POLITECNICO
DI TORINO

Politecnico di Torino

Master degree course in
Computer Engineering

Clustering algorithms for rock porosity categorization

Candidate: **Riccardo Callà**
Supervisors: **prof. Elena Baralis, prof. Paolo Garza, Dott. Andrea Pasini**

April 2019

Abstract

In this thesis we will discuss about the results of a research project conducted for Politecnico di Torino about the classification of geological pores in rock samples. We analyzed data collected by domain experts by using the most well-known clustering techniques of data mining. The aim is to understand which clustering algorithms give the best result in pores data categorization. In general, clustering studies sets of objects properties by finding relationships between data elements based on the similarity of their properties. In our case pores, have a great variety of attributes mainly based on their shape and size. An important aspect is that some of their attributes have values that are not uniformly distributed and this means that is possible to create groups where locate pores characterized by similar attributes values. For example, in nature there exist pores of very different sizes, so measuring their diameter is possible to create distinct groups by using this feature. The thesis will be organized in parts where we will present the problem in more detail, discuss the used techniques from a mathematical and implementation point of view, then the most significant results we have obtained. This study can be thought as a starting point for classifying the pores through the clustering methods. Moreover it can be used as an approach to inspect the feasibility of an automatic way to address the first part of the procedure to analyze rock permeability.

Contents

1 Thesis structure

This thesis was structured in the following way:

Chapter Introduction. In this chapter we discuss about the goal of this research and we give the information useful to understand the analysis discussed in the various chapters. In particular, we introduce the geological background with a brief description about the pores that were the object of study. Moreover, an introduction of the data mining methods explaining which of them we choose to achieve the results.

Chapter Details of the designed process. In this chapter we present our own methodology to approach the problem by defining the various steps of the pipeline that were followed to achieve the results. Then, there is a deep explanation of the adopted methods and algorithms. In particular, we report the mathematical defeinition and the main aspects of Kmeans, Hierarchical and DBSCAN. Moreover, other procedures that helped to improve the clusters quality like the *feature selection*, the *PCA*, or the *silhouette* analysis.

Chapter Implementation. In this chapter we discuss about the *python* environment used for the analysis and how we combined the python libraries with our custom code. Here we analyze the configuration of the clustering algorithms and the hardware restrictions that must be faced off when executing the code in real cases. Then, the strategies adopted to solve the raised problems and how we improve the execution of the algorithms. Moreover, we reported some information about the execution time a some quality tests about the different types of configuration.

Representative experimental results. In this chapter we present the most significative result we obtained for all the three experimented algorithms: Kmeans, Hierarchical and DBSCAN. We report different test cases and which of these algo-

rtihms returned the best results explaining the reasons through the plots we have configured.

Conclusion. In this last setcion we treated the steps in chronological order of how we produced the analyzes and the results obtained. We also provide some suggestions for future analysis that could be done to continue this research.

2 Introduction

2.1 Thesis objectives

This report describes the methodology and the procedures proposed for an applied research, focused on clustering of geological pores. All the analyzed data are in the form of structured datasets, where each dataset is a set of pores. In these data collections, each pore is described with a set of specific attributes, derived from its shape and size.

The objective of this research activity is to split each input set of pores in groups (i.e., clusters) of homogeneous pores, based on the attribute values characterizing the input pores. To achieve our goal, we considered the most well-known clustering algorithms. With this study we aim to identify which clustering algorithms are best suited to this task and type of data by validating the results together with domain experts.

2.2 Geological Background

We introduce some information to facilitate the understanding of the analyzed data. From the geological point of view, a pore is a cavity present in the rock that can have different sizes. In a sample of rock there are usually a lot of small pores and few big pores. Their presence allows the flow of liquids, such as water, and the displacement and accumulation of natural gas (Figure 1). The experts can retrieve information about the pores by analyzing sections of rock. In particular, with sophisticated photographic tools supplied with advanced software they are able to scan the characteristics of the pores present in a rock sample. For instance a lot of information about its dimensions can be retrieved. Then, all the relevant properties about each scanned pore are converted in numerical format and saved in a digital data structure. This data structure is exported as a csv dataset where each row contains information about a single pore and in the corresponding columns there are its attributes.

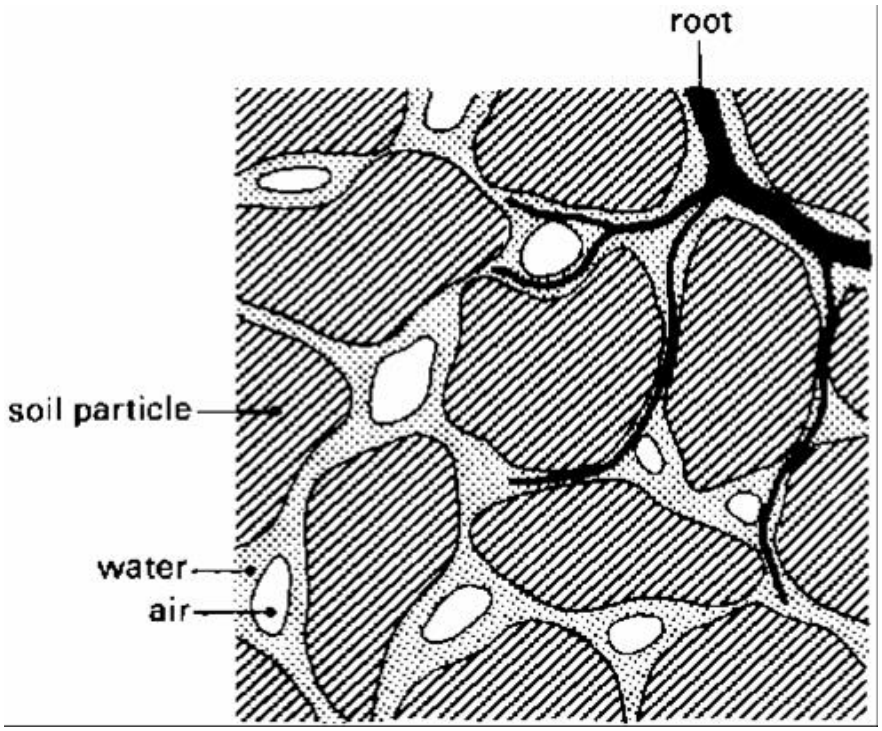


Figure 1: Rock pores

2.3 Data Mining background

Data Mining is a set of techniques and methods which aim to find useful information from huge quantity of data that can be for example big databases or big datasets like in our case. These procedures can extract (mining) hidden pattern or correlations that are impossible to be found by human brains. Nowadays many companies adopt this solutions to improve several aspects of the productive process. For instance, helpful information about customers can be obtained in order to increase the profitability of sales processes. We will present in details the techniques and algorithms we adopted in this research.

2.3.1 Data Mining process

We briefly introduce what are the common standard step typically involved in the Data Mining process. This process is call *Knowledge Discovery in Databases (KDD)*[1] and consists of some sequential stages aiming to discover in data the desired information and hidden patterns as we explained previously. Therefore, first we will give a description of the general stages in the schema of Data Mining application, then we are going to discuss in details our personal method adopted for the purposes of this research.

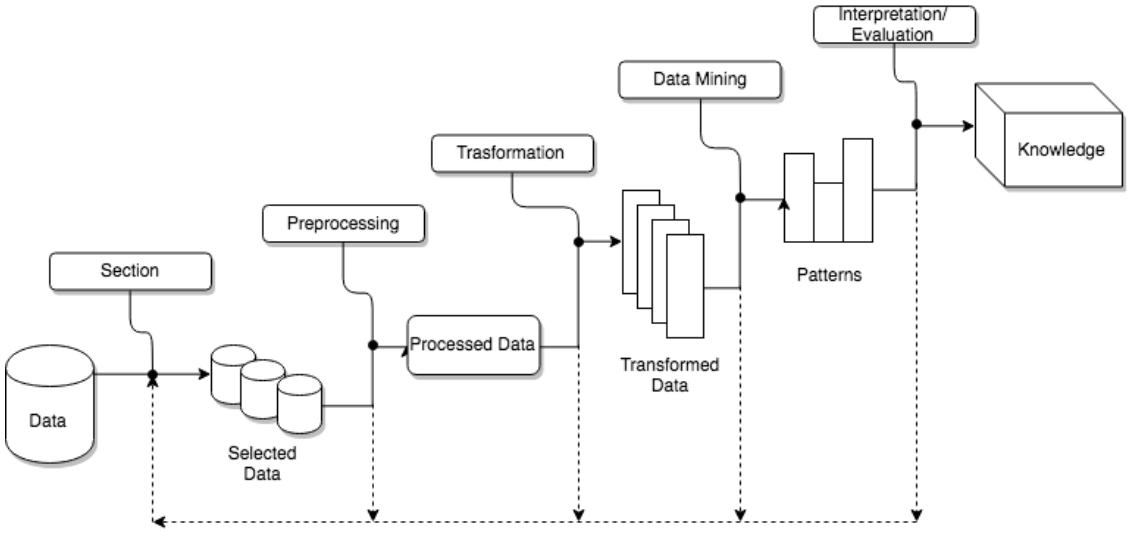


Figure 2: KDD pipeline

Selection. At the beginning the input data requires a preliminary step where experts choose its content by satisfying the application domain and the final purpose application requirements.

Pre-processing. In this step is executed a data selection. It means to decide which part of data need to be discarded for the further steps. It is very important to have suitable data in order to reduce probability of errors, improve the result accuracy and reduce the usage of computational resources. Data can be discarded for many

reasons. One is the presence of noise that if not omitted can lightly falsify the final result because automatic procedures can give to the noise the same importance as to the useful data.

Transformation This is a fundamental step because the data is prepared and transformed to be ready to use for the next step. Since in Data Mining techniques exist a lot of different procedures and each of them require a defined data format, it is important to know already which of the procedures will be used in the next step.

Data mining. It is the main step of this chain because here the Data Mining procedures are executed. As mentioned above the current data is ready to be analyzed by using the selected method. Details of the different Data Mining approaches will be discussed later on. Now we can simply consider that after the procedure execution we obtain the patterns that will be analyzed in the next step.

Interpretation/evaluation. The final step is where the found patterns are analyzed. Usually it means that they are edited and printed in plots or tables. The goals of this step is to create the best possible visual representation in order to help the readability and the comprehension of the final result.

2.3.2 Data Mining approaches

As mentioned above Data Mining is a set of techniques and procedure that aim to extract useful information and hidden patterns from huge quantity of data. In this section we highlight three of the most important well-known methods. They are association rule mining, classification and clustering.

Association rule mining. Consists of finding any kind of relationship in the same bunch of data. For example, is mainly used by companies transactions to find correlation between the purchased products by customers.

Classification. It is one of the most used methods. The input data consists in a

domain of defined samples that are used to predict other non defined cases in the same domain. It is also called *Supervised* problem because the decision about the undefined data is taken by training on the starting labeled data. For this reason the train data is strictly necessary to use this methodology. Several approaches exists in this set like statistics, neural networks,linear programming or decision Tree.

Example of classification are:
(i)**Decision tree**. Decision tree are based on multiple conditions which guide data and expanding it from the root node to the final decisions on the terminal leaves. In particular, each internal node is a sub condition of the root. A path is built selecting all the internal conditions present in the visited nodes. At the end of this path, when the leaf is reached, the tree gives the final decision.
(ii)**Neural network**. It is a network of logic neurons hosted by a machine. Its structure represents the real human brain network of neurons and like it is able to learn from the input train data and predict the label on test data.

Clustering. It is very similar to Classification. In fact, giving a set of objects it produces different kinds of groups called *clusters* to which the object are assigned. Moreover, these groups can be formed by analyzing the input data, comparing the samples properties and find most relevant similarities among them. The big difference from the Classification method is that, in this case, there aren't defined starting groups where to put the objects, otherwise they are created by considering only the input data. Result that there is no dependency with additional data as happened with the training data of the classification method.

For our purpose we adopted the Clustering method because, as explained before, the aim of this research is to analyze and evaluate the different possibilities of classifying the pores in the datasets. Since do not exist predefined groups for this domain, the approach we choose is the Clustering method. In the next sections we will describe in details the clustering algorithms and measures adopted for this study.

3 Details of the designed process

3.1 Proposed methodology

This section presents the main building blocks of the proposed methodology, depicted in Figure 3. The whole procedure is applied separately to each dataset, in order to obtain clusters which are specific of a single geological sample.

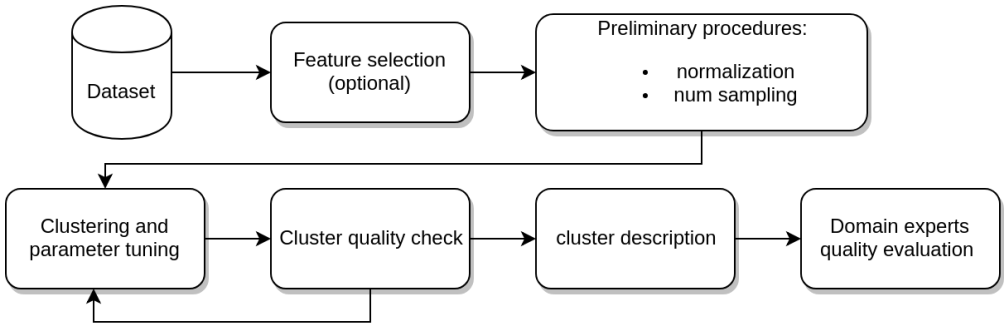


Figure 3: Clustering pipeline

Feature selection. This is the first (optional) step, where all the attributes (also called *features*) that are not suitable for the following analysis are removed. This phase is performed by analyzing the characteristics of the attributes such as their value distribution and their mutual correlations. Initially, we remove the attributes which are useless for the analysis, since they are not related to the description of the pore shapes (e.g. pore id, pore position). Afterwards we automatically drop all the features which present high statistical correlations among each other, as they would provide redundant information. This process allows the reduction of the dataset size and a more proper working of the distance measures used for clustering.

Preliminary procedures. The following step consists of dataset transformations to be applied before the clustering algorithms. Since the clustering phase requires many hardware resources (in terms of memory), if necessary the number of points is

reduced by applying (i) a filtering operation based on the pore size or (ii) random *sampling* techniques. Secondly, we *normalize* the training data in order to avoid biases related to different feature ranges. Each data attribute is normalized with *Z-Score* that consists of subtracting the population mean from an individual raw score and then dividing the difference by the population standard deviation. More formally:

$$Z = \frac{X - \mu}{\sigma}$$

Clustering and parameter tuning, **cluster quality check**. At this point our process introduces a loop between *clustering* and *cluster quality check* building blocks. At each iteration we select one of the available state-of-the-art clustering algorithms and inspect the results by varying its parameters (for example the desired number of clusters or the neighborhood radius for density based algorithms). The clustering process labels the dataset points with the assigned group and saves the results to a file. Afterwards a *cluster quality* check is performed by analyzing the generated clusters with quality measures such as *SSE*, *silhouette* and *PCA 3D representations*.

Cluster description. This building block is responsible for obtaining interpretable descriptions of the generated clusters. We use representations which allow inspecting the cluster size, the *centroid* positions (center of gravity of each cluster) and the attributes which are more significant to characterize each cluster (by means of *decision trees*).

Domain experts quality evaluation. The last step of the process involves the intervention of *domain experts* to visualize and inspect the characteristics of the clusters obtained so far. Here domain experts may use the *cluster descriptors* produced in the previous phase to ease the analysis. If the results are not sufficiently coherent with the geological interpretation, the whole process is run from the beginning in order to find a better configuration of each building block of the process.

3.2 Feature selection

Feature selection is defined as the process of selecting a subset of relevant features (i.e. attributes) among the original ones. There are three main reasons why feature selection is effective when applied before clustering.

A smaller number of attributes implies (i) an easier understanding of the cluster shapes (ii) a reduced dataset size, which makes shorter the execution time and requires less hardware resources (iii) better quality of the distance metrics used to assign data to each cluster. Indeed, higher number of features entails a well known issue called *curse of dimensionality*[2]. When data points present a very high dimensionality they tend to have approximately the same distance among each other, even if they have one or more dimensions which are very different.

The feature selection analysis presented in the following sections is performed on a first set of datasets provided by domain experts. The dataset names are: *a1, a2, a3, b1, b2, b3*. The feature selection step has not been applied on the other datasets.

3.2.1 Attribute correlation

In our analysis we considered the statistical attribute correlation in order to choose the most relevant features. Correlated attributes introduce redundant information which could damage the performances of the clustering algorithm. Feature selection can be applied to correlated attributes in order to reduce the redundant information.

We compute the correlation between attribute pairs with the *Pearson* [3] metric. This measure produces values between -1 (high negative correlation) and $+1$ (high positive correlation). Two attribute are correlated when the absolute value of the Pearson correlation is close to 1.

