Alma Mater Studiorum - Università di Bologna

Department of Physics and Astronomy
Master Degree in Physics

UNSUPERVISED MONDRIAN FOREST: A SPACE PARTITION METHOD FOR CLUSTERING

Supervisor:
Prof. Enrico Giampieri

Submitted by:
Silvia Maria Macrì

Academic Year 2020/2021

Abstract

Cluster analysis is an ensemble of techniques whose aim is to divide an unlabeled dataset into groups so that samples with similar features are assigned to the same groups and dissimilar samples are assigned to different ones; it has been applied in several fields and it consists of a wide variety of techniques, each one designed for a specific type of dataset and required prior informations on its structure. In this thesis we discuss a new formulation of clustering technique whose structure is similar to that of an unsupervised random forest; it is based on the Mondrian stochastic process and it consists of a hierarchical partition of the space of definition of the given dataset. It gives as output an estimation of the probability distribution to belong to a certain class, defined on the whole underlying space, and it shows some interesting properties, like the automatic determination of the number of clusters and the capability to deal with different shape datasets. After a brief theorical introduction about clustering, the Mondrian stochastic process and its main mathematical properties are defined. The Mondrian clustering algorithm is then described and results of its applications on two and three dimension toy datasets are presented; the discussion focuses on the role of the algorithm parameters, that characterize the method and can be possibly tuned by the user, in order to obtain better performances when dealing with different datasets, and on the comparison of the results with that of some notable clustering algorithms. Finally, some interesting aspects that could be further investigated are discussed.

Contents

Chapter 1

Clustering

Cluster analysis is a machine learning approach whose aim is to divide a population of data into a certain number of groups, by recognizing patterns and relationships among the data points. It is part of the branch of machine learning called *unsupervised* learning, in contrast with the *supervised* learning methods, like classification, in which data are grouped on the basis of some existing labeling. In this latter case, the assignment of a sample to a certain class is obtained by first training the algorithm with a labeled dataset, in order to associate each class to some corresponding features of the input samples; once the algorithm has been trained, it is able to assign unlabeled samples to the already determined classes, on the basis of their feature similarity with the training data ones. On the contrary, in the unsupervised learning case, the input data are not labeled and the division into classes is performed in order to maximize intra-class similarity and minimize inter-class similarity. Since there is not an unique and precise definition of similarity, several clustering techniques have been developed, each one characterized by a specific metric.

Cluster analysis has been widely studied and applied in various fields. A possible application is the division of a great number of biological data into groups, each one corresponding to a cell tipology. We can for example measure, for each cell, some shape feature, as the diameter, and other numerical values, as the data related to the radiation emitted, through fluorescence, after excitation with different wavelength; since different cell typologies are characterized by their own range of dimensions and react differently to the same stimulus, similar cells are expected to exhibit similar values. Since, when dealing with large number of data, a manual labeling is not viable, cluster analysis is used, in order to group data into clusters, each one corresponding to a specific cellular tipology, through the evaluation of the similarity between the features.

In this chapter, we briefly illustrate the main tipologies of clustering techniques, describing more in detail some popular algorithms, that will be used in the next chapters; then, the role of decision trees and random forests in unsupervised learning is discussed and, finally, a few methods used to compare different clustering results are presented.

1.1 Types of clustering algorithms

In order to deal with datasets with different numerosity, number of clusters or cluster shapes, a wide variety of clustering techniques has been developed; all of them are characterized by their own methods and metrics and are designed to be applied to datasets with specific features and in presence of specific prior knowledges about the samples.

A fundamental distinction is between hierarchical and partitioning clustering methods; however, several other classifications have been formulated, since the large number of algorithms can't be strictly categorized. In the following, the main typologies of clustering methods are presented.

Hierarchical clustering

In hierarchical clustering methods, clusters are formed by iteratively dividing the patterns using top-down or bottom up approach [1]; the **divisive** or top-down algorithms start by considering the whole set of data as belonging to an unique initial cluster, that is progressively divided into an increasing number of clusters; on the contrary, the **aggregative** or bottom-up algorithms start with each element of the set of data belonging to different clusters and progressively merge them; the splitting or merging procedure is carried on until a specific condition is satisfied or until all elements are completely separated (in case of divisive approach) or all gathered in a unique cluster (in case of agglomerative approach). The hierarchical structure consists on the partitioning/merging of the objects organized in levels and on the property that whenever two samples are in the same cluster at level k, they remain together at all higher levels [2]. The splitting and merging procedures are performed by defining a measure of the similarity of two groups of points; usually, a metric and a linkage criterion are defined. The metric allows to evaluate the distance between each pair of points within the clusters, while the linkage is a function of the metric and defines a criterion to evaluate the similarity between two clusters.

The main advantages of the hierarchical approach are that it gives informations at different levels of granularity of the dataset and it easily handle any metric and linkage formulation; the disadvantages are, besides the sensitivity to noise and outliers, that there is a no reliable stopping criterion for the merging/splitting procedure and that, once a merge or a split is committed, it cannot be undone or refined [3] [4] [1].

Partitioning clustering

Partitioning clustering algorithms assign data to a certain number of clusters by iteratively optimizing some objective function. In particular, since testing every possible cluster configuration is too computationally expensive, often the initial dataset partition is randomly choosen; then a relocation scheme, that reassigns points to the clusters, is iterated until convergence.

Although partitioning algorithms are relatively simple, scalable and at low computational cost, they often require the knowledge of the number of clusters in advance, are high sensitive to initialization phase, noise and outliers and are not able to deal with non-convex cluster of varying size and density [1].

k-means An example of partitioning clustering technique is the k-means algorithm; it divides data into clusters by minimizing the variance within the groups. In particular, the objective function that has to be minimized is the within-cluster sum-of-squares criterion [5], calculated for each cluster:

$$f_{obj} = \sum_{i=0}^{n} min(||x_i - c||^2) \tag{1.1}$$

where x_i, with $i = 1, \ldots, n$, are the points assigned to the cluster, c is the centroid calculated as the arithmetic mean of the point positions and $||x_i - c||$ is the euclidean distance between the chosen point and the centroid.

Once the number of clusters k is given as input, a centroid is randomly assigned to each cluster; then, the following two steps are iterated:

1. each point of the dataset is assigned to a cluster in order to minimize the objective function

2. for each cluster, new centroids are calculated in order to minimize the objective function

The two steps are iterated until convergence, that means until no change in the data assignment is observed.

Although k-means algorithm has been widely used for its low computational cost and because it provides good results in practical situation as anomaly detection and data segmentation, it is subject to some limitations; above all, there is no efficient method for recognizing the number of clusters and the initial centroid locations; moreover, it assumes that clusters are convex shaped and consequently doesn't provide good results if applied to datasets with elongated and irregular shapes and, finally, it is sensitive to outliers, noise and initial seed selection [1][6].

In the following chapters we will use the `cluster.KMeans` object of the Python `scikit-learn` package.

Density-based clustering

The density-based algorithms treat clusters as regions of high density, separated by regions of no or low density [4]; The density estimation is made by evaluating, for each sample, the number of data objects in its neighborhood. We see more in detail the fundamental ideas of the density-based approach by considering the DBSCAN algorithm.

DBSCAN The Density-Based Spatial Clustering of Applications with Noise works by finding core samples at high density and expand clusters from them [5]. It requires as input two parameters: ϵ is the maximum distance between two points such that one of them is considered in the neighborhood of the other, while n is the minimum number of samples belonging to a neighborhood of a point such that it is considered a core object. The algorithm considers each point of the dataset, computing the corresponding ϵ-neighborhood; a neighborhood with points in common to the neighborhood of a core sample is assigned to the same cluster of the core sample and its neighboring points. If a point isn't connected to a core point, it is classified as noise.

In order to determine if a sample is in the neighborhood of an other one, their distance is computed and, if the distance value is lower than ϵ, the point is assigned to the neighborhood. The evaluation of the neighboring points and, consequently, the determination of the clusters, highly depend on the chosen metric.

The main advantages of this method are that clusters found by DBSCAN algorithm can be of any shape, so they are not assumed to be convex-shaped as in k-means case; moreover, it doesn't require as input the number of clusters and it is able to detect outliers; finally,the result doesn't depend on the order of the data points considered by the algorithm. On the other hand, the performance of the algorithm can be very different depending on the chosen distance metric used to calculate the ϵ-neighborhood; moreover, since the ϵ parameter is unique for the whole dataset, the algorithm may be not able to identify groups of data with different densities.

In the following chapters we will use the `cluster.DBSCAN` object of the Python `scikit-learn` package. The ϵ parameter is set differently depending on the considered dataset, while n is set to the default value of 5; the metric is always euclidean.

Graph-based Clustering

In the graph-based approach, the dataset is represented through a similarity graph. Each node represents a sample and, for each pair of samples i and j, a similarity measure s_{ij} between them is computed; if the value of s_{ij} is positive or larger than a certain threshold, the corresponding pair of nodes in the graph are connected through ad edge, weighted by some function of s_{ij}. Once the similarity graph is created, the problem of clustering consists of finding a partition of the graph such that the edges between different groups have very low weights (which means

that points in different clusters are dissimilar from each other) and the edges within a group have high weights (which means that points within the same cluster are similar to each other) [7].

The variety of graph-based algorithms differs by both the way in which the similarity graph is buit and the criterion used to divide it into clusters. In the next paragraph we describe one grid-based algorithm in detail.

Spectral clustering The spectral clustering algorithm clusters data using eigenvectors of matrices derived from the samples. It requires as input the number n of clusters and performs the clustering through the following steps [1]:

1. creation of the similarity matrix, after the definition of a distance metric as similarity measure. In our cases the metric will be euclidean.

2. creation of the similarity graph according to a specific building criterion; in our cases, in order to build the graph, we will set the *k-nearest neighbors* criterion, that consists in connecting each point only to its k nearest points, according to the similarity matrix; k is set to 10

3. the Laplacian $L = D - W$ is computed; D is the *degree matrix* defined as the diagonal matrix with degrees on the diagonal, while W is the adjacency matrix of the graph

4. the first k generalized eigenvectors of $u_1, \ldots, u_k$ of the generalized problem $Lu = \lambda Du$ are computed

5. defining $\mathbf{U} \in \mathbb{R}^{\mathbf{n}\times\mathbf{k}}$ as the matrix containing the vectors $u_1, \ldots, u_k$ as columns, the points with coordinates the rows of $\mathbf{U}$, $y_1, \ldots, y_k$ are clustered with k-means algorithm into clusters $C_1, \ldots, C_k$

6. the final clusters are $A_i, \ldots, A_k$ with $A_i = \{j | y_j \in C_i\}$

The main advantage of the spectral clustering is that it provides good performances with datasets characterized by different shapes, since it doesn't make strong assumptions on the form of the clusters; moreover, it works efficiently if applied to large datasets. On the other hand, it is sensitive to the initial conditions and the choice of the building criterion of the similarity graph is not trivial.

1.2 Decision tree

Given a set of data $X \subseteq \mathbb{R}^D$, a decision tree on X can be defined as a hierarchical, axis-aligned, binary partitioning of X [8]. It has a network structure, where each node represents a specific restriction of the initial dataset. The binarity of the tree means that each non-leaf node is linked to exactly two children and, except for the root node that represents the whole data space X, each node has only one father; in particular, the set of data corresponding to the father node splits into two subsets assigned respectively to the two children. The term axis-aligned means that each split of a father into two children is performed by imposing a threshold value on only one of the features; if we visualize the tree structure in a product space, this means that each cut is performed orthogonally to one of the axis. The union of the subspaces corresponding to the leaf nodes coincides with the initial space X and their intersection is an empty set. Because of its hierarchical structure, the decision tree can be used as hierarchical clustering method (with both top-down and bottom-up approach), by selecting a data-dependent splitting (or merging) criterion.

1.3 Clustering comparison

In order to compare different labelings referred to the same dataset, several clustering comparison measures have been introduced. They are used as both validation measures and tools to improve the quality of the clustering result [9].

Cluster validation consists of a collection of procedures that quantitatively evaluate the result of a clustering technique; in general, there are three approaches to investigate cluster validity [10]: the *internal* validation criteria entirely rely on quantities and features inherent to the dataset, the *external* ones are based on some preexisting knowledge about the structure of the data, while the *relative* ones evaluate the result by comparing different versions of the same algorithm obtained by setting different values of the parameters. Since clustering comparison measures estimate the similarity of two different classifications of the same dataset, if one of the two set of labels is the already known *ground truth* associated to the dataset, the corresponding measure is considered an external validation one. It thus evaluate how the predicted classification fits the true one.

On the other hand, clustering comparison is also used to improve the result of the clustering procedure; in particular, it occurs in the context of ensemble (consensus) clustering, whose aim is to unify a set of clusterings, already obtained by some algorithms, into a single high quality one [9]. Consequently, the corresponding measures quantify the information shared between two clustering results.

In the following chapters, two clustering comparison measures are considered: the Fowlkes-Mallows index and the adjusted mutual information. The first one is an external validation measure used to evaluate the performance of the clustering technique; the second one is used to compare different results, in our case obtained by iterating the same version of the algorithm (with the same choice of parameters) in order to extract additional informations from the classified dataset.

The *Fowlkes-Mallows index FMI* is a *pair counting* based measure; it evaluates the similarity of two data classifications by counting the point pairs on which two clustering results agree or disagree [11]. It is defined as the geometric mean of the precision and the recall [5]. In particular, we consider two sets of labels corresponding to the true classification and the predicted one; TP (True Positive) is the number of pairs of points belonging to the same class in both the true and the predicted classification, FP (False Positive) is the number of pairs of points belonging to the same class in the predicted classification but not in the true one and FN (False Negative) is the number of pairs of points belonging to different classes in the predicted classification but to the same class in the true one. The precision $p = \frac{TP}{TP+FP}$ is the fraction of predicted positive cases that are correctly true positives, while the recall $r = \frac{TP}{TP+FN}$ is the fraction of real positive cases that are correctly predicted positives [12]. The Fowlkes-Mallows index is thus formally defined as follows:

$$FMI = \sqrt{p \cdot r} = \frac{TP}{\sqrt{(TP + FP) \cdot (TP + FN)}} \tag{1.2}$$

The index ranges from 0 to 1 and it is equal to 1 if the two labelings completely overlap. In the next chapters, it is calculated through the `metrics.fowlkes_mallows_score` function of the Python `scikit-learn` package.

The *information theoretic* measures [9] [13] [5], as the Adjusted Mutual Information, are based on fundamental concepts of information theory and, in particular, on the concept of entropy. Consider two clustering results $\mathbf{U} = \{U_1, \ldots, U_R\}$ and $\mathbf{V} = \{V_1, \ldots, V_C\}$ of the same dataset, respectively consisting of R and C number of clusters, and imagine to transmit U on a communication channel; the entropy $H(\mathbf{U})$ of $\mathbf{U}$ represents the averaged quantity of

information required to encode the whole set of labels. The conditional entropy $H(\mathbf{U}|\mathbf{V})$ of $\mathbf{U}$ given $\mathbf{V}$ is instead the averaged quantity of information required to encode $\mathbf{U}$, when $\mathbf{V}$ is already known. Finally, the Mutual Information represents how much the knowledge of $\mathbf{V}$ reduces the quantity of information needed to encode $\mathbf{U}$ and is thus the difference of the entropy of $\mathbf{U}$ and the conditional entropy of $\mathbf{U}$ given $\mathbf{V}$. Since the general expression of the entropy related to $\mathbf{U}$ is $H(U) = -\sum_{i=1}^{R} p(i) \log p(i)$, where $p(i)$ is the probability that an object picked at random from $\mathbf{U}$ falls into U_i class, the entropy of $\mathbf{U}$, the joint entropy of $\mathbf{U}$ and $\mathbf{V}$, the conditional entropy of $\mathbf{U}$ given $\mathbf{V}$ and the mutual information of $\mathbf{U}$ and $\mathbf{V}$ are formally defined as follows:

$$H(\mathbf{U}) = -\sum_{i=1}^{R} \frac{a_i}{N} \log \frac{a_i}{N} \tag{1.3}$$

$$H(\mathbf{U}, \mathbf{V}) = -\sum_{i=1}^{R} \sum_{j=1}^{C} \frac{n_{ij}}{N} \log \frac{n_{ij}}{N} \tag{1.4}$$

$$H(\mathbf{U}|\mathbf{V}) = H(\mathbf{U}, \mathbf{V}) - H(\mathbf{V}) = -\sum_{i=1}^{R} \sum_{j=1}^{C} \frac{n_{ij}}{N} \log \frac{n_{ij}/N}{b_j/N} \tag{1.5}$$

$$I(\mathbf{U}, \mathbf{V}) = H(\mathbf{U}) - H(\mathbf{U}|\mathbf{V}) = \sum_{i=1}^{R} \sum_{j=1}^{C} \frac{n_{ij}}{N} \log \frac{n_{ij}/N}{a_i b_j / N^2} \tag{1.6}$$

N is the total number of points, a_i, b_j and n_{ij} are respectively the number of samples belonging to the i-th cluster of $\mathbf{U}$, belonging to the j-th cluster of $\mathbf{V}$ and in common to clusters U_i and V_j.

Since the mutual information is a symmetric measure ($I(\mathbf{U}, \mathbf{V}) = I(\mathbf{V}, \mathbf{U})$), it can be used as consensus similarity measure. Higher is its value, more the knowledge of one of the two clusterings helps to acquire information on the other one, meaning that more information is shared between the two sets of labels.

In the following chapters, we will use an adjusted version of the mutual information. Firstly, the measure is normalized, in order to obtain a result scaled between 0 and 1; in particular, if the normalized mutual information is equal to 1, the considered classifications exactly coincide, while, if it is equal to 0, they are completely uncorrelated. The advantage of treat with a normalized measure is that clustering results obtained from different labels are more easily comparable. Secondly, a correction for chance is introduced, in order to take into account the mutual information that entirely depends on random factors, without the presence of a real correlation between the two clustering results. This adjustement is needed in order to avoid the dependence of the information measure on the number of clusters and of samples, since I usually shows higher values for large dataset with a high number of classes. The *adjusted mutual information AMI* is defined as follows:

$$AMI = \frac{I(\mathbf{U}, \mathbf{V}) - \mathbf{E}[I(\mathbf{U}, \mathbf{V})]}{mean(H(\mathbf{U}), H(\mathbf{V})) - \mathbf{E}[I(\mathbf{U}, \mathbf{V})]} \tag{1.7}$$

where $mean(H(\mathbf{U}), H(\mathbf{V}))$ is the normalization factor and $\mathbf{E}[I(\mathbf{U}, \mathbf{V})]$ represents the adjustement for chance. It is the expectation value of the mutual information of two randomly generated clusterings, linked to have a fixed number of clusters and a fixed number of samples per cluster and it is defined as:

$$\mathbf{E}(\mathbf{U}, \mathbf{V}) = \sum_{i=1}^{R} \sum_{j=1}^{C} \sum_{n_{ij}=max(a_i+b_j-N,0)}^{min(a_i,b_j)} \frac{n_{ij}}{N} \log\Big(\frac{N n_{ij}}{a_i b_j}\Big) \frac{a_i! b_j! (N-a_i)! (N-b_j)!}{N! n_{ij}! (a_i - n_{ij})! (b_j - n_{ij})! (N - a_i - b_j + n_{ij})!} \tag{1.8}$$

In order to compute the adjusted mutual information of two sets of clustering results, the `metrics.adjusted_mutual_info_score` of the Python `scikit-learn` package is used.

Chapter 2

The Mondrian Process

The Mondrian process is a temporal stochastic process that recursively partitions the space; its partitioning procedure allows to generate a binary tree of subspaces, in which each edge corresponds to a split that divides the father subspace into two smaller children. At the end of the process, a partition of the initial space into several subspaces is obtained. If we define a set of points on the initial space that is progressively splitted, at the end of the process we will obtain, besides the space division into several subspaces, also the dataset division into smaller subsets, with similar hierarchy of partitions. Since the outcome of a hierarchical clustering algorithm run on a given dataset consists of a hierarchy of binary restrictions of the initial dataset (as explained in the first chapter), an adaptation of the Mondrian process as a clustering hierarchical algorithm can be developed. Rather than directly perform the division of the dataset, the Mondrian clustering algorithm will thus split the underlying space in which the dataset is defined; the division criterion must be adapted in order to depend on the data that we want to cluster.

In this chapter, the Mondrian process is formally defined and its main mathematical properties are briefly illustrated.

