Evaluation of Clustering

Cluster Validity

The validation of clusters created is a troublesome task. The problem here is: ”clusters are in the eyes of the beholder”

A good cluster will have:

- High inter-class similarity, and
- Low intraclass similarity

Aspects of cluster validation

- **External:** Compare your cluster to the ground truth.
- **Internal:** Evaluating the cluster without reference to external data.
- **Reliability:** The clusters are not formed by chance(randomly)- some statistical framework can be used.

External Measures:

1. Adjusted Rand index
2. Jaccard Score
3. Purity Score

1. Adjusted Rand index

N: Number of objects in the data P: $P_1, P_2, \dots, P_m$ the set of ground truth clusters C: $C_1, C_2, \dots C_n$ the set of clusters formed by the algorithm

The Incidence Matrix: N* N matrix

$P_i j = 1$ if the two points O_i and O_j belong to the same cluster in the ground truth else $P_i j = 0$

$C_i j = 1$ if the two points O_i and O_j belong to the same cluster in the ground truth else $C_i j = 0$

Now there can be the following scenarios:

1. $C_i j = P_i j = 1$ --> both the points belong to the same cluster for both our algorithm and ground truth(Agree)--- **SS**
2. $C_i j = P_i j = 0$ --> both the points don’t belong to the same cluster for both our algorithm and ground truth(Agree)--- **DD**
3. ${}_C i j = 1 but P_i j = 0$ --> The points belong in the same cluster for our algorithm but in different clusters for the ground truth (Disagree)---- **SD**
4. $C_i j = 0 but P_i j = 1$ --> The points don’t belong in the same cluster for our algorithm but in same clusters for the ground truth (Disagree)----**DS**

Rand Index= $\frac{Total Agree}{Total Disagree} = \frac{(SS+DD)}{(SS+DD+DS+SD)}$

Given the knowledge of the ground truth class assignments labels_true and our clustering algorithm assignments of the same samples labels_pred, the adjusted Rand index is a function that measures the similarity of the two assignments, ignoring permutations and with chance normalization:

In [1]:
```
from sklearn import metrics
y_true = [0, 0, 0, 1, 1, 1]
y_pred = [0, 0, 1, 1, 2, 2]
```

In [2]:
```
print("adjusted_rand_score : %0.3f" % metrics.adjusted_rand_score(y_true, y_pred
```

```
adjusted_rand_score : 0.242
```

The Adjusted Rand index is bounded between -1 and 1. Closer to 1 is good, while closer to -1 is bad.

2. Jaccard Score

Jaccard Coeeficient= $\frac{SS}{(SS+SD+DS)}$

In [3]:
```
print("jaccard_score: %0.3f" % metrics.jaccard_score(y_true, y_pred, average='mac
```

```
jaccard_score: 0.306
```

jaccard_score may be a poor metric if there are no positives for some samples or classes. Jaccard is undefined if there are no true or predicted labels, and our implementation will return a score of 0 with a warning.

A higher value of Rand Index and Jaccard's coefficient mean that the clusters generated by our algorithm mostly agree to the ground truth.

3. Purity Score:

Entropy of Cluster i, given by $e_i = -\sum p_{ij} log(p_{ij})$

For the entire clustering algorithm, the entropy can be given as: $e = \sum \frac{m_i}{n} e_i$

Purity is the total percentage of data points clustered correctly.

The purity of cluster i, given by $p_i = max(p_{ij})$

And for the entire cluster it is: $p(C) = \sum \frac{m_i}{n} p_i$

The Scikit-Learn Package hasen't yet implemented the Purity metrics. Hence, we'll write our custom code to implememt that.

In [4]:
```
import numpy as np
from sklearn import metrics

def purity_score(y_true, y_pred):
    # compute contingency matrix (also called confusion matrix)
    contingency_matrix = metrics.cluster.contingency_matrix(y_true, y_pred)
    # return purity
    return np.sum(np.amax(contingency_matrix, axis=0)) / np.sum(contingency_matri
```

In [5]:
```
print("purity_score is : ",purity_score(y_true, y_pred).round(2))
```

```
purity_score is :  83.33
```

A high value of purity score means that our clustering algorithm performs well against the ground truth.