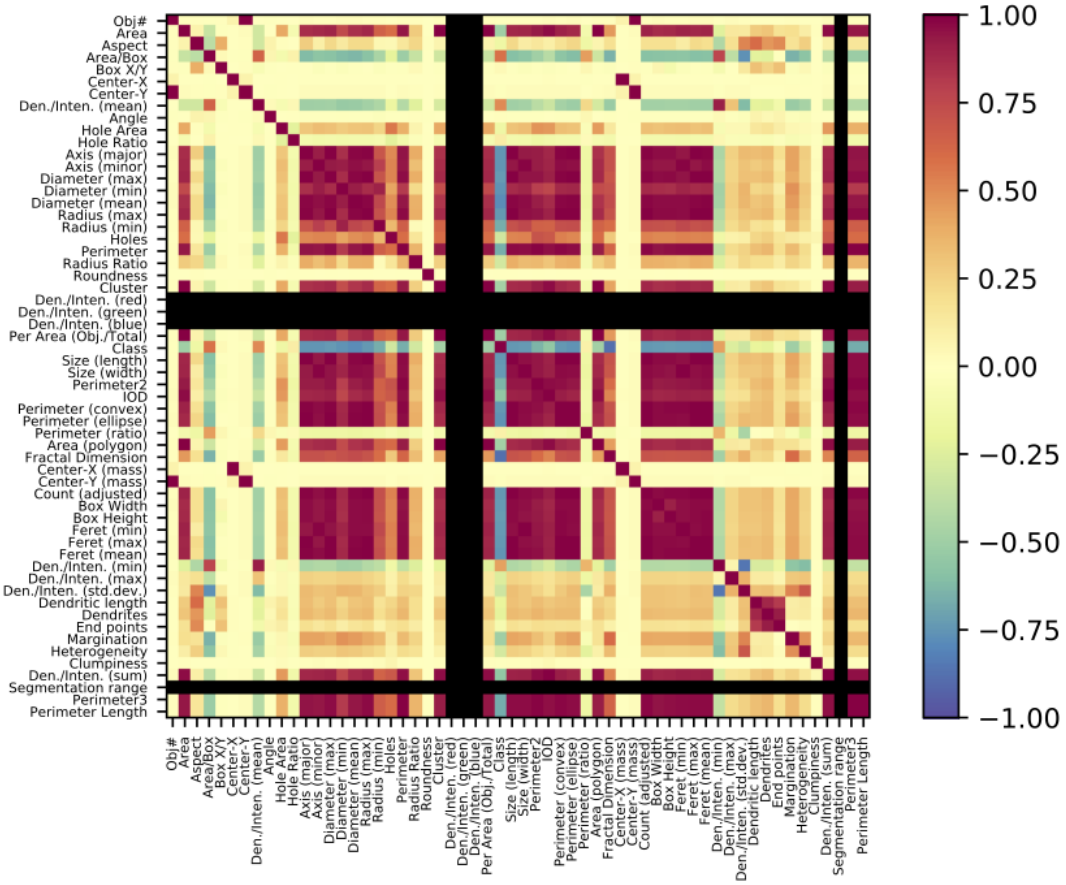


Figure 4: Correlation attributes, dataset: a1

Figure 4 represents the correlation matrix for an example dataset (a1). The color in each cell encodes the Pearson correlation between two attributes. Black cells imply that the correlation cannot be computed from the selected attributes (because they present incomplete information or missing values). Cells on the diagonal are always red, since they represent the correlation of an attribute with itself.

To decide which attribute pairs should be considered as highly correlated, we define a correlation threshold (in absolute value). The value of this threshold is the same for all the datasets under analysis. It is computed by looking at the distribution of the

absolute value of the correlations among the attribute pairs in each dataset.

Figure 5 shows the correlation for each attribute pair across different datasets. Each curve represents the values for a specific dataset. The correlation is plotted on the vertical axis, while the horizontal axis represents all the possible attribute pairs (sorted by increasing absolute correlation). At the correlation value 0.85 it is possible to observe an elbow in the shape of most of the curves. This means that for values greater than the *threshold* 0.85 all the datasets contain attribute pairs with similar correlation coefficients, closer to 1. The obtained threshold is used in the following section to group together attributes and perform feature selection.

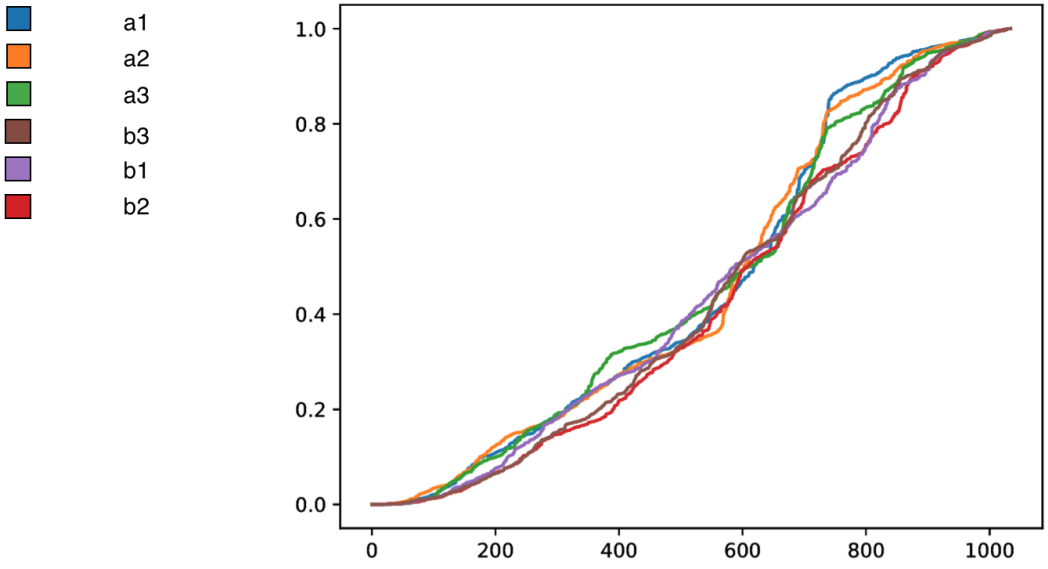


Figure 5: Correlation curves

3.2.2 Correlation families and leaders

As presented before, attributes with high correlations bring redundant information and can be grouped together. We introduce the following notation.

Family. A *family* is a set of attributes which present high correlations. Each attribute belonging to the family has a correlation greater than the selected threshold (0.85) with at least another attribute in the family. Since attributes of the same family bring similar information, we only keep the most representative one.

Family leader. We call *family leader* the selected representative attribute for a specific family. The attribute selected as family leader is the one with the highest correlation with the other attributes in the family and the lowest correlation with the attributes of other families.

Singleton. We call *singleton* the family leader which belong to families composed of a single element. The dataset *a1* contains for example 32 correlated attributes and 14 singletons.

The feature selection process applied to a specific dataset keeps as important features only the family leaders. Figure 6 shows the 17 selected attributes (family leaders) among the initial 46 for dataset *a1*.
From this chart we can also observe that the correlations among the different attribute pairs are very low after the application of feature selection. This implies that the redundant information is reduced drastically.

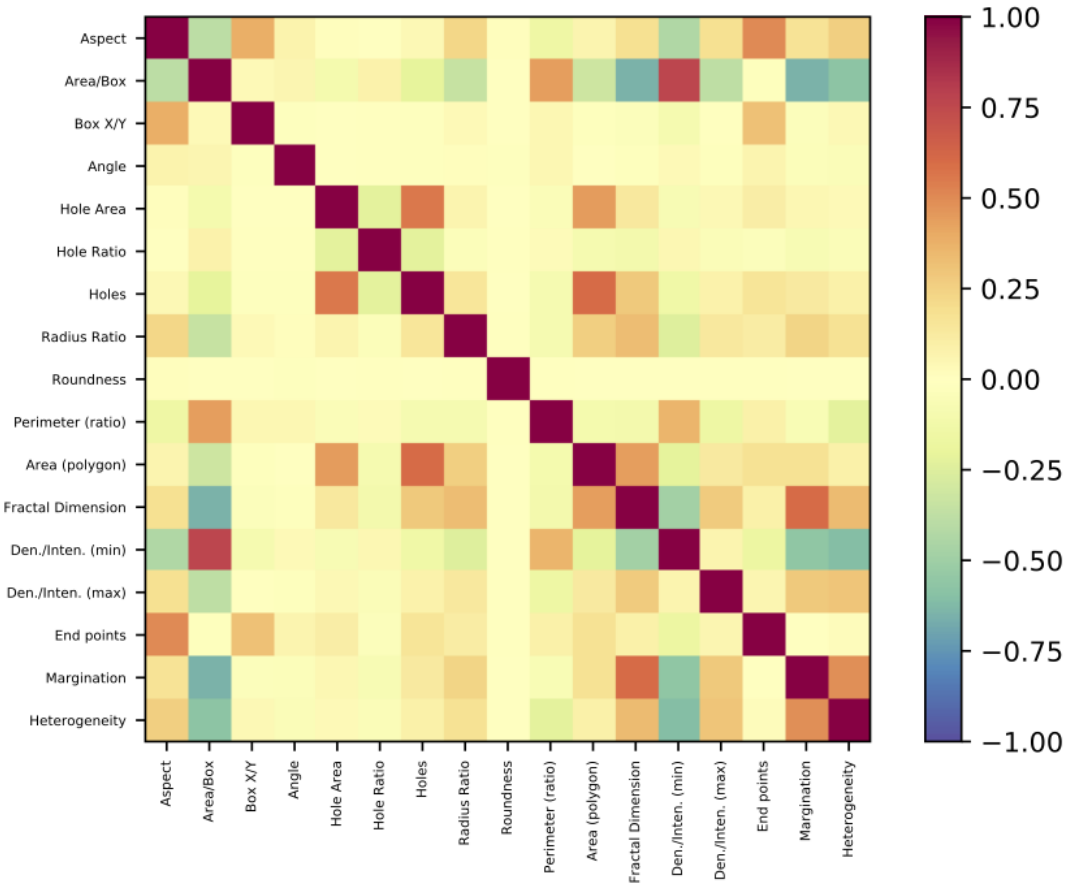


Figure 6: Selected Attributes, dataset: a1

Since the list of initial attributes for the different datasets is the same, we expect to find similar families and leaders. From the analysis we found 13 common group leaders, which are shared among all datasets. The common group leaders are: *Angle, Area/Box, Aspect, Box X/Y, Den./Inten. (max), End points, Fractal dimension, Heterogeneity, Hole Ratio, Margination, Perimeter(ratio), Radius Ratio, Roundness.* All these 13 leaders are singleton, except for *End points*, whose family is composed by *Dendrites, Dendritic length and End points.*

Another observation common for all datasets is that many of the attributes de-

scribing the size of the pores are grouped in the same family (e.g. *Area, Diameter, Feret, Perimeter*). For this reason in each dataset there is always a big family (about 10 attributes) of attributes describing the pore size.

3.3 Clustering Methods

A *clustering algorithm* takes as input a set of data points associated with their attributes and produces as output a partitioning of the input data in groups called *clusters*. Hence, each sample is labeled with the identifier of the cluster to which it is assigned. We considered only clustering algorithms that generate non-overlapping clusters (i.e., each point is assigned to one single cluster). A brief description of the algorithms that we used for the analyses is provided in the next subsections.

3.4 Clustering Metrics

The clustering metric represents the distance between a pair of samples. It is fundamental because it establishes a criterion with which to compare the various points, in fact the chosen metric influences the clusters shape. We will describe the main characteristics of the most famous metrics adopted by the clustering algorithms.

Euclidean distance. In a 2D space the distance between two points is the segment that connects the two points. In a multi-dimensional space, with more than 2 dimension, the distance is obtained by adding the distances for each dimension.

$$||a - b||_2 = \sqrt{\sum_i (a_i - b_i)^2}$$

Where a_i and b_i are the values of samples a and b for attribute i.

Squared Euclidean distance. It is very similar to the **euclidean distance** but does not have the square root. It means that it is faster to be executed from a

computational point of view.

$$||a - b||_2^2 = \sum_i (a_i - b_i)^2$$

Where a_i and b_i are the values of samples a and b for attribute i.

Manhattan distance. It works in a **taxicab geometry** that is an alternative to the **euclidean geometry**. In this geometry system there are no distances that directly connects two points of the space. For example, in a 2D space of this geometry, the distance between two points is the absolute difference between their x-axis values and their y-axis values. It means that does not exist a path that directly connect two points, otherwise this path would be composed by sub-paths that are parallel or orthogonal to the plane axis. The name "Manhattan" is inspired by this concept because represents the road network of the borough of New York City. In fact, it is known, having a chessboard shape where almost all the streets are parallel or orthogonal to each other. In the real case a taxi, that inspired the name "taxicab geometry", can't move directly from point A to B in the city, but it must follow the streets making a path similar to the one that is generated on the 2D-plane mentioned above.

$$||a - b||_1 = \sum_i |a_i - b_i|$$

Where a_i and b_i are the values of samples a and b for attribute i.

Chebyshev distance. As for the distance Manhattan, this is also based on the concept of chessboard. However, unlike the method described above, this one consider the center of the boxes instead of the length of sides. In fact, the distance between two points in the chessboard is the number of boxes that a king must cross, being able to move in all directions, but making the shortest path. From the mathematical point of view it can be represented by a uniform norm, in particular, as also shown in the

formula, the distance between two points is given by the maximum difference between their x-axis values and their y-axis values.

$$||a - b||_\infty = \max_i |a_i - b_i|$$

Where a_i and b_i are the values of samples a and b for attribute i.

Jaccard. The **Jaccard index** uses the math of sets to compare distinct objects. In particular, it calculates the similarity between two sets and higher the similarity, shorter the distance is between this two sets. The similarity is obtained with the **Jaccard similarity coefficient** that is defined as the intersection divided by the union of two sets:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Where A and B are two sets of objects. More precisely the distance between two sets is calculated with the complement of the Jaccard coefficient and it is also called **dissimilarity**:

$$D_j(A, B) = 1 - J(A, B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

Where D_j is the Jaccard distance, A and B are two sets of objects.

3.5 Clustering algorithms

3.5.1 Kmeans

This algorithm receives as parameter the expected number of clusters k. Each cluster is identified by a *centroid*. A sample is assigned to the cluster whose centroid is closer. Kmeans initially sets k centroids with random positions. At each iteration of the algorithm data points are associated with the closest centroid. Then centroids are updated towards the center of gravity of their cluster.

3.5.2 Centroids

The centroid of a cluster is defined as its center of gravity. It is computed as the mean of all the data points of the cluster:

$$centroid(c) = \frac{1}{|c|} \sum_{x_i \in c} x_i$$

where c is the considered cluster, $|c|$ is the number of points in c and x_i are the points associated with cluster c. Figure 7 depicts an example of bi-dimensional clusters with their associated centroids. Clusters are represented by colors, while centroids are depicted as black dots.

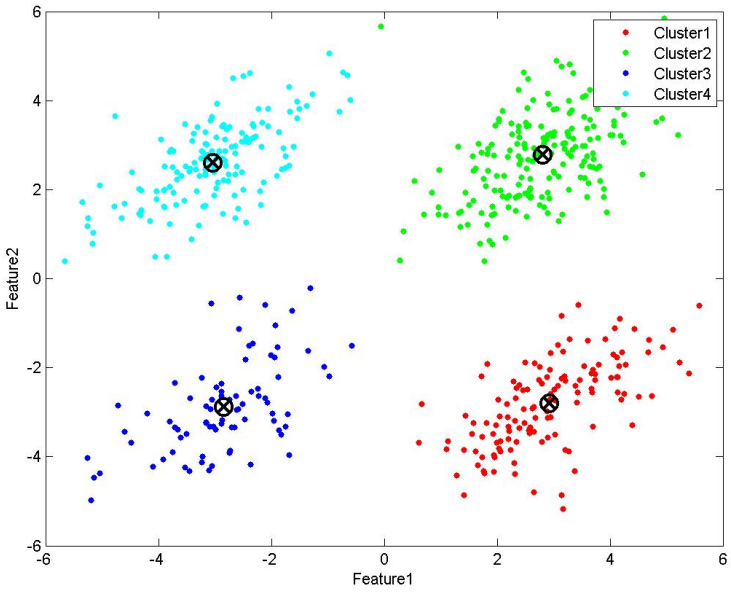


Figure 7: Example of centroids in a 2D Plot

3.5.3 DBSCAN

This algorithm groups together points based on their density in the space. Samples which are located in low-density regions (whose nearest neighbors are too far away)

are marked as *outliers* (cluster with label -1). The number of generated clusters is not known a-priori and depends on the data distribution. DBSCAN takes as input two parameters to model the concept of neighborhood and density between data samples. The first one (*epsilon*) specifies the radius of the neighborhood of a point. The second one (*minpoints*) specifies the minimum number of samples in the neighborhood of a point. Samples with less than *minpoints* neighbors are marked as noise (outliers).

3.5.4 Hierarchical

This algorithm generates clusters which are organized in hierarchy. The output hierarchy is encoded with a structure called *dendrogram*. This structure specifies the aggregations of the clusters at different levels. The algorithm does not take parameters as input. However, a user can specify k as the desired number of clusters in order to extract the grouping of the data samples at a particular level of hierarchy. Hierarchical algorithms work by repeatedly identifying and merging the two clusters which are the most close at a specific hierarchy level. Initially each point is considered as a specific cluster, while at the end of the hierarchy the algorithm ends with a single cluster.

3.6 Cluster Description

After labeling the data points with one of the proposed clustering methods, the characteristics of each cluster can be described in order to ease the analysis of the results. In the following paragraphs we briefly describe the techniques adopted for cluster description.

3.6.1 Silhouette

Since clustering is an unsupervised technique, measuring the quality of the results is often a difficult task. *Silhouette*[4] is a metric defined to measure the characteristics of each cluster without having prior knowledge on the expected clusters. This measure takes values between -1 and 1. Values closer to 1 imply a good quality of the clusters.

Silhouette is computed by inspecting the cohesion of the samples inside each cluster and their separation from the samples in other clusters. *Intra cluster separation* and *inter cluster separation* are the two components used to build the Silhouette value. The *intra cluster separation* of a cluster c is defined as:

$$intra(c) = \sum_{x_i \in c} \sum_{x_j \in c, i \neq j} ||x_i - x_j||^2$$

where x_i and x_j are two points of the cluster c and $||x_i - x_j||^2$ is the squared distance between the two points. Higher values of the intra cluster separation imply a lower density of the cluster, since its points are more distant among each other.

The *inter cluster separation* of a cluster c is defined as follows.

$$inter(c) = \sum_{x_i \in c} \sum_{x_j \notin c} ||x_i - x_j||^2$$

where x_i is a point of cluster c and x_j is a point of any other cluster. Higher values of the inter cluster separation imply a higher distance between the specified cluster and the other ones. The *silhouette score* gathers these two contributions. It is defined as follow:

$$silh(c) = \frac{inter(c) - intra(c)}{max(inter(c), intra(c))}$$

Higher values are obtained when the clusters have a relative higher inter separation with respect to the intra separation.

The intra/inter cluster separation can also be computed globally for all the clusters:

$$intra = \sum_{c \in C} intra(c)$$

$$inter = \sum_{c \in C} inter(c)$$

where C is the set of all clusters.

With these definitions it is possible to compute the global silhouette:

$$silh = \frac{inter - intra}{max(inter, intra)}$$

The global silhouette allows understanding the global quality of the produced clusters. For our analysis we will visualize charts inspecting the global silhouette, the silhouette on each cluster and the intra/inter cluster separation values.