2.1 Definition

The Mondrian process is a recursive generative process, defined over an axis-aligned box $\Theta \subseteq \mathbb{R}^D$, that hierarchically partitions the underlying space through axis-aligned cuts [14]. Given an axis-aligned box $\Theta = [a_1, b_1] \times \ldots \times [a_D, b_D] \subseteq \mathbb{R}^D$, defined as the cartesian product of bounded intervals, the process generates two subspaces $\Theta_<$ and $\Theta_>$ by splitting Θ with some hyperplane orthogonal to one of the coordinate axis; then, it similarly generates independent splits on $\Theta_<$ and $\Theta_>$, creating new subspaces that are recursively cut until a certain condition is satisfied. The following scheme shows the generative process in detail [8]:

MONDRIAN(Θ,t_0):
1. $T \sim Exp(LD(\Theta))$
2. $d \sim$ Discrete$(p_1, ..., p_D)$ where $p_d \propto (b_d - a_d)$
3. $x \sim \mathcal{U}([a_d, b_d])$
4. $M_< \leftarrow$ MONDRIAN($\Theta_<$,$t_0 + T$) where $\Theta_< = \{z \in \Theta | z_d \leq x\}$
5. $M_> \leftarrow$ MONDRIAN($\Theta_>$,$t_0 + T$) where $\Theta_> = \{z \in \Theta | z_d \geq x\}$

return $(t_0 + T, d, x, M_<, M_>)$

The process is initialized by fixing the starting time t_0 and the initial axis-aligned box Θ that will be partitioned. Firstly, in line 1, the time T of the first cut is randomly generated from

an exponential distribution, with rate the linear dimension of the box $LD(\Theta) = \sum_{d=1}^{D}(b_d - a_d)$; this means that cuts in smaller boxes are expected to occur later than in bigger ones. The absolute time $t_0 + T$ at which the cut is generated is called birth time. Then, the dimension d of the cut is generated from a discrete distribution, with a probability proportional to the length $b_d - a_d$ of the corresponding interval (line 2), and the location of the cut is subsequently uniformly chosen in the interval $[a_d, b_d]$ (line 3).
At this point, the initial space Θ has been cut by an hyperplane orthogonal to the d-dimension and intersecting the $[a_d, b_d]$ interval in $d = x$, and it has been divided into two axis-aligned boxes $\Theta^{<}$ and $\Theta^{>}$. Lines 4 and 5 generate independent Mondrians on the two new subspaces, meaning that steps 1,2 and 3 are independently repeated on $\Theta_{<}$ and $\Theta_{>}$, starting from an initial time equal to the birth time of the last cut.
The process recursively splits the space in more and more refined partitions and stops when, for every independent Mondrian that has been generated, the birth time $t_0 + T$ of the last cut is higher than a fixed lifetime parameter λ. The lifetime λ represents a budget assigned to the process and, as cuts appear, it is progressively spent; it follows that the time interval T, that is generated at each iteration, represents the cost related to a specific cut.

The Mondrian process as temporal stochastic process

The formulation of Mondrian process as recursive generative process leads to an other definition as temporal stochastic process. More precisely, given an axis-aligned box $\Theta \subseteq \mathbb{R}^D$, the Mondrian process on Θ, denoted as $MP(\Theta)$, can be defined as a temporal stochastic process $(M_t)_{t>0}$ taking values in the ensemble of guillotine partitions of Θ and with distribution specified by the generative process MONDRIAN(Θ, t_0) [8]. If we define a guillotine partition as a hierarchical partition of Θ obtained by recursively splitting the space through hyperplanes orthogonal to the coordinate axes, the random variable $(M_t)_{t>0}$ is a guillotine partition of Θ formed by the cuts with birth time $t_b \leq t$.

Tree structure of the Mondrian process

The hierarchical structure of Mondrian process reflects the structure of a rooted binary k-d tree; each node is represented by the tuple $(t_0 + T, d, x, M_{<}, M_{>})$, which entries are the birth time, the dimension and the location of the cut and the two children of the node itself. Each non-leaf node has exactly two children and the ensemble of the leaf nodes represents the final and more refined partition of the initial space. The lifetime parameter λ of the stochastic process, that regulates the total number of cuts, can be interpreted as the depth parameter of the tree.

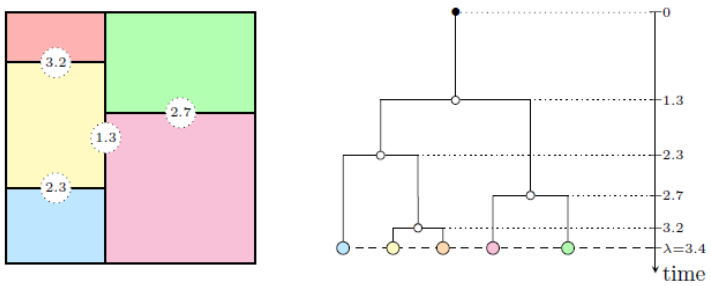


Figure 2.1: A Mondrian partition (left) with corresponding tree structure (right), which shows the evolution of the tree over time [15]

2.2 Properties of the Mondrian process

We firstly define a Poisson point process [16], through two equivalent definitions:

Definition 1 A Poisson point process is an arrival process[1] for which the interarrival times are independent and identically distributed random variables and in which the interarrival intervals have an exponential distribution function. This means that, for some $\lambda > 0$, called rate of the process, the interarrival times are represented by IID random variables $\{X_j : j = 1, 2, ...\}$ with the common density $f_X(t) = \lambda e^{-\lambda t}$, with $t \geq 0$.

Definition 2 An arrival process is a Poisson point process if its counting process $\{N(t); t > 0\}$ has the following properties:

- it is Poisson distributed $N(t) \sim Poisson(\lambda t)$
- $N(t_2) - N(t_1)$ has the same cumulative distribution function as $N(t_2 - t_1)$ for all $0 < t_1 < t_2$, that means the distribution of the number of events in some interval only depends on the size of the interval, and not on its starting point (**stationary increment property**)
- for any positive integer k and every k-tuple of times $0 < t_1 < t_2 < \cdots < t_k$, the random variables $N(t_1), N(t_2) - N(t_1), ..., N(t_k) - N(t_{k-1})$ are independent (**independent increment property**)

One-dimensional Mondrian process as Poisson point process

Consider a one-dimensional Mondrian process $MP(\Theta)$ with $\Theta = [a, b] \subseteq \mathbb{R}$ and with lifetime λ. Since, given n independent Poisson processes $N_1(t), ..., N_n(t)$ with rates $\lambda_1, ..., \lambda_n$, the combination of them $N(t) = N(t_1) + ... + N(t_n)$ is a Poisson point process with rate $\lambda = \lambda_1 + ... + \lambda_n$, and since the time intervals of $MP([a, b])$ are independent (for the memoryless property of the exponential) and generated from an exponential distribution with rate the length of the interval, the times of the cuts of the Mondrian process form a temporal Poisson process with rate $b - a$ (see **Definition 1**). It follows that the number of the times of the cuts in a time interval of length λ is $Poisson(\lambda(b - a))$ distributed.

Now consider the cut locations in the interval $\Theta = [a, b]$: as the process progresses, the cut locations are generated in each of the subintervals of Θ from a uniform distribution and, consequently, if we consider the ensemble of the cut locations generated in different subintervals, they are all uniformly generated in the whole interval Θ; moreover, as in the case of exponential distribution, also an extraction from a uniform distribution is independent from its past evolution. From these two properties, and since the number of the times of the cuts coincides with the number of the locations of the cuts and it has thus a $Poisson(\lambda(b - a))$ distribution with rate λ, the distribution of the cut locations of a one-dimensional Mondrian process run on $[a; b]$ with lifetime λ is a Poisson point process with constant intensity λ (see **Definition 2**).

Self-consistency

Consider a Mondrian process $MP(\Theta)$ over a box $\Theta = \Theta_1 \times \ldots \times \Theta_D$ and consider a smaller box contained within it, $\Phi_1 \times \ldots \times \Phi_D \subseteq \Theta_1 \times \ldots \times \Theta_D$. The stochastic process induced by $MP(\Theta)$ through its cuts intersecting the smaller box Φ is again a Mondrian process $MP(\Phi)$. This means that, considering all the cuts generated by the Mondrian process on Θ, the ones

[1] An arrival process is a sequence of increasing random variables, $0 < S_1 < S_2 < \cdots$, where $S_i < S_{i+1}$ means that $S_{i+1} - S_i$ is a positive random variable.

that don't intersect the subspace Φ don't affect the distribution of the cuts within Φ and this distribution is the same of that of a Mondrian process applied on Φ.

Mondrian slices Consider a subspace $\Phi \subseteq \Theta$, that is a cartesian product space of the following intervals: $\Phi_1 = \{x\}$ and $\Phi_d = \Theta_d$ for $d > 1$. Since the probability of having a cut in the $d = 1$ dimension is zero, no cut occurs in the first dimension. Consequently, for the self-consistency property, the restriction on the subspace Φ of a D-dimensional Mondrian process on Θ is a $(D-1)$-dimensional Mondrian process. In particular, if Φ is a one-dimensional subspace of Θ, the restriction of the Mondrian process on it is a one-dimensional process; since a one-dimensional Mondrian process is a Poisson point process, the location of the cuts of a D-dimensional Mondrian process, marginally considered in each dimension, follows a Poisson point process.

Extension on $\mathbb{R}^D$ A consequence of the self-consistency property is that the definition of the Mondrian process on an axis-aligned box can be relaxed by defining it on the entire $\mathbb{R}^D$. The structure of the Mondrian process on $\mathbb{R}^D$ is an infinitely deep tree with no root and infinite number of leaves.

Conditional Mondrians [8]

Consider again a Mondrian process $MP(\Theta)$ over a box Θ and the Mondrian process $MP(\Phi)$ over the restriction $\Phi \subseteq \Theta$. Given the Mondrian process over Φ, the property of self-consistency allows to have information on the unknown Mondrian process running on the whole Θ; in other words, it is possible to calculate the conditional distribution $MP(\Theta)$, given $MP(\Phi)$. In particular, conditionally given the restriction $MP(\Phi)$ of a Mondrian process $MP(\Theta)$ to a smaller box Φ, consider the first cut C^Φ in $MP(\Phi)$ and let t^Φ be its time. Then:

- with probability $exp(t^\Phi(LD(\Theta) - LD(\Phi)))$, the cut C^Φ is the first cut in Θ (it extends throughout Θ)
- with complementary probability $1 - exp(t^\Phi(LD(\Theta) - LD(\Phi)))$ the first cut in Θ misses Φ, its time has the truncated exponential distribution with rate $LD(\Theta) - LD(\Phi)$ and truncation at t^Φ, and the cut location is uniformly distributed along the segment where making a cut doesn't hit Φ.

Chapter 3

Clustering algorithm based on the Mondrian Process

The tree structure of the Mondrian process, that reflects the way in which the Mondrian process partitions the space, naturally leads to its adaptation to a clustering hierarchical algorithm. The adapted process, that we developed for this thesis, differs from the pure Mondrian process firstly because the splitting criterion must rely on some characteristics of the data that we want to cluster and, secondly, because the cuts are no more linked to be orthogonal to the axis but can in principle take any direction.

The Mondrian clustering algorithm consists of a two-step process. The first part has a top-down approach and is the one that directly exploits the Mondrian process; it partitions the space on which the set of data is defined by evaluating the similarity of groups of data and, consequently, favouring the cuts that separate subsets of points belonging to different clusters; for each leaf node, it assigns all the data contained in the corresponding subspace to the same class. In order to completely isolate the clusters, the number of cuts may be enough high to divide each single cluster into more than one subspace; consequently, a further procedure is required, in order to combining the subspaces that have been separated during the partitioning phase but that belong to the same cluster. The second part of the two-step process is thus a merging procedure and has a bottom-up approach; it starts from considering each final subspace (corresponding to a leaf node of the tree) as an independent cluster and hierarchically merges them on the basis of the same metric used in the previous splitting process.

After a brief mathematical description of how cuts and subspaces are defined, the Mondrian clustering algorithm is described; in particular, the partitioning and the merging phases are separately illustrated by making references to specific procedures of the implementation, that is written in the Phyton programming language. The metrics used to evaluate the similarity between groups of points are subsequently defined and, finally, the advantages of considering a forest (that averages the result of several trees) rather than a single tree are highlighted.

3.1 Description of hyperplanes and polytopes

If we consider a D-dimensional space Θ, an hyperplane is a $(D-1)$-dimensional subspace defined on Θ; this means that any hyperplane belonging to Θ cuts the underlying space into two D-dimensional subspaces. It can be described by a linear equation $a_1 x_1 + ... + a_D x_D = b$, that is written in vector form as $\boldsymbol{a} \cdot \boldsymbol{x} = b$; in particular, the vector $\boldsymbol{a}$ determines the direction orthogonal to the hyperplane. A generic hyperplane can be thus univocally characterized by the radius vector $\boldsymbol{a}$ with norm equal to 1 and by the scalar b, that represents the distance between the hyperplane and the origin of the axes.

As regards the mathematical description of Θ, any D-dimensional space processed by the algorithm is a convex polytope; a convex polytope is defined as the convex hull of a non-empty finite set of points belonging to a D-dimensional Euclidean space [17]. In particular, a convex hull is the smallest convex set containing the points, while a subset $C \subseteq \mathbb{R}^D$ is convex if $\lambda_1 x_1 + \lambda_2 x_2$ belongs to C for all $x_1, x_2 \in C$ and for all $\lambda_1, \lambda_2 \in \mathbb{R}$, with $\lambda_1 + \lambda_2 = 1$ and $\lambda_1, \lambda_2 \geq 0$. Since a closed convex D-dimensional set is bounded by a set of intersecting $(D-1)$-dimensional hyperplanes [18], a polytope can be described through its *half-space representation*, that means it is defined as the intersection of a finite number of half spaces. In particular, starting from the equation of the hyperplane that splits the whole space $\mathbb{R}^D$ into two half spaces, we can write each half space as a linear inequality: $a_1 x_1 + ... + a_D x_D \leq b$ and $a_1 x_1 + ... + a_D x_D \geq b$. A polytope is thus described as a system of linear inequalities, and the number m of inequalities is equal to the number of sides of the polytope:

$$\begin{cases} a_{1,1} x_1 + ... + a_{1,D} x_D \leq b_1 \\ ... \\ a_{m,1} x_1 + ... + a_{m,D} x_D \leq b_m \end{cases}$$

The system of inequalities can be written in matrix form $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$; a polytope is thus characterized by a $m \times D$ matrix $\mathbf{A}$ and a m-dimensional vector $\mathbf{b}$.

As said before, any $(D-1)$-dimensional hyperplane intersecting a D-dimensional space Θ cuts the space itself into two D-dimensional subspaces $\Theta_<$ and $\Theta_>$; if Θ is a convex polytope, the two subspaces are polytopes as well, with the hyperplane representing the face that $\Theta_<$ and $\Theta_>$ have in common, and the convexity property holds on both of them.

In order to perform geometric operations on generic-dimensional polytopes, the `Polytope` package of Python is used [19]. In particular, a specific polytope is described by the class `polytope.Polytope`, that requires as input two arrays representing the matrix $\mathbf{A}$ and the vector $\mathbf{b}$.

3.2 Mondrian clustering tree

In this section, we will describe how the unsupervised tree based on the Mondrian process performs the clustering on a set of points. Besides separating the whole dataset $\boldsymbol{X}$ into a certain number of groups, each one corresponding to a specific class, the algorithm univocally associates a class to each point of the space Θ in which the dataset is defined. In particular, albeit the cutting criterion is exclusively based on the structure of the dataset, the splitting procedure doesn't involve only the dataset, but is applied to the whole Θ; consequently, at the end of the whole process, each subset of points corresponds to a specific subspace of Θ, in which is contained. Every class, that represents a particular cluster of the data, is thus assigned not only to a specific subset of samples, but also to its corresponding subspace. This means that, even if the clustering procedure is exclusively performed on the basis of the informations derived from $\boldsymbol{X}$, if we consider an other point not belonging to the dataset but defined on Θ, we are able to assign it to a specific cluster.

3.2.1 Partitioning phase

The partitioning of the space is the phase of the method that directly involves the adaptation of the Mondrian process. Analogously to what was described in Chapter 2, once the initial space Θ is defined, it is recursively splitted by hyperplanes; the recursivity consists on the fact that the same splitting criterion is independently applied to the different subspaces that are progressively generated. In particular, each subspace is splitted by a single hyperplane and is consequently divided into two smaller subspaces, that will be similarly splitted in the following iterations.

Since the goal of the algorithm is to find out a classification of the given dataset that reflects some intrinsic properties of the data, the first difference from the pure Mondrian process is that both the definitions of the initial space and the cut choice depend on the input unlabeled data $\boldsymbol{X} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_n) \in \mathbb{R}^D$. In particular, since we perform a cut in order to separate groups of points that have a high within similarity and a low between similarity, the method to choose the cut must be related to a metric that evaluates the similarity between points. An other important peculiarity of this implementation is that the cuts are no more bound to take a specific direction of the space, differently from the pure Mondrian process, that only considers axis-aligned cuts. Before the splitting procedure begins, both the initial space and the ensemble of possible cutting hyperplanes are defined.

Initial space Given a dataset $\boldsymbol{X} \in \mathbb{R}^D$, the initial space is defined as any convex subspace of $\mathbb{R}^D$ containing $\boldsymbol{X}$. The convexity property is required in order to guarantee some space features that are necessary to apply the Mondrian process. Firstly, since the Mondrian process performs a binary splitting of the space, each space, cut by a hyperplane, has to be divided into no more than two subspaces, and this property holds only if the space is convex. Moreover, the convexity property is preferred because it is related to other features that make the mathematical description of the space simpler (like, for example, the verification that a point belongs to a certain space); in addition, since the goal of the algorithm is to assign a probability to belong to a certain class to each point of the space, every subspace that will be considered must contain a subset of the initial dataset; in case of concave spaces, this is not guaranteed, with the consequence that it may be not possible to assign a class to some regions of the space. Finally, the fact that a hyperplane cutting a convex polytope generates two convex polytopes allows to recursively apply the Mondrian process under the same conditions.

In our cases, the initial space is defined as the smaller axis-aligned box Θ that contains $\boldsymbol{X}$; it is thus a particular case of convex polytope. Since polytopes, generated by cutting a convex polytope with an hyperplane, are convex as well and since the splitting procedure of the Mondrian process is performed through hyperplanes, all the subspaces created in the partitioning procedure are convex polytopes (and, in case of axis-aligned cuts, all the subspaces are axis-aligned boxes). This guarantees the iterative applicability on each subspace of the splitting procedure.

The initial box Θ is described by defining the input parameters $\mathbf{A}$ and $\mathbf{b}$ of `polytope.Polytope`. As said in the previous section, each row of $\mathbf{A}$ represents the radius vector (with norm equal to one) that indentify the orthogonal direction to the corresponding hyperplane, while each element of the $\mathbf{b}$ vector represents the distance between the origin and the hyperplane. Since each face of an axis-aligned box is orthogonal to a specific axis, for each i-coordinate axis, two radius vectors (rows of $\mathbf{A}$) with corresponding magnitudes (elements of $\mathbf{b}$) are defined. The first vector (pointing towards the positive direction) consists of all zero elements, except the i-one that is equal to 1; the second one points towards the negative direction and has all zero elements except the i one, equal to -1. The corresponding magnitude elements of $\mathbf{b}$ depend on the data: considering the ensemble of i-coordinates of all the points, the distance h between

the maximum and the minimum element is computed; then, the first positive radius vector is associated to the maximum coordinate increased of $0.05h$, while the second negative radius vector is associated to the minimum coordinate value decreased of $0.05h$.

Ensemble of hyperplanes As said before, differently from the pure Mondrian process case, the cuts are no more constrainted to be orthogonal to the coordinate axis, but can in principle take any direction of the space. In fact, considering that cuts are performed in order to divide different clusters, it may happen that the more suitable direction of the cut, for a given dataset shape, is not that orthogonal to one of the axis; since the dataset is unknown, we prefer not to be linked to specific direction when splitting the space and the condition of orthogonality is thus relaxed.

The ensemble of all possible cutting hyperplanes intersecting the space Θ is restricted to a number of hyperplanes equal to the number of sample pairs of the dataset; in particular, an hyperplane is associated to each pair of points $\boldsymbol{x}_i, \boldsymbol{x}_j \in \boldsymbol{X}$ and it orthogonally intersects the segment connecting the two samples in its middle point. More precisely, the ensemble of hyperplanes from which the cut is extracted is created by applying the followingsteps to each pair of points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$:

1. $\boldsymbol{m}_{ij} = (m_{ij,1}, \ldots, m_{ij,D})$ with $m_{ij,d} = (x_{i,d} + x_{j,d})/2 \ \forall d = 1, \ldots, D$
   midpoint
2. $\boldsymbol{v} = (v_1, \ldots, v_D)$ with $v_d = x_{i,d} - x_{j,d} \ \forall d = 1, \ldots, D$
   vector orthogonal to the hyperplane
3. $\boldsymbol{n} = \boldsymbol{v}/d$ with $d = dist(x_i, x_j)$
   single-norm vector orthogonal to the hyperplane
4. $h = \boldsymbol{n} \cdot \boldsymbol{m}_{ij}$
   hyperplane distance from the origin of the axis
5. $h_k = \boldsymbol{n} \cdot \boldsymbol{x}_k - h \ \forall \boldsymbol{x}_k \in \boldsymbol{X}$
   distance between the hyperplane and every point of the dataset

Firstly, for each pair of samples, their midpoint $\boldsymbol{m}_{ij}$ is calculated, in line 1. Then, the hyperplane perpendicular to the segment connecting them is computed; it is determined by its orthogonal vector $\boldsymbol{n}$ with norm equal to 1, that is parallel to the segment connecting the two points. Each vector coordinate is calculated by subtracting the two corresponding coordinates of the points (line 2) and normalizing the result to their euclidean (line 3). The distance h between the cutting hyperplane and the origin of the axis is then computed, in line 4, through the scalar product of its single-norm orthogonal vector and the cut point, that is the midpoint of the two considered samples.

The last step performed before the splitting process starts is the computation of the distance h_k between each hyperplane and every point of the dataset. It is performed by firstly computing the scalar product between the orthogonal vector determining the hyperplane and the considered point; then the point-hyperplane distance is calculated by subtracting the distance between the hyperplane and the origin of the axis to the obtained result. The point-hyperplanes distances will be useful during the assignation of subsets of points to the corresponding subspaces obtained after the splitting procedure.