So far we have discussed the External Measures. But as it is an unsupervised learning problem, most of the times there is no ground truth to compare to. Let's discuss the internal measures.

Internal Measure

1. silhouette_score
2. Davies-Bouldin Index (DB Index)
3. Dunn Index

1. silhouette_score

Note: BSS+WSS is always a constant.

The silhouette can be calculated as:

$$s(x) = \frac{b(x) - a(x)}{\max\{a(x), b(x)\}}$$

Where a(x) is the avarage distance of x from all the other points in the same cluster and b(x) is the avarage distance of x from all the other points in the other clusters.

And the Silhoeutte coefficient is given by:

$$SC = \frac{1}{N} \sum S(x)$$

In [6]:
```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Generating the sample data from make_blobs

X, Y = make_blobs()

cluster = KMeans(n_clusters = 5)
cluster_labels = cluster.fit_predict(X)

# The silhouette_score gives the
# average value for all the samples.
silhouette_avg = silhouette_score(X, cluster_labels)

print("The average silhouette_score is :", silhouette_avg)
```

```
The average silhouette_score is : 0.45794199236538047
```

2. Davies-Bouldin Index (DB Index) :

- The DB Index is calculated by the following formula:

$$DB = \frac{1}{n}\sum_{i=1}^{n} max_{j \neq i}\left(\frac{\sigma_i + \sigma_j}{d(c_i, c_j)}\right)$$

where:

n is the number of clusters

σi is the average distance of all points in cluster i from the cluster centroid ci.

- The score is defined as the average similarity measure of each cluster with its most similar cluster, where similarity is the ratio of within-cluster distances to between-cluster distances. Thus, clusters which are farther apart and less dispersed will result in a better score.

In [8]:
```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import davies_bouldin_score
import warnings
warnings.simplefilter('ignore')

# loading the dataset
X, Y = make_blobs()

# K-Means
cluster = KMeans(n_clusters = 4)
cluster_labels = cluster.fit_predict(X)


print("DB Index Score: ",davies_bouldin_score(X, cluster_labels))
```

```
DB Index Score:  0.5765098067760293
```

3. Dunn Index

- The aim of this Dunn index to identify sets of clusters that are compact, with a small variance between members of the cluster, and well separated, where the means of different clusters are sufficiently far apart, as compared to the within cluster variance.
- Higher the Dunn index value, better is the clustering.

$$\text{Dunn index}(U) = \min_{1 \le i \le c} \left\{ \min_{1 \le j \le c,\, j \ne i} \left\{ \frac{\delta(X_i, X_j)}{\max_{1 \le k \le c} [\Delta(X_k)]} \right\} \right\}$$

where:

$\delta(X_i, X_j)$ is the intercluster distance i.e. the distance between cluster X_i and X_j

$\Delta(X_k)$ is the intracluster distance of cluster X_k i.e. distance within the cluster X_k

- it gets better when clusters are well-spaced and dense. But the Dunn Index increases as performance improves.

```
import pandas as pd
from sklearn import datasets
from jqmcvi import base

# loading the dataset
X = datasets.load_iris()
df = pd.DataFrame(X.data)

# K-Means
from sklearn import cluster
k_means = cluster.KMeans(n_clusters=3)
```

```
k_means.fit(df) #K-means training
y_pred = k_means.predict(df)

# We store the K-means results in a dataframe
pred = pd.DataFrame(y_pred)
pred.columns = ['Type']

# we merge this dataframe with df
prediction = pd.concat([df, pred], axis = 1)

# We store the clusters
clus0 = prediction.loc[prediction.Species == 0]
clus1 = prediction.loc[prediction.Species == 1]
clus2 = prediction.loc[prediction.Species == 2]
cluster_list = [clus0.values, clus1.values, clus2.values]

print(base.dunn(cluster_list))
```