K-Means Clustering (Advanced)

Measures of Cluster

- Radius of Cluster
- Diameter of Cluster
- Homogeneity Score

The Measures Themselves

1. Radius Of Cluster

Quite crudely, the radius of a cluster is defined as the radius of the smallest circle that can be drawn such that it covers all the points in that cluster. There is no standard package or provision in python to calculate this measure automatically. Thus a small code was written in python to achieve this. The snippet is shown here.

```
clusters_centroids=dict()
clusters_radii= dict()

'''looping over clusters and calculate Euclidian distance of
each point within that cluster from its centroid and
pick the maximum which is the radius of that cluster'''

print("radii are : ")

for cluster in list(set(y)):

    clusters_centroids[cluster]=list(zip(estimator.cluster_centers_[:, 0],estimator.cluster_centers_[:,1]))[cluster]

    bb = zip( X[ ( y_kmeans == cluster ), 0 ], X[ ( y_kmeans == cluster ), 1 ] )

    gp = []

    for i in bb:
        kg = math.sqrt( pow( ( i[0] - clusters_centroids[cluster][0] ) , 2 ) + pow( ( i[1] - clusters_centroids[cluster][1] ), 2 ) )
        gp.append( kg )

    clusters_radii[cluster]=max( gp )

    print ( cluster, ":", clusters_radii[cluster] )
```

```
radii are :
0 : 1.1876061757670695
1 : 1.2362928455669395
2 : 1.2767280940117136
```

2. Diameter Of Cluster

This measure is very related to the one above in that fact that it is 2x it's value, as the diameter is twice its radius. The code above can be used exactly to calculate this.

3. Homogeneity Score

Homogeneity is defined as a situation where each cluster contains only members of a single class. A clustering result satisfies homogeneity if all of its clusters contain only data points which are members of a single class.

This metric is independent of the absolute values of the labels: a permutation of the class or cluster label values won't change the score value in any way.

This metric is not symmetric.

That is, the class distribution within each cluster should be skewed to a single class, that is, zero entropy. We determine how close a given clustering is to this ideal by examining the conditional entropy of the class distribution given the proposed clustering. In the perfectly homogeneous case, this value, H(C|K), is 0. However, in an imperfect situation, the size of this value, in bits, is dependent on the size of the dataset and the distribution of class sizes. Therefore, instead of taking the raw conditional entropy, we normalize this value by the maximum reduction in entropy the clustering information could provide, specifically, H(C). Note that H(C|K)is maximal(and equals H(C)) when the clustering provides no new information the class distribution within each cluster is equal to the overall class distribution. H(C|K) is 0 when each cluster contains only members of a single class, a perfectly homogenous clustering. In the degenerate case where H(C) = 0, when there is only a single class, we define homogeneity to be 1. For a perfectly homogenous solution, this normalization, H(C|K) / H(C) , equals 0. Thus, to adhere to the convention of 1 being desirable and 0 undesirable, we define homogeneity as:

$$h = \begin{cases} 1 & \text{if } H(C,K) = 0 \\ 1 - \frac{H(C|K)}{H(C)} & \text{else} \end{cases} \quad (1)$$

where

$$H(C|K) = -\sum_{k=1}^{|K|}\sum_{c=1}^{|C|}\frac{a_{ck}}{N}\log\frac{a_{ck}}{\sum_{c=1}^{|C|}a_{ck}}$$

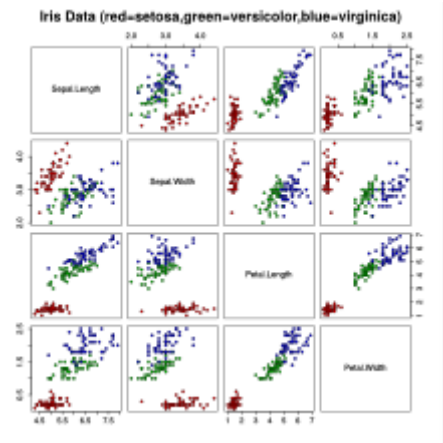
$$H(C) = -\sum_{c=1}^{|C|}\frac{\sum_{k=1}^{|K|}a_{ck}}{n}\log\frac{\sum_{k=1}^{|K|}a_{ck}}{n}$$

Pythons SKLearn package gives a predefined library to help you calculate this score, and can be invoked using:

```
from sklearn.metrics.cluster import homogeneity_score
```

The Dataset

The dataset used was the IRIS dataset that contains 4 features of 150 flowers.



