UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

Master Thesis in
Data Science and Business Informatics

Data Partitioning through Tree-based Clustering Method

Thesis Supervisor:
Riccardo Guidotti

Candidate:
Marianna Abbattista

ANNO ACCADEMICO 2021/2022

Contents

Abstract

The expanding field of eXplainable Artificial Intelligence research is primarily concerned with the development of methods for interpreting Supervised learning approaches. But we know that exists also the Unsupervised approach that maybe can have benefit, but are not so much interpretative as the Supervised one. We know that with existing clustering methods we have a result that is not easy to understand clearly. They typically return the assignment of each record to the corresponding cluster without providing the reason why we have that partitioning. Unlike previous works, we have decided to define various Tree-based clustering methods that can explain the data partitioning using a shallow Decision Tree. They make it possible to explain each cluster assignment by using a concise and simple set of split conditions. Numerous experiments demonstrate that, in terms of standard evaluation metrics and run-time on both synthetic and real datasets, our proposals is in line with both the traditional and interpretable clustering approaches that are currently in use. Last but not least, a Case Study involving real humans demonstrates the value of the interpretable clustering trees our proposal returns.

Chapter 1

Introduction

In the world of Big Data, numerous XAI approaches to explain supervised learning have been developed as a result of the growing interest in eXplainable Artificial Intelligence (XAI). [1, 19, 27]. Our goal is to improve Unsupervised learning algorithms' interpretability features paying special attention to how easily clustering algorithms can be understood.

In this context, there are two aspects to consider when evaluating interpretability. The first consideration is, how the user should interpret the record-to-cluster assignment that a clustering algorithm returns, or more specifically, what are the clustering insights? The second aspect, on the other hand, is about comprehending the reasoning behind the algorithm's data partitioning so that a human user can repeat the process, i.e., what rules are used to separate the data records?

The results returned by traditional clustering algorithms are not easy to understand. For instance, the result of a K-Means [33] clustering can be analyzed by inspecting the centroids of the clusters that characterize the different groups. Moreover, using the centroids as a "explanation" for how the clusters are globally characterized, may be difficult while there are numerous features in the datasets or similar feature-values in the centroids. Furthermore, without using an automated system to calculate the distance between the record and the centroids, a human user might encounter difficulties assigning a record to a cluster. As another example, we can consider the clustering returned by a Max-Linkage hierarchical clustering approach [33]. This time, in addition to the record assignments for a specific level of the hierarchy, the method also returns a dendrogram to the user. The visualization of the hierarchy, in the case of large datasets, only provides an idea of the distance at which the clusters were joined, not why, because it is not possible to see the instances at the bottom of the dendrogram. Moreover, a human user cannot understand which branch of the dendrogram to follow to assign a record to a cluster while attempting to replicate the logic behind the Max-Linkage algorithm. These considerations are in line with various recent studies on interpretable clustering [5, 10, 14, 16, 24, 34].

Some of these works argue that clustering should be derived from Unsupervised binary trees [5,10,14,24]. In a binary tree, labels on the leaves represent clusters, and each node is linked to a feature-threshold pair that recursively divides the dataset. Any cluster assignment can be explained with a small number of thresholds if shallow trees are taken into account. An Unsupervised binary tree, on the other hand, provides more information than conventional clustering methods for large and high-dimensional datasets, because the user only needs to read the tree to comprehend the criteria used to perform the clustering and to assign a record to a cluster. Trees are well known for being easy to interpret and simple to train with, for that reason Decision Trees are typically used for Supervised problems and can handle both Classification or Regression problems being interpretable by design [7, 19, 31].

In this thesis, we improve the state-of-the-art by developing a **Tree-based clustering method** that yields an Unsupervised binary tree to interpret the data partitioning. The main innovation, in our proposal, is in the clustering method, which simultaneously builds the tree and partitions the data, which searches for the best splits over subsequent iterations. On the other hand, current state-of-the-art approaches [5, 10, 14, 24] returning clustering trees, first apply a traditional clustering algorithm, e.g. K-Means, and then try to infer a tree that approximates the clustering. The following are the main benefits of our suggestion. First, the methodology used to identify clusters is the same as that used to construct the tree. Second, there is no need to use another clustering algorithm with all the associated issues with effectiveness, parameter tuning, and appropriate distance function selection. We name our proposal Partitioning Tree (ParTree), and we employ three versions that differ from one another in terms of the criterion used to perform the data splitting to demonstrate its efficacy. Additionally, in contrast to the development of mainstream clustering techniques, ParTree can be applied to numerical, categorical, and mixed datasets.

We tested ParTree as well as a wide range of conventional clustering techniques and interpretable clustering techniques on both synthetic and real datasets. The study demonstrates that ParTree is competitive with or frequently outperforms state-of-the-art methodologies. Additionally, we carried out a Case Study, with the help of Google Forms, in which participants were required to complete specific tasks, based on the clustering explanations provided by various clustering algorithms. The class of first year students of the Data Science course participated in the survey. The survey's results demonstrate that using Tree-based clustering techniques improves how well users perform their tasks.

The rest of the thesis is organized as follows. Related works are reviewed in Section 2. In Section 3 we formalize the problem we face and we report some background definition, also on the competitors against which we are going to test the results and the measures involved, while in Section 4 we illustrate our proposals, inserting first a part with the main algorithm, and then three subsections for each split criterion that we are going to use. Section 5 reports the dataset used, other specifics about the competitors, evaluation metrics that we decide to take into account, and experimental setting and relative results while Section 6 illustrates the case study and shows the results obtained. The summary of our contributions and future research directions is presented in Section's 7.

Chapter 2

Related Works

In this section, we present a self-contained literature review of interpretable clustering methods using Tree-based models as well as Tree-based Unsupervised approaches. We classify Unsupervised trees that divide the data space without taking into account any loss function in the first group, such trees are not optimal for a specific purpose but are typically adopted for other supervised tasks.

For instance, K-D trees [23], often used for speeding up nearest neighbor computing, is a space-partitioning data structure for organizing points in a K-dimensional space that splits a node using the median value of a feature as the threshold. The selected feature often has the largest variance in the data. Other similar approaches consist of Random Projection (RP) tree [12] and Principal Component Analysis (PCA) trees [36]. These methods are similar to K-D trees, except that they choose a random direction in the case of RP trees and the median value along the first eigenvector of the covariance matrix in the case of PCA trees. Other PCA trees [15] use direction spanned by centers of a 2-means clustering as the separating direction. Approximate Principal Direction (APD) trees [26] presents a procedure that ensures accuracy similar to PCA trees but being computationally faster. The overall idea of these approaches is to create balanced trees and, similarly to K-Means [33], to reduce the average distance between points in the same partition as much as possible. We continue the line of inquiry begun by these approaches in our proposals, but we rely on splitting conditions designed to locally identify the best split that ensures data similarity with respect to heuristically maximized splitting criteria.

As an alternative, we classify methods that attempt to locally or globally optimize various metrics as belonging to the second group of approaches for determining the partitioning features. In [6] is proposed one of the very first approaches that adapt the basic top-down Induction of Decision Trees method towards clustering (TIC). TIC is a first-order clustering system as it does not employ the classical attribute value representation but that of first-order logical decision trees. The maximum distance between two prototypes of two partitions is used as the splitting criterion, while the F-test is used as the stopping condition. In [24] is presented CLTree (CLustering based on Decision Trees), a technique based on a supervised decision

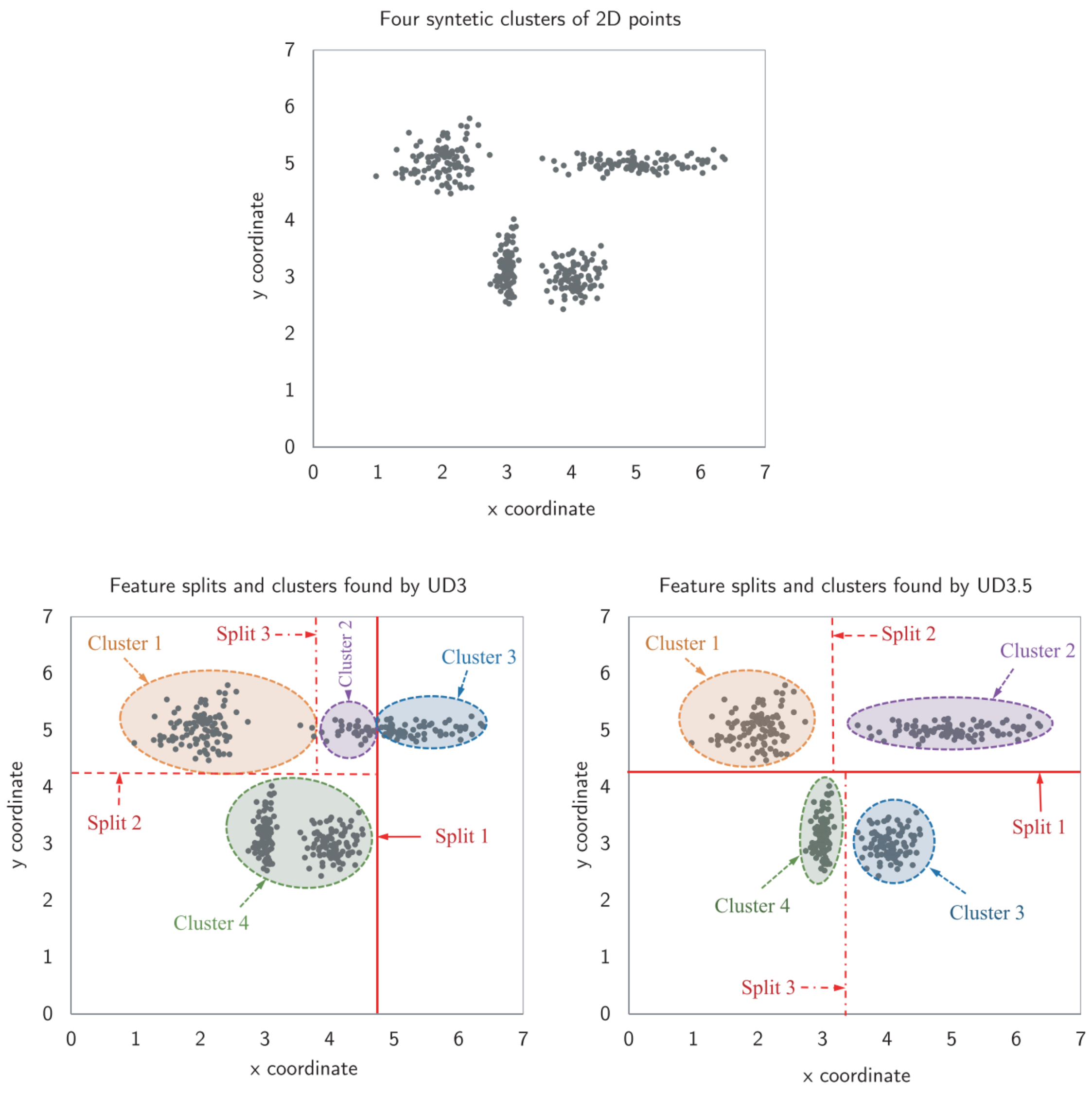


Figure 2.1: Synthetic 2D points with the clusters computed by UD3 and by the algorithm UD3.5.

tree construction. CLTree adopts a modified decision tree algorithm to construct a cluster tree to capture the natural distribution of the data by inserting uniformly at random synthetic points in sparse regions at different levels of detail. After that, the tree is built, and an interactive pruning step simplifies the tree to find useful clusters. In [35] is described a simplified solution w.r.t. CLTree to perform sales forecasting. In particular, K-Means is executed, and the clusters assignment labels are used as classes for a Decision Tree classifier. In [11] is proposed TASC, Two-Attribute-Set clustering through Decision Tree construction for (rare) cases in which two sets of features are available. TASC partitions data space by constructing a decision tree using one attribute set and measures the degree of similarity using another one.

In [14] is presented a three-stage procedure to retrieve interpretable clustering using Unsupervised Binary Trees (CUBT). The first step is to perform a series of recursive binary splits to reduce the heterogeneity of the data. After that, the pruning step, aggregates adjacent nodes. In the last step, such procedure joins similar clusters. Finally, during the joining step, similar clusters are joined together,even if they do not share the same parent. In our opinion, this last step decreases the interpretability of the clustering as the tree structure can be destroyed. Thus, we will consider only 'valid' partitioning trees. In [17] is proposed an extension of CUBT to nominal data through heterogeneity criteria and dissimilarity measures, based on mutual information and entropy. In [20] is proposed a Pattern-based clustering algorithm for Numerical datasets (PCN) that extracts a subset of patterns to cluster the data. The procedure starts by deriving multiple Unsupervised trees using an approach similar to TIC [6]. However, the final goal is to obtain clusters described by patterns and not a single tree responsible for the overall clustering structure.

In [20] is presented UD3, which is a Pattern-based Clustering algorithm for numerical datasets that extracts just a subset of patterns useful for clustering, without applying an a priori discretization on numerical features. After that, create binary unsupervised decision trees making splits that maximize the difference between the feature-value means (centroids) of the children nodes. In [25] is presented UD3.5 an extension of UD3 [20]. UD3.5 introduces two novel quality measures to control the split, it takes into account both separation and compactness, and it does not require any empirical parameter to control the depth of the trees. An important advantage of this proposal, is that it can provide patterns associated with each cluster, describing the whole database with a few patterns by keeping those which are more general. This kind of description is useful in some applications,but works only with continuous features. In the Figure 2.1 are shown how UD3 and UD3.5 divide the space making split for creating the clusters. The more recent approaches returning clustering trees are explicitly thought to answer interpretability issues as they were realized after the increase of interest in XAI [1, 19].

DReaM, Discriminative Rectangle Mixture, presented in [10], learns a rectangular decision rule for each cluster through a probabilistic discriminate model. In DReaM, the features for generating rules do not have to be the same as the features used for discovering cluster structures. We notice that it creates an asymmetry between the clustering procedure and the insights given by the returned interpretation. In [28] is presented an explainable K-Means and K-Medians Clustering named Iterative Mistake Minimization. It starts by running K-Means or K-Medians, then it builds a binary tree recursively following a top-down procedure that splits a node identified by a rectangle of thresholds if it contains two or more of the centers previously identified. The idea is to minimize the number of mistakes at each split, i.e., the number of points separated from their corresponding cluster center. In [16] is shown how ExKMC extends IMM by allowing the construction of less interpretable but more accurate clustering trees. In [34] is proposed an extension of K-Means named K-Means Tree (KMT), to be fast both when building the clustering and when assigning a point to a cluster by learning the tree and the centroids optimally and jointly.

The problem is faced as a constrained minimization problem and then solved using a quadratic penalty method. KMT learns clusters from K-Means and gradually adapts centroids to the outputs of an optimal oblique tree. Finally, in [4, 5] is presented ICOT, Interpretable Clustering via Optimal Trees [3]. ICOT uses Mixed Integer Optimization techniques to generate a Tree-based clustering model, can naturally determine the optimal number of clusters, and manage mixed numerical and categorical data. In ICOT, interpretability is considered during cluster creation rather than as a later analysis step.

We place our proposal between these two groups. Indeed, as for the second group, our objective is to *(i)* cluster a dataset effectively and efficiently and *(ii)*, to obtain an interpretable tree explaining the criteria to separate the data among the various partitions. However, similarly to the first group, we aim at *(iii)* building/inducing the tree directly on the dataset analyzed without relying on other (post-hoc) clustering approaches but *(iv)* by following local heuristic splitting criterion. According to our suggestion, the returned clustering tree will match the logic that was successfully applied to cluster the data. Our proposal also aims to handle mixed, continuous, and categorical data.

Finally, in the literature there are also other approaches, mainly related to biological applications, adopting trees as indexing micro-clusters for anytime stream mining [22], as a starting point for the clustering (TreeCluster) [2], and for visualization purposes, after having applied a traditional clustering algorithm (ClusTree) [38].

Chapter 3

Setting the Stage

In this section we define the problem we want to solve and we provide the notions necessary to comprehend our proposal. Given a set of n records $X = \{x_1, \ldots, x_n\}$, where each observation is a d-dimensional vector, we define the clustering problem as the partitioning of X into $k < n$ disjoint sets (or clusters) $\mathcal{C} = \{C_1, \ldots, C_k\}$, such that $\mathcal{C}$ is optimal in terms of homogeneity and simplicity. This means that the records in each cluster C_i are typically more similar to the records of C_i than to the records of any other cluster C_j with $i \neq j$. In other words, inter-cluster distances should be minimized while intra-cluster distances should be maximized.

In the following Sections 3.1, 3.2 and 3.3, we initially introduce what are the main families of clustering algorithms with which we are going to compare ParTree. In each of these, the algorithms that we decided to test in the experimental phase in Chapter 5 will be explained.

3.1 Centroid-based Clustering Methods

In Centroid-based clustering, each cluster is represented by a center vector called a *centroid*, which is not necessarily a member of the data set. Now we introduce some of the most popular algorithms of this family.

K-means [33] is a clustering method for numerical data that aims to partition n records into k clusters in which each record belongs to the cluster with the nearest mean $\mu_i = \text{mean}(C_i) \in \mathbb{R}^d$, i.e., cluster centers or centroid, serving as a prototype of the cluster. The distance of a record x with a center μ is typically measured using the Euclidean distance. In practice, K-means minimizes within-cluster variances.

K-modes is an adaptation of K-Means to categorical data [33]. In this setting the mean is replaced with the mode, i.e., $\mu_i = \text{mode}(C_i) \in \mathbb{D}^d$ while as distance is used the Mismatch Edit distance or the Jaccard distance instead of the Euclidean distance.

Bisecting K-Means,or BK-means, algorithm is a modification of the K-means algorithm. Any size and shape of cluster can be recognized by it. This algorithm is practical due because when measuring entropy, it outperforms K-Means. Furthermore BK-means is more efficient when K is large. While, the K-means method uses K centroids and each data point from the data to compute a result, on the other hand, in each Bisecting stage of BK-means, only the data points from one cluster and two centroids are utilized. For that reason is faster than K-means. Finally, it is well known that K-means produces clusters of various sizes, BK-means produces clusters of similar sizes.

X-means [29] is a clustering algorithm that seeks to automatically calculate the number of clusters based on Bayesian Information Criterion (BIC) scores. It is an expanded K-means algorithm. After each iteration of K-means, the X-means algorithm enters the picture, deciding locally which subset of the existing centroids should divide themselves to better fit the data. The BIC score is used to make the splitting decision and stop the algorithm.

3.2 Density-based Clustering Methods

Density-Based clustering refers to Unsupervised learning methods that identify distinctive groups/clusters in the data, based on the idea that a cluster in a data space is a contiguous region of high point density, separated from other such clusters by contiguous regions of low point density. The data points in the separating regions of low point density are typically considered noise/outliers. [32].

DBSCAN stands for density-based spatial clustering of applications with noise. It can locate clusters with noise and clusters of arbitrary shapes (i.e. outliers). DBSCAN's central tenet is that, a point belongs to a cluster if it is near numerous other points from that cluster. There are two essential components of DBSCAN.: *eps*: the isolation defining the neighborhoods. If there is a difference between two points of less than or equal to eps, they are said to be neighbors. *minPts*: the bare minimum of data points required to define a cluster. Points are grouped together into a cluster based on these two factors.

OPTICS clustering, stands for ordering points to identify cluster structure. It takes its cues from the clustering algorithm DBSCAN. It expands on the DBSCAN clustering concepts by two more terms. The *core distance* is the smallest radius necessary to designate a given point as a core point and the *reachability distance* that between two points p and q, it is the maximum between the *core distance* of p and the euclidean distance (or some other distance metric) between p and q.

