Master's Degree in Mechatronic Engineering

Politecnico di Torino

Master's Degree Thesis

UNSUPERVISED MACHINE LEARNING ALGORITHMS FOR EDGE NOVELTY DETECTION

Supervisors

Prof. Marcello CHIABERGE

Dott. Umberto ALBERTIN

Dott. Gianluca DARA

Candidate

Ariel PRIARONE

03 2024

Acknowledgements

I would like to thank the PoliTO Interdepartmental Centre for Service Robotics (PIC4Ser) for giving me the opportunity to work on this project. The guidance and infrastructure provided by the centre have been invaluable during the development of this work.

To my parents, who have given me everything.
Thank you for always making me believe that anything is possible.
Ai miei genitori, che mi hanno dato tutto.
Grazie per avermi sempre fatto credere che tutto sia possibile.

Ariel

Table of Contents

List of Tables

List of Figures

Chapter 1

Unsupervised Learning

In the previous **??**, an overview of the most common supervised learning algorithms has been provided, but all these techniques require a labelled dataset. As anticipated in the introduction, most of the time a model of the machine to be monitored is not available. Furthermore, usually a prior knowledge of the behaviour of the machine in the *healthy* or a *faulty* state is not available either. Even in the best-case scenario, where some data collection has been done, the data will be unlabeled.

To address this scenario, two approaches are possible: either label the data or use an unsupervised learning algorithm. The former would be tedious and time-consuming in the case the dataset contains both healthy and faulty data. This is because the faulty data would have to be labelled by hand. If the dataset contains only healthy data, it would be trivial to automatically label all the instances as healthy and use a supervised learning algorithm, but this would be a stretch of the definition of supervised learning. The latter is a more linear approach since unsupervised learning algorithms are designed to work with unlabeled data.

The most common unsupervised task is dimensionality reduction [1, p. 260] but, in this thesis, the main focus will be on novelty detection, fault detection and predictive maintenance, so the considered UML algorithms will be clustering and density estimation.

Clustering Clustering is the task of grouping together similar instances. The definition of *similar* depends on the algorithm used. The most common algorithms are k-means and DBSCAN. The former is a centroid-based algorithm, it is fast to evaluate a new instance and produce a lightweight model but performs poorly in some conditions that will be described in detail. The latter is a density-based algorithm, it performs better in the condition where k-means fails but has the drawbacks of being much slower to evaluate a new instance, and to perform novelty detection, the DBSCAN implementation would have to keep all the train data in memory. Both will be described in detail in the following **section 1.1** and

section 1.2.

Gaussian mixture models The second approach to novelty detection is the use of Gaussian Mixture Models (GMM). This approach is based on the assumption that the data is generated from a mixture of several Gaussian distributions with unknown parameters [1, p. 283]. Then the distribution model can be used for novelty detection. This approach will be described in detail in the following **section 1.3**.

Other approaches At the end of the chapter, some other approaches will be briefly described and tested on the same dataset used for demonstrating clustering and GMM. These approaches are: iForest, LOF and ν-SVM. The first two are based on the assumption that outliers are instances that are isolated from the rest of the data, while the latter is based on a kernelized SVM algorithm.

1.1 K-means

This problem is called k-means because it aims to describe the "clustering" by separating the data into clusters ($\boldsymbol{\mathcal{C}}$), and define each cluster with its mean ($\boldsymbol{c}$). Note that the mean of the cluster, from a physical point of view, is the centre of mass of the cluster itself as if it is composed of unitary point masses located at the positions of the data points. The mean is not necessarily a point belonging to the cluster.

Let's assume to have extrapolated F features from each of our signals, to produce a set $\mathbf{S}$ of n snapshots $\boldsymbol{\mathcal{S}}_i, i \in [1, n], \boldsymbol{\mathcal{S}}_i \in \mathbf{S}$ (every snapshot is a vector of features $\in \mathbb{R}^F$). The task is to define a set $\boldsymbol{\mathcal{C}}$ of k clusters ($k \leq n$) $\boldsymbol{\mathcal{C}}_i, i \in [1, k], \boldsymbol{\mathcal{C}}_i \in \boldsymbol{\mathcal{C}}$ that minimize the squared sum of the distances between the snapshots and the centroids $\boldsymbol{c}_i$ of the clusters they belong to. This is equivalent to finding the centroids that minimize the **variance** of the clusters themselves, so the problem can be formulated as in the **equation 1.1**.

$$\arg\min_{\boldsymbol{\mathcal{C}}} \sum_{i=1}^{k} \sum_{\boldsymbol{\mathcal{S}}_j \in \boldsymbol{\mathcal{C}}_i} \|\boldsymbol{\mathcal{S}}_j - \boldsymbol{c}_i\|^2 = \arg\min_{\boldsymbol{\mathcal{C}}} \sum_{i=1}^{k} |\boldsymbol{\mathcal{C}}_i| \operatorname{Var} \boldsymbol{\mathcal{C}}_i \tag{1.1}$$

Unfortunately, this problem is NP-hard, even for as little as $F = 2$ features considered [2], so it is not possible to guarantee to find the global optimum in a reasonable time.

Anyway, heuristic clustering algorithms were already developed in the 1950s. The first appearance of the term "K-means" was used in 1957 by MacQueen [3], and the algorithm settled to a "standard" version, published by Stuart Lloyd in 1982 [4] (but developed at Bell Labs in 1957).

Among all the unsupervised clustering algorithms, a survey from 2002 [5] stated that K-means "is by far the most popular clustering algorithm used in scientific and industrial applications". A more recent survey from 2019 [6], cited this algorithm as first in the group of four most popular algorithms.

Nowadays, the K-means algorithm is implemented in many libraries, such as `scikit-learn` for `Python`, and others for `C`, `R`, `MATLAB`, etc. However, the runtime performances vary widely depending on the implementation [7]. The problem of the algorithm returning a local minima instead of the global one is still present. Most implementations try to minimize the probability of returning this sub-optimal result by running the algorithm multiple times with different initializations and then selecting the best result, so the problem of getting a sub-optimal result is not a common issue in practice.

1.1.1 Training

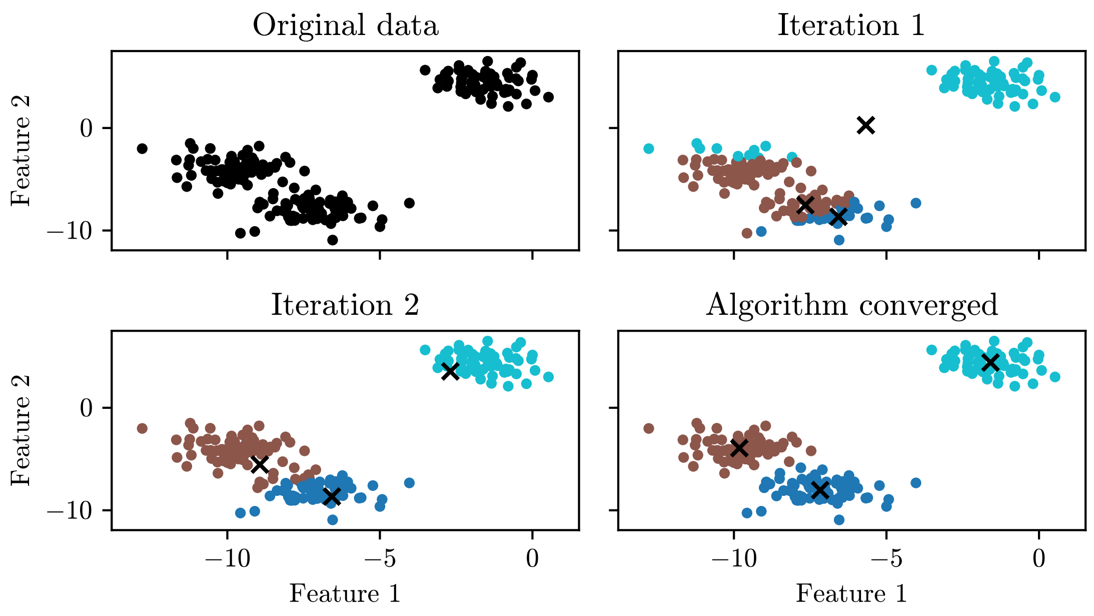


Figure 1.1: *K-means algorithm in the 2-dimensional space*

The naive k-means algorithm consists of a series of iterations. First, the centroids $\boldsymbol{c}_i$ are initialized randomly, then the snapshots are assigned to the nearest centroid, and finally, the centroids are updated as the mean of the snapshots assigned to them. These steps are repeated until the position of the centroids does not change anymore, or a defined maximum number of iterations is reached. This naive algorithm is summarized in the **algorithm 1**.

