PW4-Unsupervised machine learning: Kmeans, Hclustering, DBSCAN

ESILV (A4-All) De Vinci Group

Zhiqiang WANG @ 2023-2024

Requirements for the work: the exercise 10 and exercise 11 should be uploaded on the DVL repository before the deadline. You should create your colab file correctly named.

Learning outcomes PW4

In this section, we will talk about three unsupervised machine learning algorithmes: Hierarchical Clustering, Kmeans and DBSCAN. We use the rand index metric to evaluate the performances for each algorithmes and compare them with the random clustering algorithm.

1. Part I: Generate the orignam data with label manually and define the metric function (rand index).
2. Part II: Clustering the orignal data by Hierarchical Clustering.
3. Part III: Clustering the orignal data by Kmeans.
4. Part IV: Clustering the orignal dayta by DBSCAN.

Part I: Generate the orignam data with label manually and define the metric function (rand index).

Import tools. Please import others if needed

In [ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
import scipy as sc
from scipy import cluster
import sklearn as sk
from sklearn import datasets
import pandas as pd
```

Exercise 1: Generate a synthetic dataset

1. Generate a set of 100 points in a 2 dimensional space split into 4 non overlapping clusters. You may need the function ***np.concatenate*** and the function ***np.random.rand()***

In [ ]:
```
syn=np.empty([0, 2]);
synLabels=[];
for k in range(4):
    syn = np.concatenate((syn, k*2+np.random.rand(25, 2)), axis=0);
    synLabels = np.concatenate((synLabels, np.ones(25)*k));
```

In [ ]:
```
syn
synLabels
```

1. Display the set with one color per cluster using the scatter function from matplotlib.pyplot. Here, you may use ***plt.scatter(X, Y, color)***

In [ ]:
```
plt.scatter(syn[:, 0], syn[:, 1], c=synLabels);
plt.show();
```

1. Cluster this orignal data by the random clustering algorithm as the **Baseline**.

Cluster this dataset into k clusters by assigning a random integer value between 0 and k-1 to each point. You may need the function ***np.random.randint()***

In [ ]:
```
def randomClustering(syn, k):
    return np.random.randint(0, k, syn.shape[0])

rand = randomClustering(syn, 4)
```

In [ ]:
```
rand
```

Exercise 2: Creat the rand index as the metric to evaluate the performance of clustering.

The Rand index or Rand measure (named after William M. Rand) in statistics, and in particular in data clustering, is a measure of the similarity between two data clusterings. A form of the Rand index may be defined that is adjusted for the chance grouping of elements, this is the adjusted Rand index. The Rand index is the accuracy of determining if a link belongs within a cluster or not.

Given a set of n elements $S = o_1, \ldots, o_n$ and two partitions of S to compare, $X = X_1, \ldots, X_r$, a partition of S into r subsets, and $Y = Y_1, \ldots, Y_s$, a partition of S into s subsets, define the following:

```
a, the number of pairs of elements in S that are in the same subset
in X and in the same subset in Y;
b, the number of pairs of elements in S that are in different
subsets in X and in different subsets in Y;
c, the number of pairs of elements in S that are in the same subset
in X and in different subsets in Y;
d, the number of pairs of elements in S that are in different
subsets in X  and in the same subset in Y.
```

The Rand index, R, is:

$$R = \frac{a+b}{a+b+c+d} = \frac{a+b}{C_n^2} = \frac{a+b}{n(n-1)/2}$$

Intuitively, $a + b$ can be considered as the number of agreements between X and Y and $c + d$ as the number of disagreements between X and Y.

Since the denominator is the total number of pairs, the Rand index represents the frequency of occurrence of agreements over the total pairs, or the probability that X and Y will agree on a randomly chosen pair.

Implement the rand index criterion (see https://en.wikipedia.org/wiki/Rand_index for reference)

In [ ]:
```python
def rand_index(ref, est):
    num = 0
    if len(ref) != len(est):
        print('arrays must be of the same size')
        error()
    for k in range(len(ref)):
        for l in range(len(ref)-k-1):
            if (ref[k]==ref[k+l+1] and est[k]==est[k+l+1]) or (ref[k]!=ref[k+l+1] a
                num += 1
    den = len(ref)*(len(ref)-1)/2
    return num/den
```

Calculate the rand Index for random clustering (the ground truth is given at the first step as **synLabels**)

