Machine Learning and PCA

Andy Hsu

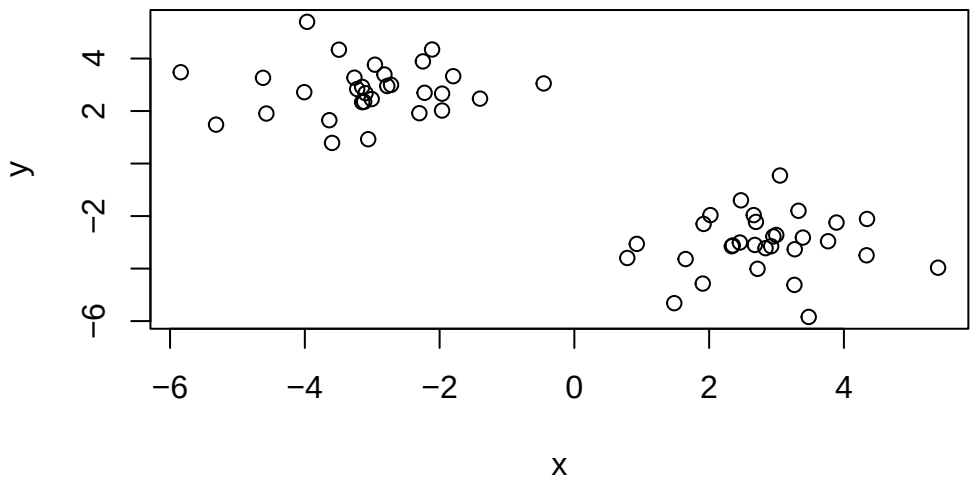
Clustering

K-means Clustering

We will start today's lab with clustering methods, K-means in particular. The main function for this in R is `kmeans()`.

First, let's fabricate a data set with a known distribution.

```
tmp <- c(rnorm(30,mean=3),rnorm(30,mean=-3))
x <- cbind(x=tmp,y=rev(tmp))
plot(x)
```



Based on the `plot()` returned by R, we should expect any clustering function to easily sort this set into 2 clusters.

```
k <- kmeans(x,centers=2,nstart=20)
k
```

```
K-means clustering with 2 clusters of sizes 30, 30

Cluster means:
          x         y
1 -3.062042  2.809235
2  2.809235 -3.062042

Clustering vector:
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
[39] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Within cluster sum of squares by cluster:
[1] 65.59161 65.59161
 (between_SS / total_SS =  88.7 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

Within the `kmeans()` function, the `centers=` argument tells the algorithm how many groups there should be in the group, and the `nstart=` argument tells the algorithm how many iterations to run. The function then returns the best result from all iterations, along with a dataset with information on the clusters.

```
# Size of each cluster
k$size
```

```
[1] 30 30
```

```
# Membership of each point
k$cluster
```

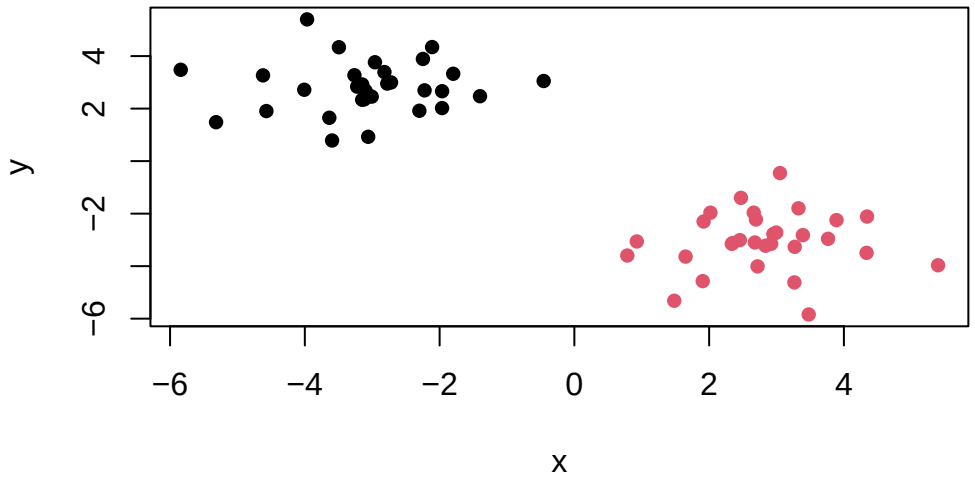
```
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
[39] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
# Center of each cluster
k$centers
```

