Solutions 14

Jumping Rivers

Clustering

Here we will use cluster analysis to try to discover groups of states in America that are similar in crime profile. The data can be loaded with

```
import jrpyml
usarrests = jrpyml.datasets.load_usarrests()

usarrests, states = usarrests.drop(columns='State'), usarrests['State'].values
```

a) Preprocess your data to get a new array of rescaled variables ready for clustering

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
usarrests_scaled = scaler.fit_transform(usarrests)
```

b) Draw a dendrogram of the states. Experiment with different linkages to see how this affects the number of clusters you might choose in an agglomerative heirarchical clustering routine.

```
from scipy.cluster.hierarchy import dendrogram, linkage
from matplotlib import pyplot as plt

linked = linkage(usarrests_scaled, 'complete')
labelList = states
dendrogram(linked,
  orientation='top',
  labels=labelList,
  distance_sort='descending',
  show_leaf_counts=True
)
plt.show()
```

c) Perform agglomerative clustering to find 4 clusters using complete linkage

```
from sklearn.cluster import AgglomerativeClustering
cluster = AgglomerativeClustering(n_clusters=4, linkage ='complete')
cluster.fit_predict(usarrests_scaled)
```

```
## array([0, 0, 2, 1, 2, 2, 1, 1, 2, 0, 1, 3, 2, 1, 3, 1, 1, 0, 3, 2, 1, 2,
##        1, 0, 1, 3, 3, 2, 3, 1, 2, 2, 0, 3, 1, 1, 1, 1, 1, 0, 3, 0, 2, 1,
##        3, 1, 1, 3, 1, 1])
```

d) Draw a scatter plot of the states, using two of the measured variables, highlighting the clusters they belong to.

```
plt.scatter(
  usarrests['Murder'], usarrests['Assault'],
  c = cluster.labels_
)
plt.show()
```

e) Since we have multiple dimensions to our data, it can be difficult to draw. One thing we might do is use principal components as a data projection into 2d. This allows us to capture some of the variation of each of the 4 measured variables on the 2d plot. Use a PCA to transform the data into 2 principal components

```
from sklearn.decomposition import PCA

pca = PCA()

pca.fit(usarrests_scaled)

usarrests_pc = pca.transform(usarrests_scaled)[:,:2]

plt.scatter(
  usarrests_pc[:,0], usarrests_pc[:,1],
  c = cluster.labels_
)
plt.show()
```

e) One thing that can help with interpretation of the principal components plot is called a biplot. Here we plot the principal components as the main x and y axes, but overlay the direction of the original variable axes. We could write a function to draw the cluster labels in principal component space with the overlaid original axes.

```
import numpy as np

def myplot(score, coeff, colors, labels=None):
  xs = score[:,0]
  ys = score[:,1]
  n = coeff.shape[0]
  scalex = 1.0/(xs.max() - xs.min())
  scaley = 1.0/(ys.max() - ys.min())
  plt.scatter(xs * scalex,ys * scaley, c = colors)
  for i in range(n):
    plt.arrow(0, 0, coeff[i,0], coeff[i,1],color = 'r',alpha = 0.5)
```

```
    if labels is None:
      plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, "Var"+str(i+1), color = 'g', ha = 'center', va = 'c
    else:
      plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, labels[i], color = 'g', ha = 'center', va = 'center
  plt.xlim(-1,1)
  plt.ylim(-1,1)
  plt.xlabel("PC{}".format(1))
  plt.ylabel("PC{}".format(2))
  plt.grid()

#Call the function. Use only the 2 PCs.
myplot(
  usarrests_pc[:,0:2],np.transpose(pca.components_[:2, :]),
   cluster.labels_, usarrests.columns
)
plt.show()
```