```
In [ ]: ri = rand_index(rand, rand)
        print(ri)
        ri = rand_index(synLabels, rand)
        print(ri)
```

Compute the rand index between the reference clustering and 100 runs of the baseline algorithm.

```
In [ ]: ri = np.zeros((100))
        for k in range(100):
            ri[k] = rand_index(synLabels, randomClustering(syn, 4))
```

Display results and compute the mean and standard deviation.

```
In [ ]: plt.bar(0, np.mean(ri), yerr = np.std(ri), alpha=0.5, ecolor='black', capsize=10)
        plt.show()
```

Part II: Hierarchical Clustering

Exercise 3: Compute the euclidean distance matrix using the pdist function from *scipy.spatial.distance*.

```
In [ ]: import scipy
        from scipy.cluster.hierarchy import linkage
        euc_distance_matrix = scipy.spatial.distance.pdist(syn)
```

Display and interpret its shape

```
In [ ]: #We compute the squareform with scipy.spatial.distance.squareform
        #and plot a heatmap through seaborn
        import seaborn as sbn
        squareform = scipy.spatial.distance.squareform(euc_distance_matrix)
        plt.title("Eucledian distance matrix ")
        sbn.heatmap(squareform)
        plt.show()
```

answer here:

Compute the single link hierarchical clustering using the linkage function from ***scipy.cluster.hierarchy***.

```
In [ ]: l = sc.cluster.hierarchy.linkage(d)
```

Exercise 4: Display the corresponding dendrogram using the dendrogram function from *scipy.cluster.hierarchy*.

In [ ]:
```
plt.figure()
dn = sc.cluster.hierarchy.dendrogram(l)
```

Implement a clustering algorithm that cuts the dendrogram in order to produce k clusters using the fcluster function from ***scipy.cluster.hierarchy***.

In [ ]:
```
def agglomerativeClustering(x, k):
    d = sc.spatial.distance.pdist(x)
    l = sc.cluster.hierarchy.linkage(d)
    return sc.cluster.hierarchy.fcluster(l, k, criterion='maxclust')

agg = agglomerativeClustering(syn, 4)
```

Exercise 5: Compute the rand index between the reference clustering and 100 runs of this clustering algorithm.

In [ ]:
```
ri = np.zeros((100))
for k in range(100):
    ri[k] = rand_index(synLabels, agglomerativeClustering(syn, 4))
```

Display results and compute the mean and variance.

In [ ]:
```
plt.bar(0, np.mean(ri), yerr = np.std(ri), alpha=0.5, ecolor='black', capsize=10)
plt.show()
```

Explain why the standard deviation is 0.

The algorithm is deterministic.

Part III: Partitional Clustering - Kmeans

Exercise6: Implement the k-means algorithm (see https://en.wikipedia.org/wiki/K-means_clustering section Standard algorithm for reference).

Hint: please consider the cdist function from scipy.spatial.distance to compute the distance of the points to the centroids.

In [ ]:
```
def kMeans(dataset, k, nbIterations=1000, show=False):

    centroids = np.random.random((k, dataset.shape[1]))*np.max(dataset)
    labels = np.zeros((dataset.shape[0]))
    for i in range(nbIterations):
        if show:
            plt.scatter(dataset[:, 0], dataset[:, 1], c=labels);
            plt.scatter(centroids[:, 0], centroids[:, 1], s=200);
            plt.show();