```
          x         y
1 -3.062042  2.809235
2  2.809235 -3.062042
```

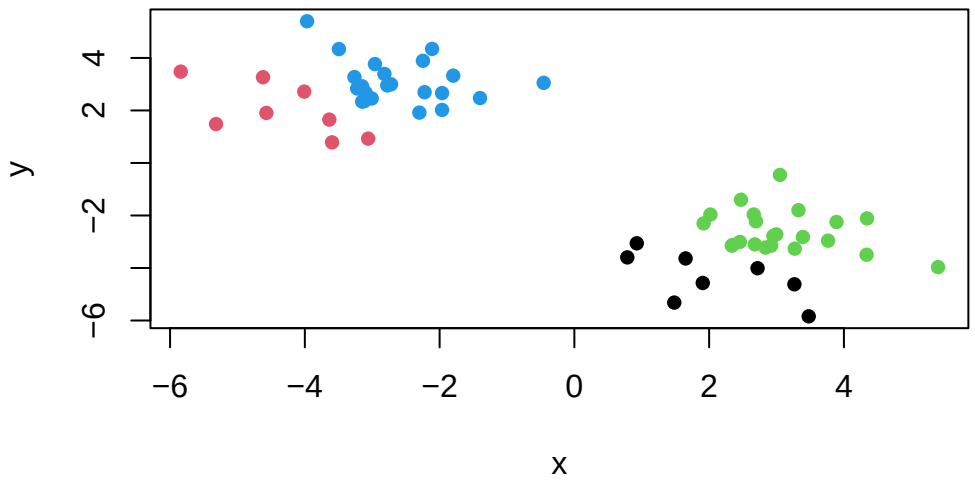
To visualize the results, I can plot using the following code, displaying where the clusters are and which points are in which cluster.

```
plot(x,col=k$cluster,pch=16)
```



But what happens if we try to separate this dataset into 4 groups?

