# TSIA2: Pattern Recognition

***Electronic Department - option TSI, 3rd year***

Rémi Giraud
*remi.giraud@enseirb-matmeca.fr*
2026-2027

# TP Unsupervised Classification

## 1 Introduction

The objective of this practical is to implement several popular unsupervised classification techniques discussed in class, particularly hierarchical agglomerative clustering methods as well as the $K$-means method.

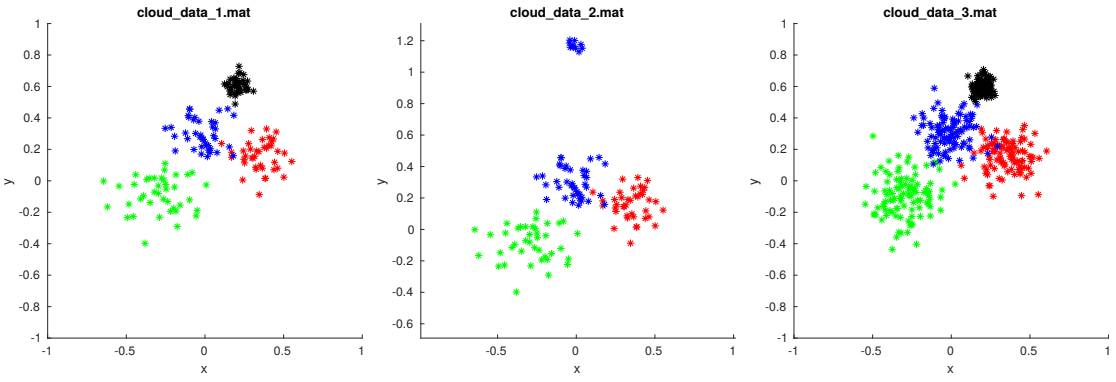
### 1.1 To do

Complete the provided Python codes to implement the hierarchical agglomerative clustering algorithm and then the $K$-means method and its variants. These algorithms will be applied to 2D point cloud data. The intra-class, inter-class, and total inertia measures will be calculated at each step. A final part will be dedicated to over-segmentation using the $K$-means method, which can then be applied to a 3D LiDAR point cloud.

### 1.2 Data

#### 1.2.1 2D Point clouds

Three sets of 2D spatial data are provided in files *cloud_data_#{1-3}.npy* of size $n\times3$. The data are stored such as $data = (y, x, class)$ * and are represented as point cloud in Figure 1.




$n = 160$, $K = 4$ $n = 130$, $K = 3$ $n = 400$, $K = 4$

Figure 1 : 2D data point clouds

### 1.2.2 Bonus: 3D LiDAR point cloud

In section 3.3, we will consider a point cloud from a LiDAR acquisition (remote sensing using infrared light pulses) from the SemanticKITTI dataset[1] (Figure 2). This point cloud captures a scene in the context of autonomous driving, and the classes associated with the different points represent the semantics of objects (road, pedestrian, tree, etc.). The data is stored in the file *point_cloud_LiDAR.npy* as an array *data* of size $n \times 5$ containing in order the information for $n$ points: positions $X$, $Y$, $Z$, reflectance $R$, and ground truth labels $L$. The folder *semantic-kitti-api-master* contains functions for efficiently visualizing the point cloud.

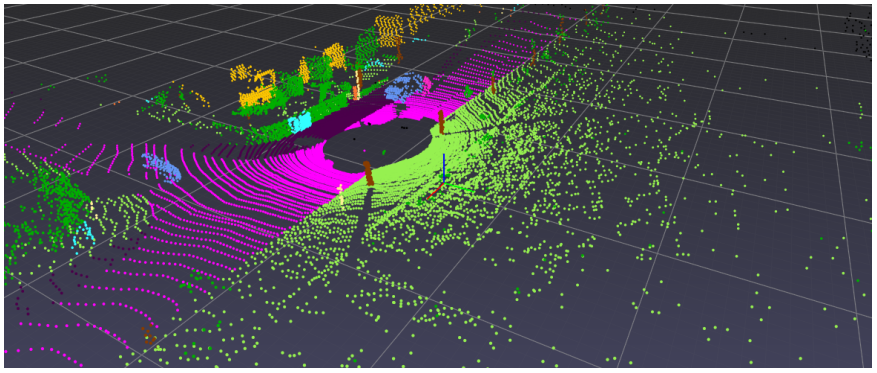


Figure 2 : 2D point clouds

# 2 Hierarchical Ascending Classification

1. Implement the hierarchical agglomerative clustering algorithm using the Average Link strategy (comparison of the centroids of each class).

   As a reminder: when two classes merge, by convention, we keep the smallest label and the centroid is recalculated over all the points that make up the new class.

   We propose to use the following data structures:

   - Initial data *data* of size $n \times p$ (here $n = 160$ and $p = 2$ for *cloud_data_1.npy*)
   - Classification table `classif_exp` of size $n \times n$ to modify at each iteration:
     (here at iteration 1, clusters 1 and 3 are merged)

| Itération | 1 | 2 | 3 | ... | i | ... | $n$ |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | ... | i | ... | $n$ |
| 1 | 1 | 2 | 1 | ... | i | ... | $n$ |
| ... | | | | | | | |
| $n$ | 1 | 1 | 1 | ... | 1 | ... | 1 |

   - Barycenters $G_i$ updated for each new association (if class 1 and 3 are merged, we only have to update the barycenter $G_i$ corresponding to the chosen label between the two (always the lowest one for instance), since the other disappear from the rest of the classification table.

2. Bonus: To implement and compare the three aggregation strategies (simple, complete, Ward).

[1] `http://www.semantic-kitti.org/`

# 3 $K$-means

## 3.1 General algorithm

First, we consider 2D point clouds (1.2.1).

1. Implement the k-means algorithm for an arbitrary number of classes $K$.

   The centers will first be randomly chosen within the data space.

   The Euclidean distance $L_2$ will be used as norm.

   What solutions can be considered to automatically stop the algorithm?

2. Visualize the evolution of the algorithm on the "cloud_data_1.mat" point cloud data for $K = 4$.

   Is it possible to have a variable number of classes as an output of the algorithm?

## 3.2 Variants

6. Observe the difficulty of getting a good clustering on the point cloud *cloud_data_2.npy*.

   Implement the $K$-medoids method (where the centers are the points with the lowest average distance to other points in the same class).

   Implement the $K$-means++ initialization optimization method.

   Visually verify that the algorithm converges faster.

   Compare the time costs of the methods on the point cloud *cloud_data_3.npy*

## 3.3 Bonus: Over-segmentation

9. Perform clustering with a number of classes $K >> 4$. Can it be evaluated in the same way as before?

   We are now in the context of *over-segmentation*. Achievable Segmentation Accuracy (ASA) is a relevant metric to assess the consistency of the clustering with the ground truth objects:

$$ASA(\mathcal{S}, \mathcal{G}) = \frac{1}{\sum\limits_{S_k \in \mathcal{S}} |S_k|} \sum_{S_k \in \mathcal{S}} \max_{G_i \in \mathcal{G}} |S_k \cap G_i| \tag{1}$$

   where $\mathcal{S}$ is the estimated clustering into $K$ classes and $\mathcal{G}$ the groundtruth.

10. Bonus: Let us now consider the 3D LiDAR point cloud.

    Perfom a clustering into $K = K_LiDAR$ classes. Do the results seem good? What is the problem?

11. Increase the number of classes $K >> K_LiDAR$ and observe the metric evolution.