        cd = sc.spatial.distance.cdist(centroids, dataset)
        pastLabels = labels
        for d in range(dataset.shape[0]):
            labels[d] = np.argmin(cd[:, d])
        for c in range(k):
            centroids[c, :] = np.mean(dataset[labels==c, :], axis=0)

        if (np.all(pastLabels == labels)):
            break
    return labels

labels = kMeans(syn, 4)
```

Exercise7: Compute the rand index between the reference clustering and 100 runs of this clustering algorithm.

In [ ]:
```
ri = np.zeros((100))
for k in range(100):
    ri[k] = rand_index(synLabels, kMeans(syn, 4))
```

Display results and compute the mean and variance.

In [ ]:
```
plt.bar(0, np.mean(ri), yerr = np.std(ri), alpha=0.5, ecolor='black', capsize=10)
plt.show()
```

Part IV: DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu in 1996. It is a density-based clustering non-parametric algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away). DBSCAN have two parameters important: eps and min_sample. Unlike Kmeans, we don't have to specify the number of clusters before clustering.

Exercise8: Clustering the data syn by DBSCAN.

Here, you can use directly *sklearn.cluster.DBSCAN(eps=0.5, min_samples=5, metric='euclidean')*

In [ ]:
```
from sklearn.cluster import DBSCAN
clustering_DBSCAN = DBSCAN(eps=0.5, min_samples=5).fit(syn)
```

After clustering, give us the labels for each observation.

In [ ]:
```
clustering_DBSCAN.labels_
```

Compute the rand index between the reference clustering and 100 runs of this DBSCAN clustering algorithm.

In [ ]:
```
ri = np.zeros((100))
for k in range(100):
    clustering_DBSCAN = DBSCAN(eps=0.5, min_samples=5).fit(syn)
    ri[k] = rand_index(synLabels, clustering_DBSCAN.labels_)
```

In [ ]:
```
plt.bar(0, np.mean(ri), yerr = np.std(ri), alpha=0.5, ecolor='black', capsize=10)
plt.show()
```

Exercise9: Performance Analysis

Display the performance of the 4 clustering algorithms (Random, Hierarchical Clustering, Kmeans and DBSCAN) on the synthetic dataset using the bar function from matplotlib.pyplot.

In [ ]:
```
ri = np.zeros((4, 100))
for k in range(100):
    ri[0, k] = rand_index(synLabels, randomClustering(syn, 4))
    ri[1, k] = rand_index(synLabels, agglomerativeClustering(syn, 4))
    ri[2, k] = rand_index(synLabels, kMeans(syn, 4))
    ri[3, k] = rand_index(synLabels, DBSCAN(eps=0.5, min_samples=5).fit(syn).labels

x = [0, 1, 2, 3]
plt.bar(x, np.mean(ri, axis=1), yerr = np.std(ri, axis=1), alpha=0.5, ecolor='black
plt.ylabel('rand index')
plt.xticks(x, ['Random', 'Hierarchical', 'kmeans', 'DBSCAN'])
plt.show()
```

# Exercise10: Load the iris dataset using the *sklearn.datasets.load_iris* function from scikit-learn and perform the same performance analysis using this dataset.

This data sets consists of 3 different types of irises' (Setosa, Versicolour, and Virginica) petal and sepal length, stored in a 150x4 numpy.ndarray. This dataset have reference label in the '*target*' column. The 3 features are included inside the '*data*'.

The rows being the samples and the columns being: *Sepal Length, Sepal Width, Petal Length and Petal Width.*

# Exercise11: Load the Breast cancer wisconsin (diagnostic) dataset

Use the sklearn.datasets.load_breast_cancer function from scikit-learn and perform the same performance analysis using this dataset.

The breast cancer dataset is a classic and very easy binary classification dataset. The featuers are stored in 'data'. And the reference are stored in the 'target' .

Exercise12: Determining the number of clusters for KMeans

Implement the gap statistic method for determining the optimal number of clusters for the 3 datasets. Gap statistics compares the change in within-cluster dispersion with the uniform distribution. A large gap statistics value means that the clustering is very different from the uniform distribution.

In [ ]:
```
#We use K-means from scikit-learn in this method.
#The code is taken from the article: https://grabngoinfo.com/5-ways-for-deciding-nu

from sklearn.cluster import KMeans
def optimalK(data, nrefs=3, maxClusters=15):
    """
    Calculates KMeans optimal K using Gap Statistic from Tibshirani, Walther, Hasti

