Lab Class ML:XI

By 2018-01-31 solutions for the following exercises have to be submitted: 3, 4, 5.

Exercise 1 : Data Quality

Which of the following statements are true?

- ☐ Noise can sometimes be interesting or desirable.
- ☐ Outliers can sometimes be interesting or desirable.
- ☐ Noise can introduce outliers.

Exercise 2 : Cluster Analysis Principles

Which of the following statements are true?

- ☐ k-means is a supervised algorithm since the centroids are specified.
- ☐ The runtime of k-medoid is higher than that of k-means due to the medoid computation.
- ☐ Density-based cluster analysis is more efficient than single link.
- ☐ DBSCAN is particularly efficient in high dimensions.

Exercise 3 : Hierarchical Cluster Analysis

Consider hierarchical agglomerative clustering in a two-dimensional feature space with Manhattan distance. Construct a minimal example where the single link and group average link cluster distance measures produce different dendrograms.

Exercise 4 : Hierarchical Cluster Analysis

Given are the following objects from a two-dimensional feature space:

$x_0 = (-5, 4),\ x_1 = (-3, 4),\ x_2 = (-4, 2),\ x_3 = (6, 4),\ x_4 = (8, 4),\ x_5 = (5, 1),\ x_6 = (0, -2),\ x_7 = (2, 0)$

(a) Apply a hierarchical agglomerative clustering algorithm with complete link and Euclidean distance and draw the resulting dendrogram. Note the exact distances on the dendrogram's distance axis.

(b) The dendrogram gives rise to multiple possible clusterings, among them one with two and one with three clusters. Compute the Dunn Index cluster validity measure for both of these clusterings.

Exercise 5 : $\boxed{\text{P}}$ Exemplar-based Clustering

(a) Implement the exemplar-based iterative clustering algorithm as a Python function, based on the pseudocode given on the lecture slides, with the following signature:

```
def exemplar_based_clustering(X, k, dist, pick_exemplar, t_max):
    ### [ your code here ... ] ###
```

Your function should expect the following parameters:

- An n-by-p numpy array X, containing n points in p dimensions.
- An integer k specifying the number of clusters.
- A distance function dist, which takes two arguments—earch of them a p-dimensional point—and returns a real number.
- A function pick_exemplar which takes an m-by-p array (containing all the points in one cluster) as its only argument, and returns a p-dimensional point.
- An integer t_max specifying the maximum number of iterations.

Note that this deviates slightly from how the algorithm is specified on the slides. You may use the following "convergence" criterion: your implementation should terminate if it finds the exact same clusters in two consecutive iterations. The return value of your function should be a 2-tuple (C, R). The value C is a 1-dimensional array with n elements, each of which is an integer between zero and $k-1$, where the i-th element of C is your algorithm's cluster assignment for the i-th point in X. The value R is a k-by-p array, containing the k cluster exemplars selected by your algorithm.

(b) Verify that your function can be used to implement the k-means algorithm in the following way:

```
import numpy as np
euclidean_distance = lambda a, b: np.linalg.norm(a - b, ord=2)
pick_centroid = lambda cluster: np.mean(cluster, axis=0)
kmeans = lambda X, k, t_max: exemplar_based_clustering(
    X, k, euclidean_distance, pick_centroid, t_max)
```

Example usage:

```
>>> kmeans(np.array([[0, 0], [1, 1]]), 2, 100)
(array([1, 0]), array([[1, 1], [0, 0]]))
```

(Note that your implementation is still correct if the order of the elements in the returned arrays is different)

(c) Implement a function pick_medoid. Use this function, as well as euclidean_distance and exemplar_based_clustering, to implement the k-medoids algorithm.

(d) The following code snippet generates two artificial datasets of two-dimensional points. Run your implementations of k-means and k-medoids on the blobs and moons dataset, with different values of k. Visualize the points in a scatter plot, using different colors for the clusters found by your implementation, and mark the cluster exemplars.

```
import sklearn.datasets as sd
np.random.seed(42)
blobs = sd.make_blobs(1000, 2, 3)[0]
moons = sd.make_moons(1000, noise=.05)[0]
```

Note: you may have to install the scikit-learn library.