DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Data Engineering and Analytics

# Integration and Visualization of Sparse-Grid based Clustering Methods in the SG++ Datamining Pipeline

Vincent Bennet Bautista Anguiano

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Data Engineering and Analytics

# Integration and Visualization of Sparse-Grid based Clustering Methods in the SG++ Datamining Pipeline

# Integration und Visualisierung von dünngitter basierten Clustering Methoden in der SG++ Datamining Pipeline

| | |
|---|---|
| Author: | Vincent Bennet Bautista Anguiano |
| Supervisor: | Prof. Dr. rer. nat. habil. Hans-Joachim Bungartz |
| Advisor: | M.Sc (Hons) Paul Cristian Sarbu |
| Submission Date: | 15th April, 2020 |

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Garching bei München, 15th April, 2020 Vincent Bennet Bautista Anguiano

# Acknowledgments

I would like to extend my special thanks to my advisors Paul Sarbu and Kilian Röhner for giving me the opportunity to work in this topic and for their excellent guidance, support and availability throughout the duration of the Master Thesis.

I would also like to express my wholeheartedly gratitude towards my colleagues from Wirecard AG, who have accompanied me for almost all of the duration of the Master Program and whose knowledge and guidance have been a valuable complement to my professional and personal formation.

I would like to thank also my friends, both from TUM and external, who made my stay at Germany one of the most enjoyable experiences of my life.

Finally, I would like to thank my family, who have given me their total support for the duration of my studies and without who I would had never been able to pursue a Master Degree in the first place.

# Abstract

The SG++ Datamining Pipeline is a component of the SG++ Toolbox whose main purpose is to provide an interface to generate Machine Learning Models based on Sparse Grid Methods. These are numerical techniques which have been previously proven to be successful in solving tasks handling large high dimensional data sets [29].

Until recently, the pipeline provided support only to train Sparse Grid based Density Estimation, Classification and Regression Models. In this thesis, we took the task of integrating the support for Clustering Models. We did it so by implementing the Sparse Grid based Clustering Algorithm created by Peherstorfer [28] along with a special augmentation designed by Fischer [8] to generate a Hierarchical Clustering.

Additionally, we implemented a series of metrics used to evaluate the quality of the clustering and a series of processes used to generate an output with the purpose of generating graphical representations of these models. We provide in this thesis the design of our implementation and the results of a series of tests conducted to show and to evaluate its functionality.

# Contents

# 1 Introduction

Clustering is one of the most common tasks in Machine Learning, along with Classification and Regression, whose objective is to find groups within a given data set based only on its internal structure. A certain advantage of Clustering Algorithms over other Machine Learning Methods is their unsupervised learning nature, that means, data with previously predefined labels is not needed. However, just like many other Machine Learning Methods, Clustering Algorithms have their share of challenges to overcome, specially when it comes down to handling large high dimensional data sets efficiently.

Sparse Grid Methods come to mind when handling such problems, resulting in the development of Machine Learning Models based on them to process large high dimensional data sets efficiently. Some of them, like Classification and Regression, have already been implemented into the SG++ Datamining Pipeline, a component of the SG++ Toolbox, which is a programming library written in C++, which provides interfaces to use and apply Sparse Grid Methods. In this thesis, we have extended the functionalities of the pipeline by integrating into it the support for Sparse Grid based Clustering Models.

We will begin by introducing all the theory behind our Clustering implementation in Chapter 2. In this chapter, we will give an overview of Sparse Grid Methods, we will introduce the additional algorithms and data structures which were used in our implementation, we will talk about Clustering Quality Analysis and we will make a small introduction to the current structure of SG++ Datamining Pipeline along with its functionality and configuration.

The design of our implementation will be presented in Chapter 3. In this chapter, we will go into detail about the software architecture, the changes made to the pipeline in order to integrate our Clustering Models and the description of the main classes implemented along with their functionality.

In Chapter 4, we will present the tests used to show the functionality of our implementation along with a description of the data sets used and the running configurations. The results of these tests will be presented in Chapter 5 along with a small analysis of them. Finally, in Chapter 6, we will present our final conclusions and make an overview of the future improvements that could still be made to our implementation and in general to the Datamining Pipeline.

# 2 Theoretical Background

## 2.1 Sparse Grid Methods

Sparse Grid Methods are special discretization techniques, which allow to cope with the curse of dimensionality of grid based mathematical approaches to a certain extent [12].

The original idea can be traced back to the works of the russian mathematician Smolyak [34], who used them for numerical integration. They were later however, developed to approximate the solution of partial differential equations [13] and in the development of certain Machine Learning algorithms and applications [29] [28] [11] [8] [36] [19] .

### 2.1.1 Hierarchical Basis and Sparse Grid Interpolation

Sparse Grid Methods are based on the hierarchical basis, a hierarchical decomposition of base functions underlying the approximation of spaces [29]. The parent function of this basis is the standard hat function, which in the 1-dimensional case is defined as:

$$\phi(x) = \max(1 - |x|, 0) \tag{2.1}$$

The basis functions are derived by dilatation and translation of the hat function,

$$\phi_{l,i}(x) := \phi(2^l x - i) \tag{2.2}$$

where $l$ indicates the level of the function and $i$ an index in the range of $(0, 2^l)$. These basis functions are centered at the grid points $x_{l,i} = 2^{-l} i$. It is important to remark that these index values are restricted to the following hierarchical index sets

$$I_l = \{i \in \mathbb{N} : 1 \leq i \leq 2^{l-1}, i \text{ is odd}\} \tag{2.3}$$

Per level $l$ we obtain a set of hierarchical subspaces $W_l$

$$W_l := \text{span}(\phi_{l,i} : i \in I_l\} \tag{2.4}$$

A space of piecewise linear functions $V_n$ on a grid $h_n$ for a given level $n$ can be then defined as the direct sum of $W_l$,

$$V_n = \bigoplus_{l \leq n} W_l \tag{2.5}$$

so any function $f(x) \in V_n$ can be approximated by the interpolant $u(x) \in V_n$, the latter being a linear combination of these basis functions

$$f(x) \approx u(x) = \sum_{l \leq n, i \in I} \alpha_{l,i} \phi_{l,i}(x) \tag{2.6}$$

The idea is extended to higher dimensional spaces through the tensor product approach

$$\phi_{\vec{l},\vec{i}}(\vec{x}) := \prod_{j=1}^{d} \phi_{l_j,i_j}(\vec{x}_j) \tag{2.7}$$

with $\vec{l},\vec{i}$ being the multidimensional indexes equivalent to $l$ and $i$ in the 1-dimensional case. The other notations are similarly transferred to the high dimensional case

$$I_{\vec{l}} = \{\vec{i} : 1 \leq i_j \leq 2^{l-1}, i_j \text{ is odd}, 1 \leq j \leq d\} \tag{2.8}$$

$$W_{\vec{l}} := \text{span}(\phi_{\vec{l},\vec{i}} : i \in I_{\vec{l}}) \tag{2.9}$$

$$V_n = \bigoplus_{|\vec{l}|_\infty \leq n} W_{\vec{l}} \tag{2.10}$$

This leads to the high dimensional interpolant $u(\vec{x}) \in V_n$

$$u(\vec{x}) = \sum_{|\vec{l}|_\infty \leq n, i \in I_{\vec{l}}} \alpha_{\vec{l},\vec{i}} \phi_{\vec{l},\vec{i}}(\vec{x}) \tag{2.11}$$

using a full grid with $(2^n - 1)^d$ grid points.

The $L_2$ error of the interpolant $u(x)$ to the original function $(f(x))$ is bounded by $O(h_n)$, however $O(h_n^{-d}) = O(2^{nd})$ function evaluations are needed. This is unfeasible for high dimensional problems. Therefore, we only select those subspaces $W_l$ which contribute the most to the solution. This generates the sparse grid space

$$V_n^{(1)} = \bigoplus_{|\vec{l}|_1 \leq n+d-1} W_{\vec{l}} \tag{2.12}$$

and our sparse grid interpolant results in:

$$u(\vec{x}) = \sum_{|\vec{l}|_1 \leq n+d-1, i \in I_{\vec{l}}} \alpha_{\vec{l},\vec{i}} \phi_{\vec{l},\vec{i}}(\vec{x}) \tag{2.13}$$

This significantly reduces the number of functions evaluations from $O(h_n^{-d})$ to $O(h_n^{-1}(\log h_n^{-1})^{d-1})$ at an $L_2$ error bounded by $O(h_n^2(\log h_n^{-1})^{d-1})$. This negligible difference in accuracy makes the use of sparse grids an attractive option. Additionally, our interpolation problem becomes only dependent on the size of the grid used, which most of the time is significantly smaller than the size of the data set to be processed.

### 2.1.2 Sparse Grid based Density Estimation

Density Estimation is one of the central areas of statistics whose purpose is to construct an estimate of the probability density function $\hat{p}$ describing the distribution of the observed data set $S$ [33]. This density function can be in turn used to visualize, represent or extract information of the data [28] or as a tool for other Machine Learning methods.

Density Estimation Methods can be either parametric or non-parametric. Parametric Density Estimation Methods assume that the data follows the structure of a certain family of known density functions and that the density could be estimated by estimating the corresponding parameters describing it [33] (e.g. $\mu$ and $\sigma^2$ in the case of the Gaussian distribution). On the other hand, non-parametric Density Estimation Methods make no such assumptions and therefore, no parameters are estimated. One of the most widely used non-parametric methods is the Kernel Density Estimation [28] defined as:

$$\hat{p}(x) = \frac{1}{M} \sum_{i=1} K(\frac{x - x_i}{\sigma^2}) \tag{2.14}$$

In this method, the density function is approximated as a linear combination of all of the possible kernel functions $K$, centered in all of the points $x_i \in S$, with $M$ being the cardinality of $S$. Using this approach however, requires that all possible kernel functions be considered in the calculation and this number scales with the number of data points, making this method inefficient when handling large data sets.

Peherstorfer overcomes this issue by proposing a Sparse Grid based Density Estimation method in [28]. Given a data set $S = \{x_1, ..., x_M\}$ with real density $f(x) \in V$ and an initial guess of the estimated density $\hat{p}_\epsilon(x)$, we are looking for the estimated density estimation function $\hat{p}(x) \in V$ such that:

$$\hat{p}(x) = \underset{f(x) \in V}{\arg\min} \int_\Omega (f(x) - \hat{p}_\epsilon(x))^2 dx + \lambda ||\Delta f||^2_{L_2} \tag{2.15}$$

Note that the second term of the sum is a regularization term imposing a smoothness constraint and $\Delta$ is the regularization operator.

We set then the initial guess of the estimated density to $\hat{p}_\epsilon(x) = \frac{1}{M}\sum_{i=1}^{M}\phi(x_i)$, with $\phi(x_i) \in \Phi_l$ and $\Phi_l$ being the hierarchical basis of the sparse grid space $V_l$ of level $l$. We obtain the variational equation [17] of 2.15 and after some transformations our optimization problem turns into finding $\hat{p}(x)$ such that:

$$\int_\Omega \hat{p}(x)\phi(x)dx + \lambda \int_\Omega \Delta\hat{p}(x)\Delta\phi(x)dx = \frac{1}{M}\sum_{i=1}^{M}\phi(x_i) \tag{2.16}$$

holds $\forall\phi \in \Phi_l$.

The estimated density function is then expressed as a linear combination of the hierarchical basis functions of the sparse grid space $V_l$ of level $l$

$$\hat{p}(x) = \sum_{l,i}\alpha_{l,i}\phi_{l,i}(x) \tag{2.17}$$

Substituting 2.17 in 2.16 reduces the latter to a system of equations of the form:

$$(R + \lambda C)\vec{\alpha} = \vec{b} \tag{2.18}$$

with $R_{ij} = (\phi_i, \phi_j)_{L_2}$, $C_{ij} = (\Delta\phi_i, \Delta\phi_j)_{L_2}$ and $b_i = \frac{1}{M}\sum_{j=1}^{M}\phi_i(x_j)$ which can be then easily solved to find $\vec{\alpha}$.

## 2.2 Clustering

Clustering refers to the unsupervised Machine Learning method, whose objective is to identify disjointed groups in a series of data points based only on their internal structure. These groups are known as clusters. Formally it can be defined as [3]:

Given a data set $D = \{x_1, x_2, ..., x_n\}$ with $x_i \in \mathbb{R}^d$, a simple clustering of $D$ is defined as $C = \{C_1, C_2, , ..., C_k\}$ such that:

$$\cup C_i = D$$

$$C_i \cap C_j = \emptyset \Leftrightarrow C_i \neq C_j$$

There is not a unique way of doing clustering. For example, DBSCAN [4] assigns clusters to areas of high density, while K Means [21] iteratively finds clusters by calculating the centroids of the current clusters and reassigning the points to the nearest one. This causes different algorithms to return different results for the same data set.

### 2.2.1 Sparse Grid based Clustering

One of the applications of Sparse Grid Methods developed by Peherstorfer at [28], is the clustering of data points using Sparse Grid based Density Estimation Methods. In this algorithm, clusters are defined as groups of data points belonging to high density areas. On the other hand, data points in low density areas are not assigned to any cluster at all, but rather classified as noise. This makes this algorithm a density based clustering method. Peherstorfer's Clustering Algorithm is summarized in the following steps.

Given a data set $S = \{x_1, ..., x_n\}$ :

1. Construct a nearest neighbors' graph $G = (S, \xi)$ to represent the similarities of the points in $S$.

2. Employ a Sparse Grid based Density Estimation model to calculate the probability density function $p$ describing the density distribution of $S$.

3. Create subgraph $\hat{G} = (\hat{S}, \hat{\xi}) = (S \setminus \tilde{S}, \xi \setminus \tilde{\xi})$ with $k$ connected components by deleting vertices $\tilde{S}$ and related edges $\tilde{\xi}$ at which the estimated density function $p$ evaluates to values below a given threshold $\epsilon$.

4. Depending on their component, assign labels $y_1, ..., y_M \in \{1, ..., k\}$ to the remaining data points in $\hat{S} = \{\hat{x}_1, ..., \hat{x}_M\}$. Previously deleted points are labeled as noise.

### 2.2.2 The nearest neighbors' problem. The Vantage Point Tree

Generating a nearest neighbors' graph in a naive manner is a very expensive task, since the running time complexity lies unfortunately in O($n^2$). Generating our graph in such a way would render the benefits of Sparse Grid Methods useless. Fortunately, the problem has been amply studied and many algorithms and data structures have been developed in order to circumvent this issue [10] [26] [37] [38]. Among all of these data structures, the Vantage Point Tree is the one that was selected for our specific problem.

Vantage Point Trees have certain advantages over some other nearest neighbor data structures in that they tend to perform relatively better than most of them in terms of efficiency when handling large high dimensional data sets [20]. Additionally, they are simple to implement, they can be generated and updated in an online manner and they have a space complexity of O($n$) with $n$ being the number of data points.

Vantage Point Trees were introduced by Yianilos [37] as a data structure used to partition and index a metric space based on distances to specific points in the space (the vantage points) in order to perform efficient nearest neighbors' queries. These

vantage points are taken from the data itself at random (although the selection can be optimized) and the partition is done in such a way that, for every selected vantage point, half of the points remain inside an hypersphere, while the other half remain outside. The radius is determined by the median of all distances from the vantage points to the rest.

A Vantage Point Tree is actually a binary tree structure. Each node contains the vantage point itself and the median distance to all of the processed points. The left child points to the subtree containing all of the points whose distance to the vantage point is less than the median, while the right child points to the one containing all of the points greater or equal than the median.

The pseudocode to build a Vantage Point Tree in $O(n \log n)$ running time can be found in Algorithm 1:

**Algorithm 1** VpTree Creation

```
1:  function makeVpTree(S)
2:      if S == ∅ then
3:          return null
4:      node ← new Node()
5:      node.vantagePoint ← S.first()
6:      node.μ ← Median_{x∈S} distance(vantagePoint, x)
7:      L ← {x ∈ S − {p} | distance(vantagePoint, x) < μ}
8:      R ← {x ∈ S − {p} | distance(vantagePoint, x) ≥ μ}
9:      node.right = makeVpTree(R)
10:     node.left = makeVpTree(L)
11:     return node
```

Searching for the nearest neighbors in a Vantage Point Tree is similar to searching in balanced binaries trees, since a Vantage Point Tree is approximately balanced due to the way the tree is built. This means that the running time of each nearest neighbor query lies approximately in $O(\log n)$. Therefore, the running time of searching $k$ nearest neighbors lies approximately in $O(k \log n)$.

During the search, we keep track of the distance to the farthest point so far found in a variable named $\tau$. For every node visited, we calculate the distance $d$ from our target point $q$ to the vantage point of the node and add it to a list. If $d < \tau$, we add the vantage point to our list and update the value of $\tau$. If there are already $k$ points in our list we delete the one that is farthest away from $q$.

We compare now the $d$ to the radius $\mu$ of the ball which partitions the space. If $d < \mu$ we will look first on the left child else we will look first on the right and we will repeat the same process.

It could be however, that there might be a nearest neighbor on the opposing child. In the case that we have looked on the left child first, if $\tau + d \leq \mu$ then it means there is no way that another nearest neighbor exists outside the hypersphere and therefore we skip the right child. A similar principle applies when looking on the right child first. The difference lies in that the condition to be fulfilled is $\tau - d \geq \mu$. Otherwise, we must continue looking on the corresponding opposite child and repeat recursively the same process. The pseudocode for the search algorithm can be found in Algorithm 2:

**Algorithm 2** VpTree Nearest Neighbor Search

```
1:  procedure SEARCH(node, target, nearestNeighbors, k, τ)
2:      if node == null then
3:          return
4:      d : = distance(target, node.vantagePoint)
5:      if d <τ then
6:          τ ← d
7:          if nearestNeighbors.size() > k then
8:              nearestNeighbors.popFarthestNeighbor()
9:          nearestNeighbor.insert(node.vantagePoint)
10:     if d < node.μ then ▷ If inside the ball we force the search on the left child first
11:         if d-τ ≤ node.μ then
12:             searchNeighbor(node.left, target, nearestNeighbors, k, τ)
13:         if d+τ ≥ node.μ then
14:             searchNeighbor(node.right, target, nearestNeighbors, k, τ)
15:     else                ▷ If outside the ball we force the search on the right child first
16:         if d+τ ≥ node.μ then
17:             searchNeighbor(node.right, target, nearestNeighbors, k)
18:         if d-τ ≤ node.μ then
19:             searchNeighbor(node.left, target, nearestNeighbors, k)
```

### 2.2.3 Hierarchical Clustering

One aspect of Peherstorfer's Sparse Grid based Clustering Algorithm, is the that the number of clusters is determined by the hyperparameters, these being the number of nearest neighbors $k$ and the minimum density threshold $\epsilon$. As remarked by Fischer at [8], one obvious drawback of this Flat Clustering Algorithm is that the best minimum density threshold is actually a data dependent parameter. To overcome this issue, Fischer proposes an augmentation to Peherstorfer's Clustering Algorithm, which results

in the creation of a hierarchy of clusters for different values of $\epsilon$. This augmentation was originally designed in the context of identifying related topics in recommendation systems, it can however be easily extended to a much more general case.

Fischer's augmentation consists of an iterative process. We run Peherstorfer's Clustering Algorithm in an $n_{steps}$ number of steps, while varying the minimum density threshold $\epsilon$ from a range defined by $[\rho_{min}, \rho_{max}]$ and also while increasing the threshold each step by $\epsilon_{step}$. For all of these iterations, we keep a fixed value of $k$, although one must be aware that this parameter also plays a significant role in the clusters' identification. For each iteration, we store the clusters in a tree-like structure and check if from a previous cluster new clusters are generated. These are appended as children to the cluster whom they belonged in a previous iteration. If these children are significantly dissimilar to their parent cluster, the parent cluster is then deleted and its children take its place in the hierarchy. This dissimilarity is measured as the ratio of the connectivity of the child cluster with its parent and the connectivity of the points of the parent cluster. To determine if a cluster should be replaced or not, Fischer defines the split threshold $t$. A ratio under this threshold indicates that the child is sufficiently dissimilar to its parent, and therefore, a replace operation is executed. For higher density thresholds, this will converge to a certain hierarchical structure, which in turn can be analyzed to define the best minimum density threshold which will yield the best clustering. Note that points which do not meet the minimum density threshold $\rho_{min}$ or which are not assigned to a final cluster are labeled as noise. The pseudocode for the hierarchal cluster generation routine and the cluster split routine as stated by Fischer at [8] can be found in Algorithms 3 and 4, respectively.

