Integration and Evaluation of Different Kernel Density Estimates in Hierarchical Density-Based Clustering

by

Kriti Khare

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

© Kriti Khare, 2016

Abstract

Most machine learning methods make assumptions about data. Parametric statistics assume that the data is sampled from a distribution with fixed properties set by the algorithm or user. In contrast, non-parametric statistics do not assume the properties of a distribution. Instead, they assume that the data comes from a distribution and then they estimate the properties of this distribution for each object.

Cluster analysis is a machine learning method for finding meaningful groups in the data when there is little or no prior information available about the data being analyzed. Clustering techniques can be broadly divided into partitional algorithms, that produce unnested clusters, and hierarchical clustering, in which there are nested clusters.

In the last decade, there has been an increased interest in developing non-parametric clustering algorithms with increase in computation power of machines. However, so far, the literature on non-parametric clustering has focused on partitional techniques. This thesis proposes a generalization of the hierarchical density based clustering algorithm, HDBSCAN*, that implements arbitrary non-parametric density estimates.

For that, we first analyze the challenges faced in using kernel density estimates for a hierarchical clustering algorithm, and present different ways of defining connectivity in density space. Kernel density estimates have a parameter known as the bandwidth, which determines the degree of influence of distant objects on the density estimate of the object of interest. We do an exhaustive search for the best bandwidth, and compare the accuracy of the cluster assignment obtained from it with the accuracy obtained by using heuristic bandwidth estimators found in literature.

The evaluation of hierarchical clustering results is typically done by extracting a flat partition from a hierarchy and comparing this extracted partition with a"ground-truth" partition of the same dataset, which may come from class labels given independently to a dataset. In order to truly evaluate the quality of a constructed hierarchy using different density estimates, however, one would need hierarchical ground truth. Hierarchical structures that could be used as ground truth are only available in rare cases, and therefore, we also develop in this thesis, a data generator that produces the hierarchical ground truth along with the dataset to provide additional data sets for our experimental evaluation.

We do an extensive study comparing the accuracy of the cluster assignments when using different strategies for setting the bandwidth for certain Kernels from literature, and comparing it with previous proposals for hierarchical density based clustering based on unnormalized knn-Kernels. Based on the experimental results, using both flat and hierarchical ground truth labels, we conclude that there are connectivity methods that give good results, however, they may be computationally expensive for some Kernels. Also, the heuristic bandwidth estimators do not perform better, in general, than previous proposals based on knn-Kernels. Therefore, new heuristics must be developed for finding the bandwidth to use for clustering purposes with the proposed connectivity methods.

Preface

This thesis is the original work by Kriti Khare. No part of this thesis has been previously published.

To my family,

for believing in me and supporting me through every step, especially, these past two years.

If 'may' is not working, use July!

– Kairav, 6 years old, on sentence structure
and, subconsciously, on research.

Acknowledgements

These past two years have been a memorable journey that would not have been possible without my supervisor, Dr. Jörg Sander. It has been a learning experience working with him. The constructive feedback and guidance he provided have helped me to develop as a researcher.

I am thankful to Dr. Russell Greiner and Dr. Mario Nascimento, who taught graduate level courses that were my introduction to research. With their feedback and encouragement, I have gained many skills. The discussions with Dr. Ricardo J. G. B. Campello were also very helpful.

I would also like to thank my friends, Roberto Vega, Antonio Furatdo Jr., Hamman Samuel, Maryam Mirzaei, Talat Syed, Mohammed Shazan, Luke Kumar, Jadson Castro, Antonio Cavalcante, Fateme Bahri and Queenie Luc, with whom I enjoyed many a conversations on grad life, thesis writing, presentation and research, in general. I learned more about my research field from Dr. Davoud Moulavi and Dr. Pablo A. Jaskowiak.

Most importantly, I am grateful to my family for always being there. Without the support and love of my parents, I would not be here. I will never forget the nights Treena Khare gave me company while I worked on my thesis, and she proofread it. My brother, Kairav, has been supportive in his 6 year old way, always encouraging me to get some exercise on the weekends. I appreciate the effort my boyfriend, Clinton Bunker, put in keeping me positive and focused on my research.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction and Motivation

We live in a world that thirsts for knowledge. Large amounts of data are collected everyday, to be analyzed hoping to find answers to questions that may be asked one day, build strategies that may be needed one day, and find patterns that can help us understand the world better. Whether the data comes from a hospital, represents all of the weekly transactions from a grocery store or consists of the performance of students in a class, it must be analyzed to be meaningful. To do so, some basic categories are required that represent characteristics of the parts of the whole data.

One of the biggest challenges that a data scientist faces, is not knowing how many of these categories actually exist. Cluster analysis is a tool for such tasks where there is little or no prior information available to the scientist about the data being analyzed. Clustering builds these categories based on the properties of the given data, and how similar two objects are. Based on the questions we want answered, cluster analysis may give two kinds of clusterings of data: first, in which each object belongs only to one category and second, where a hierarchy represents the various stages at which objects become more and more similar to each other.

Cluster analysis is an active area of research. In the data mining community, there is no single accepted definition of a cluster. This is the reason why a number of clustering algorithms have been proposed, each interpreting the notion of a cluster in a different way. For example, k-means [46] exemplifies each cluster by using a representative object, DBSCAN [21] perceives

clusters as regions of high density of objects while Gaussian Mixture models [4, 22, 79] have the underlying assumption that the data is drawn from a finite number of Gaussian distributions and hierarchical methods [12, 24, 37] define connectivity between objects to show when they may be similar.

1.1 Density-Based Clustering

With each notion of clustering comes its own properties and parameters. Density based clustering covers numerous algorithms [2, 12, 16, 20, 21, 24] that aim to find clusters without making any assumptions about their shape. The main idea behind density based clustering is that data has been sampled from an unknown probability density distribution. With no prior assumptions about the data, these methods fall into the category of what is known as 'non-parametric' methods in statistics.

Density-based clustering techniques for obtaining a partition of data where each object is assigned a single label have been widely studied. While there has been a focus on using distance between objects to assign them to clusters [2, 21], in the last decade, much research has been done to use kernel density estimates (K.D.E.) for density-based clustering [3, 49].

Alternative to partitional clustering, we have hierarchical cluster analysis, HCA, that aims at building hierarchies of clusters. It can be thought of as building a tree like structure where the lowest level consists of all objects and as we move higher up, these objects are merged together to form specialized groups, based on their properties. Thus, the data can have nested properties. Since the first hierarchical clustering algorithm proposed by Johnson in 1967 [37], numerous hierarchical clustering algorithms have been proposed including a few density-based hierarchical clustering algorithms [2, 12, 24].

In this study, we examine hierarchical cluster analysis based on density, not distance, in detail. HDBSCAN* [10] is one such algorithm that builds a hierarchy based on the density structure of data. It uses the unnormalized k-nearest neighbor (kNN) kernel. This thesis proposes a method to generalize HDBSCAN* to use arbitrary kernel density estimates. We discuss different

edge weight estimation methods to define connectivity between objects using these density estimates and the analyze the effect each method has on the final clustering result.