Algorithm 1 Training of the K-means model

```
1: function K-MEANS.TRAIN($\mathbf{S}, k$)
2:     ▷ $\mathbf{S}$ is the set of snapshots to be clustered
3:     ▷ $k$ is the number of clusters to be obtained
4:     $\boldsymbol{c}_i \leftarrow$ random initialization, $\forall i \in [1,k], \boldsymbol{c}_i \in$ Domain of $\mathbf{S}$
5:     repeat
6:         ▷ Every snapshot is assigned to the nearest centroid. Every centroid defines a cluster containing the assigned snapshots
7:         $\boldsymbol{\mathcal{C}}_i \leftarrow \left\{\boldsymbol{\mathcal{S}}_p : \|\boldsymbol{\mathcal{S}}_p - \boldsymbol{c}_i\|^2 \leq \|\boldsymbol{\mathcal{S}}_p - \boldsymbol{c}_j\|^2 \forall j \in [1,k]\right\} \forall i \in [1,k]$
8:         ▷ The centroids are updated as the mean of their snapshots
9:         $\boldsymbol{c}_i \leftarrow \frac{1}{|\boldsymbol{\mathcal{C}}_i|} \Sigma_{\boldsymbol{\mathcal{S}}_j \in \boldsymbol{\mathcal{C}}_i} \boldsymbol{\mathcal{S}}_j, \forall i \in [1,k],$    ▷ $|\boldsymbol{\mathcal{C}}_i|$ is the cluster size
10:    until All the centroids do not change anymore, or max iterations reached
11:    $r_i \leftarrow \max \|\boldsymbol{\mathcal{S}}_j - \boldsymbol{c}_i\|$, $\forall \boldsymbol{\mathcal{S}}_j \in \boldsymbol{\mathcal{C}}_i$, $\forall i \in [1,k]$
12:    return $\mathcal{M}_{\texttt{k-means}}$ ▷ The model contains the centroids $\boldsymbol{c}_i$, the radii $r_i$ of the clusters, and the labels of the snapshots
13: end function
```

As an example, we can consider $F = 2$ features, and generate some test points shaped like three separated clusters. In the **figure 1.1** are shown the original data, the first two iterations of the algorithm, and the final result. The K-means algorithm had $n = 200$ snapshots, and $k = 3$ clusters as input. The colours of the dots and the shaded areas represent the clusters and the decision boundaries. The centroids are represented as black crosses. The decision boundaries are a Voronoi tessellation of the space, and they are defined as the set of points that are equidistant from the centroids of two different clusters. The algorithm itself does not compute the boundaries, but it is useful to plot them for visualization purposes.

1.1.2 Variations of the K-means algorithm

Kmeans Finding the optimal solution to the k-means problem (**equation 1.1**), as said, is NP-hard. To address this problem, some other heuristics algorithms have been proposed in the last two decades. Simple implementations have time complexity $\mathcal{O}(n^2)$ [8]. This means that most algorithms do not scale well as the number n of snapshots increases. With respect to the number of clusters, the problem has a linear complexity $\mathcal{O}(k)$.

It is worth noticing that an exact solution to the problem has been published [9], with a time complexity $\mathcal{O}(n^{k \cdot F})$. This is still impractical for actual applications, as it is exponential with respect to both the number of clusters and the number of features F.

Lloyd's algorithm The classic Lloyd algorithm [4] has a complexity $\mathcal{O}(n \cdot k \cdot F)$.

Various improvements to Lloyd's algorithm Keeping the same basic idea, various modifications of the Lloyd algorithm have been proposed to improve the performances. For example, Kanugo [10] proposed a local search algorithm that has a complexity $\mathcal{O}(n^3)$. Another result by Malay [8] has a linear complexity $\mathcal{O}(n)$.

Another improvement has been developed by Elkan [11], by keeping track of the bounds from the instances (snapshots) and the centroids. This algorithm becomes convenient when the number k of clusters is large (≥ 20), and up to a dimensionality of $F = 1000$ features.

A variant of the Lloyd algorithm regarding both the speed of execution and the memory consumption has been proposed by Sculley [12]. This solution achieves a reduction of the execution time by orders of magnitude and enables performing the classification even for datasets that don't fit in the memory of the machine. This is achieved by using a *mini-batch* approach, where the centroids are updated after each batch of snapshots.

Concerning the problem of converging to a local minimum, the most common approach is to run the algorithm multiple times with different initializations and then select the best result, this is avoided by Reddy [13], by using the Voronoi tessellation of the hyperspace using the data points to generate the initialization for the centroid positions, this algorithm performs better in the sense that is less likely to get trapped in a local minimum.

K-means++ The last improved algorithm reported in this section has its own paragraph because it is the one used in this thesis. It was developed and named Kmeans++ by Arthur and Vassilvitskii in 2007 [14]. The difference from the Lloyd algorithm is only in the first initialization of the centroids $\boldsymbol{c}_i \forall i \in [1, n]$. In this case, instead of a random initialization for all the centroids, the first centroid $\boldsymbol{c}_1$ is chosen randomly from the snapshots, and then the other centroids are chosen from the remaining snapshots with a probability that depends on the distance of the candidate snapshot to the closest already chosen centroid. This approach is summarized in the **algorithm 2**.

For the development of the framework of this thesis, the K-means++ algorithm has been implemented in `Python`, using the `scikit-learn` library. The library function has been modified, adding a method that returns the radii of the clusters, this information is crucial for our scope, as it will be needed for evaluating if a new snapshot is a novelty, normal or fault, as it will be explained in the **subsection 1.1.6** and **subsection 1.1.7**.

Algorithm 2 K-means++ algorithm

```
1: function K-MEANS++.TRAIN($\mathbf{S}, k$)
2:     ▷ $\mathbf{S} = \{\boldsymbol{\mathcal{S}}_1, \boldsymbol{\mathcal{S}}_2, \dots, \boldsymbol{\mathcal{S}}_n\}$ is the set of snapshots to be clustered
3:     ▷ $k$ is the number of clusters to be obtained
4:     $\boldsymbol{c}_1 \leftarrow$ random initialization, $\boldsymbol{c}_1 \in \mathbf{S}$
5:     for $i \leftarrow 2$ to $k$ do
6:         ▷ $D(\boldsymbol{\mathcal{S}})$ is the distance of the snapshot $\boldsymbol{\mathcal{S}}$ from the closest centroid already chosen
7:         $c_i \leftarrow \boldsymbol{\mathcal{S}}' \in \mathbf{S}$ with probability $\frac{D(\boldsymbol{\mathcal{S}}')^2}{\Sigma_{\boldsymbol{\mathcal{S}} \in \mathbf{S}} D(\boldsymbol{\mathcal{S}})^2}$
8:     end for
9:     perform the Lloyd algorithm using the calculated $\boldsymbol{c}_i, \forall i \in [1, k]$ as initialization, get the model.
10:    return $\mathcal{M}_{\texttt{k-means}}$ ▷ The model contains the centroids $\boldsymbol{c}_i$, the radii $r_i$ of the clusters, and the labels of the snapshots
11: end function
```

1.1.3 Selecting the number of clusters

It is important to notice that, even being an *unsupervised* learning algorithm, the K-means algorithm needs to know the number of clusters k in advance. There are some methods to decide what is the best number of clusters, but they usually need to perform more iterations of the algorithm with different values of k, and then compare the results. This task is automatable so, during the training phase, the user can decide the number of clusters to be used, or leave the selection to the algorithm itself.

To compare the results of the different iterations, it is possible to use some metrics on the data and the centroids. The most common metrics are the *inertia* and the *silhouette score*, described in the following paragraphs.

Inertia The inertia metric measures the total (sum) distance of each point belonging to a cluster from the centroid of the cluster itself, as shown in the **equation 1.2**. This is called inertia because, in the physical sense, it is the sum of the moment of inertia of each cluster if all the snapshots were considered as point masses (with unitary mass). This analogy is useful to understand that the lower the inertia, the more compact the clusters are.

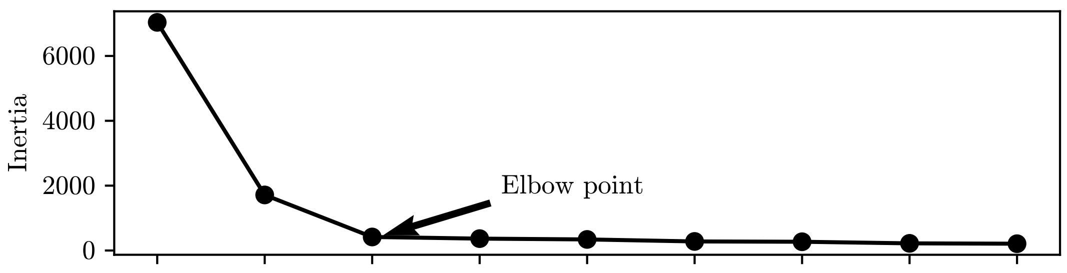
Let's span $k \in [1{,}9]$ and plot the inertia of the clusters for each value of k, on the previous dataset. The result is shown in the **figure 1.2a**. As expected, the inertia decreases as the number of clusters increases. This is not desirable behaviour, if the aim is selecting the number of clusters, the best guess is to select (by eye or with some automatism) the Pareto optimal point (POF) of the curve [15].

$$I = \sum_{i=1}^{k} \sum_{\boldsymbol{\mathcal{S}}_j \in \boldsymbol{\mathcal{C}}_i} \|\boldsymbol{\mathcal{S}}_j - \boldsymbol{c}_i\|^2 \tag{1.2}$$

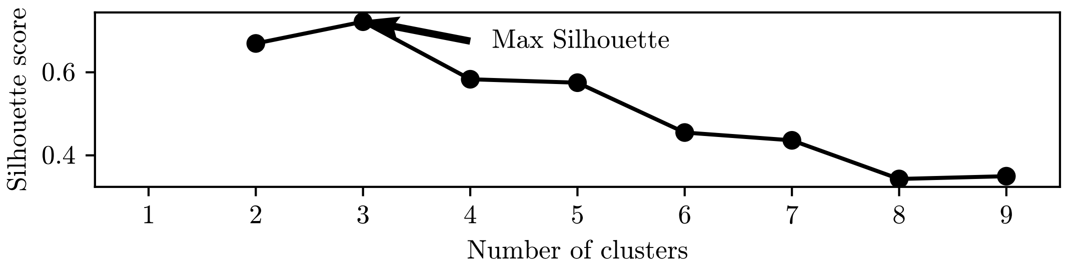
Silhouette score A better metric that can be used to select the number of clusters is the silhouette score. The silhouette score is defined for each snapshot as in **equation 1.3**, where a is the mean distance of the snapshot from the other snapshots in the same cluster, and b is the mean distance of the snapshot from the snapshots in the nearest cluster. The resulting silhouette S_i of a snapshot $\boldsymbol{\mathcal{S}}_i$ is a scalar: $S_i \in [-1{,}1]$. The three relevant cases are:

- a value close to 1 means that the snapshot is far inside its own cluster and far from snapshots of other clusters;

- a value close to 0 means that the snapshot is on the boundary between two clusters;



(a) *Inertia of the clusters for different values of k*



(b) *Silhouette score of the clusters for different values of k*

Figure 1.2: *Metrics for selecting the number of clusters*

- a value close to -1 means that the snapshot is far from its own cluster and close to another cluster, so it may have been misassigned.

$$S_i = \frac{b_i - a_i}{\max(a_i, b_i)} \tag{1.3}$$

At this point, the global silhouette score S_g can be computed as the mean of the silhouette scores of all the snapshots (**equation 1.4**). The global silhouette score, for the same example dataset, is shown as a function of the number of clusters k in the **figure 1.2b**. Note that this time $k \in [2,9]$, because the silhouette score is not defined for a single cluster.