In the next chapter, we will evaluate the differences between outcomes of the algorithm when cuts are forced to be orthogonal to the coordinate axis and when they are generated through the just described procedure. In case of orthogonal cuts, the ensemble of hyperplanes is created by firstly considering the projections of each point $\boldsymbol{x} \in \boldsymbol{X}$ on the coordinate axis. For each dimension d, an hyperplane is associated to each pair of consecutive projections: it is orthogonal to the corresponding d coordinate axis and intersects the segment connecting the two projection points in its middle point.

Recursive generative process Once the initial polytope and the ensemble of possible cutting hyperplanes are defined, the space is recursively splitted, similarly as in the pure Mondrian process. Following the scheme presented in Section 2.1, the new data-dependent Mondrian recursive generative process is shown below:

DATA-DEPENDENT-MONDRIAN($\boldsymbol{X}$,Θ,t_0):
1. $T \sim Exp(\text{vol}(\Theta(\boldsymbol{X})))$
2. $h \sim \text{Discrete}(h_1, \ldots, h_n)$ with probabilities of extraction $p_1, \ldots, p_n$
3. definition of $\Theta_<$, $\Theta_>$ with $\Theta_< \cup \Theta_> = \Theta$
4. $\boldsymbol{X}_<$, $\boldsymbol{X}_>$ assigned to $\Theta_<$, $\Theta_>$, with $\boldsymbol{X}_< \cup \boldsymbol{X}_> = \boldsymbol{X}$
5. $M_< \leftarrow$DATA-DEPENDENT-MONDRIAN($\boldsymbol{X}_<, \Theta_<, t_0 + T$)
6. $M_> \leftarrow$ DATA-DEPENDENT-MONDRIAN($\boldsymbol{X}_>, \Theta_>, t_0 + T$)

return $(t_0 + T, \Theta_<, \Theta_>, \boldsymbol{X}_<, \boldsymbol{X}_>, M_<, M_>)$

Firstly, in line 1, the time of the cut is generated from an exponential distribution with rate the volume of the initial space Θ. For the random extraction, the `random.exponential` function of `numpy` is used; it requires as input the scale parameter, that is the inverse of the rate; the volume of the polytope is computed through the `polytope.Polytope.volume` function. The dependence on the volume is defined in analogy with the pure Mondrian process, in which the rate of the exponential distribution is represented by the linear dimension; in particular, the linear dimension is a measure of the size of the axis-aligned box, as the volume represents the size of the polytope. In both cases, therefore, the cut is expected to be sooner in larger polytopes than in smaller ones.

In line 2, the cutting hyperplane is randomly generated from the previously defined ensemble of possible cuts, through the `numpy.random.choice` function; each hyperplane h_i is characterized by a specific probability of extraction p_i. In order to associate a probability value to each possible cut, a metric, that estimates the similarity of the two groups of samples belonging to Θ and divided by the cut, is considered; once the metric value is computed, the probability is set equal to its normalized value. The metrics used to evaluate the similarity of groups of different samples are presented in Section 3.2.3; as the value of the metric increases, the corresponding two groups of points are decreasingly similar and the probability to extract the corresponding hyperplane increases as well. Depending on the distribution of the normalized metric values over all the possible hyperplanes, it could be necessary to raise the metric to a certain power exp, before the normalization. This happens, for example, in presence of few cut with high probability and a large number of cuts with similar and low probability; if the number of low probability cuts is high enough, the high probability hyperplanes are no more favoured in the extraction. This implies that the choice of the cut is more randomized and the partitioning doesn't take into account the similarity evaluation between different groups of points. In order to favour the extraction of cuts corresponding to a higher metric value, the metric is thus raised to a certain power $exp > 1$.

Once the hyperplane has been extracted, the two new subspaces generated after the splitting are formally defined through the `polytope.Polytope` object (line 3); this requires the determination, for each subspace, of the input parameters $\mathbf{A}$ and $\mathbf{b}$. As explained in Section 3.1, any polytope is described by the inequality $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$, representing the intersection of half spaces; each row of $\mathbf{A}$ is the orthogonal radius vector of a specific face and each element of $\mathbf{b}$ is the face distance from the origin of the axis. Since the initial space Θ has been cut into two parts by the previously generated hyperplane, each new subspace is the result of the intersection between Θ and one of the two half spaces defined by the hyperplane. Consequently, in order to find the new $\mathbf{A}$ and $\mathbf{b}$ to assign to each new polytope, a row, representing the new

intersecting half space, has to be added to the inequality that characterizes Θ. More precisely, if $\boldsymbol{a}_{cut}$ is the radius vector of the new hyperplane and b_{cut} is its distance from the origin, the inequalities $\boldsymbol{a}_{cut} \cdot \boldsymbol{x} \leq b_{cut}$ and $-\boldsymbol{a}_{cut} \cdot \boldsymbol{x} \leq -b_{cut}$ have to be added to those of Θ, in order to respectively define the two new subspaces $\Theta_<$ and $\Theta_>$.

Besides the splitting of Θ into $\Theta_<$ and $\Theta_>$, the introduction of the cut determines the division of the set of points $\boldsymbol{X} \in \Theta$ into two subsets. In line 4, the two subsets of points $\boldsymbol{X}_<$ and $\boldsymbol{X}_>$ are associated to the corresponding subspaces $\Theta_<$ and $\Theta_>$. The identification of the two groups is performed by considering their distance from the cutting hyperplane, computed before the extraction of the time of the cut; in particular, all the points locating in the same half space, defined by the hyperplane, are characterized by distances from the hyperplane itself with the same sign. It is thus possible to determine the two subsets, separated by the hyperplane, by grouping them on the basis of their distance sign. Once the determination of the two subsets $\boldsymbol{X}_<$ and $\boldsymbol{X}_>$, they have to be assigned to the corresponding subspaces. In particular, in order to check if a point is contained in a polytope, an expression already implemented in the `polytope` package is used; if `x` is the list of coordinates of the considered point and `p = polytope.Polytope(A,x)` is the Python object identifying the polytope, the sintax of this expression is `x in p` and its output is boolean. In our case, once the two objects $p_<$ and $p_>$ corresponding to the new polytopes are created (line 3), a single point $\boldsymbol{x} \in \boldsymbol{X}$ is tested through the just mentioned expression, referring to one of the two polytopes; if the point belongs to the polytope, all the points characterized by the same distance sign are assigned to it and those with opposite sign are assigned to the other polytope; if the point doesn't belong to the considered polytope, the assigment is opposite. The polytope-hyperplane matching is an essential step in the partitioning phase because it allows to correctly apply the recursive process to the previously created subspaces, since the criterion used to split the polytope is based on the data defined on it. Also in the case of leaf polytopes, that will not be splitted since they represent the final partition of the whole space, the matching allows to associate each subset and its corresponding subspace to the same cluster; this is important in the case of assigment of a point, that doesn't belong to the clustered subset, to a class.

Finally, as in the pure Mondrian process, in lines 5 and 6, the previous steps are independently repeated over the two new subspaces.

Result of the partitioning process and considerations The whole process stops when, for each split that is independently generated on different subspaces, the characteristic absolute time of the cut is higher than the lifetime parameter λ (as in the pure Mondrian process), or when the set of points $\boldsymbol{X}$ contained on the considered polytope consists of only two points. The output of the algorithm keeps track of the history of the space splitting, meaning that it stores, at each iteration of the recursive process, the time of the cut, the polytope object that is splitted and its two children, with their corresponding subsets of points; this allow the program to reconstruct the full process and obtain the full hierarchical structure of the splitting.

As mentioned before, the Mondrian hierarchical structure is that of an unsupervised decision tree, whose ensemble of leafs represents the final and more refined partition of the initial space and dataset. The main difference in the splitting procedure, between the Mondrian and the decision trees, consists in the different cutting directions, since the second one performs axis-aligned cuts, by imposing, at each step, a threshold on each considered feature.

The tree structure reflects also on the fact that the evaluation of the similarity within and between groups of points, during the partitioning procedure, is carried on locally: more precisely, since, in order to perform the cut, only the data belonging to the considered subspace are involved in the similarity estimation, only in the case of the first split, the whole dataset is considered; as the splitting process goes on, the similarity evaluation is made on smaller and smaller subspaces and data subsets until, at the beginning of the merging procedure, each

subspace and the corresponding subset of data are assigned to the same class.

Figure 3.1 shows an example of one possible output of the partitioning phase run on a 2D dataset consisting of two circular clusters. It is obtained by hierarchically splitting the initial space through 15 cuts and the final partition consists of 16 adjacent polygones. Each subspace is associated to its characteristic number.

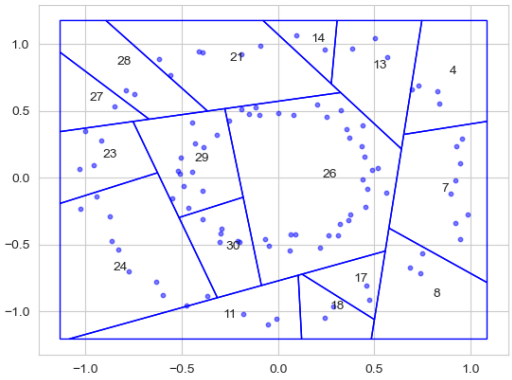


Figure 3.1: Final output of the partitioning phase run on a dataset consisting of two concentric circles. Each polygon represents a leaf node of the corresponding tree and contains only data belonging to the same cluster; it is identified by a number.

3.2.2 Merging phase

As described in the previous section, the output of the partitioning phase consists of a division of the dataset and its underlying space respectively into subsets and subspaces. Since the aim of the method is to separate different clusters, each final subspace is considered to contain only data belonging to the same class. Depending on the dataset shape, in order to completely separate the clusters, more than one cut may be required and, if the number of cuts is high enough, at the end of the partitioning process, each cluster is divided into more than one subspace. A further procedure is thus needed, in order to collect the more similar subspaces into clusters and assign them to the same class.

The merging procedure is performed by hierarchically evaluating the similarity of adjacent polytopes; the similarity metric used is the same of the partitioning phase while, in regards to the evaluation of adjacency, a list of neighboring subspaces is associated to each leaf polytope as preliminary computation. The measure is restricted to adjacent subspaces because we expect that, if two polytopes are not closed, the corresponding subsets of points are dissimilar (since the similarity is based on the concept of euclidean distance). In order to do that, the `polytope.is_adjacent` function is used; it requires as input two polytopes objects and it checks if they are adjacent, providing a boolean output. Every pair of leaf subspaces is tested and a list of neighbors is assigned to each polytope.

A bottom up approach is used; this means that, firstly, no more than one subspace is assigned to the same cluster and, since all the points enclosed in the same subspace belong to the same class, the number of clusters is equal to the number of subspaces. The initial state is described by an undirected network G with number of nodes equal to the number of subspaces and without edges; each node represents a specific leaf polytope. One by one, edges are added to the network. The presence of an edge connecting two nodes means that two subspaces are merged, consequently assigning the corresponding groups of data to the same class; only two nodes corresponding to adjacent polytopes are thus allowed to be connected with an edge. At

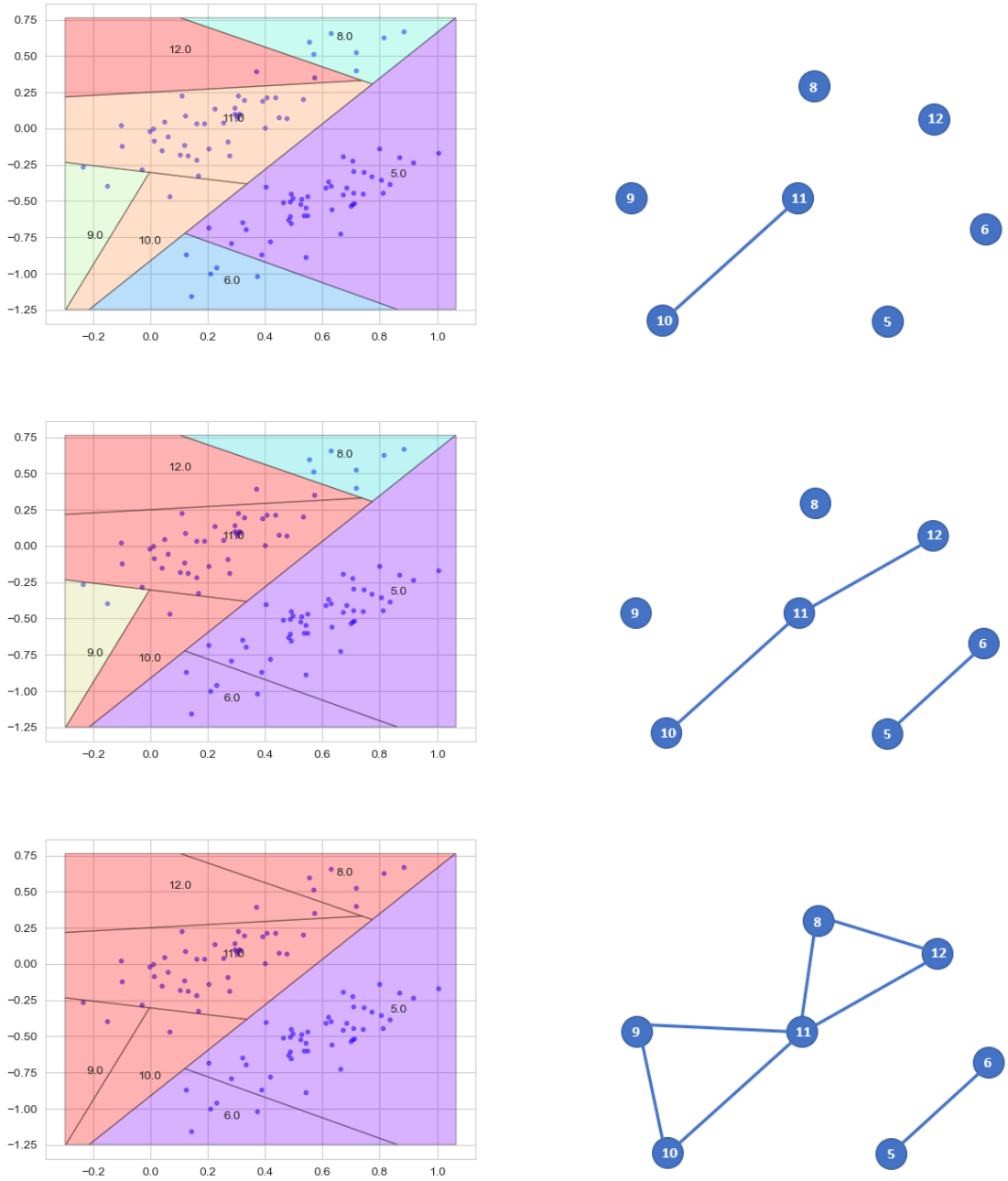


Figure 3.2: Three steps of the merging process, corresponding to a division of the two cluster dataset into 6, 4 and 2 clusters. On the left, the partitioned space is shown, with the subspaces belonging to the same class characterized by the same color; on the right, the corresponding undirected network describing the state of the clusterization. Each node of the graph represents a subspace and an edge between two nodes indicates that the corresponding subspaces are merged; the process starts by considering all the subspaces as separate clusters and progressively joins them. The number of clusters is equal to the number of connected components of the graph.

each step, the total number of custers is equal to the number of connected components of the network. In order to treat with graph objects, the `networkx` package is used.

The order in which the edges are added to the network is determined by assigning a score to every possible edge between neighbouring subspaces; it represents a similarity measure between adjacent groups of data and is the same metric used to partition the space in the first part of the algorithm (see Section 3.2.3 for the description of the considered metrics). With the decreasing of the metric value, two subspaces are considered more and more similar, with consequent higher probability to belong to the same cluster. After the scores of all the possible edges are calculated, the edge with lower score is added to the network, indicating that the two subspaces corresponding to the nodes linked by the edge are merged. The lists of neighbors, corresponding to the polytopes that have been combined, are merged as well and the new list is assigned to the subspace resulting from the merging; the lists of its new neighbouring subspaces are equally modified. Finally, the metric is again calculated for the pairs of new neighbouring subspaces and the score list is updated.

Because the metrics considered in Section 3.2.3 compare two groups of data with numerosity higher than one, they can't be computed if at least one of the two polytopes contains only one point. In order to avoid this issue, before creating the initial network G, each subspace with single data is merged to the corresponding nearest neighbouring subspace containing more than one data, by using a different metric (described in the same section).

Figure 3.2 shows three steps of the merging process applied to a partition of a dataset consisting of two clusters elongated in the diagonal direction.

3.2.3 Metric choice for splitting and merging subspaces

Since a way to estimate the similarity of two groups of data is to evaluate how they are spatially separated, the chosen metrics are functions of the euclidean distance *dist*. Lower is the value of the metric, more similar are the two groups of points; for this reason this value can be used as probability of extraction of the corresponding cut (higher is the value of the metric, less similar are the two groups of data and higher is the probability to extract the hyperplane). The considered metrics are described in the following.

Variance based metric

1. $\boldsymbol{d} = dist(\boldsymbol{X})$, $\boldsymbol{d}_< = dist(\boldsymbol{X}_<)$, $\boldsymbol{d}_> = dist(\boldsymbol{X}_>)$
2. $v = var(\boldsymbol{d})$
3. $v_{<,>} = var(\boldsymbol{d_<} \cup \boldsymbol{d_>})$
4. $metric = v/v_{<,>}$

This metric is based on the computation of the variance, that is an estimate of the dispersion of a set of measurements from their average value. The dispersion is separately computed on the whole dataset and on the subset of points that may be divided by the cut; a great decrease of the dispersion value, after the space splitting, means that the internal averaged distance between samples of each of the two final groups is lower than those of the initial unified one and this is probably due to the presence of an empty space between the two groups. The variance is mathematically defined as follows:

$$var(d_1, \ldots, d_n) = \frac{\sum_{i=1}^{n}(d_i - \bar{d})^2}{n} \tag{3.1}$$

where $d_1, \ldots, d_n$ are the considered measurements and $\bar{d}$ is their mean value.

Firstly, the set of points $\boldsymbol{X}$ belonging to the initial space and the two subsets $\boldsymbol{X}_<$ and $\boldsymbol{X}_>$ created after the splitting are considered. In line 1, the pairwise distances between points within the three sets are computed. Later, in step 2 and 3, we calculate the variance of the set of distances respectively corresponding to $\boldsymbol{X}$ and to the union of $\boldsymbol{X}_<$ and $\boldsymbol{X}_>$. Finally, the metric is set equal to the ratio of the two variances.

Centroid based metric

1. $c = C(\boldsymbol{X})$, $c_< = C(\boldsymbol{X}_<)$, $c_> = C(\boldsymbol{X}_>)$
2. $\boldsymbol{d} = dist(c, \boldsymbol{X})$, $\boldsymbol{d}_< = dist(c_<, \boldsymbol{X}_<)$, $\boldsymbol{d}_> = dist(c_>, \boldsymbol{X}_>)$
3. $\bar{d} = mean(\boldsymbol{d})$, $\bar{d}_{<,>} = mean(\boldsymbol{d}_< \cup \boldsymbol{d}_>)$
4. $metric = \bar{d} - \bar{d}_{<,>}$

The concept of this metric is similar to that of the previous one, since we calculate the dispersion of the dataset before and after the possible cut; in this case, we use a different estimator of the dispersion and, more precisely, the averaged distance of the samples from their centroid. Firstly, in line 1 we calculate the centroid for each of the three sets of points $\boldsymbol{X}$, $\boldsymbol{X}_<$ and $\boldsymbol{X}_>$; the centroid C of a set of points is defined as the arithmetic mean of the point positions. Then, in line 2, the distances between the centroid and all the samples within each set are considered; in line 3, we compute the mean of the distances of the whole set $\boldsymbol{X}$ and the mean of the union of the distances corresponding to the two subsets $\boldsymbol{X}_<$ and $\boldsymbol{X}_>$. The metric is finally defined as the difference between the two averaged values.

Minimum distance based metric

1. $\boldsymbol{d} = dist(\boldsymbol{X}_<, \boldsymbol{X}_>)$
2. $d_{min} = min(\boldsymbol{d})$
3. $\boldsymbol{d}_{j,i} = dist(\boldsymbol{x}_i, \boldsymbol{X}_j)$ $\forall \boldsymbol{x}_i \in \boldsymbol{X}_j$ where $j = <, >$
4. $d_{j,i} = \min(\boldsymbol{d}_{i,j})$ $\forall \boldsymbol{x}_i \in \boldsymbol{X}_j$ where $j = <, >$
5. $\bar{d} = mean(d_{<,1}, \ldots, d_{<,n}, d_{>,1}, \ldots, d_{>,m})$
6. $metric = |d_{min} - \bar{d}|$

In this case, the base concept is different from the two previous ones since we don't calculate the same quantity before and after the cut, but we directly consider the situation after the splitting. In particular, we evaluate how neighbouring points are close in the two separate groups and we compare the values obtained with minimum separation between the two groups; if the distance between groups is comparable with the distances that separe neighbouring samples, the two groups are considered similar.

Firstly, in line 1 and 2, we calculate the minimum distance between two points respectively belonging to different subspaces. Then, for each sample in $\boldsymbol{X}_<$ and $\boldsymbol{X}_>$, in line 3 the distances from all the other points enclosed in the same subspace are computed and, in line 4, the minimum distance value is taken. The mean value of the minimum distances is considered, in line 5. The metric is finally set equal to the absolute value of the difference between the minimum distance between the groups and the mean of the minimum distances within the groups.