**Algorithm 3** Fischer's Hierarchical Cluster Generation

**procedure** CLUSTER_HIERARCHICAL($S, k, n_{steps}$)
    $\rho_{min} \leftarrow min(p(S))$ ▷ p(S) is the Sparse Grid Based Density Function applied over the data set S.
    $\rho_{max} \leftarrow max(p(S))$
    $\epsilon_{step} \leftarrow \frac{\rho_{max} - \rho_{min}}{n_{steps}}$
    compute $G_k$
    $G_{k,\rho_{min}} \leftarrow G_k$
    **for** i = 1 to $n_{steps}$ **do**
        $\epsilon \leftarrow \rho_{min} + i \cdot \epsilon_{step}$
        Calculate $G_k$ and get the set of connected components $C$
        updatedClusters $\leftarrow \emptyset$
        **for** $C_j \in C$ **do**
            point $\leftarrow c \in C_j$
            parent $\leftarrow point.getCluster()$
            parent.add_Child($C_j$)
            updatedClusters $\leftarrow$ updated $\cup$ parent
        **for** parent in updatedClusters **do**
            **if** parent.num_children > 1 **then**
                **if** split(parent, $G_{k,\epsilon-\epsilon_{step}}$) **then**
                    parent.get_parent().add_children(parent.get_children())
                    parent.get_parent().remove_child(parent)
            **else**
                parent.remove_children()

**Algorithm 4** Fischer's Cluster Split Routine

**function** SPLIT_CHILD(parent, child, $G = (V, E)$)
    $V_p \leftarrow$ parent.nodes()
    $V_c \leftarrow$ child.nodes()
    connections_parent $\leftarrow |\{e = \{i, j\} \in E, \forall i \in V_p \ , \forall j \in V_c\}|$
    max_connections_parent $\leftarrow \frac{1}{2}|V_p|(|V_p| - 1)$
    connectivity_parent $\leftarrow \frac{\text{connections\_parent}}{\text{max\_connections\_parent}}$
    connections_child_parent $\leftarrow |\{e = \{i, j\} \in E, \forall i, j \in V_p\}|$
    max_connections_child_parent$|V_c|(|V_p| - |V_c|)$
    connectivity_child_parent $\leftarrow \frac{\text{connections\_child\_parent}}{\text{max\_connections\_child\_parent}}$
    **return** $\frac{\text{connectivity\_child\_parent}}{\text{connectivity\_parent}} < t$

**function** SPLIT_CLUSTER(cluster, child, $G = (V, E)$)
    **for** child in cluster.children() **do**
        **if** split_child(cluster, child, G) **then**
            **return** True
    **return** False

### 2.2.4 Clustering Quality

Assessing clustering quality is not a straightforward task as with other Machine Learning Algorithms, like Classification or Regression, since one does not know a priori the real number of clusters found within the data. Different frameworks and schemes have been throughly studied and developed in the past [2] [14] [25] in order to determine cluster quality measures which indicate if a given clustering is adequate.

For the purpose of this thesis we will use the terms measure and metric as synonyms. Formally, a clustering quality measure is a function that maps pairs of the form $(\text{dataset}, \text{clustering})$ to some ordered set (the set of non-negative real numbers for example), so that the values reflect how good is that specific clustering [2]. Current metrics may follow one of these three approaches when assessing the quality of a clustering:

- External validity criteria: Results are validated against pre-specified clustering structures [15]. In other words, our data has previously been labeled with a cluster class to which we can compare the clustering against.
- Relative validity criteria: Results are validated against others generated by some other clustering algorithm [25].

- Internal validity criteria: Quality is validated in terms of measures that involve the data set itself [14]. The measures can be, but are not limited to, intra cluster measures, inter cluster measures, centroid based measures and mixed measures [25].

For the purpose of our implementation, we have selected the following clustering evaluation metrics:

- **Fowlkes-Mallows Index**: This metric was introduced as a tool to measure the similarity of two hierarchical clusterings $C$ and $K$ [9], so it can be considered that this measure is based on the relative validity criteria. The metric, defined as $B_k$, is derived from the matching matrix $M$ whose elements $m_{ij}$ represent the matching entries in both clusterings. In other words, element $m_{ij}$ represents the number of entries belonging to cluster $i \in C$ and cluster $j \in K$.

  The metric is calculated as:

$$B_k = \frac{T_k}{\sqrt{P_k Q_k}} \tag{2.19}$$

  where

$$T_k = \sum_{i=1}^{|C|} \sum_{j=1}^{|K|} m_{ij}^2 - n \tag{2.20}$$

$$P_k = \sum_{i=1}^{|C|} (\sum_{j=1}^{|K|} m_{ij})^2 - n \tag{2.21}$$

$$Q_k = \sum_{j=1}^{|K|} (\sum_{i=1}^{|C|} m_{ij})^2 - n \tag{2.22}$$

  The final score lies in the range $[0, 1]$, with 1 indicating a perfect matching and 0 no match at all.

- **V-Measure :** This is a metric based on the external validity criteria. It is an entropy based metric which explicitly measures how successfully the criteria of homogeneity and completeness have been satisfied [30]. Homogeneity refers to the condition that a clustering should assign only those data points that are members of a single class to a single cluster. In other words, a cluster should contain points of only one class. Completeness refers to the condition that a

clustering should assign all of the data points that are member of a single class to a single cluster. In other words, all of the points of a given class should be assigned to only one cluster.

Given the clustering $K$, the set of predefined labels $C$, the number of datapoints $N$ and $a_{ck}$ as the number of points in cluster $k \in K$ with predefined label $c \in C$, the V-Measure is obtained as follows:

The homogeneity score is calculated through the following entropy formulas:

$$h = \begin{cases} 1 & \text{if } H(C,K) = 0 \\ 1 - \frac{H(C|K)}{H(C)} & \text{else} \end{cases} \tag{2.23}$$

where

$$H(C|K) = -\sum_{k=1}^{|K|}\sum_{c=1}^{|C|}\frac{a_{ck}}{N}\log\frac{a_{ck}}{\sum_{c=1}^{|C|}a_{ck}} \tag{2.24}$$

$$H(C) = -\sum_{c=1}^{|C|}\frac{\sum_{k=1}^{|K|}a_{ck}}{N}\log\frac{\sum_{k=1}^{|K|}a_{ck}}{n} \tag{2.25}$$

The completeness score is calculated through the following entropy formulas:

$$c = \begin{cases} 1 & \text{if } H(K,C) = 0 \\ 1 - \frac{H(K|C)}{H(K)} & \text{else} \end{cases} \tag{2.26}$$

where

$$H(K|C) = -\sum_{c=1}^{|C|}\sum_{k=1}^{|K|}\frac{a_{ck}}{N}\log\frac{a_{ck}}{\sum_{k=1}^{|K|}a_{ck}} \tag{2.27}$$

$$H(K) = -\sum_{k=1}^{|K|}\frac{\sum_{c=1}^{|C|}a_{ck}}{N}\log\frac{\sum_{c=1}^{|C|}a_{ck}}{n} \tag{2.28}$$

The metric is computed as the harmonic mean of distinct homogeneity and completeness scores [30]

$$V_\beta = \frac{(1+\beta)\cdot h\cdot c}{(\beta\cdot h)+c} \tag{2.29}$$

The $\beta$ parameter is used to indicate the weight of the homogeneity and completeness score. Values greater than 1 favor the completeness score, while values less than 1 favor the homogeneity score. For the purpose of our implementation, both scores are weighted equally. The final score lies in the range of $[0, 1]$ with 1 indicating a perfect clustering.

- **Calinksi-Harabasz Index:** This is a metric based on the internal validity criteria. It is a variance ratio criterion [5] which measures the quality of a cluster in a similar way in which the F-statistic does when measuring the quality of a fitted regression model.

  For $N$ data points, $k$ clusters and clustering $C$, the Calinski-Harabasz Index is defined as:

$$CH = \frac{N-k}{k-1} \cdot \frac{BGSS}{WGSS} \tag{2.30}$$

  where $BGSS$ and $WGSS$ represent the between-cluster dispersion and within-cluster dispersion values respectively.

  The between cluster dispersion measures the dispersion of the clusters' centroids and is calculated as:

$$BGSS = tr(B_k) \tag{2.31}$$

  where $B_k$ is a weighted covariance matrix of the data set containing the cluster centroids $c_q$, with mean $c_E$ (this being the center of the data). The weights are given by the number of data points $n_q$ belonging to each cluster

$$B_k = \sum_{q=1}^{k} n_q (c_q - c_E)(c_q - c_E)^t \tag{2.32}$$

  The within cluster dispersion measures the dispersion of the data points in each cluster and is calculated as:

$$WGSS = tr(W_k) \tag{2.33}$$

  where $W_k$ is the sum of the covariance matrices of each cluster $C_q$, with the mean $c_q$ being the centroid of each cluster.

$$W_k = \sum_{q=1}^{k} \sum_{\forall x_i \in C_q} (x_i - c_q)(x_i - c_q)^t \tag{2.34}$$

  The score lies in the range of $[0, \infty)$ with higher values indicating a better clustering.

- **David-Bouldin Index:** This is a metric based on the internal validity criteria. This metric is used to measure the similarity of clusters [6]. The less similar these clusters are, the better the clustering obtained.

  This metric is a function of the ratio of the sum of within-cluster scatter to between-cluster separation [24]. Given a number of clusters $K$, the David-Bouldin Index is defined as:

  $$DB = \frac{1}{K} \sum_{i=1}^{K} max_{i \neq j} R_{ij} \tag{2.35}$$

  where

  $$R_{ij} = \frac{S_i + S_j}{d_{ij}} \tag{2.36}$$

  with $S_i$ representing the scatter of the cluster $C_i$ with centroid $z_i$

  $$S_i = \frac{1}{|C_i|} \sum_{x \in C_i} ||x_i - z_i|| \tag{2.37}$$

  and $d_{ij}$ the distance between clusters $C_i$ and $C_j$ with centroids $z_i$ and $z_j$ respectively

  $$d_{ij} = ||z_i - z_j|| \tag{2.38}$$

  This score lies in the range $[0, \infty)$ with 0 representing the best possible clustering.

## 2.3 The SG++ Datamining Pipeline

The SG++ Toolbox is a universal open-source toolbox written in C++ [31], which provides the necessary interfaces needed to utilize Sparse Grid Methods with minimum effort. The toolbox is comprised of several different components each specifically developed to solve a specific problem like Partial Differential Equation Solvers, Function Interpolation, Uncertainty Quantification, etc [1].

The SG++ Datamining Pipeline is the component of the SG++ Toolbox in charge of providing developers the interfaces to train, execute, evaluate and visualize Sparse Grid based Machine Learning Methods. Currently, the pipeline gives support to four different Machine Learning Models: Regression, Density Estimation, Classification [11] and Clustering, the last being the one developed in this thesis.

The pipeline structure is comprised of the following modules:

- **Datasource:** Loads the data from a given data source and feeds it to the fitter to train the model.
- **Fitter:** Trains the model itself through Sparse Grid Methods. The training is done in an online manner across different batches of data and in multiple epochs if configured so.
- **Scorer:** Delivers a metric which measure the quality of a given model. The type of metric used will depend on the model trained and the configuration given by the user.
- **Hyperparameter Optimizer [19]:** Obtains the optimal training hyperparameters for a model through methods like Bayesian Optimization.
- **Visualization [1]:** Delivers an output in either csv or JSON format, which can be later be used as the input of a graphic library in order for the end user to visualize their trained models and data. The JSON output is designed so that it can be processed by the plotly library [18] to generate a graphical output.
- **PostProcessing:** Module developed in this thesis to do extra processing on the models which are either not directly related to sparse grid numerical techniques or which are incompatible with the online training procedure of the fitter.

Additionally, the pipeline is configured through a configuration file following the JSON format. This file specifies where the data is to be obtained, the parameters to train and to evaluate the model and the parameters to run the visualization module. A complete list of configurations can be found at [32].

# 3 Implementation

## 3.1 Clustering Configurations

We can deduce from sections 2.2.1 and 2.2.3, that there exist five configurable hyper-parameters that need to be given in order to train a clustering model. These are the number of nearest neighbors, the minimum and maximum density threshold for graph pruning, the split threshold to determine if a cluster should be replaced and the number of steps in which to run Fischer's Algorithm to generate the Hierarchical Clustering.

All of these configurations are to be given in the JSON configuration file as an extra dictionary with the key "clustering" within the fitter configuration dictionary. Additionally, two extra parameters were implemented for the purpose of configuring the storing of information related to the Cluster Hierarchy. Table 3.1 shows a general description of the implemented parameters:

| Attribute Name | Attribute Type | Valid value range | Default Value |
|---|---|---|---|
| noNearestNeighbors | Positive Integer | $[1, \infty)$ | 10 |
| minDensityThreshold | Float | $[0, 1]$ | 0.1 |
| maxDensityThreshold | Float | $[0, 1]$ | 0.5 |
| splitThreshold | Float | $[0, 1]$ | 0.4 |
| steps | Positive Integer | $[1, \text{inf})$ | 10 |
| storeHierarchy | Boolean | True or False | False |
| outputDirectory | String | Any valid directory | The executable file's directory |

Table 3.1: Clustering Configuration Parameters

Special remarks have to be mentioned regarding the density thresholds. Density Estimation Models generated by the pipeline, once evaluated, return values way outside the range $[0, 1]$, even negative values. Since the end user will not know a

priori the range of values delivered by the Density Estimation model, it was decided that the density threshold parameters accept only values in the range $[0, 1]$. When evaluating the densities and pruning the graph, these values will be interpreted as the percentage of the maximum density threshold value delivered by the Density Estimation model. However, data points with negative density values will always be deleted in the pruning step. Note that the Flat Clustering Algorithm can be run by setting both density thresholds to the same value and the number of steps to 1, resulting in a Hierarchy Tree of only one level.

Regarding the default values for the density thresholds, these were chosen due to the fact that the range defined by them is the one most likely to contain the most significant information related to the clustering. Densities lower than 10% of the maximum density are highly likely to describe only noisy data, while analysis done on extremely high densities (more than the half of the maximum density) may result in extremely small clusters and therefore, in overfitting. The default value for the split threshold is the one used by Fischer at [8] where it was shown that this value delivers good results in general.

## 3.2 Vantage Point Tree Implementation

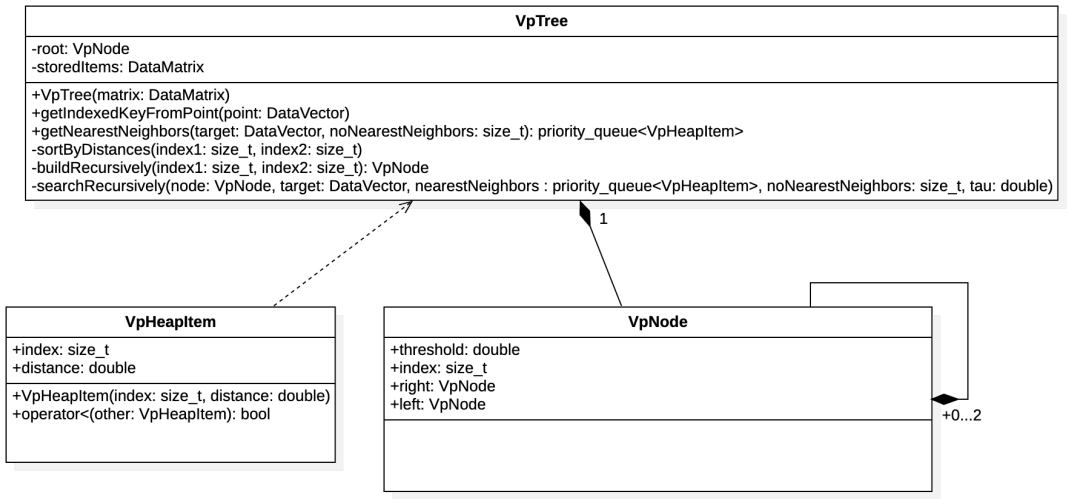



Figure 3.1: Vantage Point Tree Implementation Design

Figure 3.1 shows a summarized version of the implemented classes for the Vantage Point Tree. Our Vantage Point Tree implementation is largely based on the one created by Steve Hanov's at [16], with changes made to adapt ourselves to the structure of the SG++ Datamining Pipeline and to make use of the data structures provided by the SG++ Toolbox. The VpNode class is our basic structure unit to create the tree, while the VpTree contains the reference to the root of the tree and gives access to the rest of the structure.

One can notice that the VpNode class contains no attribute to store the coordinates of the Vantage Point. All of the Vantage Points will be stored instead inside the VpTree class using the DataMatrix attribute *storedItems*. The attribute *index* of each VpNode object will point to the row of this matrix where the corresponding Vantage Point is stored. Additionally, the value of this *index* attribute will be propagated to the Graph and HierarchyTree classes to maintain constant and unique references to all of the points used during the training of the model.

The creation of the tree is implemented in the constructor of the VpTree class and the recursive method *buildRecursively*. The former triggers the recursive call while the latter implements the pseudocode stated in Algorithm 1. There is however a small difference in our implementation. The method *buildRecursively* receives the indexes which describe the range of rows of the *storedItems* attribute to process in the current call. In every call, the points within this range are reordered based on their distances to the vantage point. We split the range into two subranges of equal length, both representing respectively the indexes of the points which were ordered before and after the point with the median distance to the Vantage Point. The recursive call is then made by calling the method *buildRecursively* twice with both subranges respectively until the length of the subrange equals to 1. This was done in order to save memory space during the construction of the tree and to index the data as to easily identify it through the whole training of the model.

The search for the $k$ nearest neighbors is implemented with a similar execution structure in the methods *getNearestNeighbors* and *searchRecursivey*, the former being the trigger of the process and the latter being the recursive algorithm defined in Algorithm 2. One detail which can be observed in both methods is the use of a priority queue instead of a list in order to store the current nearest neighbors found. This was decided in order to speed up the update of the current nearest neighbors. Deletion in a list is in the worst case in $O(n)$, while in a priority queue is in $O(1)$ if the queue is defined in a proper way. Additionally, a priority queue can order the elements automatically every time a new element is inserted and does it in $O(\log n)$ time.

To make use of the priority queue, the VpHeapItem class was implemented. This class contains the essential information to identify a nearest neighbor inside the Vantage Point Tree, namely the *index* pointing to the row in the attribute *storedItems* of the VpTree

class and the *distance* to the target, whose nearest neighbors are being searched. In order for the priority queue to order the elements by the distance attribute, the operator "<" had to be defined for this class in a way that the order of a VpHeapItem is determined by the value of the *distance* attribute.

Finally, a search method named *getIndexedKeyFromPoint* was added in our VpTree class in order to recover the index of a given data point. This was implemented mainly for evaluation purposes, since the index given by the VpTree class will also indicate the final label assigned to this data point in the the HierarchyTree class. The way the tree is built will result in an approximately balanced structure. In consequence, our search will remain approximately in $O(\log n)$.