3.6.2 Decision trees

Decision trees are classification models designed to classify data based on the values of the input attributes. Although these models are typically used for classification, here we use them to describe the clusters produced by each of the algorithms presented in the previous section (i.e. Kmeans, DBSCAN, Hierarchical).

For this purpose we train a decision tree to classify each sample with the cluster identifier assigned by the clustering method under inspection. After the learning process, the generated decision tree performs a sequence of tests on the input attributes to decide the cluster identifier. The test sequence can be visualized as a tree graph, where each internal node represents a "test" on an attribute, each branch selects a possible outcome of the test, and each leaf node represents a class label (cluster id). Tests near to the tree root perform coarser splits among high quantities of samples. Tests near to the leaves perform finer splits trying to assign the correct cluster id for most of the samples. The objective of the learning phase of a decision tree is to grow until the leaves contain only samples of a specific cluster. Such objective would require trees with a very high number of leaves. For this reason we stop the growing of the tree by setting a parameter of the algorithm called *maximum gain*. In this way the objective of the training becomes minimizing the impurity of data inside each leaf. Leaves with lower impurity contain samples which belong mainly of a specific cluster id. The impurity is measured with a value called *Gini index*[5].

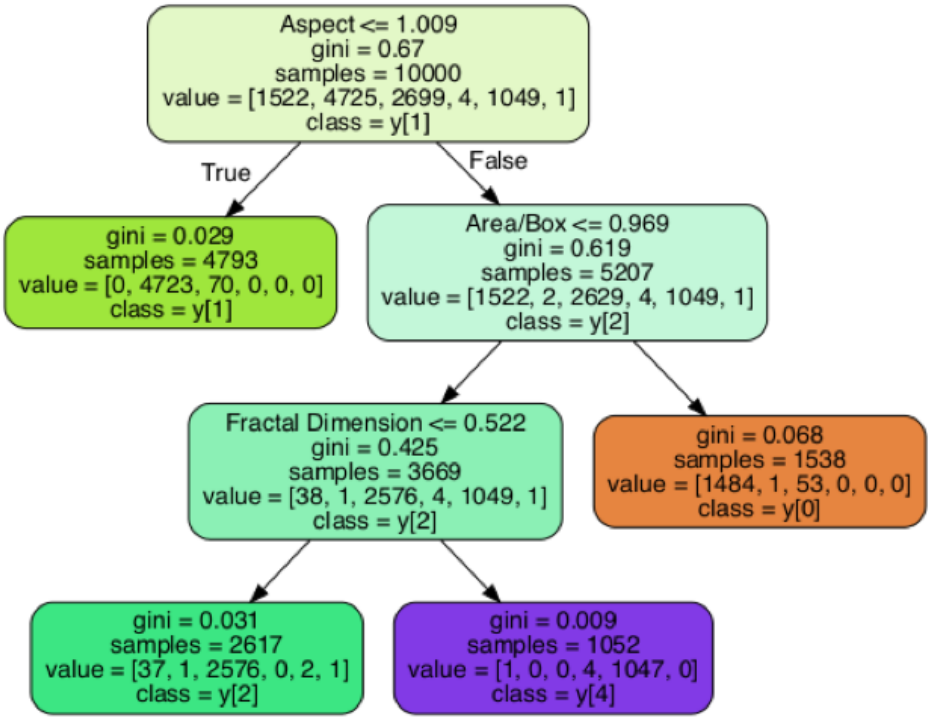


Figure 8: Example of decision tree, the leaves represent the final clusters

Figure 8 shows an example of decision tree employed for cluster description. From Figure 8, we can understand that the most relevant attribute is *Aspect*, since it appears at the root of the tree. This attribute is exploited to split all the samples in two groups. Samples with a value of $Aspect <= 1.009$ flow through the left branch, while the others flow through the right one.

If we consider the orange leaf, we can observe that the number of samples which flow towards this bucket is 1538. Among these, 1484 belong to cluster 0, 1 to cluster 1 and 53 to cluster 3. The impurity of this node is low ($gini = 0.068$), since most of the samples belong to the same cluster.

3.6.3 3D representation with PCA.

The *Principal Component Analysis (PCA)* is a linear dimensionality reduction technique based on *Singular Value Decomposition (SVD) [7].* Every data point is projected to a lower dimensional space, with less features than the original one. Each of the new generated features can be obtained from the original ones by applying a linear transformation. By specifying three output features, PCA can produce a three-dimensional representation of the sample distribution. Moreover we can enhance this visualization technique by coloring the points according to the cluster which they are assigned to. From these kind of charts experts can visually inspect whether the clusters are well separated in the space and their characteristics such as density or shape.

Figure 9 shows an example of dataset where PCA was applied to derive the 3D representation. The three colors (yellow, green, blue, purple) represent the generated clusters. For example we can observe that the yellow cluster is the most dense among the others and that the purple cluster is smaller, with a lower density.

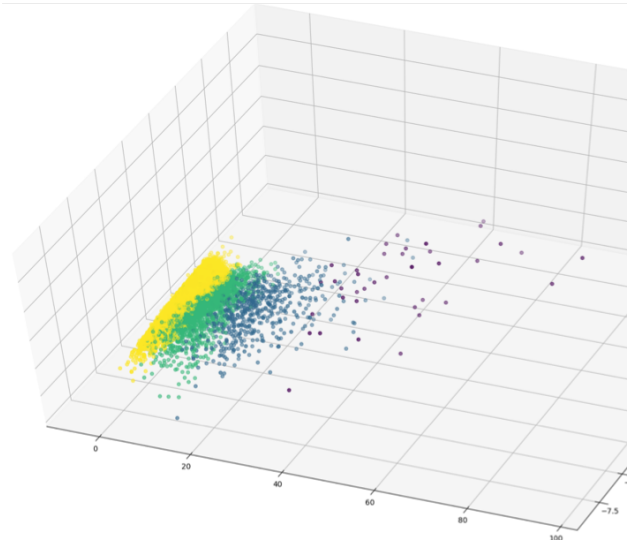


Figure 9: Example of PCA visualization, different colors represent the clusters

4 Implementation

In this section we will analyze the clustering procedures described above from the point of view of the implementation. In particular, what library we used to implement the technique, the extra code we add to build the result and our consideration about issues and problems that occurred in the development process. In Table 1 the details of the hardware we used to execute tests and analysis.

cpu	Intel® Core™ i7-8700K (5 GHz)
ram	G.Skill Trident Z - DDR4 16GB - (3200 MHz)
hard drive	SSD Crucial MX500
os	Windows 10
python	2.7.9

Table 1: Hardware configuration

4.1 Environment

The whole application development for this research was done in Python. There are many reason why we adopted this language to implements the algorithms and procedures. Python is an high-level programming language mainly based on scripts and can be used for any kind of programming purpose. Its easy syntax and the good readability have helped to grow its fame during the last years, making it one of the most used languages today. Moreover, there are some other advantages like that it is free and the cross-platform compatibility. In our specific case we found a very advanced community from the point of view of data mining. In fact, exist some useful open-source libraries that have already implemented some basic code constructs that helps to faster build complex algorithms. We will discuss which are the main libraries we

used in the next section.

4.2 Libraries

This section describes the most important libraries we used to implement the code.

4.2.1 Managing data

Pandas. Is a library designed for the management of data structures. In fact, it provides several tools that facilitate the reading, editing and writing of relational data. It was very useful to import the datasets in the csv format and to store it content in the data structures where can be edited and analyzed.

Numpy. Is instead a fundamental package for scientific computing with Python that allows to easily manage arrays and execute functions in multi-dimensional data.

4.2.2 Scientific

Scikit-learn. Is the main library of the project that provides all the functions to implement the classification and the clustering algorithms. **Scipy**. Is another support library, useful to manage the Hierarchical strategy.

4.2.3 Visualization

Matplotlib This library allows to visualize all the result in plots.

4.3 Implementation process

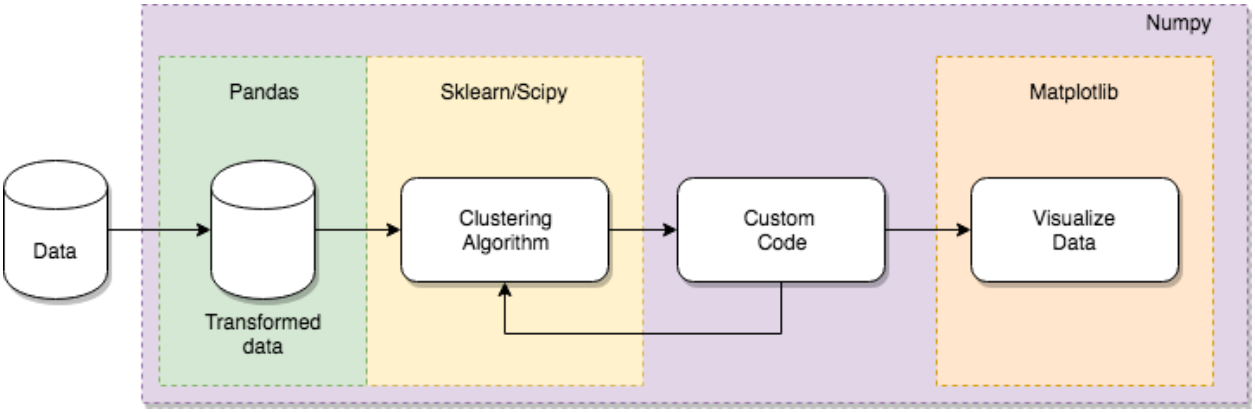


Figure 10: Implementation Diagram

In Figure 10 the common steps about the implementation process and how the support tools and libraries were adopted through the various stages:

Data. The starting point is to retrieve the data that must be analyzed and in our case this data is in csv format.

Transformed Data. In this step we use Pandas to import the source csv in a Pandas structure. After that it is possible to use this data structure for sampling or just going to the next step where to execute clustering algorithms.

Clustering algorithm. Here we execute algorithms taken from the scientific libraries. In most of the cases is the Sklearn library.

Custom Code. At this points we implement our custom code with the goal to analyze the given result from the previous steps and prepare the data to be visualized or

reused as input for an other clustering algorithm step.

Visualize Data. This is the last step where we collect and transform the results in suitable data structures that are going to used as input for the Matplotlib library. Then, the plot that describes the desired analysis is shown.

4.4 Kmeans

This section represents the details for the Kmeans clustering implementation. A summary is proved in Table 2.

Library	SkLearn
Average time complexity	$O(knT)$
Worst case time complexity	$O(n^{k+2/p})$
Original time complexity	$O(n^{dk+1})$

Table 2: Kmeans clustering properties

Listing 1: In first row the data structure of Kmeans is generated by setting the number of cluster K and the seed, in the second and third is fit the data structure created before in the fourth row are extracted the labels of the generated clusters.

```
1 km = KMeans(k, random_state=seed)
2 X = dsNorm.values
3 km.fit(X)
4 labels = km.labels_
```

4.4.1 Development and tuning

Kmeans clustering initialize k centroids and assign each point to the closest centroid. At the beginning the algorithm generates k centroids with random values for each of the attributes. In each iteration a point is assigned to the closest centroid. Centroids attribute values are updated by calculating the mean of the points inside its cluster algorithm belong to the NP-hard problem and has a complexity of $O(n^{dk+1})$, briefly means that the whole execution can not be performed in a polynomial time. However the sklearn implementation of this algorithms uses Lloyd's [8] or Elkan's [9] variation approach that optimize its' execution. In fact, its computational time is $O(n^{k+2/p})$ for the worst case where n is the number of samples, p is the number of features. $O(knT)$ is the average time, were n is the number of samples and T is the number of iterations. Thanks to the fast convergence of the algorithm there are low computational costs and short execution times, so its it very suitable to test big dataset without sampling them. However, since it requires a seed to initialize the centroid it could give very different result when changing these values. In our tests we choose a fixed seed that gives the Kmeans always the same initialization in order to compare later the results obtained in different execution times.

4.4.2 Execution Time

We measure the execution time of the kmeans algorithm. The test was executed without sampling the dataset because kmeans does not require huge quantity of memory like the hierarchical one. In fact, the kmeans data structure only consists of a set of centroids with their associated attributes. At each step all points are assigned to the closest centroid, whose positions are updated as we explained before. For this reason the convergence of this algorithm is usually faster than the hierarchical and DBSCAN. More in depth, we measured the execution time starting from 10^3 to the entire size of the dataset, increasing the number of samples of 10^3 in each execution step. After that, we approximate the obtained coordinate to a linear curve in order to

see the correspondence with the expected prediction of $O(knT)$. We report in Figures 11,12 the results about same dataset analyzed before in the hierarchical case. In both cases the trend of the measures seems to respect well the expectations, in fact the orange line, that corresponds to the approximation, well fills the blue points. To notice that, also for the measure done with huge quantity of samples ($>10^5$), there is not many differences in terms of time compared to the taken measure with few samples. Moreover, there are some measures that are lower even if the number of samples is smaller.

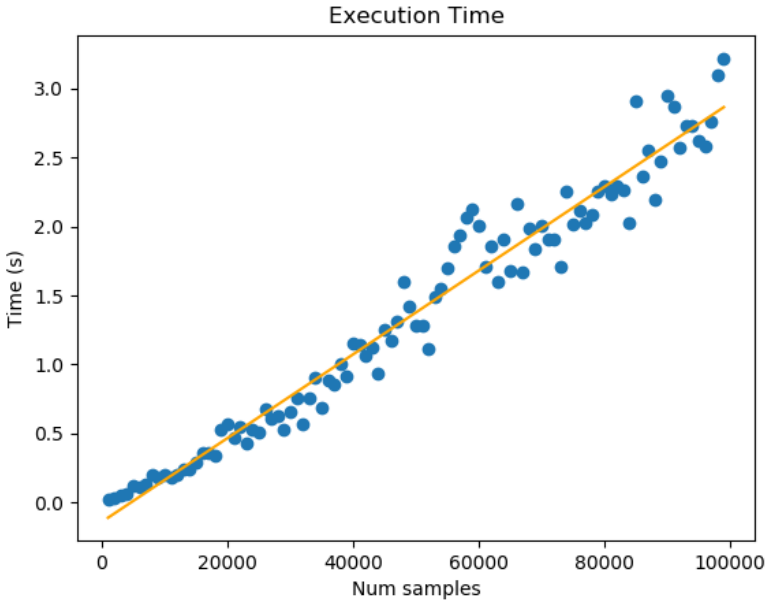


Figure 11: Execution time dataset c1, x axis is the number of tests, y axis is the execution time in seconds.

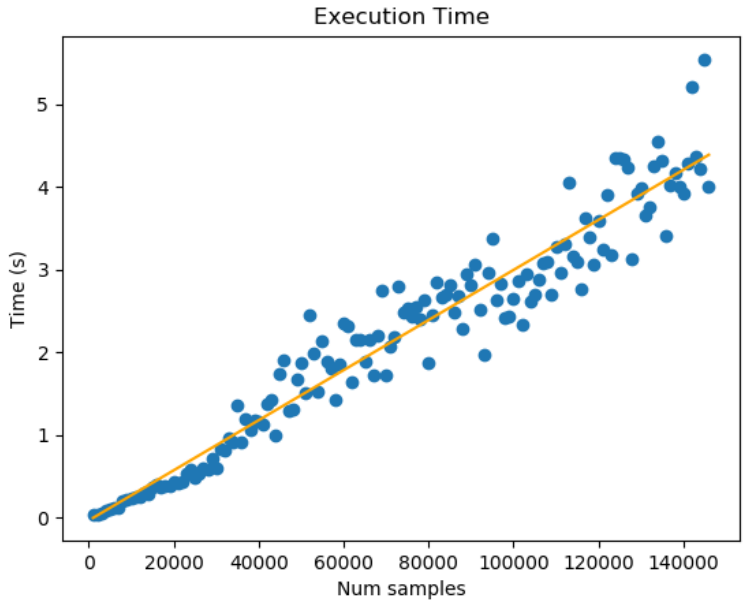


Figure 12: Execution time dataset c3, x axis is the number of tests, y axis is the execution time in seconds

4.5 DBSCAN

This section represents the details for the DBSCAN clustering implementation. A summary is proved in Table 3.

Library	SkLearn
Time complexity Get neighbor	$O(\log n)$
Time complexity with distance matrix	$O(n \cdot \log n)$
Time complexity without distance matrix	$O(n^2)$
Memory complexity	$O(n^2)$

Table 3: DBSCAN clustering properties

Listing 2: In first row the data structure of DBSCAN is generated by setting the value of *eps* and *minsamples*, in the second and third row is fit the data structure created before, in the fourth row are extracted the labels of the generated clusters.

```
1 db = DBSCAN(eps=eps, min_samples=minSamples)
2 X = dsNorm.values
3 db.fit(X)
4 labels = db.labels_
```

4.5.1 Development and tuning

DBSCAN works in a different way from the Hierarchical and Kmeans, in fact its biggest difference is that it does not require as configuration parameter the number of clusters (k) that will be generated. The two parameters used to generate labels are *eps* and *min samples*. During the execution phase DBSCAN visits one point at a time tracing an area with ray of *eps* around the point itself. If, in this area there are other points as the value of *min samples*, a new cluster is generated and the point marked as core and all points in the neighborhood are assigned to it, otherwise the visited point is temporarily labeled as noise. In fact, later in the execution, this point could be part of a new cluster because it could be a neighbor of another core point. Note that, when a new cluster is generated,in addition to the core and neighborhood points, also the neighbors of neighborhood points are added to the cluster, this until all points have been visited. Therefore, to correctly configure DBSCAN it is necessary to run several tests by varying the two input parameters *eps* and *min samples*. In the analysis we set *min samples* with heuristic values of 5 10 and 15 in order to give consistency to the size of the clusters and then we test the value of *eps* and analyzed the results. We used two support procedures for tuning: the KNN and a custom procedure that measures the value of k at varying *eps*. Both will be described later.