Minimum distance based metric with correction

1. $\boldsymbol{d} = dist(\boldsymbol{X}_<, \boldsymbol{X}_>)$
2. $d_{min} = min(\boldsymbol{d})$; the corresponding nearest points are $(\boldsymbol{x}_{<,1}, \boldsymbol{x}_{>,1}) \in \boldsymbol{X}_< \times \boldsymbol{X}_>$

3. $\boldsymbol{d}_{j,i} = dist(\boldsymbol{x}_i, \boldsymbol{X}_j)\ \forall \boldsymbol{x}_i \in \boldsymbol{X}_j$ where $j = <, >$
4. $d_{j,i} = \min(\boldsymbol{d}_{i,j})\ \forall \boldsymbol{x}_i \in \boldsymbol{X}_j$ where $j = <, >$
5. $\bar{d} = mean(d_{<,1}, \ldots, d_{<,n}, d_{>,1}, \ldots, d_{>,m})$
6. $\boldsymbol{x}_{<,2}, \boldsymbol{x}_{>,2}$, with $dist(\boldsymbol{x}_{<,1}, \boldsymbol{x}_{<,2}) = \min(dist(\boldsymbol{x}_{<,1}, \boldsymbol{X}_<))$ and $dist(\boldsymbol{x}_{>,1}, \boldsymbol{x}_{>,2}) = \min(dist(\boldsymbol{x}_{>,1}, \boldsymbol{X}_>))$
7. $d_{min,<} = \min(dist(\boldsymbol{x}_{<,2}, \boldsymbol{X}_>))$, $d_{min,>} = \min(dist(\boldsymbol{x}_{>,2}, \boldsymbol{X}_<))$
8. $metric = |d_{min} - \bar{d}| + d_{min,<} + d_{min,>}$

In the last case, the minimum distance based metric is modified with a correction. The first five steps of the scheme are exactly the same of the previous metric, with the difference that, in line 2, the nearest points $\boldsymbol{x}_{<,1}$ and $\boldsymbol{x}_{>,1}$ that belong to different subspaces are identified. In line 6, for each of these two nearest points, its closest point belonging to the same subset is considered; more precisely, $\boldsymbol{x}_{<,2}$ is the nearest point to $\boldsymbol{x}_{<,1}$ belonging to $\boldsymbol{X}_<$ and $\boldsymbol{x}_{>,2}$ is the nearest point to $\boldsymbol{x}_{>,1}$ belonging to $\boldsymbol{X}_>$. In line 7, for both $\boldsymbol{x}_{<,2}$ and $\boldsymbol{x}_{>,2}$, the minimum distance between them and a point of the opposite subset is computed. The correction to add to the previously defined minimum distance metric is the sum of these two distances. Figure 3.3 shows a scheme describing the lengths considered in the computation of this metric. The correction, as will be explained in Chapter 4, is a way to take into account inaccuracies in the partitioning phase.

Metric for single data merging

1. $\boldsymbol{d} = dist(\boldsymbol{x}_<, \boldsymbol{X}_>)$, with $\boldsymbol{x}_< \equiv \boldsymbol{X}_<$
2. $d_{min,1} = min(\boldsymbol{d})$; the nearest points are $\boldsymbol{x}_<$ and $\boldsymbol{x}_{>,1}$
3. $\boldsymbol{x}_{>,2}$ with $dist(\boldsymbol{x}_{>,1}, \boldsymbol{x}_{>,2}) = \min(dist(\boldsymbol{x}_{>,1}, \boldsymbol{X}_>))$
4. $d_{min,2} = dist(\boldsymbol{x}_<, \boldsymbol{x}_{>,2}))$
5. $metric = d_{min,1} + d_{min,2}$

As explained in Section 3.2.2, before the merging procedure, each subspace containing only one sample is merged with the nearest one with higher numerosity; the metric used in the merging is the equivalent of the minimum distance metric with correction, with a variation in order to take into account that one of the two groups consists of a single point (that, in this notation, is $\boldsymbol{X}_<$). The minimum distance between the two subsets is considered (step 1 and 2), with $\boldsymbol{x}_{>,1}$ the nearest point to the single data set. In line 3, the closest point to $\boldsymbol{x}_{>,1}$ is considered and, in line 4, its distance with the point belonging to the other subspace is computed. The metric is finally set equal to the sum of the two previously computed distances.

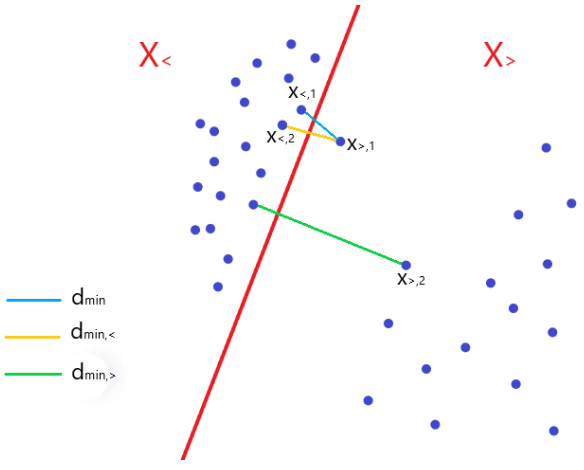


Figure 3.3: Points and segments considered in the calculation of the minimum distance based metrics; $\boldsymbol{X}_<$ and $\boldsymbol{X}_>$ are the two sets of data separated by the red line.

3.3 Ensemble prediction using multiple process

Given a dataset $\boldsymbol{X} \in \Theta$, the output of the Mondrian clustering tree run on $\boldsymbol{X}$ consists of an ensemble of partitions of Θ. Each partition is characterized by a specific number of clusters and the subspace classification is performed such that each point of Θ is univocally associated to a cluster. Since the partitioning phase is randomized, at fixed number of clusters, it is possible that outcomes of different iterations of the algorithm exhibit different subspace shapes, even in the case that the corresponding data classifications exactly coincide. In order to improve the result of the clustering procedure, an unsupervised Mondrian forest, that averages the results of different trees, is considered. The advantages of considering the forest are, besides the limitation of the effect of possible misclassifications (resulting from the averaging process), the estimation of a probability distribution of belonging to each cluster, associated to each point of the space, and the capability to automatically recognize the correct number of clusters in which the dataset is divided.

Firstly, the main difference between the forest and the tree outcomes is that the forest result associates, to each point of the space, a measure of the probability to belong to a certain cluster. In particular, a specific point is no more univocally related to a single cluster but to more than one, depending on the specific output of the tree belonging to the forest; the number of times that a point is assigned to the same class gives an estimation of the probability that the point belongs to that cluster. The advantage of treating with probability distributions is that we are able to evaluate the uncertainty that characterizes the attribution of the class and, in particular, the variation of uncertainty among different regions of the space. Figure 3.4, on the left, shows a visualization of the averaged output of 10 Mondrian trees in case of a dataset consisting of two interleaving half circles. The different classifications of the space into two clusters are overlapped and the two regions characterized by the lighter and the darker colors correspond to a perfect agreement of all the tree results; in particular, the region characterized by a lighter color has a probability $p_1 = 1$ to belong to the first class and a probability $p_2 = 0$ to belong to the second class and, on the contrary, the darker region has $p_1 = 0$ and $p_2 = 1$. The region between the two clusters is characterized by a range of intensities between the lighter and the darker ones and is thus associated to different values of probabilities in the interval $[0, 1]$; a point in that region is thus not univocally associated to one of the two classes but to both of them with different probabilities. The variation of the probability of belonging to a class, depending on the location, reflects the uncertainty on the evaluation of some regions of the space, that is due to the fact that we can not have exact informations about the regions in which there are no data. Moreover, the introduction of the probability allows to recognize and eventually reject the outputs of the trees that misclassify the data (since they are expected to be the less probable).

Besides the intrinsic randomization of the partitioning process, due to the formulation of the cut random extraction, with probabilities proportional to the normalized metric values, the level of determinism of the extraction procedure is also influenced by the power to which the metric is raised. In particular, with the increase of the exponent, the more the probability values, assigned to the different hyperplanes to be used to divide the polytope, diverge and the more the extraction of higher probability cuts is favoured. The forest outcome will consequently show a greater accordance between the tree cluster's shapes, since the frequencies of extraction of the most probable hyperplanes are higher; however, even by setting the exponent to higher and higher values, the splitting process can't be completely deterministic, since its formulation intrinsicly includes the random component of the extraction. The exponentiation of the normalized metric is related to the concept of *regularization*. The regularization in machine learning has been introduced in order to reduce the overfitting and the overly complexity of the model used to describe the data; overfitting occurs when the model excessively adapts to

the data and consequently is not able to distinguish the general behaviour that characterize the distribution of points from other informations that have a certain weight only if considered in a restricted region of the dataset. In our case, the excessive adaptation to data is a consequence of the fact that the evaluation of the best split, in phase of partitioning, is performed locally (as explained in Section 3.2.1); in fact, it may occur that a specific splitting is highly convenient if we estimate the metric on a restricted portion of the dataset, but it is nearly equivalent to the other choices, if it is evaluated by considering the whole dataset. If this particular choice is enhanced by a high exponentiation of the metric, the frequency of extraction of the corresponding cut will be higher, in case of iteration of the algorithm; the final cluster shapes obtained by a forest may consequently exhibit some patterns that don't reflects the real shape of the whole dataset. The regularization consists in limiting this local effects by increasing the randomization of the process (in our case, by lowering the exponent), since it may occur (as will be shown in Section 4.4 and 5.2), that the theoric best choices of cutting hyperplanes, that correspond to the ones separating less similar groups of points, lead to results that are worst with respect to more randomized outcomes.

An other advantage of averaging the result of different trees is that the forest allows the algorithm to automatically determine the number of clusters in which the data and the corresponding space are divided. For this purpose, we consider, for each tree, the labeling associated to each point of the dataset; for each possible number of clusters, we have a different set of labeled data. The labels obtained by two different trees are compared, through the *adjusted mutual information*, for each possible fixed number of clusters, and this measure is repeated for each possible pair of considered trees. As explained in Section 1.3, the adjusted mutual information quantifies the information shared by two classifications and is thus a measure of the similarity of two labeling of the same set of data, invariant to permutations of the label names. It ranges between 0 and 1 and, higher is its value, more similar are the two sets of labels. Since pairs of groups of points with similar density and distance between them have similar values of the metric, we expect that the space partitioning within each cluster is more randomized than the partitioning between two clusters; consequently, if we consider different tree outcomes, we expect that the separation of the same cluster into subspaces varies in a more significant way with respect to the separation of the whole dataset in the correct number of clusters. We thus assume that each pair of tree outputs shows a different value of compatibility for a different division of the space into classes and, in particular, that the adjusted mutual information index is higher in correspondence of the real number of clusters. The number of clusters corresponding to the higher value of the AMI score is thus considered the most probable. Figure 3.4, on the right, shows a plot of the adjusted mutual information in function of the number of clusters; all the values of the score, corresponding to different pairs of trees, are averaged, for each fixed number of clusters, and the mean value is considered. The plot is referred to the forest result of 10 trees shown on the left of the figure and it correctly exhibit the higher value of compatibility for the number of clusters equal to 2. The different degree of randomization, due to different choices of the exponent, is also connected to the automatic determination of the number of clusters. In fact, more deterministic is the process, more the different tree outputs are compatible. This happens not only in regards to the division of the dataset into the right number of classes, but there may be a great accordance in the substructure of each cluster, with consequent difficulty in the identification of the correct number.

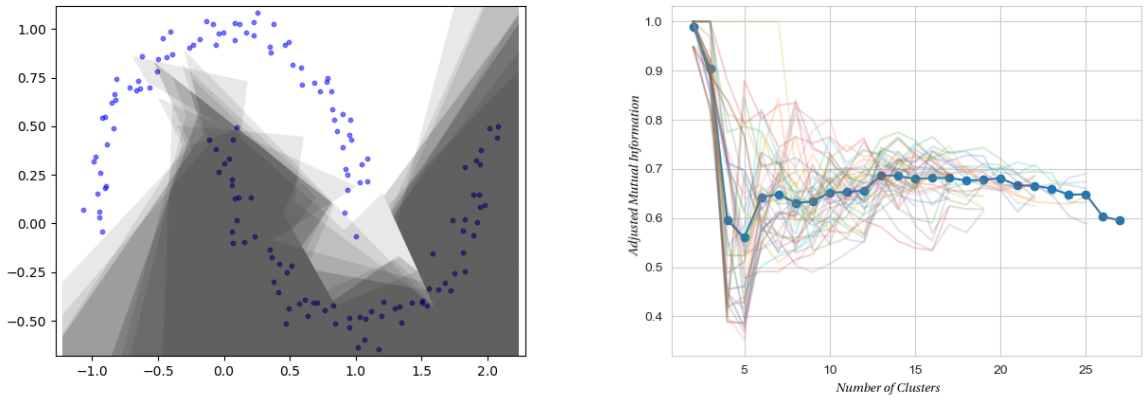


Figure 3.4: On the left, output of a Mondrian clustering forest (averaged over 10 trees) applied on a dataset consisting of two interleaving half circles. On the right, corresponding plot of the adjusted mutual information in function of the number of clusters; each line corresponds to a specific pair of tree outputs, while the more intense blue dotted line represents their average value. The algorithm is able to identify the correct number of clusters, since the higher compatibility between predicted labelings is shown for 2 clusters.

Chapter 4

Application to 2D toy datasets

The formulation of the algorithm, described in Chapter 3, allows a certain flexibility as regards the definition of similarity used to firstly partition the space and subsequently merge the obtained subspaces; the flexibility consists in the possibility to set different metrics, that are based on different concepts of similarity. Since the same definition of similarity may be not suited for every structure of the data, it may happen that the performance of the algorithm based on different metrics isn't the same when applied to datasets with different shapes or densities; in the following, the algorithm based on alternative metric choices is thus run on datasets with different characteristics, in order to compare the clustering results.

An other aspect that the algorithm allows to be adjusted by the user is the choice of the cutting hyperplanes. More precisely, besides the setting of both axis-aligned and generic direction cuts, whose outcome comparison is shown in Section 4.2, it is possible to tune the randomization of the hyperplane extraction process; in fact, since the probabilities of extraction of the cuts are raised to a certain power, we are able to decide how much favouring the extraction of the most probable cuts.In Section 4.4, the effect of different levels of randomization in the hyperplane extraction are evaluated.

The comparison between different versions of the algorithm is made by separately observing the partitioning and the merging phase results and the forest outcomes. As regards the evaluation of the partitioning phase, since the merging phase doesn't assign more than one cluster to each subspace, we should firstly check if each final subspace only contains data belonging to the same class; moreover, since the number of splits is related to the computational cost of the algorithm, we should measure how many splits are necessary to completely separate the two clusters. Regarding instead the evaluation of the merging phase, we observe how the predicted classification and the true one are closed. In order to evaluate the classification overlap, the Fowlkes-Mallows index *FMI*, introduced in Section 1.3, is used.

4.1 Datasets

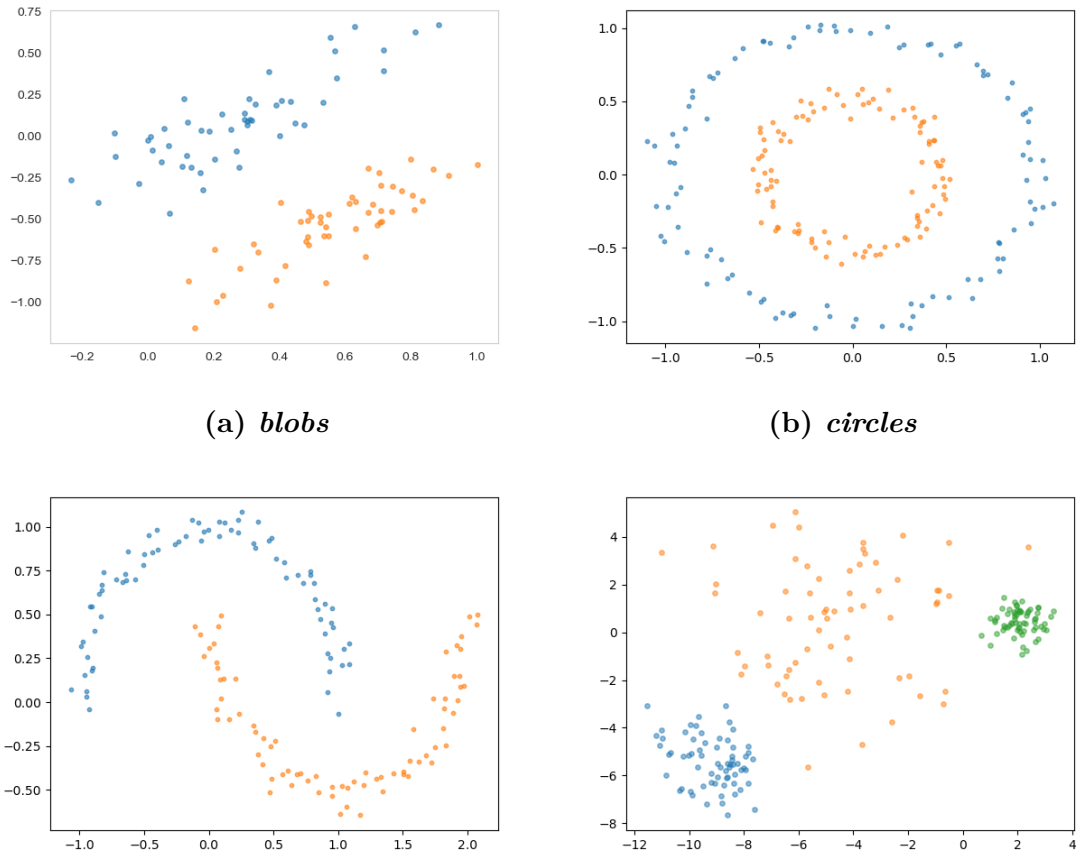
The 2 dimension datasets, considered in the analysis and described in the following paragraphs, are artificial datasets provided with true labelings. The dataset choice has been made in order to represent different dataset structures and to consequently test different characteristics of the algorithm; in particular, we consider a simple data structure consisting of two well separated convex shaped clusters, an other one consisting of three convex shaped clusters with different densities and two more datasets characterized by non convex clusters that can not be separated with a single linear cut. In particular, the density variation between clusters may be problematic in case of metrics based on the minimum distance, since the averaged minimum distance isn't the same for all the clusters, while the presence of non convex clusters may negatively influence the performance of the variance or centroid based algorithm, since the measure of the spreading

of the data is not so meaningful in case of elongated and irregular shapes.

blobs The first dataset consists of two groups of 50 points elongated in the diagonal direction. Each set of points is generated from a multivariate normal distribution and, in particular, the `random.multivariate_normal` function of `numpy` is used; the input parameters are the mean, the covariance matrix and the random seed. The covariance matrix is the same for both clusters and is a 2×2 matrix with diagonal and non-diagonal entries respectively equal to 0.07 and 0.06; the values of the mean and the random seed are respectively (0.2, 0) and 2 for one cluster and (0.6, -0.5) and 1 for the other one.

circles It is generated through the `datasets.make_circles` function of `scikit-learn` library and it consists of two concentric circles, centered in (0,0), with different radius and some added noise. The distance between the inner and the outer circle is determined by the scale factor parameter, that is set to 0.5; the noise is set to 0.05 and the random seed to 500. The whole dataset is constituted by 200 samples, equally divided between the two clusters.

moons This dataset consists of two interleaving half circles with some added noise and is generated through the `datasets.make_moons` function of `scikit-learn`. The noise and the random seed are respectively set to 0.05 and 500. The total number of samples is 150 and the points are equally divided between the two half moons.



(a) ***blobs*** **(b)** ***circles*** **(c)** ***moons*** **(d)** ***varied***

Figure 4.1: 2D datasets

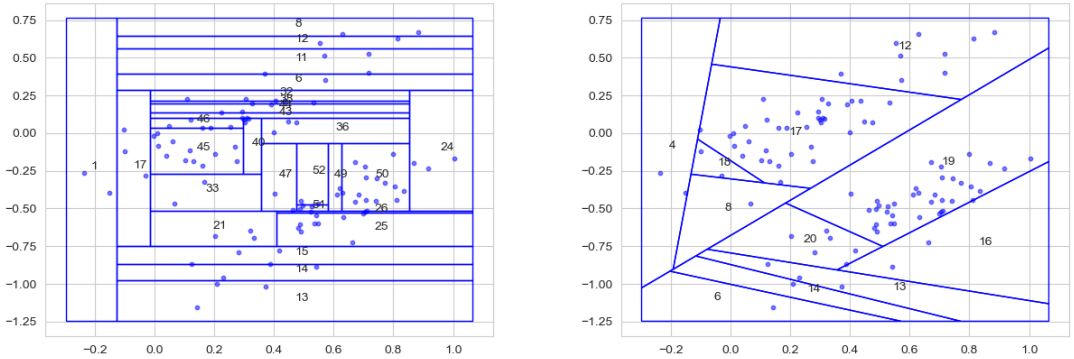
varied Also the last considered dataset is generated by a function of `scikit-learn` library, `datasets.make_blobs`, and consists of three adjacent isotropic Gaussian blobs with different density and equal numerosity (the total number of points is 200). In particular, two blobs of higher density are separated by the third blob of lower density. The parameter that determines the spreading of the points of each cluster is the standard deviation, that is set to 1, 2.5 and 0.5; the random seed is 170.

