Data Science Lab
Lab #4

Politecnico di Torino
October 30-31, 2019

Intro

In this laboratory, you will learn more about clustering techniques. You will first implement your own version of the K-means algorithm. Then you will apply it to different datasets and evaluate the performance achieved.

1 Preliminary steps

1.1 NumPy

Make sure you have the NumPy library installed, its use is strongly recommended for this laboratory.

NumPy is the fundamental package for scientific computing with Python. You can read more about it on the official documentation.

1.2 Datasets

For this lab, two different datasets will be used. Here, you will learn more about them and how to retrieve them. They are both synthetic datasets, i.e. they contain data that has been generated by hand to match a specific scientific need. Synthetic data are often used to test machine learning algorithms under conditions that are unlikely to occur with real-world data.

Both of the datasets have been used in Fränti and Sieranoja 2018. You can find them and many more (eventually, more complex) on their official web page.

1.2.1 Synthetic 2-D Gaussian clusters

The first dataset contains several two-dimensional points distributed among separated globular clusters.

Globular clusters are also known as Gaussian clusters. In a Gaussian cluster, points coordinates follow a Gaussian distribution that share the same mean. In the context of probability theory, a Gaussian cluster is a specific case of a mixture of probability distributions, where every component follows a normal distribution. Remember that, given a mean μ and a standard deviation σ, the probability density function of a normal variable is:

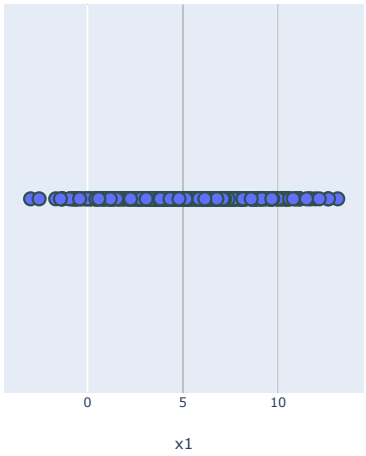
$$N(x;\mu;\sigma) = \sqrt{\frac{1}{2\pi\sigma^2}} exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)$$

A comprehensive description of mixtures of distributions can be found in the Probability and Information Theory chapter of Goodfellow, Bengio, and Courville 2016. Figure 1 shows some examples of N-dimensional clusters.

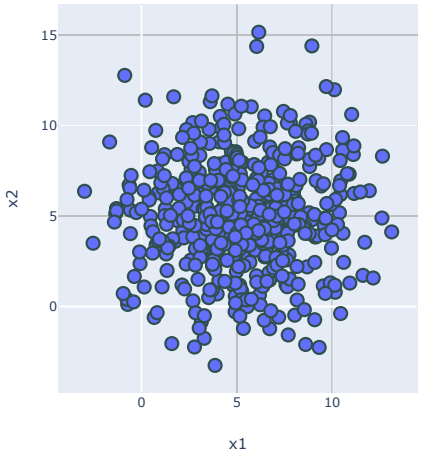
For your convenience, we parsed the original dataset and saved it as a `txt` file. You can download it at:

`https://raw.githubusercontent.com/dbdmg/data-science-lab/master/datasets/2D_gauss_clusters.txt`

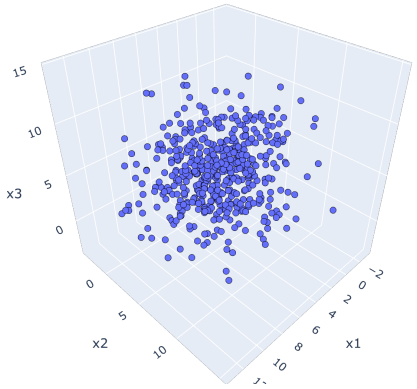
Each of the 5,000 rows contains the x and y coordinates of a single point. These points are grouped in the Euclidean space in 15 different globular clusters.



(a) 1-D Gaussian cluster



(b) 2-D Gaussian cluster



(c) 3-D Gaussian cluster

Figure 1: Gaussian clusters obtained from $\{x1, x2, x3\} \sim N(x; 5; 3)$

1.2.2 Chameleon

This synthetic dataset was originally introduced in Karypis, Han, and Kumar 1999. It contains again two-dimensional data points distributed along interleaved clusters with different shapes. For your convenience, we parsed the original dataset and saved it as a `txt` file. You can download it at:

`https://raw.githubusercontent.com/dbdmg/data-science-lab/master/datasets/chameleon_clusters.txt`

Each of the 8,000 rows contains the x and y coordinates of a single point. These points are grouped in the Euclidean space in 6 different clusters.