1.2 Contributions

Our research addresses two major areas in density-based clustering. Our main goal is to use K.D.E. in a hierarchical density-based clustering setting. Performing partitional clustering with K.D.E. has shed some light on how to use these techniques in clustering. However, hierarchical density-based clustering using K.D.E. has been largely untouched. We elaborate on the challenges it has, such as defining connectivity between objects, and the possible ways to solve it.

There is an abundance of artificial data generation programs for clustering – generators by Handl and Knowles [27] and Pei and Zaïane [54] to mention a few. However, no data generator has been proposed before that produces datasets and records their hierarchical structure as well. Thus, to better test our proposed hierarchical algorithm, we created a data generator that generates data with a genuine hierarchical clustering structure. This is the second area we contribute towards for better testing of hierarchical clustering algorithms.

We designed an extensive case study to answer the following questions:

1. For datasets with a single labeling of data (a flat partition) -
    (a) Can integration of an arbitrary kernel density estimate into hierarchical clustering do better than the kNN approach?
    (b) Does one kernel always outperform others?
    (c) Is one of our edge estimation methods always better than the others?
    (d) Since we do not have ground truth in practical applications, can we use an internal validation measure to predict the best parameter to use?

(e) There are statistical methods for estimating bandwidths of kernels. Is it feasible to use them?

2. For datasets with an underlying hierarchical structure -

   (a) Are we capable of identifying the hierarchical structure of a dataset using arbitrary kernels?

   (b) How well the statistical bandwidths perform in the hierarchical setting?

The contributions of this thesis can be summarized as

- We generalize the density based hierarchical algorithm, HDBSCAN*, to use kernel density estimates other than the unnormalized kNN kernel.
- We perform an extensive study of different kernels and connectivity definitions in a Minimum Spanning Tree over a wide range of datasets.
- We also conduct a study of how well statistical bandwidth estimators for the smoothing parameter in kernel density estimates perform when used to set this parameter for hierarchical density-based clustering.
- To facilitate the evaluation of hierarchical density-based clustering algorithms, we propose a hierarchical data generator which generates the data and also outputs the hierarchy for it.

1.3 Outline

This thesis is organized as follows: In Chapter 2, we review the background material and related works, elaborating on non-parametric statistics and density-based clustering as a non-parametric approach. In Chapter 3, we present the algorithm, HDBSCAN* [10] that we want to extend. In Chapter 4, we propose the extension of HDBSCAN* to incorporate arbitrary density estimates. In Chapter 5, we propose a novel data generation algorithm that generates clustering with an underlying hierarchical ground truth. We finally present and discuss an extensive case study using a variety of datasets and kernels on our

algorithm in Chapter 6. Chapter 7 concludes the thesis with ideas for future work.

Chapter 2

Background and Related Work

Density-based clustering methods are non-parametric approaches as they do not make assumptions about the underlying density f of data. In this chapter, we first explain non-parametric statistic approaches and then elaborate on their application in density-based clustering.

2.1 Non-parametric Methods

There are both parametric and non-parametric density estimation methods that have been used in clustering. Consider a random variable $\mathbf{X}$ that is drawn from a random probability density function, $\mathbf{F}$.

Parametric statistic approaches assume that $\mathbf{F}$ belongs to a family of distribution functions that can be described by a small number of parameters, such as the Normal distribution. These parameters are unknown and must be estimated. For example, assume that $\mathbf{X} \sim \mathcal{N}(\mu = 5)$. All our inferences for $\mathbf{X}$ would now be based on these two assumptions. However, if the data had a mean of 10, the inferences would be quite far from reality.

Non-parametric statistics, on the other hand, make few assumptions about the data: $\mathbf{F}$ can be a function that satisfies the definition of a cumulative density function (c.d.f.). Thus, instead of estimating parameters, we estimate functions. As Silverman stated in 1986 [66], "... the data are allowed to speak for themselves in determining the estimate". Non-parametric statistics have a vast application in estimating and testing aspects of $\mathbf{F}$, estimating the density of $\mathbf{X}$, calculating regression functions, and so on. Detailed information about

each of these techniques have been presented by Wasserman [76], Simonoff [67] and Silverman [66]. For this thesis, we concentrate on the estimation of density. Silverman [66] and Scott [64] are well known references on non-parametric density estimation.

2.1.1 Non-parametric Density estimation

Given a set of examples, non-parametric statistics model the density function of the data without making any assumptions about the form of the distribution.

2.1.1.1 Histograms

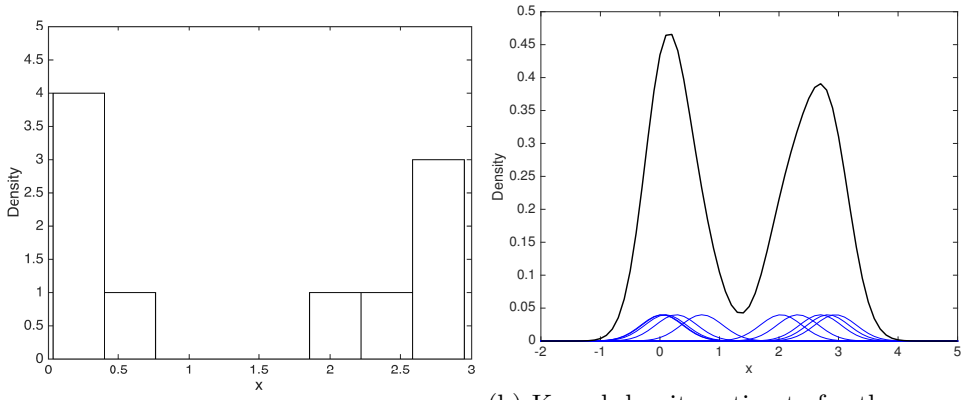
The simplest form of such a density estimation is the histogram in which we divide the sample space into some fixed number of bins. The density at the center of each bin is approximated as the fraction of points that fall into the corresponding bin. Thus, the number of data points and variance of estimator varies from bin to bin. For an object x in a N size data sample, its density in a histogram is given by:

$$P_H(x) = \frac{1}{N} \frac{\text{number of objects in same bin as } x}{\text{width of the bin constaining } x}$$

For histograms, we must specify the size of a bin (length of the interval) and the starting position of the first bin. Though histograms are a great tool for visualizing one or two dimensional data, they cannot be used for high dimensions. Further,the density estimate is highly dependent upon the starting point of the bins, and the number of bins.

