Lab

Aim

To learn about k-means, k-medoids and hierarchical (agglomerative) clustering.

Before you start

Load the following packages:

- caret
- RColorBrewer
- scales
- cluster
- rgl
- pvclust
- fpc
- rattle

Download file delta.csv from Moodle. This file should be loaded with the option "rownames" set to "first column" i.e. `row.names=1`. For example,

```
delta <- read.csv(file="delta.csv", header=T, sep=",", row.names=1)
```

This ensures that the values in the first column (Aircraft) is only used as a way of naming each row, and not for applying analytical techniques to it.

Delta

The delta dataset contains data about a number of aircrafts used by Delta airlines. Load and inspect this dataset.

Pre-procesing

We need to center and scale the data. In addition, we are going to apply Principal Component Analysis (PCA)– a data transformation technique which reduces the dimensionality of the dataset while minimising information loss. Attributes are replaced by a smaller set of uncorrelated attributes – the Principal Components. **Note that PCA is not always necessary for clustering**, but it is needed for this dataset. If you want to know how PCA works, check out the StatQuest.

```
prepDelta <- preProcess(delta,
```

```
                    method = c("center", "scale", "pca"))
delta2 <- predict(prepDelta, newdata = delta)
```

Finding the best k – elbow method

A method which can be used to find the best k (number of clusters) is weighted sum of squares. The values are plotted and the value in the corner of the curve (the elbow) is the best k.

The following code calculates the within-group sum of squares. We are using k values between 2 and 15.

```
# Variable wss will hold the WCSS for each clustering result.
wss <- NULL
# Run kmeans and check within cluster sum of squares
# for selected k values. Store results in wss
for (i in 2:15)
wss[i] <- sum(kmeans(delta2, centers=i, nstart=100,
            iter.max=1000)$withinss)
```

Plotting the results

```
plot(1:15, wss, type="b", xlab="k= Number of Clusters", ylab="Within
groups sum of squares")
```

It can be seen that k=4 is at the elbow of the curve.

Finding the best k – silhouette method

A method which can be used to find the best k (number of clusters) is the silhouette method, which uses a measure of how close instances are to other instances in their cluster (cohesion) when compared to instances in other clusters (separation). The values are plotted and the best k is the one which gives the highest silhouette.

The following code calculates the silhouette values. We are using k values between 2 and 15. Comment on how big the silhouette values are (Silhouette analysis | R).

```
sil <-NULL
for (i in 2:15)
{
  res <- kmeans(delta2, centers = i, nstart = 25)
  ss <- silhouette(res$cluster, dist(delta2))
  sil[i] <- mean(ss[, 3])
}
plot(1:15, sil, type="b", xlab="k= Number of Clusters", ylab="Average
silhouette")
```

k-Means

We can now apply k-means to the dataset with k=4. We will use 25 random starts (nstart value) and a maximum of 1000 iterations, so that a good solution can be found.

```
km <- kmeans(delta2, 4, nstart=25, iter.max=1000)
```

To visualise the results with one colour per cluster.

```
#set the colour palette
palette(alpha(brewer.pal(9,'Set1'), 0.5))

plot(delta2, col=km$clust, pch=16)
```

To visualise in 3D – principal components 1, 2 and 3 (attributes resulting from the principal components analysis we used in preprocessing) with one colour per cluster

```
plot3d(delta2$PC1, delta2$PC2, delta2$PC3, col=km$clust)
```

Note that you can rotate the visualisation. Are the clusters clear?

To visualise principal components 1, 3 and 4

```
plot3d(delta2$PC1, delta2$PC3, delta2$PC4, col=km$clust)
```

You can try other visualisations.

Checking clusters in order of increasing size. First, clusters must be sorted by size

```
sort(table(km$clust))
```

The following results are obtained:

```
3  4  1  2
1  4 15 24
```

Meaning cluster 3 has 1 element, cluster 4 has 4 elements, cluster 1 has 15 elements and cluster 2 has 24 elements.

```
clust <- names(sort(table(km$clust)))
```

Getting the instances in each cluster

First cluster

```
row.names(delta[km$clust==clust[1],])
```

The first cluster contains only one airplane, the Airbus A319 VIP which is not one Delta normally uses in its fleet – it is used for private hire.

Second Cluster

```
row.names(delta[km$clust==clust[2],])
```

This cluster contains 4 airplanes – the smallest passenger planes all offering only economy seats.