In this case, the best value for k is $k = 3$, because it is the value that maximizes the silhouette score. This approach is simpler and easier to automate than the inertia one.

$$S_g = \frac{1}{n}\sum_{i=1}^{n} S_i \tag{1.4}$$

1.1.4 Assignation of the new instance to a cluster

The procedure for assigning the new snapshot $\boldsymbol{\mathcal{S}}_n$ to a cluster is quite simple, it is sufficient to compute the distance between $\boldsymbol{\mathcal{S}}_n$ and the centroids $\boldsymbol{c}_m$, $\forall m \in [1, \dots, k]$. The distance is defined as the l^2-norm in the feature space, it can be computed using the **equation 1.5**, and assign $\boldsymbol{\mathcal{S}}_n$ to the cluster with the minimum distance.

$$\boldsymbol{d}_{n,m} = ||\boldsymbol{\mathcal{S}}_{n,f} - \boldsymbol{c}_{m,f}||_2 = \sqrt{\sum_{f=1}^{F} (\boldsymbol{\mathcal{S}}_{n,f} - \boldsymbol{c}_{m,f})^2} \tag{1.5}$$

1.1.5 Evaluation of a new instance

At this point, with a model trained on the data, a generic nth new snapshot instance $\boldsymbol{\mathcal{S}}_n$ can be evaluated using the K-means algorithm. From a geometric point of view, the snapshot $\boldsymbol{\mathcal{S}}_n$ is a point in the F-dimensional space, where F is the number of features used to train the model.

For demonstration purposes, in this section, since it is still feasible to show 3D plots, it is considered an example with $F = 3$ features.

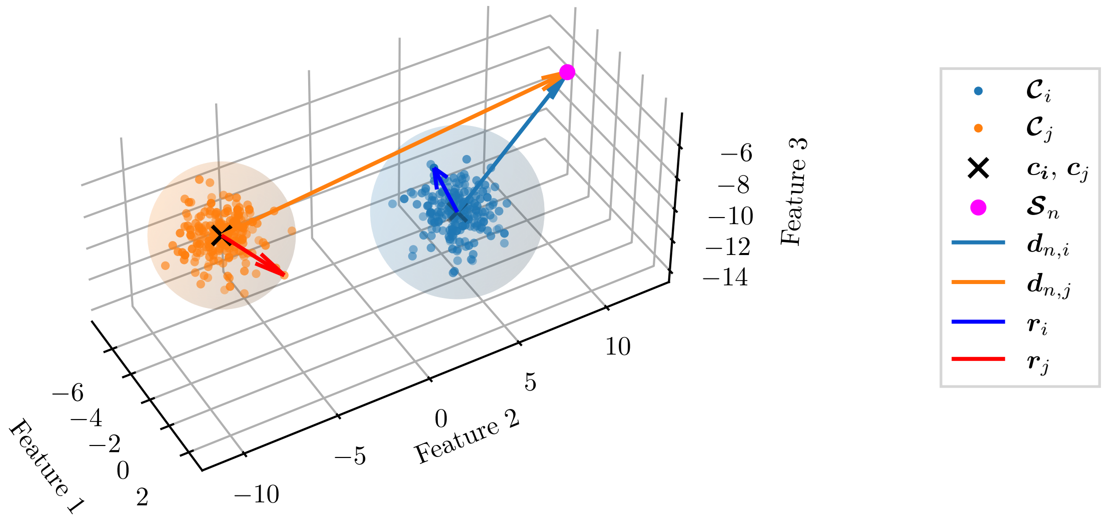


Figure 1.3: *Cluster model in the 3-dimensional space, with new snapshot* $\boldsymbol{\mathcal{S}}_n$

In the **figure 1.3**, the training data are represented in the 3-dimensional space, where the axis are the features used to train the model. The K-means model has been ideally trained with an arbitrary number k of clusters but, for display purposes, only two clusters ($\boldsymbol{\mathcal{C}}_i$ and $\boldsymbol{\mathcal{C}}_j$) are plotted.

The entities shown in the **figure 1.3** are:

- $\boldsymbol{c}_{i(j)}$ is the centroid of the ith (jth) cluster;
- $\boldsymbol{r}_{i(j)}$ is the radius of the ith (jth) cluster, it is defined as the distance between the centroid $\boldsymbol{c}_{i(j)}$ and the farthest point belonging to the cluster itself;
- $\boldsymbol{\mathcal{C}}_{i(j)}$ is the set of training snapshots belonging to the ith (jth) cluster, it has a centroid $\boldsymbol{c}_{i(j)}$ and a radius $\boldsymbol{r}_{i(j)}$;
- $\boldsymbol{\mathcal{S}}_n$ is the new snapshot to be evaluated;
- $\boldsymbol{d}_{n,i}$ is the vector between $\boldsymbol{\mathcal{S}}_n$ and $\boldsymbol{c}_i$;
- $\boldsymbol{d}_{n,j}$ is the vector between $\boldsymbol{\mathcal{S}}_n$ and $\boldsymbol{c}_j$;
- the semi-transparent spheres represent the cluster sizes, the radius of the spheres is the radius of the cluster itself, and the centre is the centroid of the cluster;

1.1.6 Metric for the new instance evaluation

Once the new snapshot $\boldsymbol{\mathcal{S}}_n$ has been assigned to the right cluster $\boldsymbol{\mathcal{C}}_i$ using **equation 1.5**, some kind of measure (a.k.a. metric) linked to how novel this snapshot is needs to be computed. In this document, this measure, referred to the n-th cluster, will be called e_n, in order to remind some sort of error, even if it is not an error in the strict sense. The first simple approach used in this project is computing the difference between the distance of $\boldsymbol{\mathcal{S}}_n$ from the centroid $\boldsymbol{c}_i$ and the radius $\boldsymbol{r}_i$ of the cluster itself. With this approach, the measure defined in the **equation 1.6** is relative to the current snapshot, so it is possible to use that as a novelty measure.

$$e_n = ||\boldsymbol{d}_{n,i}||_2 - ||\boldsymbol{r}_i||_2, \text{ where } i \text{ is the of the assigned cluster} \tag{1.6}$$

Few consideration about the result of the **equation 1.6**:

- if $e_n > 0$, the new snapshot $\boldsymbol{\mathcal{S}}_n$ is outside the sphere of radius $\boldsymbol{r}_i$ centred in $\boldsymbol{c}_i$, so it is probably a novel snapshot;
- if $e_n < 0$, the new snapshot $\boldsymbol{\mathcal{S}}_n$ is inside the sphere of radius $\boldsymbol{r}_i$, so it is probably a normal snapshot. In this case, it is worth noticing that this assumption is reasonable only if the shape of the point cloud resembles a sphere, otherwise, the radius $\boldsymbol{r}_i$ is not a good measure of the cluster size, and use it for novelty

detection would not be reasonable. **This emphasizes the importance of the standardization procedure applied to the features before the training phase**;

Using this metric it is possible to define as *novelty* all the snapshots with $e_n > 0$ and as *normal* all the snapshots with $e_n < 0$. This approach is not very robust because s snapshot that is even slightly outside the sphere of radius $\boldsymbol{r}_i$ will be considered a novelty, but since the sphere is tuned the training *measured* data, that have an aleatory component, this approach will probably detect some novelty even in normal snapshots.

1.1.7 Introducing a threshold for the metric evaluation

In order to improve the robustness of the novelty detection algorithm, it is possible to define a threshold t_i for each cluster $\boldsymbol{\mathcal{C}}_i$ and use it to detect if a snapshot is a novelty or not. Once the threshold t_i is defined, the detection of the novelty can be triggered by the condition $e_n > t_i$.

At this point, the problem is that the user would have to define a threshold for each cluster, and this is not a trivial task. This is because it is likely that the clusters have different sizes, and so one threshold for all the clusters would be more conservative for the smaller clusters and less conservative for the bigger ones. Moreover, most of the times, the clusters' shape and size will not have a physical meaning, and the act of manually defining a threshold for each cluster would go against our goal of designing a fully unsupervised framework.