Scatterplot of the data set

The ***Iris* flower data set** or **Fisher's *Iris* data set** is a multivariate data set introduced by the British statistician and biologist Ronald Fisher in his 1936 paper *The use of multiple measurements in taxonomic problems* as an example of linear discriminant analysis. It is sometimes called **Anderson's *Iris* data set** because Edgar Anderson collected the data to quantify the morphologic variation of *Iris* flowers of three related species. Two of the three species were collected in the Gaspé Peninsula "all from the same pasture, and picked on the same day and measured at the same time by the same person with the same apparatus".

The data set consists of 50 samples from each of three species of *Iris* (*Iris setosa*, *Iris virginica* and *Iris versicolor*). Four features were measured from each sample: the length and the width of the sepals and petals, in centimetres. Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other.

The Code

```
[ ]  import numpy as np
     import math
     from sklearn.cluster import KMeans
     from sklearn import datasets
     import matplotlib.pyplot as plt
     import matplotlib.patches as mpatches
```

```
[ ]  iris = datasets.load_iris()
     X = iris.data
     y = iris.target
```

```
[ ]  estimator = KMeans(n_clusters=3)
     estimator.fit(X)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

```
print({i: np.where(estimator.labels_ == i)[0] for i in range(estimator.n_clusters)}) #get the indices of points for each cluster
```

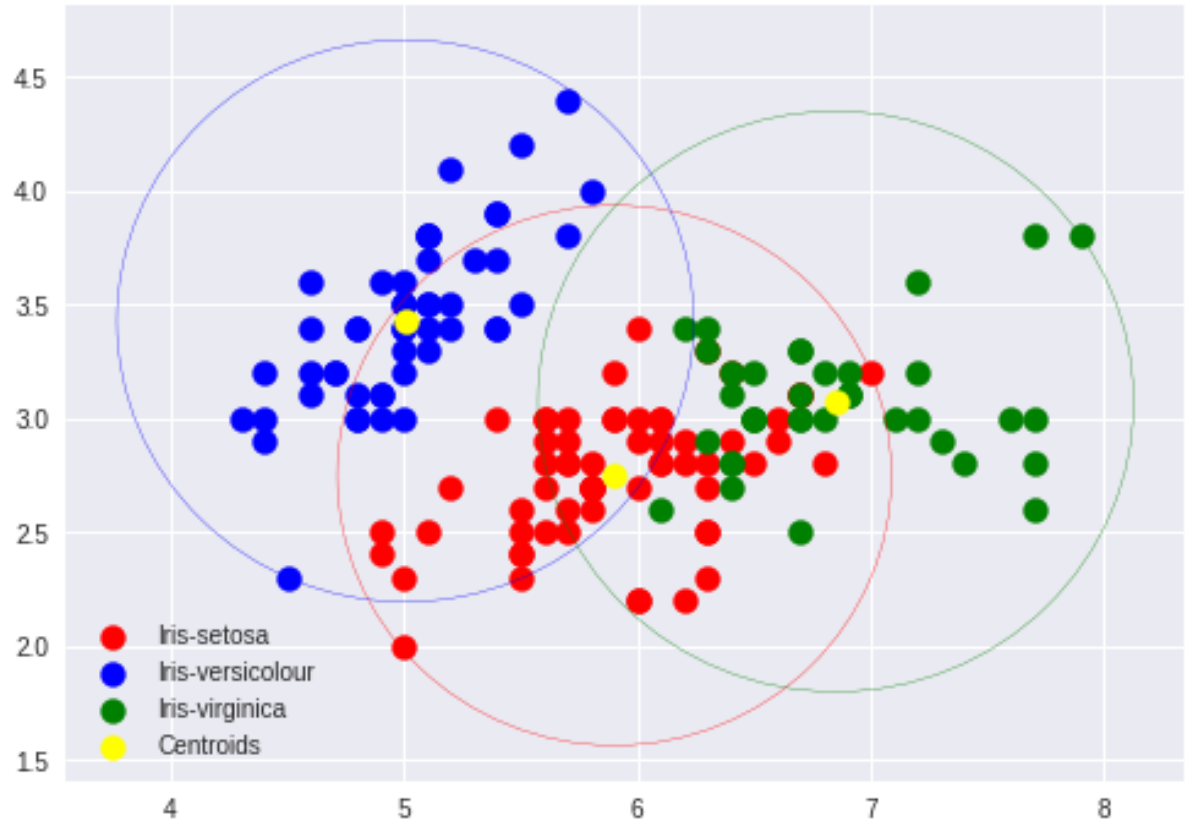
```
{0: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]), 1: array([ 50,  51,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,
        64,  65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,
        78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,
        91,  92,  93,  94,  95,  96,  97,  98,  99, 101, 106, 113, 114,
       119, 121, 123, 126, 127, 133, 138, 142, 146, 149]), 2: array([ 52,  77, 100, 102, 103, 104, 105, 107, 108, 109, 110, 111, 112,
       115, 116, 117, 118, 120, 122, 124, 125, 128, 129, 130, 131, 132,
       134, 135, 136, 137, 139, 140, 141, 143, 144, 145, 147, 148])}
```