2.1.1.2 Kernel Density Estimation

Kernel density estimation (K.D.E.) or *Parzen Rosenblatt window method*, was invented by Rosenblatt [59] and Parzen [53]. The *bandwidth* is a smoothing parameter, similar to the bin width in histograms, that controls the trade off between bias and variance. It determines how much a distant point should influence the density estimate. Unlike histograms, K.D.E. is smooth. Kernel density estimation involves placing a kernel – a non-negative function that integrates to one and has mean 0 – at each point in the data space and summing



(a) Histogram for the data

(b) Kernel density estimate for the same data

Figure 2.1: Figure shows the (a) histogram and (b) kernel density estimate for one dimensional data at x=0.71, 0.03, 0.27, 0.05, 0.10, 2.82, 2.69, 2.32, 2.95, 2.03.

the contribution of every point in the data space to get an estimate of density. The kernel defines the shape of the bumps while the bandwidth represents the width of each kernel. Figure 2.1 compares the density estimates obtained by using the histogram and the Normal kernel.

Mathematical Formulation of K.D.E.

Let $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$ be an independent and identically distributed sample from a random distribution $\mathbf{F}$. We want to estimate the density $f(x) = \mathbf{F}'(x)$. A kernel is any smooth function K with variance σ_K^2 satisfying:

$$\int K(x)dx = 1, \int xK(x)dx = 0 \text{ and } \sigma_K^2 \equiv \int x^2 K(x)dx \geq 0$$

Some commonly used kernels are presented in Table 2.1 and their respective curves are plotted in Figure 2.2.

Definition 1 (Kernel Density Estimator) *In the one dimensional case, given the kernel K and a positive number h, i.e. bandwidth, the* kernel density estimator *is defined to be*

$$\hat{f}_n(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right) \tag{2.1}$$

Kernel	Function
Box (Uniform)	$K(x) = \frac{1}{2}I(x)$
Normal or Gaussian	$K(x) = \frac{1}{\sqrt{2\pi}} \exp^{-x^2/2}$
Epanechnikov	$K(x) = \frac{3}{4}(1-x^2)I(x)$
Tricube	$K(x) = \frac{70}{81}(1-\lvert x\rvert^3)^3 I(x)$

$$\text{where } I(x) = \begin{cases} 1 & \text{if } |x| \leq 1 \\ 0 & \text{if } |x| \geq 1 \end{cases}$$

Table 2.1: Common Kernel Functions

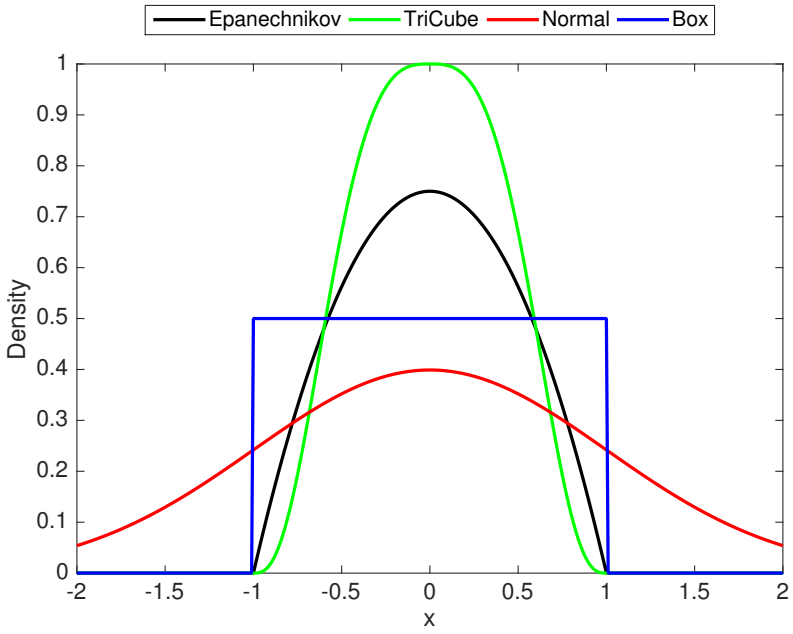


Figure 2.2: Plots of Common Kernels Functions

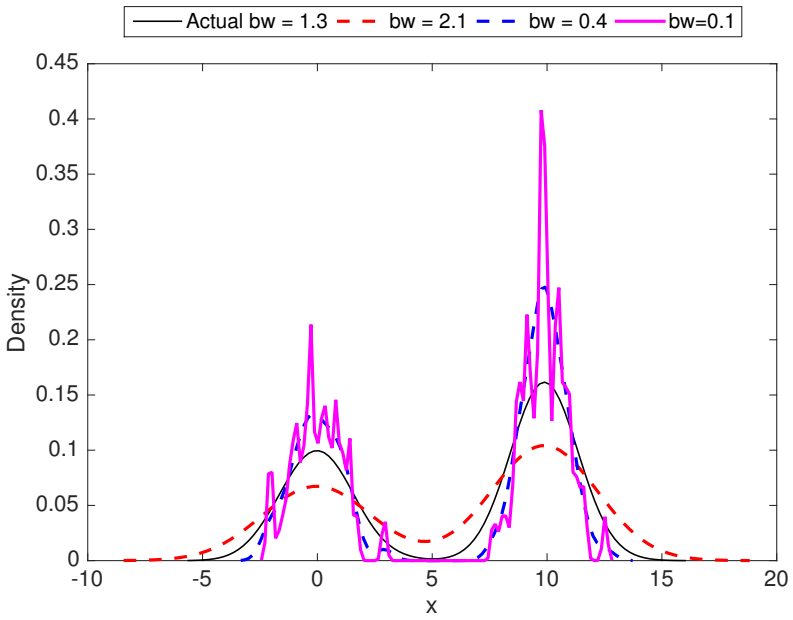


Figure 2.3: A low value for bandwidth allows one to see a more detailed structure of the data. However, with too much detail, the overall structure of the data is hard to see. On the other hand, high values tend to take away the granularity in structure, showing the major density peaks only.

This is equivalent to placing smoothed out peaks of mass of size $\frac{1}{n}$ over each data point X_i. At each point x, $\hat{f}_n(x)$ is the average of the kernels centered over the data points X_i.

If the data are d-dimensional then $X_i = \{X_{i1}, \ldots, X_{id}\}$. For the multidimensional case, the kernel estimator can be generalized to d dimensions using a product kernel where h_j represents the bandwidth in j^{th} dimension.

$$\hat{f}_n(x) = \frac{1}{nh_1 \ldots h_d} \sum_{i=1}^{n} \left\{ \prod_{j=1}^{d} K\left(\frac{x_j - \mathbf{X}_{ij}}{h_j} \right) \right\} \tag{2.2}$$

Bandwidth Selection in K.D.E.

The bandwidth is an important parameter for K.D.E., more crucial than the choice of the kernel itself. For a small bandwidth, a rough – or not smooth – estimate is obtained. Increasing the bandwidth leads to smoother curves. Figure 2.3 shows the effect of the bandwidth value on a sample dataset.

Many bandwidth estimation techniques have been proposed. A review of these techniques has been presented by Turlach et al. [73], Simonoff [67],

Wasserman [76] and Scott [64], Jones et al. [38], Loader [44] to name a few. The most common techniques include cross validation and plug-in formulae such as, normal reference rule [66] and Scott's rule [64]. For multivariate kernels, Bowman and Azzalini's rule [7] and adaptive kernels proposed by Terrell and Scott [71] are some options. We summarize Silverman's rule of thumb or the normal reference rule and cross validation here.