To address this problem, it is possible to change the definition of the metric itself, so that is not dependent on the cluster size. This can be done by normalizing the already defined metric e_n with the radius $\boldsymbol{r}_i$ of the cluster itself, as shown in the **equation 1.7**. In this way, t_i can be defined as a percentage of the cluster size, so that the user can define a single threshold for all the clusters, and selecting the number to assign to t_i has a more intuitive meaning. From now on if not otherwise specified, the metric e_n will be this normalized version. Obviously, the metric can be easily displayed as a percentage: $e_{n,\%} = e_n \cdot 100$. This value can be evaluated in real time and plotted in a graph so that the user can see the novelty metric behaviour over time.

$$e_n = \frac{\|\boldsymbol{d}_{n,i}\| - \|\boldsymbol{r}_{n,i}\|}{\|\boldsymbol{r}_{n,i}\|} = \frac{\|\boldsymbol{d}_{n,i}\|}{\|\boldsymbol{r}_i\|} - 1, \text{ where } i \text{ is the of the assigned cluster} \tag{1.7}$$

After applying this scaling, the metric now follows this rule of thumb:

- $e_n \in [-1,0] \implies \boldsymbol{\mathcal{S}}_n$ is a normal snapshot;
- $e_n \in (0,+\infty) \implies \boldsymbol{\mathcal{S}}_n$ is a novelty;

1.1.8 Transformation of the metric for the fault detection

In the previous **subsection 1.1.7** a metric has been proposed to detect how novel a snapshot is.

Let's assume now to have trained the model on a dataset of snapshots collected in a period in which the system was having a known malfunction. In this case, the metric applied to any future snapshot will carry the information of "how faulty" the system is, or better "how similarly the system is behaving w.r.t. a known malfunction".

Since the metric already defined is the normalized distance of the current snapshot from the centroid of the cluster, if the clustered data are indicative of a malfunction, the metric e_n will be in the range $[-1,0]$ when the system has a known malfunction present in the training data and e_n will be in the range $(0,+\infty)$ when the system is not behaving as a known malfunction.

To maintain the same behavior of the metric as in **subsection 1.1.7**, a trasformation that maps the range $[-1,0]$ into $(0,+\infty)$, and the range $(0,+\infty)$ into $[-1,0]$ is needed. This can be done by applying a logarithmic transformation to the metric, as shown in the **equation 1.8**. This transformation is applied only if the model is working in fault detection mode, otherwise, the metric has to remain the same as in **equation 1.7**.

$$e'_n = -\ln(e_n + 1) = -\ln\left(\frac{\|\boldsymbol{d}_{n,i}\|}{\|\boldsymbol{r}_i\|}\right), \text{ where } i \text{ is the the assigned cluster} \tag{1.8}$$

In practice, to avoid numerical representation problems, it is mandatory to avoid mapping -1 (when the new snapshot happens to be perfectly in the centre of a cluster) into $+\infty$. To do that, a constant slightly smaller than 1 is multiplied to the metric before applying the logarithmic transformation, so that the function will have the vertical asymptote slightly before -1, and all the inputs $\in [-1,0]$ will be mapped into real values. The plot of the settled function is shown in the **figure 1.4**.

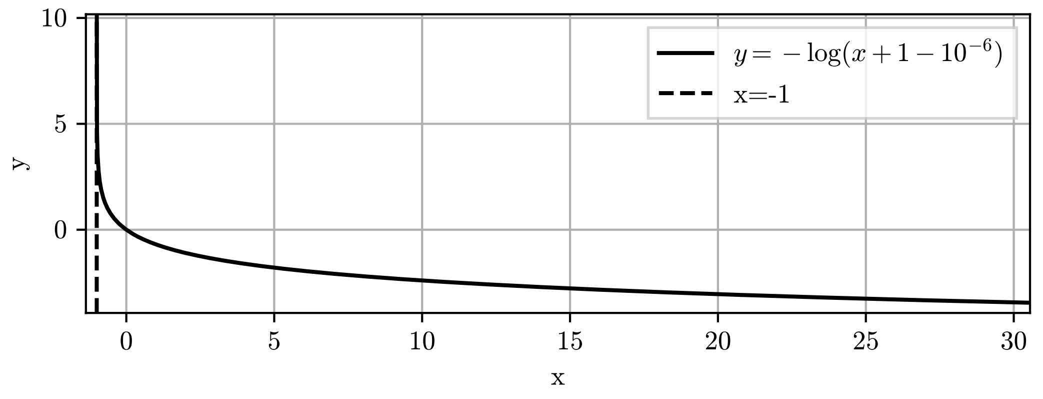


Figure 1.4: *Logarithmic Transformation applied to the metric in case the model is working in fault detection mode*

1.1.9 Evaluation procedure

The evaluation procedure developed to address the novelty/fault detection scope of this thesis can be summarized in the **algorithm 3**.

Algorithm 3 Evaluation of a new snapshot with a K-means model

```
 1: procedure EVAL($\mathcal{M}_{\texttt{k-means}}, \boldsymbol{\mathcal{S}}, t$)
 2:     ▷ $\mathcal{M}_{\texttt{k-means}}$ is the trained K-means model
 3:     ▷ the model contain the centroids $\boldsymbol{c}_i$ and the radii $\boldsymbol{r}_i$ of the clusters
 4:     ▷ $\boldsymbol{\mathcal{S}}$ is the new snapshot to be evaluated
 5:     ▷ $t$ is the threshold for the novelty detection
 6:     $k \leftarrow$ number of clusters in $\mathcal{M}_{\texttt{k-means}}$
 7:     $\min \leftarrow \infty$                                ▷ initialize the minimum distance
 8:     for $i \leftarrow 1$ to $k$ do
 9:         $\boldsymbol{d}_i \leftarrow \boldsymbol{\mathcal{S}} - \boldsymbol{c}_i$
10:         if $\|\boldsymbol{d}_i\| < \min$ then
11:             $\min \leftarrow \|\boldsymbol{d}_i\|$
12:             $i_{\min} \leftarrow i$
13:         end if
14:     end for
15:     $e \leftarrow \dfrac{\|\boldsymbol{d}_{i_{\min}}\|}{\|\boldsymbol{r}_{i_{\min}}\|} - 1$               ▷ compute the novelty metric
16:     if fault detection mode then
17:         $e \leftarrow -\ln(e + 1 - 10^{-6})$     ▷ apply the logarithmic trasformation
18:     end if
19:     if $e > t$ then
20:         return novelty flag, $e$                    ▷ the snapshot is novelty
21:     else
22:         return normal flag, $e$                     ▷ the snapshot is normal
23:     end if
24: end procedure
```

1.1.10 Comment about selecting the wrong value of k

A brief comment about what has been done in the experimental phase, in the cases where the number of clusters k was difficult to define based on the silhouette plots. This happened when the maximum value of k was shared between more than one value of k (multiple peaks with similar values or a flat shape of the peak).

In this case, the best choice is to select the maximum value of k that is still compliant with the silhouette criterion. This is because the in scope of this thesis the final goal is to detect novelty.

For display purposes, let's consider an example in the plane ($F = 2$), with a dataset that clearly has three distinct clusters. Looking at **figure 1.5** it is clear why it is better to select k larger than the actual number of clusters, rather than selecting it too small. This is because, if k is larger than the actual number of clusters, the algorithm will still be able to detect the novelty, but if k is too small, somewhere the algorithm will be forced to join two clusters together, and a new snapshot that happens to be in the middle of the two clusters will not be detected as novelty.

In the case of the **figure 1.5**, the best actual number of clusters was $k = 3$, so selecting $k = 2$ the algorithm would have joined the two clusters on the right, and the new snapshot would not have been detected as a novelty. On the other hand, by selecting $k = 4$, the algorithm would have correctly detected the novelty, even if one cluster had been split in two.

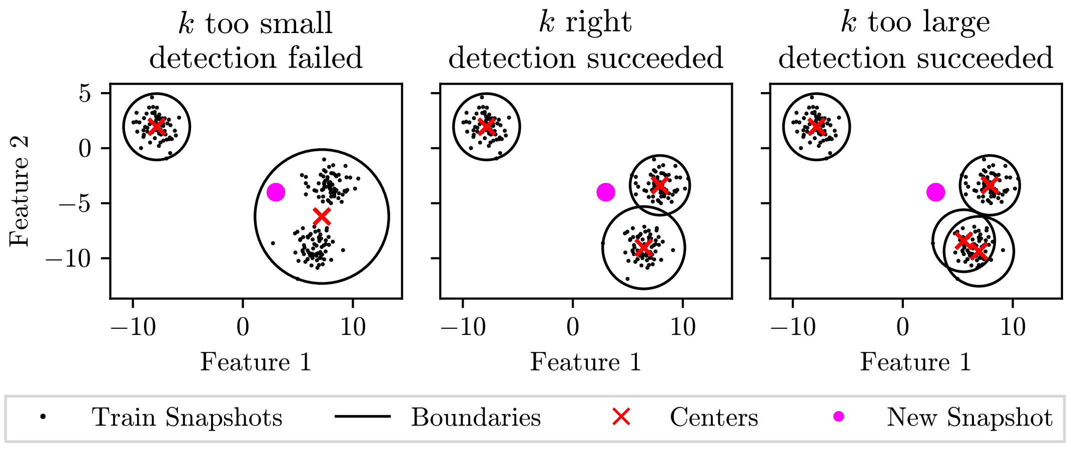


Figure 1.5: *Novelty detection of a new $\boldsymbol{\mathcal{S}}_j$ with different values of k*

1.1.11 Limitations of the algorithm

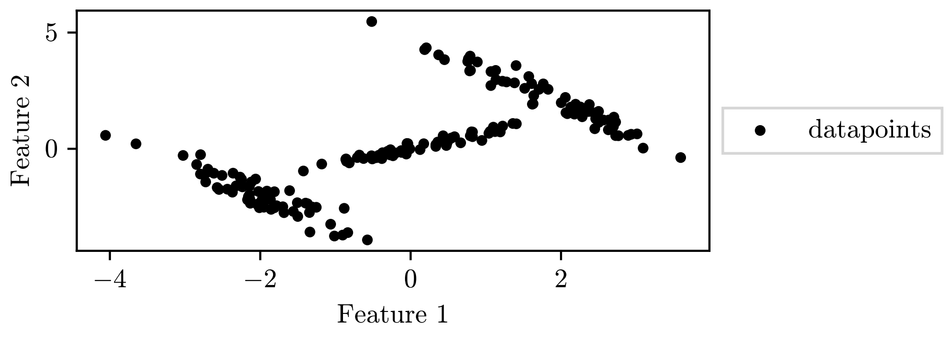
This algorithm has many advantages: it's simple, popular (easy to find in libraries), it's fast, produces a model that does not need to store the original data in order to perform the classification of a new instance and is well scalable. But it has also some limits, the most important are [1, p. 273]:

- it performs poorly if the clusters are not spherical;
- it performs poorly if the clusters have very different sizes;
- it performs poorly if the clusters have different densities.

