WIBS R Codes

#R #StatisticalModeling #cluster #offline #instrumentation

the following are codes used with the WIBS data

Hierarchical Clustering of Single Particle Data

Past studies using WIBS data have determined that a hierarchical agglomerative cluster model is best option out of the clustering models available. [1] These studies also tend to work with the single particle data (SPD) rather than the averaged data produced using the WIBS toolkit.

Hierarchical Clustering doesn't work well for large data sets [2]. The WIBS tends to produce very large datasets with only a handful of variables, especially in the case of SPD. Using a machine learning approach, a sample data set will be used to train the model and then be applied to the dataset as a whole.

The following is a modified version of the hierarchical clustering code originally developed to demonstrate a simple clustering example.

Packages and Data Import

Start by installing the necessary packages and import the data.

```
#install packages
library(tcltk)
library(lubridate)
library(dplyr)
library(tidyr)
library(psych)
library(ggplot2)
library(MASS)
#hierarchical clustering packages
install.packages("factoextra")
library(cluster)
library(fpc) #flexible procedures for clustering
library(clusterCrit) #cluster criteria

#pick the dataset to be clustered
df <- read.csv(file.choose(), sep=",", header = TRUE)
```

Data Prep

The data frame should only consist of the SPD variables to be modeled. This is done in a previous step, or the modeler can modify the data frame prep code to remove unwanted variables.

```
#make a working data frame of variables to be clustered and a ID variable
df_working <- df[,c("ID","FL1","FL2","FL3")]
```

All rows with missing data will be removed, as cluster models do not work with missing data.

```
#remove missing data
df_working <- na.omit(df_working)
# #can also use this but it will rewrite the col names
# df_working <- df_working[complete.cases(df_working),]
```

Remove outliers from the dataset. If the dataset was cleaned before importing, this step can be skipped.

There are two methods to removing outliers. The first involves searching for outliers in each variable individually. This method tends to remove too many observations from the original data set and significantly limits the the cluster model's ability to accurately represent the data as a whole. The second method involves removing multivariate outliers from an n-dimensional space.

Method 1: IQR Method
Start by creating a function that removes outliers from individual variables using the interquartile range (IQR) method. Here, **df** is the working data frame and **j** is the column number for the variable being cleaned. Then run the function for each variable in the dataset.

```
#Method 1: IQR outlire detection
#make a function that removes outlires from
outliresFUN <- function(df,j,name)
{
  # Compute Q1, Q3, and IQR
  Q1 <- quantile(df[,j], 0.25)
  Q3 <- quantile(df[,j], 0.75)
  IQR <- Q3 - Q1

  # Compute lower and upper bounds
  lower_bound <- Q1 - 1.5 * IQR
  upper_bound <- Q3 + 1.5 * IQR

  #create data frames for outlire flag
  x <- as.data.frame(matrix(ncol = 1,nrow = nrow(df)))
  #flag rows without outlires
  x <- ifelse(df_working$FL1 < lower_bound | df_working$FL1 > upper_bound, NA, 1)
  #combine data frames remove outlires
  x <- bind_cols(x,df)
  x <- na.omit(x)

  #remove the outlire flag and save to environment
```

```
  x <- x[,-c(1)]
  assign(name, x, envir = .GlobalEnv)
}

#run the outlire function for each variable in the data frame
outliresFUN(df_working,2,"df_working")
outliresFUN(df_working,3,"df_working")
outliresFUN(df_working,4,"df_working")
```

Method 2: Mahalanobis Distance (**Multivariate Outlier Detection**)
Mahalanobis distance considers correlations between variables and measures how far each observation is from the multivariate mean in an n-dimensional space. This is useful when dealing with multiple correlated variables.

- `mahalanobis()` computes the Mahalanobis distance of each observation.
- **Chi-square distribution** is used as a cutoff to detect outliers.
- **Multivariate outliers are removed** rather than checking variables one at a time.

For this part, *df* is the working data frame. The ID part of the function should be the column number with the ID variable (this should be the first variable in the data frame).