2 Exercises

Note that exercises marked with a (*) are optional, you should focus on completing the other ones first.

2.1 K-means design and implementation

This exercise will focus on the K-means clustering technique. You will implement your own version of the algorithm and then you will test it on two different datasets.

1. Load the synthetic 2-D dataset containing Gaussian clusters.

   Info: you can use the `numpy.loadtxt` method to easily parse and store the dataset into a NumPy array.

2. Plot the data points as a scatter chart using the Matplotlib library. At first sight, you should see 15 different globular clusters. Given this distribution, which could be the most suitable clustering technique among the ones that you known? Why?

3. Focus now on the K-means technique. You can find a thorough explanation of the algorithm on the course slides on clustering (slides 16-22).

   From next week laboratory you will start using the `scikit-learn` package. Many of its functionalities are exposed via an object-oriented interface. With this paradigm in mind, implement now the K-means algorithm and expose it as a Python class. The bare skeleton of your class should look like this (you are free to add as many functions as you want):

   ```
   class KMeans:
       def __init__(self, n_clusters, max_iter=100):
           self.n_clusters = n_clusters
           self.max_iter = max_iter
           self.centroids = None
           self.labels = None

       def fit_predict(self, X):
           """Run the K-means clustering on X.

           :param X: input data points, array, shape = (N,C).
           :return: labels : array, shape = N.
           """
           pass

       def dump_to_file(self, filename):
           """Dump the evaluated labels to a CSV file."""
           pass
   ```

   The core method is `fit_predict`. It should execute the K-means with `K=self.n_clusters` finding the centroids and assigning the label to each data point. Note that the class is intended to be stateful: it must keep track of the obtained centroids and labels. The `max_iter` parameter should be used to specify how many iterations are allowed for the main loop of the algorithm.

4. As you can see from the provided class, you have to include also a method to dump the obtained labels to a CSV file. If the input data are:

   ```
   x1,x2
   845753,636607
   812954,643720
   868217,609046
   ...
   ```

you should create an output file like this:

```
Id,ClusterId
0,0
1,0
2,1
...
```

We will soon use an online submission platform to evaluate you job. Most of the time this requires to save your results into a specific format, so pay enough attention to this point.

5. Once you get to a fully functional version of your class, load also the Chameleon data (see Section 1.2.2) and run the K-means algorithm on both the datasets. Feel free to run multiple times the algorithm varying `n_clusters` (i.e. K). For each run, you will get two list of labels. In the next points you will try to inspect the results you obtained.

6. (*) Since you are working with two-dimensional data, you are able to visualize them and inspect the results after an algorithm run. Specifically, using the Matplotlib package, create a figure containing a scatter plot with you original data points. Then, draw onto the figure itself the centroids you obtained with a different marker and color (e.g. `marker="*", color="red"`). Create this chart for both your datasets. Where are your centroids located? Based on this, can you assess which clustering was successful and which not (or, in other words, which dataset the K-means was more efficient on)? Can you figure why?

7. (*) Let's improve your K-means class with an additional visualization tool. Add two parameters to your `fit_predict` method: `plot_clusters=False` and `plot_step=5`. If `plot_clusters` is set to True, the intermediate positions of your centroids should be displayed every `plot_step` loop iterations. Also, choose a different color for each cluster and assign it to every point belonging to it. Figure 2 shows one of such intermediate charts.

