인대영 (2022311154)
2022-06-02

HW12 Report

1. (a) After saving Fisher's Iris data, we normalize the data using 'MinMaxScaler' function. Then after dropping the 'Species' column, we save x1~x4 characteristics as 'x' and the species as 'y'.

In [3]:
```python
# Normalization
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
iris2 = iris.copy()
iris2.iloc[:,:4] = scaler.fit_transform(iris2.iloc[:,:4])
iris2
```

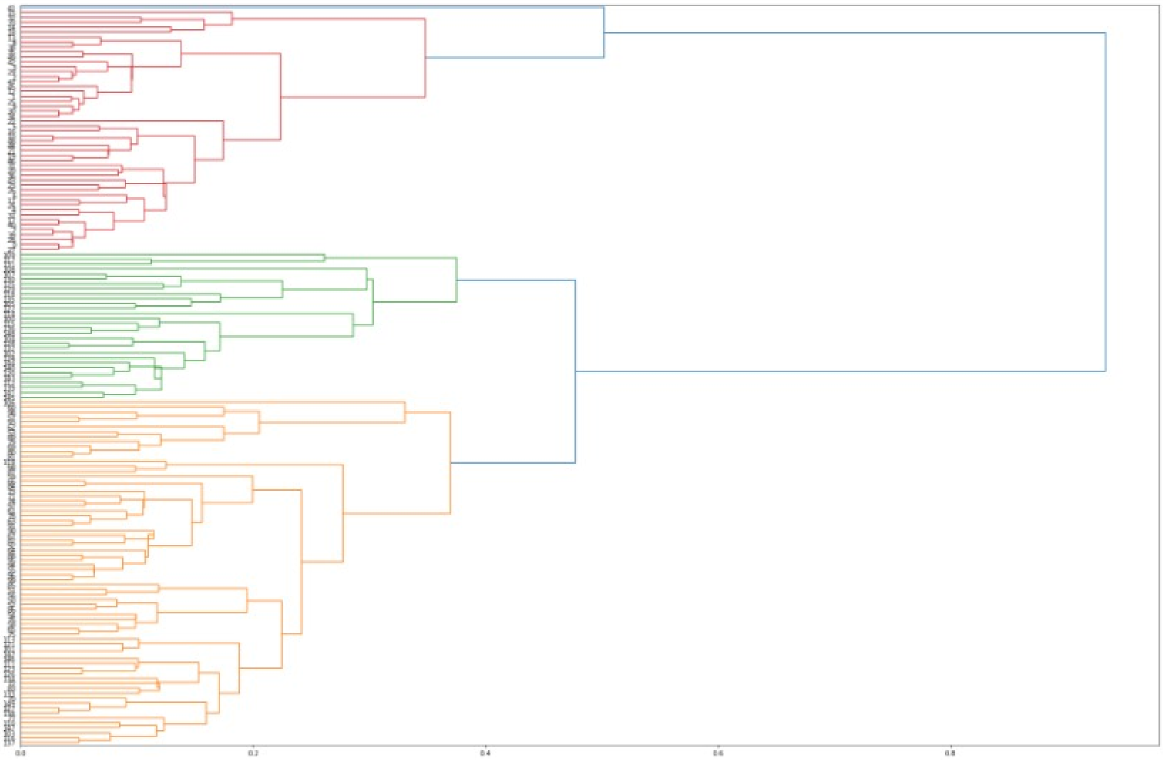
Out[3]:

	Sepal length(x1)	Sepal width(x2)	Petal length(x3)	Petal width(x4)	Species
0	0.222222	0.625000	0.067797	0.041667	1
1	0.166667	0.416667	0.067797	0.041667	1
2	0.111111	0.500000	0.050847	0.041667	1
3	0.083333	0.458333	0.084746	0.041667	1
4	0.194444	0.666667	0.067797	0.041667	1
...	...	...	...	...	...
145	0.666667	0.416667	0.711864	0.916667	3
146	0.555556	0.208333	0.677966	0.750000	3
147	0.611111	0.416667	0.711864	0.791667	3
148	0.527778	0.583333	0.745763	0.916667	3
149	0.444444	0.416667	0.694915	0.708333	3

150 rows × 5 columns

We then perform hierarchical cluster analysis using centroid method via function 'linkage' and 'dendrogram' from the 'scipy.cluster.hierarchy' module. By the following code, we are able to plot the dendrogram of our data which classifies our data into three clusters shown in red, green, and orange. We do have one observation (index = 41) which is classified as its own, but we consider this to be part of the red cluster which is the closest. We used threshold = 0.4 because it yields an appropriate number of clusters.