Third Cluster

```
row.names(delta[km$clust==clust[3],])
```

Contains larger planes

Fourth Cluster

```
row.names(delta[km$clust==clust[4],])
```

Contains even larger planes, although there are some planes in common with the 3rd cluster.

As the clustering appears to have arranged the planes by size (seating accommodation), let's visualise the clusters using the Accommodation attribute. The Accommodation attribute tells us the number of seats available.

```
boxplot(delta$Accommodation ~ km$cluster,
        xlab='Cluster', ylab='Accommodation',
        main='Plane Accommodation by Cluster')
```

For comparison, select another attribute and produce a boxplot of that against cluster. Be careful, though, many of the attributes will correlate with "plane size". The cluster to watch for is the "VIP" plane cluster, because it's a plane with very few seats but is a reasonable size in terms of wingspan. Even things like the seat pitch in first class will give an indication of the plane size. Some of the attributes will provide better separation of the clusters than others. If two clusters are easily separable, their box plots will also look separable (similar to the confidence intervals plotted previously).

The largest 2 clusters appear to be divided according to seat classes. Let's see this. These are attributes 30, 31, 32 and 33 in the original dataset.

```
delta[km$clust==clust[3],30:33]
```

```
delta[km$clust==clust[4],30:33]
```

With one exception, one cluster has instances with first class accommodation whereas the other one has instances with business class accommodation instead.

Exercise

1 In R you can use different types of k-means algorithms: "Hartigan-Wong", "Lloyd", "Forgy","MacQueen". The default is "Hartigan-Wong". The k-means algorithm is set in the kmeans options using "algorithm=", for example

```
km2 <- kmeans(delta2, 4,  algorithm="Lloyd", nstart=25, iter.max=1000)
```

Repeat the experiments with each of the algorithms and compare the results.

Adding the cluster number to the dataset

The cluster number can be added to the dataset in order to:

- Enhance classification.
- Use a classification algorithm to understand what each cluster contains.

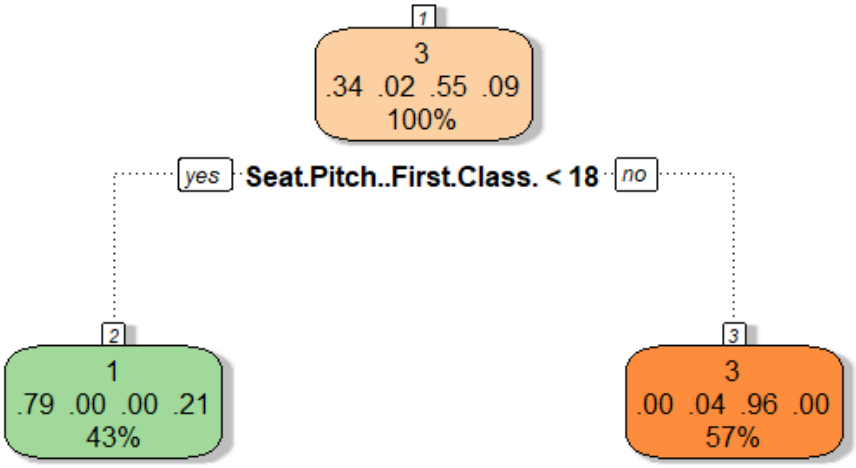
For example the cluster number can be added to the original dataset (the new dataset is called delta3 below) and to the processed dataset (called delta4 below).

```
delta3 <- delta
delta3$cluster <-as.factor(as.character(km$clust))

delta4 <- delta2
delta4$cluster <-as.factor(as.character(km$clust))
```

A classification algorithm (rpart in this example) can be used to predict cluster. The use of the delta3 dataset (as defined above) will make the predictions more understandable as it has not been processed using PCA.

```
ctrl <- trainControl(method="repeatedcv", number=10, repeats=3)

set.seed(123)
mod.rpart1 <- train(cluster~., data=delta3, method="rpart",
trControl=ctrl)
print(mod.rpart1)
plot(mod.rpart1)
fancyRpartPlot(mod.rpart1$finalModel)
```