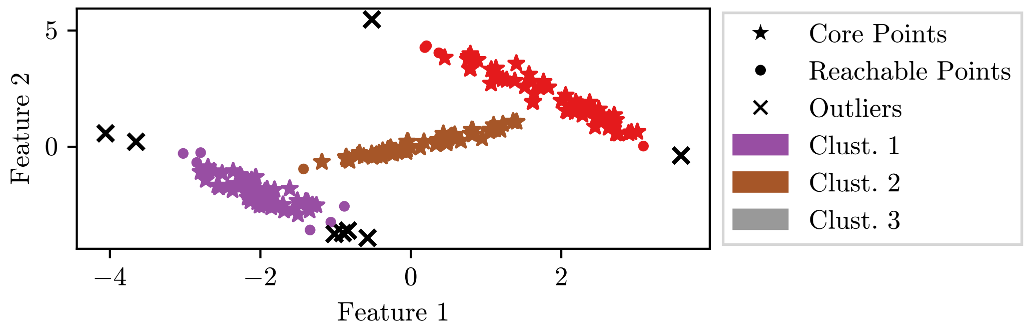
For this reason, it is very important to perform a good preconditioning of the data. In this thesis, the data are standardized before the training phase, this makes the clusters more spherical and with similar sizes.

1.2 DBSCAN

1.2.1 Overview



(a) *training data*



(b) *training result*

Figure 1.6: *DBSCAN clustering*

Looking at **figure 1.6a** we can see that the data are clearly divided into 3 clusters, but a simple K-means will fail to cluster correctly the data because the clusters are very stretched. But how do we humans see that there are 3 clusters? We instinctively look at the density of the points, and we can see that there are 3 areas with a high density of points, and the rest of the space is empty. DBSCAN is a clustering algorithm that tries to mimic this behaviour.

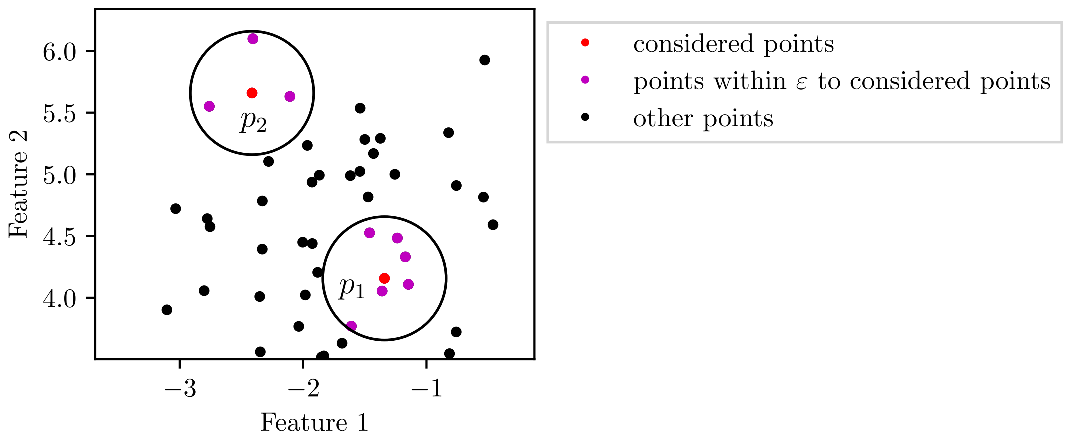


Figure 1.7: *Example of core and border points*

DBSCAN is a density-based clustering designed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu [16]. The algorithm is based on the definition of what a *core point* is, and what a *density-reachable* point is.

The algorithm inputs are:

- D: the dataset
- ε: the radius of the neighbourhood
- $MinPts$: the minimum number of points to form a cluster

The basic idea is that if a point has a sufficient number of other points ($MinPts$) in its neighbourhood of radius ε, then it is a *core point*. Chosen a core point, all the other core points in its neighbourhood are assigned to the same cluster, and the process is repeated for all the core points in the neighbourhood of the just assigned ones and so on. At a certain point there will be a cluster of core points that are not near any other core point, when this happens the idea is to include in the cluster also the points that are not core points but are in the neighbourhood of the core points of the cluster.

To better visualize the process, **figure 1.7** shows some data points. Let's assume $MinPts = 5$ The point p_1 is a core point because, in its neighbourhood of radius ε, there are $7 > MinPts$ points. The point p_2 is not a core point because in its neighbourhood there are only $3 < MinPts$ points. even if it is not a core point, it may be assigned to a cluster, depending on if there is a core point in its neighbourhood. If a point is not a core point and there is not a core point in its neighbourhood, then it is a noise point and it is not assigned to any cluster.

Some of the definitions presented in [16] can be resumed for our purpose as follows:

- $N_\varepsilon(p) = q \in D : \|\boldsymbol{d} \leq \varepsilon\|$ is the set of points in the ε-neighbourhood of p;
- a point p is a *core point* if there are at least $MinPts$ points in the ε-neighbourhood of p;
- a point p is *directly density-reachable* from q if p is in the ε-neighbourhood of q and q is a core point;
- a point p is *density-reachable* from q if there is a chain of points $p_1, \ldots, p_n$ such that $p_1 = q$ and $p_n = p$ and p_{i+1} is directly density-reachable from p_i;
- a point p is *density-connected* to q if there is a point o such that both p and q are density-reachable from o;

In the paper [16] the authors propose a detailed pseudocode of the algorithm, **algorithm 4** is a more abstract version of it.

Algorithm 4 Train DBSCAN

```
1: procedure DBSCAN($D, \varepsilon, MinPts$)
2:     for $p \in D$ do
3:         if $|N_\varepsilon(p)| \geq MinPts$ then
4:             mark $p$ as a core point
5:         end if
6:     end for
7:     $i \leftarrow 0$                                                ▷ cluster index
8:     while there are unassigned points do
9:         $cluster_i \leftarrow \emptyset$
10:        choose an unassigned point $p$
11:        $cluster_i \leftarrow cluster_i \cup p$                     ▷ add $p$ to the cluster
12:        for $q \in$ all reachable points from $p$ do
13:            ▷ all density-reachable points from $p$ are added to the cluster
14:            $cluster_i \leftarrow cluster_i \cup q$                 ▷ add $q$ to the cluster
15:        end for
16:        if there are no unassigned core points then
17:            drop all unassigned points from $D$                     ▷ they are noise
18:        end if
19:        $i \leftarrow i + 1$
20:    end while
21: end procedure
```

1.2.2 Choosing the parameters

Running the algorithm implemented in the `sklearn` library on the dataset of **figure 1.6a** with $\varepsilon = 0.78$ and $MinPts = 10$ we obtain the result of **figure 1.6b**. The algorithm correctly identifies the 3 clusters, also the noise points number is affected by the choice of ε and $MinPts$.

Note that the DBSCAN is able to identify clusters with arbitrary shapes, even if they are not convex and are not linearly separable. This is a big advantage over the K-means, which is not able to identify clusters with arbitrary shapes.

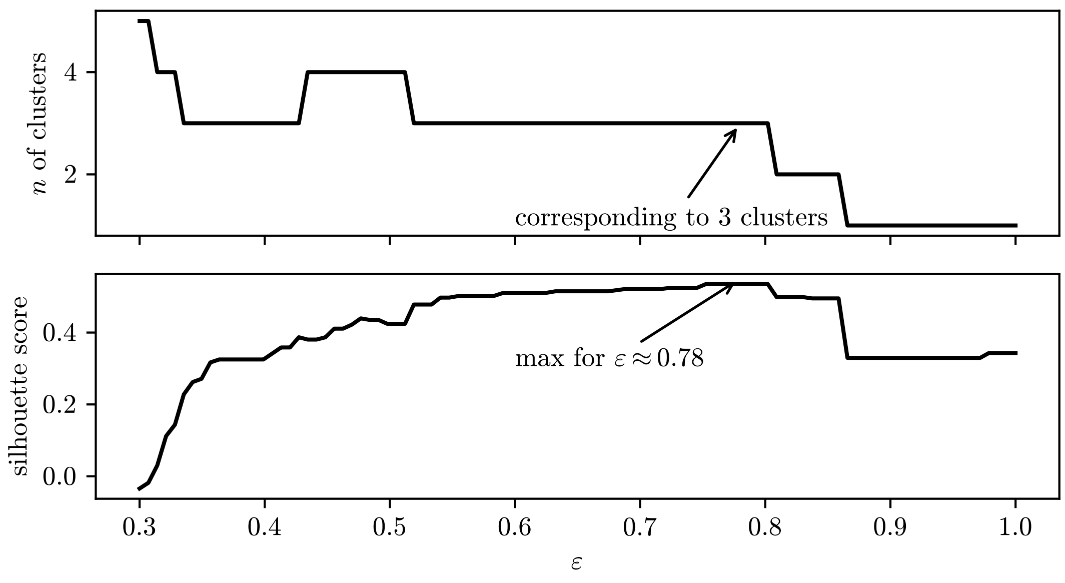


Figure 1.8: *Silhouette score for different values of* ε

Even being the DBSCAN an unsupervised algorithm, it has some parameters that need to be set by the user. The first parameter is ε, the radius of the neighbourhood. This gives a measure of the density we expect from the clusters. As seen for the K-means in **section 1.1**, we can try to find the best ε by using the silhouette score.