4.2 Evaluation of clustering results with different choices of the cutting hyperplanes

Figure 4.2 shows the partitioning phase output of ***blobs*** dataset in case of axis-aligned and generic direction cuts. In both cases, the space is splitted in order to have each polygon containing data belonging only to one class; from this point of view, it means that the two methods are equivalent and the partitioning output of both the algorithms can be considered a good input for the next merging phase.

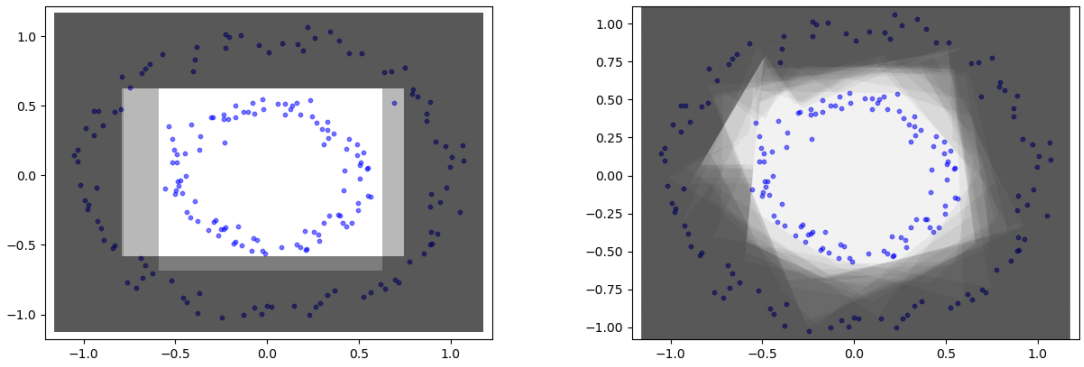
As regards the evaluation of computational costs, in the partitioning procedure performed through non axis aligned cuts, the first split is enough to totally separate the two groups of data; on the contrary, in the other case the complete separation is obtained after nine splits. This means that the method with cuts that are not linked to be orthogonal to the coordinate axis is much more convenient from a computational point of view, especially when the datasets require a splitting procedure on the diagonal direction.

An other reason to prefer the non-axis-aligned cuts is the absence of preferred directions. Figure 4.3 shows the Mondrian forest output, obtained by averaging the result of 20 trees, that performed the cuts with the two different choices of hyperplanes. The considered dataset is the ***circles*** one, that is rotation-invariant with respect to the axis perpendicular to the plane and intersecting the center of the circles. We see that the right plot reflects the rotation-invariant structure of the dataset, while the left plot shows the presence of preferred directions that is not a real characteristic of the data.



(a) axis-aligned cuts **(b)** generic direction cuts

Figure 4.2: Output of the partitioning phase run on ***blobs*** dataset, for different choices of the cuts; $\lambda = 4$, $exp = 50$. In both cases the clusters are completely separated by the splits but the computational cost is much higher in the case of axis-aligned cuts, since the separation is obtained after nine splits in **(a)** and after only one split in **(b)**.



(a) axis-aligned cuts

(b) generic direction cuts

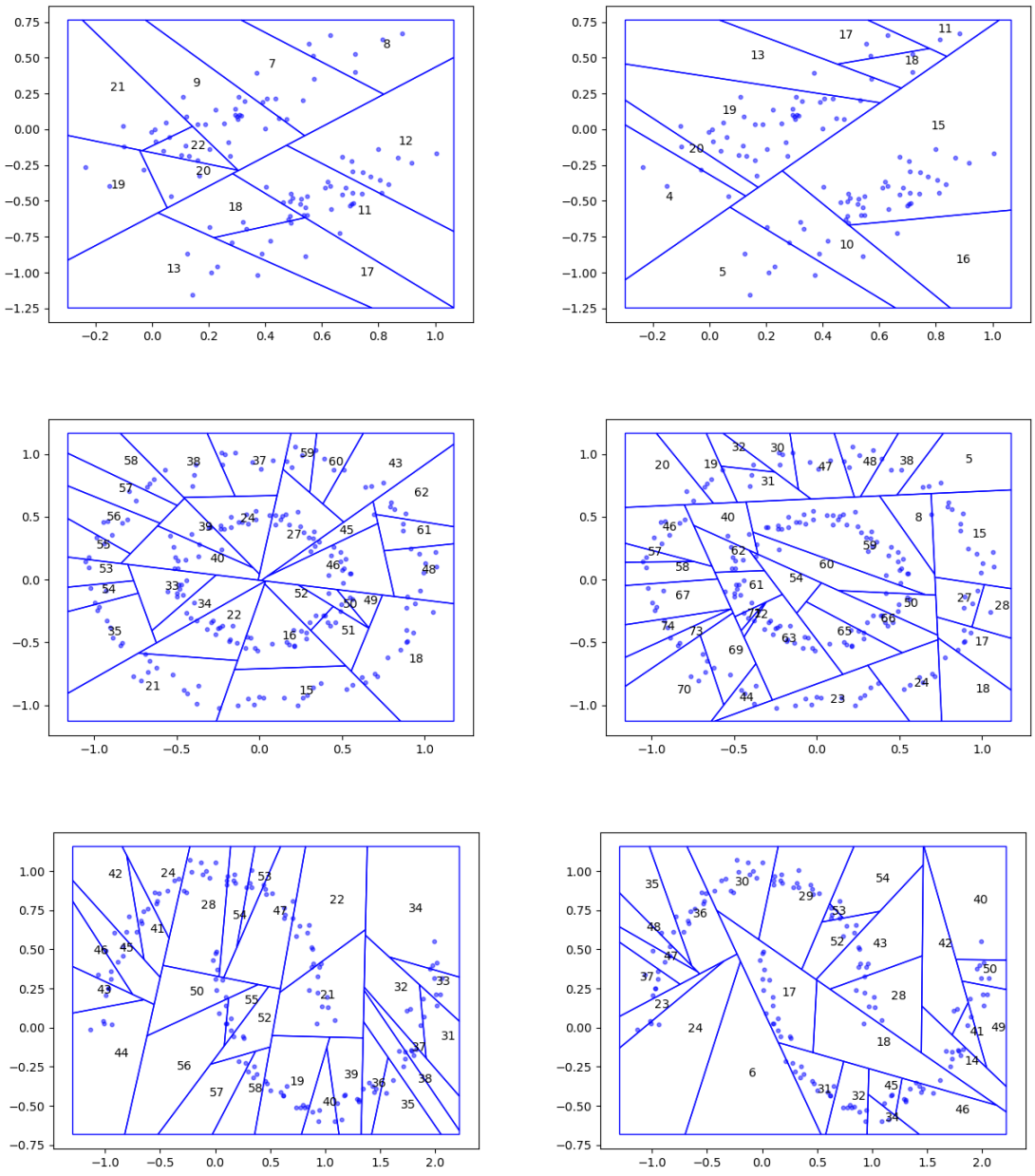
Figure 4.3: Output of the Mondrian forest run on ***circles*** dataset, for different choices of the cuts; $\lambda = 6$, $exp = 50$. Both the outputs well classify the data but in **(b)** the probability distribution that describes the two clusters reflects their shape, while in **(a)** it shows the presence of preferred directions that doesn't characterize the dataset.

4.3 Evaluation of clustering results with different choices of the metric

In this section we highlight the differences between versions of the algorithm based on different metrics; we only consider the splitting procedure through generic direction cuts, since its performance is more convenient with respect to the axis-aligned one, as explained in the previous paragraphs.

4.3.1 Evaluation of the partitioning phase

If the value of λ is high enough, each subspace that forms the final partitioning, obtained with any metric criterion, only contains points belonging to the same class. However, the splitting procedure is different in case of variance and centroid based metric or in case of minimum distance based metric. Figure 4.4 shows the partitioning phase output of ***blobs***, ***circles*** and ***moons*** datasets, obtained by using the variance based metric (left) and the minimum distance based metric with correction (right). In case of symmetric distributions of data, the variance based method tends to firstly split the set of points in order to obtain two groups with the same numerosity; the same behaviour is verified in the case of centroid based metric. On the contrary, the minimum distance based methods tend to firstly separate the groups of points with cuts passing through empty regions. Depending on the specific dataset, the variance/centroid based method is equivalent or less convenient than the other one, in terms of number of cuts required to completely separate the groups; in particular, for ***blobs*** and ***moons*** datasets, the two methods are equivalent, while for ***circles*** dataset, the two groups of points are completely separated after 4 cuts for the minumum distance based method, and after 15 cuts for the variance based one.



(a) variance **(b)** minimum distance with correction

Figure 4.4: Output of partitioning phase with variance based metric (left) and minimum distance based metric with correction (right); the algorithm is applied to ***blobs***, ***circles*** and ***moons*** datasets. $\lambda = 4$ for the first dataset and equal to 6 for the other ones, $exp = 50$. The variance based method tends to firstly split the space in order to obtain to groups with same numerosity, while the minimum distance based one tends to favour cuts passing through empty regions.

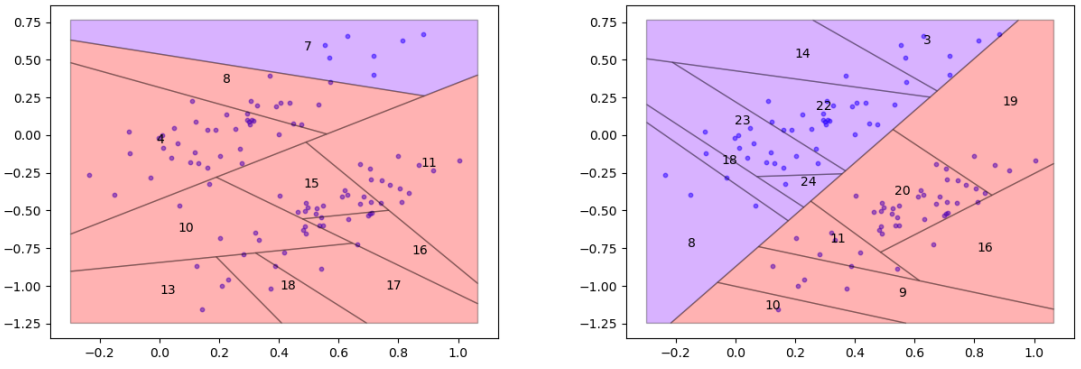
4.3.2 Evaluation of the merging phase

As regards the classification obtained through the merging phase, the performance of the algorithms based on different metrics is different according to the dataset structure. In order to evaluate the compatibility between the true and predicted labelings, the Fowlkes-Mallows index is calculated, for each combination of metrics and datasets; Table 4.1 shows the Fowlkes-Mallows index value, averaged over 20 iteration of the algorithm, while Figure 4.5, 4.6 and 4.7 show some examples of single merging outputs.

Metric	***blobs***	***circles***	***moons***	***varied***
Variance	0.66 ± 0.01	0.67 ± 0.02	0.67 ± 0.01	0.9 ± 0.1
Centroid	0.99 ± 0.02	0.61 ± 0.07	0.647 ± 0.008	0.70 ± 0.01
Minimum distance	0.97 ± 0.05	1	0.97 ± 0.09	0.748 ± 0.007
Minimum dist. with corr.	0.999 ± 0.004	0.999 ± 0.002	0.987	0.551 ± 0.002

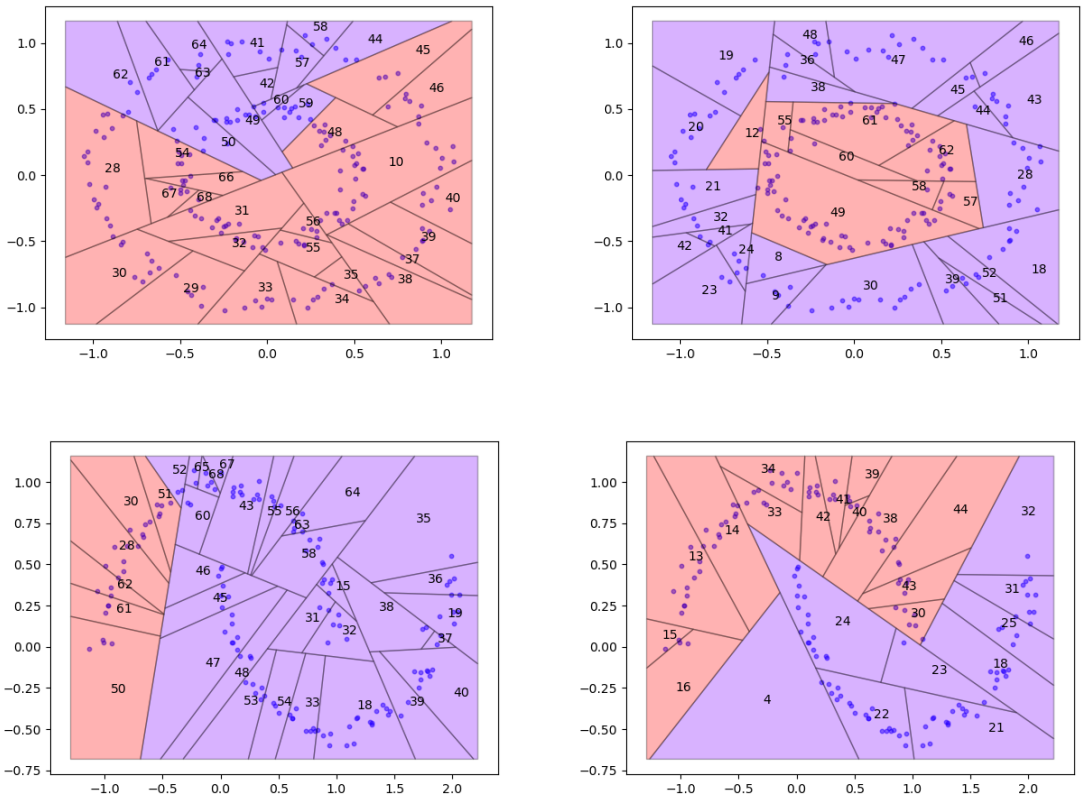
Table 4.1: Fowlkes-Mallows index: the value is averaged over 20 iterations of the algorithm for each combination of metrics and datasets. The versions of the algorithm based on the minimum distance metric generally provide better results, except in case of datasets with different density clusters; in this latter case the version based on the variance is the only one that is able to well classify the samples.

Considering ***blobs*** dataset, the algorithms based on centroid, minimum distance and minimum distance with correction metrics have very similar performances and the predicted labeling overlaps almost in every case the true one (see the right plot of Figure 4.5). The variance based method instead doesn't recognize the two groups of points but tends to separate single subspaces from the rest of the ensemble; the corresponding averaged *FMI* is indeed much lower than that of the other metrics.



(a) variance **(b)** minimum distance with correction

Figure 4.5: Output of merging phase run on ***blobs*** dataset, with different choices of the metric; $\lambda = 4$, $exp = 50$. The variance based method, differently from the minimum distance based one, is not able to correctly divide the samples into two clusters.



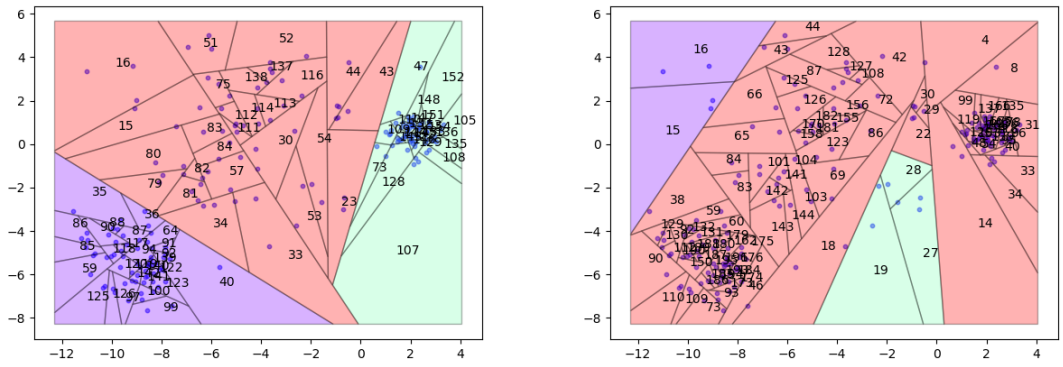
(a) centroid **(b)** minimum distance with correction

Figure 4.6: Output of merging phase run on ***circles*** and ***moons*** dataset; centroid based metric (left) and minimum distance based metric with correction (right). $\lambda = 6$, $exp = 50$. Only the minimum distance based method is able to well classify the dataset.

As regards ***circles*** and ***moons*** datasets (Figure 4.6), the result of the minimum distance based algorithms (with and without correction) is still very good, since it reflects almost in any case the true classification (right plot). The variance method keeps to behave similarly as in the ***blobs*** case (it isolates single subspaces), while the centroid based algorithm is no more able to well classify the two groups of data (left plot); the Fowlkes-Mallows indexes related to the variance and the centroid based methods have indeed similar values.

In case of ***varied*** dataset, the performances of the different methods are opposite: the variance based algorithm is the only one able to well recognize the three sets of points on the basis of their different densities, while the other three methods fail in any iteration of the algorithm (see the corresponding Fowlkes-Mallows indexes). The misclassification, in case of metric based on the minimum distance, is due to the fact that the algorithm isn't able to recognize two sparse groups of points as similar. More specifically, the ranking of the minimum distance values is unique for the whole dataset but each cluster is characterized by a specific averaged minimum distance, that is higher for the cluster with lower density; consequently, the algorithm tends to not consider the low density set of points as a single cluster.

Finally, we evaluate the difference between the two minimum distance based metrics, with and without correction. In order to compare them, we consider the merging procedures obtained by starting from the same output of partitioning phase. Figures 4.8 shows a space partition of ***moons*** dataset that don't completely separate the two groups of data; in particular, the subspace 14 contains one point belonging to the lower semicircle and all the other points belonging to the upper one. If the groups of data are not well divided by the partitioning procedure, meaning that some final subspace contains points belonging to more than one class,



(a) variance **(b)** minimum distance with correction

Figure 4.7: Output of merging phase run on ***varied*** dataset, with different choices of the metric. $\lambda = 6$, $exp = 50$. The minimum distance based method, differently from the variance based one, is not able to correctly divide the samples into three clusters; in fact, the ranking of the minimum distance is unique for the whole dataset, it considers groups of points belonging to the low density cluster as the most dissimilar.

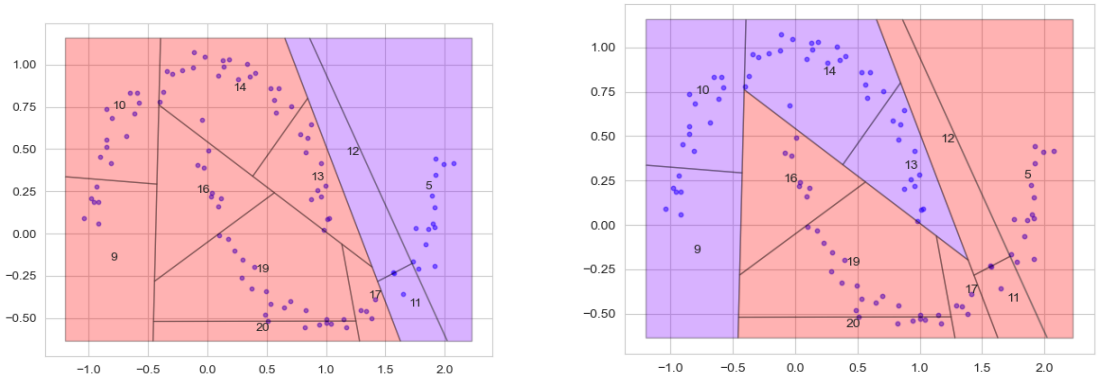
the metric without correction is not able to correctly assign the clusters (that means to choose the classification more similar to the true one) and we need the correction to well classify the data.

The left plot of the figure shows the result of the merging procedure in the case of metric without correction: by only considering the difference between the minimun distance between subspaces and the mean of the minimum distances within the subspaces, the subspaces 14 and 10 are considered more similar than the subspaces 11 and 17 and, consequently, they are separated later; we see that, in this case, the division into two cluster doesn't correspond to the division into two semicircles. If we add the correction to the metric, the score depends on more than one minimum distance between subspaces and it is thus a way to take into account the presence of a single outlier; the right plot of Figure 4.8 shows the correct classification of the dataset. However, in presence of more than one extra point in the same subspace, the correction is no more able to balance the incomplete partitioning of the initial dataset.

Moreover, an other difference between the two metrics is observed if we compare the outputs of the Mondrian forests averaged over 20 trees; Figure 4.9 shows that the one obtained with the metric with correction better reflects the structure of the dataset. This improvement is, however, observed only in case of ***moons*** dataset and thus depends on the particular shape of the clusters.

In conclusion, the centroid based method works only for very simple structures of data, like well defined and separated blobs. The minimum distance based methods are instead able to classify well separated groups of data with more complicated shapes; in particular, the metric with correction covers some specific cases, because it is less conditioned by the presence of single outliers in each subspace. Finally, the variance based method is the only one able to recognize adjacent clusters characterized by different densities.

