the libraries. We'll create some sample data to do this example.

```
library(meanShiftR)
library(LPCM)
set.seed(100)
iter <- 1000
n <- 500
h <- c(0.5,0.5)
x1 <- matrix( rnorm( n ),ncol=2)
x2 <- matrix( rnorm( n ),ncol=2) + 2
x <- rbind( x1, x2 )
plot(x, col=rep(c('red','green'),each=n/2),
     cex=2, xlab='x',ylab='y',pch=20)
```

We run the mean shift algorithm with different parameter settings.

```
run.time <- proc.time()
result <- meanShift(
  x,
  x,
  algorithm="KDTREE",
  bandwidth=h,
  alpha=0,
  iterations = iter,
  parameters=c(10,100)
)
meanShiftR_kd_runtime <- (proc.time()-run.time)[3]
meanShiftR_kd_assignment <- result$assignment
meanShiftR_kd_value <- result$value
```

```
run.time <- proc.time()
result <- meanShift(
  x,
  x,
  bandwidth=h,
  alpha=0,
  iterations = iter
)
meanShiftR_runtime <- (proc.time()-run.time)[3]
meanShiftR_assignment <- result$assignment
meanShiftR_value <- result$value
```

```
runtime <- proc.time()
result <- ms(
  x,
  h=h,
  scaled=FALSE,
  iter=iter,
  plotms=-1)
LPCM_runtime <- (proc.time()-runtime)[3]
LPCM_assignment <- result$cluster.label
LPCM_value <- result$cluster.center[LPCM_assignment,]
```

```r
options(mc.cores = 4)
z <- t(x)
runtime <- proc.time()
result <- msClustering(
  X=z,
  h=h,
  kernel="gaussianKernel",
  tol.stop=1e-08,
  tol.epsilon=1e-04,
  multi.core=T)
MeanShift_runtime <- (proc.time()-runtime)[3]
```

```r
MeanShift_assignment <- result$labels
MeanShift_value <- t(result$components[,result$labels])

plot(x, col=sapply(meanShiftR_assignment,function(x)c('red','green','blue')[x]),
     cex=1.5, xlab='x',ylab='y',pch=20)
```

We can tally and compare the results.

```r
result <- data.frame(
  runtime=c( meanShiftR_runtime,
             meanShiftR_kd_runtime,
             LPCM_runtime,
             MeanShift_runtime),
  maxDiff=c(max(abs(meanShiftR_value - LPCM_value)),
            max(abs(meanShiftR_kd_value - LPCM_value)),
            0,
            max(abs(MeanShift_value - LPCM_value))
  ),
  assignmentDiff=c(sum(meanShiftR_assignment != LPCM_assignment),
                   sum(meanShiftR_kd_assignment != LPCM_assignment),
                   0,
                   sum(MeanShift_assignment != LPCM_assignment)
  )
)
```

```r
colnames(result) <- c('Run-Time',
                      'Maximum Absolute Difference',
                      'Label Disagreements')

rownames(result) <- c('meanShiftR',
                      'meanShiftR K-D Tree',
                      'LPCM ms',
                      'meanShift msClustering')
print(result)
```

In another example, we can try to look at how the centers converge.

```r
iter <- 10
n <- 500
m <- 20
h <- c(0.5,0.5)
x1 <- matrix( rnorm( n ),ncol=2)
x2 <- matrix( rnorm( n ),ncol=2) + 2
x <- rbind( x1, x2 )
y1 <- matrix( rnorm( m ) ,ncol=2)
y2 <- matrix( rnorm( m ),ncol=2) + 2
y <- rbind( y1, y2 )

plot(x, col=rep(c('salmon','greenyellow'),each=n/2),
     cex=1.5, xlab='x',ylab='y',pch=20)

points(y,col=rep(c('red','green'),each=m/2),
       cex=2,pch=19)

points(y,cex=2)


result <- meanShift(
  y,
  x,
  algorithm="KDTREE",
  bandwidth=h,
  alpha=0,
  iterations = iter,
  parameters=c(10,100)
)

y0 <- rbind(y,result$value)

for( i in 2:iter) {

  result <- meanShift(
    result$value,
    x,
    algorithm="KDTREE",
    bandwidth=h,
    alpha=0,
    iterations = 1,
    parameters=c(10,100)
  )
y0 <- rbind(y0,result$value)
}

for( i in 1:m ) {
  pointIndex <- seq(from=0,to=(m*(iter-1)),by=m)+i
  points(y0[pointIndex,] , type='l',lwd=2)
}
```