Run this function more than once to screen for outliers in layers.

```
#method 2: Mahalanobis Distance (Multivariate Outlier Detection)
multivar_outliers <- function(df, ID, confidence, threshold = 3)
{
  #remove ID variable
  x <- df[,-c(ID)]
  # Compute Mahalanobis distance
  cov_matrix <- cov(x, use = "complete.obs")  # Covariance matrix
  mean_vec <- colMeans(x, na.rm = TRUE)       # Mean of each column
  mahal_dist <- mahalanobis(x, mean_vec, cov_matrix)

  # Define outlier threshold (e.g., 3 standard deviations)
  cutoff <- qchisq(0.975, df = ncol(x))  # Chi-square threshold (for 0-1 confidence
(i.e., 0.975))

  # Remove outliers
  df_clean <- df[mahal_dist < cutoff, ]

  return(df_clean)
}

# Apply the function to your data
df_working <- multivar_outliers(df_working,4,0.975)
```

As stated previously, the data set is too large to be clustered. A random number of rows (**n**) will be sampled from the data set to create a training data set. Use as many sample rows as possible to ensure there is a good training data set for the model.

Two identical data frames will be created for the sampled data. One will not be modified and will only be used again at the end of the code in the final data set. The other will be a working data frame used to preform the actual modeling.

```
#make the sample set with n observations
df_working <- slice_sample(df_working, n = 1000, replace = FALSE)
df_sample <- df_working
  #remove the ID variable from the working data frame
  df_working <- df_working[,-c(1)]

#plot the variables
ggplot(df_working,aes(x=FL1,y=FL2)) +
  geom_point()

#scale each variable to have a mean of 0 and sd of 1
df_working <- scale(df_working)

#view first six rows of data set
head(df_working)

#look at dstats
describe(df_working)
```

Find the Linkage Method to Use

This step of the code takes the longest to run. Because of this, an even smaller sample set is created to determine which linkage model will best fit the dataset.

After the `sapply(m, ac)` line is done running, determine which method has the largest value. This will be the linkage method used for the cluster model.

```
#create a sub-sample set
df_lm <- slice_sample(df_sample[,-c(1)], n = 1000, replace = FALSE)
df_lm <- scale(df_lm)
head(df_lm)

#define linkage methods
m <- c("average", "single", "complete", "ward")
names(m) <- c( "average", "single", "complete", "ward")

#function to compute agglomerative coefficient
ac <- function(x)
```

```
{
  agnes(df_lm, method = x)$ac
}

#calculate agglomerative coefficient for each clustering linkage method
sapply(m, ac)
```

Run The Cluster Model

Using the working data frame, run a cluster model with the appropriate linkage method. The agnes() code computes **agglomerative hierarchical** (**HA**) **clustering** of the dataset.

```
#perform hierarchical clustering using best method option
clust <- agnes(df_working, metric = "euclidean", method = "ward")

#prodclust#produce dendrogram
pltree(clust, cex = 0.6, hang = -1, main = "Dendrogram")
```

Determine the Optimal Number of Clusters

The fist method involves relying on only the gap statistic to determine the optimal number of clusters. This method alone might not be enough for SPD modeling.

If this line doesn't run, try forcing the packages with `factoextra::`

```
###GAP STATISTIC###
#calculate gap statistic for each number of clusters (up to K clusters)
gap_stat <- clusGap(df_working, FUN = factoextra::hcut,  K.max = 15, B = 50)

#produce plot of clusters vs. gap statistic
factoextra::fviz_gap_stat(gap_stat)
```

The next method is from the PennState clustering code. It was originally developed for k-means clustering but here it was adapted for HA clustering from `agnes()`.

Unlike k-means, HA clustering requires you to manually calculate the SSW and SSB.
TSS (**Total Sum of Squares**): Measures total variance in the dataset.
SSW (**Sum of Squares Within**): Measures the variance within each cluster.
SSB (**Sum of Squares Between**): Measures the variance between cluster centroids and the overall mean.
The relationship is:

$$TSS = SSW + SSB$$

`agnes()` produces an agglomerative hierarchical clustering object, but you need `cutree()` to define the clusters.

SSW should be *lower*, and SSB should be *higher* for good clustering performance.

Next, the Root Mean Squared (RMS) needs to be calculated. Since RMS is the square root of the mean squared deviations, we can modify our previous code to include these calculations.

- **RMS within clusters** – Root Mean Squared distance of points within their clusters.
  - Lower `RMS_within` indicates that points are tightly clustered.
- **RMS between clusters** – Root Mean Squared distance between cluster centroids and the overall mean.
  - Higher `RMS_between` means clusters are well-separated.