3.3 Hierarchical Clustering Methods

In a Hierarchical clustering process, each observation is first treated as a separate cluster, then, it repeatedly completes the next two actions: determine the two clusters that are most similar to one another, and then combine those two clusters. This iterative procedure keeps going until all of the clusters are combined. For Hierarchical clustering, it is essential to know how close together the two clusters are.

There are several methods for measuring the distance between two clusters, and these methods determine the clustering rule.

Linkage methods are the name given to these measures. The **Single/Min Distance** is the closest distance of the clusters between the closest Points. The **Complete/Max Distance** represents the greatest separation between the centers of two distinct clusters. Since it creates more compact clusters than single-linking, it is one of the widely used linkage techniques. We will call this method later H-Max. The **Average Distance** is used to find the average distance between two clusters by adding up the distances between each pair of datasets and dividing the result by the total number of datasets. At least, **Ward's Distance**, contrary to the others, examines the variance of clusters rather than taking a straight measurement of distance.

3.4 Impurity Measures and Error Measures

In this section, we introduce the different measures involved in the thesis work. Initially, the Impurity measures 3.4.1 like Entropy and Gini, which will be mainly used in the ParTree Impurity-based 3.4.1 version, then the Mean Squared Error, which will be used in the ParTree Center-based 4.1 version Finally, the Bayesian Information Criterion metric 3.4.3, which unlike the others will be used in all the ParTree versions as the stop criterion of the 'algorithm.

3.4.1 Impurity Measures

The Impurity function, measures the extent of purity for a region containing data points from possibly different classes or in our case cluster, for that reason the closer to zero the better. There are three commonly used impurity measures used in binary decision trees that we also decide to use in our setting: *Entropy, Gini index, and Misclassification Error.*

Entropy It is a way to measure impurity, which is 0 if all samples of a node belong to the same class, and the entropy is maximal if we have a uniform class distribution.

$$-p\log(p) - (1-p)\log(1-p)$$

Gini Index Computes the degree of probability of a specific variable that is wrongly being classified when chosen randomly and a variation of the Gini coefficient. It works on categorical variables, provides outcomes either be "successful" or "failure" and hence conducts binary splitting only.

The degree of the Gini index varies from 0 to 1, where 0 depicts that all the elements are allied to a certain class, or only one class exists there. The Gini index of value 1 signifies that all the elements are randomly distributed across various classes, and a value of 0.5 denotes the elements are uniformly distributed into some classes. [7]

$$1 - p^2 - (1-p)^2$$

Classification Error Is a metric that describes the percentage of observations that were incorrectly predicted by some classification model. The value is in a range from 0 to 1 where: 0 represents a model that had zero incorrect predictions, and 1 represents a model that had completely incorrect predictions. The lower the value, the better a classification model is able to predict the outcomes of the response variable.

$$1 - \max(p, 1-p)$$

3.4.2 Mean Squared Error

The Mean Squared Error (MSE) it's a kind of metric that was used for minimizes the error, it is the distance between the points and the centroid of each cluster that was created. In section 4.1 it was explained how it was used in our algorithm. The objective like we said before is to minimize this error.

Consider a dataset X_{ij} where $(i = 1, ..., n)$ and n is the amount of data that we are considering for the split, $(j = 1, .., m)$ with m are the features where we calculate the centroid. In the formula, x_{ij} is the data point considering the value and the features which belong, and c_{Kj} is the centroid of that cluster for that features. Giving the name of MSE_{iK} can be used Euclidean formula, namely :

$$MSE_{iK} = \sqrt{\sum_{j=1}^{m}(x_{ij} - c_{Ki}) \cdot 2}$$

A data will be a member of the K-Cluster if the data distance to the center of the K-Cluster is the smallest compared to the distance to the other Cluster center.

3.4.3 Bayesian Information Criterion

Bayesian Information Criterion, or BIC, is a metric that, given the data D and a family of alternative models (split in our case) M_j, different models correspond to solutions with different splits, stop the splitting criterion for finding the best K for the number of clusters.

For deciding when is the moment to stop, use the posterior probabilities $Pr[M_j|D]$ to score the split/models. To approximate the posteriors, up to normalization, use the following formula from Kass and Wasserman(1995):

$$BIC(M_j) = \hat{l}_j(D) - \frac{p_j}{2} \cdot \log R$$

Where $\hat{l_j}(D)$ is the log-likelihood of the data according to the $j - th$ split/model and taken at the maximum likelihood point, and p_j is the number of the parameters in M_j. The number of parameters p_j is simply the sum of $K - 1$ class probabilities, $M \cdot K$ centroid coordinates, and one variance estimate. To extend this formula for all centroids instead of one, use the fact that the log-likelihood of the points that belong to all centroids in question is, the sum of the log-likelihood of the individual centroids, and replace R above within the total number of points which belong to the centroid under consideration. [30]

3.5 Principal-component analysis

Here we introduce Principal-Component Analysis (PCA), this technique will be used in our algorithm for creating the Principal-component based split 4.3. The purpose is to find the directions in which the tuples best align in the plane and best perform the split.

Principal-component analysis, or PCA, is a technique for, taking a dataset consisting of a set of tuples representing points in a high-dimensional space, and finding the directions along which the tuples line up best.

The idea is to treat the set of tuples as a matrix M and find the eigenvectors for MM^T or M^TM. The matrix of these eigenvectors can be thought of as a rigid rotation in a high-dimensional space. When applied to this transformation to the original data, the axis corresponding to the principal eigenvector, is the one along which the points are most "spread out". More precisely, this axis is the one along which the variance of the data is maximized. Put another way, the points can best be viewed as lying along this axis, with small deviations from this. Likewise, the axis corresponding to the second eigenvector (the eigenvector corresponding to the second-largest eigenvalue) is the axis along which the variance of distances from the first axis is greatest, and so on.

We can view PCA as a data-mining technique. The high-dimensional data can be replaced by its projection onto the most important axes. These axes are the ones corresponding to the largest eigenvalues. Thus, the original data is approximated by data that has many fewer dimensions and that summarizes well the original data [21].

Chapter 4

ParTree Algorithm

In this section, we describe Partitioning Tree (ParTree). Inspired by BK-means [33], ParTree is an interpretable Tree-based clustering method. The interpretability aspect is given by the Unsupervised binary tree returned by the algorithm, besides the record to cluster assignments. More in detail, ParTree is based on Hierarchical top-down iterative bisections to find the best feature to partition the data.

We highlight that, differently from BK-means, the crucial point of ParTree is that the data partitioning at every tree level is done by selecting a unique feature value to separate the data to guarantee cohesion among the records in the same partition. Indeed, in short, ParTree aims at guaranteeing interpretability without sacrificing the clustering quality. We start by introducing the overall algorithm, then we present three implementations of ParTree following different partitioning principles.

In line with [18,30], we handle the clustering problem through a top-down, divide-and-conquer strategy: ParTree starts from an initial set containing a single cluster, then, iteratively it tries to partition a cluster in two sub-clusters.

The general schema of ParTree, which implements this approach, is illustrated in Algorithm 1. ParTree starts by initializing an empty set of sets $\mathcal{C}$ containing the clustering result (line 1) and by building the root of the tree $\mathscr{R}$ containing the whole dataset X (line 2). Then, it pushes both $\mathcal{C}$ and $\mathscr{R}$ into a priority queue $\mathcal{Q}$ that keeps track of the set of candidate clusters to be considered for partitioning (line 3). The priority of the objects in the queue is given by a q-score$(\cdot)$ function that can be implemented in different ways. In our experiments, we considered two alternatives. First, the size of the cluster analyzed, i.e., priority is given to larger clusters which should be partitioned before smaller ones. Second, by using the $bic(\cdot)$ function, i.e., priority is given to more variegated clusters according to the BIC [30] measure.

At each iteration, a candidate cluster C and the tree node $\mathscr{N}$ modeling it are extracted from $\mathcal{Q}$ (line 5). If the cluster is too small (min_sample) to be partitioned or the tree node $\mathscr{N}$ is too deep (max_depth), then C is added to the clustering result from $\mathcal{C}$, the node $\mathscr{N}$ is turned into a leaf $\mathscr{L}$, and the control passes to the

Algorithm 1: $\textsc{ParTree}(X, \mathit{max\_clusters}, \mathit{max\_depth}, \mathit{min\_sample}, \varepsilon)$

```
Input  : $X$ - dataset, $max\_clusters$ - maximum number of clusters,
         $max\_depth$ - maximum tree depth, $min\_sample$ - minimum
         cluster size, $\varepsilon$ - percentage of BIC parent discount
Param  : q-score - queue score function
Output: $\mathcal{C}$ - clustering, $R$ - clustering tree

1  $\mathcal{C} \leftarrow \emptyset$;                                    // init. clustering result
2  $\mathscr{R} \leftarrow$ make_node($X$);                               // init. tree root
3  $\mathcal{Q} \leftarrow push(\mathcal{Q}$, q-score($X$), $\langle X, \mathscr{R} \rangle)$;   // init. priority queue
4  while $|\mathcal{Q}| > 0 \wedge |\mathcal{C}| + |\mathcal{Q}| < max\_clusters$ do
5      $\langle C, \mathscr{N} \rangle \leftarrow pop(\mathcal{Q})$;      // extract tree node from queue
6      if $|C| < min\_sample \vee depth(\mathscr{N}) > max\_depth$ then
7          $\mathcal{C} \leftarrow \mathcal{C} \cup C$; $\mathscr{L} \leftarrow$ make_leaf($\mathscr{N}$);   // add cluster and make leaf
8          continue;                                                      // go next iteration
9      $f, C_1, C_2 \leftarrow$ make_split($C$)                            // make split
10     if $bic(C) < bic([C_1, C_2]) - \varepsilon |bic(C)|$ then
11         $\mathcal{C} \leftarrow \mathcal{C} \cup C$; $\mathscr{L} \leftarrow$ make_leaf($\mathscr{N}$);   // add cluster and make leaf
12         continue;                                                      // go next iteration
13     $\mathscr{N}_l \leftarrow$ make_node($C_1$);                        // make left node
14     $\mathscr{N}_r \leftarrow$ make_node($C_2$);                        // make right node
15     $\mathscr{N} \leftarrow$ update_node($f, \mathscr{N}_l, \mathscr{N}_r$)   // update tree
16     $\mathcal{Q} \leftarrow push(\mathcal{Q}$, q-score($C_1$), $\langle C_1, \mathscr{N}_l \rangle)$;   // update queue
17     $\mathcal{Q} \leftarrow push(\mathcal{Q}$, q-score($C_2$), $\langle C_2, \mathscr{N}_r \rangle)$;   // update queue
18 while $|\mathcal{Q}| > 0$ do
19     $\langle C, \mathscr{N} \rangle \leftarrow pop(\mathcal{Q})$;      // extract tree node from queue
20     $\mathcal{C} \leftarrow \mathcal{C} \cup C$; $\mathscr{L} \leftarrow$ make_leaf($\mathscr{N}$);   // add cluster and make leaf
21 return $\mathcal{C}, \mathscr{R}$;
```

next iteration (lines 6–8). On the other hand, the partitioning of the candidate cluster C is performed by make_split($\cdot$) (line 9) that returns a binary partitioning function f, as well as the partitioning obtained by f on C, i.e., C_1, C_2, such that $C = C_1 \cup C_2$. Alternative ways to implement this function are illustrated and discussed in the subsection of this chapter. After that, lines 10–11 calculate the BIC (see Section 3.4.3) on the candidate cluster C and the two sub-clusters C_1, C_2. If the *data partitioning* of C into C_1 and C_2 *is not advantageous* then keeping C united to the BIC score, then C is added to the clustering result in $\mathcal{C}$, the node $\mathscr{N}$ is turned into a leaf $\mathscr{L}$, and the control passes to the next iteration (lines 10–12). Otherwise, if the *data partitioning is advantageous* for the clustering, then novel tree nodes $\mathscr{N}_l$ and $\mathscr{N}_r$ are created for sub-clusters C_1 and C_2 (lines 13–14), and linked to the parent node together with the partitioning function f (line 15). After that, the novel candidate clusters C_1, C_2 with their corresponding tree nods $\mathscr{N}_l, \mathscr{N}_r$ are pushed into the priority queue $\mathcal{Q}$ (lines 16–17). The $\varepsilon \in [0, 1]$ parameter controls to which extent the BIC of the children must be lower than the BIC of the parent w.r.t. the absolute BIC of the parent. If $\varepsilon = 0$, this parameter has no impact. On the other hand, the higher ε, the lower the level of BIC of the children required to have an advantageous split. In other words, a $\varepsilon > 0$ relaxes the condition to stop the partitioning and allows to obtain of more clusters.

The algorithm termination is controlled by three conditions: *(i)* the maximum number of admissible clusters (*max_ clusters* parameter), *(ii)* the maximum admissible tree depth (*max_ depth* parameter), *(iii)* if it is worth to split the clusters obtained so far w.r.t. the BIC criterion. The while-loop (lines 4–17) ends either when the queue is empty or when the number of candidate clusters and the clusters contained in the clustering result exceeds the maximum number of admissible clusters. If the queue is not empty after the first while-loop, a further loop completes the clustering and the tree construction (lines 18–20). Finally, ParTree returns the clustering and the unsupervised binary partitioning tree.

4.1 Center-based Split

Center-based split, or **C-PT**, is our first splitting function that can handle continuous, categorical, and mixed data. This split optimize the Mean Squared Error (MSE) 3.4.2 at each iteration. We imply that the values of each training vector are normalized without losing generality. The details are implemented in Algorithm 2. We start selecting a distance function For continuous features we use *Euclidean distance*, for categorical ones the *Mode*, and for the mixed types we use *Seuclidean distance* and *Jaccard distance.* These functions will be used in lines 10-11 when we calculate the MSE selecting one among them according to the type of features we are going to treat.

In line 1, we initialize j^* which contains the index of the best feature to split over, t^* for the threshold index of that specific feature, and the MSE^* for the split.

The outermost loop performs a single selection of features j, and the induction algorithm creates, for each value v of that specific feature, two candidate splits (two new nodes) with the properties $f_j <= v_i$ and $f_j > v_i$, respectively, for making the binary partitions. In each of the two nodes, we insert the records that satisfy that particular condition (line 4). Then we do the split, if one of them is empty we skip the split and continue the iteration (lines 6–7). Then, for each new node, we compute the average of the values in the feature f_j creating the two centroids (lines 8–9). This average represents the mean of that node.

In lines 10-11, we can calculate the MSE of the two splits. After that, we sum the two MSEs obtained, and we have the total MSE for that split on that value of the features. We then compare if the obtained MSE, is better than the previous one stored within MSE^*(line 13), i.e. if the new MSE is lower we will store the new MSE value of the split. At the first iteration, this is always true and we assign the new value of MSE to MSE^*, the j features, and the t threshold.

Algorithm 2: C-PT split(X)

```
Input  : $X$ - dataset
Param  : dist - distance function
Output: $f$ - binary partitioning function, $C_1, C_2$ - cluster partitions
$j^* \leftarrow \infty$; $t^* \leftarrow \infty$; $MSE^* \leftarrow \infty$;      // init. split feature, threshold, best MSE
for $j \in [1, d]$ do                                        // for every feature
    for $t \in values(X^{(j)})$ do                           // for every value
        $f \leftarrow$ make_partitioner($cond(X^{(j)}, t)$);      // build data partitioner
        $X_a, X_b \leftarrow$ split_data($f, X$);                 // make binary partitions
        if $|X_a| = 0 \vee |X_b| = 0$ then
            continue;                                        // go next iteration
        $\mu_a \leftarrow$ get_centroid($X_a$);                   // make centroid
        $\mu_b \leftarrow$ get_centroid($X_b$);                   // make centroid
        $MSE_a \leftarrow \frac{1}{|X_a|} \sum_{x \in X_a} \text{dist}^2(\mu_a, x)$;   // calculate MSE partition a
        $MSE_b \leftarrow \frac{1}{|X_a|} \sum_{x \in X_b} \text{dist}^2(\mu_b, x)$;   // calculate MSE partition b
        $MSE \leftarrow \frac{|X_a|}{|X|} MSE_a \frac{|X_b|}{|X|} MSE_b$;      // calculate total MSE
        if $MSE < MSE^*$ then                                // if better partitioning
            $j^* \leftarrow j$; $t^* \leftarrow t$; $MSE^* \leftarrow MSE$;   // update split feature, thr, MSE
$f \leftarrow$ make_partitioner($cond(X^{(j^*)}, t^*)$);         // build data partitioner
$C_1, C_2 \leftarrow$ split_data($f, X$);                        // make binary partitions
return $f, C_1, C_2$;
```

We repeat this procedure until we finished the possible splits. Then we build the data partitioner and split the data into two clusters for having the binary partition that minimizes the MSE. We return the selected features and the two clusters.

The idea besides the splitting criterion in C-PT is similar to [6] where was implemented TIC. This algorithm given a cluster C and a test T creates two sub-cluster $C1$ and $C2$ of C. TIC computes the distance $d(p(C1); p(C2))$, where p is the prototype function, this was not implemented in C-PT. The best test T is then the one that maximizes this distance. Instead in our case is the minimization of the MSE that give us the best split.

4.2 Impurity-based Split

Impurity-based split, or **I-PT**, is the second proposal, which also aims at handling continuous, categorical, and mixed data, like the first one. In this case, we use as parameters one of the Impurity functions 3.4.1 that can be chosen between *Entropy* 3.4.1, *Gini Index* 3.4.1 and *Classification Error* 3.4.1, and *r2 score* or *mean absolute percentage error*. It is possible also, to select between three different aggregation function, the *mean* that gives the mean value between all the value in the array, *min*, which return the minimum of the array, and *max*, that returns the max of that array.

We initialize j^*, which is the index of the best feature to split over, t^*, which is the threshold index for that specific feature, and h^*, the best impurity value that we retrieve. The most external for loop performs a single selection of features j, and the induction algorithm creates, for each value v of that specific feature, two candidate splits (two new nodes) with the properties $f_j <= v_i$ and $f_j > v_i$, respectively. Also in this case, if the split produces an X split empty, it goes on to the next value v.

In line 8, we initialize an empty Impurity list H. For every target feature, if this is equal to itself we continue, otherwise, we calculate the Impurity of each of the two partitions created in line 5.

The Impurity of each partition was calculated on the basis of the feature type. Depending on the type of data in the feature that the algorithm is processing, Impurity functions for categorical or continuous features will be used to calculate the single impurity value. The total Impurity was calculated by summing the two Impurity values obtained before (line 14).

Then we put the total Impurity value into the Impurity list and verify if the new value has an Impurity smaller than the one stored in h^*. If this is true, update the split feature, the threshold, and the best impurity. Of course, at the first iteration, this is always true, then we proceed to all the features and the values. The algorithm returns the split for the binary partition that has the smallest value of Impurity and the feature where we have to split.

4.3 Principal Component-based Split

The last type of split is the Principal Component-based, or **P-PT**, which applies only to datasets with continuous features.

In [15, 26, 36], have been introduced similar versions. In these two works, the main advantages of the PCA division have been applied. PCA identifies a subspace and then splits along the principal eigenvector to reduce the diameter of the data. The features are then partitioned with the medians. At this point, it creates the tree starting from the partitions. Initially, create several random projections and use the same for all the iterations. The best ones will be used to create tree nodes.

Unlike this method, to create the partitions, we have decided to use the components instead of the medians, as the latter is a very simplistic approach.

Algorithm 4 describes in details our approach. The *number of components* is a parameter that must be less than the number of features in the dataset.

We initialize j^*, which is the index of the best feature to split over, t^*, which is the threshold index for that specific feature, and the best BIC value b^*, this will serve to stop the division iteration of the *Train Tree Regressor*.

