

Data Science and AI for industrial systems HOn#2

Politecnico di Torino

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. In the last part of the laboratory you will work on textual data, a domain where the data preparation phase is crucial to any subsequent task. Specifically, you will try to detect topics out of a set of real-world news data. Then, you will describe each cluster through frequent itemset mining.

1 Preliminary steps

1.1 NumPy

As you may have already understood, the Python language comes with many handy functions and third party libraries that you need to master to avoid boilerplate code. In many cases, you should leverage them to focus on the analysis process rather than its implementation.

That said, we listed a series of libraries you can make use of in this laboratory:

- [NumPy](#)
- [scikit-learn](#)
- [SciPy](#)

1.2 Datasets

For this lab, two different datasets will be used. Here, you will learn more about them and how to retrieve them. The firsts two are 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 datasets have been used in Fränti and Sieranoja [2018](#).

You can find them and many more (eventually, more complex) on their official web page.

The third dataset instead is a real-world dataset containing textual news belonging to different topics.

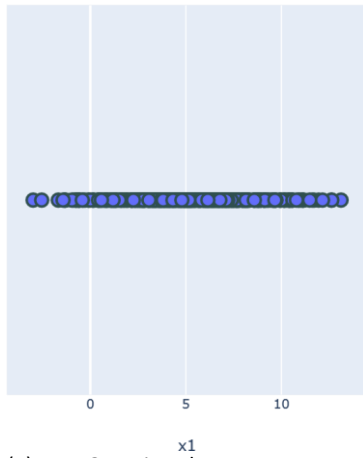
1.2.1 Synthetic 2-D Gaussian clusters

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

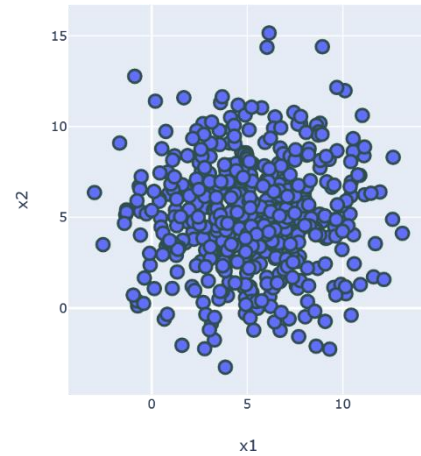
Globular clusters are also known as Gaussian clusters. In a Gaussian cluster, points coordinates follow a Gaussian distribution that share the same mean. In the context of probability theory, a Gaussian cluster is a specific case of a mixture of probability distributions, where every component follows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

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

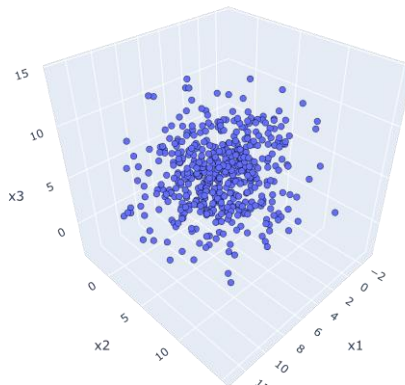
A comprehensive description of mixtures of distributions can be found in the [Probability and Information Theory chapter](#) of Goodfellow, Bengio, and Courville 2016. Figure 1 shows some examples of N -dimensional clusters.



(a) 1-D Gaussian cluster



(b) 2-D Gaussian cluster



(c) 3-D Gaussian cluster

Figure 1: Gaussian clusters obtained from $\{x_1, x_2, x_3\} \sim N(x; 5; 3)$

For your convenience, we generated a 2D 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.

1.2.2 Chameleon

This synthetic dataset was originally introduced in Karypis, Han, and Kumar 1999. It contains again twodimensional 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

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 the two synthetic datasets.