4.5.2 Execution Time

For this test we proceeded in a similar way to the hierarchical and kmeans, in fact the number of samples start from 10^3 to $44 * 10^3$, increasing by 10^3 samples each step. The configuration parameters have been configured as $eps = 1$ and $minsamples = 5$, however their influence on the final execution time can be ignored. We found two main behaviors analyzing all the plots. The first case, Figures 13,14, the prediction of $O(nlogn)$ is respected, in fact the orange curve follows well the trend of the blue points. There is only a bit of divergence in the measures of few samples. On the other hand, the second case, Figures 15, 16, does not respect the expectation. It can be observed that for few samples there is a very slow growth that suddenly changes behavior at a certain point between two measures, leaving two distinct points zones.

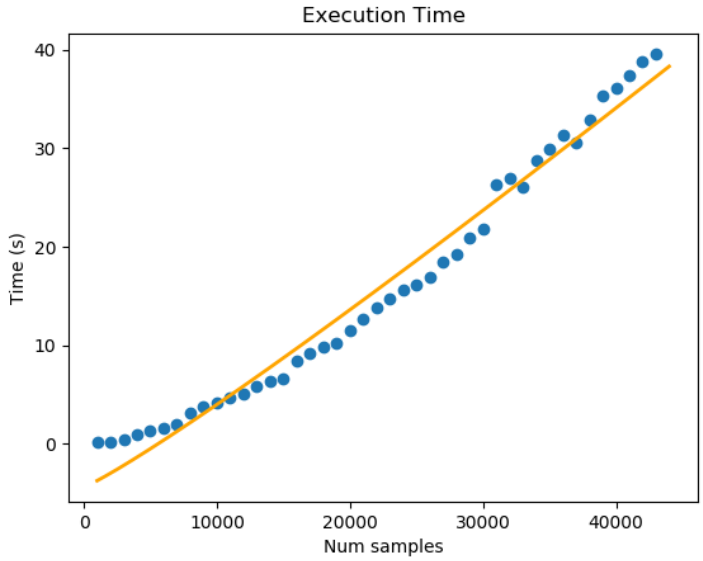


Figure 13: Case 1 dataset d5, x axis is the num of samples, y axis is the execution time. The blue points trend respect the expectations

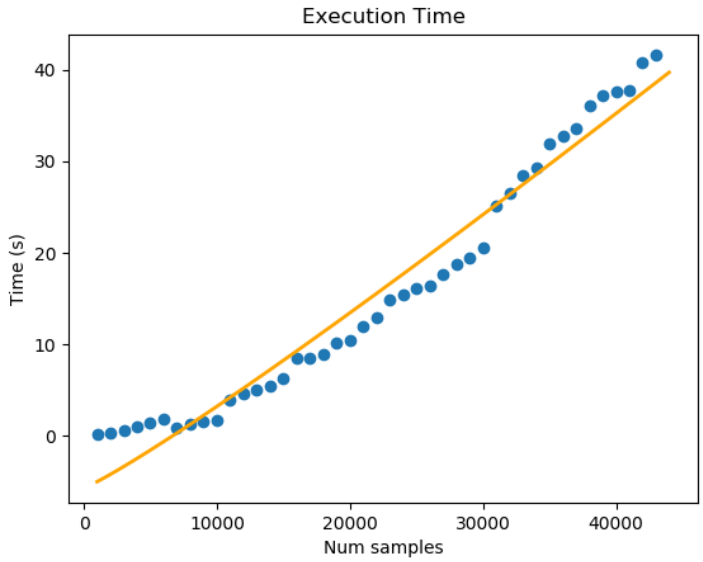


Figure 14: Case 1 dataset d6, x axis is the num of samples, y axis is the execution time. The blue points trend respect the expectations

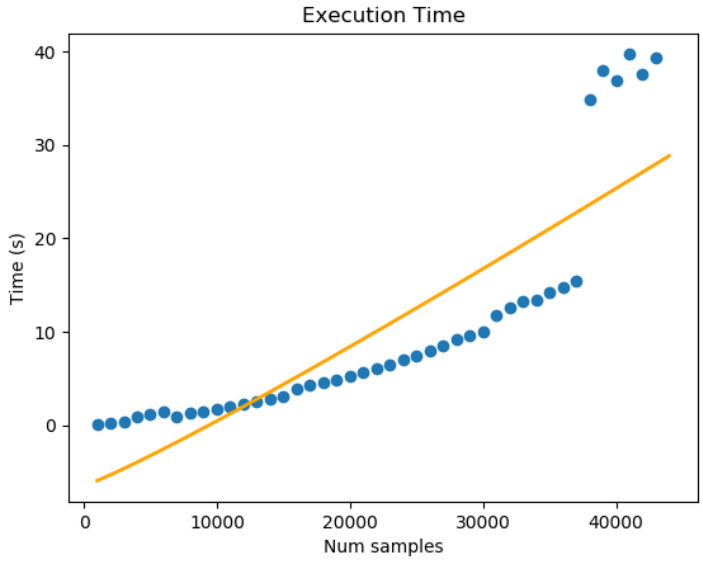


Figure 15: Case 2 dataset c6, x axis is the num of samples, y axis is the execution time. The blue points differ from the prediction

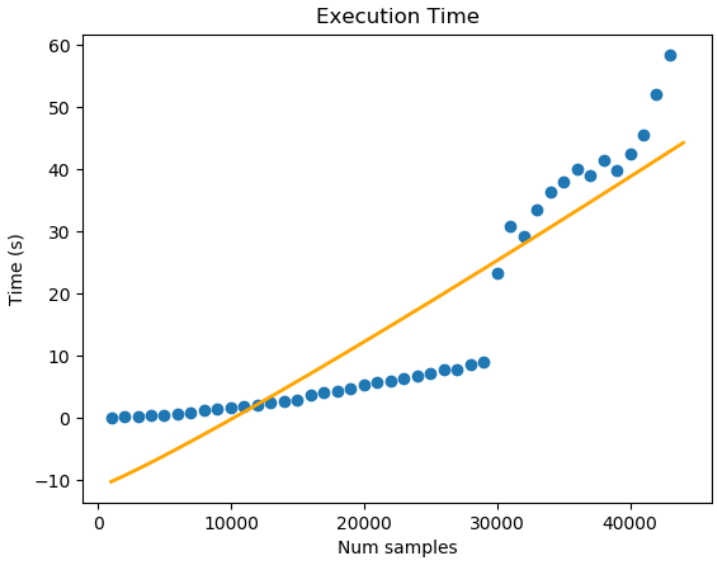


Figure 16: Case 2 dataset d3, x axis is the num of samples, y axis is the execution time. The blue points differ from the prediction

4.5.3 Neighbor Distance

As we said previously, the DBSCAN algorithm creates clusters by using the points distance. For this reason we adopted a method that allow to calculate the points distances and show them as a function, then it is possible to ease the choose of the configuration parameter *eps* that represents the distance used by the algorithm to generate the clusters labels.

Listing 3: In first row are retrieved the values form the dataset, in the second row is fit the data structure of the nearest neighbor by setting the number of neighbors and the 'auto' algorithm, in the third row are retrieved the distances generated in the second row, in the fourth row is extract only the fifth column of the distances matrix in the sixth row the extracted values in row four are sorted in ascending order

```
1 X = dsNorm.values
```

```
2 nbrs = NearestNeighbors(n_neighbors=5, algorithm='auto').fit(X)
3 distances = nbrs.kneighbors(X)
4 distances = distances[:, [minSamples - 1]]
5 distances = np.sort(distances, axis=0)
```

To configure this procedure must be select the value of *min points* first. In the example we will describe here we set *min points* = 5. Then the following steps consist in executing the NearestNeighbors function provided by Sklearn. This function return a matrix structure that contains as rows as points in the dataset. The number of columns depends on the selected values of *min points*. In this example we have 5 columns. In particular the last column represents the distance that there is between a point and the fifth point closest to it. The next step consist in sorting this last column in ascending order. It means that the points are now sorted according to the distance they have from their fifth nearest point in ascending order. Then, the result plot is shown in Figure 17. It is possible to see that the distance value does not grows till reaching $8 * 10^5$ samples, then there is a knee and after that the distance grows very quickly. This means that in the analyzed dataset there are approximately $8 * 10^5$ very close points and as we discuss in Section 2.2 correspond to the small pores. Where the curve grows quickly there is a set of approximately $5 * 10^3$ of very spread points that correspond to the big pores, in fact the reason of this growth is due to the great variance in their attribute values.In order to create cohesive and homogeneous clusters, it is common to select the value of *eps* in the knee part of the plot, because it allows to generate clusters where all the points below the threshold are assigned to a cluster while all the points above are marked as noise.

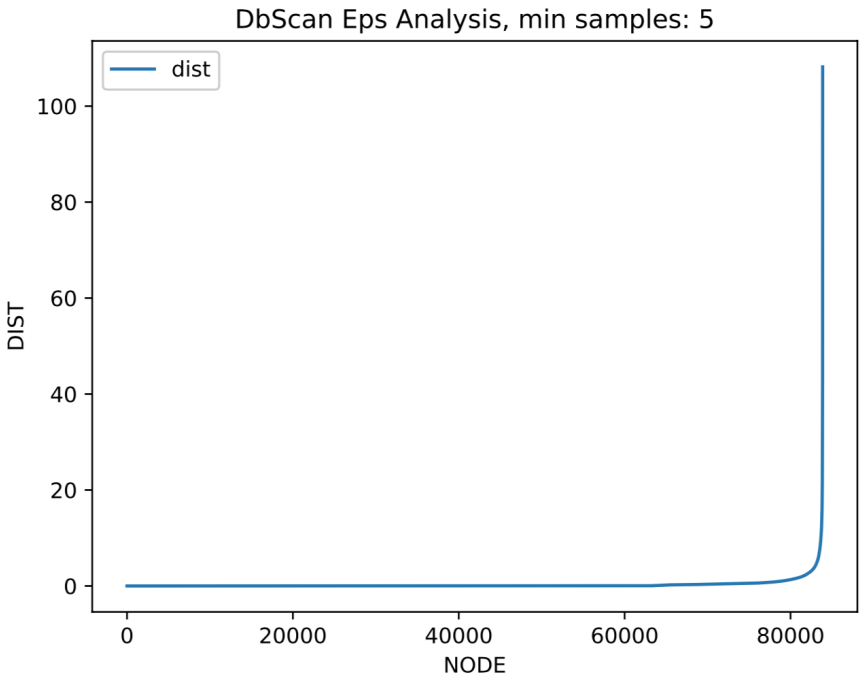


Figure 17: Neighbor distances, x axis is the num of point, y axis is the distance. The blue points differ from the prediction

4.5.4 Eps Analysis

We present here an alternative method to choose the value of *eps* to configure the DBSCAN algorithm. As we discuss the DBSCAN receives as input the parameters *eps* and *min samples*, therefore there is no control on the number of generated clusters. But, in some cases could be useful to ask the algorithm the number of cluster (k) to be generated to compare the result with other cluster algorithms results. For this reason we inspected the results of this method that shows the correlation between the *eps* values and the number of clusters (k). Since it is only possible to know the number of clusters after executing the DBSCAN, this method consist of executing the algorithm by fixing the number of *min samples* and varying *eps* in a range of values, then plotting the result. In Figure 18 there is an example of how this method works. The value of *min samples* was set to 5. Then, in the x axis the value of *eps* and the number

of generated cluster in the y axis. It is possible to choose the number of clusters by watching the y axis and finding where the curve assumes the desired value of L and then read the correspondent value of *eps*. If the algorithm is then configured with this precise value of *min samples* and the selected value of *eps* it returns exactly k clusters. Notice that can exist more than one value of *eps* that gives the desired value of k. For this reason we introduced a more detailed version of this method. In Figure 19 there is an example. The top sub plot corresponds to the previous example, but it was combined with the silhouette and the outliers analysis. More in details, the silhouette gives the information of what could be the best choice of *eps* where the score is high, and can be useful when more value of *eps* correspond to a single value of k, in this case it better to choose the higher value of the silhouette. In the bottom plot is shown the number of outliers that will be generated. It is a good idea to choose the parameters to have this value as low as possible.

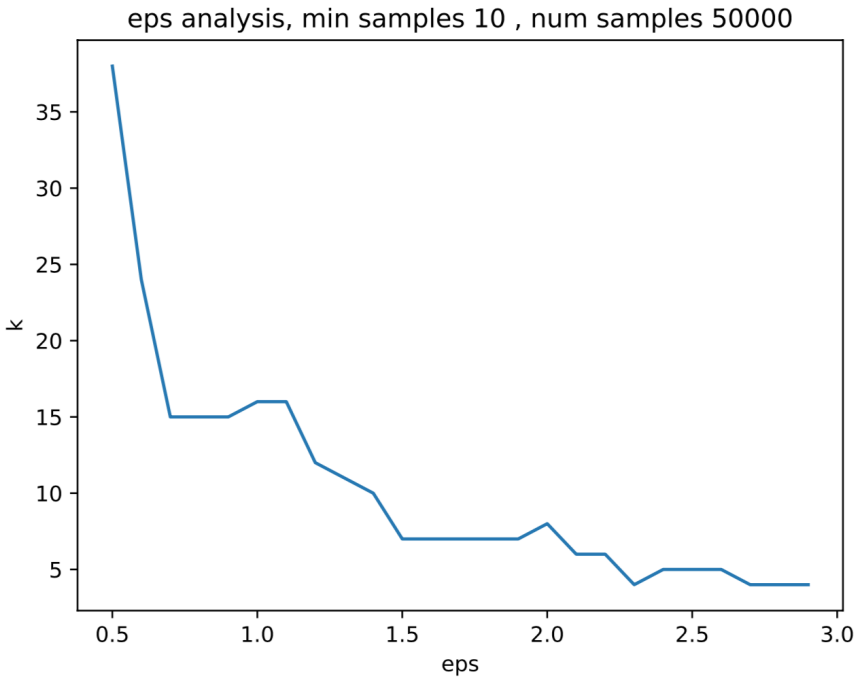


Figure 18: Eps - k correlation details, x axis is the value of eps, y axis is the number of clusters k.

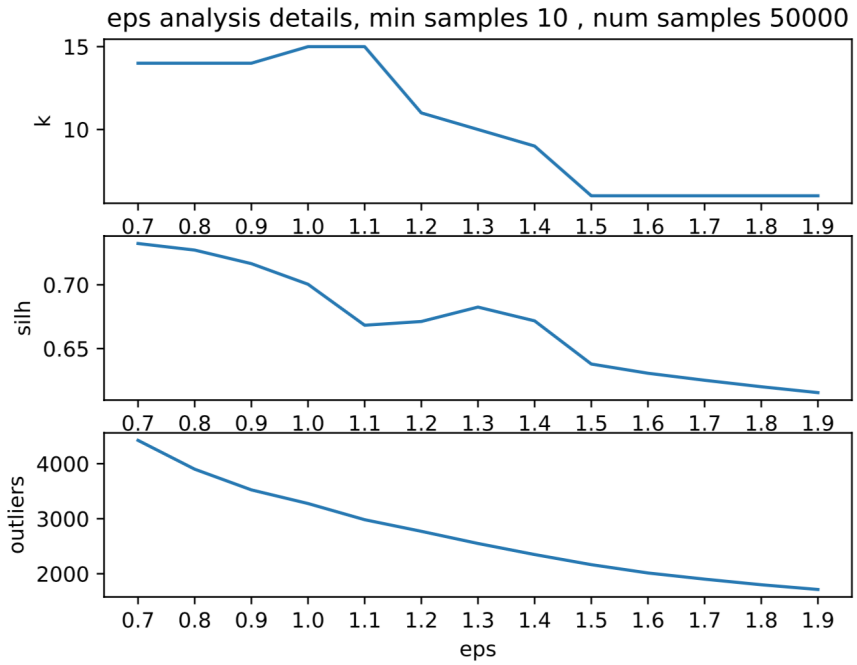


Figure 19: eps - k correlation details, Top plot is fig 18, mid plot is the silhouette, bottom plot is the ouliers analysis

4.6 Hierarchical clustering

This section represents the details for the hierarchical clustering implementation.

A summary is proved in Table 4.

Library	Scipy
Time complexity	$O(n^3)$
Memory complexity	$O(n^2)$

Table 4: hierarchical clustering properties

Listing 4: In first row is created the distances matrix by setting the dataset, and the linkage type. In this case we used the *ward* type, in second row the clusters labels are generated by setting the distance matrix, the number of cluster k, and the criterion (in this case we used ='maxclust')

```
1 Z = linkage(dsNorm, link = 'ward')
2 clusters = fcluster(Z, k, criterion='maxclust')
```

4.6.1 Development and tuning

To implement the Hierarchical clustering we adopted the **Agglomerative** strategy. It is a "bottom up" approach. It means that at the beginning each sample belongs to a different cluster. Later, in each execution step, the clustering algorithm merges the two closest clusters by measuring their distance till, in the last step, only one cluster will remain. All of this merging steps are saved in a data structure that can be called *linkage matrix*. Then, an other function receives as input this matrix and the number of cluster to generate (k). Finally, returns as output the labels associated to the samples in the starting dataset. There are three parameters that tune the algorithm:

(i) **K**. it is the number of clusters to generate

(ii) **Metric**. it is the formula to calculate the distance between two points and can influence the final shape of clusters. In our analysis we used the "Euclidean" metric distance. (See Section 3.4).

(iii) **Linkage**. it measures the distance between two sets of points. The algorithm uses this formula to find what are the closest cluster to merge at a certain moment of execution. In our analysis we used the linkage "ward":

4.6.2 Hardware restriction

With the Hierarchical clustering we had to face with the problem of the memory requirement. In fact, this clustering technique requires $O(n^3)$ in terms of time and

$O(n^2)$ in terms of memory. The last requirement means that this algorithm is difficult to apply on medium and large dataset because requires big quantity of memory. For example, in our real case we used 16Gb of memory ram where part of it is used by the OS and other processes, but to approximate we can still consider to have available all the 16 GB. The formula to calculate the maximum number of samples in a dataset that can be processed within the memory is:

$$M \sim DS^2$$

$$S_{\max} \sim \frac{\sqrt{M_{\max}}}{D}$$

where S is the maximum number of samples in the dataset, M is the available memory (Bytes), and D is the size (in bytes) of the numbers used to store distances in the distance matrix. In our case the result was approximately:

$$S \sim \frac{\sqrt{16 * 10^9}}{8} \sim 44721$$

Where the value of the kind of data is 8 because we used *doubles*.