- Silverman' Rule of thumb [66]:
  Given a sample of standard deviation s and interquartile range Q, for smooth densities and a Normal kernel, the bandwidth is given by

$$h_n = \frac{1.06\hat{\sigma}}{n^{1/5}} \tag{2.3}$$

  where

$$\hat{\sigma} = min\left\{ s, \frac{Q}{1.34} \right\}$$

- Cross Validation: Rumedo [61] invented the cross-validation method for histograms and kernel density estimates in 1982. There are numerous cross validation methods ranging from biased [63] and unbiased [6, 61], maximum likelihood [18, 25] and complete cross-validation [39]. In each case, the Asymptotic Mean Integrated Squared Error (AMISE) is being minimized. In Maximum Likelihood Cross Validation [18, 25], the pseudo likelihood is maximized using the leave one out method. The optimal h is

$$h_{mlcv} = \underset{h \geq 0}{\operatorname{argmax}}\, MLCV(h) \tag{2.4}$$

  where

$$MLCV(h) = \left(n^{-1} \sum_{i=1}^{n} \log \left[\sum_{j \neq i} K\left(\frac{X_j - X_i}{h} \right) \right] - \log \left[\frac{n-1}{h} \right] \right)$$

There are many packages in R and the Statistical Toolbox in MATLAB that allow bandwidth estimation.

K.D.E. is a computationally expensive process. Recently, there have been developments in research to make this process faster [57, 65, 81].

In 2010, Raykar et al. [57] proposed a technique that reduces the estimation for one dimension kernels from $\mathcal{O}(mn)$ time to $\mathcal{O}(n+m)$ where n is the number of objects for which density has to be estimated and m is the number of objects that influence density for each object. In 2013, Zheng et al. [81] proposed randomized and deterministic algorithms for kernel density estimation on large datasets. They implemented their techniques such that it can be used with any large scale data processing framework such as MapReduce [62]. Shaker et al. [65] demonstrated a new technique to calculate kernel density estimates and their derivatives for linearly separable kernels, e.g. the Normal and Epanechnikov kernels, with significant savings, especially for high dimensional data and higher order derivatives. They were able to reduce the number of multiplication and derivate calculations, while keeping the estimate value the same as doing all multiplications and derivate calculations.

Oyegue et al. [52] developed a heuristic to estimate which kernel would be better for given multivariate data, taking the Epanechnikov kernel as a baseline.

In computer vision, *scale-space theory* [14, 15] is a non-parametric curve estimation technique. Instead of trying to estimate a single optimal value of bandwidth for the data, scale-space theory involves simultaneously looking at a wide range of bandwidths. Different levels of smoothing may reveal different useful information.

2.1.1.3 The kNN Density Estimate

The k-Nearest Neighbor density estimate is a special form of non-parametric density estimation. By choosing a fixed value of k, the volume of data surrounding the object of interest is calculated. The kNN kernel density estimate was proposed by Loftsgaarden [45] in 1965. For a dataset X with N number of points and dimensionality d, the kNN density estimate value at object x due to k nearest neighbors (inclusive of the point itself) is given by :

$$\hat{f}(x) = kNN(x) = \frac{k-1}{N}\frac{1}{\mathbf{A}(k,N,X)}$$

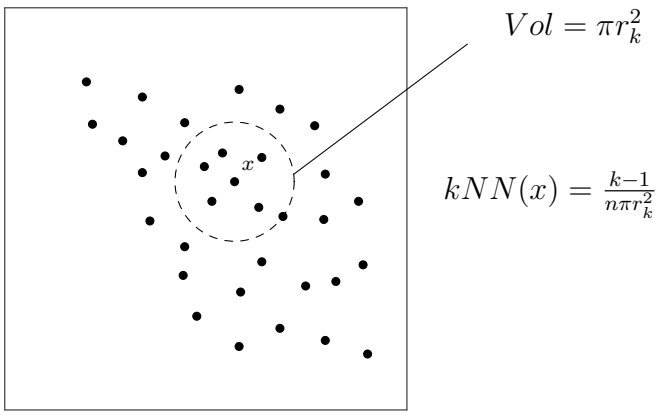


Figure 2.4: Consider an n sample space in two dimensions. For $k = 7$, inclusive of x, the dotted circle shows the volume of the surroundings of x with k points. The kNN estimate value would be $(k-1)/(n\pi r_k^2)$.

where $\mathbf{A}(k, N, X)$ is the volume of the set of points in X whose distance from x is less than r_k, the distance between x and its k^{th} nearest neighbor. In Euclidean distance space, $\mathbf{A}(k, N, X)$ is a hypersphere with radius r_k :

$$\mathbf{A}(k, N, X) = \frac{2r_k^d \pi^{\frac{d}{2}}}{d\Gamma\frac{d}{2}}$$

where Γ represents the Gamma function.

Thus, the kNN kernel density estimate can be written as

$$\hat{f}(x) = kNN(x) = \frac{k-1}{N} \frac{d\Gamma\frac{d}{2}}{2r_k^d \pi^{\frac{d}{2}}} \tag{2.5}$$

A special property of the kNN estimate is that it is multidimensional. Thus, a product kernel is not required for multidimensional data. Figure 2.4 shows the volume around an estimation point, x, in the two-dimensional case.

2.1.1.4 All-Points-Core-Distance kernel

The development of new kernel density estimates is an open field. One new kernel that we know of is a parameterless kernel, recently proposed by Moulavi et al. [51], which uses distance to estimate the density at each point, however, does not require any user defined parameter like k. In [51], this kernel was used for cluster validation and then later, in [50], Moulavi formalized it as a kernel for a dataset.

Definition 2 (All-Points-Core-Distance) *The All-Points-Core-Distance (inverse of the density) of an object* $\mathbf{o}$*, belonging to the* d *dimensional dataset* $\mathbf{X}$ *with respect to all other* $n-1$ *objects in the dataset is defined as :*

$$all_{pts}coredist(\mathbf{o}) = \left(\frac{\sum_{o_i \in \mathbf{X}, o_i \neq \mathbf{o}} \left(\frac{1}{d(\mathbf{o}, o_i)} \right)^d}{n-1} \right)^{-1/d} \tag{2.6}$$

Here $d(\cdot, \cdot)$ represents the pairwise distance between a pair of objects.

Using this definition, the kernel density estimate of an object x is given by

Definition 3 (All-Points-Core-Distance kernel)

$$\hat{f}(x) = all_{pts}coredist(x) \tag{2.7}$$

The kernel estimates the density of objects. To define the denisty between objects, Moulavi [50] proposed the New Mutual Reachability Distance.