```
j <- kmeans(x,centers=4,nstart=20)
plot(x,col=j$cluster,pch=16)
```

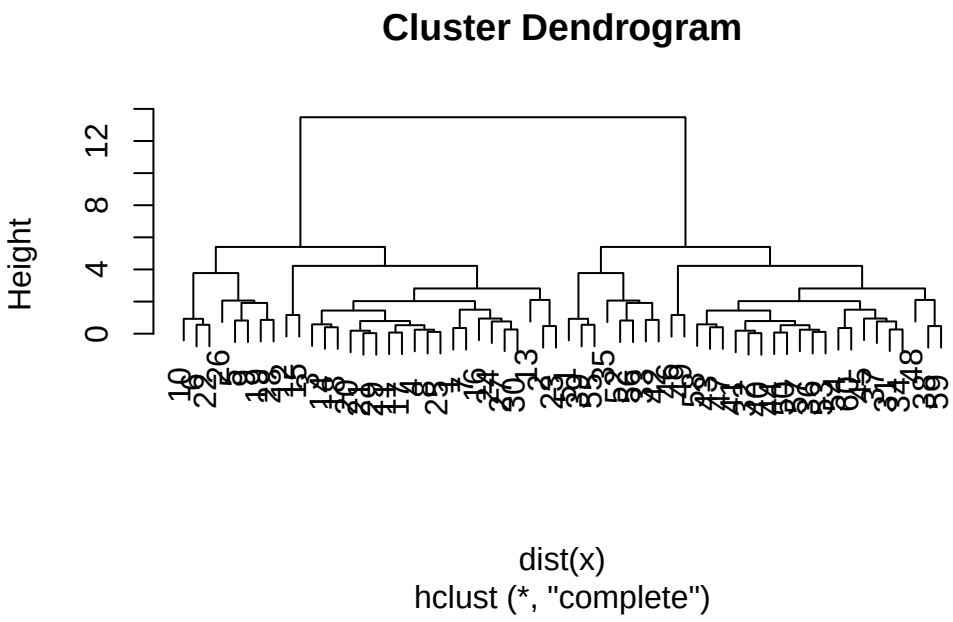


Hierarchical Clustering

As quick and easy as kmeans is, a huge drawback of the function is that `centers=` needs to be defined, which can lead to inaccurate categorization of your data and confirmation bias. H-clusting can circumvent this by not requiring a defined number of clusters and instead discerning the value itself.

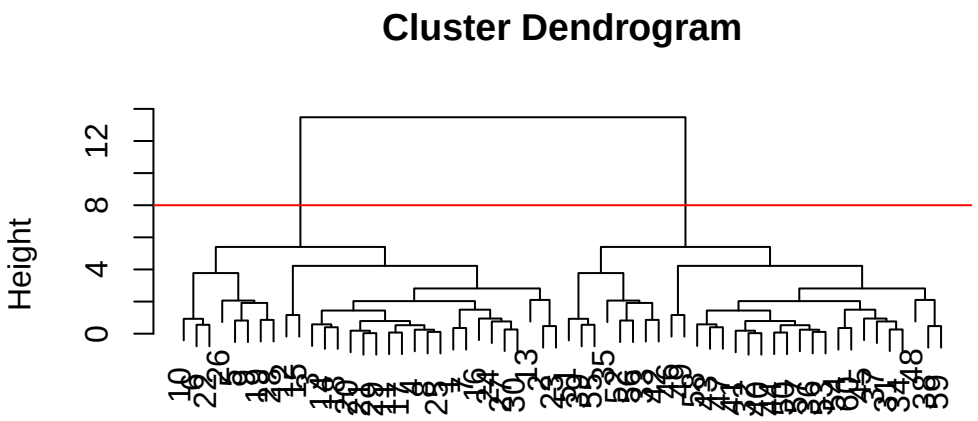
The `hclust()` function performs hierarchical clustering on your given dataset, and requires some more setup in comparison to `kmeans()`. First, it needs an input of a distance matrix, which can be done with the `dist()` function.

```
hc <- hclust(dist(x))
plot(hc)
```



To find the clusters from this result, we can cut the tree at a certain height, splitting the data below into groups.

```
plot(hc)
abline(h=8,col="red")
```

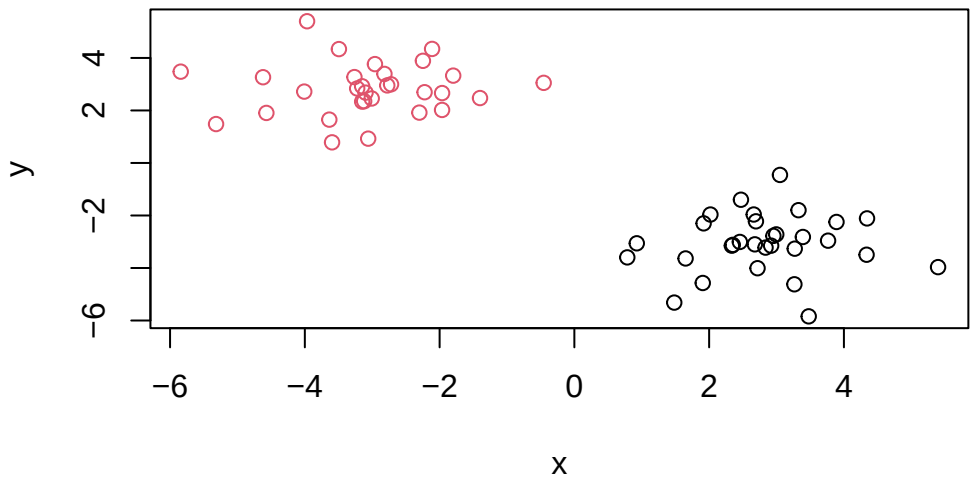



```
grps <- cutree(hc,h=8)
grps
```

```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

Plotting again by clusters, we can use this code.

```
plot(x,col=grps)
```



Principal Component Analysis (PCA)