## 3.3 Graph Implementation

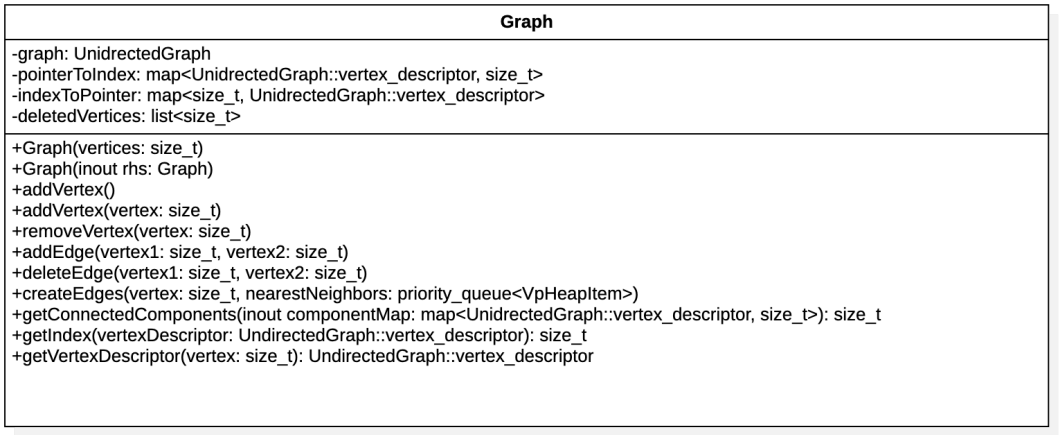



Figure 3.2: Graph Implementation Design

For the graph implementation we used some functionalities provided by the Boost Graph Library [35]. These include, but are not limited to, interfaces to define all graph structure operations, a plethora of graph algorithms like the connected component detection and the possibility to customize properties for vertices and edges.

The Boost Graph Library offers three different container types for vertex and edge storage. These are vectors, lists and sets containers. Using one or the other has an impact on the performance of certain operations and algorithms. We decided that the best containers for our graph implementation would be a list container for the vertices

and a set container for the edges, due to the following reasons:

- Deletion of vertices using list containers has a constant amortized running time in comparison to vector containers, whose deletion running time is in $O(V + E)$.
- Iterators of vertices using list containers don't get invalided after deleting vertices.
- Set containers automatically detect bidirectional edges between two vertices in undirected graphs, and do not add them if an edge with the same endpoints already exists. This saves memory consumption when creating the connections.

All of the graph algorithms needed for the clustering model have been implemented in our Graph class, whose summarized design can be seen in Figure 3.2. The attribute *graph* represents the Boost Graph instance. The methods *addVertex*, *removeVertex*, *addEdge*, *deleteEdge* and *getConnectedComponents* are actually wrapper methods, which run the equivalent Boost Graph algorithms on the *graph* attribute every time they are called. The method *createEdges* receives the results of the nearest neighbors' query executed by the VpTree class and inserts the nearest neighbors' connections.

One particularity in our design is the use of two map attributes, *indexToPointer* and *pointerToIndex*. The former maps the index of a certain vertex to a memory location in the list container, while the latter delivers the inverse mapping. These were implemented due to the fact that elements of a list in C++ do not have a random access operator and can only be accessed through the memory location of the element. Both maps are built when creating the graph using the indexes given by the VpTree class and are automatically updated every time a new point is added or deleted. External classes have access to both maps through the methods *getIndex* and *getVertexDescriptor* respectively.

Each graph will also keep track of the points which were previously deleted, through a list containing the indexes of the previously deleted vertices. We need to rebuild the indexes again when copying a graph, and this lists contains the information of those who must be skipped so that the references to the points are maintained correctly.

## 3.4 Hierarchical Clustering Implementation

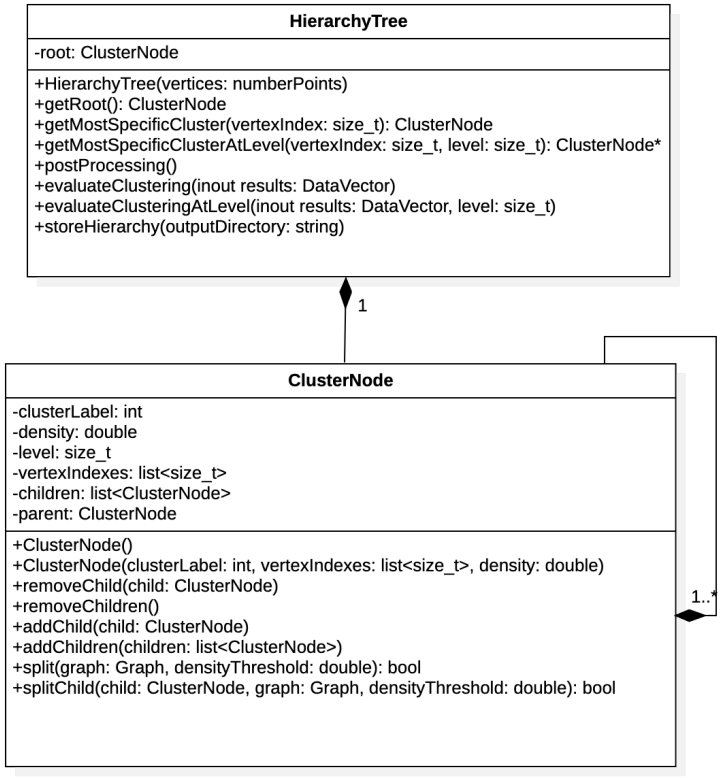



Figure 3.3: Hierarchical Clustering Tree Implementation Design

Figure 3.3 shows the summarized design of the data structure used to process and store the hierarchy of clusters obtained during the training of the model.

The ClusterNode class represents one node inside the Hierarchy Tree. Apart from the list of vertex indexes and the references to its children and parent nodes, additional attributes have been added to store the information of the cluster label assigned, the density threshold at which this node was created and the level in the tree in which the node was inserted. Fischer's splitting routine described in Algorithm 4 is implemented in the methods *split* and *splitChild*. The former can be subject to further parallelization in order to speed up the hierarchy generation. Of all the children, only one needs to fulfill the split condition in order for the node to be replaced. By parallelizing the *split* method, concurrent executions of the *splitChild* can be run and if one of them fulfills

the split condition, the others can be interrupted and the process can continue.

Similarly to the VpTree class, the HierarchyTree class contains the reference to the root of the tree, giving it a unique access point for traversal. Since the Hierarchical Tree is itself a representation of the final clustering, additional methods were implemented for evaluation purposes. These methods are *evaluateClustering* and *evaluateClusteringAtLevel*. Both deliver the clustering labels inside a DataVector object, following the indexation given by the VpTree. The difference between these two methods lies that the former delivers the label found at the deepest level of the tree, while the latter delivers the labels at the given level.

The method *postProcessing* is a special method used to assign unique cluster labels to each node after the hierarchy of clusters has been completely processed. Per default, the root will have the label $-1$, representing all the points classified as noise. In practice this means that all points that are found in the root, but not in subsequent levels are assigned as final label the noise label, which means that either they didn't satisfy the minimum density threshold specified by the user or that a node in the first level was replaced by its children in a subsequent iteration.

Finally, the method *storeHierarchy* will store the general information of the Hierarchy Tree in JSON format, for the purpose of saving it and for further analysis.

## 3.5 Fitter Implementation

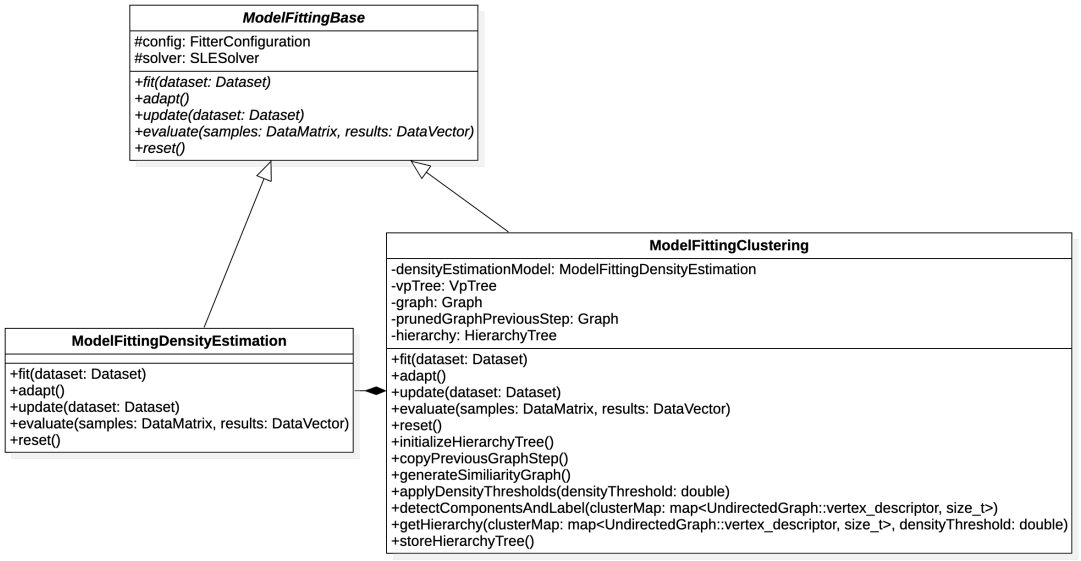



Figure 3.4: Fitter Implementation Design

We have based ourselves on the implementation of the fitter for Classification Models at [11] to create the one for Clustering Models. Figure 3.4 shows a summarized design containing the main elements of our fitter implementation inside the ModelFittingClustering class. Similarly to the fitter for Classification Models, our model extends from the abstract class ModelFittingBase and contains an attribute *densityEstimationModel* of the class ModelFittingDensityEstimation, which is the one used to train and to evaluate the Density Estimation model described in Section 2.2.1. Additional attributes relevant to the fitter include:

- The *vpTree* attribute of the VpTree class used to store our Vantage Point Tree and to do nearest neighbors' queries.
- The *graph* and *prunedGraphPreviousStep* attributes of the Graph class. The former contains the graph in its latest status during the generation of the hierarchy and the latter contains a copy of *graph* at the end of the previous iteration step.
- The *hierarchyTree* attribute of the HierarchyTree class used to store the Hierarchical Clustering from Fischer's Algorithm described in Section 2.2.3

Regarding the extended methods *fit*, *adapt*, *update* and *reset*, they will execute the method of the same name of the attribute *densityEstimationModel*. The behavior of the method *evaluate* will depend on the current status of the training of the model. If the Clustering Hierarchy has not been generated, the method will return the density values of the Density Estimation model, else, the method will return the cluster labels found inside the Hierarchy Tree. To evaluate the cluster label of a given data point, we first search in the Vantage Point Tree to obtain its corresponding index. If the point is not found, the noise label ($-1$ in our case) will be assigned, else, we search in the Hierarchy Tree the cluster at the deepest level of the tree to which the point belongs. Once found, we obtain the label corresponding to this cluster and return it.

The rest of the methods will be executed inside the Post Processing Module. These encompass all of the steps required to generate the Hierarchical Clustering using Fischer's Algorithm. These are:

- *initializeHierarchyTree*: Method used to initialize the *hierarchyTree* attribute, by creating the root of the tree.
- *copyPreviousGraphStep*: Method used to copy the content of the *graph* attribute into the *prunedGraphPreviousStep* attribute at the end of every iteration.
- *generateSimilarityGraph*: Method which builds the Vantage Point Tree inside the *vpTree* attribute, does the nearest neighbors' queries and creates the nearest neighbor graph inside the *graph* attribute. This method can be subject to parallelization since the nearest neighbors' queries for each point are independent of each other.
- *applyDensityThresholds*: Method which evaluates the trained Density Estimation model and deletes from the *graph* attribute the vertices associated to the data points which do not meet the specified minimum density threshold.
- *detectComponentsAndLabel*: Method used to trigger the connected components algorithm of the *graph* attribute and to assign the corresponding labels to the vertices.
- *getHierarchy*: Method implementing the two inner loops of Fischer's Algorithm described in Algorithm 3 (lines 11 to 22) to process and update the Hierarchical Clustering. The latter part of this method can be parallelized by running the split routine concurrently in all of the updated Cluster Nodes in order to generate the hierarchy faster.
- *storeHierarchyTree*: Method used to store the info of the Hierarchy Tree by calling the method *storeHierarchy* of the *hierarchyTree* attribute.

## 3.6 Metrics Implementation

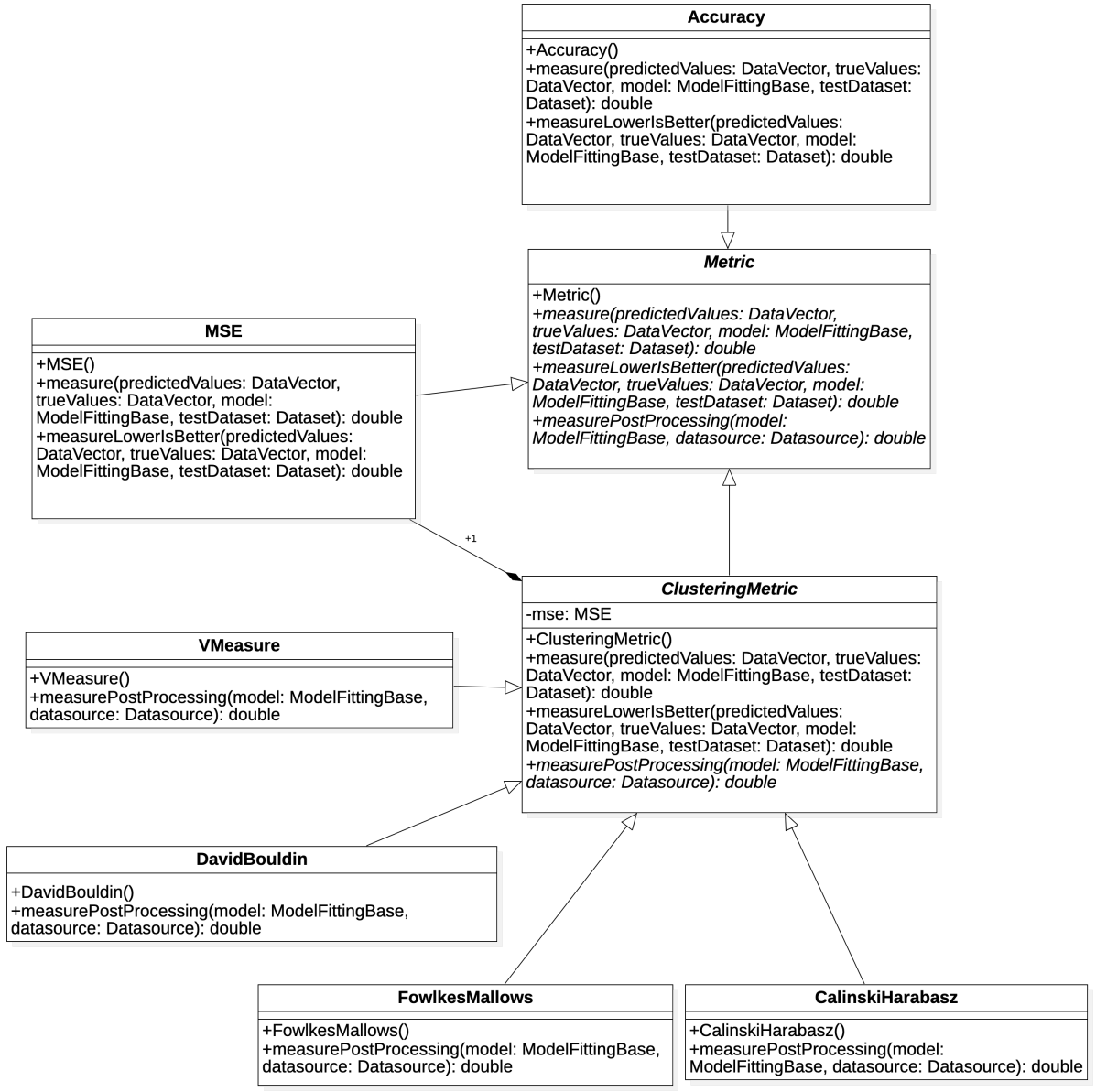



Figure 3.5: Implementation Design of the Scorer Module with the new structure for the Clustering Metrics

The implementation of the metrics to evaluate the clustering of our method required certain changes to be made in the structure of the pipeline. In comparison to the metrics already implemented for the other models, clustering metrics require the data set as a whole for their calculation. Additionally, the metric must be calculated once the whole training process is over. However, the quality of the Density Estimation model must be also evaluated during the training.

In order to fulfill both requirements, we have created a new abstract class called ClusteringMetric, from where all of the classes for the metrics described in section 2.2.4 are derived. Figure 3.5 shows a summarized version of the design. This class contains an attribute called *mse* of the MSE (Mean Squared Error) class. During the online training part, the methods *measure* and *measureLowerIsBetter* of the ClusteringMetric class will call the methods of the same name of the *mse* attribute in order to evaluate the quality of our Density Estimation model.

The method *measurePostProcessing* will be the one called outside the online component of the pipeline and will deliver the final score for the clustering itself. To keep the structure of the Scorer Module, the *measurePostProcessing* method was also added in the classes of the metrics used to evaluate other models, however its calling will only execute the *measure* method of the corresponding metric.

A special remark has to be made about the handling of noise when calculating the value of a metric. Noise is not a cluster per se, however we consider it as a one for quality measurement purposes without generating misleading results. In the case of the Fowlkes-Mallows Index and the V-Measure, the predefined labels can also contain noise labels to compare against.

In the case of the Calinski-Harabasz Index, noise is naturally penalized if we consider it as a separate cluster. The bigger the set of noisy points is, the smaller the value of the in between cluster dispersion will be. This occurs due to the fact that the centroid of the noisy points will be closer to the centroid of the whole dataset and therefore, the component of the variance will be smaller. This will cancel the large weight given by the number of noisy points and will result in a small between cluster dispersion value and therefore, a smaller score.

Something similar happens in the case of the David-Bouldin Index. If the noise points are irregularly distributed through all over the data set, the similarity measures between the noise and the real clusters will increase, penalizing the score. On the other hand, if the noisy points are concentrated in just one single area (a possible result when clusters are not dense enough), then the similarity with the real clusters will decrease, improving the score.

## 3.7 Visualization Implementation

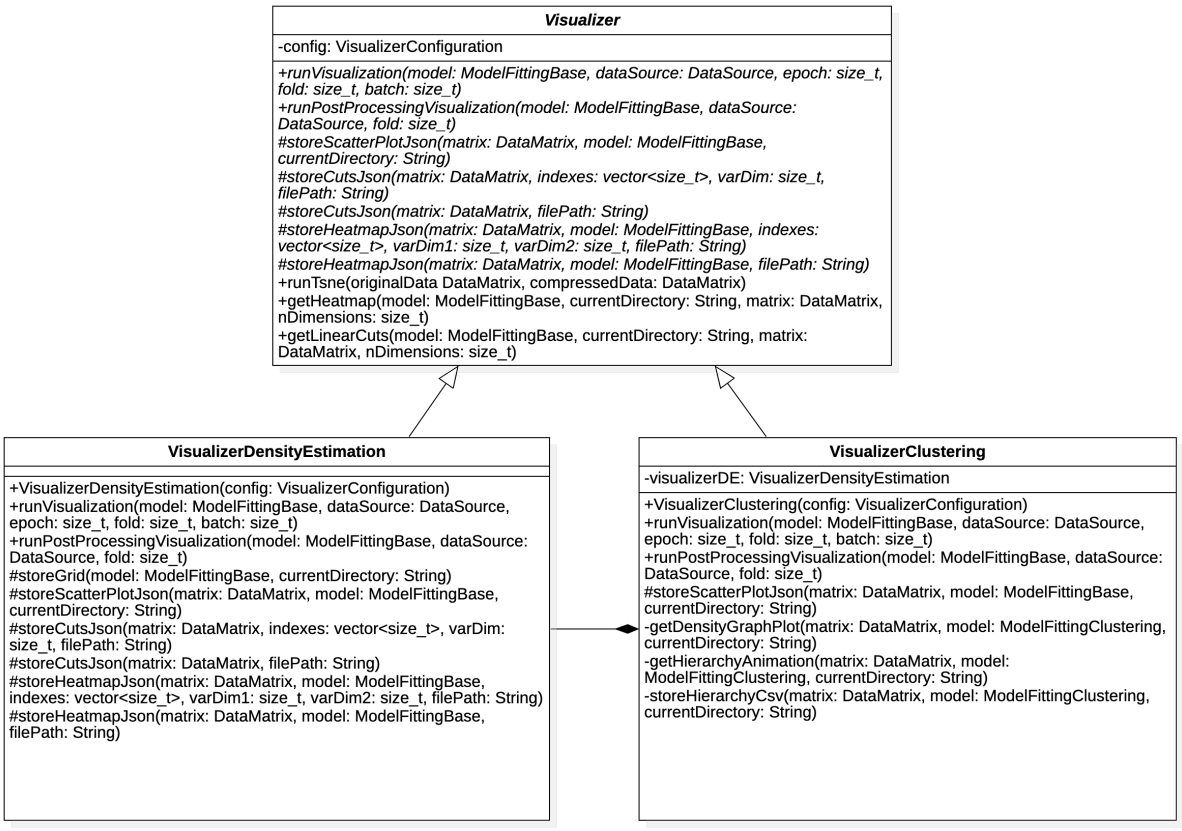



Figure 3.6: Implementation Design of the Visualization Module for Clustering Models

Regarding the Visualization Module, a new class called VisualizerClustering was created to visualize the Clustering Models generated by the pipeline. The summarized design of this class is shown in Figure 3.6. Similarly to the ModelFittingClustering class, the VisualizerClustering class was implemented in a way that it consists of an online and an offline visualization process.

