Lab7: Clustering

Deadline for submission: 13 October 2022, 11:55 PM

You are given with Iris flower dataset file. The file `Iris.csv` consists of 150, 4-dimensional data which includes 50 samples from each of the three species of Iris (Iris setose, Iris virginica and Iris versicolor). Column 1 to 4 of the given file are the four features (attributes) that were measured from each sample: the length and the width of the sepals and petals (in centimetres) respectively. Column 5 is the class label (species name) associated with each of the samples of Iris flower.

Task here is to reduce the data into 2-dimensional data using PCA and then partition (cluster) the reduced dimensional data using different clustering techniques. While performing the PCA, ignore the fifth column of the data. *Use the class information in fifth column only for computing purity score*.

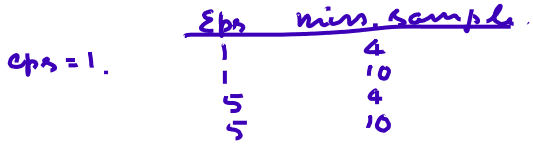
1. Apply PCA on the 4-dimensional and select first two directions (eigen vectors corresponding to 2 leading eigenvalues) to convert the data in to 2-dimensional data. (Exclude the attribute "`Species`" for PCA). => Plot eigen values - Plot data (2-d)
2. Apply *K*-means (*K*=3) clustering on the reduced data.
    a. Plot the data points with different colours for each cluster. Mark the centres of the clusters in the plot.
    b. Obtain the sum of squared distances of samples to their closest cluster centre (distortion measure).
    c. Compute the purity score after examples are assigned to clusters.

    Note: Use `kmeans.fit` to train the model and `kmeans.labels_` to obtain the cluster labels

3. Repeat the *K*-means clustering for number of clusters (*K*) as 2, 3, 4, 5, 6 and 7. Record the distortion measure for each of the *K* values. Give the plot of *K* vs distortion measure. Find the optimum number of clusters using elbow method for *K*-means clustering. Compute the purity score for the different number of clusters after examples are assigned to clusters. Compare the purity score for clusters with different values of *K* with that of the purity score for optimum cluster.

4. Use Gaussian mixture model (GMM) to cluster reduced dimensional data points into 3 clusters. Assign the data points to the clusters for which the posterior probability of cluster is maximum.
    a. Plot the data points with different colours for each cluster. Mark the centres of the clusters in the plot. Compare the boundary of clusters with that of k-means
    b. Give the total data log likelihood at the last iteration of the GMM as distortion measure.
    c. Compute the purity score after examples are assigned to clusters.

5. Repeat the GMM as soft-clustering techniques for number of clusters (*K*) as 2, 3, 4, 5, 6 and 7. Record the total data log likelihood (distortion measure) for each of the *K* values. Give the plot of *K* vs total data log likelihood. Find the optimum number of clusters using elbow method for GMM clustering. Compute the purity score for the different number of clusters after examples are assigned to clusters. Compare the purity score for clusters with different values of *K* with that of the purity score for optimum cluster.



6. Apply the DBSCAN clustering using different values for *eps* and *min_samples*. Consider *eps* = 1 and 5 & *min_samples* = 4 and 10. For each combination of *eps* and *min_samples* observe the number of clusters. Here, *eps* (Epsilon) is a value of radius of boundary from every example and *min_samples* is minimum number of examples present inside the boundary with radius of *eps* from an example.
    a. Plot the data points with different colours for each cluster for each combination of *eps* and *min_samples*.
    b. Compute the purity score after examples are assigned to clusters obtained for each combination of *eps* and *min_samples*.

Functions/Code Snippets:

 # K-means
```
from sklearn.cluster import KMeans

K = 3
kmeans = KMeans(n_clusters=K)
kmeans.fit(data)

kmeans_prediction = kmeans.predict(data)
```

 # GMM
```
from sklearn.mixture import GaussianMixture

K = 3
gmm = GaussianMixture(n_components = K)
gmm.fit(data)

GMM_prediction = gmm.predict(data)
```

 # DBSCAN
```
from sklearn.cluster import DBSCAN
dbscan_model=DBSCAN(eps=1, min_samples=10).fit(train_data)

DBSCAN_predictions = dbscan_model.labels_
```

 # Purity scores function
```
from sklearn import metrics
from scipy.optimize import linear_sum_assignment

def purity_score(y_true, y_pred):
    # compute contingency matrix (also called confusion matrix)
    contingency_matrix=metrics.cluster.contingency_matrix(y_true, y_pred)
    #print(contingency_matrix)

    # Find optimal one-to-one mapping between cluster labels and true labels
    row_ind, col_ind = linear_sum_assignment(-contingency_matrix)

    # Return cluster accuracy
    return contingency_matrix[row_ind,col_ind].sum()/np.sum(contingency_matrix)
```

 # Code snippet for assigning test points to clusters using DBSCAN:

Approach:
Runs through core points and assigns the test point to the cluster of the first core point that is within

eps radius of the test point. Then it is guaranteed that our test point is at least a border point of the assigned cluster.

Problem:
Differing prediction outcomes which stems from the possibility that a border point can be close to multiple clusters.

```
import numpy as np
import scipy as sp
from scipy import spatial as spatial

def dbscan_predict(dbscan_model, X_new, metric=spatial.distance.euclidean):
    # Result is noise by default
    y_new = np.ones(shape=len(X_new), dtype=int)*-1

    # Iterate all input samples for a label
    for j, x_new in enumerate(X_new):
        # Find a core sample closer than EPS
        for i, x_core in enumerate(dbscan_model.components_):
            if metric(x_new, x_core) < dbscan_model.eps:
                # Assign label of x_core to x_new
                y_new[j] =
                      dbscan_model.labels_[dbscan_model.core_sample_indices_[i]]
                break

    return y_new

dbtest = dbscan_predict(model, reduced_test_data, metric =
                                          spatial.distance.euclidean)
```

Instructions:

Your python program(s) should be well commented.

In case the program found to be copied from others, both the person who copied and who help for copying will get zero as a penalty.