Definition 4 (New Mutual Reachability Distance) *The* New Mutual Reachability Distance *between two objects* $\mathbf{x_p}$ *and* $\mathbf{x_q}$ *in dataset* $\mathbf{X}$ *with respect to their core distances and the distance between them is defined as*

$$d_{nmreach}(\mathbf{x_p}, \mathbf{x_q}) = \frac{d_{core}(\mathbf{x_p}) + d_{core}(\mathbf{x_q})}{2} + d(\mathbf{x_p}, \mathbf{x_p}) \tag{2.8}$$

$d_{nmreach}$ is an improvement over the Mutual Reachability Distance defined by Lelis and Sander [41], and used in OPTICS [2] and HDBSCAN* [10]. This new definition is symmetric and by using the average of core distances and adding the distance, one can differentiate whether an object with low density is close to a dense object or the same object with low density is close to a non-dense object.

2.2 Density-Based Clustering

To the best of our knowledge, the first attempt at clustering using the notion of 'density' was by Wishart in the 1960s. He proposed *One Level Mode Analysis* and its hierarchical version, *Hierarchical Mode Analysis – HMA* [77]. For each object in a dataset, HMA found the radius of a sphere in which the object would have n_ε number of neighbors and then a hierarchy of object labels was

built over multiple iterations by sorting the radii and maintaining the distance of an object and its n_ε nearest neighbors. This radius of a sphere is precisely the value of r_k in the kNN density estimate in Equation (2.5).

The concepts proposed by Wishart [77] are extensively used in modern day density-based clustering algorithms. For example, Wishart argued that the modes of the underlying density function should determine the "natural" groupings of the dataset, i.e. for a given probability distribution f and the density level λ "... if f has two or more modes at the level of probability λ, then the covering will be partitioned into two or more disjoint connected subsets of points."

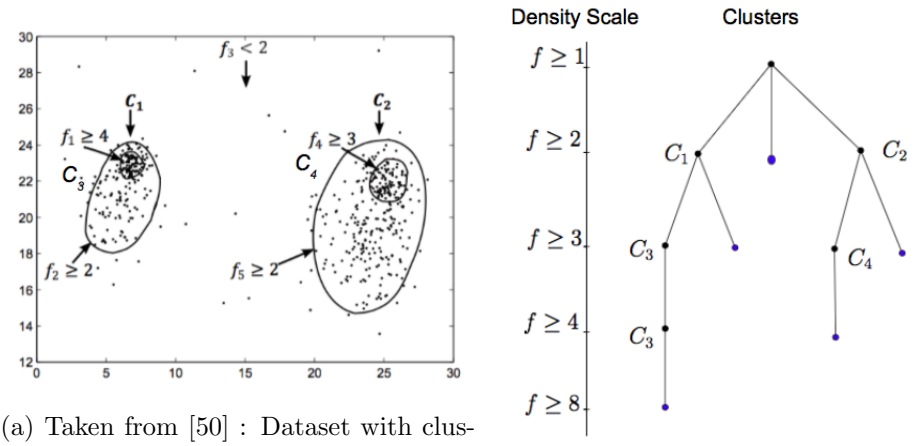
The 1971 dissertation of Bryan [8] is one of the first works using kernel density estimation techniques for classification and clustering. For clustering, nearest neighbor information was used to decide the 'modes' in data and then a hill climbing technique was used to assign labels to the objects.

Later in 1975, Hartigan extended Wishart's ideas and defined the concept of *density-contour clusters* and *density contour tree*. He stated that

> Clusters may be thought of as regions of high density separated from other such regions by regions of low density.

For example, consider this two dimensional dataset in Figure 2.5a with 4 clusters, C_1, C_2, C_3 and C_4, where C_3 and C_4 are embedded in C_1 and C_2 respectively. The density in each region is given by f. The clusters are separated from each other by regions of low density. He showed in [28] that the using the above concept, we could get a hierarchical structure of high density clusters, forming a tree, as shown in Figure 2.5b. When the density level is at least 1, all the points would exist. But as we increase the density, the regions of lower density would separate the density regions, lower density points becoming noise (shown in blue).

Hartigan's book [28], as well as most of the related literature it is based on, uses the concept of distance as a density estimate. There were papers that pursued Hartigan's concept of separability between high density clusters, the most notable of which is by Wong and Lane [78] that uses a kNN density



(a) Taken from [50] : Dataset with clusters and density levels

(b) Cluster tree

Figure 2.5: Figure shows the (a) sample dataset with clusters and density of each cluster, given by f and (b) the cluster tress obtained by varying the density.

estimate in their clustering procedure. The tree structure of the clusters, however, is not examined.

Many papers in statistics and data mining have adopted methods that use density estimates. In the area of data mining, the most notable technique that comes close to Hartigan's definitions is DBSCAN [21]. Using the number of points in the vicinity of an object, the cluster structure of data is estimated. The extension of DBSCAN to produce a hierarchical clustering is OPTICS [2].

In 1997, Roberts [58] proposed two clustering algorithms - one parametric and the other non-parametric. The parametric approach was based on Gaussian Mixture Models and the non-parametric approach performed successive smoothing with the Gaussian kernel. He also tested his two algorithms using a two-step framework. In the first step, model fitting was performed and in the second, the model was validated. Roberts found that the non-parametric approach was more robust.

With an increase in computational power, the direct use of kernel density estimates into clustering algorithms has seen a rising popularity in the two last decades. In its earliest version, DENCLUE [33] was a K.D.E. based approach for clustering for multimedia datasets. K.D.E. was used to model the overall point density, and clusters were identified by using the gradient of the density function. In its later extension, DENCLUE [32] combined hill climbing

techniques with density estimation. Instead of using all points, a sampling method was proposed that would sacrifice some accuracy, however, would be faster. The bandwidth is manually tuned and two user defined parameters are required to control the neighborhood and to define lowest density level.

Nearest neighbors have been used widely in clustering. Sometimes, the kNN distance is employed, other times the kernel is used, and some algorithms directly count the number of shared neighbors. Ertoz et al [20] defined two objects to be similar in terms of the number of neighbors they shared, alleviating the problems of varying density and high dimensionality.

In 2006, Tran et al [72] proposed a clustering technique that uses a combination of the kNN kernel with other kernels for high dimensional data.

In 2007, Azzalini and Torelli [3] proposed a novel clustering technique using kernel density estimation by using connectivity from Delaunay triangulation. They used a pre-calculated bandwidth for their experiments. Delaunay triangulation, however, is limited to lower dimensions because of its computational complexity. Recently, Menardi and Azzalini [49] proposed an advanced version of the previous method [3], trying to overcome the shortcomings of using the Delaunay triangulation. Any two observations in a multi-dimensional setting are connected if the density function, evaluated along the segment joining the two observations, does not have a local minimum, i.e. the density value does not drop on the path. Thus, by measuring the extent of valleys of density along the segment connecting pairs of objects, groups in data are identified. Both of these clustering techniques give flat partitions, i.e. one single labeling of data.

CORE [70] is a parameterless algorithm that integrates grid techniques with density estimation. It uses an AD-tree to store the summary of kernel density estimation of data points. The user does not have to define the kernel or its bandwidth. 'Core' points are defined as objects which are robust to small density fluctuations. Each node of the AD-Tree is a d-dimensional grid that stores the density values at grid points. Once the core objects have been identified, grid points are assigned to each core object. The local and global density maximas are found using a sample of the data on the grid.