```
y_kmeans = estimator.fit_predict(X)
```

```
clusters_centroids=dict()
clusters_radii= dict()

'''looping over clusters and calculate Euclidian distance of
each point within that cluster from its centroid and
pick the maximum which is the radius of that cluster'''

print("radii are : ")

for cluster in list(set(y)):

    clusters_centroids[cluster]=list(zip(estimator.cluster_centers_[:, 0],estimator.cluster_centers_[:,1]))[cluster]

    bb = zip( X[ ( y_kmeans == cluster ), 0 ], X[ ( y_kmeans == cluster ), 1 ] )

    gp = []

    for i in bb:
        kg = math.sqrt( pow( ( i[0] - clusters_centroids[cluster][0] ) , 2 ) + pow( ( i[1] - clusters_centroids[cluster][1] ), 2 ) )
        gp.append( kg )

    clusters_radii[cluster]=max( gp )

    print ( cluster, ":", clusters_radii[cluster] )
```

```
radii are :
0 : 1.1876061757670695
1 : 1.2362928455669395
2 : 1.2767280940117136
```

```
fig, ax = plt.subplots(1,figsize=(7,5))

plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Iris-setosa')
art = mpatches.Circle(clusters_centroids[0],clusters_radii[0], edgecolor='r',fill=False)
ax.add_patch(art)

plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Iris-versicolour')
art = mpatches.Circle(clusters_centroids[1],clusters_radii[1], edgecolor='b',fill=False)
ax.add_patch(art)

plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Iris-virginica')
art = mpatches.Circle(clusters_centroids[2],clusters_radii[2], edgecolor='g',fill=False)
ax.add_patch(art)

#Plotting the centroids of the clusters
plt.scatter(estimator.cluster_centers_[:, 0], estimator.cluster_centers_[:,1], s = 100, c = 'yellow', label = 'Centroids')

plt.legend()
plt.tight_layout()
plt.savefig('kmeans.jpg',dpi=300)
plt.show()
```




```
from sklearn.metrics.cluster import homogeneity_score
print("Homogeneity Score: %.6f" % homogeneity_score(y_kmeans, y))
```

```
Homogeneity Score: 0.764986
```

All the codes can be found on the github repository here.

Interpretation

As the graph shows, the clusters are of the same size (from the radii – 1.18, 1.23, and 1.27) and the mixing between green and red is high but blue is completely separated. Also, the homogeneity score shows us that the clustering is comparatively very accurate and mispredictions are low in number.