After we calculate the PCA of each number of components and we put it into a list A (line 2). For each feature in the range from 1 to the chosen number of components, we build a data partitioner, *Train Tree Regressor* that uses *Oblique*

Algorithm 3: I-PT split(X)

```
Input  : X - dataset
Param  : impurity - impurity function, agg_fun - impurity aggregation
         function
Output: f - binary partitioning function, C_1, C_2 - cluster partitions
1  $j^* \leftarrow \infty$; $t^* \leftarrow \infty$; $h^* \leftarrow \infty$;     // init. split feature, threshold, best impurity
2  for $j \in [1, d]$ do                                          // for every feature
3      for $t \in values(X^{(j)})$ do                              // for every value
4          $f \leftarrow$ make_partitioner($cond(X^{(j)}, t)$);        // build data partitioner
5          $X_a, X_b \leftarrow$ split_data($f, X$);                    // make binary partitions
6          if $|X_a| = 0 \vee |X_b| = 0$ then
7              continue;                                          // go next iteration
8          $H \leftarrow \emptyset$;                                         // init. impurity list
9          for $l \in [1, d]$ do                                  // for every target feature
10             if $j = l$ then                                    // if same feature
11                 continue;                                      // go next iteration
12             $h_a \leftarrow$ impurity($X_a^{(l)}$);                     // calculate impurity partition a
13             $h_b \leftarrow$ impurity($X_b^{(l)}$);                     // calculate impurity partition b
14             $h \leftarrow \frac{|X_a|}{|X|} h_a \frac{|X_b|}{|X|} h_b$;                 // calculate total impurity
15             $H \leftarrow H \cup \{h\}$;                                  // update impurity list
16         $h' \leftarrow$ agg_fun($H$);                                // aggregate impurities
17         if $h' < h^*$ then                                      // if better partitioning
18             $j^* \leftarrow j$; $t^* \leftarrow t$; $h^* \leftarrow h$;   // update split feature, thr, best impurity
19 $f \leftarrow$ make_partitioner($cond(X^{(j^*)}, t^*)$);             // build data partitioner
20 $C_1, C_2 \leftarrow$ split_data($f, X$);                              // make binary partitions
21 return $f, C_1, C_2$;
```

House Holder Split for making the splits. If one of the two splits created X_a or X_b is empty, go to the next iteration. This technique was already used in [34] for creating K-means Tree (KM-tree).

In line 8, we calculate the BIC for each of the two partitions created by the Oblique House Holder split. If the BIC obtained is smaller than the BIC stored in b^*, the split should be done. If so, the function, threshold, and corresponding new BIC value are updated.

Then we build the data partitioner with the *Train Tree Regressor*, but, in this case, on the updated feature j. In the end, like in the other split, we make the binary partition, splitting the data on the feature and creating the two clusters C_1 and C_2.

Algorithm 4: P-PTsplit(X)

```
Input  : $X$ - dataset
Param  : $nbr\_components$ - number of components
Output: $f$ - binary partitioning function, $C_1, C_2$ - cluster partitions
1  $j^* \leftarrow \infty$; $t^* \leftarrow \infty$; $b^* \leftarrow \infty$;        // init. split feature, threshold, best BIC
2  $A \leftarrow$ get_PCA($X$, $nbr\_components$));                 // calculate PCA
3  for $j \in [1, nbr\_components]$ do                 // for every principal component
4      $f \leftarrow$ train_tree_regressor($X, A^{(j)}$));              // build data partitioner
5      $X_a, X_b \leftarrow$ split_data($f, X$);                       // make binary partitions
6      if $|X_a| = 0 \vee |X_b| = 0$ then
7          continue;                                          // go next iteration
8      $bic\_score \leftarrow bic(X_a, X_b)$;                      // calculate BIC partitions
9      if $b < b^*$ then                                    // if better partitioning
10         $j^* \leftarrow j$; $t^* \leftarrow t$; $b^* \leftarrow b$;          // update split feature, thr, BIC
11 $f \leftarrow$ train_tree_regressor($X, A^{(j^*)}$));                // build data partitioner
12 $C_1, C_2 \leftarrow$ split_data($f, X$);                             // make binary partitions
13 return $f, C_1, C_2$;
```

4.4 Mixed Data Types

ParTree in the C-PT and I-PT versions, can work with datasets with continuous, categorical, and mixed features. While for the version called P-PT we can only deal with datasets with continuous features.

For the categorical features we decided to follow two types of approaches. The first one consists of the use of a data encoding method that turns categorical features into numerical ones, typically, One-Hot Encoding. The first approach is One-Hot Encoding. It uses a data coding method that transforms the categorical features into numerical ones. The problems with this approach are, *(i)* binary columns with only two values are considered as continuous, *(ii)* splitting thresholds are not very meaningful, thus it may increase the number of features dramatically. It depends on the number of values of each categorical feature.

For these reasons, we also decided to use another data encoder, Label Encoder. This transforms non-numeric features into numeric features.

Chapter 5

Experiments

In this chapter, are present and discuss the experiments performed to observe the performance of our proposals. We first introduce the tested datasets, then we present the competitor methods we will test against ParTree. Then we will present the metrics used to evaluate the results. Before presenting the results in the related section, a further section introduces the setting of our experiments with ParTree.

5.1 Datasets

The different developed methods of ParTree have been tested with datasets coming from different sources. Many of them have been taken from public sources, including Sklearn, Fixed dataset from `https://github.com/deric/clustering-benchmark` and the Real dataset from Kaggle. Table 5.1 displays the collection of chosen datasets together with details on their dimensionalities, features, and actual cluster number (as stated in the heading).

A total of eight *Synthetic* datasets and five *Real* datasets were tested, belonging to a variety of contexts, and selected according to different criteria. As future work, we would like to test ParTree on a much more consistent set of datasets, both for Synthetic and Real datasets. The main objective is the identification of the clusters as close as possible to the real ones provided by the datasets. For all the *Synthetic* and *Real* datasets with only continuous features were applied the Standard Scalar standardization [37]. With this type of standardization, the value distribution is scaled to have a mean of 0 and a standard deviation of 1. It is possible to think of this as either centering the data or removing the mean value.

We selected datasets with all-numeric features or mixed, to test the performance of the algorithms with various types of data. The selected datasets contain only tabular data. From as few as 2 clusters, reflecting a binary categorization, to as many as 15.

For the *Real* titanic dataset with mixed data types a preprocessing was done. We filled the missing values with the mean or the mode, and drop the column that was not useful for our scope: *'Passenger-Id', 'Name', 'Ticket', 'Fare'*. After that,

Dataset	n	m	m_{num}	m_{cat}	*real cluster*
Synthetic					
make-blobs	500	2	2	0	4
aggregation	788	2	2	0	7
tetra	400	3	3	0	4
zelnik6	238	2	2	0	3
make-moons	500	2	2	0	2
2d-10c	2990	2	2	0	9
set2	5000	2	2	0	15
cure-t1	2000	2	2	0	6
Real					
home	492	7	7	0	2
diabetes	768	8	8	0	2
titanic	891	10	5	5	3
iris	150	5	5	0	3
wine	6497	10	10	0	13

Table 5.1: Dataset description: n record, m features, m_{num} numerical features, m_{cat} categorical features, *real cluster* number of real cluster identified by the label.

we created two types of titanic dataset, the *titanic-ohe* with the transformation with One-Hot-Encoding for categorical features and Standard Scalar for numerical features, and the *titanic-lab* with the transformation Label Encoder for categorical features and Standard Scalar for the numerical features.

5.2 Competitors

We decided to compare the different versions of our ParTree algorithm with the state-of-the-art clustering algorithms and two other tree-based developed competitors. The goal is to try to have similar or better general performance w.r.t ParTree, considering different datasets.

As cited in the Section 3, we compare our algorithm to three distinct clustering families. The first one is the Centroid based (see Section 3.1) where we find K-means, BK-means, and K-mode. The second one is the Density-based, (see Section 3.2), where we decide to test only DBSCAN, the main algorithm of this type of clustering. The third one is the Hierarchical clustering (see Section 3.3), where we decide to test the linkage method call 'complete' or 'max', in our scenario H-Max, it creates compact cluster and it is one of the most used methods. We decide also to test the X-means algorithm (see Section 3.1) because used the same stopping criterion of our ParTree and his one of the most relevant comparisons to do. In the end, we test the KM-Tree algorithm from [24], explained in Section 2, and Variance ParTree,

Name	**Library**	**Paramenters**
K-means	*Sklearn*	k = [3, 5, 10, 20]
BK-means	*Sklearn*	k = [3, 5, 10, 20]

K-modes	*Sklearn*	k = [3, 5, 10, 20]
		init = ['Huang', 'Cao' , 'random']

X-means	*PyClustering*	k max = [3, 5, 10, 20]
DBSCAN	*Sklearn*	eps = [0.001, 0.05, 0.1, 0.2] min samples = [4, 5, 10, 20, 100]
H-Max	*Sklearn*	n cluster = [3, 5, 10, 20]
KM-tree	*GitHub*	k = [3, 5, 10, 20] labels as tree leaves = [False,True]
V-PT	*GitHub*	max depth= [3, 4, 5, 6] max number clusters= [3, 5, 10, 20] min samples leaf = [3, 5, 10, 20] min samples split = [3, 5, 10, 20] BIC eps = [0.0, 0.1, 0.3, 0.5, 0.8] max oblique features = [2, 3, 4] max_nbr_values_list = [100] max_nbr_values_cat_list = [10] q score = ['len']

Table 5.2: Competitors info tested for our experimentation

a different type of our algorithm that uses the variance for making the split. These last two are the most similar to our ParTree and those that we will most try to overcome in the final results.

The Table 5.2 shows the libraries used to test the datasets and the parameters involved with their range values. For algorithms that require a *random state* value, we will test them in the range 0 to 9.

5.3 Evaluation Metrics

A critical step involves evaluating our results. Now we introduce the metrics used, as background knowledge, we will divide them into two groups, Supervised and Unsupervised. Many metrics can be used to evaluate the performance of our algorithm, and almost all of them focus on narrowing the difference between the expected and actual clusters. We want our clustering to be as accurate as possible, and in line with the results obtained with state-of-the-art methods.

In our scenario, for evaluating ParTree we decide on different types of metrics. Supervised, because we are creating a tree, and Unsupervised because we are targeting clusters.

5.3.1 Supervised Metrics

- **Rand index,** *RI*, is then given by, if C is a ground truth class assignment an K the clustering, we can define a as the number of pairs of elements that are in the same set in C and the same set in K and, b as the number of pairs of elements that are in different sets in C and different sets in K. The RI is given by:

$$RI = \frac{a+b}{C_2^{n_{samples}}}$$

where $C_2^{n_{samples}}$ is the total number of possible pairs in the dataset.

- **Adjusted Rand index**, *ARI*, since RI does not guarantee the random label assignments will get a value close to zero, we can discount the expected RI $E[RI]$ of random labeling by defining the Adjusted Rand index (ARI) as follows:

$$ARI = \frac{RI - E[RI]}{max(RI) - E[RI}$$

- **Mutual information**, *MI*, assume two label assignments (of the same N objects),U and V, the mutual information between them is calculated by:

$$MI(U,V) = \sum_{i=1}^{|U|}\sum_{j=1}^{|V|} \frac{|U_i \cap V_j|}{N} \log\left(\frac{N(|U_i \cap V_j|)}{|U_i||V_j|}\right)$$

- **Normalized mutual information**, *NMI*, is defined as:

$$NMI(U,V) = \frac{MI(U,V)}{mean(H(U),H(V))}$$

- **Adjusted mutual information**, *AMI*, use the expected value, then be calculated using a similar form to that of the ARand:

$$AMI = \frac{MI - E[MI]}{mean(H(U),H(V)) - E[MI]}$$

- **Homogeneity**, *Hom*, is given by:

$$Hom = 1 - \frac{H(C \mid K)}{H(C)}$$

where $H(C \mid K)$ is the conditional entropy of the classes given the cluster assignments and is given by:

$$H(C \mid K) = -\sum_{c=1}^{|C|}\sum_{k=1}^{|K|} \frac{n_{c,k}}{n} \cdot \log\left(\frac{n_{c,k}}{n_k}\right)$$

and $H(C)$ is the entropy of the classes and is given by:

$$H(C) = -\sum_{c=1}^{|C|} \frac{n_c}{n} \cdot \log\left(\frac{n_c}{n}\right)$$

with n the total number of samples, n_c and n_k the number of samples respectively belonging to class c and cluster k, and finally $n_{c,k}$ the number of samples from class c assigned to cluster k.

- **Completeness**, *Cmp*, is given by:

$$Cmp = 1 - \frac{H(K \mid C)}{H(K)}$$

The conditional entropy of clusters given class $H(K \mid C)$ and the entropy of clusters $H(K)$ are defined in a symmetric manner as conditional entropy of the classes given the cluster assignments and the entropy of the classes.

- **V-measure**, *V-S*, is the harmonic mean of homogeneity and completeness:

$$V\text{-}S = 2 \cdot \frac{Hom \cdot Cmp}{Hom + Cmp}$$

- **Fowlkes mallows**, *F-M*, can be used when the ground truth class assignments of the samples is known. The F-M score is defined as the geometric mean of the pairwise precision and recall:

$$F\text{-}M = \frac{TP}{\sqrt{(TP + FP)(TP + FN)}}$$

5.3.2 Unsupervised Metrics

- **Silhouette coefficient**, *Sil*, is defined for each sample and is composed of two scores: a, the mean distance between a sample and all other points in the same class; b, the mean distance between a sample and all other points in the *next nearest cluster*. The Sil, s, for a single sample is then given as:

$$s = \frac{b - a}{max(a, b)}$$

For a set of samples is given as the mean of the Sil for each sample.

- **Davies Bouldin index**, *D-B*, is defined as the average similarity between each cluster C_i for $i = 1, ..., k$ and its most similar to C_j. In the contexts of this index, similarity is defined as a measure R_{ij} that trades off: s_i, the average distance between each point of cluster i and the centroid of that cluster, also know as cluster diameter and, $d_{i,j}$, the distance between cluster centroids i and j. $R_{i,j}$ is defined as:

$$R_{i,j} = \frac{s_i + s_j}{d_{i,}}$$

Then, the D-B index is defined as:

$$D\text{-}B = \frac{1}{k} \sum_{i=1}^{k} \max_{i \neq j} R_{ij}$$

5.4 Experiments Setting

We experimented with ParTree in all three different versions presented, the C-PT (see Section 2), I-PT (see Section 3), and P-PT (see Section 4 on both Synthetic and Real datasets with some difference in the parameter tuning. Table 5.3 reports the parameters experimented for the algorithm for testing on the Synthetic datasets, while Table 5.4 shows the parameters tested for the Real datasets. The two tables mainly differs in the configuration of C-PT and I-PT. We set two different setting versions named **C-PT-a** and **C-PT-b**, **I-PT-a** and **I-PT-b**, for testing the Real datasets. The versions with 'a' as the appendix, were tested with all Real datasets. The versions with 'b' as an appendix were tested with the dataset *titanic-ohe*.

Also, as mentioned in Section 5.1, the titanic dataset has two versions, *titanic-ohe* and *titanic-lab*, depending on the type of encoding performed. The *titanic-ohe* version was tested with all competitors and all versions of ParTree, while the *titanic-lab* version was tested with ParTree methods with 'a' as an appendix. Table 5.3 and Table 5.4 show respectively, the configuration for the Synthetic dataset and the Real dataset.

Algorithm	Parameters
C-PT	max depth = [3, 4, 5, 6] max number clusters = [3, 5, 10, 20] min samples leaf = [3, 5, 10, 20] min samples split = [3, 5, 10, 20] BIC eps = [0.0, 0.1, 0.3, 0.5, 0.8] max number values = [100] max number values categorical = [10] metric = ['euclidean']
I-PT	max depth = [3, 4, 5, 6] max number clusters = [3, 5, 10, 20] min samples leaf = [3, 5, 10, 20] min samples split= [3, 5, 10, 20] BIC eps= [0.0, 0.1, 0.3, 0.5, 0.8] criteria clf = ['gini', 'entropy', 'me'] criteria reg = ['mape', 'r2'] max number values = [100] max number values categorical = [10] aggregation function = ['mean', 'min', 'max']
P-PT	max depth= [3, 4, 5, 6] max number clusters= [3, 5, 10, 20] min samples leaf = [3, 5, 10, 20] min samples split = [3, 5, 10, 20] BIC eps = [0.0, 0.1, 0.3, 0.5, 0.8] oblique splits = [False, True] max oblique features = [2, 3, 4] max number values = [100] max number values categorical = [10] number of components = [1, 2]

Table 5.3: ParTree configuration for Synthetic datasets

5.5 Experimental Results

This section reports the results obtained. We organize this section into two subsections, one regarding the Synthetic experimentation (see Section 5.5.1), and one regarding the Real experimentation (see Section5.5.2). In both sections, the first two tables reports the performance of our proposal against the selected competitors considering the best performer on a datatset w.r.t. Silhouette and NMI, respectively. After that, using *Orange Evaluation library*, we construct the Critical Difference plot (CD-plot) [13], with this plot is simple to see which configuration is the best. They use the statistical significance of the results obtained with critical differences using the test Nemenyi or Bonferroni-Dunn. Furthermore are based on a specified alpha for the mean ranks and the number of datasets tested. These plots are often used to illustrate how different strategies work on different numbers of datasets.

The last part show some visualization about the distribution of the two metrics cited before, for every dataset and every method, and the relationship with the execution time.

5.5.1 Synthetic Dataset

Table 5.5.1 reports a comparison of the results among the eight competitors and the three ParTree algorithm selecting the best parameters w.r.t the Silhouette metric.

Algorithm	Parameters
C-PT-a	max depth = [3, 4, 5, 6] max number clusters = [3, 5, 10, 20] min samples leaf = [3, 5, 10, 20] min samples split = [3, 5, 10, 20] BIC eps = [0.0, 0.1, 0.3, 0.5, 0.8] max number values = [100] max number values cat = [10] metric = ['euclidean']
C-PT-b	max depth = [3, 4, 5, 6] max number clusters = [3, 5, 10, 20] min samples leaf = [3, 5, 10, 20] min samples split = [3, 5, 10, 20] BIC eps = [0.0, 0.1, 0.3, 0.5, 0.8] max number values = [10] max number values cat = [10] metric = ['jaccard']
I-PT-a	max depth = [3, 4, 5, 6] max number clusters = [3, 5, 10, 20] min samples leaf = [3, 5, 10, 20] min samples split= [3, 5, 10, 20] BIC eps= [0.0, 0.1, 0.3, 0.5, 0.8] criteria classifier = ['gini', 'entropy', 'me'] criteria regressor = ['mape', 'r2'] max number values = [100] max number values categorical = [10] aggregation function = ['mean', 'min', 'max']
I-PT-b	max depth = [3, 4, 5, 6] max number clusters = [3, 5, 10, 20] min samples leaf = [3, 5, 10, 20] min samples split= [3, 5, 10, 20] BIC eps= [0.0, 0.1, 0.3, 0.5, 0.8] criteria classifier = ['gini', 'entropy', 'me'] criteria regressor = ['mape', 'r2'] max number values = [100] max number values categorical = [10] aggregation function = ['mean', 'min', 'max']
P-PT	max depth= [3, 4, 5, 6] max number clusters= [3, 5, 10, 20] min samples leaf = [3, 5, 10, 20] min samples split = [3, 5, 10, 20] BIC eps = [0.0, 0.1, 0.3, 0.5, 0.8] oblique splits = [False, True] max oblique features = [2, 3, 4] max number values = [100] max number values categorical = [10] number of components = [1, 2]

Table 5.4: ParTree configuration for Real datasets

Starting from analyzing the timing, the faster is BK-means, but we can observe that almost all the algorithms are fast except K-mode. Concerning the RI score, the best one is X-means but our C-PT is the second one, followed by P-PT. Instead, I-PT, in general, does not perform very well. Other metrics like ARI, MI, AMI, NMI, Hom, Cmp, V-S and F-M have as best performer DBSCAN.