Non-parametric mixture models for clustering have also been proposed. Recently, Mallapragada and Jain [48] developed a novel technique that is less sensitive to kernel bandwidth and gives a probability based assignment of data points to different clusters.

In the statistical literature, Cuevas et al [16, 17] recently tried to address the problem of automatically selecting the number of clusters in data using a convolution kernel. Stuetzle at al. [68, 69] proposed a cluster tree to keep track of the density in the dataset. The modes of density correspond to the leaves of the cluster tree. Estimation of this cluster tree was tackled as a non-parametric problem by a graph-based method. Excess of mass was used as a measure for size of the each branch of the tree and could be used for pruning the cluster tree.

As we have seen, there have been numerous algorithms that have used K.D.E., however, the integration of nonparametric techniques into hierarchical density based clustering has been lacking. The hierarchical clustering algorithms closely related to our work are HDBSCAN* [10], OPTICS [2] and AUTO-HDS [24].

HDBSCAN* [10] can be regarded as an improvement over OPTICS [2]. It employs the concepts introduced in DBSCAN [21] and formalizes them to produce a tree of clusters. Campello et al. [10] show that it adheres to the model of density contour clusters that Hartigan [28] proposed. In this thesis, we base our algorithm on HDBSCAN* and elaborate on it in detail in the next chapter.

AUTO-HDS [24] is a hierarchical clustering algorithms for micro-array datasets. Thus, it was proposed to handle large datasets. It aims to simplify clustering hierarchies, referring back to the work of Herbin et al. [30]. As has been shown by Campello et al. [9], the clustering hierarchy obtained from AUTO-HDS is a subset of the one obtained by HDBSCAN*. The AUTO-HDS framework was found to have two major limitations – first, the user-defined parameter, r_{shave}, had to be sufficiently low to prevent underestimation of the stability of the clusters and/or to not miss clusters, and secondly, the stability measure for one cluster may be affected by clusters on other branches.

In 1971, Zahn [80] formalized the problem of detecting inherent separations between objects in a metric space. In his seminal paper, he studied the properties of Minimum Spanning Trees (MSTs), and why they are a powerful tool for detecting and describing structures of point clusters. Many MST-based algorithms were proposed that were able to identify severals kinds of cluster structures. Hartigan's analysis of single linkage for identifying high density clusters shows that in high dimensions, single linkage can detect modes in data that are separated by a sufficiently deep valley [29].

Since different density clusters are not caught by the standard Euclidean distance MST, Wang et al [75] proposed a new edge weight measure based on kNN distance and the distance in Euclidean Space. Their algorithm is similar to HDBSCAN* – a MST is obtained from the new edge weights and the longest edges are removed to form the clusters.

Chapter 3

HDBSCAN*: Hierarchical Density Based Clustering

Hierarchical clustering algorithms do not require any prior information about the number of clusters and output an informative hierarchical structure that shows how the objects can be grouped together, and that can be visualized. HDBSCAN* or Hierarchical DBSCAN* [10] is a recent novel density-based hierarchical clustering approach that can be seen as a conceptual and algorithmic improvement over OPTICS [2]. It generates a complete density-based clustering hierarchy that represents all DBSCAN-like solutions for all possible density thresholds. The hierarchy can be visualized using a reachability plot, a silhouette-like plot, a dendrogram or a compact cluster tree. Also, a simplified tree of significant clusters can be extracted from the hierarchy. In their paper [10], Campello et al. describe not just the algorithm, HDBSCAN*, but also present a way of extracting a non-overlapping collection of clusters from the hierarchy. In a more detailed paper, [12], they present a novel outlier detection method, "Global-Local Outlier Score from Hierarchies (GLOSH) with HDBSCAN*.

HDBSCAN* has a single input parameter, m_{pts}. It is a classic smoothing factor whose role is well understood in literature. HDBSCAN* is based on a reformulation of the algorithm DBSCAN [21], called DBSCAN*. DBSCAN* conceptually finds clusters as the connected components of a graph in which the objects are vertices, and every pair of vertices is adjacent if, and only if, the corresponding objects are ε-reachable with respect to the user-defined

parameters ε and m_{pts}, where m_{pts} is the number of nearest neighbors to consider and ε is a user defined distance such that points with at least m_{pts} in the circle of this distance are dense. Different density levels in the resulting density-based cluster hierarchy will then correspond to different values of the radius ε.

In this chapter, we present HDBSCAN* as presented by the Campello et al. [10]. We first describe DBSCAN* in Section 3.1, HDBSCAN* in Section 3.2, its hierarchy simplification, computational complexity and the technique for extraction of flat partitioning from the hierarchy based on cluster stability in Sections 3.3, 3.4 and 3.5 respectively.

3.1 DBSCAN*

Following the definitions used in Campello et al. [10], let $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ be a dataset of n d-dimensional objects. Let D be an $n \times n$ matrix containing the pairwise distances, $d(\mathbf{x}_p, \mathbf{x}_q)$ between each pair of objects $\mathbf{x}_p$ and $\mathbf{x}_q \in \mathbf{X}$.

Definition 5 (Core Object) *An object $\mathbf{x}_p$ is called a core object w.r.t. ε and m_{pts} if its ε-neighborhood contains at least m_{pts} many objects, i.e., if $|N_\varepsilon(\mathbf{x}_p)| \geq m_{pts}$, where $N_\varepsilon(\mathbf{x}_p) = \{\mathbf{x} \in \mathbf{X} | d(\mathbf{x}, \mathbf{x}_p) \leq \varepsilon\}$ and $|\cdot|$denotes cardinality. An object is noise if it is not a core object.*

Figure 3.1 shows a core object, $\mathbf{x}_p$ w.r.t. $m_{pts} = 6$ and given ε.

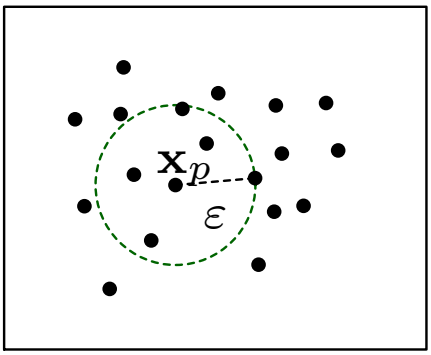


Figure 3.1: $\mathbf{x}_p$ is a *core object* w.r.t. $m_{pts} = 6$ and given ε.