The online process, implemented in the method *runVisualization*, is in charge of generating all of the outputs corresponding to the Density Estimation model. These outputs are the Density Estimation heatmaps and the linear cuts [1] generated by the attribute *visualizerDE* of the class VisualizerDensityEstimation. The offline process however, is run in a new method called *runPostProcessing*, which is called by the Post

Processing Module. The addition of this method was also reflected in the rest of the already implemented models in order to maintain the structure of the module. In those models, the original scatterplots [1] are now processed by this method instead of being generated online during the training of the models.

In Clustering Models, the *runPostProcessing* method delivers two outputs designed to generate scatterplots. If the data dimensionality is higher than 2, the t-SNE component of the Visualization Module [1] is first applied to find a 2-dimensional embedding which can be easily visualized.

The outputs themselves will depend on the format specified. If the format given is CSV, the first output will be a csv file containing either the points or the embedding and their evaluated densities, while the second one, generated by the method *storeHierarchyCsv*, will deliver another csv file containing either the points or the embedding with their corresponding labels of the most specific cluster found within the Hierarchy Tree.

If the format given is JSON, the method *getDensityGraphPlot* is executed for the first output and a JSON file is generated. This file purpose is to be used to generate a scatterplot in plotly consisting of either the data points themselves or the embedding, with both the evaluated densities from the Density Estimation model and the lines showing the nearest neighbors' graph connections. For the second output, the method *getHierarchyAnimation* is executed and generates a JSON output whose purpose is to create a dynamic animation plot in plotly, showing the clusters generated at each level, their corresponding pruned graphs and the clusters generated in the previous level. This last element was added to easily visualize the parent cluster from where the new ones are generated. Examples of both of this plots can be seen in Figures 3.7 and 3.8, respectively.

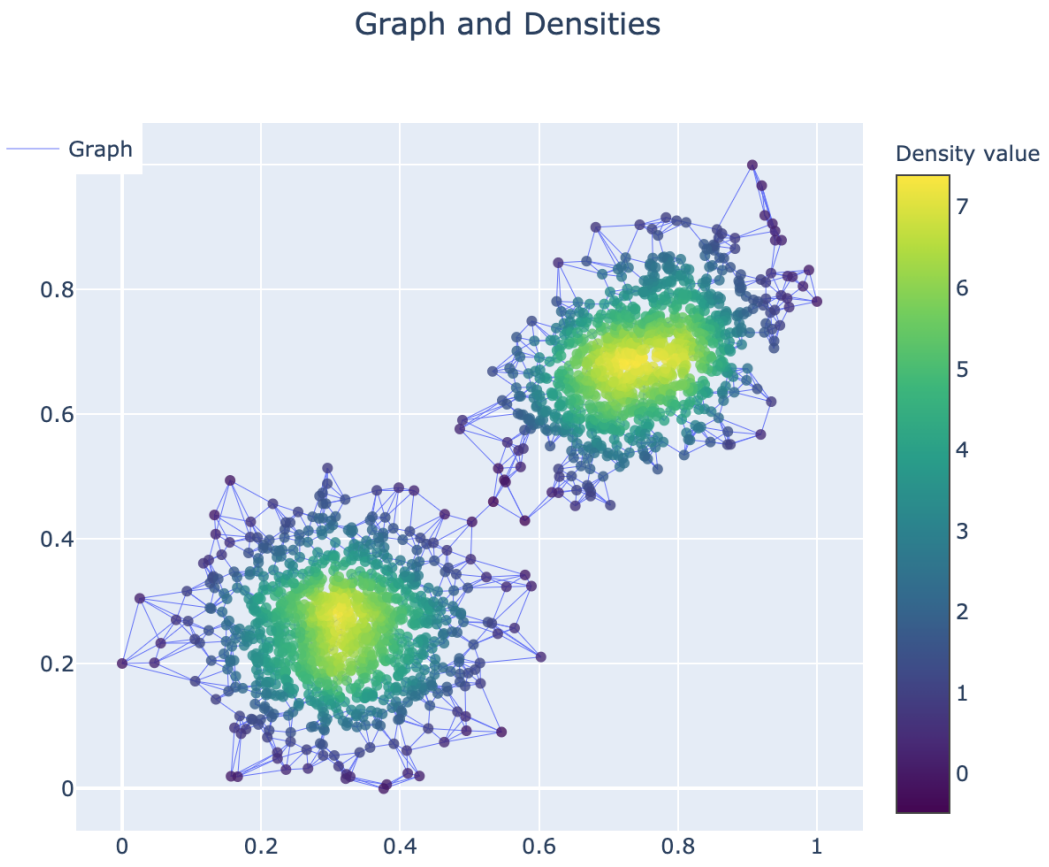



Figure 3.7: Example of a Density Graph visualization output as generated by Plotly

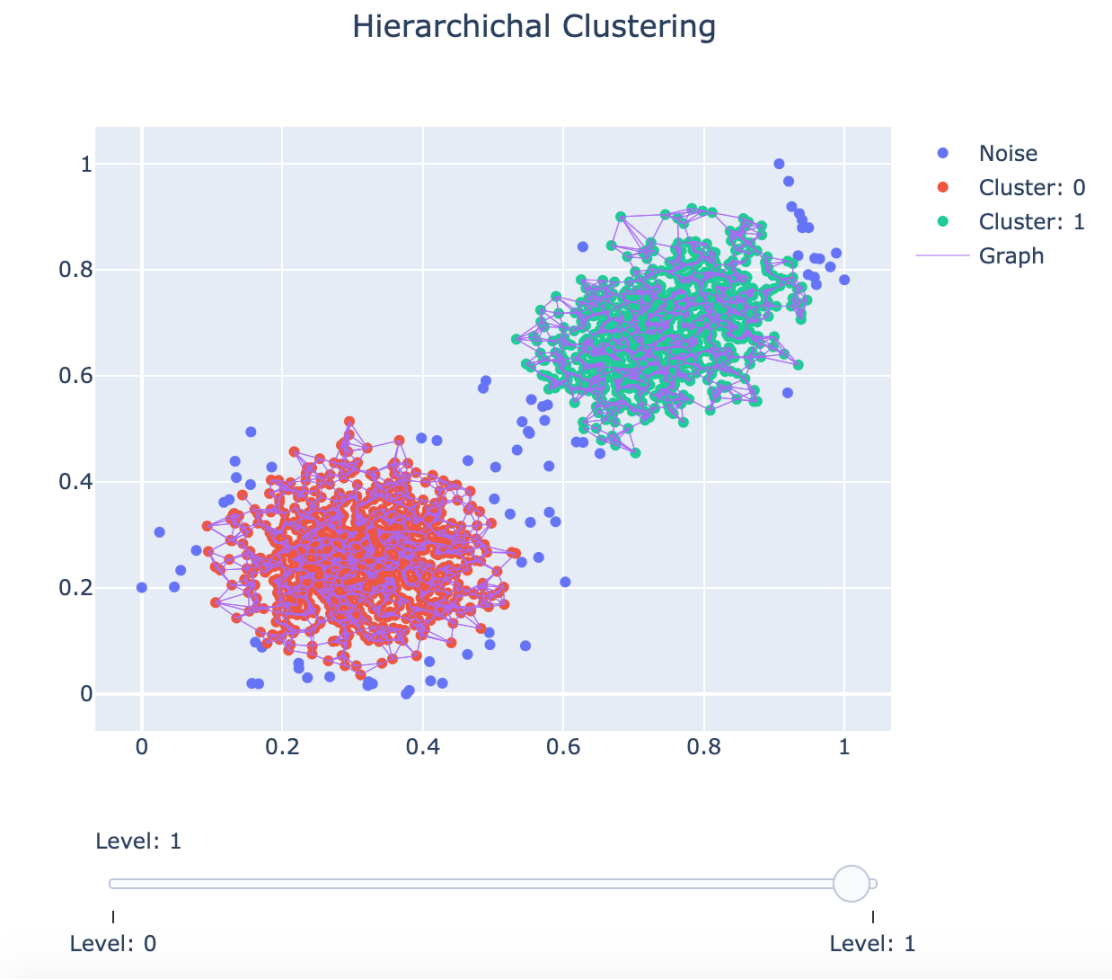



Figure 3.8: Example of a Hierarchical Clustering visualization output as generated by Plotly

## 3.8 Post Processing Module

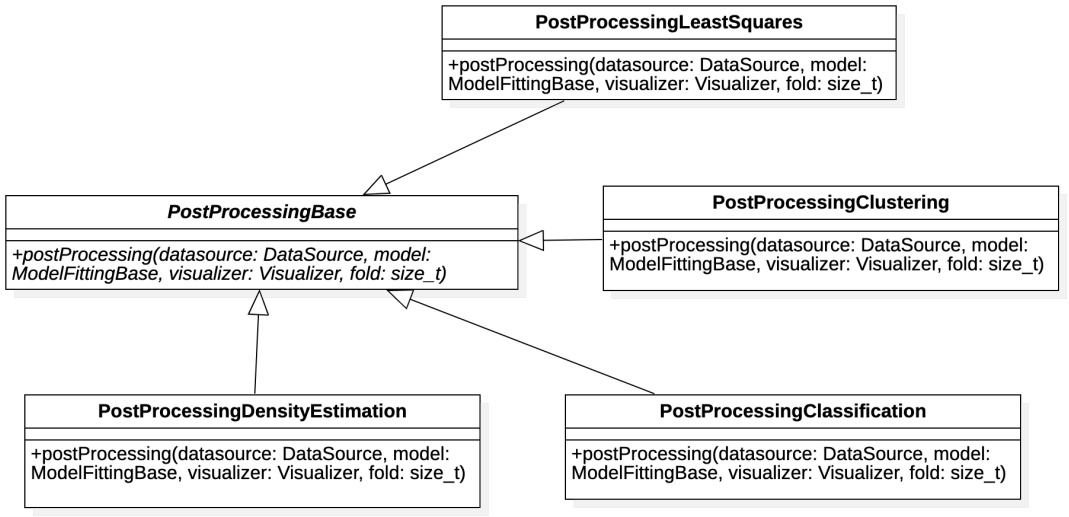



Figure 3.9: Post Processing Module Implementation Design

This is a new module implemented for all models in order to run processes which are not directly related to the online training of Sparse Grid based Machine Learning Methods. The implementation's design can be seen in Figure 3.9. So far the module contains only one method named *postProcessing*. For Density Estimation, Classification and Regression Models, this method executes the *runPostProcessing* method of their corresponding Visualizer objects.

For Clustering Models, this method implements Fischer's Algorithm described in Algorithm 2.2.3, by calling all the related methods of the ModelFittingClustering class in addition to the *runPostProcessing* of the VisualizerClustering class. It additionally stores, if previously configured, the information of the Cluster Hierarchy and the data points with their respectively assigned cluster labels.

A small modification to Fischer's Algorithm was added in order to preserve the root node of our Hierarchy Tree and therefore, a single access point to it. The root will never be replaced, even if the split criterion is fulfilled. This does not affect the end result and the root node is useful to identify those points which were ultimately labeled as noise.

# 4 Data sets and Tests

## 4.1 Data sets' Description

Four different labeled data sets were used in order to test our implementation. Table 4.1 shows a general description of them.

| Data Set Name | Number of Dimensions | Number of Samples | Number of Real Clusters |
|---|---|---|---|
| 2 Moon | 2 | 1,000 | 2 |
| Circles | 2 | 2,000 | 2 |
| 5D Gaussians | 5 | 3,000 | 3 |
| HTRU2 | 8 | 17,898 | 2 |

Table 4.1: Data sets' general description used to test our implementation

The first three data sets were synthetically generated with the help of the scikit-learn machine learning library in python [27] using the following procedures and configurations.

- The 2 Moon Data set was generated with the method *make_moons* of the package *sklearn.datasets* (Figure 4.1) while setting the number of samples to 1,000, shuffling the data and utilizing a Gaussian noise value of 0.05. The class label indicates to which of the two moons a data point belongs.
- The Circles Data set was generated with the method *make_circles* of the package *sklearn.datasets* (Figure 4.2) while setting the number of samples to 2,000, shuffling the data, utilizing a Gaussian noise value of 0.05 and a scale factor of 0.3. The class label indicates to which of the two circles a data point belongs.
- The 5D Gaussians Data set was generated with the method *make_blobs* of the package *sklearn.datasets* (Figure 4.3) while setting the number of samples per Gaussian to 1,000, the number of features to 5, a cluster standard deviation of 1

for all Gaussians and the centers of each Gaussian to $(0,0,0,0,0)$, $(5,5,5,5,5)$ and $(-5,-5,-5,-5,-5)$ respectively. The class label indicates to which of the three Gaussians a data point belongs.

```
from sklearn.datasets import make_moons
data, labels = make_moons(n_samples=1000, shuffle=True, 0.05)
```

Figure 4.1: Code used to generate the 2 Moon Data set using scikit-learn

```
from sklearn.datasets import make_circle
data, labels = make_circles(n_samples=2000,
       shuffle=True, noise=0.05 ,factor=0.3)
```

Figure 4.2: Code used to generate the Circles Data set using scikit-learn

```
from sklearn.datasets import make_blobs
data, labels = make_blobs(n_samples=[1000,1000,1000],
       centers = [[0,0,0,0,0],[5,5,5,5,5],[-5,-5,-5,-5,-5]],
          n_features=5, cluster_std= [1,1,1])
```

Figure 4.3: Code used to generate the 5D Gaussians Data set using scikit-learn

The HTRU2 Data set [23] is a publicly available data set within the *UCI Machine Learning Repository* [7]. This is an 8-dimensional data set containing measures of astronomical phenomena, which were gathered, classified and originally presented at [22]. The class label determines if the phenomenon measured represents a legitimate pulsar candidate (label 1) or a fake one (label 0). The data is heavily skewed towards the latter, containing 1,639 data points labeled as legitimate pulsar candidates and 16,259 as fake ones.

All data sets were normalized utilizing a Minimum-Maximum normalization approach in order for all data points to be contained in an n-dimensional box whose minimum and maximum dimension boundaries were set to 0.1 and 0.9, respectively.

## 4.2 Tests

### 4.2.1 Description

For the purpose of testing our implementation, two tests were conducted for each of the data sets described in the previous section. These were:

1. We generated a Hierarchical Clustering using our implementation within the SG++ Datamining Pipeline over the whole range of possible density values. We saved the hierarchy meta data file, measured the quality of the final clustering and generated the visualization plots.

2. With the help of the hierarchy meta data file and the visualization plots of the previous test, we determined the best density estimation threshold that would deliver us the clustering that best matched the predefined labels distribution. We proceeded then to run the Flat Clustering Algorithm while applying this density threshold. We measured again the quality of our clustering and compared these results against the previous ones in order to check the improvement in quality.

### 4.2.2 Running Configurations

To run the first test, the running configurations used for each data set in order to generate their corresponding Density Estimation Models are described in Table 4.2. The parameters of the pipeline, specifically the fitter parameters, which are not shown in this table, were set to the their respectively default values.

| Data Set Name | Grid Level | Batch size | Regularization Lambda | Plots generated |
|---|---|---|---|---|
| 2 Moon | 5 | 1,000 | $1x10^{-6}$ | Scatterplots, Heatmaps |
| Circles | 7 | 2,000 | $1x10^{-6}$ | Scatterplots, Heatmaps |
| 5D Gaussians | 4 | 3,000 | $1x10^{-5}$ | Scatterplots |
| HTRU2 | 4 | 2,000 | $1x10^{-5}$ | Scatterplots |

Table 4.2: Specific running configurations for each data set

Regarding the clustering parameters, we set for all data sets a value of 5 to the number of nearest neighbors and a value of 10 to the number of steps used to generate

the Hierarchical Clustering. The minimum and maximum density thresholds were set to 0 and 1 respectively. The chosen split density threshold value chosen was the default one.

For the Visualization Module, the output selected was JSON in order to easily generate plots by using the plotly library. In the case of the high dimensional data sets, the t-SNE configurations used to generate the 2-dimensional embedding were a perplexity value of 30, a theta value of 0.5, a random seed of 150 and a maximum number of iterations of 1,000.

For the second test, most of the previous configurations were maintained for all data sets. The only applied changes were the number of steps, which was set to 1, and the minimum and maximum density thresholds, which were set to the value obtained from the analysis of the first test's results.

Regarding the metrics used to evaluate the clustering, all four implemented metrics were used in both tests for all data sets.

# 5 Results

## 5.1 Hierarchical Clustering Run

### 5.1.1 2 Moon Data set

We first take a look at the Density Estimation model obtained for the 2 Moon Data set, which is shown in Figure 5.1. From the heatmap, potential subclusters can already be identified in extremely high density areas inside each of the 2 Moons. Figure 5.2 shows the nearest neighbors' graph along with the evaluated densities for each point. Again the potential subclusters of the 2 Moons can be easily detected along with the linkage points between them.

| Level | # Clusters | # Points in Level | Density Threshold Range | Average Points Per Cluster |
|---|---|---|---|---|
| 0 | Unclustered | 1,000 | N/A | N/A |
| 1 | 2 | 1,000 | 0.0 | 500 |
| 2 | 7 | 980 | 0.3-0.4 | 127.14 |
| 3 | 12 | 717 | 0.4-0.8 | 59.75 |
| 4 | 14 | 342 | 0.5-0.7 | 24.42 |
| 5 | 11 | 98 | 0.7-0.9 | 8.9 |

Table 5.1: General Description of the Hierarchical Clustering obtained from the 2 Moon Data set.

Table 5.1 contains a summarized information of the Hierarchical Clustering obtained from the 2 Moon Data set. Additionally, graphical representations of the Hierarchy are shown in Figures 5.3, 5.4 and 5.5. We obtained a total of 46 clusters distributed in 5 hierarchical levels. In level 1 (Figure 5.3b), we already obtain a clustering that matches the predefined labeling of the data set, however, based on our density estimation model, we know that additional subclusters exists within these ones. These are shown in levels 2, 3 and 4 (Figures 5.4a, 5.4b and 5.5a respectively) and they match approximately the high density areas of the Density Estimation Heatmap. An interesting fact to remark is the overlap between the density threshold ranges of levels 3 and 4. This is

an indicator that clusters originally inserted in level 3 were deleted and substituted by their corresponding children in level 4. After level 4, the model starts to suffer from overfitting due to the low number of remaining points after the pruning of the graph. A closer look to Figure 5.5b confirms this and also shows that most of these new clusters contain mostly a small number of data points, and therefore, almost no relevant information.

| No. noise Points | Fowlkes-Mallows | V-Measure | Calinski-Harabasz | David-Bouldin |
|---|---|---|---|---|
| 0 | 0.333 | 0.358 | 125.122 | 1.29 |

Table 5.2: Quality Scores obtained for each implemented metric of the Hierarchical Clustering of the 2 Moon Data set

| Homogeneity | Completeness | V-Measure |
|---|---|---|
| 1 | 0.218 | 0.358 |

Table 5.3: Completeness and Homogeneity of the Hierarchical Clustering of the 2 Moon Data set.