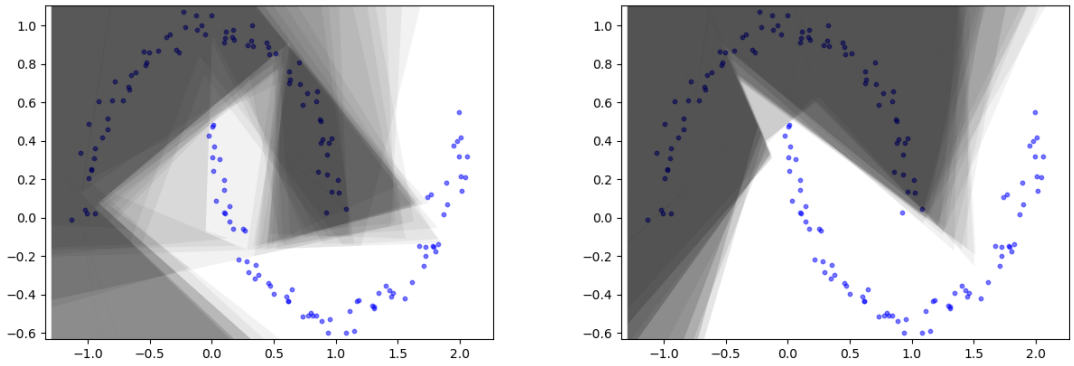
In the next chapter, we will only consider the versions of the algorithm based on the variance and on the minimum distance with correction metrics.



(a) minimum distance

(b) minimum distance with correction

Figure 4.8: Output of the merging phase run on ***moons*** dataset, with minimum distance based metrics with and without correction. $\lambda = 6$, $exp = 50$. The presence of the correction allows to take into account the presence of the outlier in the subspace 14.



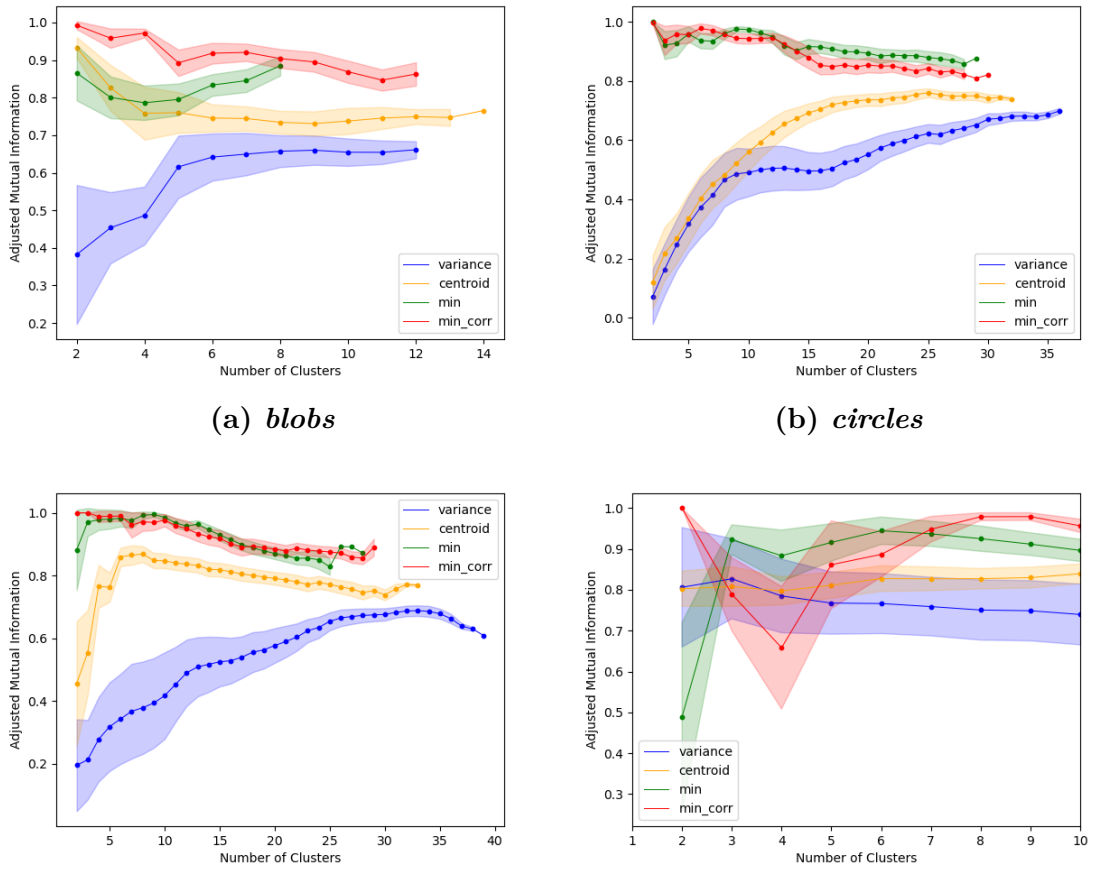
(a) minimum distance

(b) minimum distance with correction

Figure 4.9: Output of the Mondrian forest (averaged over 20 trees) run on ***moons*** dataset, with minimum distance based metrics with and without correction: both of them well classify the data but the version with correction better reflects the structure of the dataset. $\lambda = 6$, $exp = 50$.

4.4 Evaluation of clustering results with different levels of the cut extraction randomization

The clustering Mondrian forest gives as output a probability distribution: since a single tree uniquely associates a class to each subspace (and hence to each point of the space), when we overlap different space classifications obtained by several trees, each point of the space is no more related to a single class, but to more than one class with a certain probability. As explained in Section 3.3, the absence of a complete overlap of different tree results is due to the randomization of the partitioning procedure and, in particular, to the hyperplane extraction, that can be tuned by setting different values of the normalized metric exponent. In the following paragraphs we will firstly evaluate the effect of the intrinsic randomization, by considering the overlapping of different tree results obtained at fixed exponent, and we will see that it allows



(a) *blobs* (b) *circles*

(c) *moons* (d) *varied*

Figure 4.10: Averaged adjusted mutual information as a function of the number of clusters, for different choices of the metric. In **(a)**, **(b)** and **(c)** the minimum distance based metrics exhibit the highest values of compatibility, and the metric with correction is always able to automatically determine the correct number of clusters; in every case, the methods showing higher compatibility are those that provide a better classification of the dataset. In **(d)**, all the results show similar levels of compatibility, that don't correspond to similar performances as regards the classification of the dataset; in this case, the variance based method is the only one able to cluster the data.

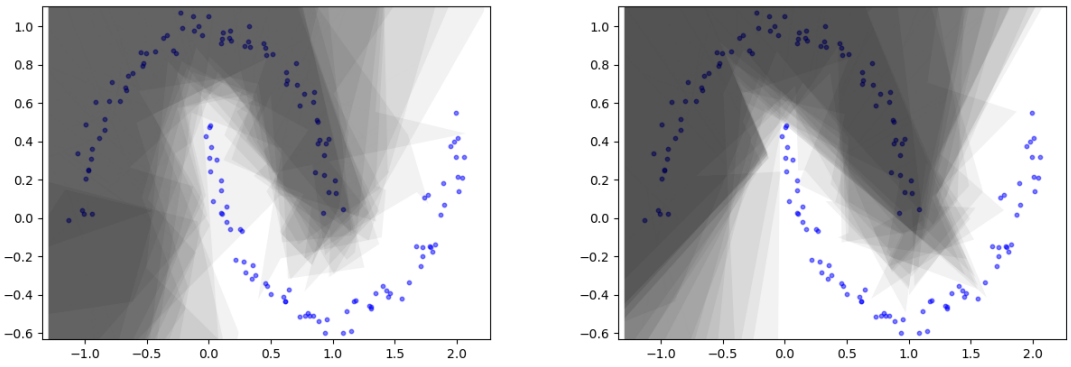
to automatically determine the number of clusters that characterizes the dataset. Secondly, the randomization due to different settings of the exponent is considered, by evaluating the consequences in the final probability distribution of each class and in the determination of the number of clusters.

Automatic determination of the number of clusters Since the cuts that splits different clusters are expected to be the most probable, while the ones separating homogeneous groups of points are extracted with similar probabilities and are more in number, if we iterate the algorithm several times we should observe that the cuts with higher probabilities are more frequently extracted with respect to the cuts splitting homogeneous clusters. We thus expect to observe a higher agreement of the different tree results in the shape of the clusters, rather than in the internal division of each cluster.

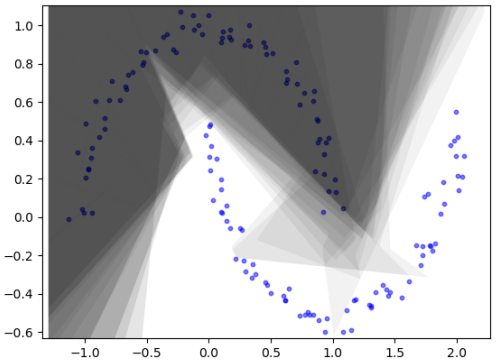
Consequently, by evaluating the overlap between predicted labelings, we expect to obtain an estimation about the most probable number of cluster in which the dataset is divided. As explained in Section 1.3, the evaluation of the compatibility of the different predicted data

classification is obtained by computing the Adjusted Mutual Information.

Figure 4.10 shows, for each considered dataset, the adjusted mutual information averaged over all possible pairs of tree outcomes, in function of the number of clusters in which the dataset is hierarchically divided. In general, the predicted results that are more compatible with the true classification (and that are consequently characterized by a Fowlkes-Mallows index closed to 1), show a greater level of internal compatibility, represented by a high value of the AMI coefficient. Firstly, we consider the versions of the algorithm run on ***blobs*** and ***circles*** datasets that well classify the points (that are the ones based on centroid and minimum distance metrics for the first dataset and the ones based on the minimum distance metrics for the second); in each of these cases, the highest value of averaged AMI index is observed for the number of clusters equal to 2, that is the true number of clusters in which the dataset is divided. Consequently, the algorithm is able to automatically determine the correct number of clusters. If we consider the curves corresponding to the variance based method (and the centroid based one for ***circles*** datasets), the compatibility between the predicted labelings is particularly low at lower number of clusters. This is consistent with the fact that, if the number of cuts performed in the partitioning phase is high enough, the final division of the whole dataset into subsets is similar between different tree outputs; however, since the algorithm, during the merging phase, isn't able recognize more similar subspaces, the final tree outcomes are less similar for lower number of clusters than for higher ones. As regards the plot corresponding to ***moons*** dataset, we observe the same previously described behaviour, except for the results corresponding to the minimum distance based metric without correction; in this case, although the high level of compatibility between the predicted and the true classification (see Table 4.1 for the Fowlkes-Mallows index value), the algorithm is not able to indentify the correct number of clusters, since the AMI value in case of dataset divided into two classes is lower than the values corresponding to higher numbers of clusters. Finally considering ***varied*** dataset, the curve corresponding to the variance based metric shows a slightly higher AMI value for the correct number of clusters (that is equal to 3); as concerns the other three curves, we don't evaluate the presence of the peak, since the corresponding results don't predict a correct classification of the data, but we observe that the averaged AMI value is higher than in the variance based case. The inability to identify the correct number of clusters combined with a high matching between true and predicted labelings is related to the opposite case of high compatibility among predicted labelings combined with the misclassification of the points for any iteration of the algorithm. In particular, in the first case, the behaviour of the averaged AMI curve is a sign of the recognition of patterns of points inside each cluster, although they are not considered in the true classification; similarly, the methods that misclassify the data show a great accordance in the identification of clusters, although they don't correspond to the true ones.



(a) $exp = 1$ **(b)** $exp = 5$



(c) $exp = 20$

Figure 4.11: Outputs of the forest (averaged over 20 trees) run on ***moons*** dataset, with different values of the exponent. Increasing the exponent, the process becomes more deterministic and the more the cluster shapes obtained by different trees overlap. For too low values of the exponent (**(a)**), some tree results misclassify the dataset, while, for too high values of the exponent, the hyperplane extraction is conditioned by local features that may not reflect the overall structure of the data (**(c)**); the preferred choice is thus a more randomized method that well classify the data (**(b)**).

Role of the exponent As previously said, given that the probability of extraction of a specific hyperplane consists of the corresponding metric value raised to a power, it is possible to evaluate how the forest prediction changes by varying the exponent. In particular, higher is the power, more deterministic the process becomes, since the difference in probability, between more and less possible cuts, is increased.

Figure 4.11 shows three outputs of the Mondrian forest performed on ***moons*** dataset (it averages the result of 20 trees); each one is characterized by a specific value of the exponent, that is respectively set to 1, 5 and 20, while the metric used is always the minimum distance based metric with correction. The variation in the randomization of the process is reflected by the superposition of the single tree outputs: for the higher value of the exponent, the shapes of the two subspaces that divide the whole dataset into two clusters are more similar and the transition between the two regions is sharper; moreover, the exponentiation enhance the distance of some metric values that actually correspond to equivalent choices of hyperplanes, and this enhancing may produce some border figures that don't reflect the shape of the dataset. With the decreasing of the exponent, the range of the possibile probabilities in the regions separating the two groups of points is wider than in the previous case; as a consequence, the edge

separating the two subspaces are less sharp and this means that in the regions without points the probability to belong to a specific class has on average lower values. On the other hand, if the exponent is too low, the process may be too randomized and the classification obtained from each tree may consequently not correspond to the real one; The plot corresponding to the power equal to 1 shows that, although the final result of the forest well reflects the data separation into classes, the output of some tree misclassifies the data. Since we can not have exact informations about the regions in which there are no points, the preferred choice of exponent is the one corresponding to the less deterministic method that allows to well classify the data.This behaviour corresponds to the overfitting phenomenon, that is limited by the regularization approach, explained in Section 3.3; the regularization, in our case, consists in limiting the value of the exponent, which resuls in favouring the randomness of the splitting, although, in theory, the better cut choices are always those with higher metric. A further confirmation of this phenomenon is observed in Figure 4.9 that shows that, for $exp = 50$, a single point is sistematically miclassified.

In order to automatically determine the number of clusters in which the whole dataset is divided, the Adjusted Mutual Information is calculated, as discussed in the previous paragraphs. Figure 4.12 shows the averaged AMI coefficient in function of the number of clusters, computed for different settings of the exponent. In the left plot, the results corresponding to the exponent set in the range between 1 and 3 are shown; we see that, while for the higher value of the power the line is clearly peaked in correspondence to two number of clusters (that is the true number for ***moons*** dataset), the ones corresponding to the exponent equal to 1, 2 and 2.5 show the higher average AMI coefficient for 8 and 3 number of clusters. Moreover, increasing the exponent, the mean value of the averaged AMI index, calculated over all possible numbers of clusters, increases; this means that, as said before, the rising of the power leads to a higher compatibility among the different tree classifications.

The right plot of Figure 4.12 shows the averaged AMI coefficient with the exponent set in the range between 5 and 65. While, in case of exponent equal to 5, 20 and 35, the higher compatibility is observed for two number of clusters, for the two higher values of the power (50 and 65), there is more than one possible classification with the maximum AMI coefficient. This is related to what said before about the great accordance in the predicted substructure of the clusters. In particular, each line is characterized by two peaks in correspondence to the same values of number of clusters (5 and 10); this means that all the forest outputs agree on dividing the single clusters into the same subgroups. Moreover, the rising of the curve pointed out in the previous plot is less significant with the increasing of the exponent: while for low values of the power a small change of it determines a great improvement on the compatibility of the results, for higher values of the exponent the curves almost overlap and the variations don't positively affect the compatibility.

In conclusion, also as concerns the automatic determination of the number of classes, the preferred choice of the exponent is the more randomized one that well classify the dataset; in case of no labeled data, expecially when defined in more than three dimensions, the determination of the better choice of the exponent is not trivial, since there can actually be several levels of the dataset classification, that reflects the presence of some substructure of each main cluster.

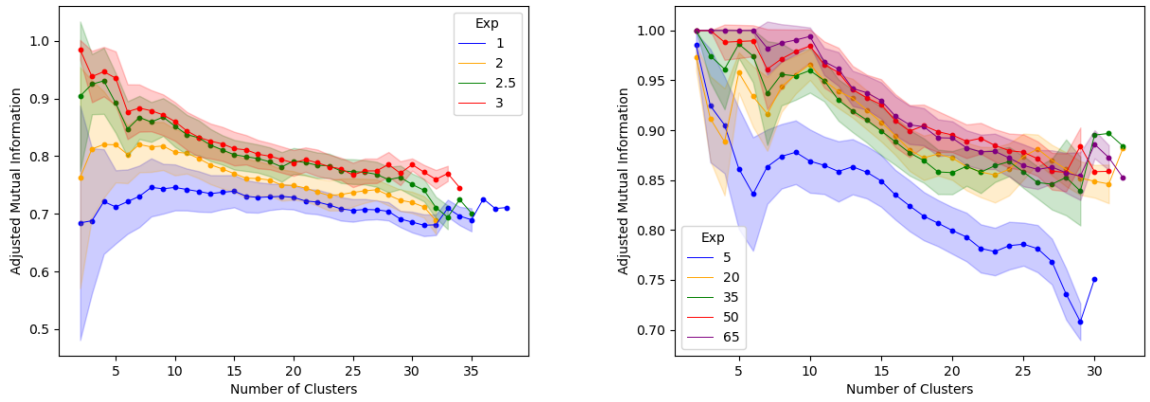


Figure 4.12: ***moons*** dataset: averaged adjusted mutual information in function of the number of clusters, for different values of the exponent. For too low and high values of the exponent, it is not possible to automatically determine the number of clusters; in the first case, the process is too randomized and some tree results are misclassified; in the second case, the outcomes correspoding to different number of clusters are highly compatible, because of the high level of determinism.

4.5 Comparison with other clustering algorithms

In this section we compare the outcomes of our algorithm with the results of the clustering methods described in Section 1. As expected, the k-means algorithm provides good results only if applied to datasets with convex shaped clusters (in our analysis, ***blobs*** and ***varied*** datasets); the classification predicted by the DBSCAN coincides with the true one in any case, except for the ***varied*** dataset since, as explained in Section 1.1, it usually doesn't have good performances when dealing with different density clusters. The spectral clustering is instead always able to well classify the data. The parameters given as input of the scikit-learn functions that perform the clustering are the number of clusters for k-means and spectral clustering algorithms, and ϵ for the DBSCAN; ϵ represents the maximum distance between two points in order to belong to the same cluster. Table 4.2 shows the Fowlkes-Mallows index that evaluates the matching between the predicted and the true labelings, for each clustering technique and for each considered dataset. If iterated with different random seed, the k-means and the spectral clustering algorithms provide almost always the same data classifications and the Fowlkes-Mallows score has exactly the same value.

	blobs	***circles***	***moons***	***varied***
k-means	0.980	0.495	0.623	0.889
DBSCAN	1	1	1	0.907
Spectral	1	1	1	0.980
Mondrian	0.999 ± 0.004	0.999 ± 0.002	0.987	0.9 ± 0.1

Table 4.2: Averaged Fowlkes-Mallows index, for each combination of datasets and clustering algorithms; except the case of k-means algorithm, the compatibility with the true labeling is high in all the other cases. For the Mondrian method, the choice of parameters is the one that provides better results.

In regards to the k-means algorithm, its performance is comparable to that of the Mondrian clustering method with centroid or minimum distance based metrics, if run on ***blobs*** dataset.

Also when applied to ***varied*** dataset, if we consider the variance based Mondrian method, the two algorithms have similar performances; in both cases, they are able to recognize the three clusters, with some difficulties in the identification of the border points of the low-density group of points. On the contrary, when run on ***circles*** and ***moons*** datasets, the k-means algorithm isn't able to well classify the data, since the clusters have no convex shapes; in fact, it simmetrically divides the whole dataset in two parts of equal numerosity. In these cases the Mondrian based method (with minimum distance metrics) provides much better results than k-means algorithm.

Considering the DBSCAN clustering technique, the input values of the parameter ϵ are set to 0.23, 0.25, 0.25 and 1.3 respectively for ***blobs***, ***circles***, ***moons*** and ***varied*** dataset. Its predicted labelings exactly match with the true ones, except for ***varied*** dataset; in this case, although the Fowlkes-Mallows index value is compatible with the one obtained from the variance based Mondrian outputs, the algorithm is not able to recognize the true number of clusters. In fact, it groups the data into five classes and, moreover, 13 points belonging to the low-density cluster in the true classification are labeled as outliers and are thus not associated to any cluster. The Mondrian algorithm run on the same dataset is instead able to recognize the true number of classes and, in general, correctly gathers the points, although some border points of the low-density cluster are misclassified. On the contrary, for the other datasets, the performance of DBSCAN algorithm is slightly better than the one of the Mondrian method with minimum distance based metrics; the results are though comparable.

The spectral clustering algorithm requires the number of clusters as input parameter. When run on every dataset, it provides optimal results; it slightly outperforms the Mondrian clustering method based on the minimum distance metric with correction in each considered case.

To summarize, the spectral clustering, the DBSCAN and the Mondrian based algorithm, run on 2 dimensional datasets, have very similar performances, while the k-means algorithm works only for a more limited variety of cluster shapes. The main difference between them consists of the different initial informations about the configuration of the dataset, required in order to make them able to well classify the data. More specifically, the spectral clustering and the k-means algorithms require the knowledge of the number of clusters, the DBSCAN the information about the minimum distance separating two clusters and the Mondrian algorithm requires to know if the cluster are characterized by different densities, in order to set the most suitable metric. If compared with the spectral clustering, the Mondrian based method has the advantage of being able to automatically determine the number of clusters, while the advantage of the spectral clustering is to have very good performances independently from the structure of the dataset. As concerns the comparison with the DBSCAN, both of the algorithms are able to automatically determine the number of clusters, but the DBSCAN doesn't provide good results for datasets with different density clusters; By changing the metric, the Mondrian based algorithm is able to overcome the difficulties in the determination of the clusters derived by dealing with different structures of data. Finally, an important added value of the Mondrian clustering method is that, other than providing a predicted labeling to the samples, it associates a probability distribution to each point of the space. This means that, once the dataset has been classified, given any other point with coordinates in the considered space but not belonging to the initial dataset, it is possible to estimate the probability of that point to belong to a specific class.