    Parameters
    ----------
    data:
        ndarry of shape (n_samples, n_features)
    nrefs:
        number of sample reference datasets to create
    maxClusters:
        Maximum number of clusters to test for

    Returns
    -------
    (gaps, optimalK): tuple
    """
    gaps = np.zeros((len(range(1, maxClusters)),))
    resultsdf = pd.DataFrame({'clusterCount':[], 'gap':[]})
    for gap_index, k in enumerate(range(1, maxClusters)):
        # Holder for reference dispersion results
        refDisps = np.zeros(nrefs)
        # For n references, generate random sample and perform kmeans getting resul
        for i in range(nrefs):

            # Create new random reference set
            randomReference = np.random.random_sample(size=data.shape)

            # Fit to it
            km = KMeans(k)
            km.fit(randomReference)

            refDisp = km.inertia_
            refDisps[i] = refDisp
        # Fit cluster to original data and create dispersion
        km = KMeans(k)
        km.fit(data)

        origDisp = km.inertia_
        # Calculate gap statistic
        gap = np.log(np.mean(refDisps)) - np.log(origDisp)
        # Assign this loop's gap statistic to gaps
        gaps[gap_index] = gap

        resultsdf = resultsdf.append({'clusterCount':k, 'gap':gap}, ignore_index=Tr
    return (gaps.argmax() + 1, resultsdf)  # Plus 1 because index of 0 means 1 clus
```

In [ ]:
```
# We calculate the three gap-statistics and print for the baseline algorithm:
k, gapdf = optimalK(syn, nrefs=3, maxClusters=11)
print('Optimal k is: ', k)
# Visualization
plt.plot(gapdf.clusterCount, gapdf.gap, linewidth=3)
plt.scatter(gapdf[gapdf.clusterCount == k].clusterCount, gapdf[gapdf.clusterCount =
plt.grid(True)
plt.xlabel('Cluster Count')
plt.ylabel('Gap Value')
plt.title('Gap Values by Cluster Count for synthetic data')
plt.show()
```

In [ ]:
```
#We repeat the analysis for the iris data.
dataset = sk.datasets.load_iris()
iris_data = dataset['data']
k, gapdf = optimalK(iris_data, nrefs=3, maxClusters=10)
print('Optimal k is: ', k)
# Visualization
plt.plot(gapdf.clusterCount, gapdf.gap, linewidth=3)
plt.scatter(gapdf[gapdf.clusterCount == k].clusterCount, gapdf[gapdf.clusterCount =
plt.grid(True)
plt.xlabel('Cluster Count')
plt.ylabel('Gap Value')
plt.title('Gap Values by Cluster Count for iris_data')
plt.show()
```

In [ ]:
```
#We repeat the analysis for the cancer data.
dataset = sk.datasets.load_breast_cancer()
cancer_data = dataset['data']
k, gapdf = optimalK(cancer_data, nrefs=3, maxClusters=11)
print('Optimal k is: ', k)
# Visualization
plt.plot(gapdf.clusterCount, gapdf.gap, linewidth=3)
plt.scatter(gapdf[gapdf.clusterCount == k].clusterCount, gapdf[gapdf.clusterCount =
plt.grid(True)
plt.xlabel('Cluster Count')
plt.ylabel('Gap Value')
plt.title('Gap Values by Cluster Count for cancer_data')
plt.show()
```