7 Clustering

7.1 K-means clustering

7.1.1 Theory

K-means is a clustering algorithm. Clustering algorithms are unsupervised techniques for sub-dividing a larger data set into smaller groups. The term "unsupervised" indicates that the data do not originate from clearly defined groups that can be used to label them a priori. For example, say that you're exploring a group of 300 hens in a barn and you've measured a bunch of parameters from each bird: height, weight, egg size, laying regularity, egg color, etc. Based on all these parameters, you want to figure out if the hens fall into a small number of distinct groups or if they constitute just a single homogeneous population. Note that this question has two features: 1) At the outset you don't know how many groups there are and therefore 2) you have no way of assigning a given hen to a given group. All hens are treated equally. For problems such as this, you can use k-means clustering to objectively assign each hen to a group. Using further techniques (described below) you can objectively test whether the assignments you've made are reasonable.

The k-mean clustering includes some steps:

- Step 1: Initialize randomly centroids of clusters using points from the dataset.
- Step 2: Assign datapoints to the clusters using a distance metric.
- Step 3: Compute new centroids using the mean of the clusters.
- Repeat step 2 and step 3 until there is no change in the clusters.

7.1.2 Python Exercise

In this exercise, you will implement k-means clustering algorithm to subdivide the `sklearn`'s *digits* dataset. The dataset contains images of hand-written digits from 0 to 9. Table 1 provides information regarding this dataset.

Table 1: *Digits* dataset.

Classes	10
Samples per class	~ 180
Samples total	1797
Dimensionality	64
Features	integers $0-16$

You are going to do the following steps.

- **Step 1**: In this step, you will load the dataset *Digits* with 5 classes. You can choose how much data you want to use, but the whole dataset will take you more time to process. Then, you will visualize some images in this dataset using `matplotlib`. Also, you will be provided the code to visualize the dataset using `t-SNE`, which is one of the strongest method for visualizing high dimensional data. An optional step is to try different data reduction methods for visualization such as PCA[10].

- **Step 2**: You have to implement function `kmeans` in this step. The function takes two inputs: the data from `Digits` and the number of clusters. The expected outputs are the centroids of clusters and the labels of the data.

- **Step 3**: This step will evaluate your clustering results. You will use the function `kmeans` to make clusters. Also, you will visualize some images from the clusters to make sure that members of a cluster are indeed similar.

- **Step 4**: In this step, you will calculate the *silhouette* score. *Silhouette* is a metric to measure the quality of clustering. The score is bounded between -1 for incorrect clustering and +1 for highly dense clustering. Scores around zero indicate overlapping clusters. The score is higher when clusters are dense and well separated, which relates to a standard concept of a cluster.

7.2 DBSCAN Clustering

During the last exercise session, we have worked with k-mean clustering algorithm. In this exercise, we will get hands-on experience with another the DBSCAN algorithm, which is also widely-used.

7.2.1 Theory

DBSCAN stands for "Density-Based Spatial Clustering of Application with Noise". Unlike the k-mean clustering algorithm, DBSCAN can discover clusters of arbitrary shapes, in which each sluter is a maximal set of density-connected points.

DBSCAN works in an iterative manner. It scan randomly points in a given dataset and mark the points as *core points*, *border-points* or *outliers*. If a *core point* is detected, a new cluster with all reachable points from this *core point* is found and saved. This process finishes when all the points are marked.

Unlike the k-mean clustering algorithm, DBSCAN does not require the number of clusters to be given as a parameter. This is a very important advantage of DBSCAN, since the number of clusters is normally difficult to determine. On the other hand, DBSCAN requires two parameters, namely the radius ϵ and the minimum number of

[10] http://scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html

points in a cluster. In fact, DBSCAN is highly sensitive to these two parameters. A way to effectively determine the suitable parameters is to calculate and analyze the distances between pairs of points in the dataset. Please refer to the lecture slides for more details.

7.2.2 Python Exercise

In this exercise, you will make a simple implementation of the DBSCAN algorithm and run experiments with it on the digit dataset from sklearn.

It should be noted that in this implementation, we work mostly with indices of points rather than the points themselves. We will only use the points for calculating the distances.

- **Step 1: Load the Digit dataset.** Use the function load digits from sklearn to load the dataset. In this exercise, we will only work with samples from 5 classes instead of all the samples in this dataset.
- **Step 2: Euclidean distance.** In this implementation, we will use the eu- clidean distance as our metric. For your convenience, a function to calculate the distances between pairs of points between two given matrices, in a vectorized form, has been provided.
- **Step 3: ϵ-neighborhood.** In this step, you need to implement a function, named find eps neighborhood, to find the ϵ-neighborhood of a given point of interest. The ϵ-neighborhood is defined as all the points which are within a distance ϵ from the point of interest. For convenience in the later steps, you need to return the resulting neighborhood as a set in Python.
- **Step 4: Reachable points.** In this step, a function named find reachable pts is provided for you. This function call find eps neighborhood to gradually find all points in the dataset which are reachable from the point of interest, w.r.t. the parameter ϵ.
- **Step 5: DBSCAN algorithm.** After completing some the necessary utility functions in the previous steps, you are all set to write the DBSCAN function. You will need to follow the code and fill the missing parts to finish this function.
- **Step 6: Experiment.** In this step, you will run experiments with the DBSCAN implementation you have finished in Step 5. First, try with the given pa- rameters for DBSCAN $\epsilon = 20.0$ and $\mathtt{minPts} = 10$. You will see a sample visualization of points inside a random cluster, and also the silhouette score for the clustering results.