Starting from this consideration the number of samples we can process with our hardware is ~ 44721, but it is possible to process a slightly larger quantity of them. In fact, if the memory runs out, the operating system compensates using the swap space. The datasets analyzed contains many more samples than the quantity obtained above so we bypass this issue by sampling the dataset and we choose $5 * 10^4$ points for the dataset sampling size. Notice that all the problem involved with the sampling technique are discuss in an other section. We found a good use of the hierarchical if combined with a smart filter. In our case the points filtered with a Diameter max ≥ 25 in each dataset are not more than 10^4 so the can be clustered with the hierarchical method without run out of memory. In theory is possible to cluster more points if the algorithm executed on a machine with more memory ram, but must keep in mind that

the execution time grows exponentially as we see later.

4.6.3 Execution Time

We measured the execution time of the hierarchical clustering on our datasets. In this test we set the sampling size starting from the previous consideration about the number of samples. In fact, we reduced the maximum number of samples to $44 * 10^3$ because is near the maximum value our hardware can process. More in depth, we measured the execution time starting from 10^3 to $44 * 10^3$, increasing the number of samples of 10^3 in each execution step. The result consist of 44 measures for each dataset. After that, we approximate the obtained coordinate to a 3-degree polynomial curve in order to see the correspondence with the expected prediction of $O(n^3)$. In Figures 20,21 we selected two of the most significative results we obtain in all the datasets. In Figure 21 there is an example of an expected trend because the orange line, that corresponds to the positive part of the 3-degree polynomial curve, strictly follow the blue points. On the other hand we found in several datasets a different behavior shown in Figure 20. It is possible to see that the time growth is more similar to a linear curve than the expected cubic curve. In fact, the orange line doesn't not fit well the trend of all the points. To notice that the last blue points in the right of each plot are very far from their expected position. The reason is that even if $44 * 10^3$ samples do not completely fill 16 GB of Ram, in the real case this memory is also used by other processes and then it saturated before the expectation. When the ram is not sufficient for all the processes that the CPU is executing therefore, according to the rules of the scheduler of the OS, some of them are moved temporally out from the ram to the disk. It is possible to improve performance by increasing the priority of the python process that performs the analysis but remaining however in a bottleneck status.

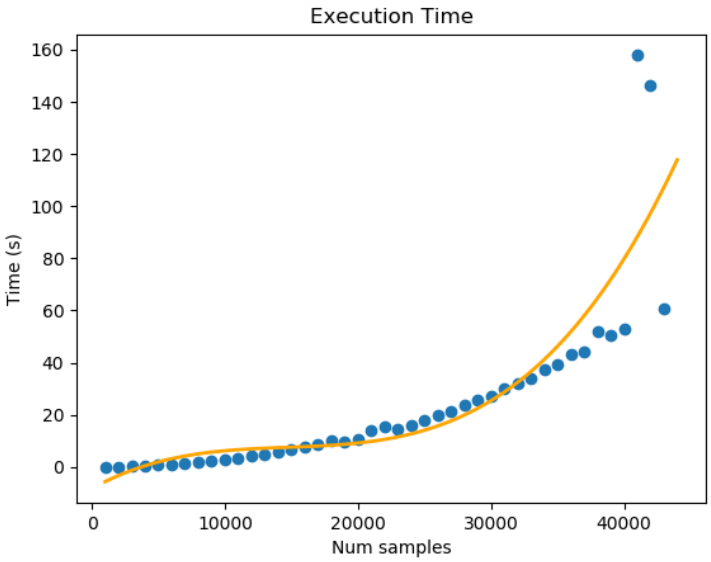


Figure 20: Execution time plot for dataset c1, x axis is the num of samples,y axis is the execution time

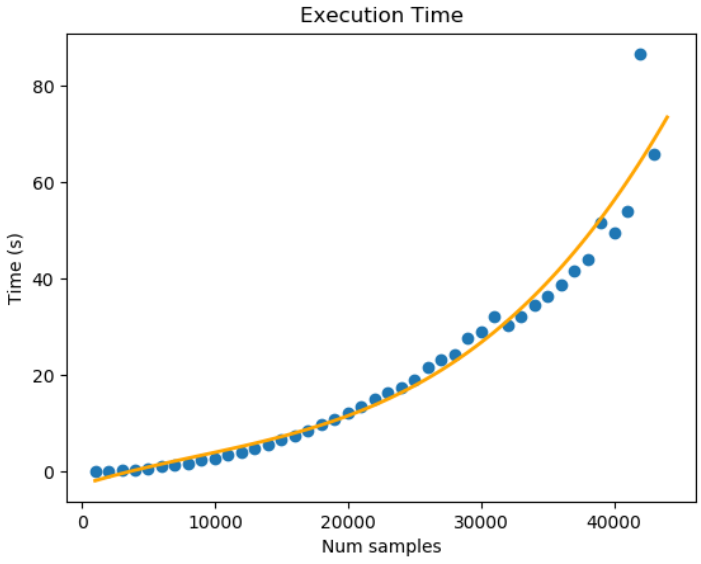


Figure 21: Execution time plot for dataset c3, x axis is the num of samples,y axis is the execution time

4.6.4 Linkage types

Single

$$d(u, v) = min(dist(u[i], v[j])$$

Where u, v are two clusters and u_i, v_i are two elements in the respective clusters. It is the distance between the closest pair of points of two different clusters. The algorithm merges the two clusters where this distance lower.

Complete

$$d(u, v) = max(dist(u[i], v[j])$$

Where u, v are two clusters and u_i, v_i are two elements in the respective clusters. It is the distance between the farthest pair of points of two different clusters. The algorithm merges the two clusters where this distance lower.

Average or **UPGMA**

$$d(u, v) = \sum_{ij} \frac{d(u[i], v[j])}{(|u| * |v|)}$$

Where u, v are two clusters and u_i, v_i are two elements in the respective clusters. The distance is the mean of the distances between any pair of points of two different clusters. The algorithm merges the two clusters where this distance lower.

Weighted or **WPGMA**

$$d(u, v) = (dist(s, v) + dist(t, v))/2$$

Where s, t are the two previous merged clusters in new cluster u and v is an other cluster. The distance is the mean of the distances between the previous merged cluster s, t and the other cluster v. The algorithm merges the two clusters where this distance

lower.

Centroid or **UPGMC**

$$d(s,t) = ||c_s - c_t||_2$$

Where s, t are two clusters and c_s, c_t are the two controids of the respective clusters. The distance is the Euclidean difference between the centroids of two different clusters. The algorithm merges the two clusters where this distance lower.

Median or **WPGMC**

$$d(s,t) = ||w_s - w_t||_2$$

Where s, t are two clusters and w_s, w_t are the two controids of the respective clusters. It is similar to the centroid method but this one creates a new centroid on the median of the two old centroids. The algorithm merges the two clusters where this distance lower.

Ward

$$d(u,v) = \sqrt{\frac{|v|+|s|}{T}d(v,s)^2 + \frac{|v|+|t|}{T}d(v,t)^2 - \frac{|v|}{T}d(s,t)^2}$$

Where s, t are the two previous merged clusters in new cluster u and v is an other cluster. The algorithm merges the two clusters where the variance is minimum.

4.6.5 Linkage selection

The choice of the linkage type in the hierarchical clustering influences the clusters shapes. Therefore we combine the hierarchical clustering linkages with the silhouette procedure. This could be useful indication about how good is a link type for the current dataset. The method consists of sampling the dataset, due to the hardware restriction of the hierarchical algorithm, then execute the clustering for each linkage type. Finally

use the labels obtained as input for the silhouette computation. In this analysis we tested two different cases: the first case we used sampled dataset of 10^5 points, in the second case we set the filter *Diameter Max >25* that usually return datasets of 10^5 point max. Their characteristics are shown in Table 5.

	Case 1	Case 2	Case 3
Number of tests	4	10	10
Sampling	10^5	10^5	Diameter Max > 25
Number of clusters	2≤K ≤6	K = 4	K = 4
Linkages	single, complete, average, weighted, centroid, median, ward		
Plot	Silhouette score		

Table 5: Executed tests on hierarchical linkage types

With the exception of the ward linkage, all the other linkage types have similar values in all the tests performed. So in this case the silhouette is not influenced by the variation of k, or by sampling and also returns similar results even on multiple datasets. We report the plot of two datasets:

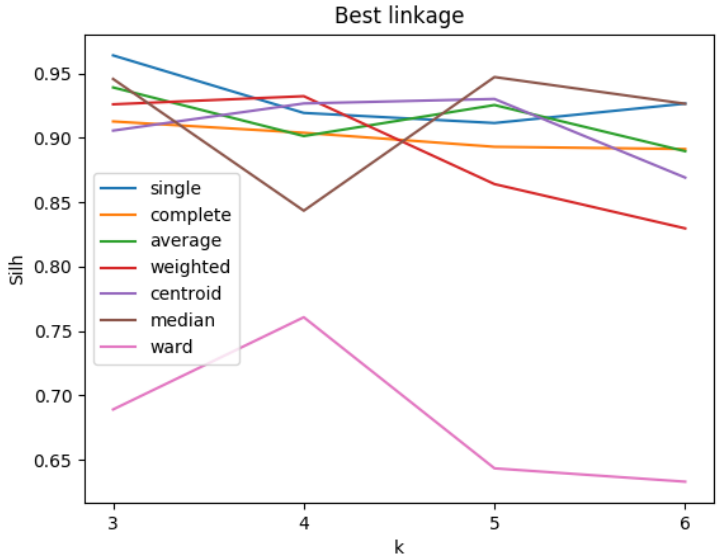


Figure 22: Case 1: dataset c1, Silhouette, x axis is the number of clusters (k), y axis is the silhouette score

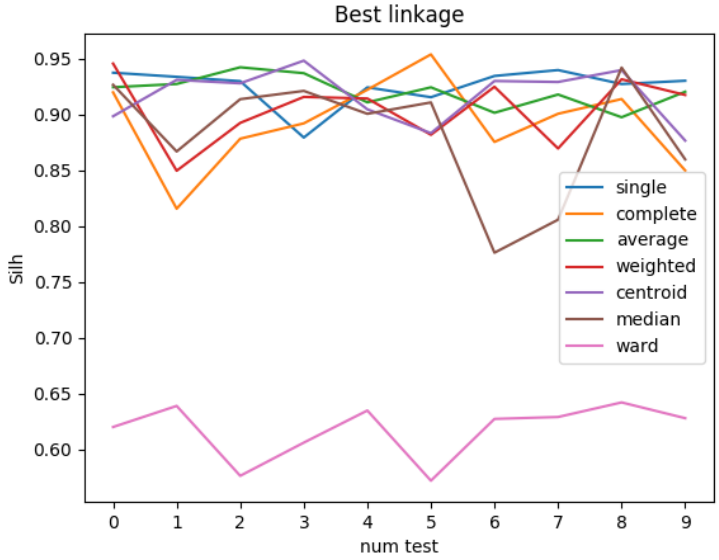


Figure 23: Case2: dataset c1, Silhouette, x axis is the num of test, y axis is the silhouette score

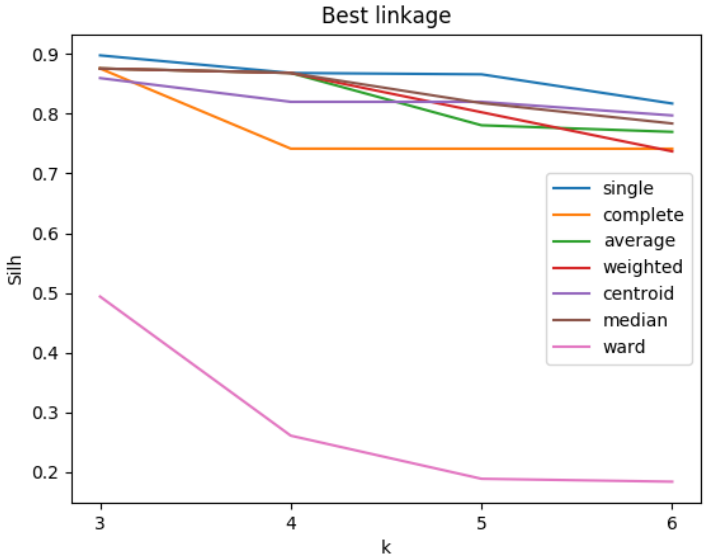


Figure 24: Case 3: dataset c3, Silhouette, filtered DiamterMax >25, x axis is is the number of clusters (k), y axis is the silhouette score

In all the executed tests the metric "ward" assumed the lower values while all the other metrics have similar values. From this results it is difficult to establish if there is a best linkage type at all because no one assumes ever the best values. In this case, the best approach is to select the linkage according to the value of k or the sampling type used to configure the hierarchical algorithm. To notice that in the hierarchical experiments was used the linkage *ward* that achieved good result confirmed by domain experts. Then, could be an interesting investigation trying the other linkages that obtain a better score in this analysis and see if they will also obtain better result in real test cases.

4.6.6 Silhouette with Dendrogram

We introduce here a custom method that we developed to analyze the silhouette scores with the hierarchical clustering. To understand the content of this analysis refer to the silhouette description above where are deeply discussed the meaning of the score, in particular what are the intra ed extra cluster values (see Section 3.6.1).

A **dendrogram** is a branches-tree plot usually used with the hierarchical clustering. In fact it graphically describes the merges that the hierarchical does at the various steps of its execution. As discuss in previous sections the hierarchical clustering start its execution having each point in a separate cluster till the final step where all the point are merged in a single cluster. In the plot configuration must be set the number of cluster to be shown. These clusters will be placed on the leaves of the plot and correspond to the execution step that has exactly the desired number of clusters that have not been merged yet.

This method combine together the dendrogram with the $inter, intra$ silhouette values. In particular, the aim of the method is to show on the branches of the dendrogram the values of the silhouette for the cluster identified by the branch itself. The dendrogram can be considered as a binary tree where each leaf is a cluster. From the point of view of the implementation, this tree it can be visited using a recursive function, thus allowing access to all clusters and how they are connected through branches. This is the basic idea behind the implementation of the method. In our case it was necessary to adapt the data structures in order that could be used following the reverse path. In fact, our recursive method starts from the root node and runs through the tree until it reaches the leaves that represent the end clusters. When the recursion reaches the terminal leaves it calculates the silhouette immediately, consequently, when an intermediate node is reached, which has sub leaves, the silhouette is calculated only after it has been calculated on all of the sub leaves. This path continues until the root of the tree is reached. In Figure 25 an example of the plot described above. We clustered the dataset c1 through the hierarchical algorithm with linkage *ward* and

metric *euclidean*. In each branch there are the information of the *inter*, *intra* values of the silhouette. Greater is the value of *inter* to the respect of the *intra* and more the referenced cluster is cohesive. In this case we have the clusters 0 and 1 that are very dense because the *inter* value is higher than the *intra* value, and the *intra* value is globally low to the respect of the other *intra* values in other clusters. As consequence the cluster 2 is the most sparse because the *intra* value is high.

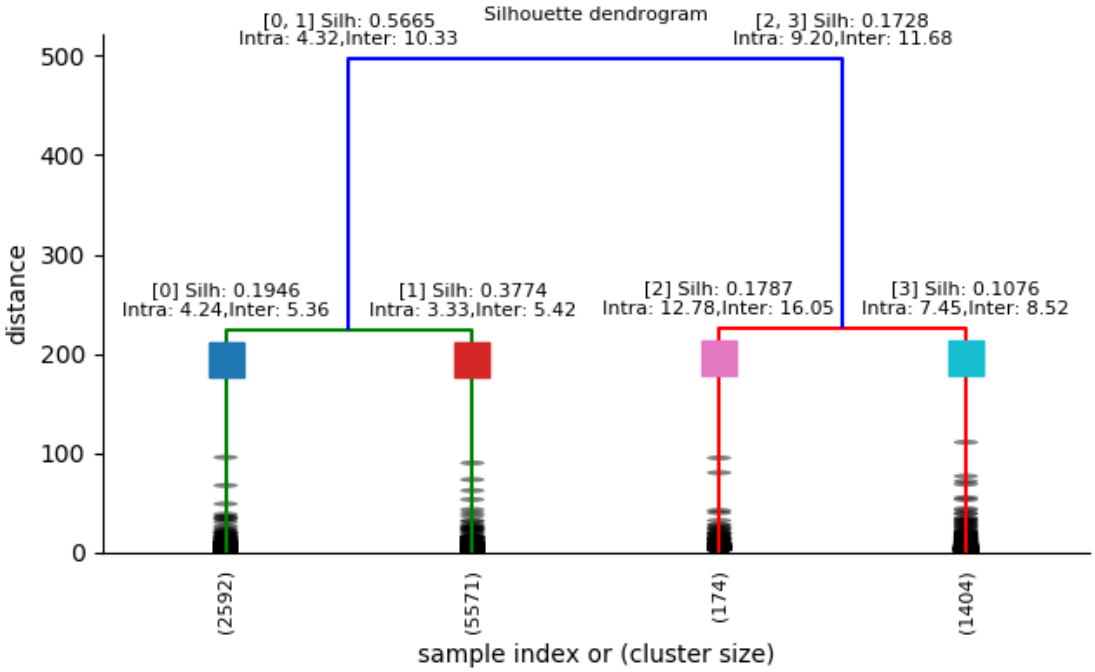


Figure 25: Silhouette with Dendrogram dataset c1 and $K = 4$, the leaves represents the generated clusters. In each branch there are the information of the inter, intra values of the silhouette. Greater is the value of *inter* to the respect of the *intra* and more the referenced cluster is cohesive

4.7 Cluster Comparison

We present here a method that can be useful to compare the labels of two different clustering result. It is based on the *Jaccard* similarity index (Section 3.4) that represents how many two clusters are similar. Figure 26 is an example of this analysis. In this example we run the filter *Diameter Max > 25* on the dataset c1 and clustered it with two different configurations:
(i) Hierarchical, k = 4, linkage = "ward".
(ii) Kmeans, k = 4, seed = 346346.