ParTree for C-PT and P-PT, performs slightly lower but remains in line with the others competitors. For the Sil, X-means has the best results but is immediately followed by C-PT and P-PT. K-modes and I-PT, have the worst results. For the D-B, C-PT and P-PT are respectively the first and the second, confirming that

	Time seconds ↓		**RI ↑**		**ARI ↑**		**MI ↑**		**AMI ↑**		**NMI ↑**	
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
K-means	0.087	0.060	0.877	0.133	0.700	0.272	*1.398*	*0.660*	0.798	0.178	0.799	0.177
K-modes	1.323	2.152	0.268	0.134	−0.004	0.011	0.015	0.026	−0.004	0.010	0.019	0.033
BK-means	**0.031**	**0.016**	0.880	0.132	0.699	0.267	1.401	0.631	0.790	0.176	0.792	0.175
X-means	0.036	0.033	**0.908**	**0.095**	*0.761*	*0.205*	1.273	0.608	0.802	0.206	0.803	0.206
H-Max	0.296	0.526	0.869	0.132	0.664	0.287	1.357	0.607	0.783	0.173	0.785	0.172
DBSCAN	*0.033*	*0.040*	0.891	0.261	**0.942**	**0.067**	**1.479**	**0.597**	**0.955**	**0.044**	**0.955**	**0.044**
KM-tree	0.074	0.042	0.817	0.066	0.552	0.162	0.941	0.349	0.674	0.174	0.675	0.174
V-PT	0.357	0.314	0.852	0.136	0.625	0.261	1.217	0.613	0.718	0.210	0.720	0.209
C-PT	0.627	0.460	*0.896*	*0.136*	0.740	0.271	1.377	0.588	*0.808*	*0.192*	*0.809*	*0.191*
I-PT	0.982	0.758	0.609	0.287	0.388	0.273	0.618	0.521	0.445	0.306	0.449	0.303
P-PT	0.095	0.083	0.884	0.134	0.716	0.268	1.332	0.606	0.791	0.179	0.792	0.178

	Hom ↑		**Cmp ↑**		**V-S ↑**		**F-M ↑**		**Sil ↑**		**D-B ↓**	
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
K-means	*0.892*	*0.173*	0.777	0.229	0.799	0.177	0.790	0.186	0.587	0.124	0.657	0.171
K-modes	0.013	0.025	0.224	0.148	0.019	0.033	0.482	0.131	−0.169	0.103	3.074	2.770
BK-means	*0.892*	*0.138*	0.749	0.216	0.792	0.175	0.784	0.182	0.561	0.133	0.694	0.196
X-means	0.791	0.217	*0.828*	*0.219*	0.803	0.206	*0.839*	*0.133*	**0.598**	**0.136**	0.605	0.204
H-Max	0.875	0.149	0.778	0.261	0.785	0.172	0.760	0.204	0.551	0.149	0.668	0.249
DBSCAN	**0.946**	**0.070**	**0.972**	**0.041**	**0.955**	**0.044**	**0.897**	**0.170**	0.571	0.179	1.716	1.661
KM-tree	0.615	0.181	0.799	0.248	0.675	0.174	0.700	0.121	0.519	0.139	0.750	0.234
V-PT	0.780	0.199	0.717	0.249	0.720	0.209	0.727	0.179	0.515	0.172	0.830	0.358
C-PT	0.881	0.097	0.782	0.239	*0.809*	*0.191*	0.808	0.189	*0.593*	*0.142*	**0.596**	**0.186**
I-PT	0.422	0.289	0.596	0.272	0.449	0.303	0.586	0.178	0.347	0.159	1.038	0.464
P-PT	0.860	0.150	0.783	0.233	0.792	0.178	0.795	0.185	0.590	0.132	*0.602*	*0.180*

Table 5.5: Synthetic dataset: best performer selected by Sil. The best competitor is in bold, and the second is in italic.

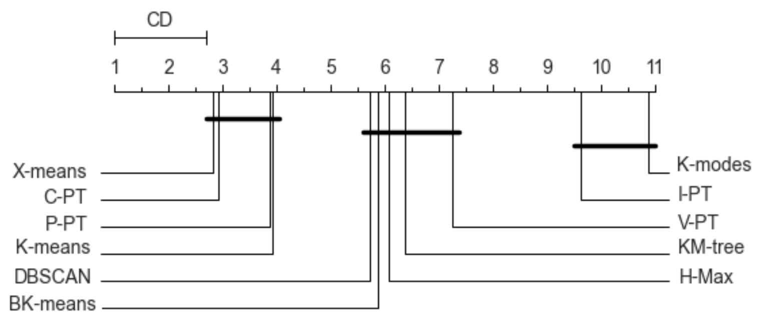


Figure 5.1: CD-plot for methods comparison w.r.t Sil for Synthetic datasets.

ParTree is in line with other competitors.

Table 5.5.1 reports a comparison of the results among the eight competitors and the three ParTree algorithm by selecting the best parameters w.r.t. Normal Mutual Information score, NMI. The best results for RI and ARI scores are obtained by X-means, H-max, and DBSCAN, but our methods (excluding I-PT) have good results in line with these baseline methods. For the MI score, P-PT is the first one, followed by C-PT, also for the AMI and NMI is quite immediately observed that also our C-PT e P-PT works well. P-PT has the best value for Hom, followed by C-PT. This confirms that our method ParTree is in line with the other methods, and, in some cases, it excels when tested under certain metrics.

As can be visualized from the CD-Plot in Figure 5.1, the first method is X-means, like the results in Table 5.5.1, followed by K-means, and after we have our version of P-PT and C-PT. It demonstrates that these two methods are in line with the others

	Time seconds ↓		**RI ↑**		**ARI ↑**		**MI ↑**		**AMI ↑**		**NMI ↑**	
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
K-means	0.088	0.058	0.896	0.129	0.746	0.249	1.431	0.625	0.819	0.169	0.820	0.167
K-modes	1.112	1.130	0.317	0.139	−0.001	0.012	0.055	0.044	0.009	0.040	0.059	0.039
BK-means	**0.031**	**0.016**	0.888	0.131	0.713	0.268	1.423	0.603	0.799	0.170	0.801	0.169
X-means	0.035	0.034	**0.919**	**0.090**	*0.783*	*0.198*	1.322	0.593	0.820	0.201	0.822	0.200
H-Max	0.293	0.528	*0.917*	*0.101*	0.781	0.206	1.408	0.596	*0.828*	*0.162*	*0.830*	*0.161*
DBSCAN	*0.032*	*0.040*	0.897	0.247	**0.827**	**0.340**	1.301	0.747	**0.846**	**0.326**	**0.849**	**0.318**
KM-tree	0.101	0.092	0.853	0.096	0.627	0.162	1.175	0.519	0.758	0.158	0.760	0.157
V-PT	0.421	0.252	0.876	0.119	0.654	0.245	1.338	0.545	0.753	0.182	0.756	0.180
C-PT	0.694	0.496	0.902	0.136	0.758	0.271	*1.455*	*0.602*	0.825	0.186	0.827	0.185
I-PT	1.189	0.814	0.710	0.175	0.393	0.238	0.839	0.462	0.546	0.204	0.552	0.201
P-PT	0.105	0.106	0.904	0.120	0.750	0.246	**1.464**	**0.596**	0.825	0.164	0.826	0.163
	Hom ↑		**Cmp ↑**		**V-S ↑**		**F-M ↑**		**Sil ↑**		**D-B ↓**	
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
K-means	0.916	0.101	0.774	0.208	0.820	0.167	0.815	0.169	*0.579*	*0.126*	0.674	0.184
K-modes	0.041	0.030	0.199	0.143	0.059	0.039	0.457	0.134	−0.678	0.213	8.316	12.047
BK-means	0.913	0.092	0.745	0.217	0.801	0.169	0.791	0.184	0.560	0.131	0.697	0.192
X-means	0.824	0.198	*0.825*	*0.215*	0.822	0.200	*0.852*	*0.131*	**0.590**	**0.130**	**0.588**	**0.206**
H-Max	0.897	0.092	0.788	0.204	*0.830*	*0.161*	0.842	0.135	0.511	0.171	0.729	0.265
DBSCAN	0.833	0.327	**0.885**	**0.260**	**0.849**	**0.318**	**0.896**	**0.177**	0.436	0.404	1.135	0.401
KM-tree	0.743	0.159	0.812	0.229	0.760	0.157	0.749	0.089	0.433	0.188	0.930	0.344
V-PT	0.860	0.071	0.703	0.236	0.756	0.180	0.740	0.171	0.484	0.216	0.770	0.289
C-PT	*0.933*	*0.078*	0.776	0.236	0.827	0.185	0.824	0.187	0.567	0.149	*0.596*	*0.186*
I-PT	0.559	0.229	0.633	0.227	0.552	0.201	0.588	0.171	0.157	0.344	1.258	0.628
P-PT	**0.939**	**0.062**	0.762	0.213	0.826	0.163	0.820	0.163	0.547	0.168	0.681	0.242

Table 5.6: Synthetic dataset: best performer selected by NMI. The best competitor is in bold, and the second is in italic.

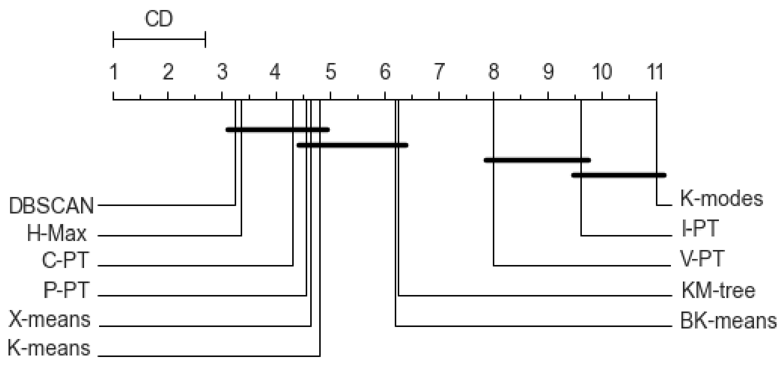


Figure 5.2: CD-plot for methods comparison w.r.t NMI for Synthetic datasets.

and other state-of-the-art methods. In the middle of the plot, we have all the other methods, except for I-PT and K-modes which are the last. The second CD-Plot 5.2, is similar to the first one, the best method is DBSCAN, followed by H-Max. Also, in this case, our algorithms C-PT and P-PT are the third and the fourth. Then we have all the other methods of state-of-the-art. Also here, K-Modes and I-PT are the last. This demonstrates that having a good result in some metrics does not means having a good performance along all the dataset tested.

Figure 5.3 and Figure 5.4 shows the scatter plots visualizing the results on the previous Table. The first scatter plot in Figure 5.3 reports the aggregated results, while the scatter plots in Figure 5.4 are zoom-in of Figure 5.3 partitioning the methods in analysis. With this second plot, we can better visualize the difference in how the methods performed concerning the metrics taken into account, NMI and Sil.

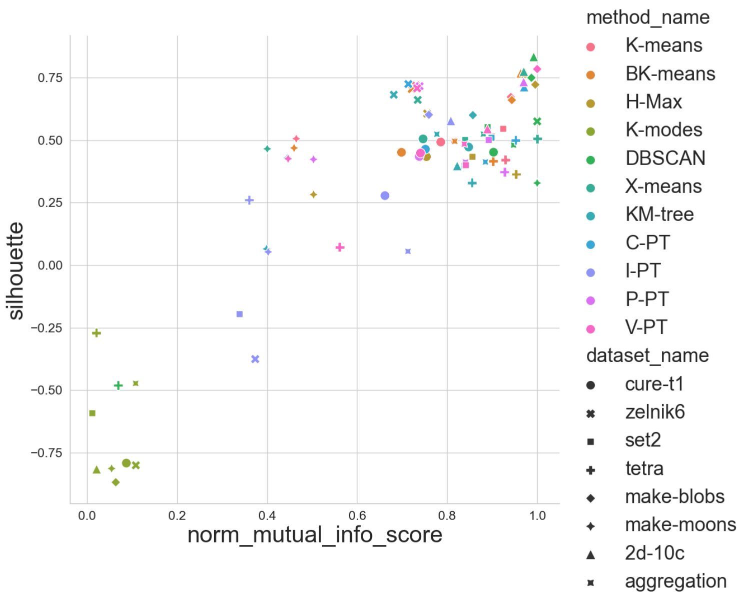


Figure 5.3: NMI w.r.t. Sil, for all the Synthetic datasets and methods, tested.

From Figure 5.3 and Figure 5.4 is immediate to observe that K-means, BK-means, X-means, and H-Max, have mostly the same results for all the datasets. Our algorithms C-PT and P-PT, are in line with previews cited methods, in contrast, I-PT has not so good results, especially for dataset *zelnik6* and *make-moons*. DBSCAN has optimal results for datasets *make-blobs* and *2d-10c* but does not perform well for dataset *tetra*. K-modes instead have the worst results of all the tests, for all the Synthetic datasets. These two visualizations continue to hold up our argument, PARTREE is in line with the state-of-the-art methods.

The last two figure of this subsection, Figure 5.5 and Figure 5.6, shows the relationship between the execution time of the algorithm for each dataset tested, and the relative metric in analysis (Sil or NMI). Higher values of Sil, are given by a lower execution time. On the right bottom of the plot we can find many of the competitor methods, and, as highlighted in the previews Table 5.5.1 and CD-plot 5.1, the best results are obtained by DBSCAN. Our methods, C-PT, and P-PT defend themselves well having results in line with competitors. I-PT on the other hand is slower but still manages to have good results. The same cannot be said of K-modes, it is the slowest method with the poorest results. The NMI scatter plot,in Figure 5.6, in line with the previews, has similar results, also in this case K-modes is the worst and the slower, especially with the dataset *cure-t1*. P-PT similarly has poor results for *set2* and *tetra* but works well with *2d-10c* and *aggregation* although it is not as fast as our versions C-PT and P-PT. P-PT has the best results, especially applied to the datasets *make-blobs* and *2d-10c*. For this last dataset, also C-PT performs very well but has a lower execution time compared to the others methods.

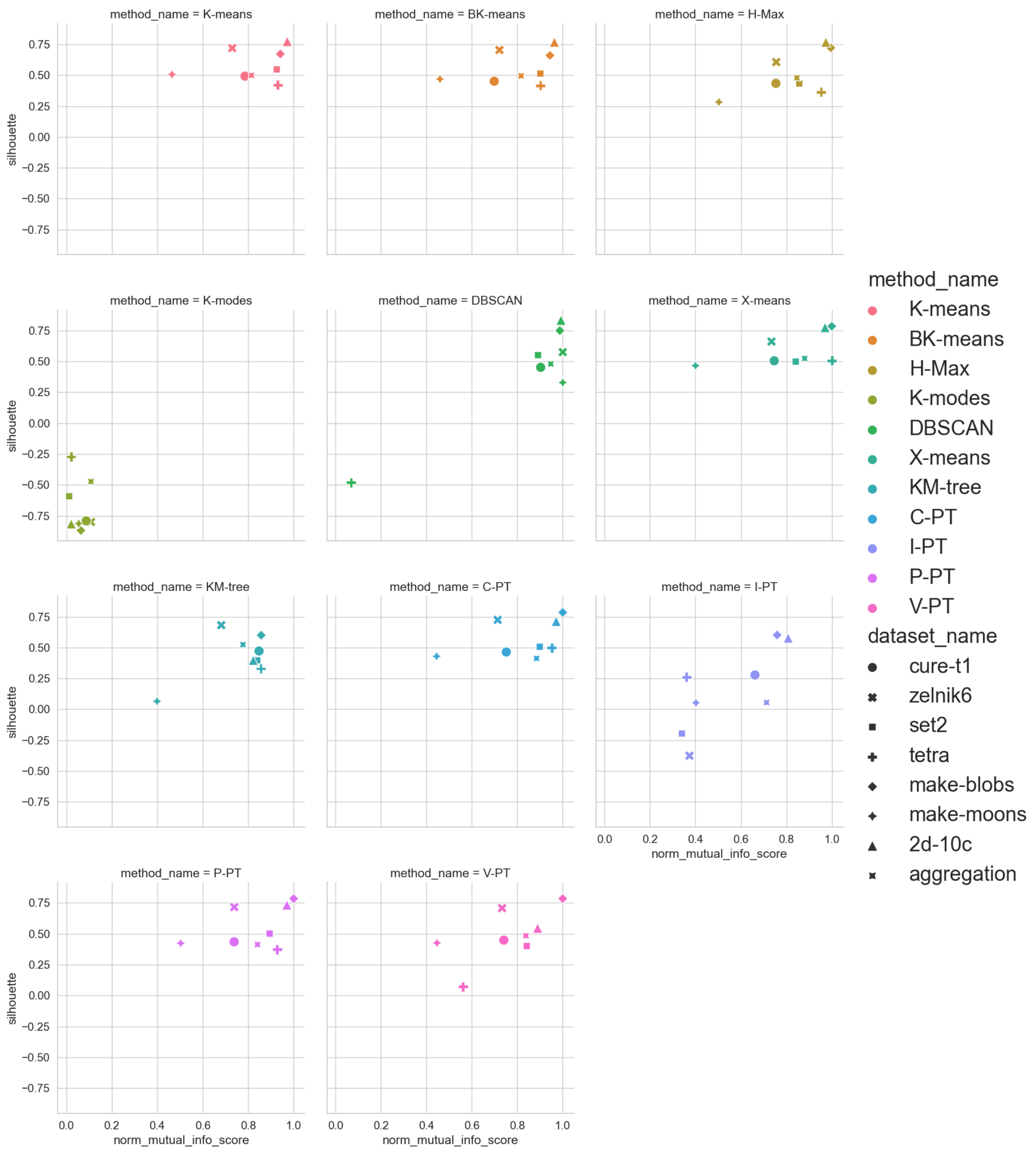


Figure 5.4: NMI w.r.t. Sil, zoom-in, for all the Synthetic datasets and methods, tested.

5.5.2 Real Dataset

Table 5.5.2 reports the comparison of the results among the eight competitors, with the setting reported in Table 5.2, and the five ParTree algorithm shown in Table 5.4 selecting the best parameters w.r.t Silhouette metric. K-means results first in several metrics, like RI, ARI, MI, AMI, NMI, Hom and V-S, and is always followed by BK-means. Looking at Sil, I-PT-a, and P-PT have the higher value, followed by C-PT-a. This means that in terms of Sil, ParTree works well compared with the competitors. For versions of ParTree, C-PT-b, and I-PT-b, is not possible

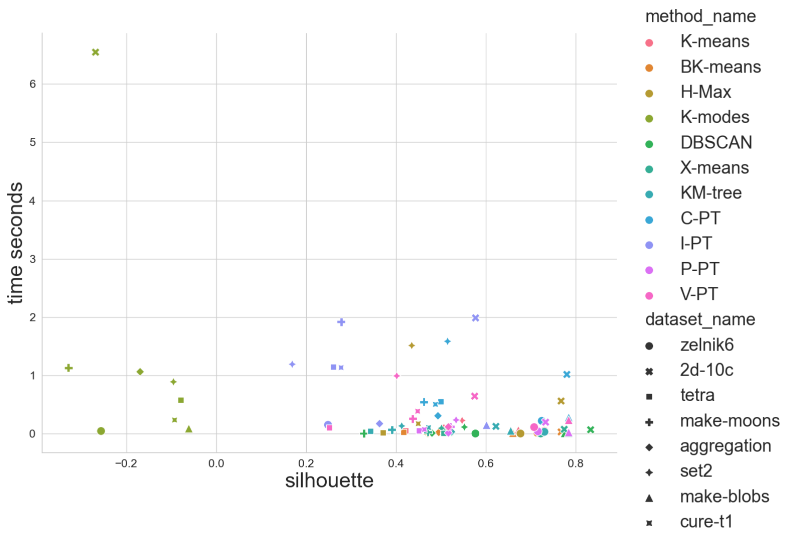


Figure 5.5: Sil w.r.t. time seconds of execution, for all the methods and Synthetic datasets.