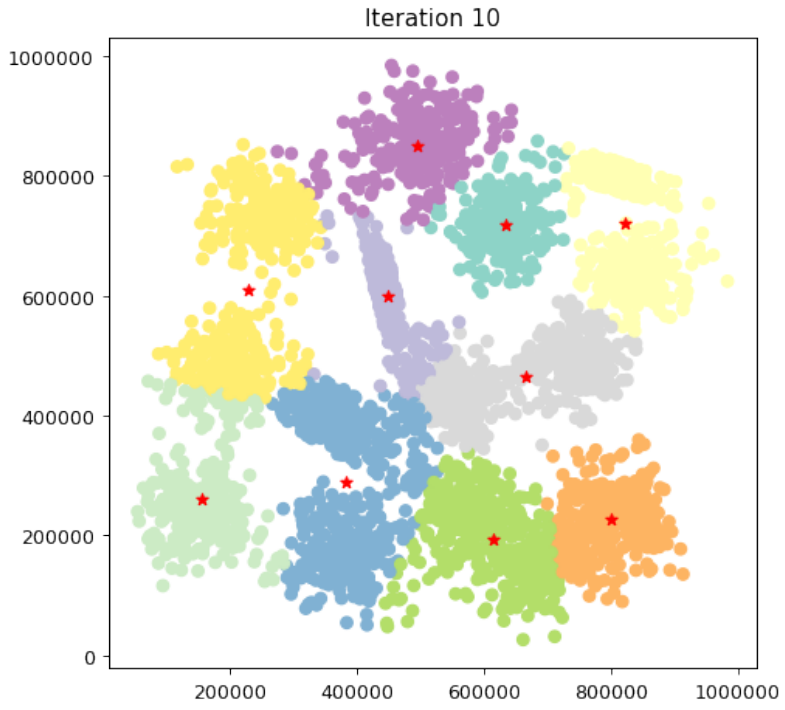


Figure 2: 2-D Gaussian synthetic dataset: clusters found at the 10*th* iteration with k=10.

2.2 Evaluate clustering performance

In this exercise you will evaluate the clustering performance of your K-means implementation. To do so, you will exploit the Silhouette measure. You can read more about it on Wikipedia.

1. Design and implement two different functions to compute the Silhouette. One should compute the metric for each sample, the other should compute the average silhouette score (you will find several evaluation scores like this in scikit-learn). You can start from this structure:

```
def silhouette_samples(X, labels):
    """Evaluate the silhouette for each point and return them as a list.

    :param X: input data points, array, shape = (N,C).
    :param labels: the list of cluster labels, shape = N.
    :return: silhouette : array, shape = N
    """
    pass

def silhouette_score(X, labels):
    """Evaluate the silhouette for each point and return the mean.

    :param X: input data points, array, shape = (N,C).
    :param labels: the list of cluster labels, shape = N.
    :return: silhouette : float
    """
    pass
```

   Note that the array `labels` is the one you generated in the previous exercise, point 5.

2. Implement a function to plot the silhouette values sorted in ascending order. This kind of chart is particularly useful to inspect the overall performance of a clustering technique. In an ideal case, the curve is heavily shifted towards the value 1 on the y-axis, i.e. most of the points have been assigned coherently. Create the chart for both your datasets and discuss the results. Do these plots match the clustering performance that you expected looking at the scatter plots from the previous exercise, point 6? Again, can you identify the dataset that best fits the K-Mean needs?

3. (*) Until now, knowing in advance the number of clusters present in our datasets (either via specifications or by visualizing them) has made us able to chose the number K accordingly. This is typically not the case in real situations, either because there are more than two or three dimensions in your data or worse, a clear cluster subdivision does not exist at all. Turning the problem around, the most common task becomes choosing the K value that leads to the best possible clustering division. For what concerns the silhouette measure, the higher is the average silhouette the best are the intra-cluster cohesion and the inter-cluster separation.

   Warning: the silhouette, like several other indices, is based on a geometrical distance. Maximizing such indices assures the best *geometrical* solution. However, the semantical meaning of the clusters could not be reflected. Can you imagine a way to address the problem?

   Define a function that, given a set of K values and a dataset, plots a line chart with the values of the average silhouette obtained for each K. By simply looking at it, you should be able to identify the best K for the task. Is it the one that you expected beforehand? Can you spot a trend (e.g. the higher the K value the higher the average silhouette)? Discuss this especially for the Chameleon dataset.

References

[1] Pasi Fränti and Sami Sieranoja. *K-means properties on six clustering benchmark datasets*. 2018. URL: http://cs.uef.fi/sipu/datasets/.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[3] George Karypis, Eui-Hong Sam Han, and Vipin Kumar. "Chameleon: Hierarchical clustering using dynamic modeling". In: *Computer* 8 (1999), pp. 68–75.