In [6]:
```python
linked = linkage(x, method = 'centroid')
labelList = list(range(150))
plt.figure(figsize = (30, 20))
dendrogram(linked,
           orientation = 'right',
           count_sort = 'descending',
           distance_sort = 'descending',
           labels = labelList,
           leaf_font_size = 10,
          get_leaves = True,
          color_threshold = 0.4)
plt.show()
```



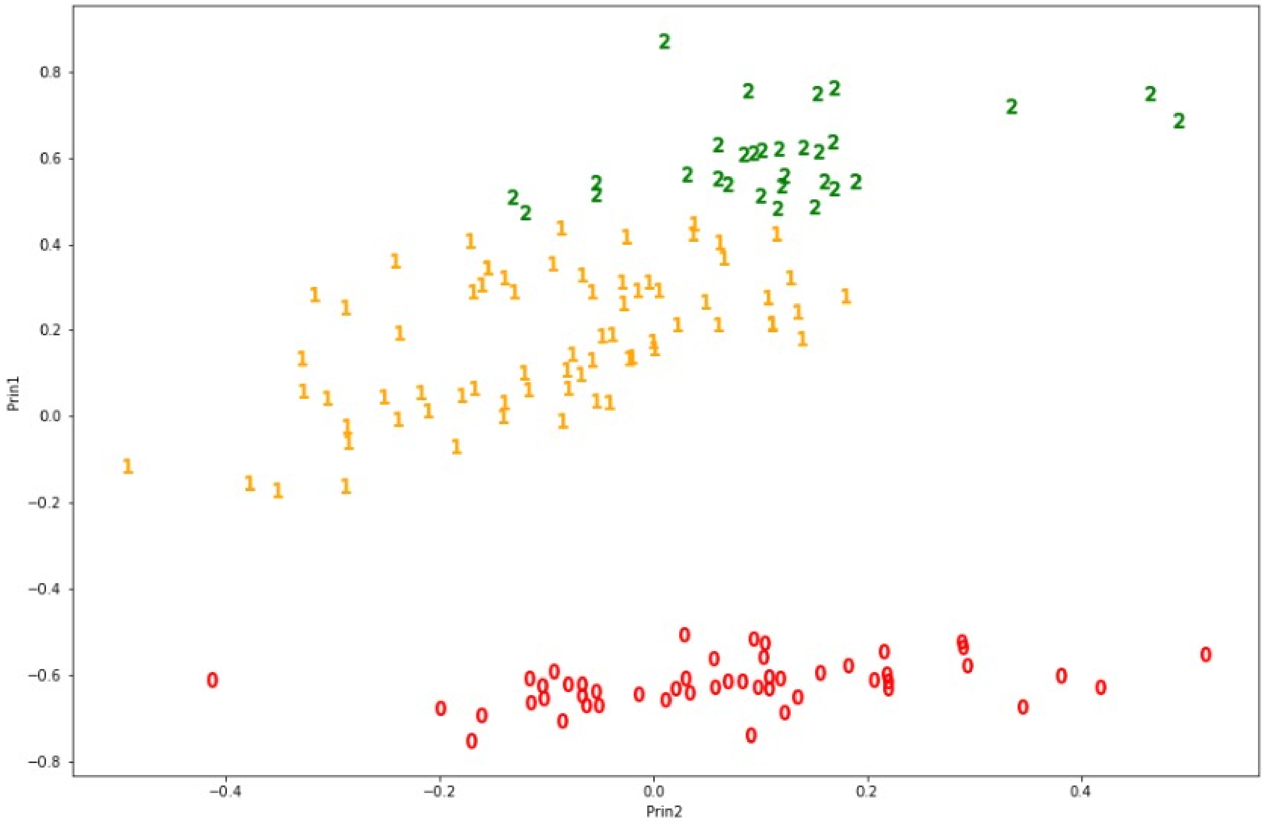
(b) By setting the parameter 'get_leaves = True', we are able to obtain the list of indices of each cluster. We label the red observation in the dendrogram as cluster 1 (index = 0), the orange observations as cluster 2 (index = 1), and the green observation in the dendrogram as cluster 3 (index = 2)

In [11]:
```
#Label clusters
hier_clusters = pd.Series(range(150))
hier_clusters = hier_clusters.replace(to_replace = hier_clust1, value = 0)
hier_clusters = hier_clusters.replace(to_replace = hier_clust2, value = 1)
hier_clusters = hier_clusters.replace(to_replace = hier_clust3, value = 2)
hier_clusters
```

Out[11]:
```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    1
147    1
148    2
149    1
Length: 150, dtype: int64
```

Using 'PCA' function from 'sklearn.decomposition' module, we plot the first two principal component scores with the cluster label color-coded consistently with the dendrogram. The x-axis is PC2 and the y-axis is PC1.