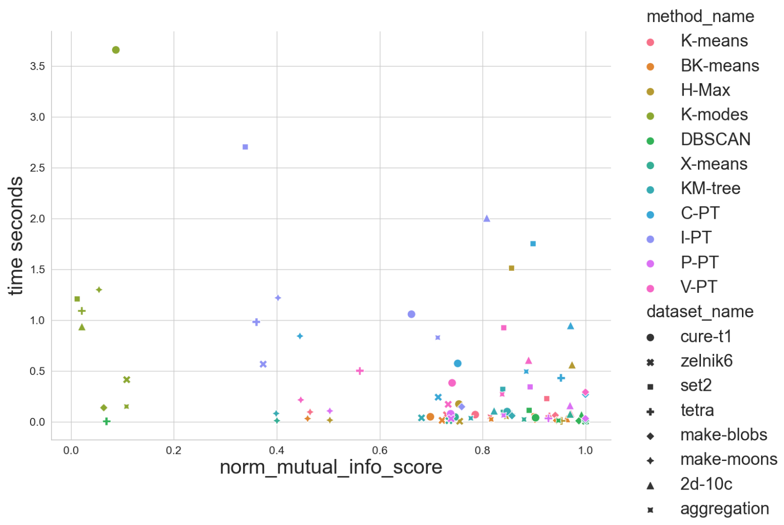


Figure 5.6: NMI w.r.t. time seconds of execution, for all the methods and Synthetic datasets.

to calculate the std, this is because we are testing a single dataset, *titanic-ohe*, furthermore the two settings have the lowest results. In contrast to this the other version C-PT-a, I-PT-a, and P-PT has a higher value compared to all the other methods. Looking at the time, the faster algorithm is KM-tree, instead, our five configurations of ParTree are the slower, except P-PT, that is in line with the competitors.

Table 5.5.2 reports a comparison of the results among the eight competitors and the three ParTree algorithm by selecting the best parameters w.r.t. Normal Mutual Information score, NMI.

Looking at the results, P-PT has the best value of the whole table for several

	Time seconds ↓		**RI ↑**		**ARI ↑**		**MI ↑**		**AMI ↑**		**NMI ↑**	
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
K-means	0.302	0.331	**0.622**	**0.136**	**0.230**	**0.227**	**0.298**	**0.306**	**0.264**	**0.253**	**0.266**	**0.254**
K-modes	3.889	7.380	0.549	0.058	0.048	0.071	0.060	0.099	0.057	0.094	0.061	0.098
BK-means	0.142	0.166	*0.621*	*0.140*	*0.218*	*0.247*	*0.283*	*0.327*	*0.242*	*0.273*	*0.244*	*0.274*
X-means	*0.071*	*0.071*	0.609	0.142	0.204	0.239	0.273	0.321	0.230	0.271	0.232	0.271
H-Max	0.651	1.349	0.601	0.117	0.139	0.249	0.218	0.295	0.199	0.266	0.203	0.266
DBSCAN	0.096	0.117	0.445	0.159	0.006	0.002	0.036	0.037	0.046	0.032	0.059	0.047
KM-tree	**0.064**	**0.027**	0.620	0.124	0.207	0.218	0.260	0.289	0.233	0.247	0.235	0.248
V-PT	1.310	1.441	0.566	0.093	0.130	0.178	0.149	0.185	0.152	0.209	0.154	0.209
C-PT-a	15.348	22.831	0.585	0.109	0.145	0.213	0.183	0.256	0.183	0.282	0.185	0.281
C-PT-b	22.454	0.000	0.503	0.000	0.043	0.000	0.013	0.000	0.015	0.000	0.017	0.000
I-PT-a	32.453	57.662	0.562	0.107	0.136	0.215	0.159	0.253	0.171	0.284	0.174	0.284
I-PT-b	81.896	0.000	0.476	0.000	0.046	0.000	0.019	0.000	0.015	0.000	0.017	0.000
P-PT	0.283	0.393	0.598	0.118	0.150	0.245	0.212	0.279	0.207	0.310	0.209	0.309

	Hom ↑		**Cmp ↑**		**V-S ↑**		**F-M ↑**		**Sil ↑**		**D-B ↓**	
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
K-means	**0.263**	**0.241**	*0.284*	*0.277*	**0.266**	**0.254**	0.543	0.128	0.239	0.184	2.023	1.542
K-modes	0.061	0.089	0.064	0.109	0.061	0.098	0.448	0.121	0.025	0.103	26.038	42.429
BK-means	*0.238*	*0.262*	0.263	0.298	*0.244*	*0.274*	0.546	0.124	0.225	0.194	1.595	0.782
X-means	0.229	0.257	0.251	0.297	0.232	0.271	0.524	0.123	0.279	0.152	1.186	0.466
H-Max	0.182	0.257	0.235	0.277	0.203	0.266	*0.625*	*0.142*	0.310	0.219	2.226	1.945
DBSCAN	0.035	0.030	**0.680**	**0.440**	0.059	0.047	**0.643**	**0.137**	0.226	0.208	**0.691**	**0.012**
KM-tree	0.230	0.242	0.250	0.261	0.235	0.248	0.550	0.116	0.233	0.166	2.196	1.607
V-PT	0.143	0.163	0.187	0.279	0.154	0.209	0.481	0.133	0.209	0.236	2.056	1.524
C-PT-a	0.157	0.217	0.240	0.390	0.185	0.281	0.540	0.150	0.280	0.378	1.753	1.390
C-PT-b	0.022	0.000	0.013	0.000	0.017	0.000	0.515	0.000	0.217	0.000	1.816	0.000
I-PT-a	0.134	0.223	0.258	0.388	0.174	0.284	0.617	0.110	**0.454**	**0.354**	*1.029*	*0.495*
I-PT-b	0.032	0.000	0.012	0.000	0.017	0.000	0.426	0.000	0.244	0.000	1.955	0.000
P-PT	0.176	0.238	0.279	0.425	0.209	0.309	0.536	0.194	*0.317*	*0.246*	1.371	0.981

Table 5.7: Real dataset: best performer selected by Sil. The best competitor is in bold, and the second is in italic.

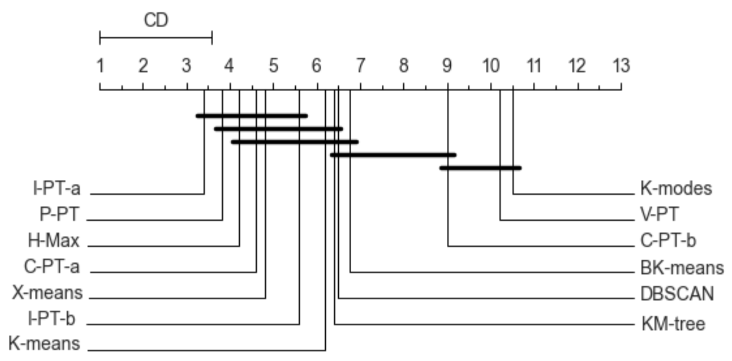


Figure 5.7: CD-plot for methods comparison w.r.t Sil for Real datasets.

metrics, especially for the Supervised ones, including RI, ARI, MI, AMI, NMI, V-S, and F-M. For Hom we have our I-PT-b as the best one, followed by P-PT, instead for the Sil the first method is K-means followed by X-means. The time of execution of our ParTree algorithm is slower than the other competitors, except for P-PT, which also, in this case, has good results. This confirms that our method ParTree in some cases can be even better than the state-of-the-art algorithms.

The CD-plot in Figure 5.7 shows that especially our versions of ParTree I-PT-a and P-PT stand out for having the best results, coming in first and second position. Also, C-PT-a has good results because is in the fourth position, then we have X-means and after our I-PT-b. All the other competitors are in the center of the plot,

	Time seconds $\downarrow$		**RI** $\uparrow$		**ARI** $\uparrow$		**MI** $\uparrow$		**AMI** $\uparrow$		**NMI** $\uparrow$	
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
K-means	0.320	0.319	*0.613*	*0.150*	*0.228*	*0.229*	0.311	0.291	0.270	0.246	0.273	0.246
K-modes	7.170	9.106	0.576	0.132	0.048	0.047	0.237	0.238	0.114	0.105	0.132	0.124
BK-means	0.426	0.725	0.603	0.159	0.210	0.252	0.344	0.287	0.266	0.256	0.270	0.255
X-means	*0.111*	*0.131*	0.610	0.153	0.221	0.234	*0.346*	*0.279*	*0.273*	*0.242*	*0.277*	*0.241*
H-Max	0.523	1.067	0.608	0.145	0.175	0.226	0.290	0.275	0.236	0.244	0.242	0.243
DBSCAN	**0.056**	**0.091**	0.433	0.121	0.010	0.011	0.099	0.080	0.091	0.071	0.111	0.064
KM-tree	0.133	0.130	0.597	0.150	0.199	0.225	0.284	0.266	0.242	0.239	0.245	0.238
V-PT	3.212	3.904	0.590	0.161	0.126	0.203	0.262	0.220	0.186	0.221	0.193	0.220
C-PT-a	65.079	121.703	0.586	0.132	0.162	0.204	0.274	0.221	0.222	0.262	0.231	0.258
C-PT-b	32.679	0.000	0.442	0.000	0.040	0.000	0.123	0.000	0.071	0.000	0.077	0.000
I-PT-a	31.169	39.490	0.617	0.181	0.218	0.317	0.290	0.343	0.235	0.304	0.240	0.302
I-PT-b	74.720	0.000	0.535	0.000	0.151	0.000	0.259	0.000	0.257	0.000	0.258	0.000
P-PT	0.253	0.303	**0.669**	**0.174**	**0.306**	**0.335**	**0.380**	**0.350**	**0.335**	**0.306**	**0.338**	**0.305**

	Hom $\uparrow$		**Cmp** $\uparrow$		**V-S** $\uparrow$		**F-M** $\uparrow$		**Sil** $\uparrow$		**D-B** $\downarrow$	
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
K-means	0.286	0.217	0.288	0.273	0.273	0.246	*0.505*	*0.183*	**0.231**	**0.181**	2.042	1.531
K-modes	0.278	0.234	0.088	0.086	0.132	0.124	0.251	0.076	−0.292	0.176	27.407	26.662
BK-means	0.331	0.230	0.275	0.288	0.270	0.255	0.445	0.207	0.151	0.271	7.261	13.181
X-means	0.337	0.237	0.281	0.273	*0.277*	*0.241*	0.459	0.214	*0.158*	*0.275*	9.035	17.279
H-Max	0.267	0.211	0.234	0.265	0.242	0.243	0.483	0.167	0.070	0.300	6.175	6.752
DBSCAN	0.133	0.158	**0.340**	**0.373**	0.111	0.064	0.574	0.125	−0.371	0.463	**1.654**	**0.840**
KM-tree	0.268	0.209	0.255	0.256	0.245	0.238	0.456	0.193	0.149	0.224	2.538	1.757
V-PT	0.267	0.183	0.173	0.237	0.193	0.220	0.328	0.194	−0.021	0.287	5.589	4.870
C-PT-a	0.284	0.189	0.256	0.374	0.231	0.258	0.441	0.208	0.144	0.454	4.256	4.720
C-PT-b	0.209	0.000	0.047	0.000	0.077	0.000	0.288	0.000	0.117	0.000	2.857	0.000
I-PT-a	0.271	0.289	0.226	0.309	0.240	0.302	0.501	0.225	−0.100	0.348	3.905	2.049
I-PT-b	**0.440**	**0.000**	0.183	0.000	0.258	0.000	0.503	0.000	0.093	0.000	*2.007*	*0.000*
P-PT	*0.372*	*0.281*	*0.323*	*0.318*	**0.338**	**0.305**	**0.581**	**0.216**	0.110	0.259	3.122	2.984

Table 5.8: Real dataset: best performer selected by NMI. The best competitor is in bold, and the second is in italic.

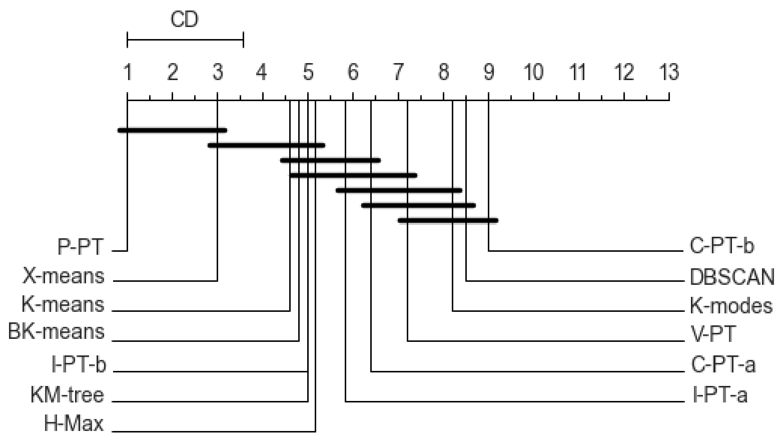


Figure 5.8: CD-plot for methods comparison w.r.t NMI for Real datasets.

excluding our C-PT-b, which is similar to the worst methods V-PT and K-modes.

This shows that our method is in some cases superior, to the competitors taken into account w.r.t Sil metric. The last CD-plot, in Figure 5.8, shows method comparison w.r.t NMI, here our algorithm P-PT is the best. Then we have methods state-of-the-art and in the center of the plot our versions I-PT-b, I-PT-a, and C-PT-a. Instead, our method C-PT-b is in the last position. With this overview, we can say that ParTree works very well in some cases, but can also not be the best one with a certain set of parameters.

Figure 5.9 and Figure 5.10 show the scatter plots visualizing the results in the

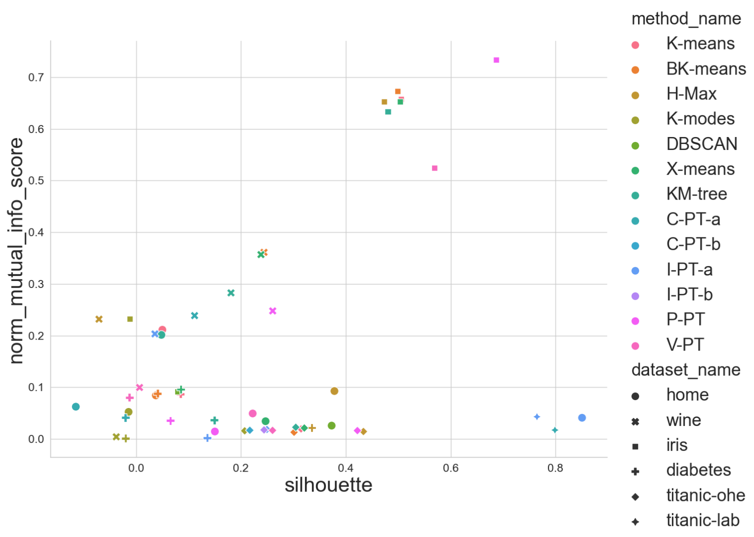


Figure 5.9: NMI w.r.t. Sil, for all the Real datasets and methods, tested.

previous Table. The first scatter plot in Figure 5.9 reports the aggregated results, while the scatter plots in Figure 5.10 are zoom-in of Figure 5.9 partitioning the methods in analysis. With this second plot, we can better visualize the difference in how the methods performed concerning the metrics taken into account, Sil and NMI. The *iris* dataset, has always the best results in all the methods, especially for our version of P-PT, C-PT-a, and I-PT-a has the highest results. For the dataset *home*, we have a high Sil in our I-PT-a, while in all the other methods it has poor results. Datasets *wine* and *diabetes* have more or less always the same values in all tested methods. For the special configuration of titanic, *titanic-lab*, tested with C-PT-a and I-PT-a has a high Sil but a low NMI, instead *titanic-ohe*, for both the metrics, has not so good results in all the methods. It is interesting to observe that for DBSCAN, the parameters taken into account do not allow to have relevant results for all tested datasets. Only for the *iris* and *home* datasets, we have plotted values, and they are also low. These two graphics continue to support our claim, ParTree is in line with other methods tested.

The last two plots, Figure 5.11 and Figure 5.12 show the correlation between the execution time of the algorithm for each dataset tested, and the relative metric in analysis (Sil or NMI). In Figure 5.11 we can observe that there is a higher Sil obtained by our methods I-PT-a on the *home* dataset in a little execution time. Also, C-PT-b and P-PT have a higher value of Sil in a little amount of time on different datasets. The *titanic-ohe*, tested with I-PT-a and I-PT-b is slower compared with the other methods but has a Sil value in line with them. Figure 5.12, is interesting to observe that the *iris* dataset has a higher value of NMI in a little amount of time, with all the methods, instead, all the other datasets are in the lower-left corner, with a value of NMI < 0.4. In this case, the *titanic-ohe* dataset tested with I-PT-b, has a higher NMI value compared with all other methods. Looking at the results with

C-PT-a and I-PT-a, they are slower than the other methods, but the value of NMI is in line with them.

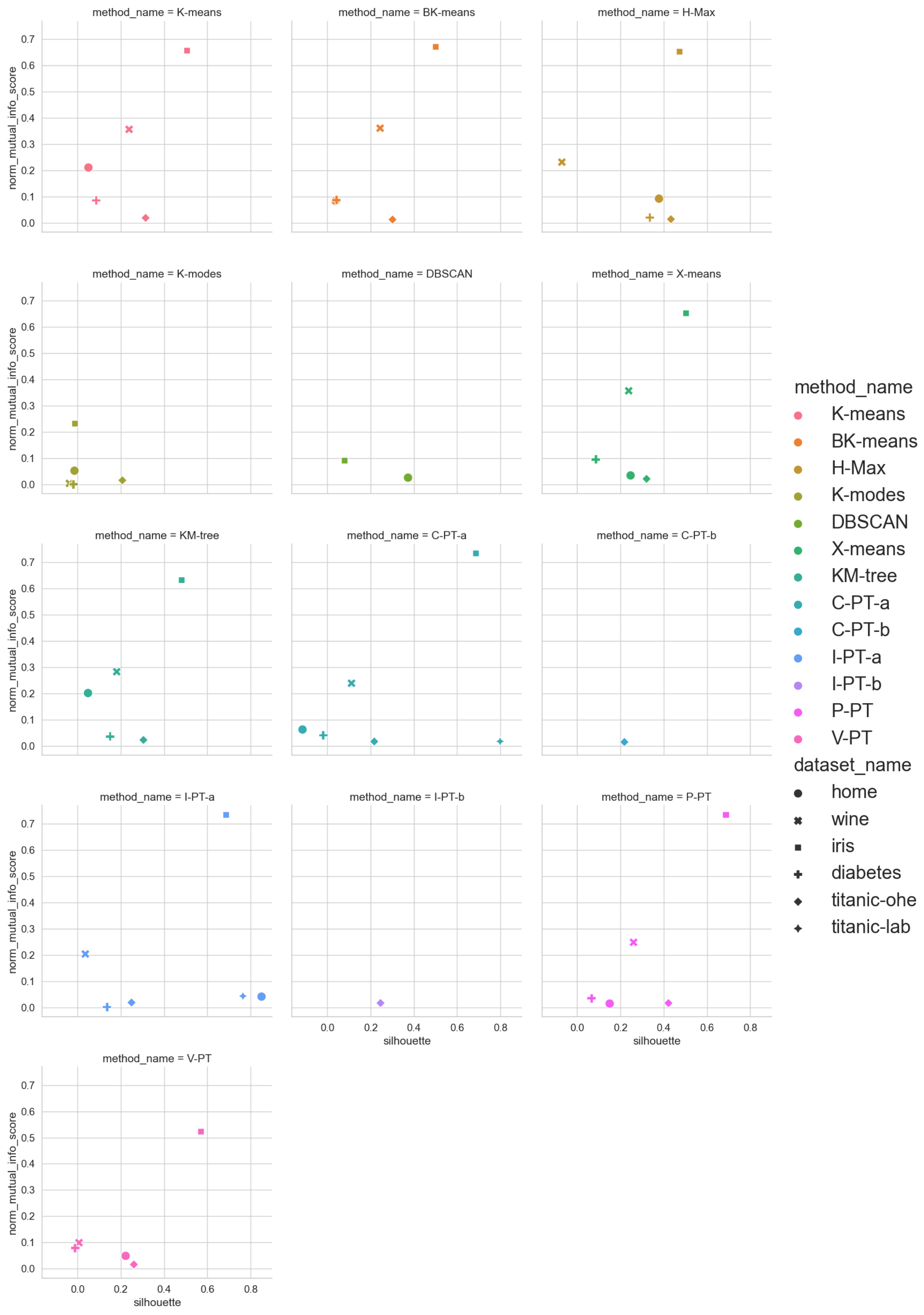


Figure 5.10: NMI w.r.t. Sil, zoom-in, for all the Real datasets and methods, tested.