Let's plot the silhouette score for different values of ε to confirm that using $\varepsilon = 0.78$ is a good choice. In the **figure 1.8** are shown the results that link the best ε value to 3 cluster being generated.

1.2.3 Evaluation of a new instance

Assume now that the training of the DBSCAN is complete, and a new snapshot $\boldsymbol{\mathcal{S}}_n$ is generated from the sensor data. How can we evaluate if the new snapshot is novel or not? The DBSCAN algorithm is not able to predict the cluster of a new instance. However, the task of this project is novelty detection, not classification. To do that, as has been done for the K-means, a metric linked to how novel a new snapshot is developed.

The idea used previously to compute the distance to a centroid and normalizing it by the radius of the cluster is not applicable here, because the DBSCAN do not use centroids to define clusters. A naive idea for our purpose is to compute the distance of the new snapshot from the closest snapshot in the training dataset. If the distance is greater than a threshold, then the new snapshot is considered novel.

And now the question: if we use the distance of a new $\boldsymbol{\mathcal{S}}_n$ from the closest $\boldsymbol{\mathcal{S}}_i$ in the training dataset (spanning the whole dataset to compute it) why did we train a model in the first place? Wouldn't it be simpler to just apply this metric to the training dataset without clustering it? As far as concerns scope of this thesis, performing the clustering first has two main advantages:

- ε has been chosen running the DBSCAN, so nobody had to guess it. This is a big advantage because the choice of ε would be very difficult to do with limited knowledge of the monitored system. Normalizing the computed distance by ε we can obtain a measure of how novel a snapshot is for which we can set a threshold without tuning it on the specific system;

- during the training phase, the DBSCAN has discarded the noise points. This improves the quality of the metric because the distance from a noise point is not meaningful.

1.2.4 Limitations of the algorithm

The first limitation of DBSCAN is about cluster density: since ε is fixed, the algorithm is not able to identify clusters with different densities. In this case, it will split low-density clusters into multiple clusters. For our purpose, this is not a problem, because we are interested in novelty detection, so even if a cluster is misidentified as multiple clusters, the new snapshot will be considered novel. To increase the sensitivity of the novelty detection ε can be decreased, but this will increase the number of noise points.

The other main limitation of DBSCAN is that it has a complexity of roughly $\mathcal{O}(m^2)$, where m is the number of points in the dataset. This means that the algorithm is not scalable to large datasets [1, p. 281]. There exist improvements to the algorithm that reduce the complexity to $\mathcal{O}(m \log m)$, like the FDBSCAN developed by Bing Liu [17]. The complexity remains linear w.r.t. the number of features.

1.3 Gaussian Mixture Model

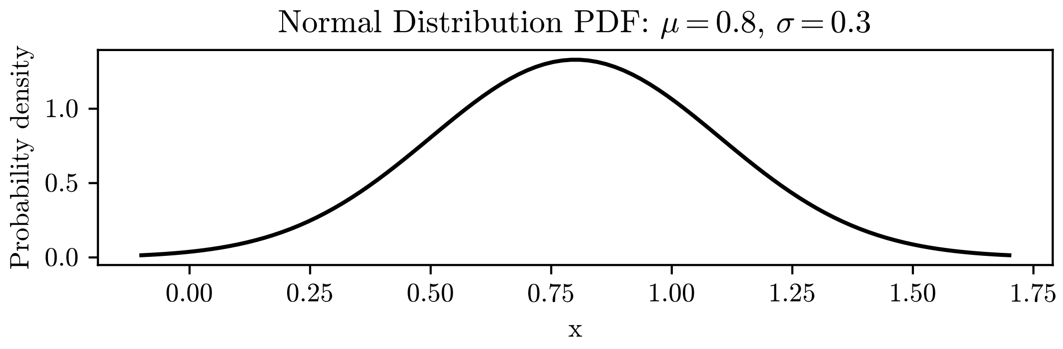


Figure 1.9: *Gaussian distribution probability density function*

The **figure 1.9** illustrates a Gaussian distribution probability density function (PDF), also known as normal distribution. The peak represents the mean μ, while the spread is determined by the standard deviation σ. The equation of the PDF is the following:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

A Gaussian Mixture Model is a probabilistic model that assumes that the data are generated from a mixture of several Gaussian distributions. Depending on the parameters of a Gaussian distribution (mean and variance) on each axis (Features), the data generated from it in the F-dimensional space will have a shape that resembles an ellipsoid with a centre and an orientation.

1.3.1 Training

As anticipated, the assumption is that every cluster has a normal PDF on each axis. So, for each feature, the total PDF is a superposition of k normal PDF. The algorithm used to train the model is the Expectation Maximization (EM) algorithm. This is a generalization of the k-means algorithm that allows to find the centroids (means), the covariance matrices of the clusters, and their weights. Unlike the k-means, the EM is a soft clustering algorithm, meaning that each point is assigned to each cluster with a probability, instead of being assigned to a single cluster. The EM algorithm is an iterative algorithm that aims to find the parameters that maximize the likelihood function.

1.3.2 Selecting the number of clusters

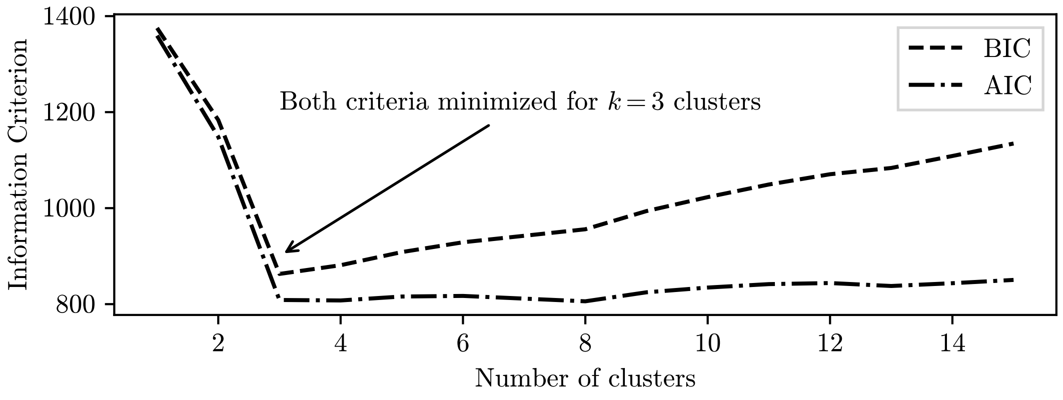


Figure 1.10: *Criteria for selecting the number of clusters*

In **subsection 1.3.1** we have seen that the EM algorithm needs to know in advance the number of clusters k to train the model. To select the best k we could use the same criteria used for the k-means, that is the elbow method or the silhouette score. But, since the EM is most suitable for ellipsoids-shaped clusters of very different sizes, it is better to avoid those metrics that work best with spherical clusters. Instead, we can use the AIC or the BIC criteria, which are functions of the size m of the training dataset, the number p of parameters, and the max value of likelihood function $\hat{\mathcal{L}}$.

$$\begin{aligned} AIC &= 2p - 2\ln(\hat{\mathcal{L}}) \\ BIC &= p\ln(m) - 2\ln(\hat{\mathcal{L}}) \end{aligned} \tag{1.9}$$

Both the criteria in **equation 1.9** penalize the models with a large number of parameters, they usually select the same model, but if they select different models, the AIC tends to select the most accurate model.

Considering, as an example, the same dataset used in **figure 1.6a**, and running the EM algorithm with different values of k, we obtain the results shown in **figure 1.10**. As expected the AIC and the BIC criteria select the same model, that is the one with $k = 3$.

Totally automated approach

Another advantage of the GMM is that it can be used in a totally automated way. This can be done using the variation called Bayesian Gaussian Mixture Model BGMM. This variation is able to assign 0 as weight to some clusters, meaning that those clusters are not used to generate the PDF of the samples. In this case, we can set the number of clusters as high as the size of the training dataset, and the BGMM will select the best number of clusters to use to generate the PDF of the samples.

1.3.3 Evaluation of a new instance

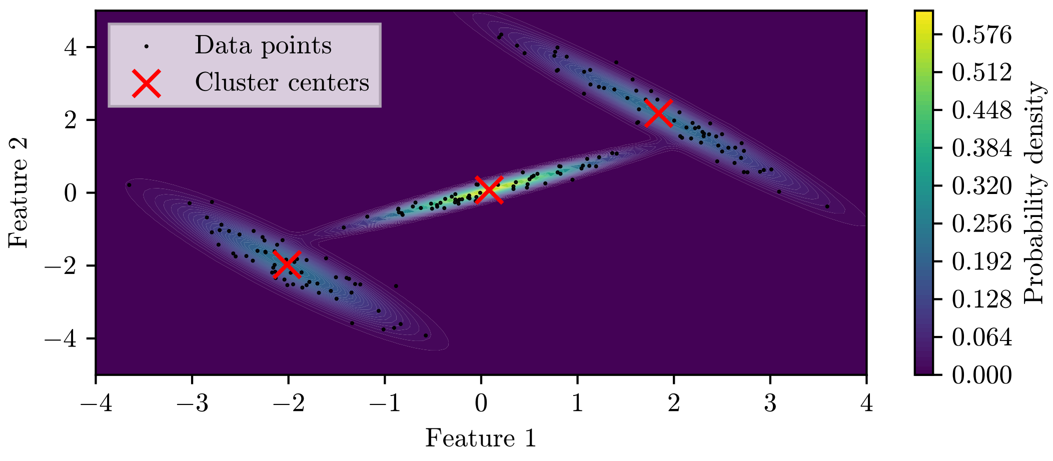


Figure 1.11: *Trained Gaussian Mixture Model*

Once the model is well-trained, we can use it to evaluate a new instance. The **figure 1.11** shows the result of the training of the GMM on the example dataset. The ellipsoids represent the clusters, the size of the ellipsoids is proportional to the weight of the cluster, while the orientation and the size of the ellipsoids are determined by the covariance matrix of the cluster.