Table 5.2 shows the final quality scores of the Hierarchical Clustering of the 2 Moon Data set for each implemented metric. The low values of the Fowlkes-Mallows Index and the V-Measure are due to the large number of clusters obtained in the hierarchy. A closer look to the Homogeneity and Completeness Scores of the V-Measure (Table 5.3) indicates that to increase our score we need to reduce the number of clusters. Basing ourselves on this, on the information of the hierarchy and on the visualization plots, we identified that to improve these scores we have to run the Flat Clustering Algorithm using only a density threshold of 0.0.

Finally, the Calinski-Harabasz and David-Bouldin Indexes obtained in this run cannot tell us much now about the quality of the clustering, since we need another score to which we can compare them against. They will be however, used as a reference in the run of the Flat Clustering Algorithm to evaluate the quality improvement.

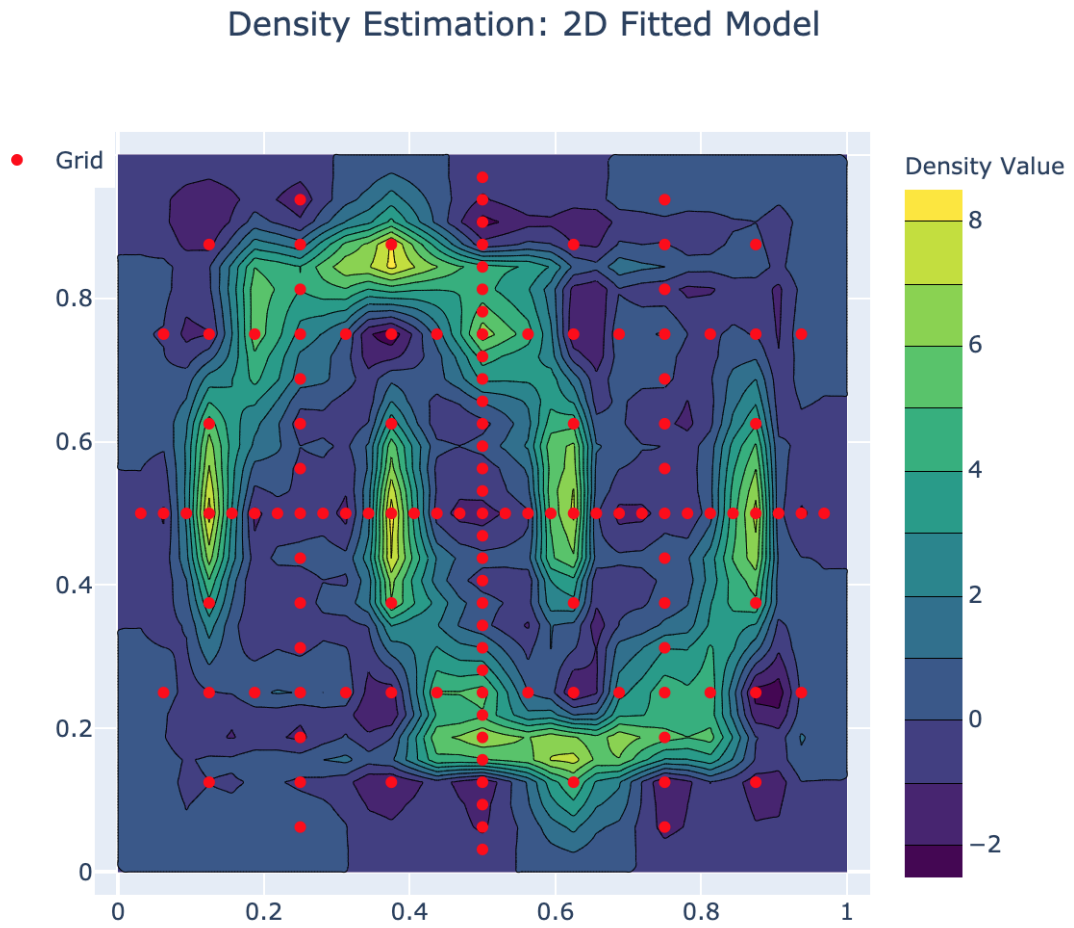



Figure 5.1: Density Estimation Heatmap of the 2 Moon Data set with a graphical depiction of the sparse grid used for its calculation. The heatmap shows within both moons multiple high density zones, which can form independent clusters when higher density thresholds are applied.

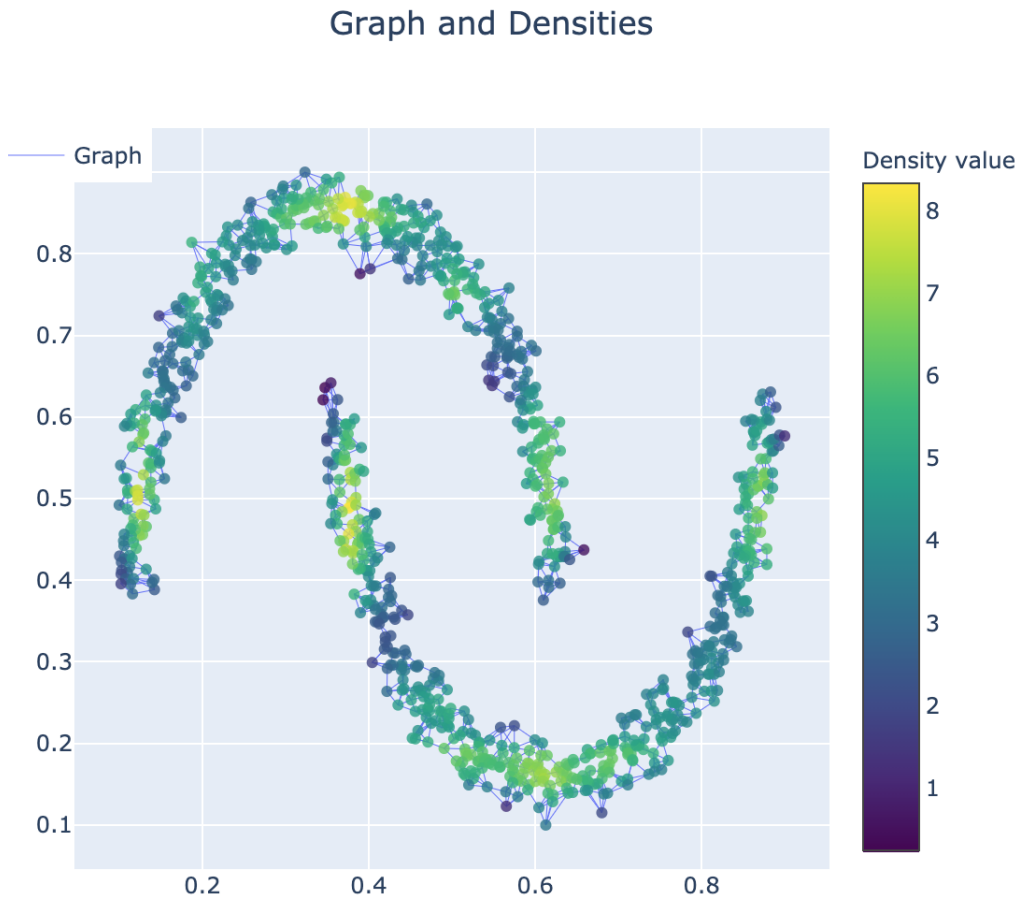
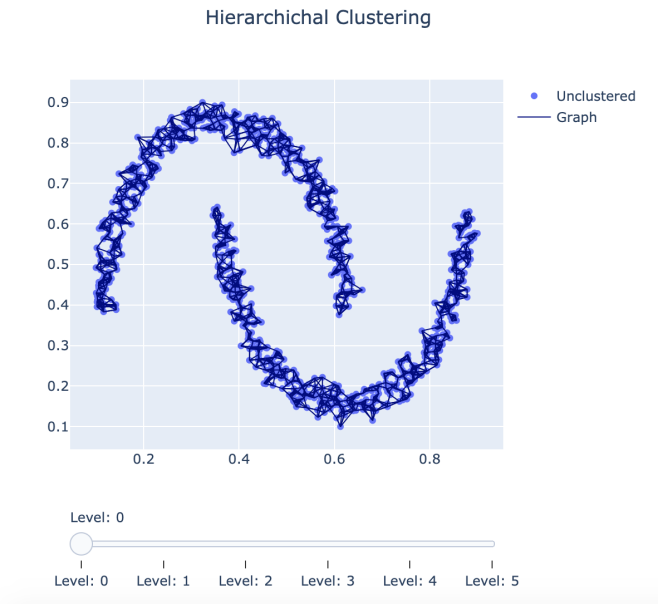
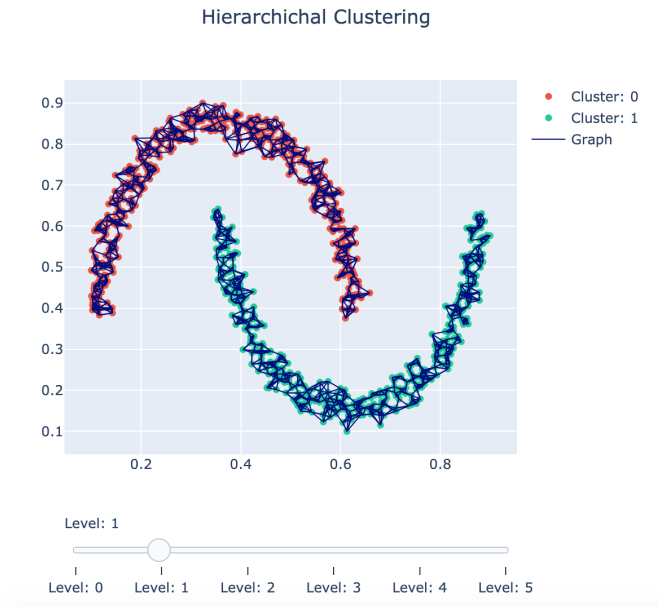



Figure 5.2: Nearest Neighbors' Graph of the 2 Moon Data set using 5 nearest neighbors along with the evaluated densities of all of the points. Just like in Figure 5.1, potential sub clusters of the 2 moons can already been seen based on the density values and on the connections among the points.

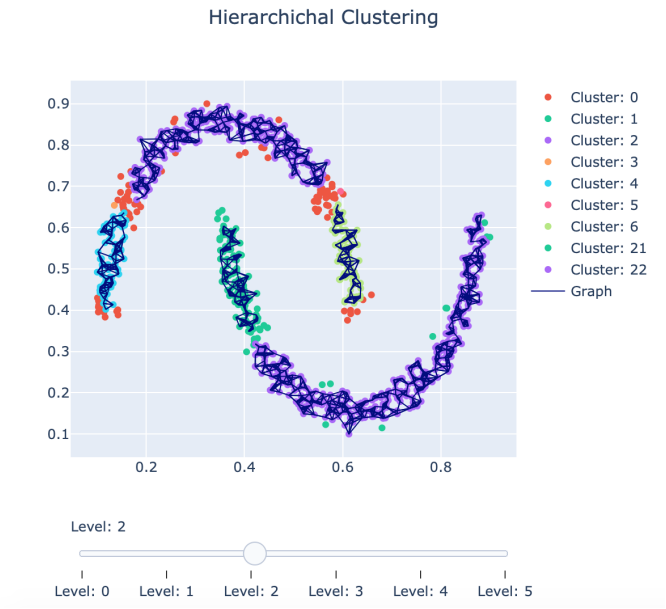



(a) Level 0

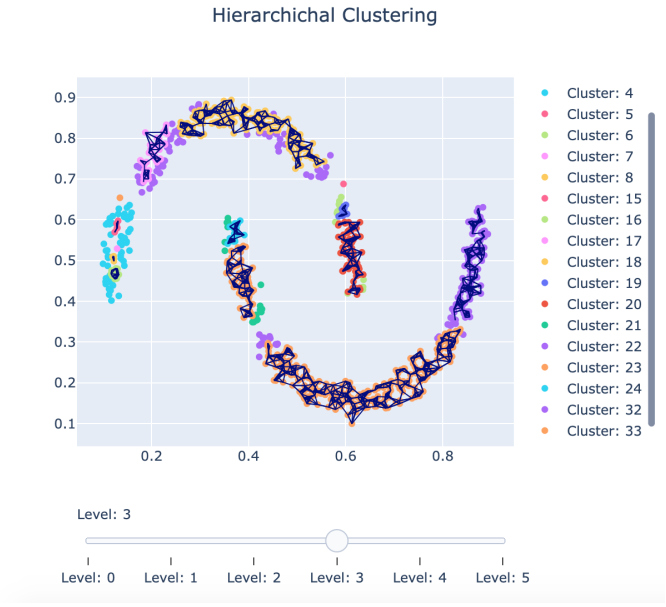



(b) Level 1

Figure 5.3: First two levels of the Hierarchical Clustering obtained for the 2 Moon Data set. In level 1, we already obtain the clustering specified by the predefined labels.

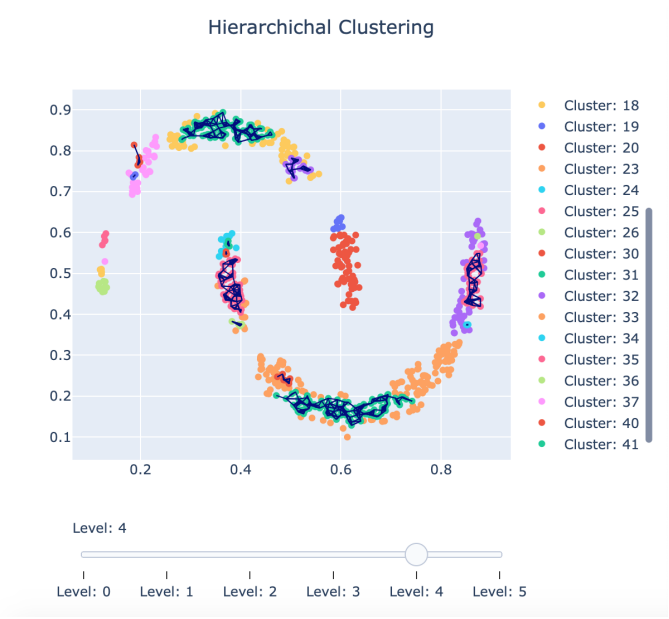



(a) Level 2

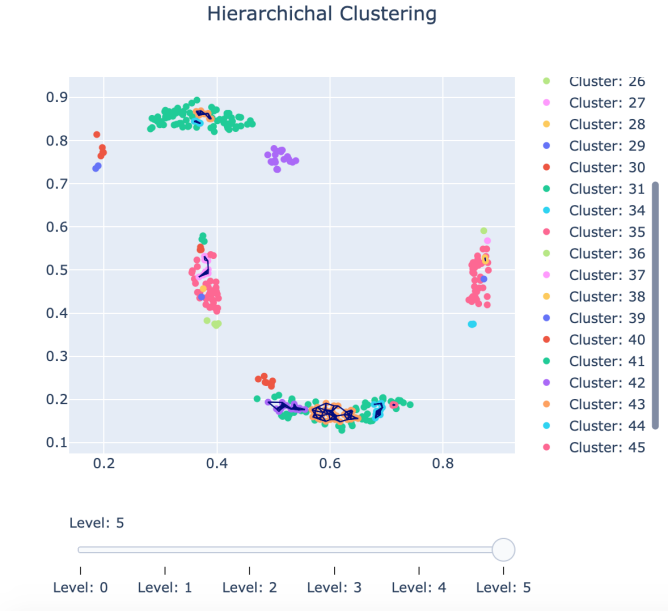



(b) Level 3

Figure 5.4: Levels 3 and 4 of the Hierarchical Clustering obtained for the 2 Moon Data set. The potential clusters mentioned in Figures 5.1 and 5.2 start to appear in these levels.




(a) Level 4




(b) Level 5

Figure 5.5: Levels 4 and 5 of the Hierarchical Clustering obtained for the 2 Moon Data set. More subclusters are obtained in these levels. These however, tend to be quite small in comparison to the size of the data set, indicating that the model is being overfitted.

### 5.1.2 Circles Data set

The heatmap of the Density Estimation model obtained for the Circles Data set is shown in Figure 5.6. Similarly to the 2 Moon Data set, potential subclusters can already be identified both circles in the areas of relative high density. Interesting to see is the presence of negative density zones in the area between the circles and in the center of the inner circle.

Figure 5.7 shows the nearest neighbors' graph of our data set along with the evaluated densities. In this case we have obtained points whose estimated density is a negative value, and therefore will be labeled automatically as noise. We can also see some high density areas which will likely form subclusters after a certain density threshold is applied.

| Level | # Clusters | # Points in Level | Density Threshold Range | Average Points Per Cluster |
|---|---|---|---|---|
| 0 | Unclustered | 2,000 | N/A | N/A |
| 1 | 2 | 1,994 | 0.0 | 997 |
| 2 | 20 | 1,126 | 0.1-0.7 | 56.3 |
| 3 | 23 | 359 | 0.2-0.8 | 15.60 |
| 4 | 2 | 13 | 0.9 | 6.5 |

Table 5.4: General Description of the Hierarchical Clustering obtained from the Circles Data set.

Table 5.4 contains the summarized information of the Hierarchical Clustering obtained for the Circles Data set and its graphical depiction can be observed in Figures 5.8, 5.9 and 5.10. We have obtained 47 clusters distributed among 4 levels. Similarly to the 2 Moon Data set, the best clustering is obtained already in level 1 (Figure 5.8b), though without perfect matching due to the presence of noisy points in the clustering. These are actually the ones with negative density shown previously in the graph plot. The potential subclusters seen in the Density Estimation Heatmap start to appear in levels 2 and 3 (Figures 5.9a and 5.9b, respectively). Again the density threshold ranges of both levels overlap, indicating substitution of previously existing clusters in level 2 by their children. In level 4 (Figure 5.10) the model overfits the data, showing only 13 of the original 2000 points clustered in 2 clusters, one of them consisting of just a single data point.

| No. noise Points | Fowlkes-Mallows | V-Measure | Calinski-Harabasz | David-Bouldin |
|---|---|---|---|---|
| 6 | 0.575 | 0.423 | 81.905 | 1.213 |

Table 5.5: Quality Scores obtained for each implemented metric of the Hierarchical Clustering of the Circles Data set.

| Homogeneity | Completeness | V-Measure |
|---|---|---|
| 0.998 | 0.268 | 0.423 |

Table 5.6: Completeness and Homogeneity of the Hierarchical Clustering of the Circles Data set.

Table 5.5 shows the final quality scores of the Hierarchical Clustering of the Circles Data set for each implemented metric. We obtain again low values for the Fowlkes-Mallows Index and the V-Measure, caused mainly by the large number of clusters obtained in the hierarchy. A closer look to the Homogeneity and Completeness Scores of the V-Measure (Table 5.6) indicates again that to increase our score we need to reduce the number of clusters. Basing ourselves on this, on the information of the hierarchy and on the visualization plots, we decided to run the flat Clustering Algorithm using only a density threshold of 0.0.

Regarding the Calinski-Harabasz and David-Bouldin Indexes obtained in this run, they seem to be at first glance indicators of a good clustering when compared to the ones obtained in the 2 Moon Data set. We still need however, to check how they are affected after the run of the Flat Clustering Algorithm to really determine the overall quality of the clustering.