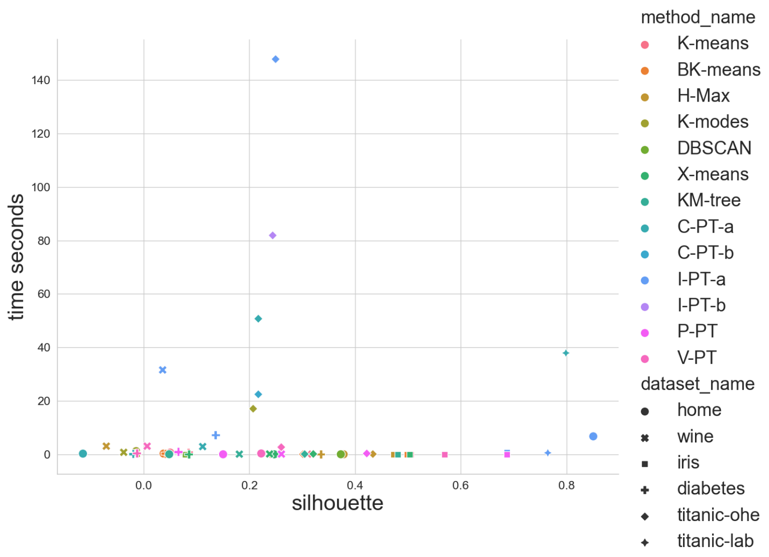


Figure 5.11: Sil w.r.t. time seconds of execution, for all the methods and Real datasets.

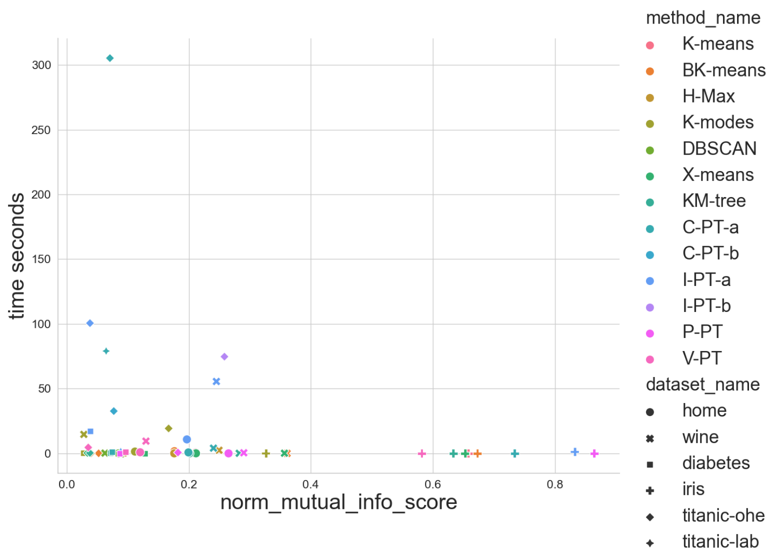


Figure 5.12: NMI w.r.t. time seconds of execution, for all the methods and Real datasets.

Chapter 6

Case Study

In this Chapter, we describe a case study carried out with human participants to evaluate the value of the interpretable clustering trees. The objectives that prompted us to create this case study are different. First of all, being able to understand if with our visualization we can reduce the cognitive workload of the end users who will use ParTree. Another important goal is to understand if the user actually makes fewer mistakes using our Tree. Finally, we want to evaluate its usability both internally, and therefore by comparing different levels of difficulty, and externally, by comparing the Trees with other plots. Theoretically, our visualization is better, this is because, on a cognitive level, performing precise steps helps the brain identify the final cluster. The purpose of the case study is to demonstrate this.

We first describe the problem faced by the participants with an overview of the scenario presented. In a separate section, we described the cognitive workload and usability questionnaires that the users have to compile after each test. Then we show examples of Google Forms that have been presented to users. The final section shows the results obtained.

6.1 Scenario Overview

In this section, we define the datasets that we have decided to use, which types of graphs will be submitted to users, and in which modality. We decided to use the real *home* and *diabetes* datasets. The dataset *home* has been reduced to five features, and defined as the 'easy' case, while *diabetes*, has eight features and is defined as the 'difficult' case. We test the ParTree version called C-PT with both datasets, setting as parameters the best combination we obtained from our experiments. We decided to compare it to two of the most popular clustering visualizations. A Parallel plot with clusters, a result of K-means, and a Dendrogram, a result of H-max. The *home* dataset was tested with K = 5, while the *diabetes* dataset was with K = 10, both with a random state = 0. Table 6.1 shows the combination of parameters chosen for the creation of the Tree, the Parallel plot, and the Dendrogram inserted in the questionnaire.

	C-PT-a	**K-means**	**H-Max**
home	max depth = 3 max number clusters = 5 min samples leaf = 3 min samples split = 10 max number values = 100 max number values categorical = 10 bic eps = 0.0 metric = 'euclidean'	K = 5	K = 5
diabetes	max depth = 5 max number cluster = 10 min samples leaf = 3 min samples split = 10 max number values = 100 max number values categorical = 10 bic eps = 0.0 metric = 'euclidean'	K = 10	K = 10

Table 6.1: Parameter tuning for each method and each dataset that will be presented in the questionnaire.

Figure 6.1 and Figure 6.2 show the Tree and the Parallel plot for the *home* dataset. In Figure 6.4 and Figure 6.5 we have the Tree and the Parallel plot for the *diabetes* dataset. For the Dendrogram in Figure 6.3 and Figure 6.6, to make it easier for users to read, we have also decided to enter the values of the centroids for each feature. For making our experiments, we select three records from the dataset *home*, and three from the dataset *diabetes*. Table 6.2, for dataset *home* and Table 6.3 for dataset *diabetes*, show the selected records with the relative ground truth.

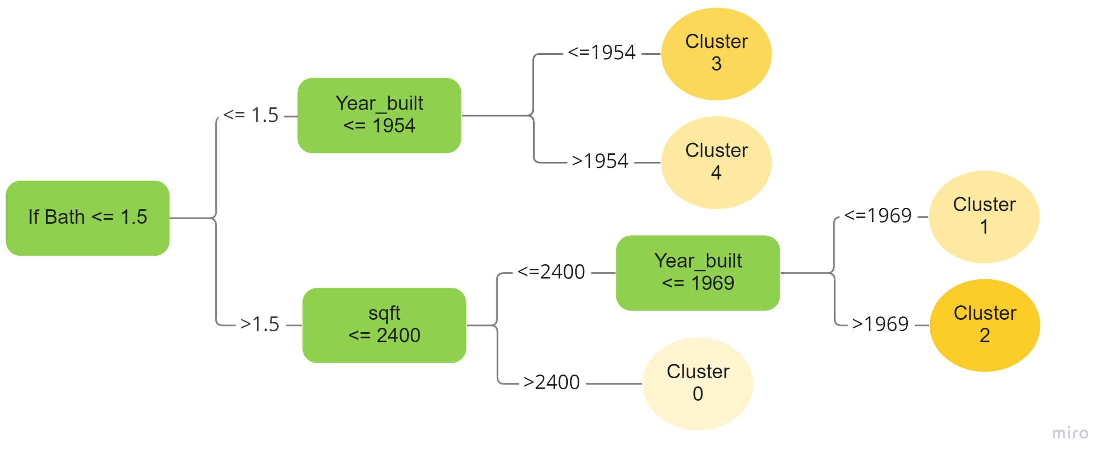


Figure 6.1: Tree for dataset *home*

	Beds	*Bath*	*Price*	*Year Built*	*Sqft*	*Cluster*
Tree	2	2	1350000	1900	2150	1
PP	2	1	999000	1960	1000	0
Dend	4	4	5500000	1923	3200	4

Table 6.2: Record selected from the dataset *home*

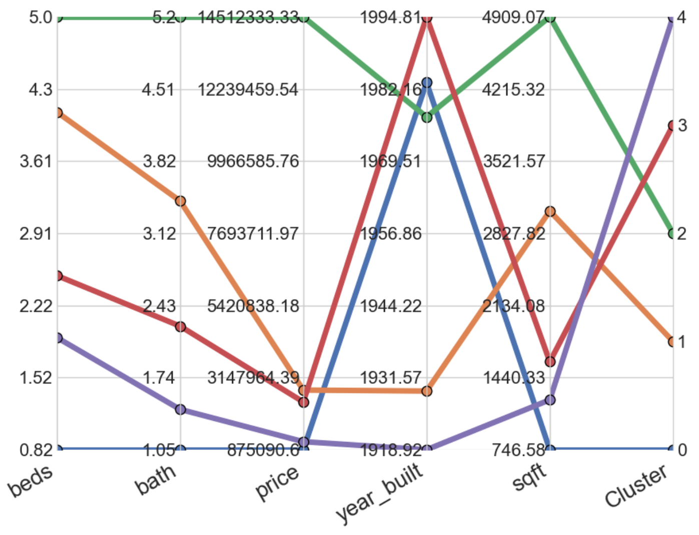
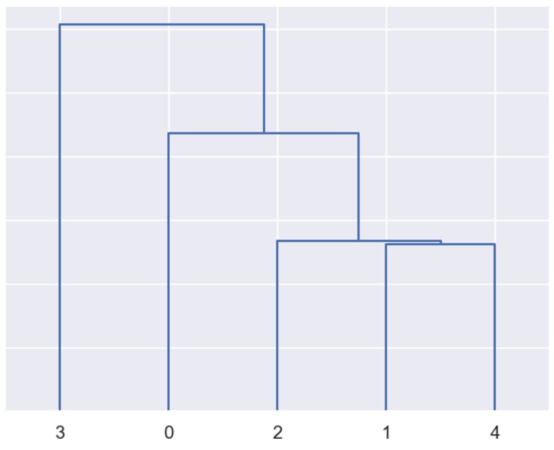
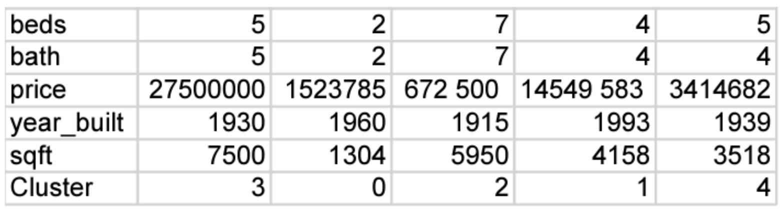


Figure 6.2: Parallel plot for dataset *home*





beds	5	2	7	4	5
bath	5	2	7	4	4
price	27500000	1523785	672 500	14549 583	3414682
year_built	1930	1960	1915	1993	1939
sqft	7500	1304	5950	4158	3518
Cluster	3	0	2	1	4

Figure 6.3: Dendrogram for dataset *home*

6.2 Cognitive Workload and Usability

For the Cognitive workload, users have been given the NASA Task Load Index (TLX) questionnaire. The NASA TLX [9] is a popular technique for measuring subjective mental workload. It relies on a multidimensional construct to derive

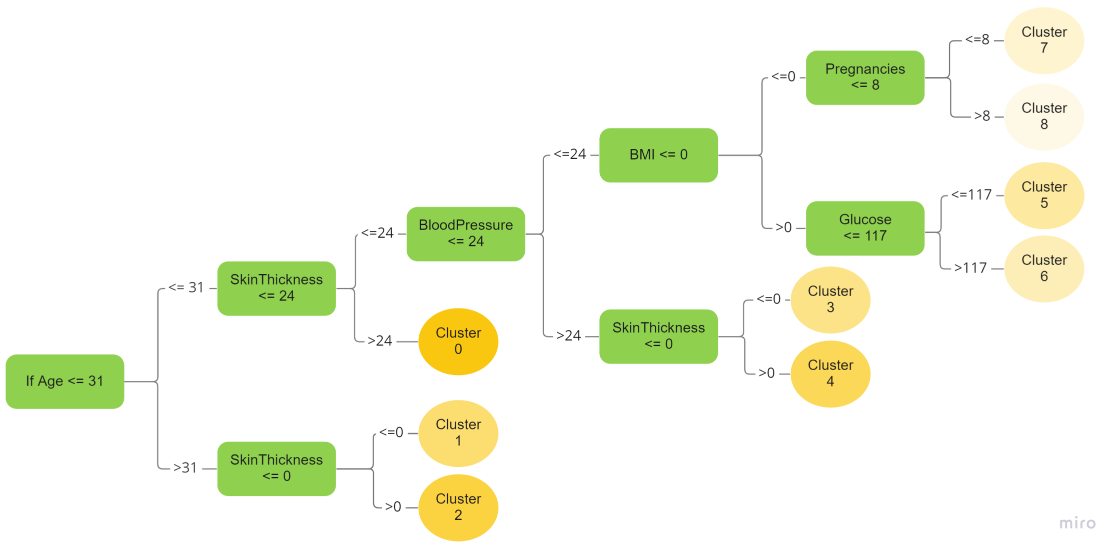


Figure 6.4: Tree for dataset *diabetes*

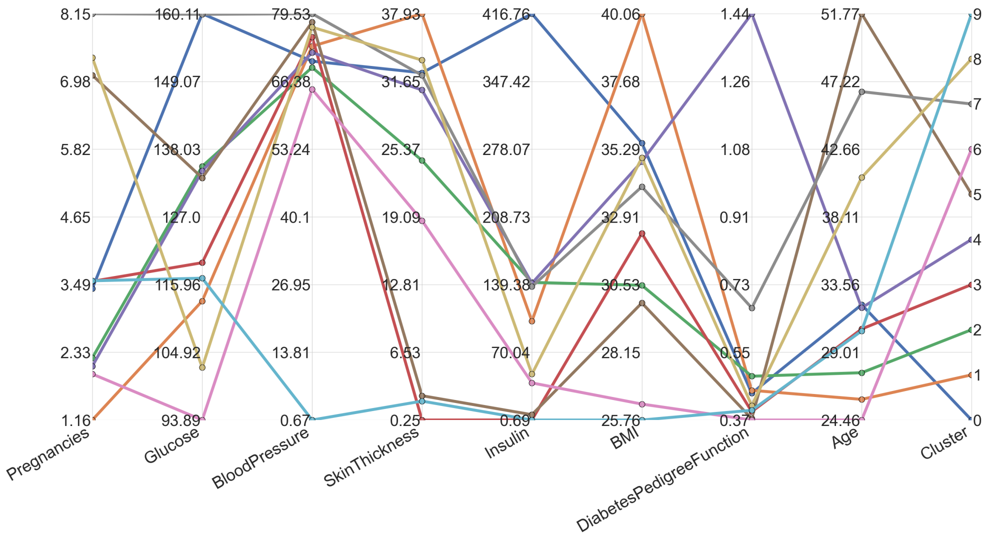
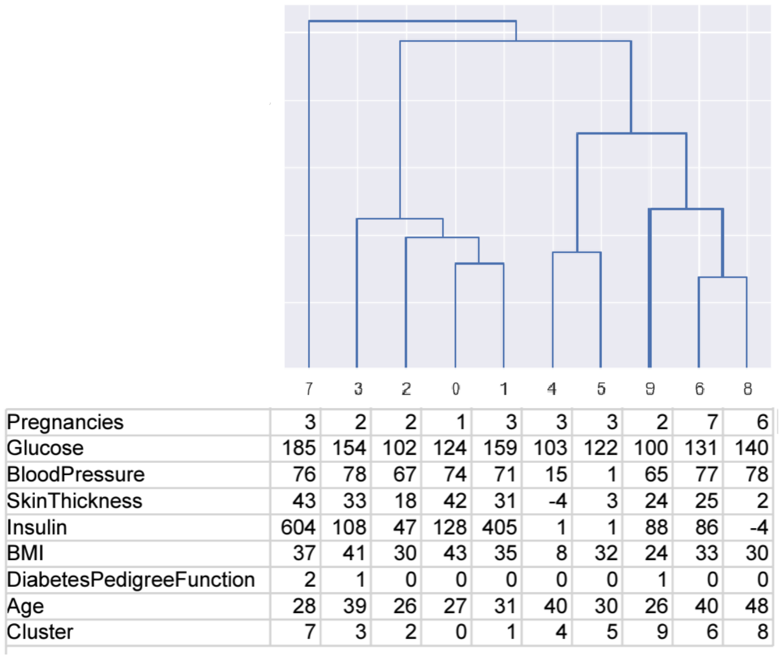


Figure 6.5: Parallel plot for dataset *diabetes*

	Pregnancies	*Glucose*	*Blood Pressure*	*Skin Thickness*	*Insulin*	*BMI*	*Diabetes Pedigree Function*	*Age*	*Cluster*
Tree	1	89	66	23	94	28.1	0.167	21	4
PP	5	132	80	0	0	26.8	0.186	69	5
Dend	4	111	72	47	207	37.1	1.39	56	3

Table 6.3: Record selected from the dataset *diabetes*

an overall workload score based on a weighted average of ratings on six subscales: mental demand, physical demand, temporal demand, performance, effort, and frus-



Pregnancies	3	2	2	1	3	3	3	2	7	6
Glucose	185	154	102	124	159	103	122	100	131	140
BloodPressure	76	78	67	74	71	15	1	65	77	78
SkinThickness	43	33	18	42	31	-4	3	24	25	2
Insulin	604	108	47	128	405	1	1	88	86	-4
BMI	37	41	30	43	35	8	32	24	33	30
DiabetesPedigreeFunction	2	1	0	0	0	0	0	1	0	0
Age	28	39	26	27	31	40	30	26	40	48
Cluster	7	3	2	0	1	4	5	9	6	8

Figure 6.6: Dendrogram for dataset *diabetes*

tration level.