In [13]:
```
plt.figure(figsize = (15,10))
plt.xlabel('Prin2') ; plt.ylabel('Prin1')
color = {0:'red', 1:'orange', 2:'green'}
for i in range(len(x)):
    plt.scatter(pc[i,1], pc[i,0],
                marker = "$ {} $".format(hier_clusters[i]),
                s = 100,
                c = color[hier_clusters[i]])
```



(c) We use the confusion matrix to compare the clusters with the actual class. We changed the labels of the actual class from 1, 2, 3 to 0, 1, 2, respectively in order to compare with the clusters. The corresponding confusion matrix is obtained using the code below:

```
In [14]: #(c) Compare the clusters with the actual class using confusion matrix.
         #Confusion matrix
         from sklearn.metrics import confusion_matrix
         y_hier = hier_clusters

         C1 = pd.DataFrame(confusion_matrix(y_hier, y),
                           index = np.sort(y_hier.unique()),
                           columns = np.sort(y.unique()))
         C1['Total'] = C1.sum(axis = 1) #row sum
         C1.loc['Total',:] = C1.sum(axis = 0) # column sum
         C1.index.names = ['From class'] ; C1.columns.names = ['Classified class']
         C1.astype(int)
```

Out[14]:

Classified class / From class	0	1	2	Total
0	50	0	0	50
1	0	50	20	70
2	0	0	30	30
Total	50	50	50	150

(d) We perform the K-mean cluster analysis using 'Kmeans' function from 'sklearn.cluster' module. We set the number of clusters as n_clusters = 3. We also need to set a random_state so that the result is consistent. The classification result of K-means clustering is shown below.

```
In [15]: #(d) Perform the K-means cluster analysis with the number of clusters you acquired in (a).
         from sklearn.cluster import KMeans
         kmeans = KMeans(n_clusters = 3, random_state = 0).fit(x)
         group = pd.DataFrame(kmeans.labels_, columns=['cluster'])
         Xkmean = x.join(group)
         Xkmean
```

Out[15]:

	Sepal length(x1)	Sepal width(x2)	Petal length(x3)	Petal width(x4)	cluster
0	0.222222	0.625000	0.067797	0.041667	0
1	0.166667	0.416667	0.067797	0.041667	0
2	0.111111	0.500000	0.050847	0.041667	0
3	0.083333	0.458333	0.084746	0.041667	0
4	0.194444	0.666667	0.067797	0.041667	0
...	...	...	...	...	...
145	0.666667	0.416667	0.711864	0.916667	2
146	0.555556	0.208333	0.677966	0.750000	1
147	0.611111	0.416667	0.711864	0.791667	2
148	0.527778	0.583333	0.745763	0.916667	2
149	0.444444	0.416667	0.694915	0.708333	1

150 rows × 5 columns

We also obtain the cluster frequency and means for each variable.

```
In [16]: # Cluster Frequency
         Xkmean.groupby('cluster').count()
```

Out[16]:

cluster	Sepal length(x1)	Sepal width(x2)	Petal length(x3)	Petal width(x4)
0	50	50	50	50
1	61	61	61	61
2	39	39	39	39