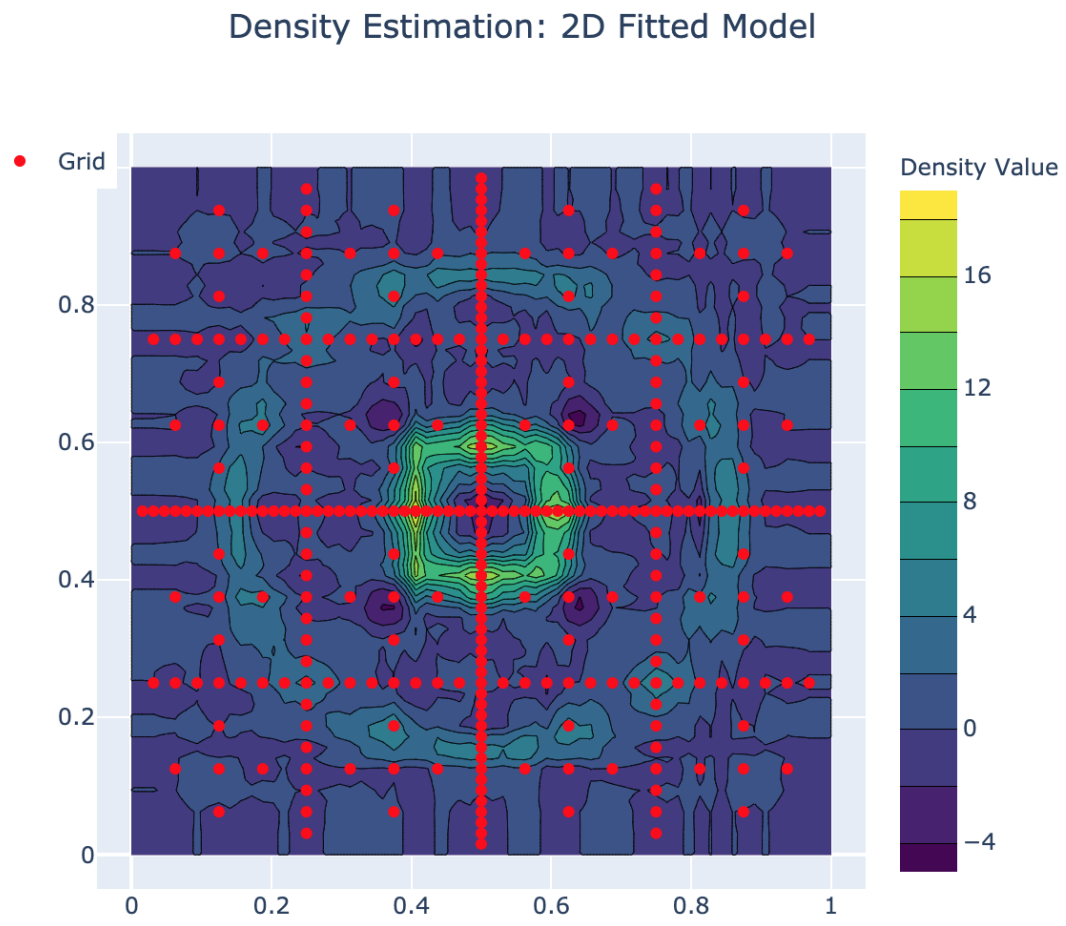



Figure 5.6: Density Estimation Heatmap of the 2 Circles Data set. The heatmap shows within both circles multiple separate high density zones, which could form subclusters of the circles when higher density thresholds are applied.

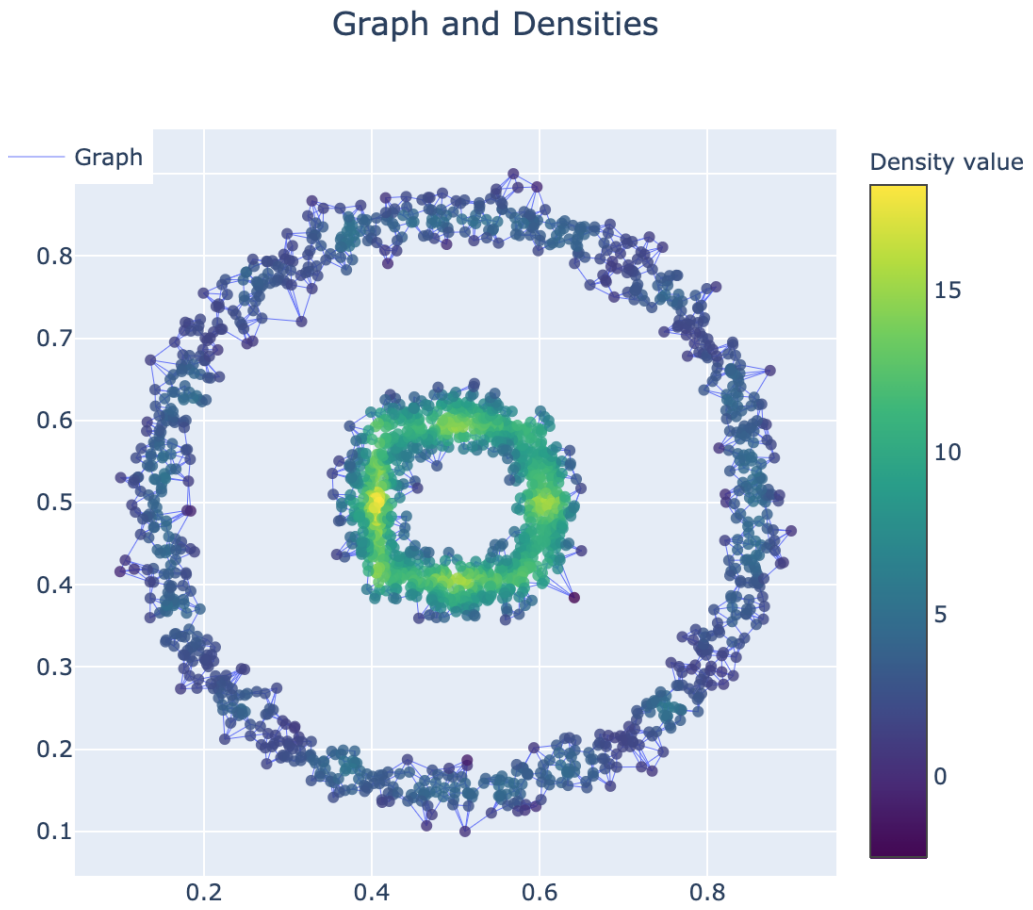
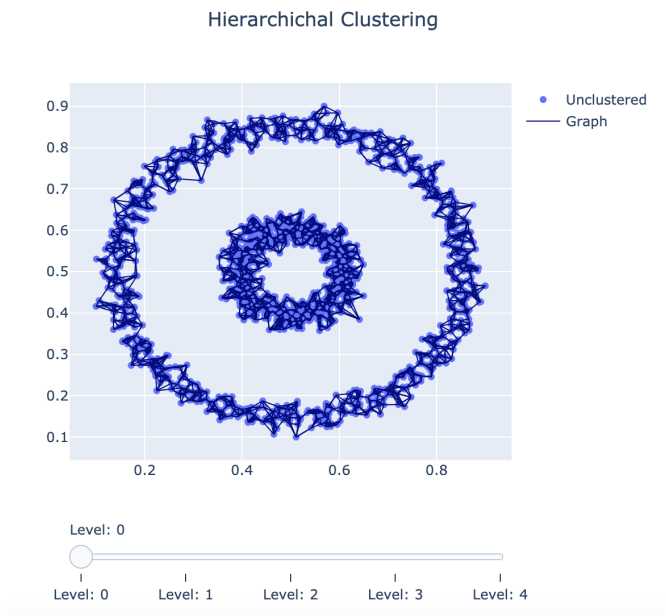
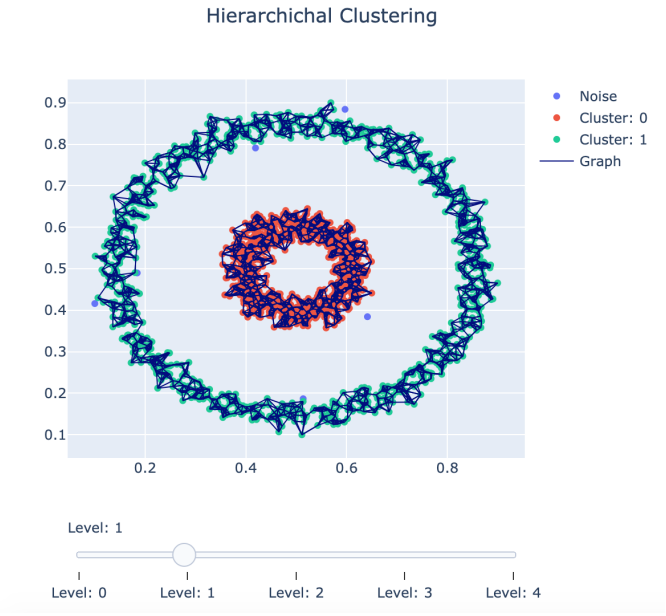



Figure 5.7: Nearest Neighbors' Graph of the Circles Data set using 5 nearest neighbors, along with the evaluated densities of all of the points. Just like in Figure 5.6, potential sub clusters can already been seen based on the density values and the connections among the points. Curious to see is the drastic change in density in some of the outer points of both circles. A closer look to the density values shows us that this points' densities have actually negative values.

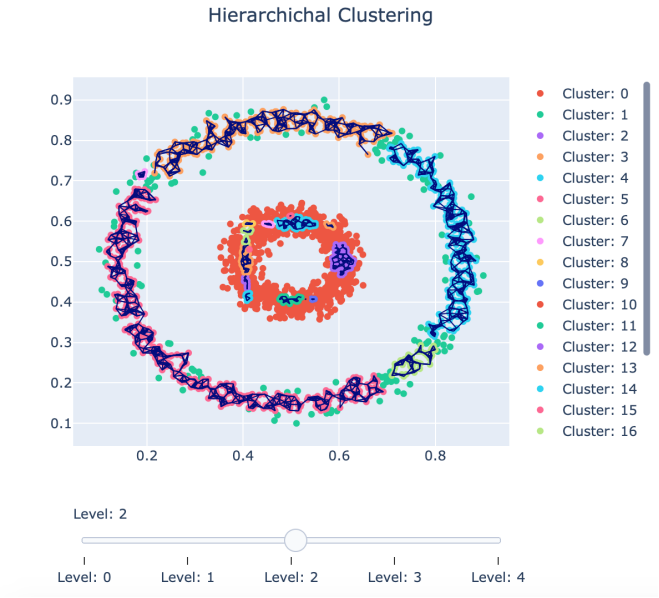



(a) Level 0

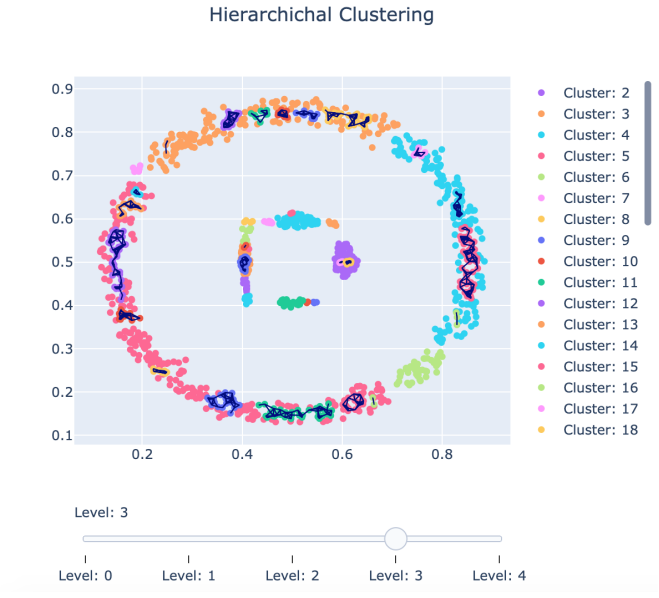



(b) Level 1

Figure 5.8: First two levels of the Hierarchical Clustering obtained for the Circles Data set. In level 1 we already obtain an almost perfect clustering based on the predefined labels in the data set. Note that even though our minimum density threshold was 0, the algorithm flags some of the points as noise due to them having negative density values.




(a) Level 2




(b) Level 3

Figure 5.9: Levels 2 and 3 of the Hierarchical Clustering obtained for the Circles Data set. The potential clusters mentioned in Figure 5.6 start to appear in level 2. In level 3, subclusters start to reduce in size, indicating that the model is starting to overfit the data.


Figure 5.10: Levels 4 of the Hierarchical Clustering obtained for the Circles Data set. The model is definitely overfitted at this level due to the low number of points remaining.

### 5.1.3 5D Gaussians Data set

In Figure 5.11 we show the 2-dimensional embedding of the graph of the 5D Gaussians Data set and its evaluated densities. It is clear from the plot, that each Gaussian forms a unique separate cluster. However, since the density values appear to be almost uniform across each Gaussian it is highly likely that subclusters are non-existent. One remarkable aspect is the apparent difference in density values in two of the Gaussians in comparison to the other one. A closer look to our data set and to the grid level used to generate the density estimation indicates that the low density Gaussians are weakly described by the model due to them being located in areas containing low numbers of grid points.

| Level | # Clusters | # Points in Level | Density Threshold Range | Average Points Per Cluster |
|---|---|---|---|---|
| 0 | Unclustered | 3000 | N/A | N/A |
| 1 | 3 | 2999 | 0.0 | 999.6 |
| 2 | 12 | 27 | 0.6-0.8 | 2.25 |

Table 5.7: General Description of the Hierarchical Clustering obtained from the 5D Gaussians Data set.

Despite this shortcoming, the Gaussians are identified correctly in the Hierarchical Clustering. The Hierarchical Clustering of the 5D Gaussians Data set is presented in Table 5.7 and its graphical depictions are shown in Figures 5.12 and 5.13. In level 1, (Figure 5.12b) we already obtain a good cluster separation with only one point flagged as noise due to its density being a negative value. The almost non-existent presence of subclusters is confirmed in level 2 (Figure 5.13), since there is only one Gaussian which contains a few subclusters. These however, consist of an insignificant number of data points due to them being generated at high density thresholds. In the end they do not provide any useful information about the data.

| No. noise Points | Fowlkes-Mallows | V-Measure | Calinski-Harabasz | David-Bouldin |
|---|---|---|---|---|
| 1 | 0.99 | 0.973 | 3359.31 | 0.244 |

Table 5.8: Quality Scores obtained for each implemented metric of the Hierarchical Clustering of the 5D Gaussians Data set.

| Homogeneity | Completeness | V-Measure |
|---|---|---|
| 1 | 0.943 | 0.973 |

Table 5.9: Completeness and Homogeneity of the Hierarchical Clustering of the 5D Gaussians Data set.

Tables 5.8 and 5.9 show the scores of the Hierarchical Clustering for the 5D Gaussians Data set. In contrast to the 2-dimensional data sets, the Fowlkes-Mallows Index and the V-Measure are actually high due to the little presence of subclusters inside the data. This seems to have had also an effect in the Calinski-Harabasz and the David-Bouldin Indexes, since they are significantly better in comparison to the 2-dimensional data sets. Just like the previous data sets we will run the Flat Clustering Algorithm using a density threshold of 0.0 and check on the improvement of our scores.

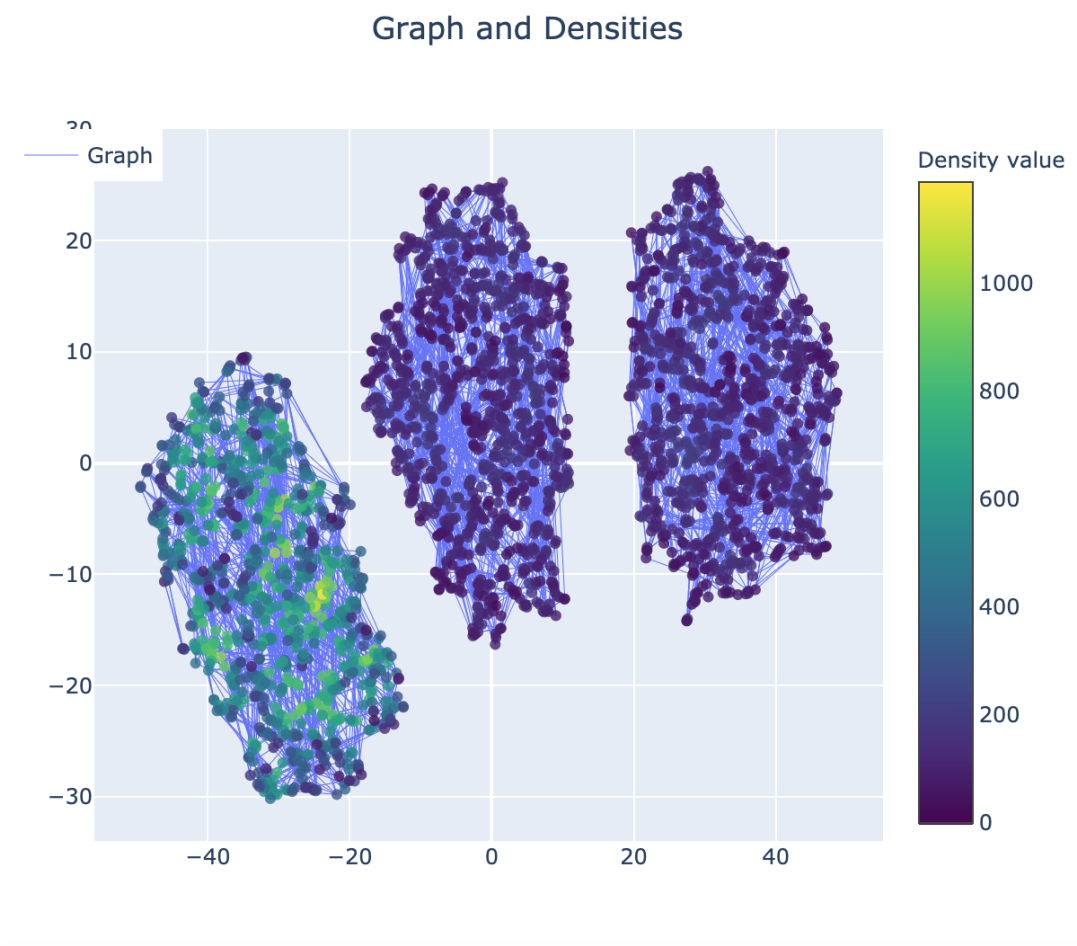
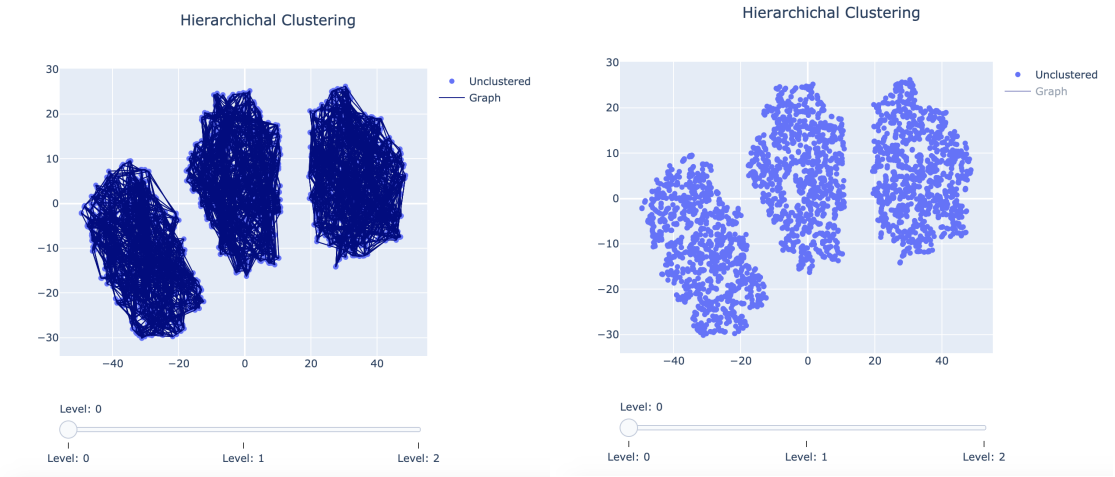
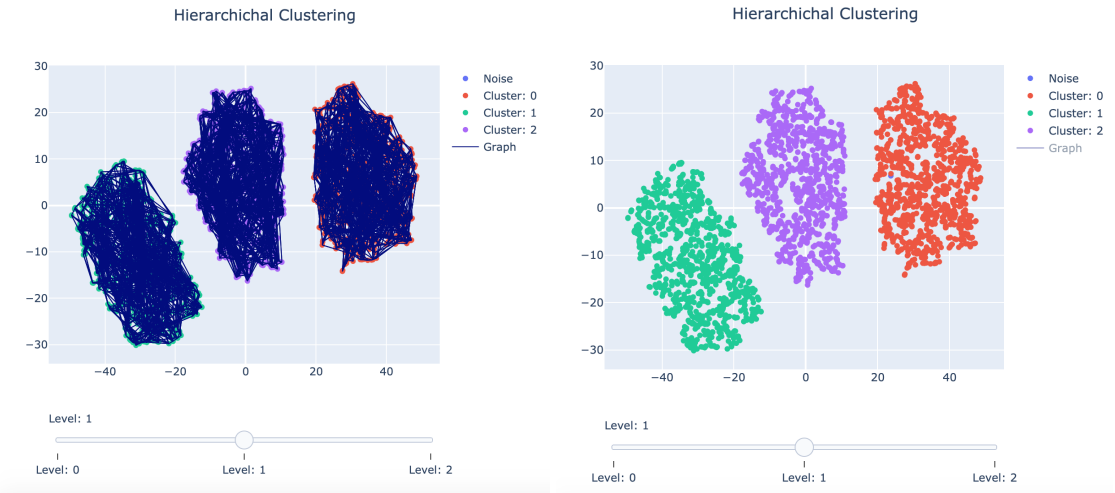



Figure 5.11: 2-dimensional embedding of the Nearest Neighbors' Graph of the 5D Gaussian Data set using 5 nearest neighbors, along with the evaluated densities of all of the points. Curious to see is the strong difference in density of the left Gaussian. due to the fact that the low density Gaussians are located in areas with low numbers of grid points. In contrast to the 2-dimensional data sets, it's not clear at first glance where potential subclusters may lie. The embedding however, along with the graph connections, gives us a small insight of how the points could be distributed in the high dimensional space.