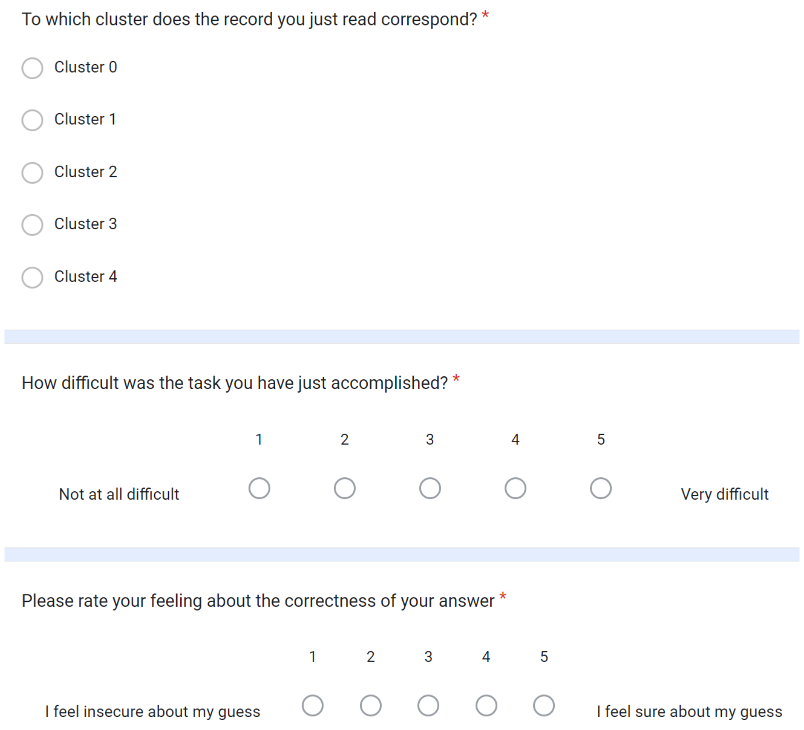
For usability, it is important to say that this does not exist in an absolute sense; can only be defined with reference to particular contexts This, in turn, means that there are no absolute measures of usability, since, if the usability of an artifact is defined by the context in which that artifact is used, measures of usability must of necessity be defined by that context too. We decide to use the System Usability Scale (SUS) [8], a simple usability scale. It has a ten-item scale giving a global view of subjective assessments of usability.

Both questionnaires mentioned are one-dimensional, for this reason, it is necessary to calculate the mean and the variance of the results obtained.

6.3 Creation of Google Forms

We created a total of three Google Forms each containing the six views presented in the previous section. The difference in Forms is simply the order in which views are presented to users. In the first Form, there is first the view of the Tree, then the Parallel Plot, and finally the Dendrogram, this for the *home* dataset, in the same order there are the three plots for the *diabetes* dataset. In the second Form instead, first, we show the Parallel plot, then the Dendrogram, and finally, the Tree, always first for the *home* dataset and then *diabetes*. Finally, the last Form first displays the Dendrogram, then the Tree, and finally the Parallel plot. We now describe how one of these Forms is organized. It is important to underline that each user will be given only one of these versions.

The first section of a Form contains the Informed Consent that the user must accept to continue. In the second one, gender information, age, school level, and major are requested. Then we have the record that the user has to classify and the relative visualization, for example, Tree. The user will no longer be asked to answer three questions. The first involves locating the cluster they think the record belongs to. The second concerns the level of difficulty of the task and, the third concerns how confident they feel with the answer given to the first question. Figure 6.7 shows the questions just described. It is important to emphasize that under each view there are these questions. If the visualization concerns the *diabetes* dataset, the number of clusters increases.



To which cluster does the record you just read correspond? *

- ◯ Cluster 0
- ◯ Cluster 1
- ◯ Cluster 2
- ◯ Cluster 3
- ◯ Cluster 4

How difficult was the task you have just accomplished? *

	1	2	3	4	5	
Not at all difficult	◯	◯	◯	◯	◯	Very difficult

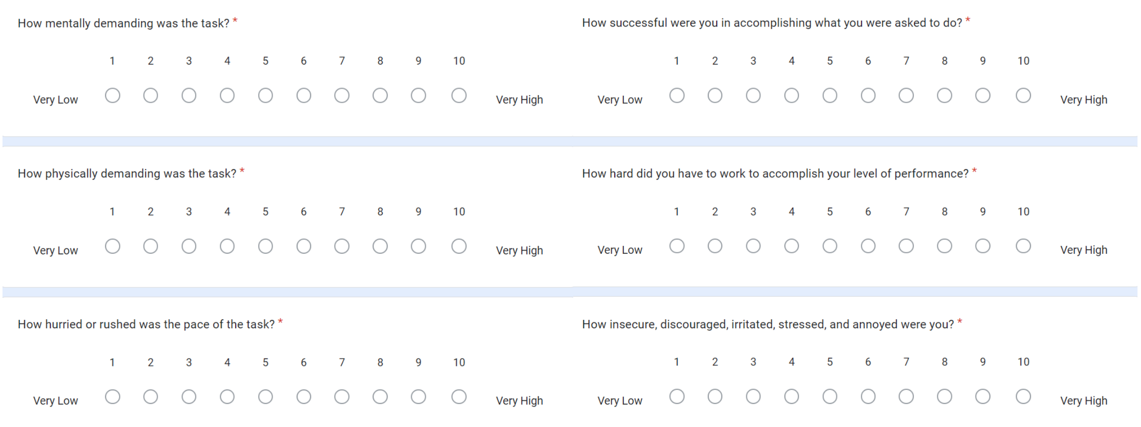
Please rate your feeling about the correctness of your answer *

	1	2	3	4	5	
I feel insecure about my guess	◯	◯	◯	◯	◯	I feel sure about my guess

Figure 6.7: Question asked to user, for each visualization.

After these questions, we have the first questionnaire. NASA TSX, in Figure 6.8 there are the six questions submitted to users. Subsequently to this, the SUS questionnaire is also administered, in Figure 6.9 there are the questions submitted. These questionnaires are about the experience with a single plot. For this reason, each user is asked to fill them both out for all six plots that he will find in the Forms.

In the last section of the Form, we give the possibility to ask open questions. After, we ask if there have been any difficulties or problems and finally, we give the user the opportunity to leave us some suggestions.



How mentally demanding was the task? *

	1	2	3	4	5	6	7	8	9	10	
Very Low	○	○	○	○	○	○	○	○	○	○	Very High

How successful were you in accomplishing what you were asked to do? *

	1	2	3	4	5	6	7	8	9	10	
Very Low	○	○	○	○	○	○	○	○	○	○	Very High

How physically demanding was the task? *

	1	2	3	4	5	6	7	8	9	10	
Very Low	○	○	○	○	○	○	○	○	○	○	Very High

How hard did you have to work to accomplish your level of performance? *

	1	2	3	4	5	6	7	8	9	10	
Very Low	○	○	○	○	○	○	○	○	○	○	Very High

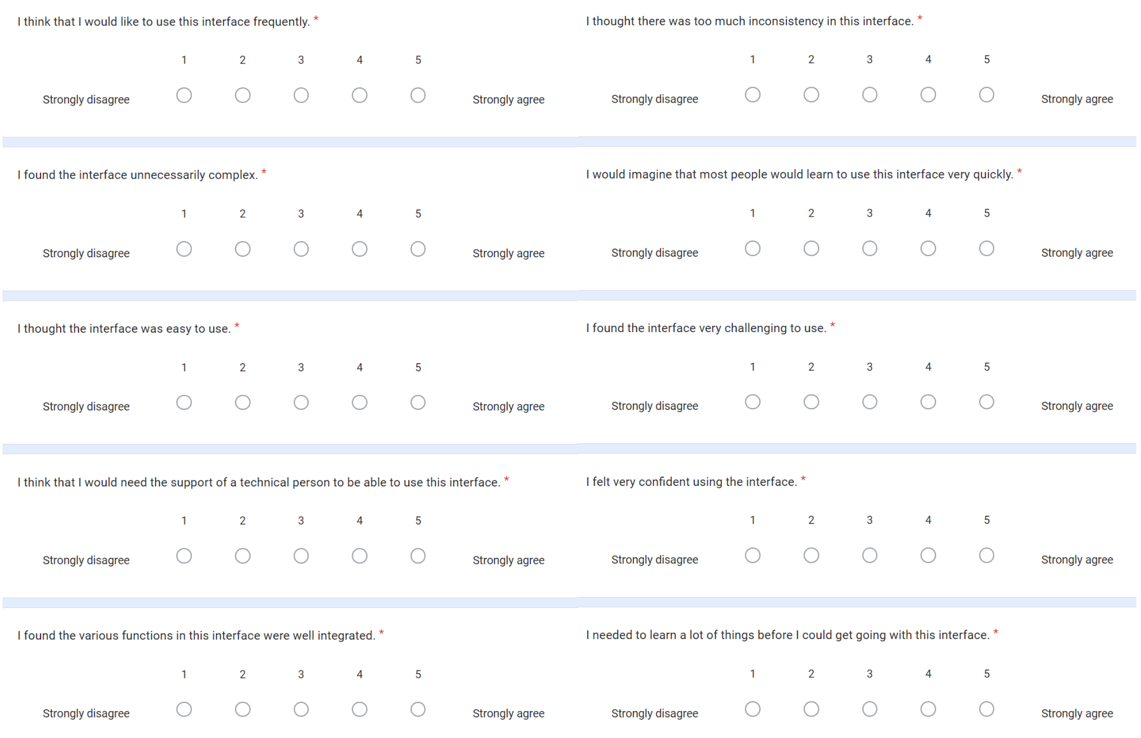
How hurried or rushed was the pace of the task? *

	1	2	3	4	5	6	7	8	9	10	
Very Low	○	○	○	○	○	○	○	○	○	○	Very High

How insecure, discouraged, irritated, stressed, and annoyed were you? *

	1	2	3	4	5	6	7	8	9	10	
Very Low	○	○	○	○	○	○	○	○	○	○	Very High

Figure 6.8: Questions for the NASA TSX.



I think that I would like to use this interface frequently. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I thought there was too much inconsistency in this interface. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I found the interface unnecessarily complex. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I would imagine that most people would learn to use this interface very quickly. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I thought the interface was easy to use. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I found the interface very challenging to use. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I think that I would need the support of a technical person to be able to use this interface. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I felt very confident using the interface. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I found the various functions in this interface were well integrated. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I needed to learn a lot of things before I could get going with this interface. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

Figure 6.9: Questions for the SUS.

6.4 Results

We now present the results obtained. First of all, from the preliminary questions on gender and age, it emerged, as can be seen in Figure 6.10, that our participants are on average 25 years old and that 70% are males while 30% are females. We recall that the test was carried out by the first-year class of the Data Mining course, a course attended by more boys than girls. In addition, the participants all have a scientific or economics Bachelor's Degree.

In Figure 6.11, we can visualize the total scores that each visualization obtained,

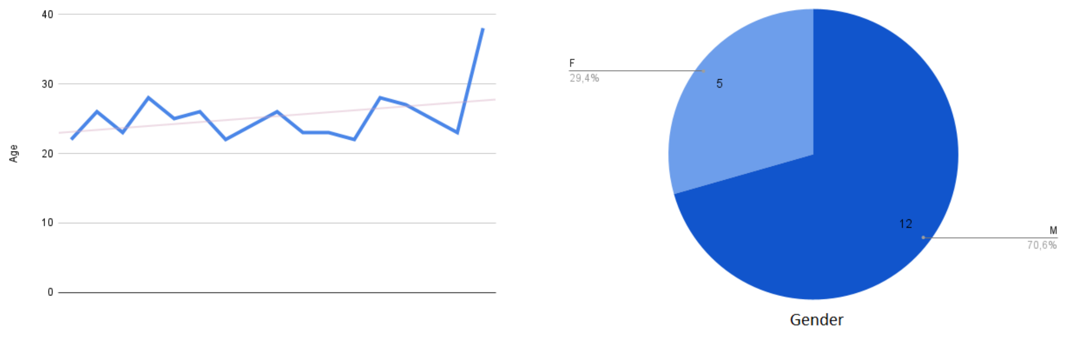


Figure 6.10: Distribution of Age and Gender

these are divided into *easy*, for the *home* dataset, and *difficult*, for the *diabetes* dataset. Analyzing the individual pie charts, it emerges that our Tree view, in both cases, has more than 80% of correct answers. This allows us to demonstrate that, regardless of the difficulty of the dataset, with Trees, it is very easy to identify the correct cluster. For visualizations with the Parallel Plot instead, in the easy case, we have more or less the same percentage of correct and wrong answers, while in the difficult case we have 76% of correct answers, demonstrating the fact that it is more complicated to identify clusters correctly with this type of plot. Finally, for the Dendrograms, from the easy case, it emerges that 88% of the answers given by our participants are correct, while in the difficult case 94% of the answers are wrong. This shows that, in this particular type of graph, the number of features to be checked to insert a record in its cluster is an important variable that, if increased, increases the level of difficulty and the possibility of making mistakes.

6.4.1 Questionnaire Results

For each of the Forms submitted to the participants, we analyze the results that emerged from the questionnaires submitted. For the **Difficulty** and **Confidence** metrics, having a good value, for the first, means having a low value, while for second, having a high value. For values from the **NASA** questionnaire, each participant was first averaged, giving them their own value, and then these individual values were aggregated, for each view. Therefore, having a low value of NASA implies a small effort at a cognitive level, conversely, a high value, a greater effort at a cognitive level. The same technique was applied to the **SUS** questionnaire. In this case, having a high value means being more usable and recommendable, while a low level means less usability.

In Figure 6.12, it is immediately to observe that the difficulty level for Tree is lower than the other two, while for Confidence, i.e. the trust one has in one's answer, the value with Tree is much higher. From the Nasa questionnaire, it can be seen that, also, in this case, Tree requires much less cognitive work, especially compared to the Parallel Plot difficulty, while for the Dend difficulty, the value is very similar to that of Tree difficulty, but we remind you that in this last case, in Dend difficulty

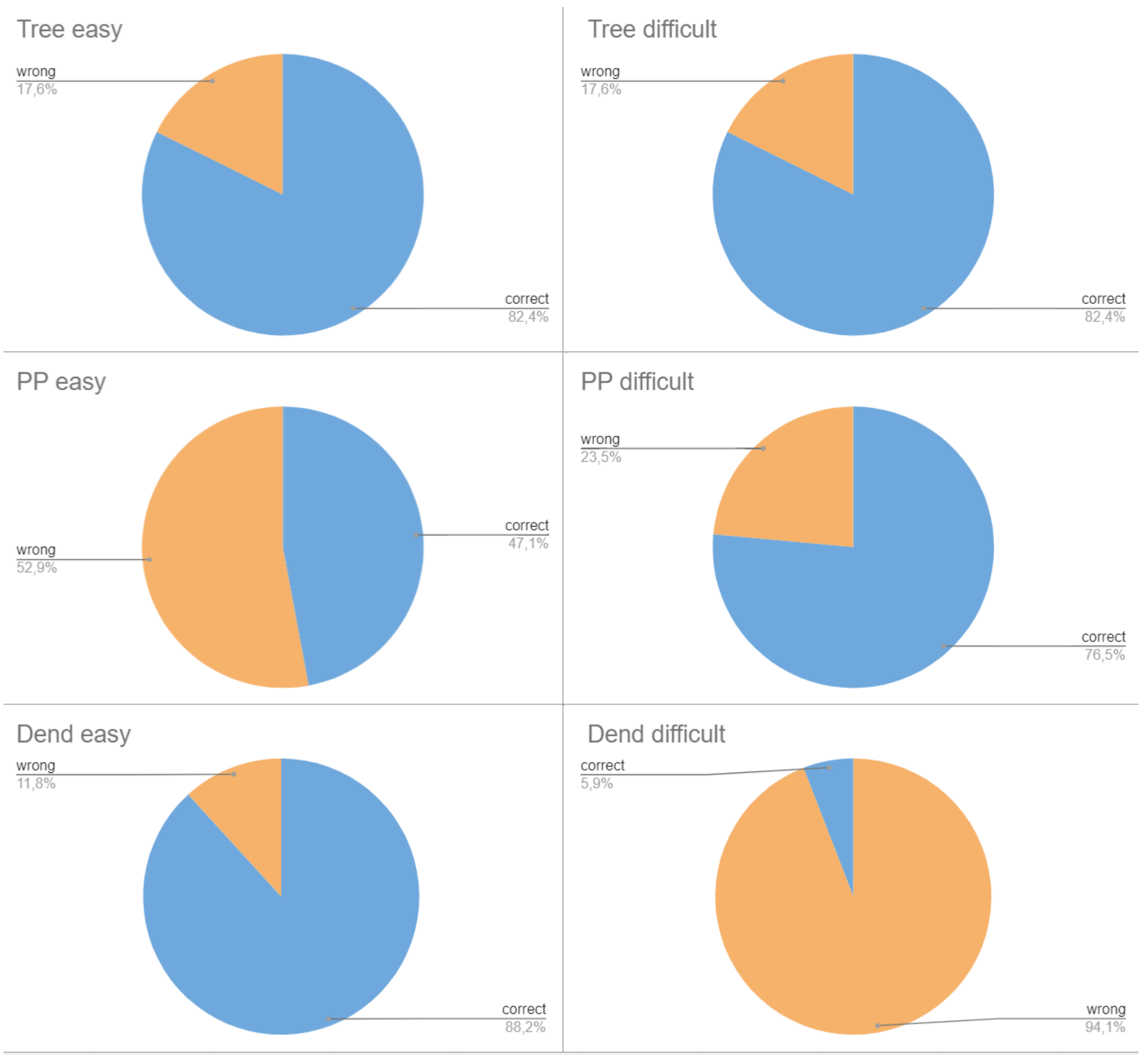


Figure 6.11: Total scores

there are 94% of wrong answers. Finally, for the SUS, the usability of our Tree is higher than the other views, as expected.

In Figure 6.13, let's analyze another form, the one with the PP as its first visualization, then the Dendrogram, and finally our Tree. Also in this case it is confirmed, even more, the fact that our Tree is simpler to use than the PP and the Dend, above all in the difficult case. The confidence in answering, also here is higher in Tree and lower in the case of Dend difficult and PP difficult. For the NASA questionnaire, the Tree easy has a very low value, demonstrating the low cognitive load of the task, while the other views have a much higher value. Finally for the SUS, more or less the PP and the Dend have the same values, demonstrating that they are less usable than our Tree, which instead has higher usability values.