The `pyhton` implementation of the GMM used in this project is the one provided by the `sklearn` library. It has an attribute called `score_samples` that returns the log value of the PDF of the samples. The bigger the value, the more likely it is to be inside a cluster. To maintain the same philosophy of the other algorithms, we can take the negative value of `score_samples` as a metric to evaluate the novelty of a new instance.

Selecting the threshold

As said for K-means, and DBSCAN, a threshold will be needed to decide if the new instance is novel or not. If, as supposed for now, the scenario is that the training dataset is composed only of normal instances, then the threshold can be selected by looking at the distribution of the novelty metric on the training dataset and selecting a value slightly higher than the maximum value for the training dataset.

If the scenario is that the training dataset is composed of both normal and anomalous instances sampled at constant intervals, and the ratio between normal and anomalous instances is known, then GMM enables us to select the threshold in a more sophisticated way.

Suppose for example that 1% of the training dataset is anomalous, since the model gives us the PDF of the samples, we can compute the threshold as the 1% percentile of the PDF of the training dataset. This will ensure that the 1% of the training instances will be classified as anomalous. If the model is correct, also future evaluations will have the same ratio of anomalous instances.

1.3.4 Limitations of the algorithm

The main limitation of the GMM is that it is not able to identify clusters with arbitrary shapes.

If the data are not shaped like some ellipsoids, the model can approximate the data anyway, splitting the clusters into smaller ellipsoids. For our purposes of novelty detection, this is not a limitation.

The complexity of the algorithm is $\mathcal{O}(kmF^2 + kF^3)$, where k is the number of clusters, m is the number of samples, and F is the number of features [1, p. 281]. This means that the algorithm is not scalable to a large number of features, but it is scalable to a large number of samples. For our purposes, it's more important to have an algorithm that is fast to evaluate a new instance, than an algorithm that is fast to train.

1.4 Isolation Forest

Similarly to the Random Forest, which is used for classification, the Isolation Forest is an ensemble method that is used for anomaly detection. The idea is to train the decision trees to isolate the data from each other, rather than profiling the normal data, and then provide a score that represents how isolated the data is. The more isolated the data is, the more likely it is to be an anomaly. This algorithm was introduced in 2008 by [18], the paper also provides a way of computing the anomaly score.

This algorithm has the advantage of being very fast to train and requiring very little memory. For hour purposes, as previously said, it is more important to have an algorithm that is fast to evaluate a new instance, than an algorithm that is fast to train.

1.4.1 Training

The algorithm is available in `sklearn` library, and it is very easy to use. This is a truly unsupervised algorithm, the function only takes the dataset as input and manages to automatically select all the rest of the parameters. As an example, I used the same dataset used in the previous sections for the DBSCAN and GMM algorithms. The result is shown in **figure 1.12**.

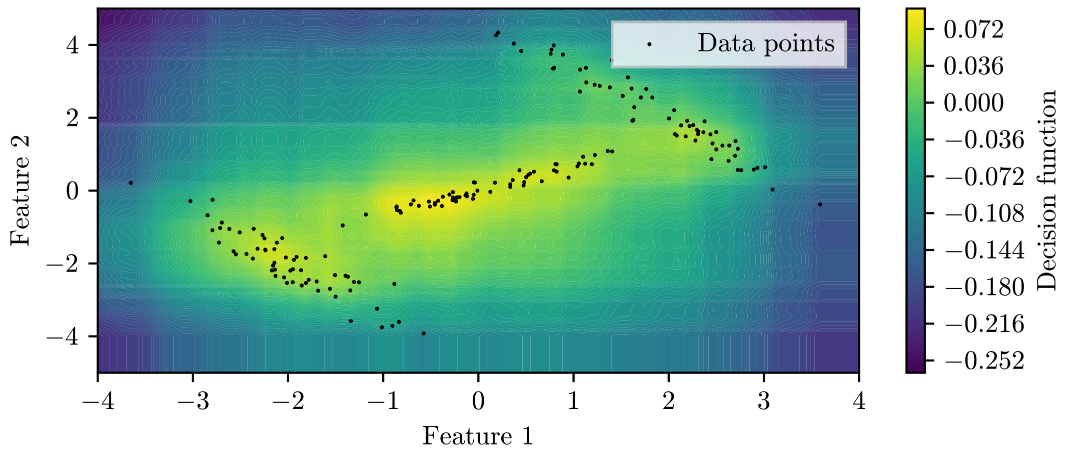


Figure 1.12: *Isolation Forest decision function.*

1.4.2 Evaluation of a new instance

The implementation in `sklearn` provides a function called `decision_function`, plotted as a heatmap in **figure 1.12**. the higher the value, the more likely it is to be a normal instance. To maintain coherence with the work done up to now, we can take the negative value of `decision_function` as a metric to evaluate the novelty of a new instance.

Selecting the threshold

With this algorithm is it difficult to geometrically interpret the decision function, so it is not possible to select the threshold a priori. The approach could be to look at the values of the decision function on the training dataset and select a value slightly higher than the maximum value for the training dataset.

1.4.3 Limitations of Isolation Forest

The main limitation is that since the algorithm is based on decision trees, the decision thresholds are defined on the features axis, so the decision boundaries are highly dependent on the alignment of clusters with the axis. looking back to **figure 1.12**, we can see that the decision function has roughly the same value in the lower right corner as in the lower left corner, even if the lower right corner is clearly more isolated than the lower left corner. This is because the decision boundaries are aligned with the axis, and the lower right corner is more isolated only in the diagonal direction.

The complexity of the training phase is $\mathcal{O}(t\psi \log \psi)$ where t is the number of trees used, and ψ is the size of the subsampling size. The complexity of the evaluation is instead $\mathcal{O}(t \log \psi)$ [18].

1.5 Local Outlier Factor

The Local Outlier Factor (LOF) algorithm is effective in identifying outliers by assessing the density of instances surrounding a particular data point in comparison to the density around its neighbouring points. Typically, an anomaly is considered more isolated than its k nearest neighbours [1, p. 293].

Formally, the authors that designed this algorithm define the LOF of an instance $\boldsymbol{\mathcal{S}}$ as the average of the ratio of the local reachability density of $\boldsymbol{\mathcal{S}}$ and the local reachability density of its $MinPts$ k-nearest neighbours [19]. This is a measure of how isolated the instance is with respect to the surrounding neighbourhood.

Using this approach, this algorithm is able to identify outliers in a dataset with arbitrary shapes. Moreover, it can identify outliers in a dataset with different densities, as a point very near to a very dense cluster can be declared as an outlier, while a point even more distant from a less dense cluster could be declared as normal.

1.5.1 Training

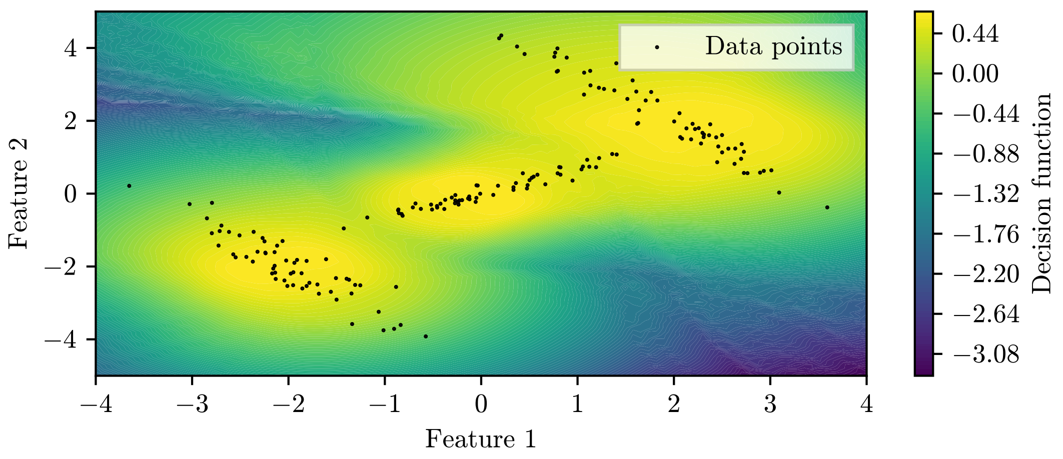


Figure 1.13: *Local Outlier Factor decision function.*

This algorithm is implemented in `sklearn` library and requires the number $MinPts$ of k-nearest neighbours to be specified. According to the authors, the value of LOF of a particular snapshot is neither increasing nor decreasing with the value of $MinPts$. They also suggest an heuristic to select the value of $MinPts$ that is hardly automatable. For this reason, it's difficult to use this algorithm in a real-time scenario in an unsupervised way.

Anyway, as an example, using the same dataset used in **figure 1.6a**, and running the LOF algorithm with the default value of $MinPts = 20$, we obtain the results shown in **figure 1.13**.

1.5.2 Evaluation of a new instance

The `sklearn` implementation of the LOF algorithm has a method that returns the LOF of a new instance. Since the bigger the value is, the more likely it is that the new snapshot is an outlier, we can take directly this value as a metric to evaluate the novelty of a new instance.