Definition 6 (ε-Reachable) *Two* core *objects $\mathbf{x}_p$ and $\mathbf{x}_q$ are ε-reachable w.r.t. ε and m_{pts} if $\mathbf{x}_p \in N_\varepsilon(\mathbf{x}_q)$ and $\mathbf{x}_q \in N_\varepsilon(\mathbf{x}_p)$*

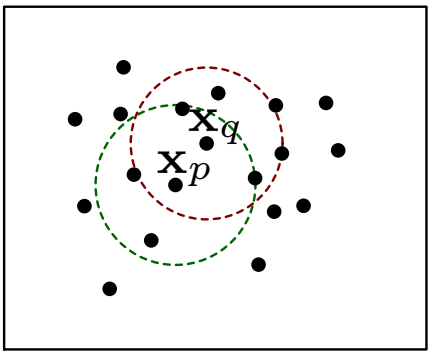


Figure 3.2: $\mathbf{x}_p$ and $\mathbf{x}_q$ are ε-reachable as they are within the ε-neighborhood of each other.

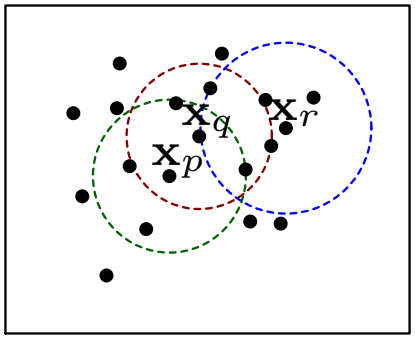


Figure 3.3: $\mathbf{x}_p$ and $\mathbf{x}_r$ are density-connected via X_q.

Figure 3.2 illustrates an example of two $\varepsilon-$reachable objects for a given ε.

Definition 7 (Density-connected) *Two* core *objects* $\mathbf{x}_p$ *and* $\mathbf{x}_q$ *are* density-connected *w.r.t.* ε *and* m_{pts} *if they are directly or transitively* ε*-reachable.*

An example of density-connected objects can be seen in Figure 3.3.

Definition 8 (Cluster) *A* cluster $\boldsymbol{C}$ *w.r.t.* ε *and* m_{pts} *is a non-empty maximal subset of* $\mathbf{X}$ *such that every pair of objects in* $\mathbf{C}$ *is density-connected.*

A sample cluster is shown in Figure 3.4.

Based on these definitions, DBSCAN* (similar to DBSCAN) can be devised that conceptually finds clusters as the connected components of a graph in which the objects of $\mathbf{X}$ are vertices, and every pair of vertices is adjacent if and only if, the corresponding objects are ε-reachable w.r.t. user-defined parameters ε and m_{pts}. The only difference between DBSCAN and DBSCAN* is the presence of *border* objects, i.e., non-core objects that are within the ε-

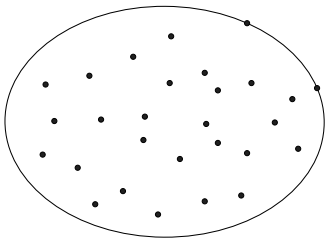


Figure 3.4: Example of a cluster.

neighborhood of a core object, are present in DBSCAN but labeled as noise in DBSCAN* as their estimated density is below the defined threshold.

3.2 Algorithm HDBSCAN*

Following the work Lelis and Sander [41] and Campello et al. [10], the density-based hierarchy formulation can be explained with the help of the following definitions:

Definition 9 (Core Distance) *The* core distance *of an object $x_p \in X$ w.r.t. m_{pts}, $d_{core}(\mathbf{x}_p)$, is the distance from $\mathbf{x}_p$ to its m_{pts}-nearest neighbor (including $\mathbf{x}_p$).*

This is the minimum radius ε for which $\mathbf{x}_p$ satisfies the conditions to be a core object w.r.t. ε and m_{pts}. Figure 3.5 shows an object and its core distance.

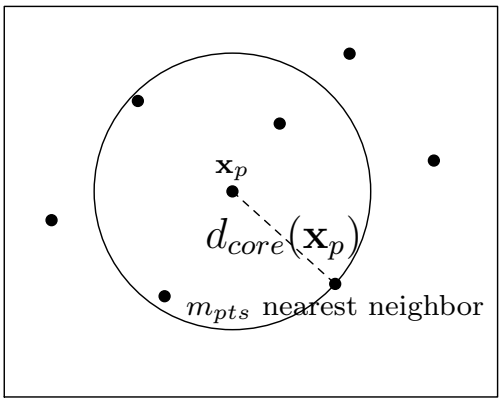


Figure 3.5: $d_{core}(\mathbf{x}_p)$ is the core distance of $\mathbf{x}_p$ w.r.t. $m_{pts} = 5$ and given ε.

Definition 10 (ε-Core Object) *An object $\mathbf{x}_p \in \mathbf{X}$ is called an ε-core object for every value of ε that is greater than or equal to the core distance of $\mathbf{x}_p$ w.r.t. m_{pts}, i.e., if $d_{core}(\mathbf{x}_p) \leq \varepsilon$.*

Definition 11 (Mutual Reachability Distance) *The* mutual reachability distance *between two objects $\mathbf{x}_p$ and $\mathbf{x}_q$ in $\mathbf{X}$ w.r.t. m_{pts} is defined as*

$$d_{mreach}(\mathbf{x}_p, \mathbf{x}_q) = max\{d_{core}(\mathbf{x}_p), d_{core}(\mathbf{x}_q), d(\mathbf{x}_p, \mathbf{x}_q)\} \tag{3.1}$$

Definition 12 (Mutual Reachability Graph) *This is a complete graph, $G_{m_{pts}}$, in which the objects of $\mathbf{X}$ are vertices and the weight of each edge is the Mutual Reachability Distance (w.r.t. m_{pts}) between the respective pair of objects.*

From definitions 4, 6 and 8, it can be inferred that the connected components of ε-core objects in the graph obtained by removing all edges from $G_{m_{pts}}$, having weights greater than ε, are the clusters according to DBSCAN* w.r.t. m_{pts} and ε. The remaining objects are noise. Thus, all DBSCAN* partitions for $\varepsilon \in [0,\infty)$ are obtained in a nested, hierarchical way by removing edges in decreasing order of weight from $G_{m_{pts}}$. This is equivalent to running Single-Linkage [37] over the transformed space of Mutual Reachability Distances and cutting the resulting dendrogram at level ε of its scale. All resulting singleton clusters would be labeled "noise" and the connected components would be the "clusters".

The MST in density space shows the connection of all objects in space such that the path connecting any two objects is the path of minimum density between them. Hartigan stated that regions of high density are separated by regions of low density [28]. The MST is able to capture this change in density, i.e., if there are two clusters in a dataset, then the edge in the MST connecting them, is the edge with the minimum density between the two clusters. Thus, the separation between clusters is captured.

A density-based cluster hierarchy must represent the fact, that an object $\mathbf{o}$ is noise below the level l that corresponds to $\mathbf{o}$'s core distance. To achieve this, the Minimum Spanning Tree (MST) of the Mutual Reachability Graph

$G_{m_{pts}}$ is extended to include self-loops, i.e., edges connecting each vertex to itself, where the edge weight for each vertex $\mathbf{o}$ is the core distance of $\mathbf{o}$. When removing edges, these "self-edges" are also removed. The different density levels correspond to different values of the radius ε in the hierarchy from HDBSCAN* w.r.t. m_{pts}.