In the last Figure 6.14, representing the Form with the order of display, Dendrogram then our Tree, and finally the Parallel Plot, we can only confirm what was said previously. The difficulty with Tree is much less, the confidence you have in giving

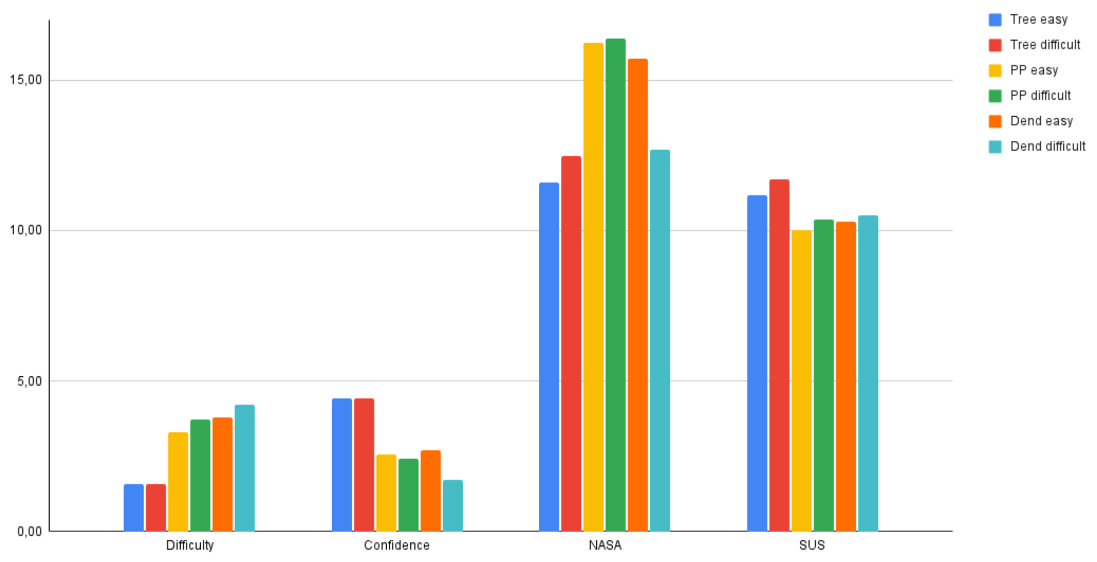


Figure 6.12: Tree-PP-Dend: questionnaire results

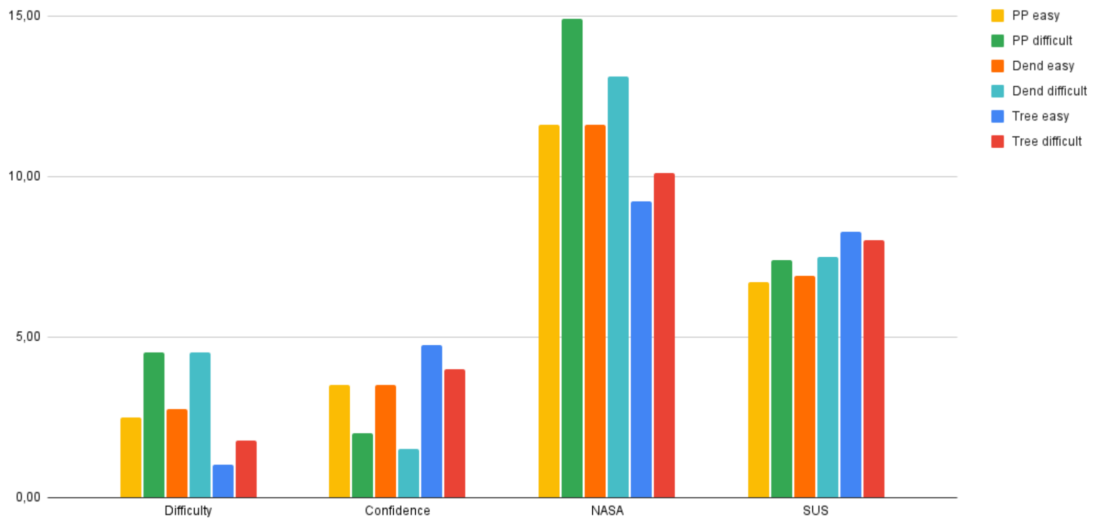


Figure 6.13: PP-Dend-Tree: questionnaire results

the answer is greater, the cognitive workload is less and its usability is greater.

We also asked our participants to tell us where they encountered the most difficulties if any. Some of them have left us comments which we report below:

- *"Except the decision trees, all other charts were so hard to try to interpret"*
- *"dendograms and lineplot seem to me quite inefficient for these task"*
- *"Identify values in dendograms"*

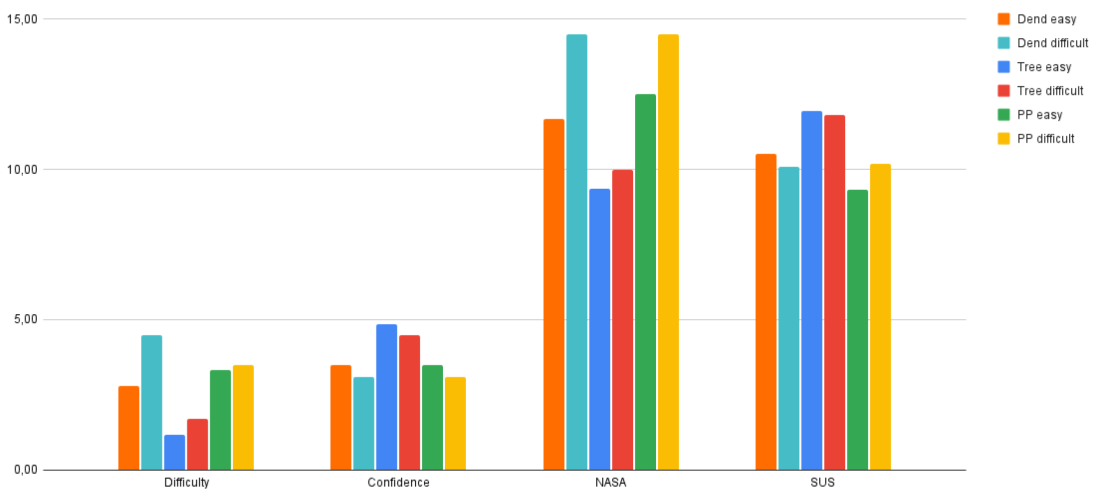


Figure 6.14: Dend-Tree-PP: questionnaire results

- *"Yes I found dendogram and line plot very difficult to interpretate especially when there were several attributes "*
- *"The biggest issue was to follow the lines in the first interface, especially where the density of points was very high"*

Thanks to these comments, we can only confirm our hypotheses. Trees are much simpler and more intuitive than Parallel plots and Dendrograms.

Chapter 7

Conclusion

In this thesis, we have tried to improve the state-of-the-art of tree-based clustering methods developing ParTree, that yields an unsupervised binary tree to interpret the data partitioning. The main innovation of ParTree is the clustering method, it simultaneously builds the tree and partitions the data, searching for the best splits over subsequent iterations by aligning the logic inductive process with the clustering extraction calculus. In other words, the methodology for identifying clusters is the same as that used to construct the tree. Furthermore, there is no need to use another clustering algorithm with all the associated issues with effectiveness, parameter tuning, and appropriate distance function selection. We create three versions of ParTree, C-PT, I-PT, and P-PT, that differ from one another in terms of the criterion used to perform the data splitting to demonstrate its efficacy. Additionally, in contrast to the state-of-the-art clustering techniques, the versions of ParTree, C-PT, and I-PT, can be applied to numerical, categorical, and mixed datasets. As can be seen from our experiments, ParTree is in line with many of the competitor methods considered. We have shown that used on real datasets, the best results are obtained with the ParTree, P-PT, and I-PT versions. Concerning synthetic datasets, ParTree is in line with state-of-the-art algorithms, the values of its metrics are in fact similar compared to the other methods. The critical point of ParTree is the execution time. As can be seen from experiments on real datasets, the computation time is greater than in other algorithms. However, this does not impact the general performance, on the contrary, as can be seen from the various plots and tables, it keeps the metric values in line, if not increases. The execution time on synthetic datasets, on the other hand, is in line with those of the state of the art. Also, it is worth mentioning that since the search for the best split is independent among the various features, then it could be re-implemented more efficiently by using a parallel search procedure. Finally, the developed case study allows us to demonstrate that our method has other advantages that state-of-the-art algorithms do not have. Trees are much easier to use than Parallel Plots and Dendrograms. From a cognitive point of view, using Trees is much less demanding. We also proved that usability is higher, thanks to the SUS questionnaire. Even the comments left by our users have confirmed this.

As future research directions, an improvement that could be made concerns P-PT, in fact, this type of split only works with continuous type features. For this reason, the split must be extended to categorical features, so that the method is complete. A possibility in this direction is to use Light-FAMD, a library for processing factor analysis of mixed data. This includes a variety of methods including principal component analysis (PCA) which would be used precisely for our P-PT split. Furthermore, we would like to test with more datasets, even of real type, and to try to solve the scalability issues of our proposal. Finally, we would like to employ ParTree on a real study case with a final user which is the one who required a clusterization of a certain population for instance for customers segmentation.

Acknowledgments

For me, this degree thesis coincides with the achievement of one of the most important goals of my life, as its drafting took place in painful and difficult months marked by a great void. Being here writing the acknowledgments today is a big win! I would therefore like to thank all those who have allowed and encouraged me. It is not easy to quote and thank, in a few lines, all the people who have contributed to the birth and development of this work: who with constant collaboration, who with moral or material support, who with advice and suggestions or just with words of encouragement. I hope I'm not forgetting anyone, but if you, my dear reader, shouldn't find your name in the next few lines… tell me and I'll thank you in person, even if you're interested in reading.

First of all, my thesis advisor, Riccardo Guidotti. Without him, none of this would have been possible. He was able to help and guide me in writing this thesis, not only thanks to his skill, but also thanks to his kindness and empathy, qualities in which I believe he excels.

I would also like to thank Andrea Beretta, without him it would not have been possible to create the case study, thank you for helping me demonstrate the value of my work also with real people and not just with numbers.

I thank my parents, Papo and Mami, they have been by my side since the beginning of my journey at this university. Since day one, especially during the COVID period, they have been there in the room next to mine cheering me on, supporting me through every exam, and ready to celebrate every little step along this path. Thank you for supporting me even from a distance, for calling me every day, and for making me feel your presence always close to me, I will never be as grateful to you as I am now.

I thank my two grandmothers of the heart, Maria and Corrada. Thanks to your wisdom and sweetness, you have pampered me during this journey. Thanks to grandma Maria for preparing me the best delicacies when I went downstairs to visit you, for hugging me when I needed it most, and for rejoicing at every little achievement, but above all thanks for being here today. Thanks to grandma Corrada, for having told me the most bizarre and compelling life stories, and for always smiling and supporting me. I thank all my relatives, Sara, Uncle Ignazio, Aunt Mariangela, Nico Aunt Ines, and Uncle Marino, for always being by my side and supporting me throughout this journey, for being by my side when I needed and for always smiling at me every time I came down to see you.

Special mention to my little Rosy, my sweet and dear little sister. I thank her, who is my best friend, my only sweet half, the person who most of all knows how much I have suffered throughout this period. She celebrated every single happy moment with me and cried with me. Thank you for bearing with my every single complaint and being able to turn it into joy, because that's what you are to me, a little fairy who is able to turn every negative side of me into a positive. Thanks for making me laugh when all I wanted to do was cry.

Thanks to my friends from Winx Club + Davide, Mary Rose, Napina, Laura, and Davide of course. Thank you for making me spend wonderful moments this summer. Thank you for listening to my outbursts, always believing in me and in what I was doing, and for making me smile when I thought I was alone. Thanks for always being there.

A special thanks to my roommates Giulia and Alice, for sharing the house with me, for always putting up with and supporting me, for being by my side when I needed it most, for always being so helpful and sweet, and for all the nice lunches/dinners that we shared together. I love you!

I thank them, the group of friends I never thought I could find in my life, my second family. Trans8x2+1+2+1-2(presi bene).For being the friends to spend the best evenings of my life. Thanks for always being there, despite the mess I know I am.

Thanks to Saverio for telling me 'you are good at all the things you do when I was convinced I was good for nothing. Thank you for helping me discover myself, and always encouraging me to be a better person than I was before I met you. Thank you for all the mental breakdowns we spent together, tears taste different when they are shared.

Thanks to Giulia for being the friend that everyone would like, the sweetest and most sensitive person I know, for putting up with me, and for always being able to make me live the moments that were full of anxiety for me lightly.

Thanks to Samu, for being my best neighbor, for driving me home in the worst conditions, for holding my head more times than allowed, and for being an excellent study partner.

Thanks to Lia, for being a faithful friend and for always being there. Thank you for your radiance and your common sense that has accompanied me on this journey.

Thanks to the brother I never had, Salvo, for being a guide for me, especially in writing this thesis, and for giving me the best life advice on the best evenings.

Thanks to Ale, for being an excellent 'witch bitch' and drinking companion, thanks to her lightness and light-heartedness she was able to make me laugh and have fun every time we spent time together.

Thanks to Gio Gio, for being an excellent lunch and study companion, but also for having organized the best dinners in the world. Thank you for showing me a completely different world than mine.

Thanks to RuggIero, for the laughs, the hugs, the random things shouted in the square, the dinners offered (by me) at the canteen, and for being a very good friend. Thanks for studying with me, TVB a lot.

Thanks to Gaetano, for supporting me from an academic point of view, for giving me excellent growth advice, for helping me whenever I needed it, and for spending very enjoyable evenings with me.

Thanks to Luke, for being the sweetest and kindest person throughout this whole journey, for listening to me when I needed it most, and for being my drinking guardian angel.Thank you for that simple question you ask anyone as soon as you meet him, that "how are you?" that can warm your heart.

Thanks to Gianni, for putting up with my every bullshit, for being an excellent drinking companion and for all the laughs you've managed to snatch from me since the bachelor degree.

Thanks to Viola, for being a good friend, an excellent cook and for all the good times we spent together.

Thanks to Nick, for his composure and kindness. Thank you for complaining to me every evening about the length of via Fratti always too long). But above all thank you for sharing with me this very special day that we both deserve. And of course, AUGURI DOTTORE!

Thanks to Matte, for making me laugh and being the smartest person I know. Thanks for explaining to me in detail how you operate mice, now that I have this knowledge too I'm complete.

A big thank you to all the dear friends who in different ways, through words, gestures, messages, laughter, walks, chats and drinks have encouraged me! Thanks to Domy, Andrea, Martina, Davide, Fabio, Giada and Mario.

At this point I should thank you too, I won't name you, but I know anyone would understand. It's not easy to write something you'll never read, you know how I'm made, I'd like you to read these lines and share this moment with me, but life has made us take different paths. Thank you for helping me choose this path and for having been by my side for many years of my life, even if you're gone now, there will always be a space for you inside me. I will love you forever.

'So let's take a ride and see what's mine, singin' la la la'

Finally, the last thank was to me. I know it’s a very trivial and obvious thing, but it’s also a must-do. I’ve been tempted to give up everything many times, especially at the beginning of this thesis, I felt alone and abandoned, I cried and suffered a lot, and I thought I couldn’t make it. Thank you, Mari, for not giving up in the worst period, for rolling up your sleeves and writing on that blackboard "Tu sei forte e importante per te" because it is. This milestone demonstrates how much you are truly worth to anyone who has abandoned you or who has never believed in you. Thank you, Mari, for being here now to write these thanks, for believing in this work, for being fantastic, and determined, for fighting until the end, and for managing all the stress that has been thrown at you in an (almost) good way.

Thank you for being the best version of yourself today, the woman you always wanted to be.

Your Mari

Bibliography

[1] A. Adadi and M. Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (XAI). *IEEE Access*, 6:52138–52160, 2018.

[2] M. Balaban, N. Moshiri, U. Mai, X. Jia, and S. Mirarab. Treecluster: Clustering biological sequences using phylogenetic trees. *PloS one*, 14(8):e0221068, 2019.

[3] D. Bertsimas and J. Dunn. Optimal classification trees. *Mach. Learn.*, 106(7):1039–1082, 2017.

[4] D. Bertsimas, A. Orfanoudaki, and H. M. Wiberg. Interpretable clustering via optimal trees. *CoRR*, abs/1812.00539, 2018.

[5] D. Bertsimas, A. Orfanoudaki, and H. M. Wiberg. Interpretable clustering: an optimization approach. *Mach. Learn.*, 110(1):89–138, 2021.

[6] H. Blockeel, L. D. Raedt, and J. Ramon. Top-down induction of clustering trees. In *ICML*, pages 55–63. Morgan Kaufmann, 1998.

[7] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, 1984.

[8] J. Brooke. Sus: A quick and dirty usability scale. *Usability Eval. Ind.*, 189, 11 1995.

[9] A. Cao, K. Chintamani, A. Pandya, and R. Ellis. Nasa tlx: Software for assessing subjective mental workload. *Behavior research methods*, 41:113–7, 03 2009.

[10] J. Chen, Y. Chang, B. Hobbs, P. J. Castaldi, M. H. Cho, E. K. Silverman, and J. G. Dy. Interpretable clustering via discriminative rectangle mixture model. In *ICDM*, pages 823–828. IEEE Computer Society, 2016.

[11] Y. Chen, W. Hsu, and Y. Lee. TASC: two-attribute-set clustering through decision tree construction. *Eur. J. Oper. Res.*, 174(2):930–944, 2006.

[12] S. Dasgupta and Y. Freund. Random projection trees and low dimensional manifolds. In *STOC*, pages 537–546. ACM, 2008.

[13] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, dec 2006.

[14] R. Fraiman, B. Ghattas, and M. Svarc. Interpretable clustering using unsupervised binary trees. *Adv. Data Anal. Classif.*, 7(2):125–145, 2013.

[15] Y. Freund, S. Dasgupta, M. Kabra, and N. Verma. Learning the structure of manifolds using random projections. In *NIPS*, pages 473–480. Curran Associates, Inc., 2007.

[16] N. Frost, M. Moshkovitz, and C. Rashtchian. Exkmc: Expanding explainable k-means clustering. *CoRR*, abs/2006.02399, 2020.

[17] B. Ghattas, P. Michel, and L. Boyer. Clustering nominal data using unsupervised binary decision trees: Comparisons with the state of the art methods. *Pattern Recognit.*, 67:177–185, 2017.

[18] R. Guidotti, A. Monreale, M. Nanni, F. Giannotti, and D. Pedreschi. Clustering individual transactional data for masses of users. In *KDD*, pages 195–204. ACM, 2017.

[19] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 51(5):93:1–93:42, 2019.

[20] A. E. Gutiérrez-Rodríguez, J. F. M. Trinidad, M. García-Borroto, and J. A. Carrasco-Ochoa. Mining patterns for clustering on numerical datasets using unsupervised decision trees. *Knowl. Based Syst.*, 82:70–79, 2015.

[21] J. U. Jure Leskovec, Anand Rajaraman. Dimensionality reduction. In *Mining of Massive Datasets*, pages 436–438. Cambridge University Press, 2019.

[22] P. Kranen, I. Assent, C. Baldauf, and T. Seidl. The clustree: indexing microclusters for anytime stream mining. *Knowl. Inf. Syst.*, 29(2):249–272, 2011.

[23] D. T. Lee and C. K. Wong. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica*, 9:23–29, 1977.

[24] B. Liu, Y. Xia, and P. S. Yu. Clustering through decision tree construction. In *CIKM*, pages 20–29. ACM, 2000.

[25] O. Loyola-González, A. E. Gutiérrez-Rodríguez, M. A. Medina-Pérez, R. Monroy, J. F. M. Trinidad, J. A. Carrasco-Ochoa, and M. García-Borroto. An explainable artificial intelligence model for clustering numerical databases. *IEEE Access*, 8:52370–52384, 2020.

[26] M. McCartin-Lim, A. McGregor, and R. Wang. Approximate principal direction trees. In *ICML*. icml.cc / Omnipress, 2012.

[27] T. Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artif. Intell.*, 267:1–38, 2019.

[28] M. Moshkovitz, S. Dasgupta, C. Rashtchian, and N. Frost. Explainable k-means and k-medians clustering. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 7055–7065. PMLR, 2020.

[29] D. Pelleg and A. W. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, page 727–734, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[30] D. Pelleg and A. W. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *ICML*, pages 727–734. Morgan Kaufmann, 2000.

[31] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, 1986.

[32] J. Sander. *Density-Based Clustering*, pages 270–273. Springer US, Boston, MA, 2010.

[33] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to data mining.* Pearson Education India, 2016.

[34] P. Tavallali, P. Tavallali, and M. Singhal. K-means tree: an optimal clustering tree for unsupervised learning. *J. Supercomput.*, 77(5):5239–5266, 2021.

[35] S. Thomassey and A. Fiordaliso. A hybrid sales forecasting system based on clustering and decision trees. *Decis. Support Syst.*, 42(1):408–421, 2006.

[36] N. Verma, S. Kpotufe, and S. Dasgupta. Which spatial partition trees are adaptive to intrinsic dimension? In *UAI*, pages 565–574. AUAI Press, 2009.

[37] I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[38] L. Zappia and A. Oshlack. Clustering trees: a visualization for evaluating clusterings at multiple resolutions. *Gigascience*, 7(7):giy083, 2018.