To select a threshold value, it holds what has been said in **subsection 1.4.2** about iForest.

1.5.3 Limitations of Local Outlier Factor

The main limitations are the difficulty of defining a threshold value to declare some instances as outliers and using this algorithm in a completely unsupervised way.

1.6 One-Class Support Vector Machine

This algorithm is based on the kernelized SVM algorithm. While the standard application is to find the hyperplane that separates two classes, in this case, the aim is to separate the instances from the origin. If a new instance is too close to the origin (of an augmented hyperspace), then it is considered an outlier. It was introduced in 2001 by Müller [20].

1.6.1 Training

As an example, we refer to the same dataset used in the previous sections for the DBSCAN and GMM algorithms. The training of the version implemented in `sklearn` requires specifying the kernel to use, and the parameter ν that is the upper bound on the fraction of outlier. Training the algorithm on the dataset, with Gaussian kernel and $\nu = 0.002$, we obtain the result shown in **figure 1.14**.

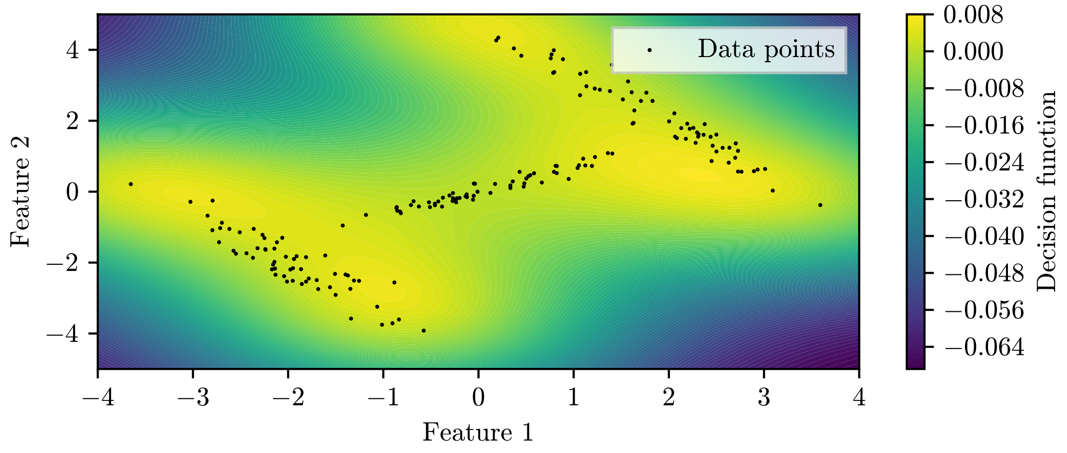


Figure 1.14: *One-Class Support Vector Machine decision function.*

1.6.2 Evaluation of a new instance

The decision function is the relative distance from the separation hyperplane, if it is positive, then the instance is considered normal, if it is negative, then the instance is considered an outlier.

This allows us to take directly the negative of this value as a metric to evaluate the novelty of a new instance (to maintain coherence with previous sections).

Furthermore, similarly to the GMM case (**section 1.3**), if the training dataset contains only normal instances, we have to guess a positive value for the threshold by looking at the value of the metric on the training dataset, but if we have a dataset with a known fraction of outliers, we can use the fact that ν is the upper bound on the fraction of outliers to select ν correctly in the training phase, and then use a threshold of zero for the decision function. This should make at most a ν fraction of future evaluation to be considered outliers.

1.6.3 Limitations of ν-SVM

The limitations are about the sensitivity to the choice of the kernel and the parameter ν, making it difficult to use in a completely unsupervised manner.

It works well in high dimensional spaces, but it is not suited for large datasets [1, p. 294]

Bibliography

[1] Aurelien Geron. *Hands-On Machine Learning With Scikit-Learn, Keras & TensorFlow*. English. O'Reilly Media, 2022, p. 850. ISBN: 9781098125974 (cit. on pp. 1, 2, 15, 20, 24, 26, 29).

[2] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. «The planar k-means problem is NP-hard». In: *Theoretical Computer Science* 442 (2012). Special Issue on the Workshop on Algorithms and Computation (WALCOM 2009), pp. 13–21. ISSN: 0304-3975. DOI: `https://doi.org/10.1016/j.tcs.2010.05.034`. URL: `https://www.sciencedirect.com/science/article/pii/S0304397510003269` (cit. on p. 2).

[3] James MacQueen et al. «Some methods for classification and analysis of multivariate observations». In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA. 1967, pp. 281–297 (cit. on p. 2).

[4] S. Lloyd. «Least squares quantization in PCM». In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137. DOI: `10.1109/TIT.1982.1056489` (cit. on pp. 2, 5).

[5] Pavel Berkhin. *Survey of clustering data mining techniques*. Tech. rep. San Jose, CA: Accrue Software, 2002 (cit. on p. 3).

[6] Abla Chouni Benabdellah, Asmaa Benghabrit, and Imane Bouhaddou. «A survey of clustering algorithms for an industrial context». In: *Procedia Computer Science* 148 (2019). The second international conference on intelligent computing in data sciences, ICDS2018, pp. 291–302. ISSN: 1877-0509. DOI: `https://doi.org/10.1016/j.procs.2019.01.022`. URL: `https://www.sciencedirect.com/science/article/pii/S1877050919300225` (cit. on p. 3).

[7] Hans-Peter Kriegel, Erich Schubert, and Arthur Zimek. «The (black) art of runtime evaluation: Are we comparing algorithms or implementations?» In: *Knowledge and Information Systems* 52.2 (Oct. 2017), pp. 341–378. ISSN: 0219-3116. DOI: `10.1007/s10115-016-1004-2`. URL: `https://doi.org/10.1007/s10115-016-1004-2` (cit. on p. 3).

[8] Malay K. Pakhira. «A Linear Time-Complexity k-Means Algorithm Using Cluster Shifting». In: *2014 International Conference on Computational Intelligence and Communication Networks.* 2014, pp. 1047–1051. DOI: `10.1109/CICN.2014.220` (cit. on p. 5).

[9] Mary Inaba, Naoki Katoh, and Hiroshi Imai. «Applications of Weighted Voronoi Diagrams and Randomization to Variance-Based k-Clustering: (Extended Abstract)». In: *Proceedings of the Tenth Annual Symposium on Computational Geometry.* SCG '94. Stony Brook, New York, USA: Association for Computing Machinery, 1994, pp. 332–339. ISBN: 0897916484. DOI: `10.1145/177424.178042`. URL: `https://doi.org/10.1145/177424.178042` (cit. on p. 5).

[10] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. «A local search approximation algorithm for k-means clustering». In: *Comput. Geom.* 28.2-3 (2004), pp. 89–112 (cit. on p. 5).

[11] Charles Elkan. «Using the Triangle Inequality to Accelerate K-Means». In: *Proceedings of the Twentieth International Conference on International Conference on Machine Learning.* ICML'03. Washington, DC, USA: AAAI Press, 2003, pp. 147–153. ISBN: 1577351894 (cit. on p. 5).

[12] D. Sculley. «Web-Scale k-Means Clustering». In: *Proceedings of the 19th International Conference on World Wide Web.* WWW '10. Raleigh, North Carolina, USA: Association for Computing Machinery, 2010, pp. 1177–1178. ISBN: 9781605587998. DOI: `10.1145/1772690.1772862`. URL: `https://doi.org/10.1145/1772690.1772862` (cit. on p. 5).

[13] Damodar Reddy and Prasanta K. Jana. «Initialization for K-means Clustering using Voronoi Diagram». In: *Procedia Technology* 4 (2012). 2nd International Conference on Computer, Communication, Control and Information Technology(C3IT-2012) on February 25 - 26, 2012, pp. 395–400. ISSN: 2212-0173. DOI: `https://doi.org/10.1016/j.protcy.2012.05.061`. URL: `https://www.sciencedirect.com/science/article/pii/S2212017312003404` (cit. on p. 6).

[14] David Arthur and Sergei Vassilvitskii. «K-Means++: The Advantages of Careful Seeding». In: vol. 8. Jan. 2007, pp. 1027–1035. DOI: `10.1145/1283383.1283494` (cit. on p. 6).

[15] Lilian Bejarano, Helbert Espitia, and Carlos Montenegro. «Clustering Analysis for the Pareto Optimal Front in Multi-Objective Optimization». In: *Computation* 10 (Mar. 2022), p. 37. DOI: `10.3390/computation10030037` (cit. on p. 7).

[16] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. «A density-based algorithm for discovering clusters in large spatial databases with noise». In: *kdd*. Vol. 96. 34. 1996, pp. 226–231 (cit. on pp. 17, 18).

[17] Bing Liu. «A Fast Density-Based Clustering Algorithm for Large Databases». In: *2006 International Conference on Machine Learning and Cybernetics*. 2006, pp. 996–1000. DOI: `10.1109/ICMLC.2006.258531` (cit. on p. 21).

[18] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. «Isolation Forest». In: *2008 Eighth IEEE International Conference on Data Mining*. 2008, pp. 413–422. DOI: `10.1109/ICDM.2008.17` (cit. on pp. 24, 26).

[19] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. «LOF: Identifying Density-Based Local Outliers». In: *SIGMOD Rec.* 29.2 (May 2000), pp. 93–104. ISSN: 0163-5808. DOI: `10.1145/335191.335388`. URL: `https://doi.org/10.1145/335191.335388` (cit. on p. 26).

[20] Klaus-Robert Müller, Sebastian Mika, Gunnar Rätsch, Koji Tsuda, and Bernhard Schölkopf. «An Introduction to Kernel-Based Learning Algorithms». In: *IEEE TRANSACTIONS ON NEURAL NETWORKS* 12.2 (2001), p. 181 (cit. on p. 28).