Chapter 5

Application to 3D toy datasets

In this chapter we will show the results of the algorithm run on datasets with three features. Since the different versions of the algorithm applied on the two dimension datasets exhibited better performances with the metric based on the variance and on the minimum distance with correction, now we will only consider these two metrics; in particular, in the first section, a dataset characterized by different density clusters is considered, in order to evaluate if the performances of the algorithm based on the two different metrics are the same than in the analogous case in two dimensions. In the following sections, the algorithm based on the minimum distance metric is applied to datasets with different shapes and we will focus, in particular, on the different clustering results obtained by varying the exponent and the lifetime parameters. The adjusted mutual information and the Fowlkes-Mallows index are evaluated by computing 20 iterations of the algorithm for each choice of parameters and datasets.

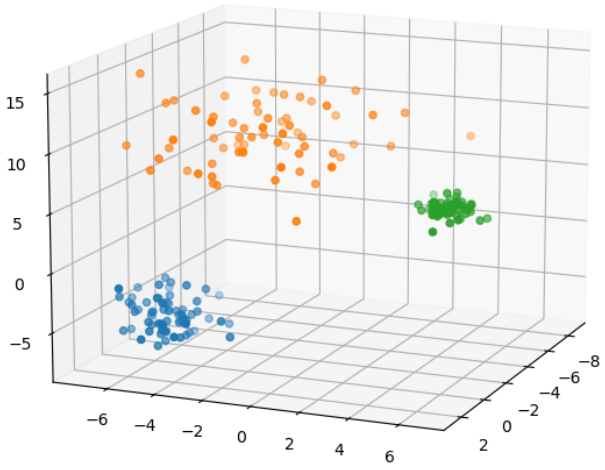


Figure 5.1: ***varied3D*** dataset, with random seed = 50

5.1 Evaluation of clustering results with different choices of the metric

The first considered dataset, ***varied3D***, shown in Figure 5.1, is the analogous in three dimensions of ***varied*** dataset and it consists of three isotropic Gaussian blobs with different densities; it has been chosen in order to check if the behaviour of the algorithm with different choices of the metric is the same in two and three dimensions. Also in this case, the `datasets.make_blobs` function of `scikit-learn` is used to generate the samples. The input parameters are the number of samples, set to 200 (the points are equally divided into the

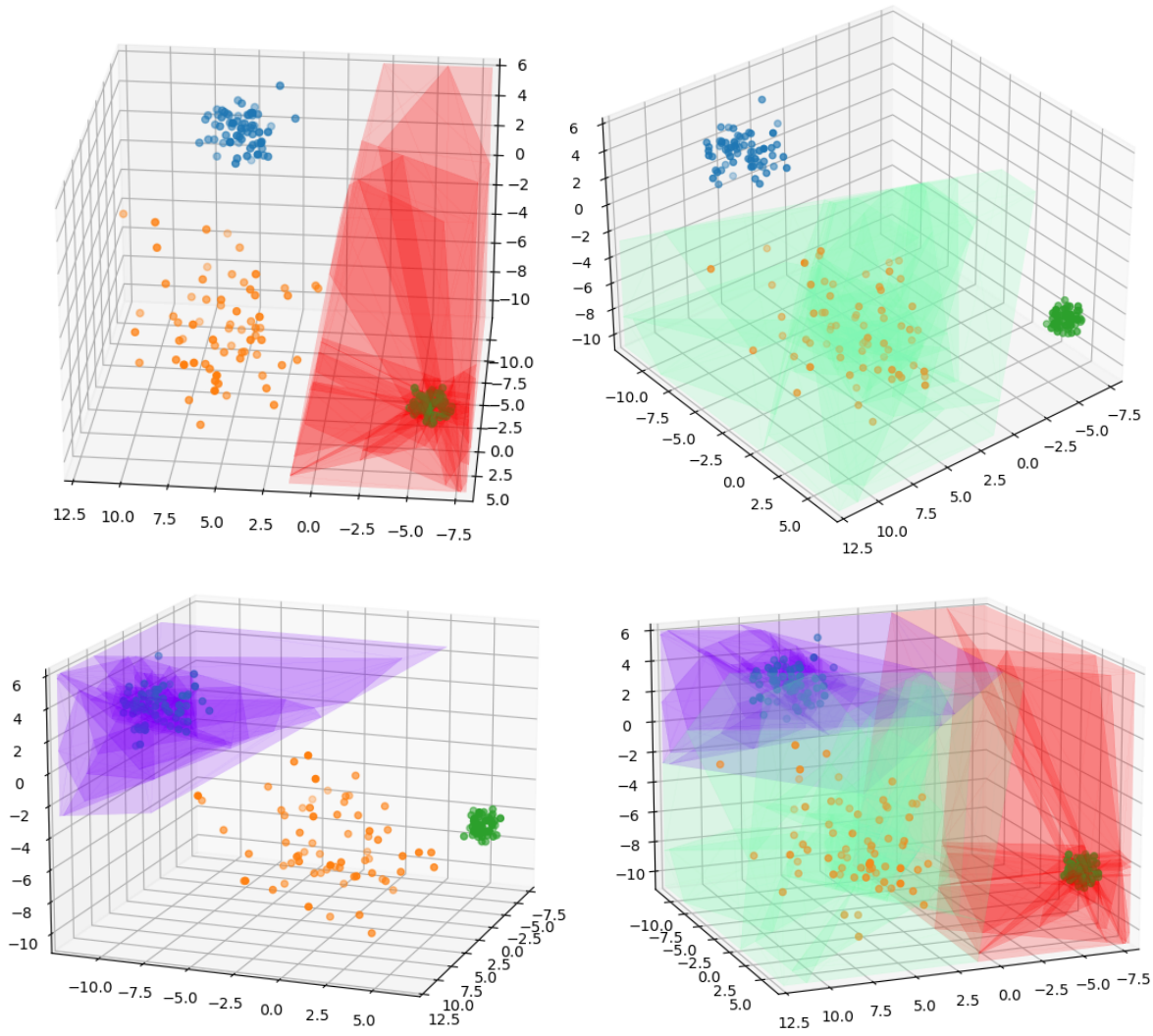


Figure 5.2: Output of Mondrian clustering tree run on ***varied3D*** dataset ($seed = 10$); the minimum distance based metric is used. The regions of the space assigned to each cluster are shown both separately and in the same plot.

custers), the standard deviation and the random seed. In particular, the standard deviations of the Gaussian distributions are set to 1, 2.5 and 0.5 and three different seeds (10, 50 and 150) are considered for the random generation of the dataset, in order to have slightly different arrangements of points.

In Figure 5.2, the space partition of the Mondrian tree run on the dataset with $seed = 10$ is shown; for clarity, the subspaces corresponding to different clusters are shown both separately and in the same plot.

While, in the case of 2 dimension datasets with similar structure, the variance based metric is the only one that well recognizes the different groups of points, in three dimensions both the considered metrics are able to classify the data, but the algorithm based on the minimum distance metric generally works better than the one based on the variance. For each iteration of the algorithm, the exponent is set to 20 and the lifetime is set high enough to make the splitting stop only when all the subspaces contain only one or two points.

In particular, if we consider the dataset with random seed equal to 10, all the results of the trees based on the minimum distance exactly coincide and the predicted classification is equal to the true one; the Fowlkes-Mallows score value (for the classification into three clusters) is consequently equal to 1 with zero standard deviation. On the other hand, the algorithm based

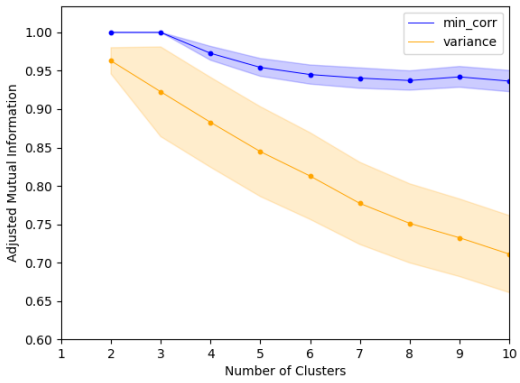
on the variance metric does not always provide the same classification of the true labeling, because it misclassifies some points belonging to the low-density blob that are more separated from the center of the cluster; however, the compatibility with the true classification is high, except in two cases where the algorithm completely misclassifies the data. The corresponding Fowlkes-Mallows score is $FMS = 0.97 \pm 0.07$.

As regards the dataset with random seed equal to 50, the considerations about the results corresponding to the minimum distance metric are exactly the same, while the performance of the variance based algorithm is worst than the previous case; in particular, in six out of twenty cases, the algorithm misclassify the data and the corresponding averaged Fowlkes-Mallows index is $FMS = 0.89 \pm 0.15$.

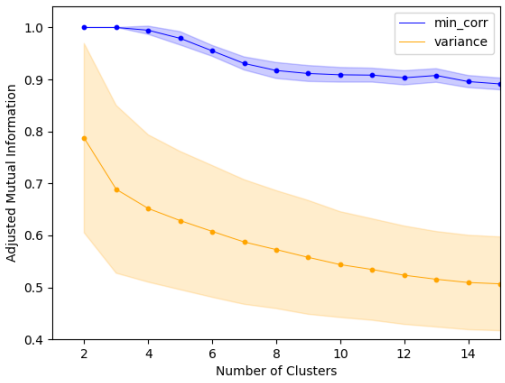
Despite the algorithm based on the minimum distance achieve in general good results, for particular configurations of the dataset it is not able to correctly recognize the groups of points. For example, considering the generation of the dataset with random seed equal to 150, we observe that two of the three adjacent clusters with different densities are considered more similar than some groups of points belonging to the lower density cluster in the true classification. This behaviour is observed in the output of every tree, and the corresponding value of Fowlkes-Mallows index is $FMI = 0.766 \pm 0.001$. In this case, the performance of the variance based method is better, since the predicted labelings are highly compatible with the true one and the averaged Fowlkes-Mallows index is $FMI = 0.991 \pm 0.007$.

Figure 5.3 shows, for each considered dataset, the averaged adjusted mutual information in function of the number of clusters. In case of random seed equal to 10 and 50, the lines corresponding to the minimum distance based algorithm show the maximum accordance in case of two and three number of clusters, while the compatibility in case of the variance metric is lower (especially for the dataset with random seed equal to 50). The different levels of compatibility between the predicted labelings, in this case, reflects the different performances of the considered configurations of parameters and datasets. However, the internal compatibility between predicted labels is not always related to a good performance of the algorithm; the plot corresponding to the random seed equal to 150 shows, for example, a high value of the averaged AMI coefficient in case of the classification into two and three clusters with the minimum distance based method; this means that different iterations of the algorithm agree on considering the same groups of points belonging to the low-density blob in the true labeling as separated clusters, although this result doesn't match with the true classification. In particular, the peak of the blue line corresponding to 5 clusters coincides with the correct separation of the two adjacent blobs, that, as said before, are divided after some groups of points belonging to the low-density cluster.

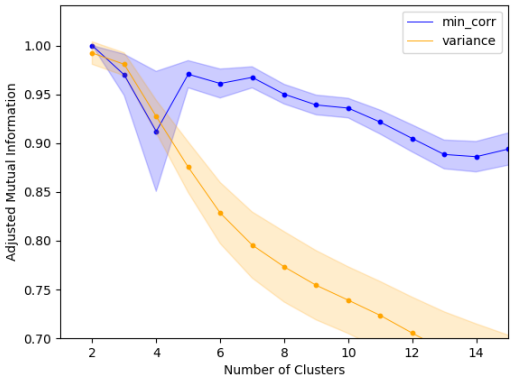
As regards the automatic determination of the correct number of clusters, it can be achieved for the minimum distance metric result of the datasets with random seed equal to 10 and 50 since, in this case, the compatibility of the predicted labelings starts to decrease from the number of clusters equal to 4. On the contrary, in all the cases based on the variance metric, the AMI score exhibit the higher value for 2 number of classes and progressively decreases with the increase of the number of clusters. This means that the different trees agree more on the first division of the dataset into two clusters than in the following one into three and, consequently, even if the algorithm is able to well classify the data, it may be not able to automatically determine the correct number of clusters.



(a) $seed = 10$



(b) $seed = 50$



(c) $seed = 150$

Figure 5.3: Averaged adjusted mutual information in function of the number of clusters for ***varied3D*** dataset, with different choices of random seed and metric. The blue line corresponds to the minimum distance based metric with correction and the yellow line corresponds to the variance based one. **(a)** and **(b)**: the minimum distance based metric generally shows the higher compatibility between predicted labelings and is able to identify the correct number of clusters; the variance based metric shows a lower compatibility and the correct number of clusters is not recognized. **(c)**: for low values of the number of clusters, the two methods have similar levels of compatibility, however, the corresponding performances of the algorithm are completely different, since the version based on the minimum distance completely misclassifies the dataset, while the variance based one provides a good classification.

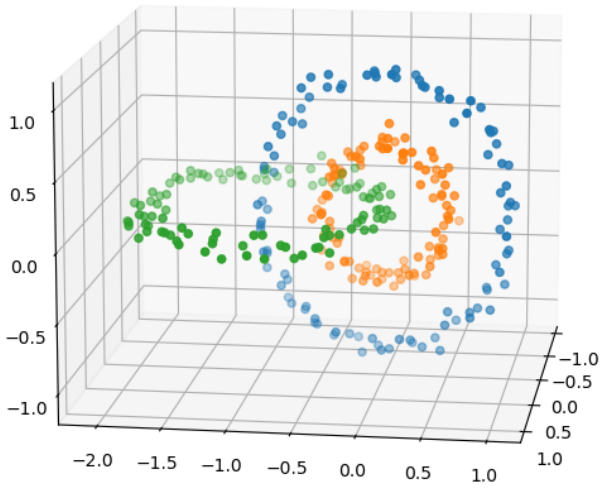


Figure 5.4: ***circles3D*** dataset

5.2 Evaluation of clustering results with different choices of the exponent and the lifetime parameters

The second considered dataset, ***circles3D***, is shown in Figure 5.4. In has been chosen in order to test the algorithm when run on clusters with non trivially separable shapes; we compare different results, obtained by varying the exponent parameter, that represent the level of randomization in phase of hyperplane extraction, and the lifetime parameter. It consists of three circles of 100 samples: two of them are concentric circles with different radius and their axis is perpendicular to the axis of the third circle, that intersects the center of the previous two. More specifically, both the single circle and the set of two concentric circles are generated through the `datasets.make_circles` function of `scikit-learn`; the noise is set to 0.05, while the number of samples, the scale factor and the random seed take different values. As concerns the single circle, the total number of points is set to 100 and the scale factor to 0.9, in order to make the function generate two overlapping circles, each one constituted by 50 points and centered in (0,0); the resulting group of points can thus be considered as a single circular cluster of 100 samples. The random seed value is 500. The obtained point coordinates are arranged in order to belong to the (x, y) plane, while the z-coordinates to associate to the points are extracted from a Gaussian distribution with 0 mean and standard deviation equal to 0.05; the function used to generate the z coordinates is the `random.normal` function of `numpy`. The cluster is finally translated along the y axis, in order to be centered in (0,-1,0). As regards the two concentric circles, they are generated by setting the random seed to 550, the number of samples to 200 and the scale factor to 0.5; we thus obtain two concentric clusters of 100 points each, centered in (0,0). This time, the 2 dimensonal data coordinates are arranged to belong to the (y, z) plane, while the x coordinates are generated from the same Gaussian distribution used in the previous case.

In order to evaluate the performance of the algorithm by varying the exponent and the lifetime parameters, we consider two values of the lifetime, 50 and 200, and, for each case, we set the exponent equal to 5, 10 and 20. Table 5.1 shows the Fowlkes-Mallows index values for each combination of parameters. The general observed behaviour consists of a significant improvement of the results with the decreasing of the exponent; the decrease of the lifetime parameter is as well related to an improvement, but not particularly relevant.

The best outcome is obtained for $\lambda = 50$ and $exp = 5$: in 17 out of 20 cases the predicted labeling is equal to the true one and the corresponding averaged Fowlkes-Mallows index shows

λ	exp	FMI
50	5	0.96 ± 0.10
	10	0.86 ± 0.15
	20	0.79 ± 0.12
200	5	0.93 ± 0.12
	10	0.80 ± 0.10
	20	0.83 ± 0.11

Table 5.1: Fowlkes-Mallows index for each combination of lifetime and exponent parameters. The greater variability is observed with the variation of exp, while the results are less affected by the variation of λ; the best outcome is obtained for the lower values of both parameters.

the highest value. With the increasing of the exponent, a lower number of predicted results is compatible with the true classification; in particular, the predicted and true classifications coincide in 10 cases for $exp = 10$ and in 4 cases for $exp = 20$. By setting the higher value of the lifetime, $\lambda = 200$, the results are similar for the exponent equal to 5 and 20, since the data are correctly classified respectively in 15 and 6 cases out of 20; on the contrary, for $exp = 10$, the result is much worst, since the true and predicted labelings coincide only in 4 cases.

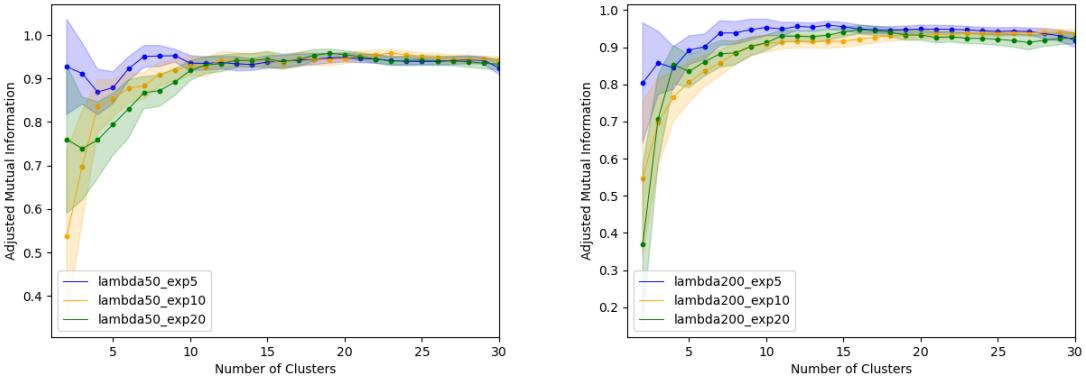
In Section 4.4, we evaluated the compatibility between the predicted point classifications in the case of the 2 dimensional ***moons*** dataset; we saw that, with the increase of the exponent, the values of the AMI coefficient tend to get closer to 1, since the exponent parameter is related to the randomness of the splitting and, higher is the power, more deterministic is the process of the extraction of the hyperplanes. In the case of ***circles3D*** dataset, however, the behaviour of the curves corresponding to different values of the exponent is exactly the opposite. As shown in Figure 5.5 (each plot corresponds to a different value of the lifetime), the curve representing the averaged Adjusted Mutual Information in function of the number of clusters for $exp = 5$ takes higher values with respect to the curves corresponding to the other values of the power.

Both the observed behaviours of the increase of the compatibility among the predicted classifications and between the predicted classifications and the true one (respectively represented by the adjusted mutual information and the Fowlkes-Mallows index values), for decreasing values of exp, can be interpreted through the concept of regularization (see Section 3.3). As previously said, in our case the regularization consists in limiting the level of determinism of the hyperplane extraction in favour of a more randomized procedure. While, in the 2 dimensional case, the increasing of the exponent doesn't produce a significant decrease on the compatibility between predicted and true classifications and among different predicted classifications, in 3 dimensions this procedure is more clearly shown by the corresponding values of Fowlkes-Mallows and adjusted mutual information indexes, that quantitatively confirm the achievement of better results for lower values of the exponent.

By considering a fixed value of the exponent, the slight difference between the performances of the algorithm corresponding to the two values of the lifetime depends on the fact that, for higher λ, the initial space has been splitted more times and, consequently, there may be a greater number of final subspaces containing only one point; Since the metric used to merge a single point subspace with one of its neighbors is different from the metric used to combine subspaces with more than one point, the inequality of the performances is due to the existence of this two merging criterions.

Finally, we consider the version of the algorithm with $exp = 5$ and $\lambda = 50$, that is the one that better classifies the data, in order to evaluate the automatic determination of the number of clusters. Athough the result shows a high level of compatibility between the predicted labelings, as well as a great matching between predicted and true classifications, the averaged AMI curve doesn't exhibit a peak in correspondence of the correct number of clusters. In fact,

the compatibility between labelings is higher in correspondence of the division of the dataset into two groups and a great accordance in the recognition of the substructure of the single clusters is shown; it consequently leads to the inability to recognize the division into 3 clusters as the most significant for the description of the dataset.



(a) $\lambda = 50$ **(b)** $\lambda = 200$

Figure 5.5: Averaged adjusted mutual information in function of the number of clusters for ***circles3D*** dataset, for different values of *exp* and λ. The outputs that better classify the dataset also show a higher compatibility between predicted labelings; in all the cases, the algorithm is not able to automatically determine the number of custers.