RMS Within Clusters (`RMS_within`):
Measures the average squared distance of points within their assigned clusters.
Formula:

$$RMS_{within} = \sqrt{\frac{SSW}{N}}$$

where **N** is the total number of points.

RMS Between Clusters (`RMS_between`):
Measures the average squared distance between cluster centroids and the overall dataset mean.
Formula:

$$RMS_{between} = \sqrt{\frac{SSB}{K}}$$

where **K** is the number of clusters.

The R-squared of each model with k clusters is calculated.

```
###SUM OF SQUARES###
# Compute Total Sum of Squares (TSS)
centroid <- colMeans(df_working)
TSS <- sum(apply(df_working, 1, function(x) sum((x - centroid)^2))) # Sum of squared
distances to global mean

#making a empty data frame
criteria <- data.frame()

#setting range of k
nk <- 1:20

#loop for range of clusters
for (k in nk)
{
  c <- cutree(as.hclust(clust), k)

  # Compute Within Sum of Squares (SSW)
```

```
  SSW <- sum(sapply(unique(c), function(cluster)
  {
    cluster_points <- df_working[c == cluster, , drop = FALSE]
    cluster_center <- colMeans(cluster_points)
    sum(apply(cluster_points, 1, function(x) sum((x - cluster_center)^2)))
  }))

  # Compute Between Sum of Squares (SSB)
  SSB <- TSS - SSW

  #comute R-squared
  rsq <- 1-(SSW/TSS)

  # Compute RMS Within Clusters
  num_points <- nrow(df_working)
  RMS_within <- sqrt(SSW / num_points)

  # Compute RMS Between Clusters
  RMS_between <- sqrt(SSB / k)

  #add to the criteria data frame
  criteria <- rbind(criteria,c(k,SSW,SSB,TSS,rsq,RMS_within,RMS_between))
}
#renaming columns
names(criteria) <- c("k","SSW","SSB","TSS","R_Squared","RMS_within","RMS_between")

#looking at criteria
round(criteria,2)

# Define a scaling factor to align secondary axis values with primary axis
scaling_factor <- max(criteria$SSW, criteria$SSB) / max(criteria$R_Squared,
criteria$RMS_within)

#scree plot
ggplot(criteria, aes(x = k)) +
  # Red Line (SSW) on Right Axis
  geom_point(aes(y = SSW), color = "red") +
  geom_line(aes(y = SSW), color = "red") +

  # Blue Line (SSB) on Right Axis
  geom_point(aes(y = SSB), color = "blue") +
  geom_line(aes(y = SSB), color = "blue") +

  # Black Line (R_Squared) on Left Axis (scaled)
  geom_point(aes(y = R_Squared * scaling_factor), color = "black") +
  geom_line(aes(y = R_Squared * scaling_factor), color = "black") +
```

```
  # Purple Line (RMS_within) on Left Axis (scaled)
  geom_point(aes(y = RMS_within * scaling_factor), color = "green") +
  geom_line(aes(y = RMS_within * scaling_factor), color = "green") +

  # Axis Labels
  xlab("k = number of clusters") +
  ylab("Sum of Squares (within = red, between = blue)") +

  # Scale Y-Axis and Add Secondary Axis
  scale_y_continuous(
    name = "Sum of Squares (within = red, between = blue)",
    sec.axis = sec_axis(~ . / scaling_factor, name = "R-Squared (Black) RMS
(Green)") # Map back to original scale
  )
```

Apply Cluster Labels to Original Data Set

This is the last step of the training part of the cluster model.

```
#compute distance matrix
d <- dist(df_working, method = "euclidean")

#perform hierarchical clustering using Ward's method
final_clust <- hclust(d, method = "ward.D2" )

#cut the dendrogram into 4 clusters
groups <- cutree(final_clust, k=14)
groups <- as.vector(groups)

# Number of members in each cluster
table(groups)
```

Create a final dataset with the `df_sample` data frame that was created in the data prep section.

```
#append cluster labels to original data
final_data <- cbind(df_sample, cluster = groups)

#display first six rows of final data
head(final_data)

#find mean values for each cluster
aggregate(final_data, by=list(cluster=final_data$cluster), mean)
```

1. Cluster analysis of WIBS single-particle bioaerosol data.pdf ↩
2. Cluster_Analysis_Chapter 4-hierarchical.pdf ↩