The analysis is divided in three subplots:
The bottom right is the histogram of the hierarchical labels that shows the size of the generated clusters, the top left is the histogram of the kmeans labels that shows the size of the generated clusters. In the bottom left position, there is the heat map that shows the Jaccard value between the aligned clusters bars. The heat map values are in a range of $[0, 100]$. The value 100 means that the two compared clusters are identical. On the other hands the value 0 means that in the two compared clusters there aren't common points. In this example, the higher value is between the hierarchical cluster 1 and the kmeans cluster 3. This is a usual situation because the higher values is in the cross of the two biggest compared clusters. In fact, the probability that there are common points is higher if the size of clusters is bigger.

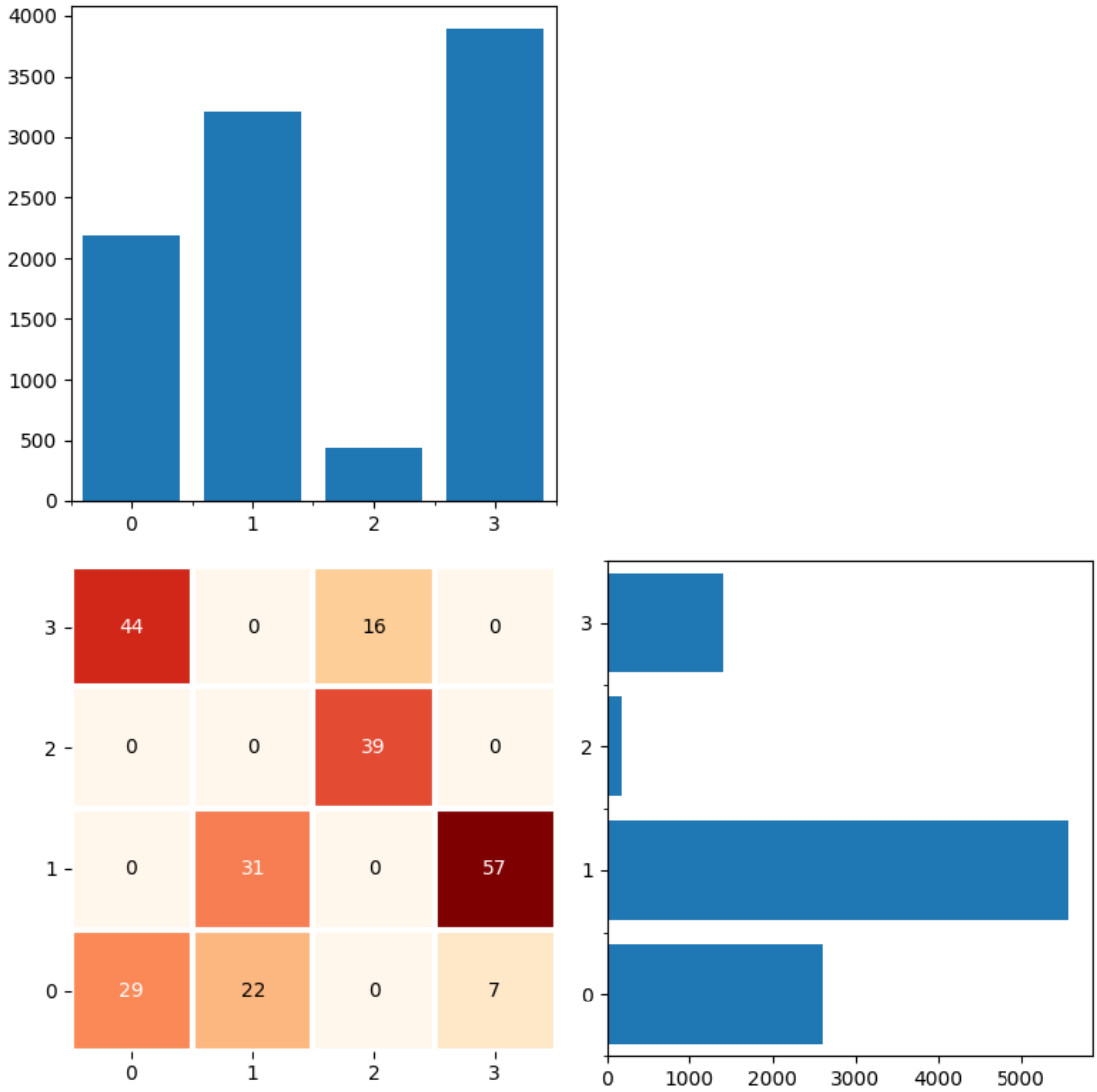


Figure 26: Cluster compare plot dataset c1, bottom right is the histogram of the Hierarchical clustering labels, top left is the histogram of the Kmeans clustering labels, bottom left is the heat map that shows the Jaccard values between the aligned clusters

5 Representative experimental results

In the following, we report some representative results obtained by using our framework with different clustering algorithms and we describe how the results can be analyzed by using a set of charts generated by our system.

5.1 Kmeans Clustering

As described in Section 3.3 Kmeans identifies each cluster with a centroid (center of gravity). Centroids are computed along the different iterations of the algorithm. The number of clusters k is specified as input parameter. Kmeans can work with a very large amount of input data and for this reason is suitable for our purposes where the datasets can reach millions of elements.

In the following analysis we apply Kmeans to each of the available datasets. We vary the value of k and inspect the obtained results with the cluster description techniques presented in Section 3.6. The analyzed datasets are: *a1, a2, a3, b1, b2, b3*. For these experiments we applied the feature selection based on attribute correlations (Section 3.2), since this group of datasets are characterized by many (i.e. 46) attributes.

5.1.1 Silhouette analysis

In this section we compute the value of the global silhouette (see Section 3.6.1) in order to inspect the quality of the clusters for different values of parameter k. Fore this example the silhouette score is computed on 10000 samples over the 86000 samples of the entire dataset (*a1*). This means that a sample of the 12% is analyzed. However the Kmeans algorithm is computed on all the samples. Sampling is necessary for silhouette computation because this measure has a quadratic complexity with the number of points. Hence, this procedure requires huge hardware resources.

Figure 27 shows global silhouette score for different values of k between 2 and 29. We selected two possible two good values at $k = 6$ and $k = 22$. These values are

located in regions of the curve where the silhouette is approximately constant. These regions could represent stable values of the silhouette when varying k. Picking the local maximum in these regions allows selecting values of k which are potentially less sensible to noisy data.

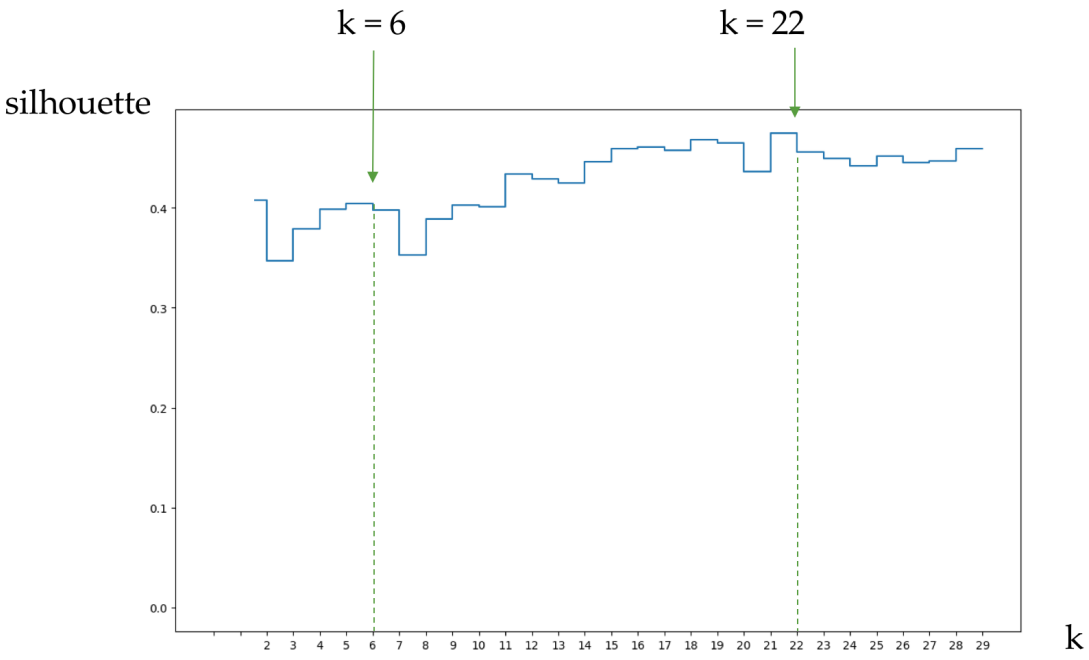


Figure 27: Silhouette a1, local max in $k = 6, k = 22$

5.1.2 Centroid description

In this section we explain how to analyze the positions of the centroids generated by the Kmeans algorithm. The centroid position allows understanding the average values of each attribute in a specific cluster.

Figure 28 shows the centroids for $k = 22$. The horizontal axis enumerates the identifiers of each cluster (from 0 to $k - 1$). For each centroid the chart shows a histogram with the values of the different attributes. The values are normalized with z-score in order to improve visualization. From the chart it is possible to detect clusters

with very high values on a particular attribute. For example there are two well-defined clusters which have bigger values for the size attributes (e.g. the first two clusters from the left of Figure 28). From this kind of descriptions it is also possible to inspect clusters which present similar characteristics, such as the first two clusters from the right of Figure 28).

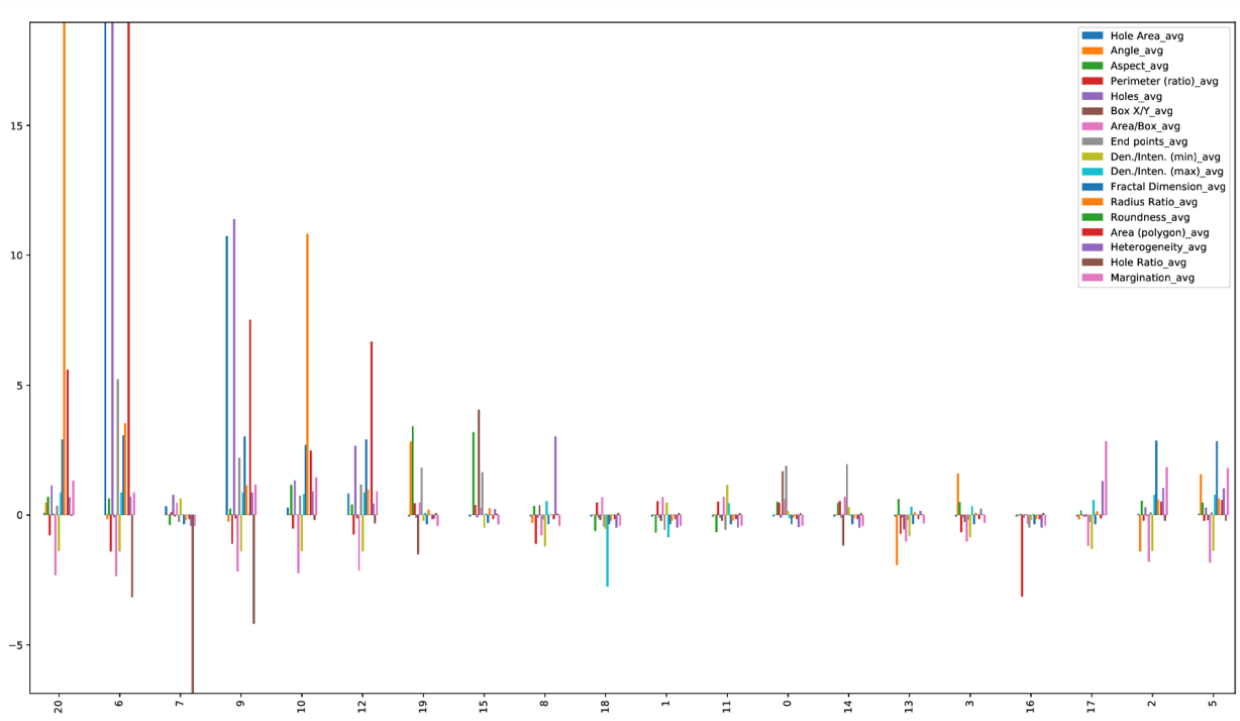


Figure 28: Centroids *a1*, K = 22, vertical bars represent the mean values for each attribute in the cluster

The similarity between cluster centroids can also be inspected by applying a hierarchical clustering on their values. This analysis considers each cluster centroid as a sample and uses the centroid attributes to compute the distance measures. The result is a dendrogram plot like the one presented in Figure 29. This chart shows the hierarchy generated among the cluster centroids. The centroids that are merged in the first (lower) levels of the hierarchy present very similar characteristics. Clusters

that are merged in the very last stages (such as the first three clusters from the left of Figure 29: cluster 4 and 21, 20) present very different attribute values from the other centroids. Notice that the height of each merge stage in the dendrogram represents the distance between the two clusters being merged. For example the blue cluster represents a possible outlier since it is merged at very high values on the vertical axis of the plot.

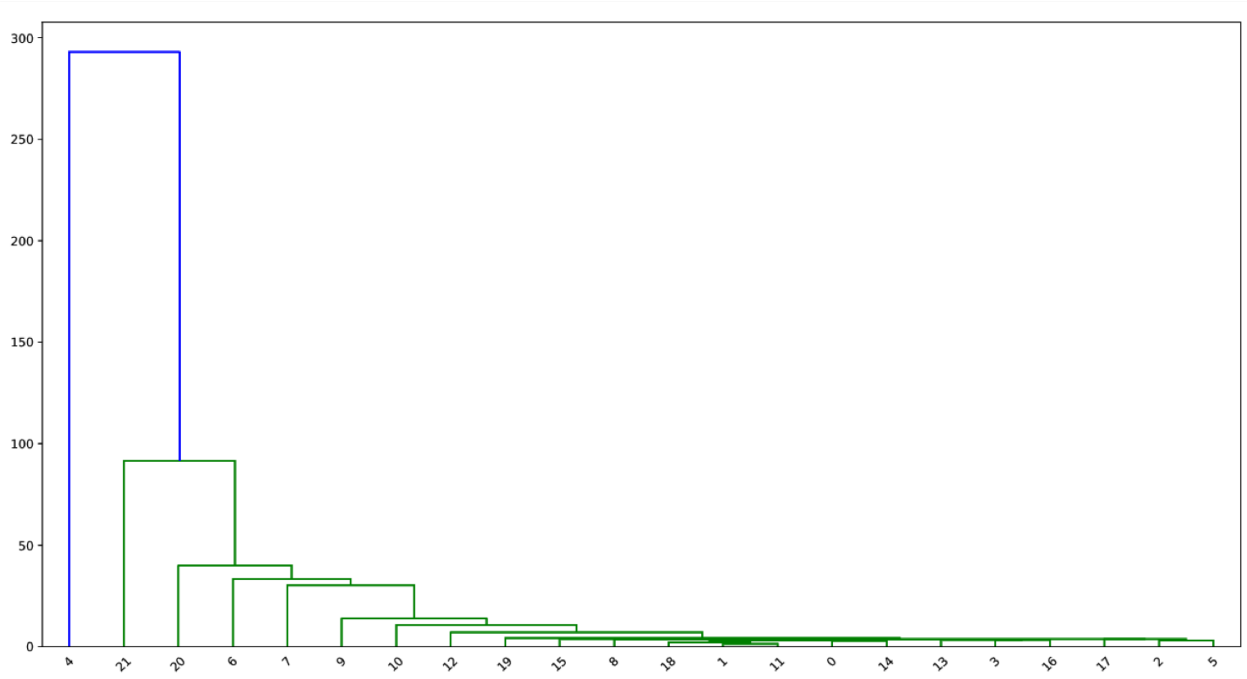


Figure 29: Dendrogram Centroids a1, K = 22

5.1.3 Decision tree description

In this section we use decision trees to describe the cluster produced by Kmeans. These representations highlight the most relevant attributes which better describe the shape of each cluster. Attributes at levels next to the root have higher discriminative power on the input data.

For these experiments we generated decision trees for the Kmeans results with k varying from 3 to 22. Figure 30 depicts an example of decision tree with $k = 6$ on dataset *Falak12210*. In this example, the most relevant attribute is Aspect, as it appears in the root.

The purple leaf shows that 12919 of the 13658 elements in cluster 5 are characterized by $Aspect > 1.005, AreaBox > 0.975$. The Gini index for this leaf is very small (0.037) and for this reason cluster 5 is well described by this decision tree. Also cluster 0 is well represented by the two orange leaves. The $40K$ samples of cluster 1 are grouped together in the leftmost green leaf. Finally the rightmost green leaf contains a mixture of all the remaining points, which are difficult to describe with shallow levels in the decision tree. For these points deeper decision tree would be required. However, notice that deep decision tree are too specific on the input data and do not generalize well the cluster shape.

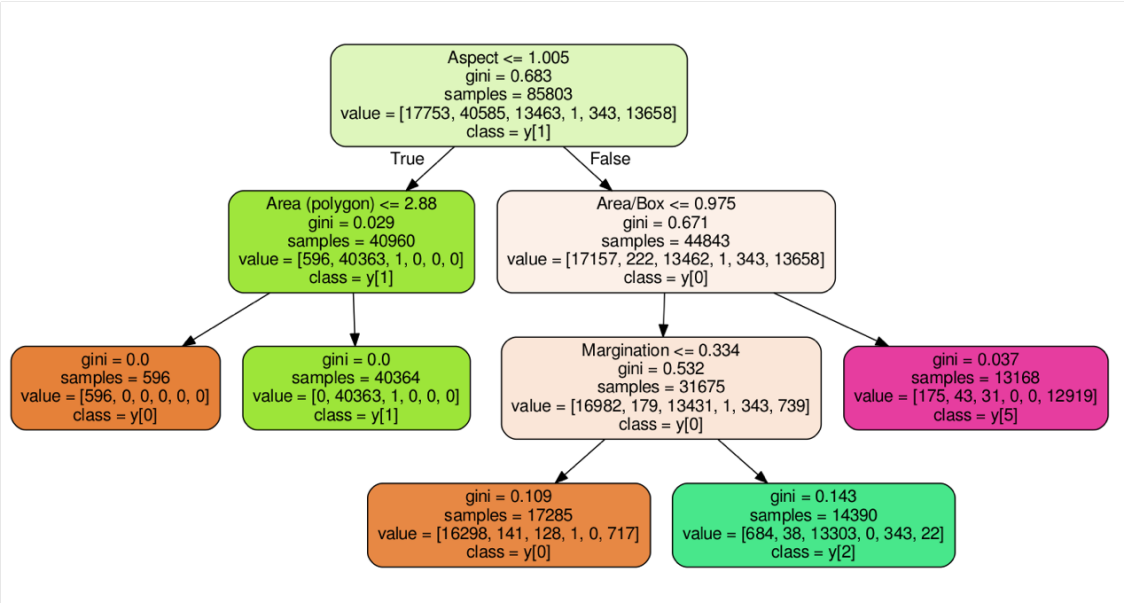


Figure 30: Centroids a1, K = 6

Table 6 represents the importance of the attributes by analyzing the decision trees

generated for k varying from 3 to 22. This result is built by counting the number of times that an attribute appears in the first three levels of all the trees built with the different values of k. Higher number of occurrences represent attributes which are more important, as they appear many times next to the root of the trees.