This will give the following decision tree

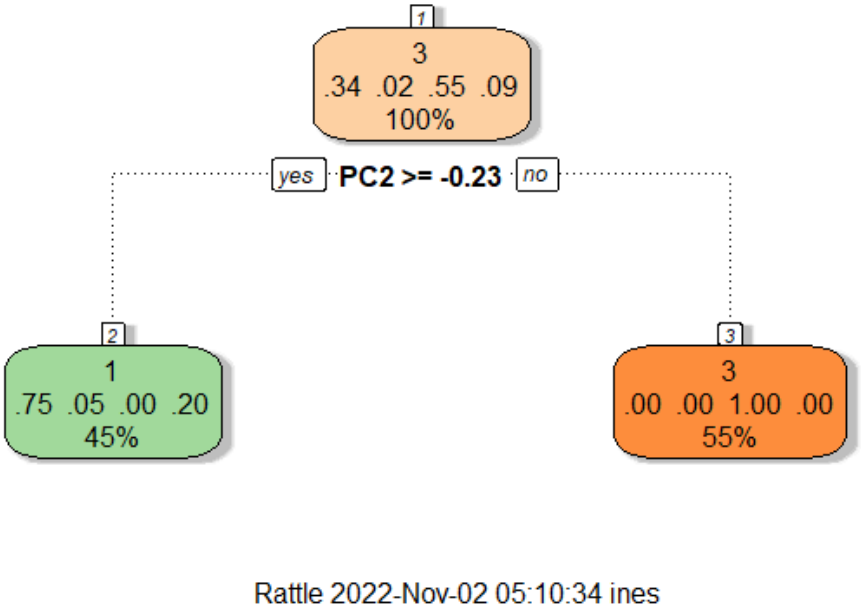


It can be seen that the seat pitch is the deciding factor to determine the cluster. Only 2 clusters are classified: clusters 1 and 3. The former contains the instances where the seat pitch for first class is less than 18 whereas the latter contains the rest of the instances. Both cluster identifications contain errors. 43% of all instances are classed as being cluster1 but 21% are incorrect (they're all actually cluster 4). Similarly, 57% of all classifications are for cluster3, but 4% of those classifications are incorrect (they're cluster 2).

The same could be applied to the processed dataset.

```
set.seed(123)
mod.rpart2 <- train(cluster~., data=delta4, method="rpart",
trControl=ctrl)
print(mod.rpart2)
plot(mod.rpart2)
fancyRpartPlot(mod.rpart2$finalModel)
```

The tree looks as follows:



Note that "PC2" is the result of PCA so it is not meaningful or interpretable. But the results show that the decision tree can successfully separate cluster 3 from all the other clusters (i.e. if this decision tree says something is cluster 3, then it definitely *is* cluster 3).

Exercise

2 Apply C5.0Tree to check if it gives better results than rpart for cluster identification.

k-medoids (Partitioning Around Medoids - pam)

The k-medoids algorithm is very similar to the k-means algorithm, except that it uses medoids (actual instances) instead of centroids. Each medoid is the representative instance of a cluster. The medoid is the instance of a cluster whose average dissimilarity to the other instances in the cluster is minimal.

The following code applies k-medoids to the Delta dataset and plots the results.

```
set.seed(1)
kmedoids <- pam(delta2, 4)
#palette(alpha(brewer.pal(9,'Set1'), 0.5))
plot(delta2, col=kmedoids$clust, pch=16)
```

Exercises

3 Compare the clustering you obtained with k-means with the clustering you got with k-medoids using the same k .
4 Use the silhouette method to establish the best k for k-medoids. Apply the algorithm and examine the clusters obtained.

Hierarchical clustering

We are going to apply hierarchical clustering to the same dataset and check whether we get the same results.

First, we need to define the distance function – Euclidean.

```
d <- dist(delta2, method = "euclidean")
```

Then we apply hierarchical clustering

```
hc <- hclust(d, method="single")
```

The method can take a number of values including "single", "complete", "average", "centroid".

We can now visualise the results - ***** please ensure that the next 3 instructions appear in the same R block. *****

```
plot(hc)
```

We cut the tree into 4 clusters

```
groups <- cutree(hc, k=4)
```

We can now visualise the 4 clusters (the look different, but are they the same 4 clusters that we observed earlier with k-means?).

```
rect.hclust(hc, k=4, border="red")
```

Exercises

5 Change the distance to "manhattan" and repeat the hierarchical clustering above but put the results in "hc2".
Are the results similar? You can visually compare the results you got using k-means and hierarchical clustering with Euclidean distance to those you got using Manhattan distance.
6 Apply other schemas. You can change the method in the hclust function to `"complete", "centroid", average".` Do they all produce the same results?