PCA of UK Food Data

PCA is a technique we can use to make sense of datasets with many dimensions. It works by creating primary components, aiming to minimize variance on most axes and maximizing variance on 1.

First, let's try conventional data analysis methods on this 17-dimension set.

```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url)
```

First, we can find the dimensions of the dataset and look at the first couple lines of data.

```
dim(x)
```

```
[1] 17  5
```

```
head(x)
```

```
               X England Wales Scotland N.Ireland
1         Cheese     105   103      103        66
2  Carcass_meat      245   227      242       267
3    Other_meat      685   803      750       586
4           Fish     147   160      122        93
5 Fats_and_oils      193   235      184       209
6         Sugars     156   175      147       139
```

Next, we can fix the row names and make them proper rownames.

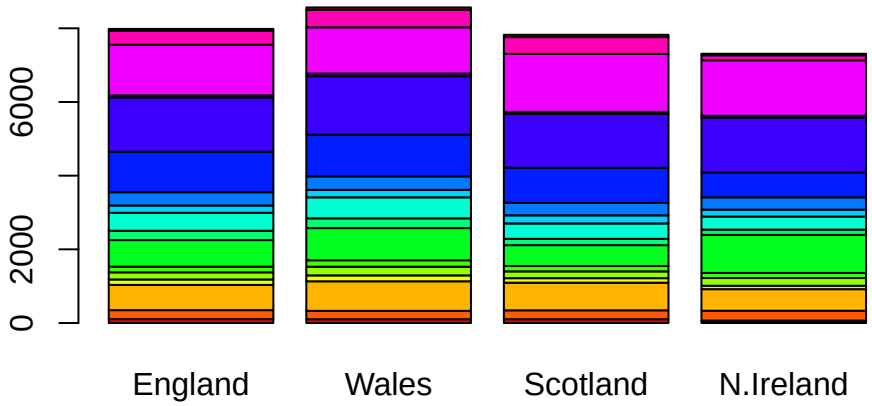
```
x <- read.csv(url, row.names=1)
head(x)
```

```
               England Wales Scotland N.Ireland
Cheese             105   103      103        66
Carcass_meat       245   227      242       267
Other_meat         685   803      750       586
Fish               147   160      122        93
Fats_and_oils      193   235      184       209
Sugars             156   175      147       139
```

This method of using `read.csv()` is more robust, as it avoids trimming another column accidentally if the code is run again.

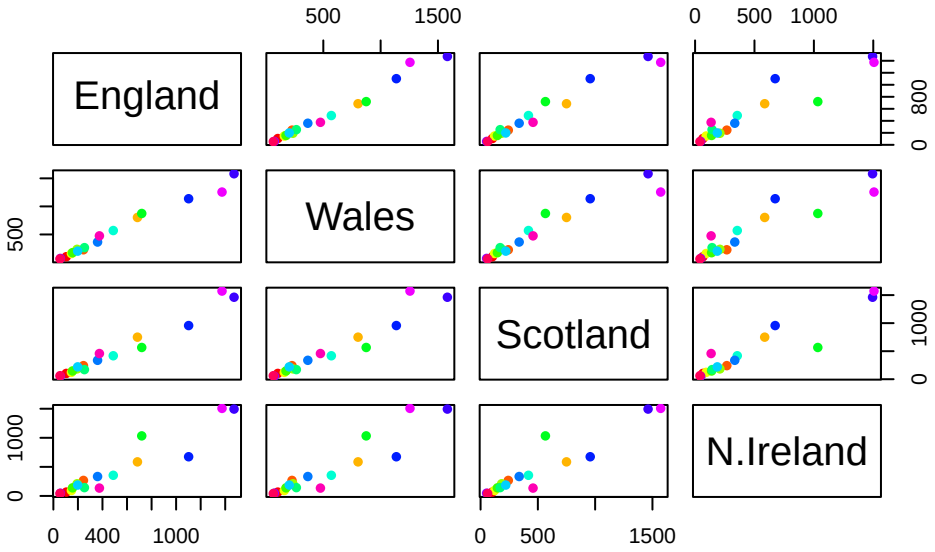
Plotting the data as a bar plot, we see a jumble of bars that is hard to interpret.

```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



Similarly, a pairwise plot with all country comparisons is not very useful, but it can help in displaying how similar countries are, judging by how close a set is to a diagonal line.