Table 6: Relevant attributes

Attribute	Occurrences in the first three levels
Aspect	25
Den./Inten. (min)	16
Fractal Dim	14
Box X/Y	10
Area/Box	6
Margination	6
End points	4

5.2 DBSCAN Clustering

DBSCAN is a density based clustering algorithm (see Section 3.5.3 for details). Density is computed by inspecting the distance of each point from all the others. This operation has quadratic computational complexity and requires much more hardware resources and time with respect to Kmeans. For this reason this clustering technique is applied after sampling each dataset. Here we use random sampling and fix the number of selected pores to $50,000$.

The analyzed datasets are: *c2* (83,875 data points), *c6* (154,074 data points).

5.2.1 Choice of epsilon

As described in Section 3.5.3 *epsilon* represents the radius of the neighborhood of a sample. The *minpoints* parameter is instead the minimum number of neighbors to avoid marking a particular sample as outlier. To choose the values of epsilon (*eps*) we fixed $minpoints = \{5; 10\}$. For brevity this report discusses only the results for $minpoints = 5$.

Figure 31 and Figure 32 show different metrics evaluated by varying *eps* on the horizontal axis. Generating these kind of charts is very time consuming, since the DBSCAN algorithm is run many times.

For example the left section of Figure 31 shows the number of clusters k obtained while varying *eps* from 0.5 to 3. This curve can be used to select significant values of *eps* (e.g. $eps = \{1.0; 1.2; 1.4\}$) which produce a specific number of clusters. The three charts on the right of Figure 31 represent the number of clusters, the silhouette and the number of outliers by varying *eps* in a smaller range of values (highlighted in green on the left side of Figure 31). For example with $eps = 1.0$ we obtain 19 clusters, a (good) silhouette near 0.7 and about 3000 outliers. It can be also noticed that the number of clusters, the silhouette and the number of outliers decreases when *eps* increases. This is expected from the behavior of DBSCAN because higher values of *eps* mean that lower densities are tolerated while merging points into clusters.

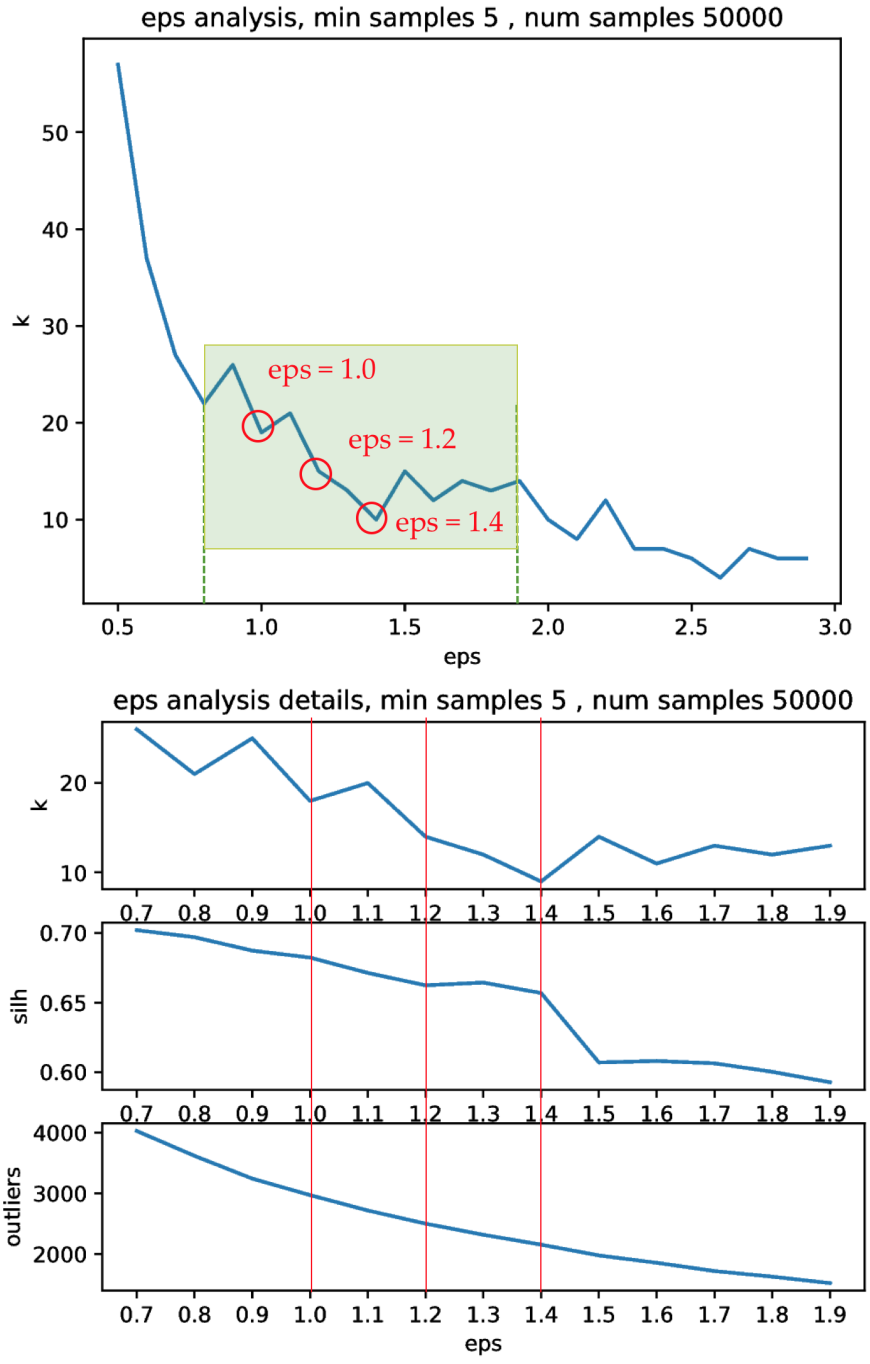


Figure 31: Dataset: a1, Choice of Eps, minpoints 5

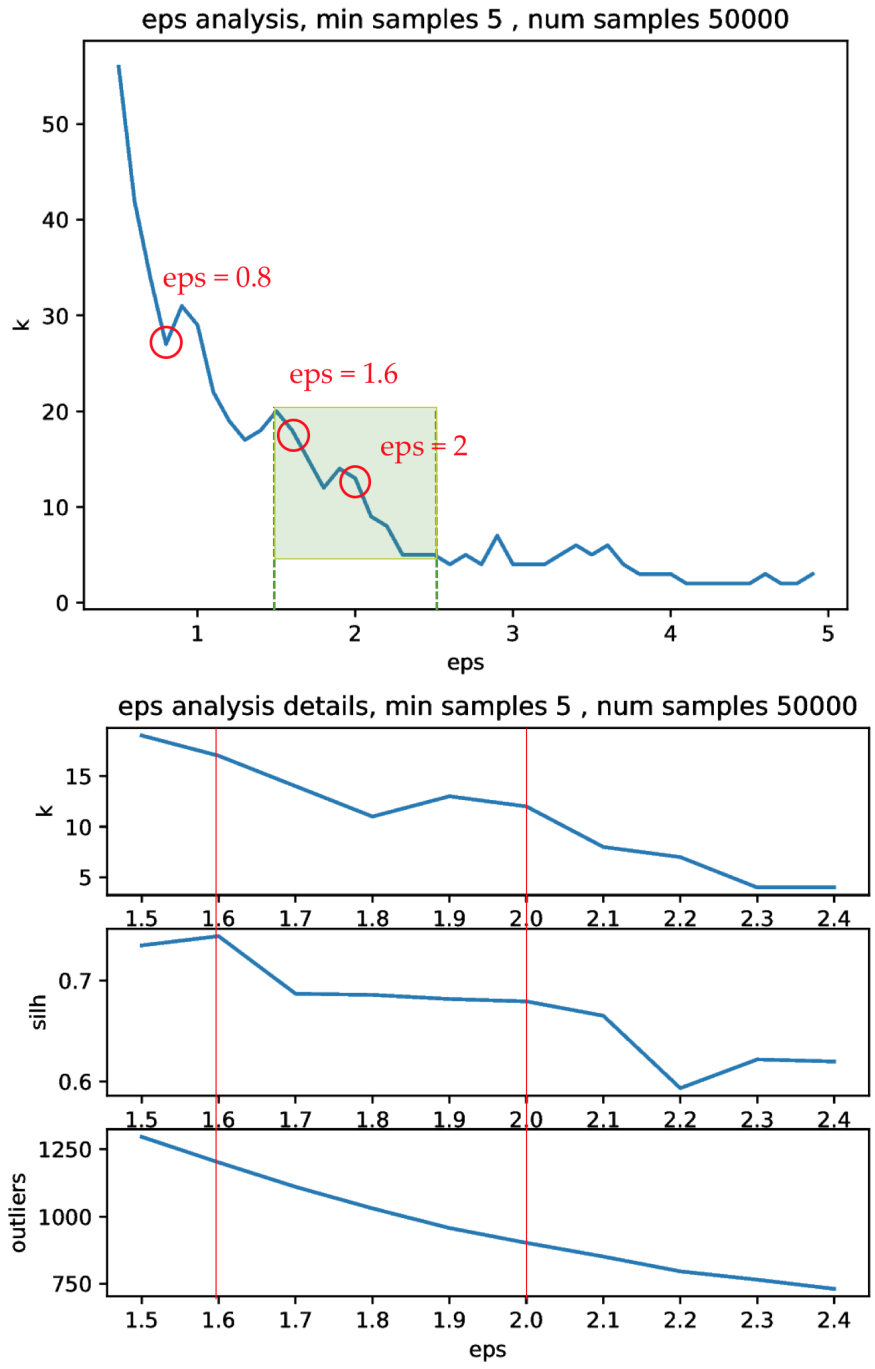


Figure 32: Dataset: a2, Choice of Eps, minpoints 5

Fig. 31, Dataset *c2*: $minpoints = 5$. The selected values are $eps = 1.0, 1.2, 1.4$.

Fig. 32, Dataset *c6*: $minpoints = 5$. The selected values are $eps = 0.8, 1.6, 2.0$.

5.2.2 Cluster size histograms

These paragraphs show some examples of clustering results by selecting the most significant values of *eps* highlighted in the previous section. In particular we show histogram charts which specify the size of the clusters obtained in each clustering configuration.

The histograms in Figure 33 and Figure 34 are built by clustering pores in dataset *c2* and *c6* respectively. We inspect three different values of *eps* for each dataset, with $minpoints = 5$. From these charts it can be noticed again that when increasing *eps* the number of clusters decreases. In each histogram the red bin (cluster -1) represents the set of pores labeled as outliers.

Clustering in Figure 33 presents three main big clusters (i.e. $0, 1, 2$) for all the three *eps* configurations. The remaining clusters have less samples and decrease in number when *eps* grows (they are merged either with noise or with the other main clusters).

Clustering in Figure 34 presents three main big clusters (i.e. $1, 2, 3$) for the first two values of *eps* (i.e. $0.8, 1.6$). The third histogram, built with $eps = 2$ has only two main clusters (i.e. $1, 2$). Cluster 3 has approximately the same size of cluster 3 in the previous two histograms. Cluster 2 is instead bigger than cluster 2 in the first two histograms. Hence, it is likely that cluster 1 and 2 obtained with $eps = 0.8, 1.6$ were merged into a single cluster (i.e. cluster 2) when $eps = 2$.

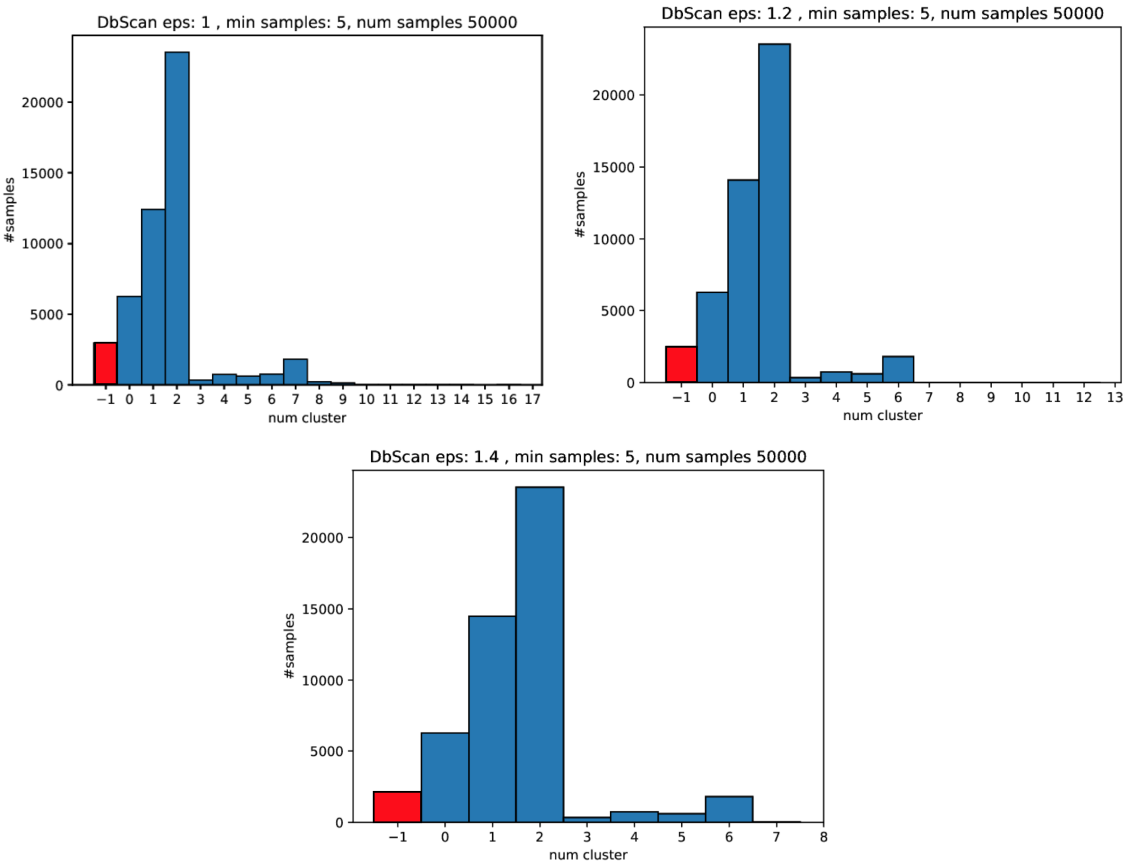


Figure 33: Cluster size, dataset *c2*, $eps = \{1.0; 1.2; 1.4\}$, $minpoints(alias\ min\ samples) = 5$. The red bin (cluster -1) contains outliers.

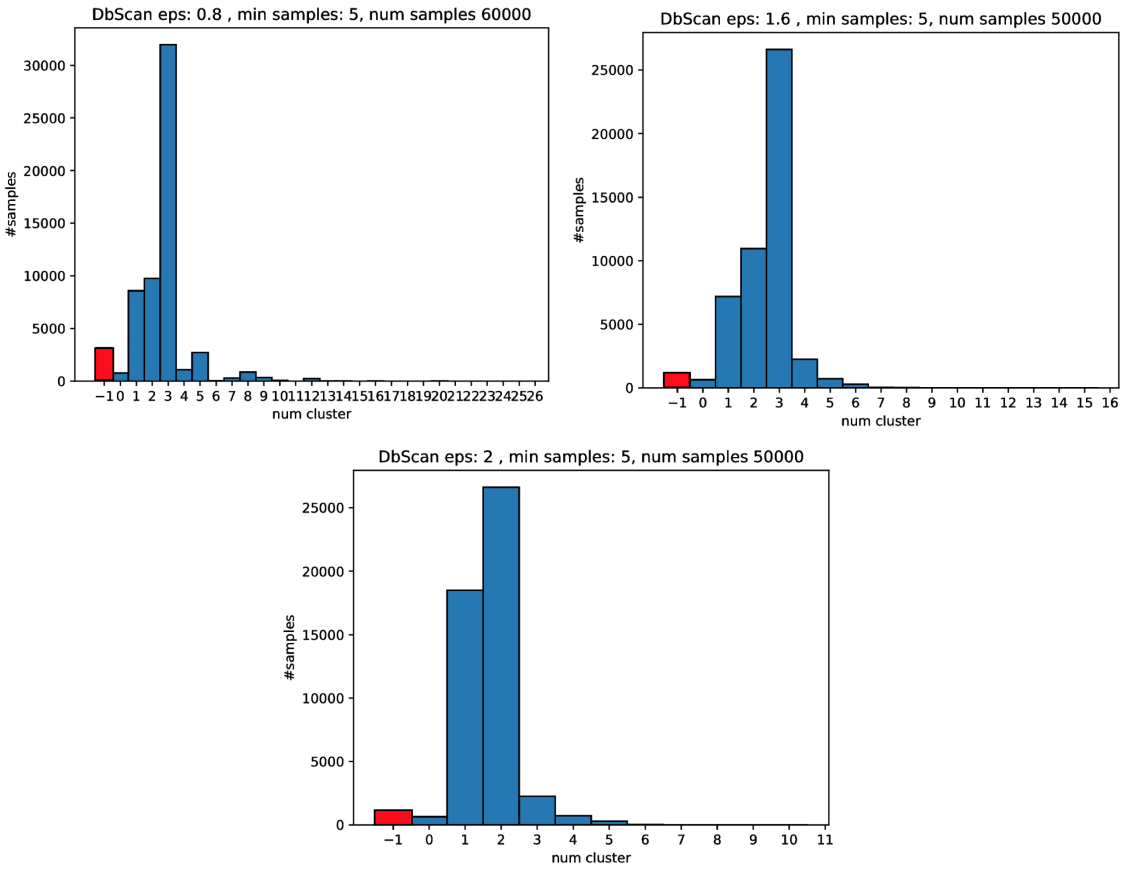


Figure 34: Cluster size, dataset *c3*, $eps = \{0.8; 1.6; 2.0\}$, $minpoints(alias\ min\ samples) = 5$. The red bin (cluster -1) contains outliers.