(a) Level 0




(b) Level 1

Figure 5.12: First two levels of the Hierarchical Clustering of the 5D Gaussians Data set. In level 1, the clustering matches almost perfectly to the Gaussians definitions, with only one point being labeled as noise due to its negative density value.

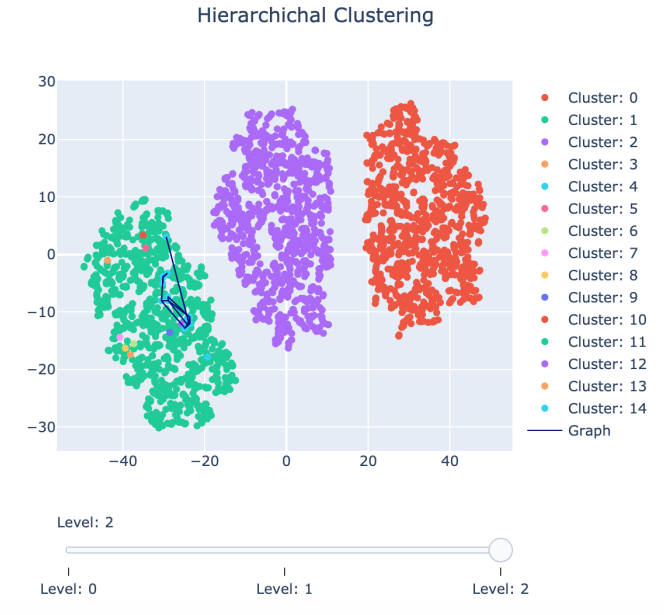



Figure 5.13: Level 2 of the Hierarchical Clustering obtained for the 5D Gaussians Data set. The model is definitely overfitted in this level, since almost all of the new clusters consist only of one data point.

### 5.1.4 HTRU2 Data set

Just like the 5D Gaussian Data set, we present the 2-dimensional embedding of the graph of the HTRU2 Data set along with the evaluated densities in Figure 5.15. We also present the same embedding of the data but with the predefined class labeling in Figure 5.14. One can notice that the low density areas of the graph plot match to some extent some of the areas described by the minority class (Label 1). Additionally, the nearest neighbors' graph connections suggest that these areas are actually related to each other and that they are potentially part of a single cluster.

| Level | # Clusters | # Points in Level | Density Threshold Range | Average Points Per Cluster |
|---|---|---|---|---|
| 0 | Unclustered | 17,898 | N/A | N/A |
| 1 | 4 | 15,982 | 0.1 | 3,995.5 |
| 2 | 3 | 15,480 | 0.2 | 5,160 |
| 3 | 14 | 12,997 | 0.3 | 928.35 |
| 4 | 42 | 6,902 | 0.4 | 164,33 |
| 5 | 73 | 100 | 0.5-0.9 | 1.36 |

Table 5.10: General Description of the Hierarchical Clustering obtained from the HTRU2 Data set.

The summary of the Hierarchical Clustering of the HTRU2 Data set and its graphical depiction can be found in Table 5.10 and the Figures 5.16, 5.17 and 5.18, respectively. We obtained a total of 136 clusters distributed among 5 levels. Starting with level 1 (Figure 5.16b), we can see that we obtained a cluster similar on size to the majority class of the data set. The lower density areas mentioned before are actually flagged as noise, despite the minimum density threshold being 0 and the data not containing negative densities. The reason behind this is that, in contrast to the previous data sets, the points are so heavily connected that separate connected components start to appear only when the first real density threshold is applied (0.1 in this case). This causes many of the low density areas to be deleted and subsequently to be flagged as noise. Additionally, the noise set obtained is almost similar in size to the minority class of the data set. Subsequent levels don't show any additional relevant information, since the big cluster obtained in level 1 is split recursively into another big cluster and multiple smaller ones until only insignificant clusters remain in level 5 (Figure 5.18b).

| No. noise Points | Fowlkes-Mallows | V-Measure | Calinski-Harabasz | David-Bouldin |
|---|---|---|---|---|
| 1,916 | 0.5417 | 0.104 | 119.04 | 1.064 |

Table 5.11: Quality Scores obtained for each implemented metric of the Hierarchical Clustering of the HTRU2 Data set.

| Homogeneity | Completeness | V-Measure |
|---|---|---|
| 0.298 | 0.063 | 0.104 |

Table 5.12: Completeness and Homogeneity of the Hierarchical Clustering of the HTRU2 Data set.

Tables 5.11 and 5.12 contain the score information of the Hierarchical Clustering for the HTRU2 Data set. Due to the extremely high number of clusters, the values of the Fowlkes-Mallows Index and the V-Measure are extremely low. This seems to be affecting the Calinski-Harabasz Index as well, since a large number of clusters is penalized by the metric. On the other hand, the David-Bouldin Index appears to indicate a good clustering at first glance, but that could be also explained through the existence of multiple single-point clusters which have per definition a sparsity measure of 0 and therefore, have an insignificant contribution to the final score. Basing ourselves in the Hierarchical Clustering results, we will proceed to run the Flat Clustering Algorithm using only a density threshold of 0.1 and check if the scores improve significantly by reducing the number of clusters obtained.

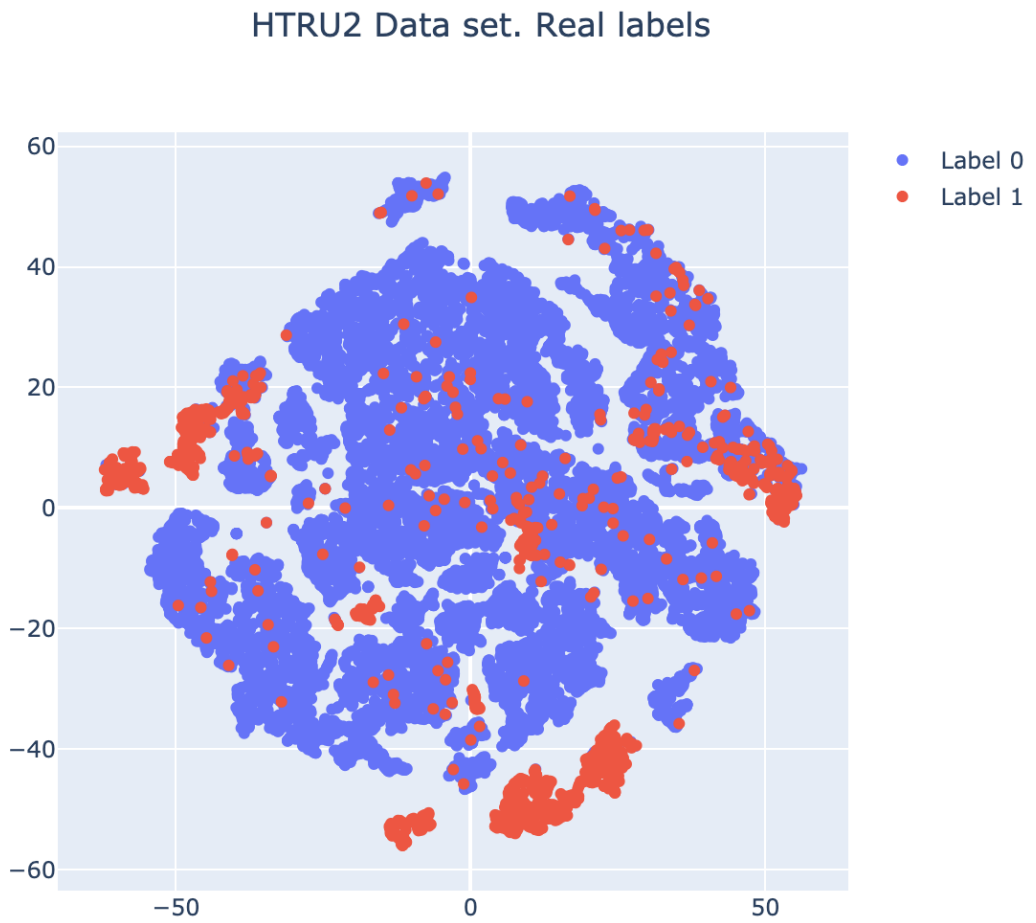



Figure 5.14: 2-dimensional embedding of the HTRU2 Data set with its predefined labeling.

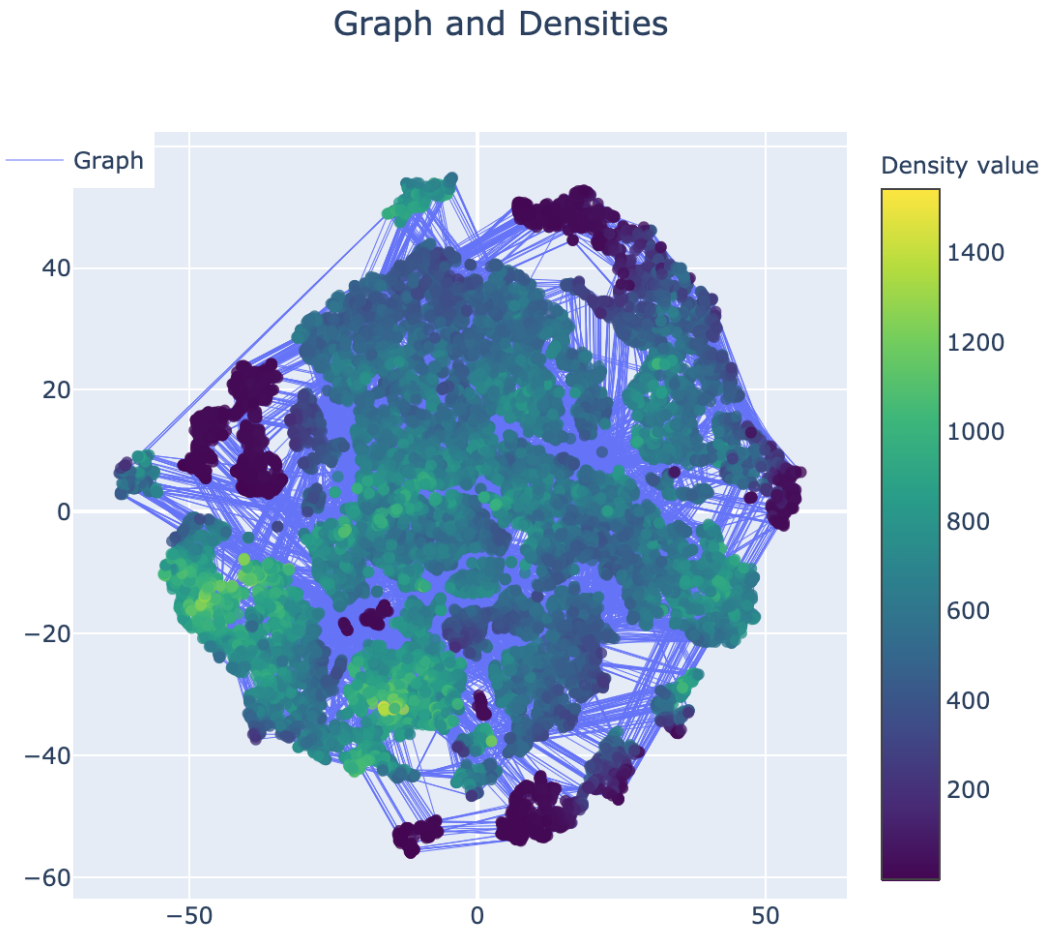
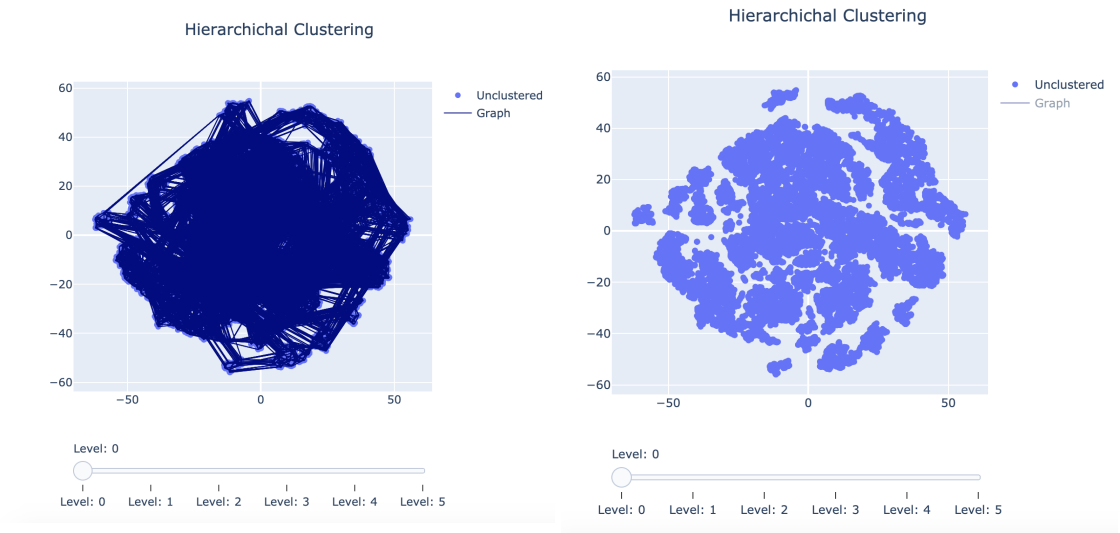
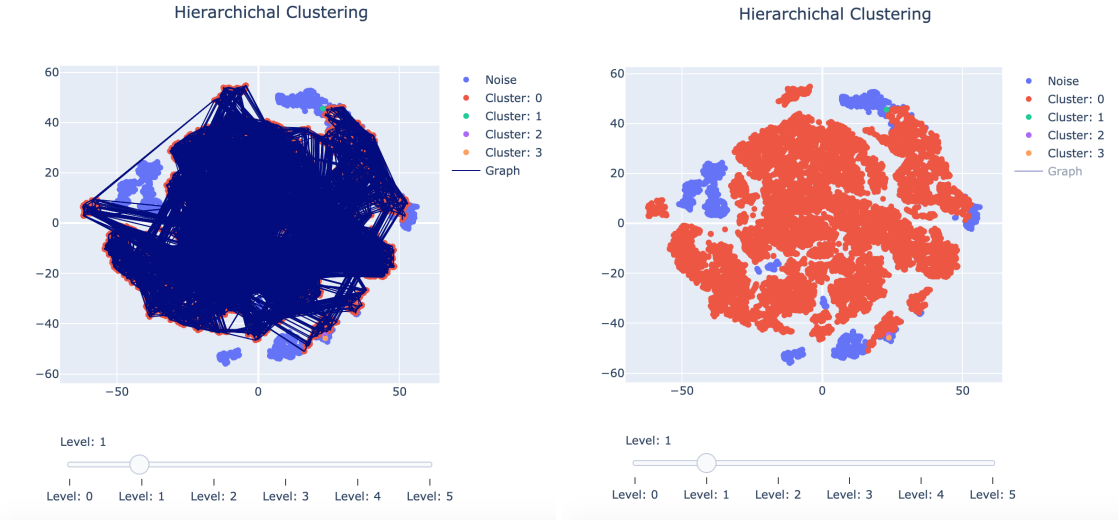



Figure 5.15: 2-dimensional embedding of the Nearest Neighbors' Graph of the HTRU2 Data set using 5 nearest neighbors, along with the evaluated densities of all of the points. Interestingly, the density distribution matches the predefined labels distribution to some extent. There is a big concentration of points whose densities values vary from medium to high, while a very small amount does not even surpass the 10 percent of the maximum density. Comparing this plot with Figure 5.14, we can see that the low density areas are mostly made of points with predefined label 1, the minority cluster. Additionally, the graph connections show that these areas are close in the high dimensional space and therefore, could be part of a single cluster entity.

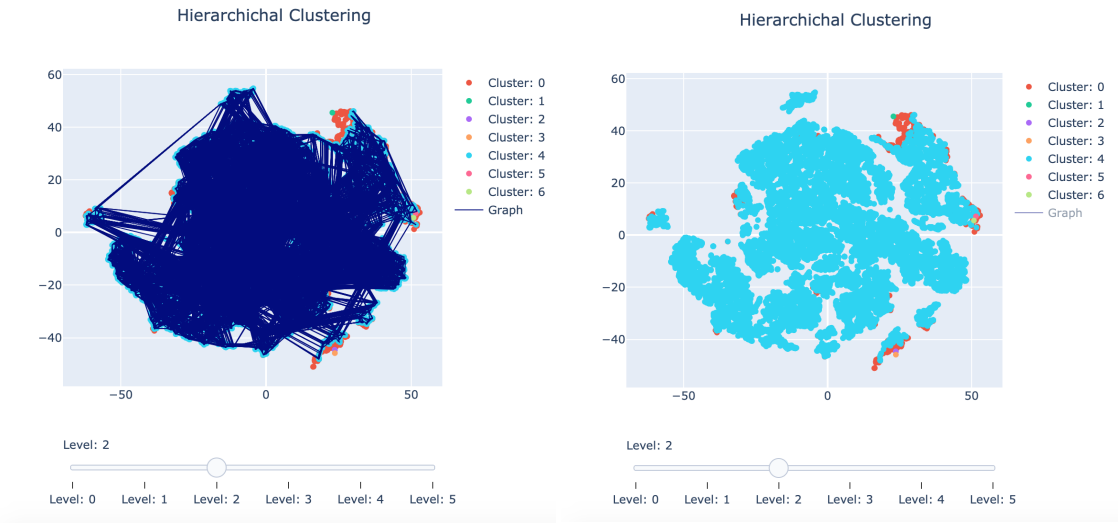



(a) Level 0

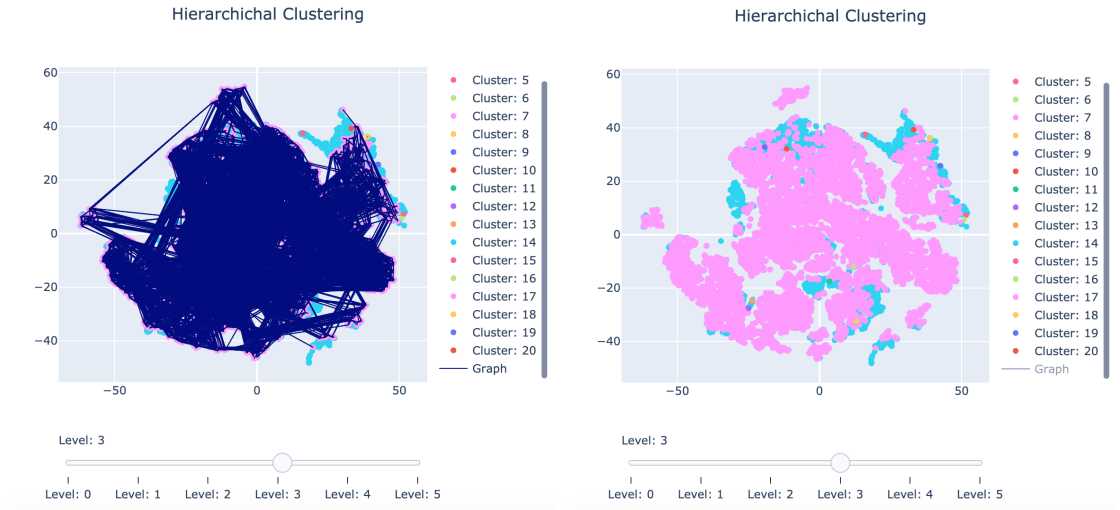



(b) Level 1

Figure 5.16: First two levels of the Hierarchical Clustering of the HTRU2 Data set. The low density areas of Figure 5.15 are assigned as noise in the first level of the hierarchy, despite them consisting of only positive density values and the defined minimum density threshold being 0. This is in fact, due to the heavy connectivity in the graph in comparison to previous data sets. Multiple connected components only start to appear after a significant density threshold is applied, deleting the points and flagging them as noise.