```
pairs(x, col=rainbow(17), pch=16)
```



From this data, we can tell that N. Ireland is quite different from the rest of the countries.

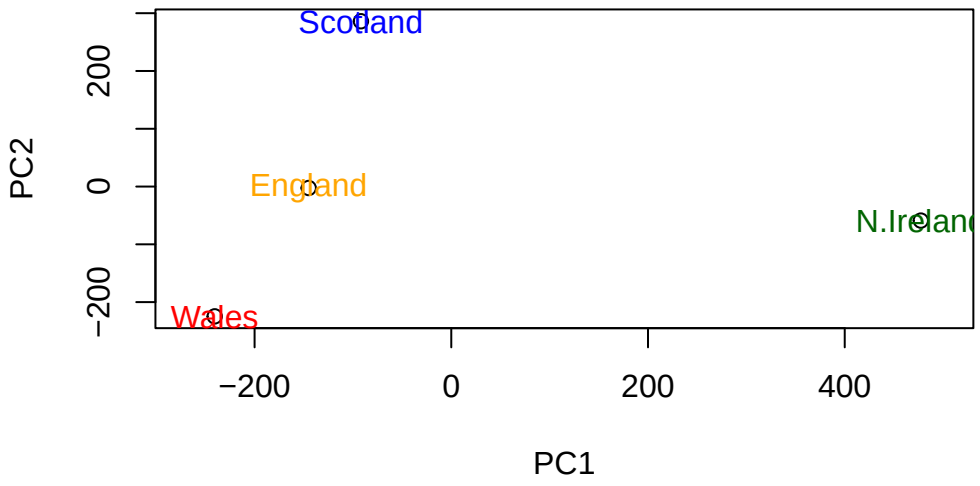
We can do better, though. Let's use PCA to interpret our data. The main function to use for PCA is `prcomp()`.

```
# Note that we transpose the data first to get the right variables on x and y.
pca <- prcomp(t(x))
summary(pca)
```

```
Importance of components:
                            PC1      PC2      PC3       PC4
Standard deviation     324.1502 212.7478 73.87622 3.176e-14
Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