5.3 Hierarchical Clustering

As previously described in Section 3.5.4 this algorithm generates clusters which are organized in a structure called *dendrogram*. We perform different experiments by setting the desired number of clusters k which allows to cut the dendrogram at a specific height. Each of the experiments works by sampling only the pores with $DiameterMax > 25$. This allows to (i) focus only on bigger pores, which are mostly micro-pores and are not interesting for our analysis, and (ii) reduce the requirements of time resources since, similarly to DBSCAN, hierarchical clustering has quadratic

complexity with the number of samples.

5.3.1 Silhouette evaluation

As first analysis we run the hierarchical clustering process on several datasets and cut the dendrogram at different values of k from 2 to 9. The quality of the result is evaluated with the silhouette metric (presented in Section 3.6.1).

The result of this analysis is shown in Figure 35. Notice that, for better visualization, the values of the silhouette score are normalized with z-score. By inspecting Figure 35 the best values of k for all datasets range in $2 < k \leq 4$.

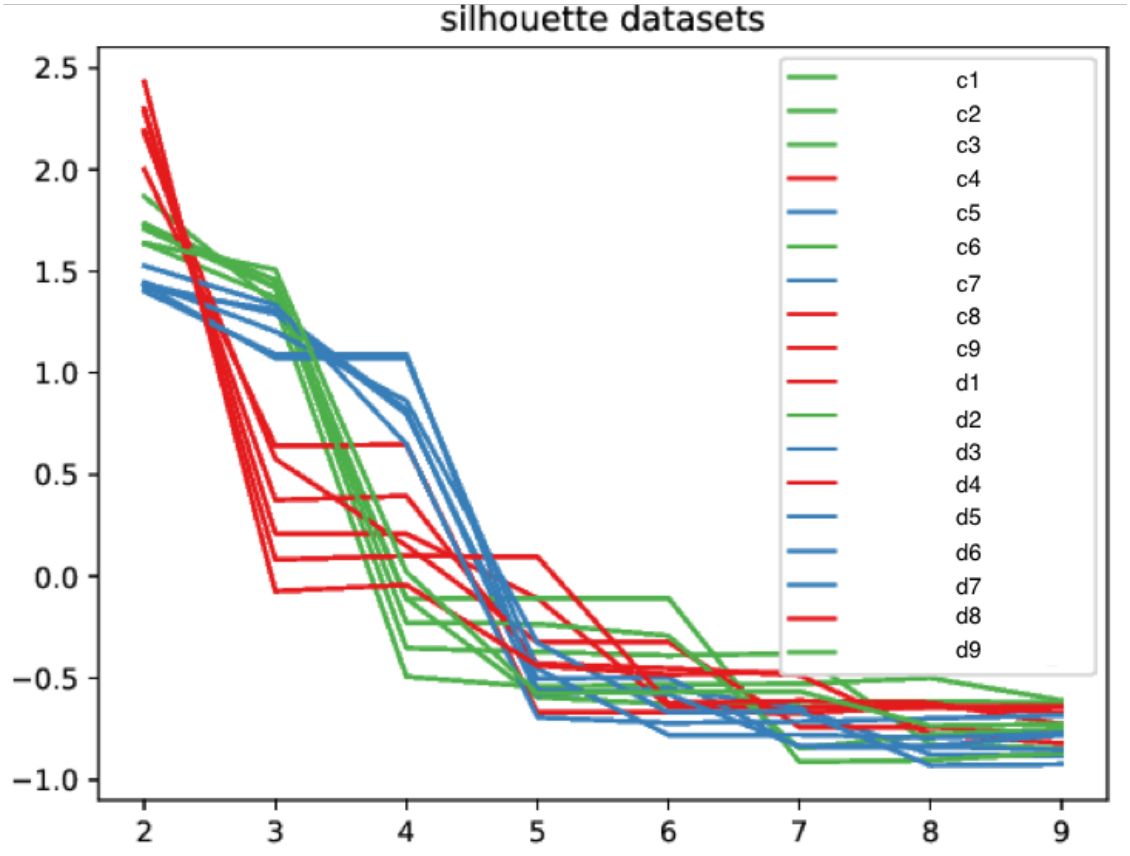


Figure 35: Silhouette score, normalized. Similar trends are marked with the same color.

From the chart in Figure 35 it is possible to detect three different groups of curve

trends. We highlighted the three groups with different colors. Each of them is composed of 6 datasets. The complete subdivision is listed below.

- Red group, curve decreases significantly after k = 2: *c4, c8, c9, d1, d4, d8*
- Green group, curve decreases significantly after k = 3: *c1, c2, c3, c6, d2, d9*
- Blue group, curve decreases significantly after k = 4: *c5, c7, d3, d5, d6, d7*

5.3.2 Cluster description

The experiments presented in this section involve hierarchical clustering on the pores with $DiameterMax > 25$ and a dendrogram cut at $k = 4$. For each clustering result we present four different plots: *dendrogram*, *PCA scatterplot*, *silhouette components* and *silhouette*. Figure 36 shows the four plots for dataset *c1*.

The first chart, shown in Figure 36a, depicts the dendrogram plot cut at $k = 4$. The dendrogram shows how the different clusters are merged together at the different levels of hierarchy. From the root of the hierarchy the clusters split in two partitions and then each of them splits again in two. The number of pores of the four partitions is shown at the bottom of the dendrogram. For example the first two clusters from the left are the biggest ones (respectively $2,592$ and $5,571$ pores).

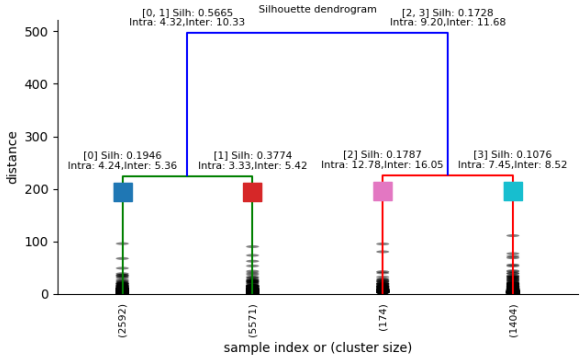
Figure 36b shows the PCA scatterplot for the different clusters. Note that the color of the clusters can be used to make comparisons with the dendrogram in Figure 36a. The clusters seem well divided as the are poorly overlapped. The red and the blue clusters are the most dense, since their points are very close among each other. From the dendrogram we can conclude that these two clusters are also the two bigger ones and they are merged together at the first stage of the hierarchy. The light-blue and the purple clusters are instead more sparse and extend to a bigger area (Figure 36b).

Figure 36c shows the intra cluster and inter cluster components of the silhouette score. The two measures are computed separately for each cluster. From this chart we

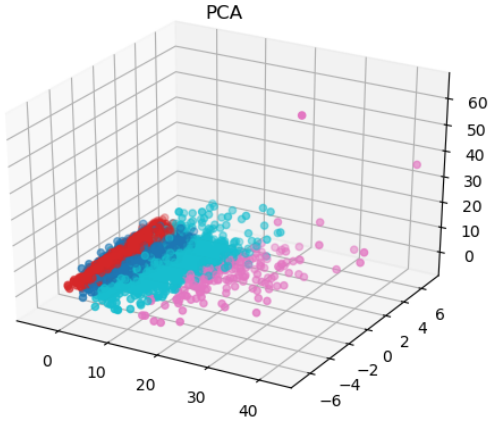
can notice that cluster 2 is the most sparse (high intra cluster component) and that cluster 1 is the most dense. This conclusion can be confirmed by looking also at the PCA scatterplot (36b).

Finally, Figure 36d presents the silhouette computed for each cluster. The cluster with the highest silhouette is cluster 1. This is confirmed by Figure 36c, since the difference between the intra cluster and the inter cluster components is very high with respect to their values.

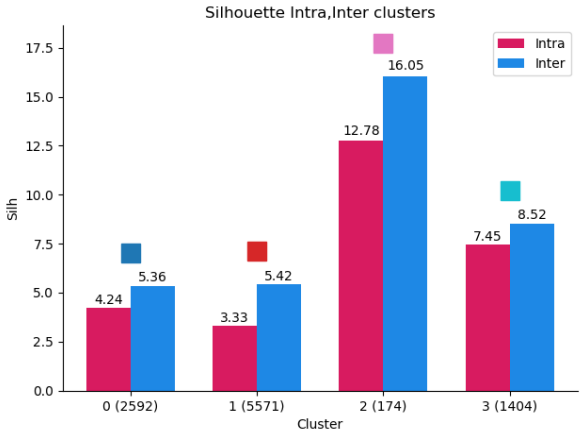
Figure 37 shows another example for the four plots presented before. Differently from Figure 36a the dendrogram in Figure36a presents three levels of hierarchy. Firstly, the two medium-sized clusters merge together (purple and blue). Afterwards the smallest (and most sparse) red cluster is added to the group. The biggest (and most dense) blue cluster does not merge until the last stage of hierarchy. The blue and purple clusters (Figure 37d) present the highest silhouette, as they are dense and well separated. In this example (Figure 37d) the smallest cluster presents only 95 elements as opposed to the 174 of Figure 36d.



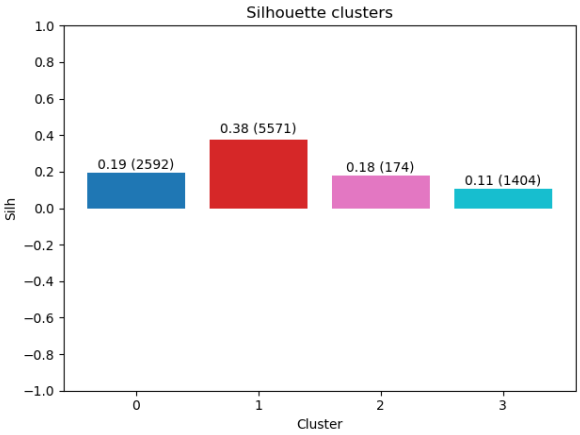
(a) dendrogram with silhouette



(b) PCA 3D scatterplot

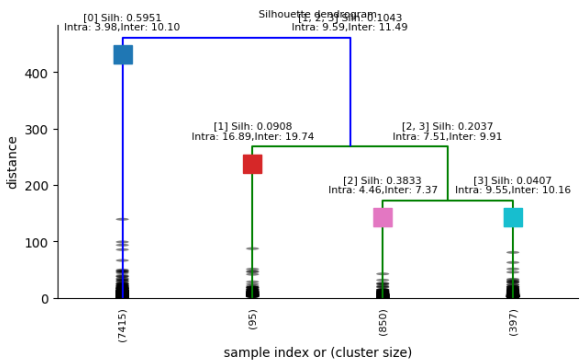


(c) Silhouette components

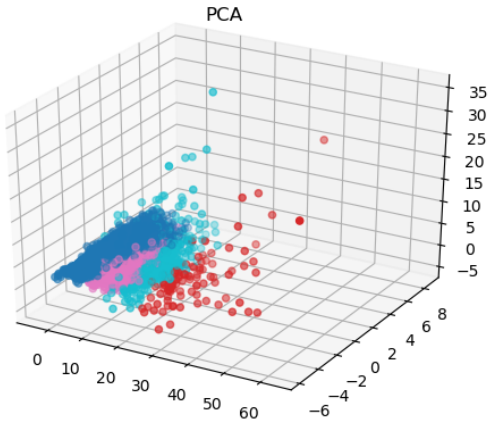


(d) Silhouette of each cluster

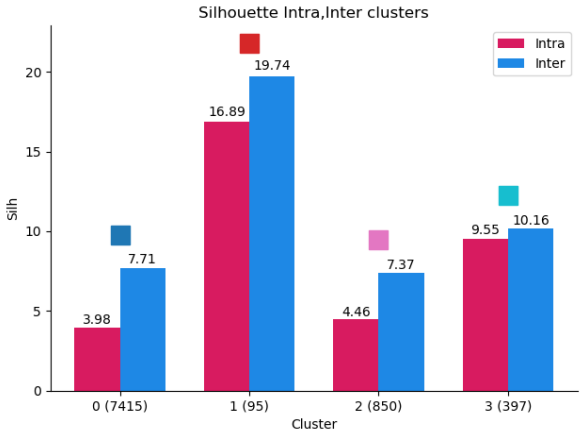
Figure 36: Clusters analysis dataset c1, k = 4, DiameterMax>25



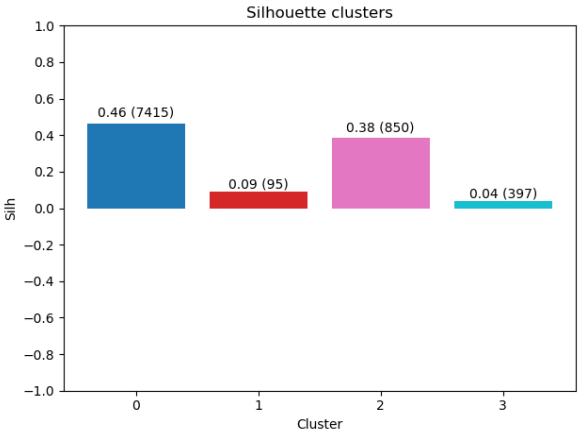
(a) dendrogram with silhouette



(b) PCA 3D scatterplot



(c) Silhouette components



(d) Silhouette of each cluster

Figure 37: Clusters analysis id d6, k = 4, DiameterMax>25

6 Conclusion

We analyzed the data collected by the experts about geological pores by using the most well-known clustering techniques of data mining. The aim was to understand which clustering algorithms give the best result in pores data classification. We report below a summary of the most significant results we achieved. The **hierarchical** clustering was initially tested on the entire datasets, but given the limitations of the algorithm in processing big datasets, we had to reduce them by using using a random sampling technique. We found two issues: the first is that the analyzed datasets did not have a fixed size, so applying a sampling that returns a fixed size we could analyze only different percentages for each dataset making the comparison between them less effectively. The second issue was that random sampling discarded the most interesting pores with a certain probability, making the result analyses less significant. Since we were interested in analyzing as much data as possible we relied on the Kmeans algorithm. The **Kmeans** algorithm (Section 5.1). This algorithm was the most efficient in terms of execution time (Section 4.4) and was able to process the entire datasets without sampling them. Although this was good because it avoided the problems due to the sampling process, it remained difficult to analyze the quality of the generated clusters because, for example, it was necessary to apply sampling for the silhouette analysis, which is quadratic in complexity. Moreover, according to domain experts it was found that the quality of clustering was not too high because the groups were not homogeneous. We also inspected the result with the **DBSCAN** algorithm (Section 5.2). In an introductory analysis we tried to find the *eps* value automatically by exploiting the distance plot in combination with the second derivative, but the results were not satisfactory. In a subsequent analysis we combined the trend of k and *eps* with the value of the silhouette to identify the best values of *eps*. Also in this case the results can be improved by using different approaches such as the multiple execution on the noise of the previous analysis, moreover DBSCAN has suffered problems related

to the sampling as for the hierarchical algorithm. After consulting with the domain experts we were able to set up a domain driven filter on the DiameterMax attribute on all datasets. This filter had the dual function of conserving the most important pores and reducing the size of the datasets, making unnecessary to perform further sampling. With this new filter we initially investigated the hierarchical algorithm (Section 5.3) because it was possible to perform the analyses without sampling. We found, inspecting the results with domain experts, that using the *ward* linkage and *euclidean* metric we obtained good results for $k = 4$. Results that we confirmed with auxiliary analysis of the dendrogram, plus silhouette and the intra/inter silhouette bar histogram. Regarding future analysis, it is worth trying the hierarchical algorithm using different linkages and metrics. In this research we conducted an introductory analysis about the silhouette score conducted on experiment with the various linkages with euclidean metric. Another future analysis concerns the study of attribute distribution within a cluster. The idea is to find which of them take in disomogeneous distribution in order to divide a cluster into sub-clusters. Also test in DBSCAN on data filtered by diameter and running multiple times to process the noise obtained from the previous executions, then comparing this result with the hierarchical could be an interesting analysis to extend our work.

References

[1] Fayyad, Piatetsky-Shapiro, Smyth, *From Data Mining to Knowledge Discovery: An Overview*, in Fayyad, Piatetsky-Shapiro, Smyth, Uthurusamy, Advances in Knowledge Discovery and Data Mining, AAAI Press / The MIT Press, Menlo Park, CA, 1996, pp.1-34

[2] Keogh, Eamonn, and Abdullah Mueen. *Curse of dimensionality* Encyclopedia of machine learning. Springer, Boston, MA, 2011. 257-258.

[3] Benesty, Jacob, et al. "Pearson correlation coefficient." Noise reduction in speech processing. Springer, Berlin, Heidelberg, 2009. 1-4.

[4] Rousseeuw, Peter J. "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis." Journal of computational and applied mathematics 20 (1987): 53-65.

[5] Gastwirth, Joseph L. "The estimation of the Lorenz curve and Gini index." The review of economics and statistics (1972): 306-316.

[6] Jolliffe, Ian. "Principal component analysis." International encyclopedia of statistical science. Springer, Berlin, Heidelberg, 2011. 1094-1096.

[7] Golub, Gene H., and Christian Reinsch. "Singular value decomposition and least squares solutions." Numerische mathematik 14.5 (1970): 403-420.

[8] Kanungo, Tapas, et al. "An efficient k-means clustering algorithm: Analysis and implementation." IEEE Transactions on Pattern Analysis & Machine Intelligence 7 (2002): 881-892.

[9] Monge, Alvaro, and Charles Elkan. "An efficient domain-independent algorithm for detecting approximately duplicate database records." (1997).