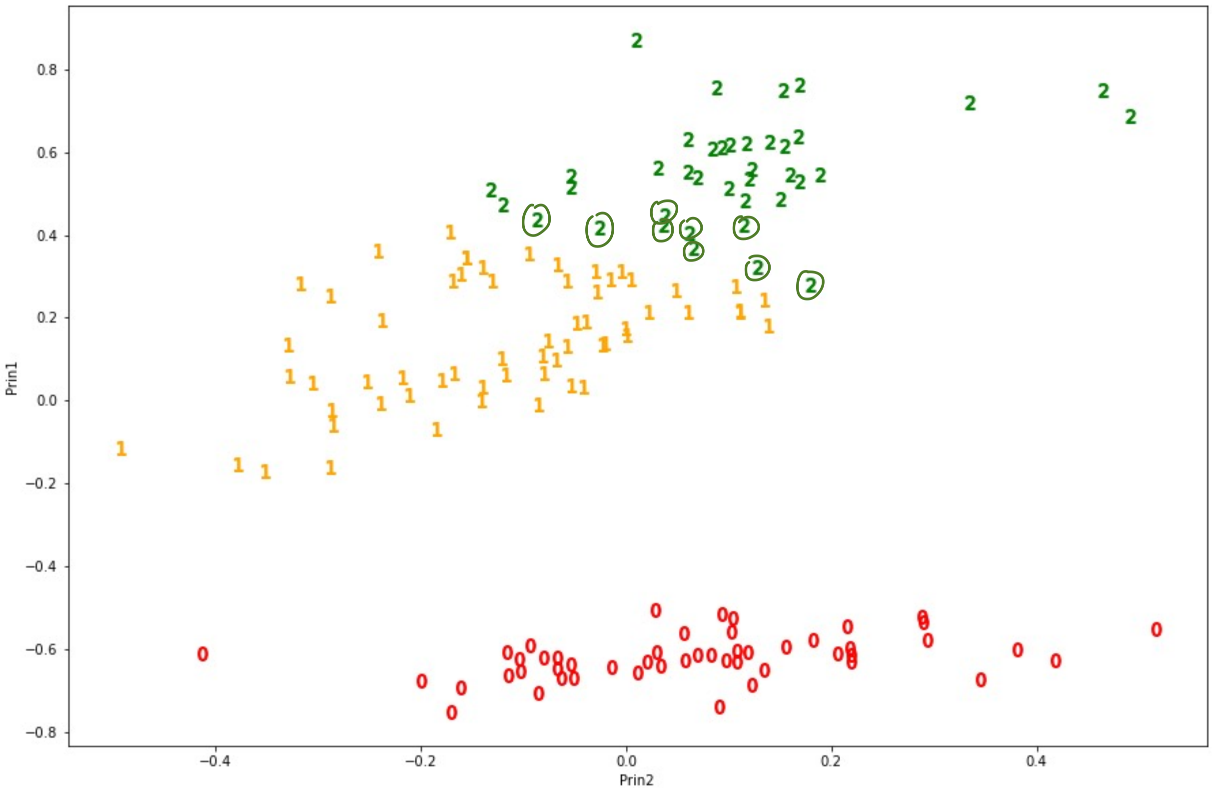
```
In [17]: # Cluster Means
         Xkmean.groupby('cluster').mean()
```

Out[17]:

cluster	Sepal length(x1)	Sepal width(x2)	Petal length(x3)	Petal width(x4)
0	0.196111	0.595000	0.078305	0.060833
1	0.441257	0.307377	0.575715	0.549180
2	0.707265	0.450855	0.797045	0.824786

(e) We plot the first two principal component scores again, but this time on our clusters obtained using K-means. The plot is shown below with the axes and color coding the same as before. Notice that some of the data which was labeled as class 1 (orange) in hierarchical cluster analysis is now labeled as class 2 (green) in K-means cluster analysis.

In [19]:
```python
plt.figure(figsize = (15,10))
plt.xlabel('Prin2') ; plt.ylabel('Prin1')
color = {0:'red', 1:'orange', 2:'green'}
for i in range(len(x)):
    plt.scatter(pc2[i,1], pc2[i,0],
                marker = "$ {} $".format(Xkmean['cluster'][i]),
                s = 100,
                c = color[Xkmean['cluster'][i]])
```



(f) We can use confusion matrix to compare the results of the two cluster analyses. We can see that the result is consistent with the principal component score plots. Data points classified to class 0 (red) are the same. However, 9 observations which were classified as class 1 (orange) in hierarchical cluster analysis is classified as class 2 (green) in K-means cluster analysis. Besides these data points, the clustering result is the same, so we can say that the two methods produce similar clusters.

In [21]:
```python
#(f) Compare the results of the two cluser analyses using confusion matrix. Do they produce similar clusters?
# Confusion Matrix
y_hier = hier_clusters
y_kmean = Xkmean['cluster']

C2 = pd.DataFrame(confusion_matrix(y_hier, y_kmean),
                  index = np.sort(y_hier.unique()),
                  columns = np.sort(y_kmean.unique()))
C2['Total'] = C2.sum(axis = 1) #row sum
C2.loc['Total',:] = C2.sum(axis = 0) # column sum
C2.index.names = ['Hierarchical'] ; C2.columns.names = ['K-means']
C2.astype(int)
```

Out[21]:

K-means / Hierarchical	0	1	2	Total
0	50	0	0	50
1	0	61	9	70
2	0	0	30	30
Total	50	61	39	150