OPTICS

To test this package, we import our libraries and create some simulated data.

```
library(dbscan)
set.seed(2)
n <- 400
cluster_pts <- cbind(
  x = runif(4, 0, 1) + rnorm(n, sd=0.1),
  y = runif(4, 0, 1) + rnorm(n, sd=0.1)
)

plot(x ~ y, data=cluster_pts, pch = 20, cex=0.5, main="Random Blobs")
```

```
optics_res <- optics(cluster_pts, minPts = 10)
plot(optics_res)
plot(cluster_pts, col = "grey", main="Point Ordering")
polygon(cluster_pts[optics_res$order,])
```

We create a function to run our OPTICS algorithm and recreate our plots.

```
opticsDbscanPlot <- function(cluster_pts, optics_res, eps) {

  dbs_res <- extractDBSCAN(optics_res, eps_cl = eps)

  # Mark noise with big crosses
  shapes <- mapply(function(x) { if (x==0) return(3) else return(20) }, dbs_res$cluster)
  sizes  <- mapply(function(x) { if (x==0) return(1) else return(0.5)}, dbs_res$cluster)

  palette = c(1:length(unique(dbs_res$cluster))) # Align palette with reachability plot

  # Render noise points (cluster 0) in grey
  palette[1] = rgb(0.6, 0.6, 0.6)
  plot(dbs_res, main=paste("Reachability, Epsilon =", eps))  ## black is noise

  plot( x ~ y, data=cluster_pts,
        col=palette[dbs_res$cluster + 1],
        pch=shapes,
        cex=sizes,
        main=paste("Clusters, eps_cl = ", eps))
}
```

Finally, run the function.

```
opticsDbscanPlot(cluster_pts, optics_res, 0.065)
```

We can change the clustering result by changing the resolution.

```
opticsDbscanPlot(cluster_pts, optics_res, 0.090)
```

We can look at a second example with points in a different configuration.

```
pointsInCircum <- function(radius, num_pts) {

  x <- mapply(function(n) {return (cos(2*pi/num_pts*n) * radius + rnorm(1, -30,30)) }, 1:num_pts)
  y <- mapply(function(n) {return (sin(2*pi/num_pts*n) * radius + rnorm(1, -30,30)) }, 1:num_pts)

  df = data.frame(x=x, y=y)
  return(df)
}

set.seed(2)
df <- pointsInCircum(100, 300)
df <- rbind(df, pointsInCircum(300, 700))

df <- rbind(df, pointsInCircum(500, 1000))

# Add noise points
df <- rbind(df, data.frame(x=runif(300, -600, 600), y=runif(300, -600, 600)))

plot(x ~ y, data=df, pch = 20, cex=0.5, main="Concentric Random Clusters")
```

Run and plot the model (now that it's already set up, this is easy).

```
optics_res_circ <- optics(df)
opticsDbscanPlot(df, optics_res_circ, 32)
```

Resources:

1. https://data-flair.training/blogs/clustering-in-r-tutorial/
2. http://www.sthda.com/english/wiki/wiki.php?id_contents=7940
3. https://search.r-project.org/CRAN/refmans/stream/html/DSC_BIRCH.html
4. https://medium.com/@noel.cs21/balanced-iterative-reducing-and-clustering-using-heirachies-birch-5680adffaa58
5. https://www.shanelynn.ie/self-organising-maps-for-customer-segmentation-using-r/
6. https://mutmainnahdj.medium.com/self-organizing-maps-using-r-studio-f7de43e16819
7. https://rpubs.com/rahulSaha/Fuzzy-CMeansClustering
8. http://meanmean.me/meanshift/r/cran/2016/08/28/meanShiftR.html
9. https://www.kaggle.com/code/pmcgovern/optics-example-in-r