5.3 Evaluation of clustering results obtained on datasets with different shapes

In this section we evaluate the performance of the algorithm run on two new datasets of different shapes, shown in Figure 5.6 and 5.7.

cylinder dataset consists of three clusters: a cylinder surrounding a thin and elongated blob and adjacent to an other isotropic Gaussian blob. The cylinder is constituted by 200 samples and has axis orthogonal to the (x, y) plane; it is generated by expressing its points in cylindrical coordinates and, more precisely, the coordinates of each sample (x, y, z) are written in the form $(\rho sin\theta + \epsilon_1, \rho cos\theta + \epsilon_2, z)$. ρ is fixed to 0.5, while θ is randomly extracted from a Uniform distribution between 0 and 2π; ϵ_1 and ϵ_2 are instead generated from a Gaussian distribution with 0 mean and standard deviation equal to 0.05 and, finally, z is extracted from a Uniform distribution between 0 and 0.5. As regards the blob surrounded by the cylinder, it consists of 50 samples; its coordinates are extracted from a Uniform distribution defined between -0.1 and 0.1, in case of x and y coordinates, and between 0 and 0.5, in case of z coordinates. The random extractions are obtained through the `random.uniform` and `random.normal` functions of `numpy`; the random seeds are set to different values for any extraction of the values. The third Gaussian cluster is generated through the `sklearn.datasets.make_blobs` function, used in the previous cases. The input parameters are the number of samples, the cluster standard deviation, the random seed and cluster center coordinates; they are respectively set to 50, 0.1, 0 and (0.8,0.8,0.8).

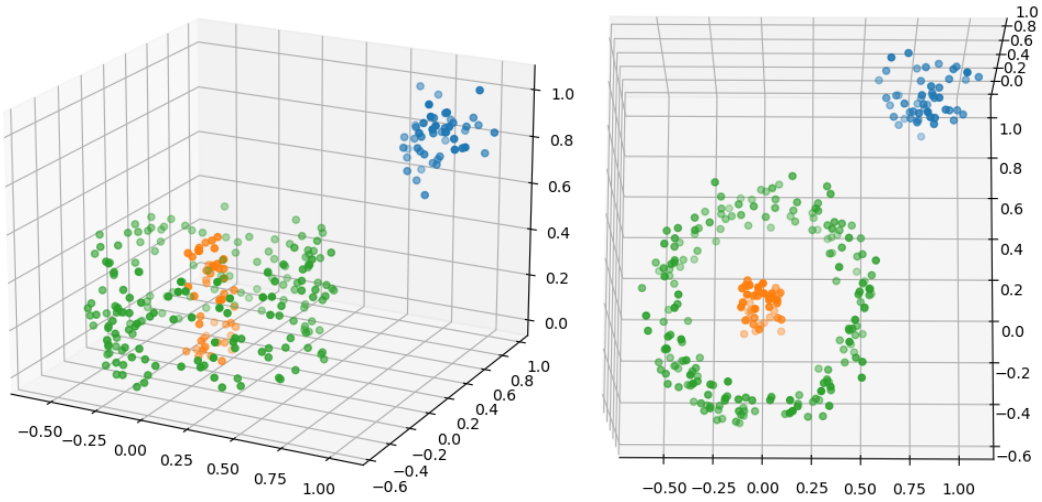


Figure 5.6: ***cylinder*** dataset

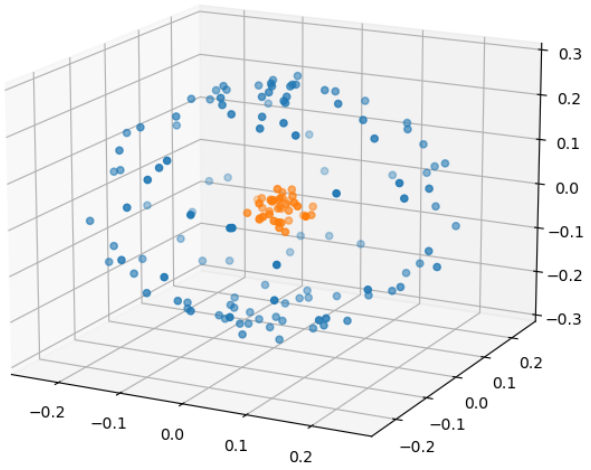


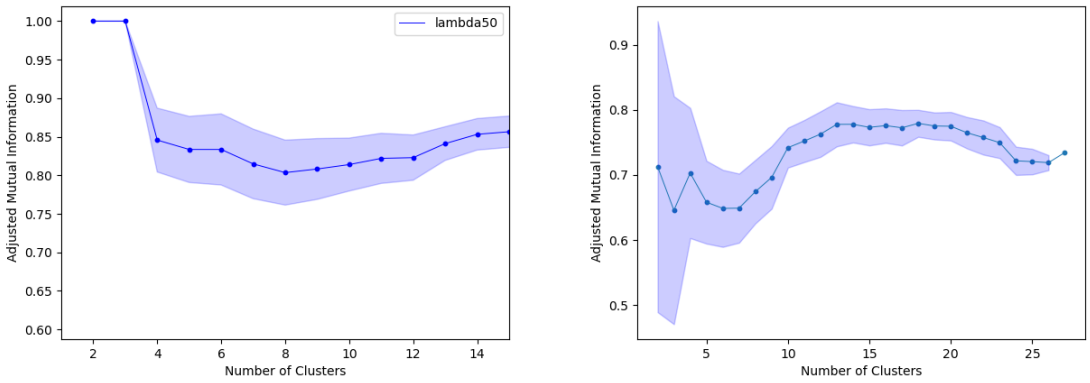
Figure 5.7: ***sphere*** dataset

sphere dataset consists of two non linearly separable clusters: a sphere with low-density homogeneous distributed points and a higher density blob, centered in the center of the sphere. Each sphere point (x, y, z) is expressed in spherical coordinates and takes the form $(\rho sin\phi \cdot cos\theta + \epsilon_1, \rho sin\phi \cdot sin\theta + \epsilon_2, \rho cos\phi + \epsilon_3)$. ρ is set equal to 0.25, while θ and ϕ are randomly extracted from a Uniform distribution, respectively in the range $[0, 2\pi]$ and $[0, \pi]$; ϵ_1, ϵ_2 and ϵ_3 are extracted from a Gaussian distribution with 0 mean and standard deviation equal to 0.01. The random seed is changed at each random generation. As in the previously described dataset, the `numpy.random.uniform` and `numpy.random.normal` functions are used to generate the points. The total number of samples constituting the sphere is 120. The blob located in the sphere center is generated through the `sklearn.datasets.make_blobs` function; the number of samples, the cluster standard deviation, the cluster center and the random seed are set respectively to 40, 0.02, (0,0,0) and 18.

Since, in the previously considered cases, the best results were obtained for $exp = 5$, we keep the same value of the exponent when running the algorithm on the two new datasets; the lifetime λ is set to 50 for the ***cylinder*** dataset, and to 200 for ***sphere*** dataset. In both cases, the predicted classification is highly compatible with the true one. In particular, for the first one, the predicted labeling exactly matches the true one for any iteration of the algorithm; the averaged Fowlkes-Mallows index is consequently equal to 1 with zero standard deviation.

When applied to the second dataset, the algorithm misclassifies the data in three out of twenty cases; the value of the corresponding Fowlkes-Mallows index is $FMI = 0.97 \pm 0.08$.

As regards the matching of the different predicted classifications, Figure 5.8 shows the two plots of the averaged Adjusted Mutual Information in function of the number of clusters. We observe full compatibility in case of ***cylinder*** dataset; in particular, the highest value of AMI coefficient is obtained for two and three number of clusters. Since, from 4 number of clusters, the averaged AMI value suddenly decreases, it is possible to automatically recognize the correct number of classes as the higher value before the lowering of the curve. In the case of ***sphere*** dataset, the overall compatibility is much lower than the previous case; this is due to the presence of some tree outcomes that completely misclassify the data although, in 17 out of 20 cases, the predicted classifications completely overlap. Despite the relative low averaged compatibility, by locally observing the behaviour of the curve (that means by considering the averaged AMI only at low values of the number of clusters), it is possible to determine in how many classes the dataset is divided, since the higher value of adjusted mutual information is locally shown for 2 clusters. Despite this procedure doesn't lead to the correct number of classes without any doubt, the local evaluation of the curve peaks at low number of clusters may help to understand the structure of the dataset; in fact, the rising of the curve may correspond to a higher compatibility among the different prediction due to different factors, like the presence of similar subsets of points in different partitioning outcomes, related to the high fragmentation of the dataset.



(a) ***cylinder*** **(b)** ***sphere***

Figure 5.8: Averaged adjusted mutual information in function of the number of clusters. **(a)** maximum compatibility between predicted labelings for 2 and 3 number of clusters; since, for 4 clusters, the AMI value suddenly decreases, it is possible to recognize the division of the dataset in three clusters as the correct one **(b)** low compatibility between different predicted labelings; however, by locally observe the behaviour of the curve at small values of the number of clusters, it is possible to correctly indentify the correct number.

5.4 Comparison with other clustering algorithms

In this section, we compare the previously described results with the outcomes obtained by running the k-means, DBSCAN and spectral clustering algorithms on the same datasets. As in the two dimensional case (see Section 4.5), while the k-means algorithm is able to well classify the data only in presence of convex-shaped clusters, the other two methods provide good results when applied to any dataset. The input parameters are the number of clusters, for k-means and spectral clustering algorithms, and ϵ for the DBSCAN method; in particular, ϵ is set to

3.5, 4.3, 2.7, 2.5, 0.2 and 0.15 respectively for the datasets considered in the order presented in this chapter. Table 5.2 shows the values of the Fowlkes-Mallows index for each combination of dataset and clustering technique.

If we consider the first 3D dataset, consisting of three blobs of different densities, for any choice of the seed the predicted classifications are highly compatible with the true ones, although a few points belonging to the low-density clusters are, in some cases, misclassified or considered as outliers (in case of DBSCAN algorithm). As observed in Section 5.1, the most problematic version of this dataset is the one with seed equal to 150: a higher number of points is misclassified and the corresponding values of Fowlkes-Mallows index are lower (although the results are still highly compatible). If we examine the performance of the DBSCAN algorithm run on the 2D ***blobs*** dataset, that is the analogous version of the 3D considered one, we observe that the results in three dimensions are much better that those in two dimensions. In particular, despite in both cases some low-density points are categorized as outliers, in the 3D case the algorithm is able to recognize the correct number of clusters. The improvement of the DBSCAN outcomes with the increase of the number of dimensions in similar to the improvement observed for the Mondrian clustering technique based on the minimum distance metric with correction results.

As regards the other three datasets with no convex shape, the k-means algorithm isn't able to well classify the points; the other two methods show optimal results, except for the three-cluster dataset consisting of one cylinder and two blobs, in which the spectral clustering algorithm completely fails in six out twenty cases. This dataset is the only one for which the resulting Fowlkes-Mallows index exhibit a variation, when the random seed of k-means and spectral clustering algorithms is changed.

In conclusion, the considerations regarding the comparison between these three clustering techniques and the Mondrian based one are analogous to those expressed in Section 4.5. The performances of the algorithms, except the case of k-means applied to concave-shaped datasets, are similar; in general, DBSCAN and spectral clustering algorithm are more accurate in the prediction, but the first one requires the knowledge of the minimum distance that separates two clusters and the second one requires as input the number of clusters. The Mondrian based method is less accurate in the classification of the dataset but is able to automatically identify the correct number of clusters without any knowledge on the shape of the dataset; however, this result is not achieved for every dataset (see the case of the three circular clusters in three dimensions). Moreover, other than the predicted data classification, it gives as output a probability distribution and, in particular, it associates to each point of the space a value that estimates the probability to belong to a certain cluster.

	varied3D ($seed = 10$)	***varied3D*** ($seed = 50$)	***varied3D*** ($seed = 150$)	***circles3D***	***cylinder***	***sphere***
k-means	0.980	0.980	0.980	0.445	0.610 ± 0.003	0.583
DBSCAN	0.995	1	0.972	1	1	1
Spectral	1	1	0.990	1	0.9 ± 0.2	1
Mondrian	1	1	0.991 ± 0.007	0.96 ± 0.10	1	0.97 ± 0.08

Table 5.2: Averaged Fowlkes-Mallows index, for each combination of datasets and clustering algorithms; except the case of k-means algorithm, the compatibility with the true labeling is high in all the other cases. For the Mondrian method, the choice of parameters is the one that provides better results.

Chapter 6

Conclusions

The proposed clustering method based on the Mondrian process provides as outcome an estimation of the probability distribution to belong to a certain class, that is defined in each point of the underlying space of the clustered dataset. The fact that the estimation of the class is not exclusively related to the data implies the possibility to classify samples not belonging to the dataset used to obtain the classification; this means that, if the initial dataset is sufficiently representative of a certain population, the clustering procedure is not required to be repeated on different samples of the same population, in order to obtain their corresponding classification.

The probability of estimation is, furthermore, hierarchical, since it is not provided for a single division of the dataset, but is related to a certain range of possible number of clusters. In the majority of the considered cases, the algorithm is however able to automatically recognize the correct number of clusters in which the dataset is divided, by evaluating the similarity of different tree outputs through the adjusted mutual information index. In this case, the classifications obtained at other values of the number of classes provides an estimation, at higher levels, of the further subclassification of each cluster that, in case of real data, may reflects some peculiar features of a subpopulation.

If compared with other notable clustering algorithms, as DBSCAN or spectral clustering, the Mondrian based method provides, in two and three dimensions, slightly lower but comparable and very good performances; on the contrary, the k-means algorithm well clusters data only in case of convex shaped datasets. All the considered techniques require some prior informations and, depending on our prior knowledge on the dataset, each of them may be preferable to the others. The advantage of the Mondrian clustering forest, with respect to spectral clustering, is that it doesn't require as input parameter the number of clusters. The algorithm is in fact able to automatically determine the correct number, as DBSCAN does, but unlike this, it doesn't need to a priori know the minimum distance that separates the clusters. In addition, it provides hierarchical informations over the overstructure and substructure of the dataset; however, it isn't able to take into account the presence of outliers. Finally, the main difference from DBSCAN and spectral clustering is that the Mondrian based technique is able to make predictions about the assignment of extra-dataset points to a specific cluster, providing a probability measure of the estimation; in fact, among the considered three notable algorithm, only the k-means one is characterized by the predictive property, although it is not comparable with the Mondrian method in regards to the quality of the clustering results.

The encouraging results obtained with toy datasets in two and three dimensions suggest that it is worth to deeply investigate the behaviour of the algorithm with datasets characterized by other distributions of data in terms of numerosity, shape and density, higher number of clusters and, expecially, with higher dimensional toy and real datasets. In particular, the algorithm may behave differently with the increase of the number of dimensions, since the concept of

closeness based on the euclidean distance changes as well. Moreover, it may be possible that, even when the unsupervised classification provides good results, the automatic determination of the number of clusters doesn't work.

In this respect, the role of the randomization, in both the reliability of the probability of estimation and in the determination of the number of clusters, needs to be further studied. We saw how different values of the exponent influence the probability of extraction of the hyperplanes and, consequently, the cluster shapes of the forest final result: too low values of the exponent, that correspond to a high randomization, may be related to misclassifications, while too high values cause a more deterministic hyperplane extraction that may be influenced by local characteristics of the data; in both cases, the algorithm may be not able to recognize the correct number of clusters. The exponent can be thus used as regularization parameter of the clustering process; in particular, the influence of the exponent on the level of randomization may depend on the dimensionality and numerosity of the dataset.

An other aspect that requires a deeper investigation is the role of the lifetime λ and, in particular, how the complexity of the tree is related to the complexity of the dataset. Since each cut is associated to a time instant, randomly extracted from an Exponential distribution, and the splitting procedure stops when the absolute time is higher than λ, the lifetime is related to the number of splits performed by the algorithm during the partitioning phase. The value of lambda, in order to be meaningful about the expected number of splits, must be evaluated in relation to the rate of the Exponential distribution, that determines the absolute value of each time increment. In our implementation of the algorithm, the rate is set equal to the volume of the polytope that will be splitted; this means that the number of cuts performed in a certain time interval λ is related to the specific considered dataset and space in which it is defined. For this reason, different datasets may require the setting of different values of the lifetime in order to obtain the same number of cuts that well split the space and, consequently, the comparison between lifetime values of different dataset has no meaning. Moreover, since the initial space is defined as any convex space containing the dataset, although we restricted it to the smallest axis-aligned box containing the samples, the size of each subspace doesn't always gives informations about the quantity of samples belonging to it; it may thus occur than a larger polytope containing few points is splitted more times than a smaller polytope containing a large amount of samples; this is due to the fact that, larger is the subspace (and higher is the rate of the Exponential), sooner the cut occurs. A more significant definition of the rate would be obtained by making it related to the numerosity of the dataset rather than of the volume of its underlying space; this would remove the dependence on the absolute size of the underlying space and would allow to make analogies between different datasets.

Finally, other alternatives to the formulation of the metric and the choice of the hyperplane should be find, in order to obtain variables that better describe specific characteristics of the dataset.

Bibliography

[1] Amit Saxena et al. "A Review of Clustering Techniques and Developments". In: *Neurocomputing* 267 (July 2017). DOI: 10.1016/j.neucom.2017.06.053.

[2] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification, 2nd Ed.* 2001.

[3] Pavel Berkhin. "Survey Of Clustering Data Mining Techniques". In: *A Survey of Clustering Data Mining Techniques. Grouping Multidimensional Data: Recent Advances in Clustering.* 10 (Aug. 2002). DOI: 10.1007/3-540-28349-8_2.

[4] Periklis Andritsos. "Data Clustering Techniques". In: (2002).

[5] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[6] Mayra Z. Rodriguez et al. "Clustering algorithms: A comparative approach". In: *PLOS ONE* 14.1 (Jan. 2019), pp. 1–34. DOI: 10.1371/journal.pone.0210236. URL: https://doi.org/10.1371/journal.pone.0210236.

[7] Ulrike von Luxburg. "A Tutorial on Spectral Clustering". In: *CoRR* abs/0711.0189 (2007). arXiv: 0711.0189. URL: http://arxiv.org/abs/0711.0189.

[8] Matej Balog and Yee Whye Teh. *The Mondrian Process for Machine Learning.* 2015. arXiv: 1507.05181 [stat.ML].

[9] Nguyen Xuan Vinh, Julien Epps, and James Bailey. "Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance". In: *J. Mach. Learn. Res.* 11 (Dec. 2010), pp. 2837–2854. ISSN: 1532-4435.

[10] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. "On Clustering Validation Techniques". In: *Journal of Intelligent Information Systems* 17 (Oct. 2001). DOI: 10.1023/A:1012801612483.

[11] Marina Meila. "Comparing clusterings - an information based distance". In: *Journal of Multivariate Analysis* 98.5 (2007), pp. 873–895. ISSN: 0047-259X. DOI: https://doi.org/10.1016/j.jmva.2006.11.013. URL: https://www.sciencedirect.com/science/article/pii/S0047259X06002016.

[12] David M. W. Powers. *Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation.* 2020. arXiv: 2010.16061 [cs.LG].

[13] Joy A. Thomas Thomas M. Cover. "Entropy, Relative Entropy, and Mutual Information". In: *Elements of Information Theory.* John Wiley and Sons, Ltd, 2005. Chap. 2, pp. 13–55. ISBN: 9780471748823. DOI: https://doi.org/10.1002/047174882X.ch2. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/047174882X.ch2.

[14] Daniel M Roy and Yee Teh. "The Mondrian Process". In: *Advances in Neural Information Processing Systems.* Ed. by D. Koller et al. Vol. 21. Curran Associates, Inc., 2009. URL: https://proceedings.neurips.cc/paper/2008/file/fe8c15fed5f808006ce95eddb7366e35-Paper.pdf.

[15] Jaouad Mourtada, Stphane Gaffas, and Erwan Scornet. “Minimax optimal rates for Mondrian trees and forests”. In: *The Annals of Statistics* 48.4 (2020), pp. 2253–2276. DOI: 10.1214/19-AOS1886. URL: https://doi.org/10.1214/19-AOS1886.

[16] Robert Gallager. *6.262, Discrete stochastic processes*. Tech. rep. License: Creative Commons BY-NC-SA. Massachusetts Institute of Technology: MIT OpenCouseWare, 2011. URL: https://ocw.mit.edu/.

[17] Arne Brøndsted. *An Introduction to Convex Polytopes*. Springer, New York, NY, 1982. DOI: https://doi.org/10.1007/978-1-4612-1148-8.

[18] Branko Grunbaum and Geoffrey C. Shephard. “Convex Polytopes”. In: 1967. DOI: 10.2307/3615008.

[19] *Polytope package of Phyton*. URL: https://github.com/tulip-control/polytope.

[20] Lauriane Castin and Benoît Frénay. “Clustering with Decision Trees: Divisive and Agglomerative Approach”. In: *ESANN*. 2018.

[21] L Castin. “Clustering with Decision Trees: Divisive and Agglomerative Approach”. Master thesis in Computer science. University of Namur, 2017.

[22] Balaji Lakshminarayanan, Daniel M. Roy, and Yee Whye Teh. “Mondrian Forests: Efficient Online Random Forests”. In: *Advances in Neural Information Processing Systems*. 2015. arXiv: 1406.2673 [stat.ML].

[23] B Lakshminarayanan. “Decision trees and forests: a probabilistic perspective”. PhD thesis. University College London, 2016.

[24] Daniel Roy. “Computability, inference and modeling in probabilistic programming”. PhD thesis. Massachusetts Institute of Technology, 2011.

[25] H. Pishro-Nik. *Introduction to probability, statistics, and random processes*. Kappa Research LLC, 2014. URL: https://www.probabilitycourse.com.