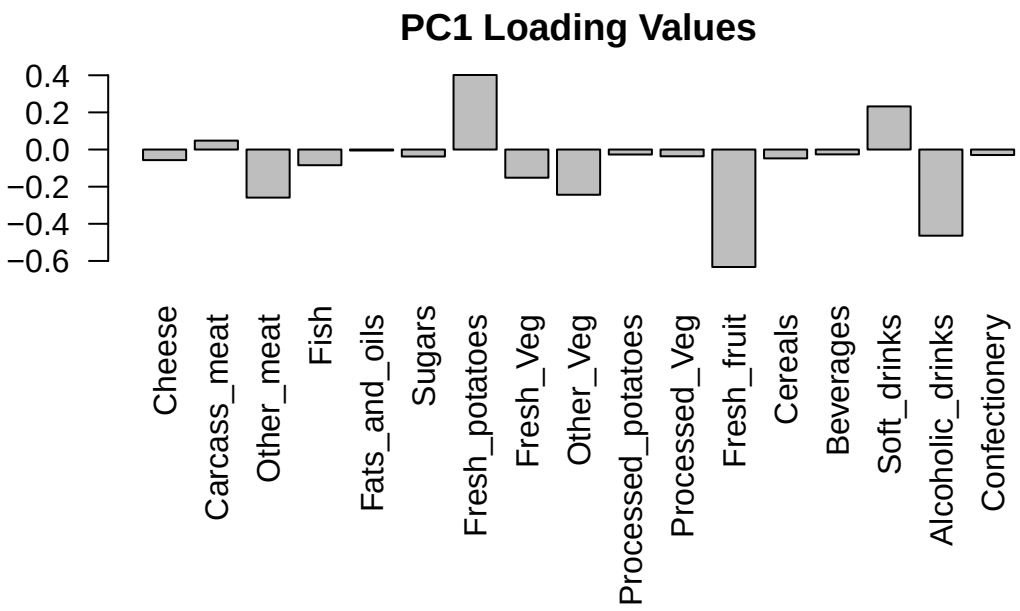
To plot PC1 vs PC2 and visualize our results, we can write this code.

```
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2", xlim=c(-270,500))
text(pca$x[,1], pca$x[,2], colnames(x), col=c("orange","red","blue","darkgreen"))
```

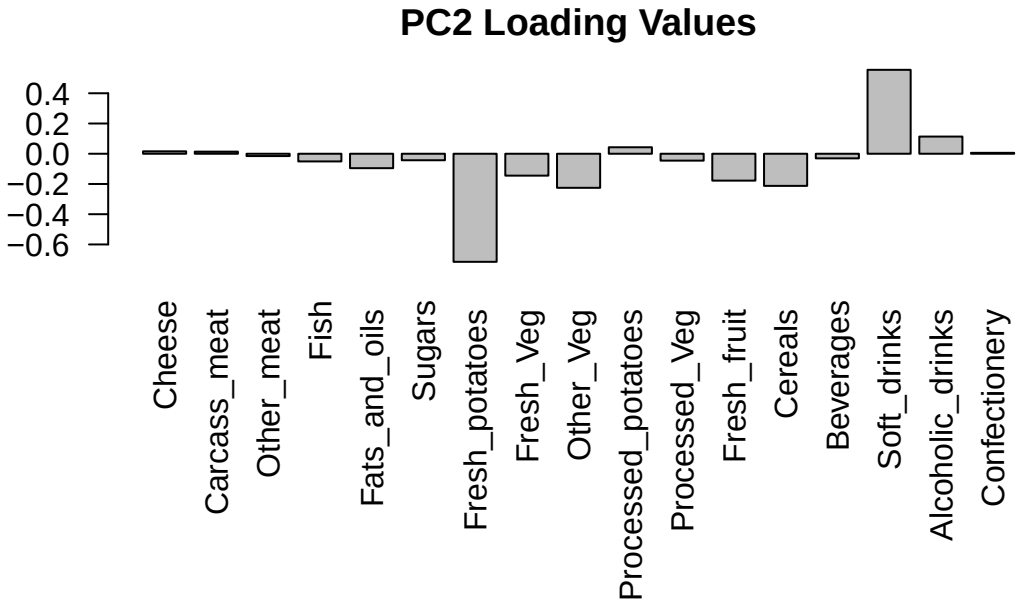


The “loadings” tell us how much the original variables (i.e. food values) contribute to our new variables (i.e. PCs). We can plot these values to a biplot to show the influence of each value.

```
par(mar=c(10, 3, 2.5, 0))
barplot(pca$rotation[,1], las=2, main="PC1 Loading Values")
```




```r
par(mar=c(10, 3, 2.5, 0))
barplot(pca$rotation[,2], las=2, main="PC2 Loading Values")
```



And those were some of the basics of using PCA to demystify datasets with a high number of dimensions.