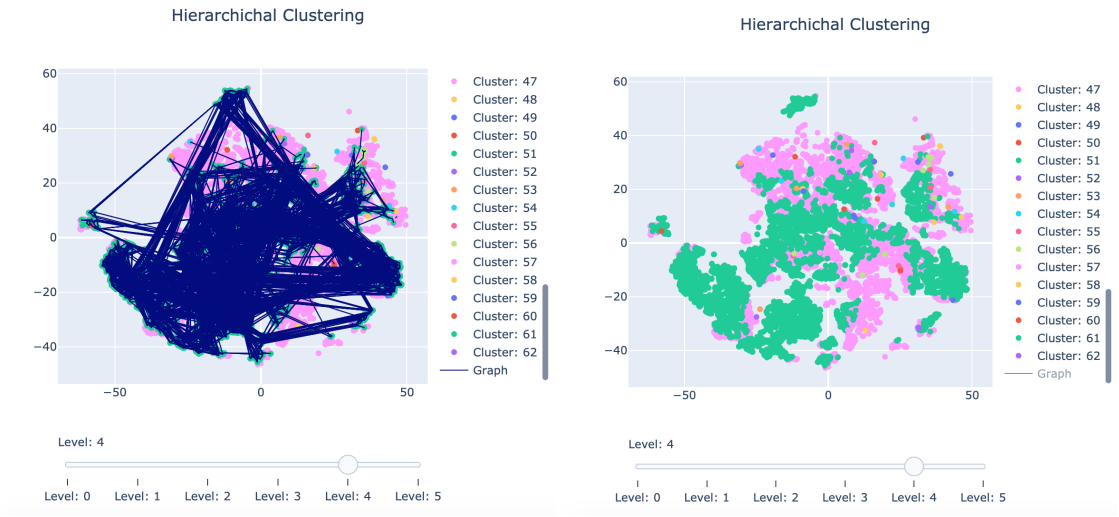



(a) Level 2

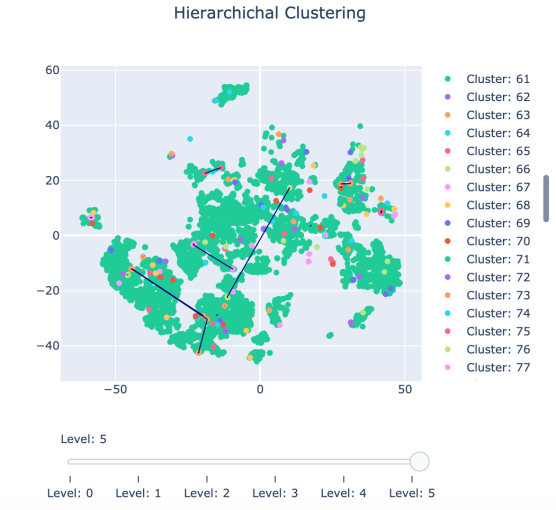



(b) Level 3

Figure 5.17: Levels 2 and 3 of the Hierarchical Clustering of the HTRU2 Data set. The big cluster area just keeps reducing in size and multiple single-element clusters start to appear.




(a) Level 4




(b) Level 5

Figure 5.18: Levels 4 and 5 of the Hierarchical Clustering of the HTRU2 Data set. In level 4, the same behavior of previous levels continues to occur. In level 5 however, the model totally overfits by generating mostly clusters consisting of 1 and 2 elements.

## 5.2 Flat Clustering Run

### 5.2.1 2 Moon Data set

After running the Flat Clustering Algorithm while applying a density threshold value of 0.0, we obtained a Clustering equivalent to the level 1 of the corresponding Hierarchical Clustering. The new quality scores are shown in Tables 5.13 and 5.14. As expected, the Fowlkes-Mallows Index and the V-Measure are now perfect since our clustering matches the predefined labeling. More interesting to see is the significant improvement of the Calinski-Harabasz Index in comparison to the previous run. The David-Bouldin Index shows improvement as well, albeit a small one in comparison to the other metrics.

| No. noise Points | Fowlkes-Mallows | V-Measure | Calinski-Harabasz | David-Bouldin |
|---|---|---|---|---|
| 0 | 1 | 1 | 905.436 | 1.002 |

Table 5.13: Quality Scores obtained for each implemented metric of the Flat Clustering of the 2 Moon Data set

| Homogeneity | Completeness | V-Measure |
|---|---|---|
| 1 | 1 | 1 |

Table 5.14: Completeness and Homogeneity of the Flat Clustering of the 2 Moon Data set

### 5.2.2 Circles Data set

Tables 5.15 and 5.16 show the scores obtained for the run of the Flat Clustering Algorithm on the Circles Data set using a density threshold of 0.0. We have obtained the same clustering depicted in level 1 of the respective Hierarchical Clustering. Similarly to the 2 Moon Data set, the Fowlkes-Mallows Index and the V-Measure increased to the point of giving almost perfect scores, held back only by the presence of a few points flagged as noise. On the other hand, the Calinski-Harabasz and David-Bouldin Indexes worsened drastically. An explanation for this behavior is that the centroids of both clusters, represented by the circles, are actually quite close to each other. This causes that both metrics misinterpret them as being too similar and therefore, both metrics result in a very low score. This example shows the necessity of using multiple metrics to evaluate the quality of a clustering, since some of them can produce misleading results for specific cluster shapes and distributions.

| No. noise Points | Fowlkes-Mallows | V-Measure | Calinski-Harabasz | David-Bouldin |
|---|---|---|---|---|
| 6 | 0.997 | 0.985 | 0.403 | 184.894 |

Table 5.15: Quality Scores obtained for each implemented metric of the Flat Clustering of the Circles Data set

| Homogeneity | Completeness | V-Measure |
|---|---|---|
| 0.998 | 0.972 | 0.985 |

Table 5.16: Completeness and Homogeneity of the Flat Clustering of the Circles Data set

### 5.2.3 5D Gaussians Data set

Tables 5.17 and 5.18 show the scores obtained for the run of the Flat Clustering Algorithm on the 5D Gaussians Data set using a density threshold of 0.0. We have obtained the same clustering depicted in level 1 of the respective Hierarchical Clustering. The Fowlkes-Mallows Index and the V-Measure show a small improvement in comparison to the previous scores. A special remark has to be made in the V-Measure, in which the reduction of the Homogeneity Score was compensated by the increase of the Completeness Score, resulting in the general improvement of the final score. The Calinski-Harabasz Index improved significantly as well, mainly due to the reduction of the number of clusters found within the data set. The same cannot be said for the David-Bouldin Index, which gives a worse score in comparison to the previous run, due to the fact that our clusters have more sparsity values than the ones generated in the Hierarchal Cluster Run. This actually makes sense since very small clusters will tend to have less sparsity values due to the nearest neighbors' connections, causing the average distances to the centroid of the cluster to be reduced significantly. This shows us the main disadvantage of using the David-Bouldin Index in our implementation, since it seems to be rewarding insignificant cluster areas, which don't provide relevant information.

| No. noise Points | Fowlkes-Mallows | V-Measure | Calinski-Harabasz | David-Bouldin |
|---|---|---|---|---|
| 1 | 0.99 | 0.998 | 16,838.2 | 0.387 |

Table 5.17: Quality Scores obtained for each implemented metric of the Flat Clustering of the 5D Gaussians Data set

| Homogeneity | Completeness | V-Measure |
|---|---|---|
| 0.997 | 0.998 | 0.998 |

Table 5.18: Completeness and Homogeneity of the Flat Clustering of the 5D Gaussians Data set

### 5.2.4 HTRU2 Data set

Finally, the scores for the Flat Clustering Algorithm run on the HTRU2 Data set using a density threshold of 0.1 can be found in Tables 5.19 and 5.20. We have again obtained the same clustering depicted in level 1 of the respective Hierarchical Clustering. The Fowlkes-Mallows Index could be interpreted as having a really good clustering, since approximately 90% of the points match the predefined labels. This is a questionable result when checking the value of the V-Measure, which on the other hand is quite low. This disagreement in scores is the result of the skewness of the predefined labels in the data set, something that the V-Measure implicitly takes into account trough the Homogeneity and Completeness Scores. The Fowlkes-Mallows Index on the other hand, only checks that the cluster labels match the predefined labeling and in a case like this, in which the label distribution is heavily skewed towards 90% of the data, grouping all points into one cluster will result into a score of approximately 0.9.

The Calisnki-Harabasz Index however, shows again a significant improvement, even though there is a significant amount of noise points. It seems that the points flagged as noise are actually part of a well defined cluster, although not dense enough to be flagged as such. For similar reasons, the David-Bouldin Index shows also an improvement although smaller in comparison.

| No. noise Points | Fowlkes-Mallows | V-Measure | Calinski-Harabasz | David-Bouldin |
|---|---|---|---|---|
| 1,916 | 0.901 | 0.24 | 3215.17 | 0.624 |

Table 5.19: Quality Scores obtained for each implemented metric of the Flat Clustering of the HTRU2 Data set

| Homogeneity | Completeness | V-Measure |
|---|---|---|
| 0.257 | 0.229 | 0.24 |

Table 5.20: Completeness and Homogeneity of the Flat Clustering of the HTRU2 Data set

Another way to evaluate how well the set of noise captures the minority class is through a confusion matrix, whose results are presented in Table 5.21. These results deliver a precision value of 0.51 and a recall value of 0.597 with a final F1-Score of 0.55. Considering the heavy skewness of the data set and the capture of almost 60% of the minority class, we conclude that our Clustering Algorithm did a relatively good job in identifying those areas were the minority class is mostly predominant and therefore provided a clustering or relatively good quality.

| Label | Label 1 | Label 0 |
|---|---|---|
| Noise Cluster Label | 978 | 661 |
| Other Cluster Labels | 938 | 15,321 |

Table 5.21: Confusion Matrix used to evaluate how well our implementation manages to capture the minority class of the HTRU2 Data set.

# 6 Conclusions and Future Work

In this thesis, the Sparse Grid based Clustering Models were integrated into the SG++ Datamining Pipeline by implementing Pehertorfer's Sparse Grid based Clustering Algorithm along with Fischer's Hierarchical Clustering augmentation. To solve the problem of efficiently generating the nearest neighbors' graph, we implemented a Vantage Point Tree data structure that not only solved the problem, but also provided us with an indexing of the data, which was used to reference the data points across other data structures. This resulted in an efficient use of memory, since we avoided copying the data across the other data structures.

Regarding the training of the model, we have introduced a new PostProcessing Module, whose main objective is to execute those processes unrelated to Sparse Grid Methods but which are still part of the training of the model. Adaptations to the other already implemented models were made accordingly in order to maintain their correct functionality.

We also implemented a series of clustering metrics used to evaluate the quality of the Clustering Models and made adaptations to the pipeline's structure in oder to integrate them without affecting the functionality of the other previously implemented metrics.

Finally, we tested our implementation using four different labeled data sets, with one of them containing real data from another study, namely the HTRU2 Data set. Our results showed that our implementation works well with different data shapes and distributions. Special remarks have to be done regarding the HTRU2 Data set, in which our implementation managed to make a good generalization of the data's internal structure, despite the data distribution being heavily skewed towards one of the predefined classes. Our results also showed the usefulness of having different metrics to evaluate the quality of our clustering, since some of them tend to give misleading results when the data has a specific distribution.

Further improvements could still be made to our implementation. For instance, the process realized in the tests, in which we selected the best density threshold to obtain the best flat clustering, could actually be automatized through the help of the Hyperparameter Optimizer Module.

Additionally, certain regions of the code could be also parallelized to increase the efficiency of our implementation. This would be specially useful during the generation of the Hierarchical Clustering, since we detected that this process was the major

bottleneck during the execution of the tests.

Another feature which could be added in the future is the implementation of other nearest neighbor algorithms and data structures. A suggestion would be implementing the Locality Sensitive Hashing algorithm, since it has been proven previously to be very efficient when generating a nearest neighbors' graph out of a large high dimensional data set [38].

Finally, new clustering metrics should also be implemented, specially ones that can handle cluster structures similar to those found in the Circles Data set.

# Bibliography


[1] V. B. B. Anguiano. "Visualization of High Dimensional Models within the SG++ Data Mining Pipeline." Studienarbeit. Technical University of Munich, Oct. 2019.

[2] S. Ben-David and M. Ackerman. "Measures of Clustering Quality: A Working Set of Axioms for Clustering." In: *Advances in Neural Information Processing Systems 21*. Ed. by D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou. Curran Associates, Inc., 2009, pp. 121–128.

[3] C. Braune, S. Besecke, and R. Kruse. "Density Based Clustering: Alternatives to DBSCAN." In: *Partitional Clustering Algorithms* (2014).

[4] C. Braune, S. Besecke, and R. Kruse. "Density Based Clustering: Alternatives to DBSCAN." In: *Partitional Clustering Algorithms*. Ed. by M. E. Celebi. Springer International Publishing, 2015, pp. 193–213.

[5] T. Caliński and J. Harabasz. "A dendrite method for cluster analysis." In: *Communications in Statistics* 3.1 (1974), pp. 1–27.

[6] D. L. Davies and D. W. Bouldin. "A Cluster Separation Measure." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-1.2 (Apr. 1979), pp. 224–227. ISSN: 1939-3539.

[7] D. Dua and C. Graff. *UCI Machine Learning Repository*. 2017.

[8] M. Fischer. "A Recommender System using Clustering with Sparse Grid Density Estimation." Technische Universität München, 2016.

[9] E. B. Fowlkes and C. L. Mallows. "A Method for Comparing Two Hierarchical Clusterings." In: *Journal of the American Statistical Association* 78.383 (1983), pp. 553–569.

[10] J. H. Friedman, J. L. Bentley, and R. A. Finkel. "An Algorithm for Finding Best Matches in Logarithmic Expected Time." In: *Acm Transactions on Mathematical software* 3 (1977), pp. 209–226.

[11] D. Fuschgruber. "Integration of SGDE-based Classification into the SG++ Datamining Pipeline." Technische Universität München, 2018.

[12] J. Garcke. "Sparse Grids in a Nutshell." In: *Sparse grids and applications*. Ed. by J. Garcke and M. Griebel. Vol. 88. Lecture Notes in Computational Science and Engineering. extended version with python code `http://garcke.ins.uni-bonn.de/research/pub/sparse_grids_nutshell_code.pdf`. Springer, 2013, pp. 57–80. DOI: 10.1007/978-3-642-31703-3_3.

[13] Z. Gerhard W. "A Sparse Grid PDE Solver; Discretization, Adaptivity, Software Design and Parallelization." In: *Advances in Software Tools for Scientific Computing*. Ed. by H. P. Langtangen, A. M. Bruaset, and E. Quak. Springer Berlin Heidelberg, 2000, pp. 133–177.

[14] M. Halkidi, M. Vazirgiannis, and Y. Batistakis. "Quality Scheme Assessment in the Clustering Process." In: *Principles of Data Mining and Knowledge Discovery*. Ed. by D. A. Zighed, J. Komorowski, and J. Żytkow. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 265–276.

[15] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. "Cluster Validity Methods: Part I." In: *SIGMOD Record* 31 (July 2002).

[16] S. Hanov. *VP trees: A data structure for finding stuff fast. Steve Hanov's Blog*. URL: `http://stevehanov.ca/blog/index.php?id=130`.

[17] M. Hegland, G. Hooker, and S. Roberts. "Finite element thin plate splines in density estimation." In: *ANZIAM Journal* 42.0 (2009), pp. 712–734. ISSN: 1446-8735.

[18] P. T. Inc. *Collaborative data science*. 2015. URL: `https://plot.ly`.

[19] E. J. Kopek. "Optimizing Hyperparameters in the SG++ Datamining Pipeline." Technische Universität München, 2018.

[20] N. Kumar, L. Zhang, and S. Nayar. "What is a Good Nearest Neighbors Algorithm for Finding Similar Patches in Images?" In: *Computer Vision ECCV* 2 (2008), pp. 364–378.

[21] S. Lloyd. "Least squares quantization in PCM." In: *IEEE Transactions on Information Theory* 28.2 (Mar. 1982), pp. 129–137. ISSN: 1557-9654.

[22] R. Lyon, B. Stappers, S. Cooper, J. Brooke, and J. Knowles. "Fifty Years of Pulsar Candidate Selection: From simple filters to a new principled real-time classification approach." In: *Monthly Notices of the Royal Astronomical Society* 459 (Apr. 2016), stw656. DOI: 10.1093/mnras/stw656.

[23] R. Lyon. *HTRU2*. Mar. 2016. DOI: 10.6084/m9.figshare.3080389.v1.

[24] U. Maulik and S. Bandyopadhyay. "Performance evaluation of some clustering algorithms and validity indices." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.12 (Dec. 2002), pp. 1650–1654. ISSN: 1939-3539.

[25] Q. Nguyen and V. Rayward-Smith. "Internal quality measures for clustering in metric spaces." In: *IJBIDM* 3 (Apr. 2008), pp. 4–29.

[26] S. M. Omohundro. *Five ball tree construction algorithms*. International Computer Science Institute, 1989.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[28] B. Peherstorfer. "Model Order Reduction of Parametrized Systems with Sparse Grid Learning Techniques." Technische Universität München, 2013.

[29] D. Pflüger. *Spatially Adaptive Sparse Grids for High Dimensional Problems*. 2010.

[30] A. Rosenberg and J. Hirschberg. "V-Measure: A Conditional Entropy-Based External Cluster Evaluation Measure." In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning(EMNLP-CoNLL)*. 2007, pp. 410–420.

[31] *SG++ ToolBox*. `https://sgpp.sparsegrids.org/`. Accessed: 2020-02-18.

[32] *SG++ User Guide. Datamining Pipeline Configuration*. `https://github.com/SGpp/SGpp/wiki/Datadriven-datamining-pipeline-configuration`. Accessed: 2020-02-26.

[33] B. Silverman. *Density Estimation For Statistical and Data Analysis*. Chapman and Hall, 1998, p. 1.

[34] S. Smolyak. "Quadrature and interpolation formulas for tensor products of certain classes of functions." In: *Dokl. Akad. Nauk SSSR* 148 (5 1963), pp. 1042–1045.

[35] *The Boost Graph Library: User Guide and Reference Manual*. USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN: 0201729148.

[36] S. Weber. "Exploiting the Data Hierarchy with Geometry Aware Sparse Grids for Image Classification." Technische Universität München, 2019.

[37] P. N. Yianilos. "Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces." In: (1993).

[38] Y.-M. Zhang, K. Huang, G. Geng, and C.-L. Liu. "Fast kNN Graph Construction with Locality Sensitive Hashing." In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by H. Blockeel, K. Kersting, S. Nijssen, and F. Železný. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 660–674.