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

Later in this laboratory you will use 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. **Try to solve this exercise by using numpy APIs.** 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
```

The core method is `fit_predict`. It should execute the K-means with $K=\text{self.n_clusters}$ 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. 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.
5. 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?
6. 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.

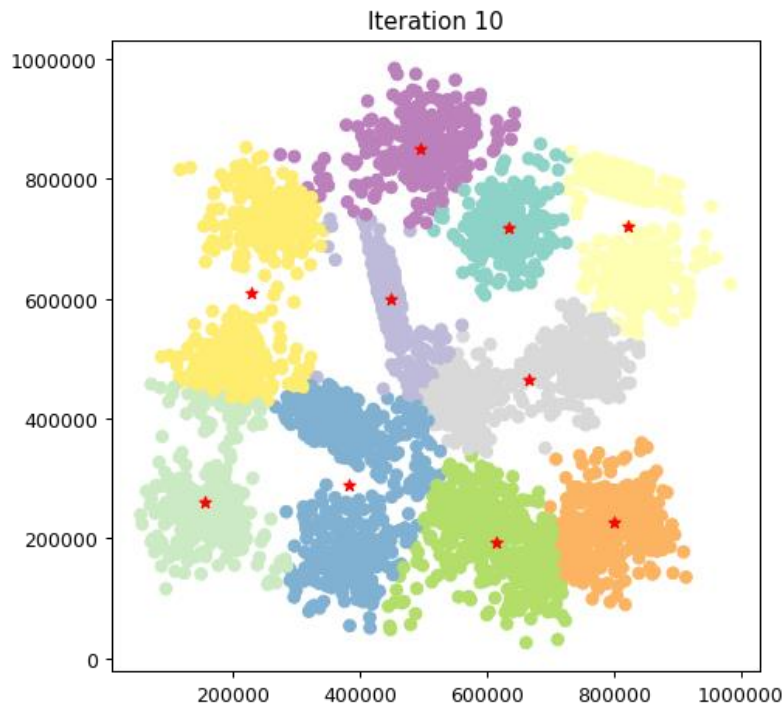


Figure 2: 2-D Gaussian synthetic dataset: clusters found at the 10th iteration with k=10.

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](https://en.wikipedia.org/wiki/Silhouette_(clustering)).

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). **Try to solve this exercise by using numpy APIs.** 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 4.

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 5? Again, can you identify on which dataset the K-Means is performing better?
3. Until now, knowing in advance the number of clusters included in our datasets (either via specifications or by visualizing them) has made us able to choose 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 better 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.

Appendix

Notions on linear transformations and dimensionality reduction

In many real cases, your data comes with a large number of features. However, there is often a good chance that some of them are uninformative or redundant and have the only consequence of making your analysis harder. The simplest example could be features that are linearly dependent with each other (e.g. a feature that describes the same information with different, yet correlated units, like degrees Celsius and Fahrenheit).

One additional detail can be addressed in your preprocessing step, other than the dimensionality reduction. There might be cases where the distribution of your data has hidden underlying dynamics that could be enhanced by choosing different features (i.e. dimensions). Figure 3 shows several points distributed in a Gaussian cluster (see Laboratory 4). Let's make now an assumption: quantitatively we assess that directions with largest variances in our space contain the dynamics of interest. In Figure 3 the direction with the largest variance is not (1,0) nor (0,1), but the direction along the long axis of the cluster.

As you will soon learn, in the known literature there is a proven method that addresses the aforementioned problems: the *Principal Component Analysis* (PCA). This technique frames the problem as a linear change of basis. The final basis vector are commonly termed *principal components*. Let's qualitatively understand why and how it is made by means of a few algebraic notions.

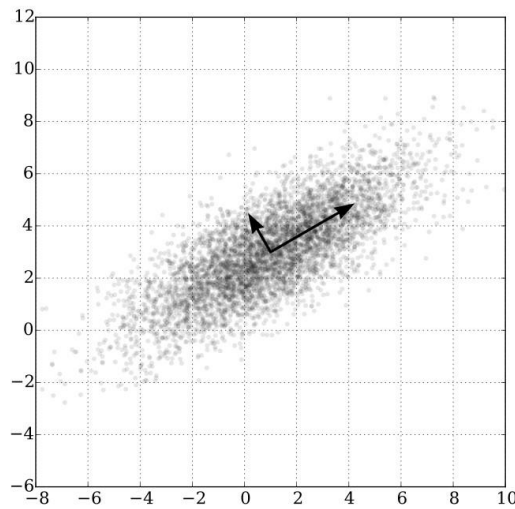


Figure 3: Points distributed in a Gaussian cluster with mean (1,3) and standard deviation 3. Source: [Wikipedia](#)

In PCA we assume that there exist a more meaningful basis to re-express our data. The hope is that this new basis will filter out the noise and reveal hidden dynamics. Following the assumption presented beforehand, the new basis must align the directions with the highest variance. Also, the change of basis follows another strict, yet powerful assumption: it is assumed that the new basis is a linear combination of the original one (studies expanded on this to non linear domains). In Figure 3 PCA would likely identify a new basis in the directions of the two black arrows.

In other words, if we call \mathbf{X} the original set of data, in PCA we are interested in finding a matrix \mathbf{P} that stretches and rotates our initial space to obtain a more convenient representation \mathbf{Y} :

$$\mathbf{P}\mathbf{X} = \mathbf{Y} \quad (1)$$

Now that foundations are laid, we know that we are looking for a new basis that highlights the inner dynamics and we assume that the change of basis can be achieved with a simple linear transformation. This linearity assumption let us solve analytically the problem with matrix decomposition techniques. Even if the simpler eigendecomposition can be used, the state-of-the-art solution is obtained through the *Singular Value Decomposition* (SVD).

Many, many theoretical and technical details have been left behind in this short summary. If you are willing to learn more, you can find a thorough tutorial about PCA, SVD and their relationship in Shlens 2014.

The use of the PCA algorithm via SVD decomposition in Python is straightforward. The following lines show how you can apply the change of basis transforming your data.

```
from sklearn.decomposition import TruncatedSVD

# X: np.array, shape (1000, 20)
svd = TruncatedSVD(n_components=5,
random_state=42)
red_X = svd.fit_transform(X) # red_X will be: np.array, shape (1000, 5)
```

Note that the TruncatedSVD class lets you choose how many top-principal components to retain (they are ranked by explained variance). Doing so, you will be applying the dimensionality reduction at the same time.

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. *DeepLearning*. <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.
- [4] Ken Lang. “Newsweeder: Learning to filter netnews”. In: *Proceedings of the Twelfth International Conference on Machine Learning*. 1995, pp. 331–339.
- [5] Jonathon Shlens. “A Tutorial on Principal Component Analysis”. In: *CoRR* abs/1404.1100 (2014). arXiv: 1404.1100. url: <http://arxiv.org/abs/1404.1100>.