The pseudo-code for HDBSCAN* algorithm is shown in Algorithm 1.

Algorithm 1 HDBSCAN* Main Steps

Input: Dataset X, parameter m_{pts}
Output: HDBSCAN* hierarchy

1. Compute the core distance w.r.t. m_{pts} for all objects in $\mathbf{X}$.
2. Compute an MST of $G_{m_{pts}}$, the Mutual Reachability Graph.
3. Extend the MST to obtain MST_{ext}, by adding for each vertex a "self edge" with the core distance of the corresponding object as weight.
4. Extract the HDBSCAN* hierarchy as a dendrogram from MST_{ext}:
   - 4.1 For the root of the tree assign all objects the same label (single "cluster").
   - 4.2 Iteratively remove all edges from MST_{ext} in decreasing order of weights (in case of ties, edges must be removed simultaneously):
     - 4.2.1 Before each removal, set the dendrogram scale value of the current hierarchical level as he weight of the edge(s) to be removed.
     - 4.2.2 After each removal, assign labels to the connected component(s) that contain(s) the end vertex(-ices) of the removed edge(s), to obtain the next hierarchical level: assign a new cluster label to a component if it has at least one edge, else assign it a null label ("noise").

3.3 Hierarchical Simplification

Campello et al. [10] proposed a simplification of the HDBSCAN* hierarchy that produces a summarized tree of only "significant" clusters extracted from the HDBSCAN* dendrogram. It is based on Hartigan's concept of *rigid clusters* [28], and pertains to a fundamental observation about estimates of the level sets of continuous-valued probability density functions(p.d.f.).

For a given p.d.f., there are three possibilities for the evolution of the

connected components of a continuous density level set when increasing the density level (decreasing ε in our context):

1. the component shrinks but remains connected, up to a density threshold at which either
2. the component is divided into smaller ones, or
3. it disappears.

By selecting only those hierarchical levels in which new clusters arise by a "true" split of a cluster, or in which clusters disappear, the most significant changes that occur in the clustering structure are captured. Other levels of the hierarchy at which data objects are assigned null labels and become "noise" are not individually maintained in this simplified hierarchy.

In many practical applications of clustering, to prevent the algorithms from finding very small clusters of objects, the user defines a minimum cluster size. Based on this well accepted practical approach, HDBSCAN* also uses a minimum cluster size, denoted by m_{clSize}, for hierarchical simplification. With $m_{clSize} \geq 1$, components with fewer than m_{clSize} objects are disregarded, and their disconnection from a cluster does not establish a "true" split. HDBSCAN* is adapted according by changing Step 4.2.2 of Algorithm 1 as shown in Algorithm 2.

Algorithm 2 HDBSCAN* step 4.2.2 with (optional) parameter $m_{clSize} \geq 1$

4.2.2 After each removal (to obtain the next hierarchical level), process one at a time each cluster that contained the edge(s) just removed, by relabeling the resulting connected subcomponent(s):

- Label *spurious* subcomponents as noise by assigning them the null label. If all subcomponents of a cluster are *spurious*, then the **cluster has disappeared**.
- Else, if a single subcomponent of a cluster is *not spurious*, keep its original cluster label (**cluster has just shrunk**).
- Else, if two or more subcomponents of a cluster are *not spurious*, assign new cluster labels to each of them (**"true" cluster split**).

The optional parameter m_{clSize} represents an independent control for the smoothing of the resulting cluster tree, in addition to m_{pts}. To simply the

use of HDBSCAN*, $m_{clSize} = m_{pts}$, turning m_{pts} into a single parameter that acts as both a smoothing factor and a threshold for the cluster size. Figure 3.6 shows the dendrogram for the hierarchy of a sample dataset. The distance to the 4^{th} nearest neighbor is considered to calculate core distance. Since $minClSize = minPts = 4$, a cluster is formed when there are at least 4 points within the density level. For all values of ε below this level, the points are considered as noise.

3.4 Computational Complexity

When the dataset $\mathbf{X}$ is available, the complexity of HDBSCAN* is $\mathcal{O}(dn^2)$ when d is the dimensionality of the dataset. The longest time is spent on computing the k-NN queries for each object. Detailed complexity of each step in Algorithm 1 can be found in Campello et al. [10]. In terms of main memory requirements, $\mathcal{O}(dn)$ space is needed to store the dataset, core distances, edges of the MST_{ext} and current hierarchical level being processed.

When the distance matrix, D, is given *instead* of the dataset $\mathbf{X}$, one can promptly access any distance $d(\cdot, \cdot)$ from D in constant time. Thus, the com-

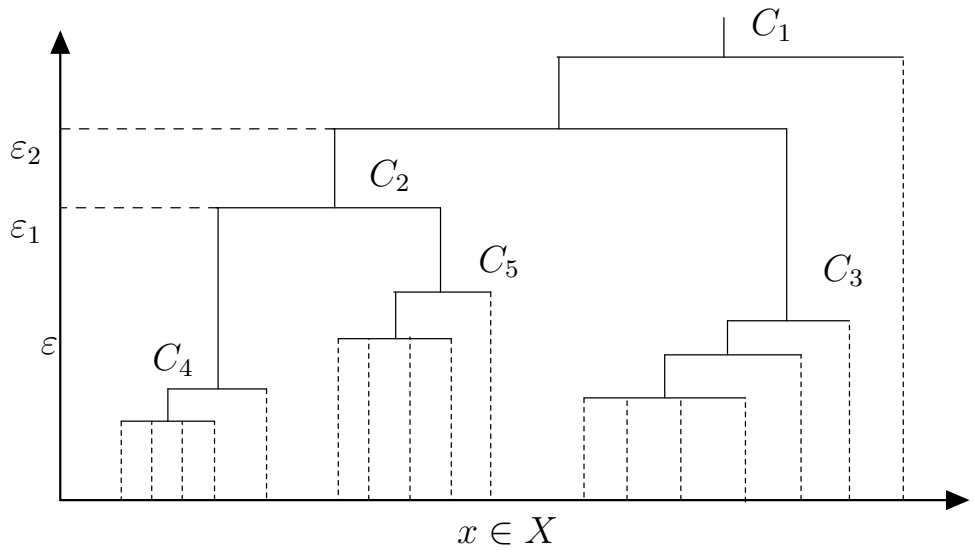


Figure 3.6: Dendrogram for a sample dataset: HDBSCAN* hierarchy for a sample dataset for $minClSize = minPts = 4$. Thinner dashed lines denote noise.

putations are no longer dependent on the dimensionality, d of the dataset and the time complexity reduces to $\mathcal{O}(n^2)$. However, the memory requirements increase to $\mathcal{O}(n^2)$.

3.5 Extraction of Prominent Clusters

Often in clustering applications, the user is interested in a *flat* or *partition-like* solution that consists of the most prominent, non-overlapping clusters. One of the approaches to extract such a partition from a hierarchy is to perform a horizontal cut through a dendrogram. Since this technique corresponds to a single, global density threshold and would [10] describe a novel optimal solution to extract clusters of varying densities by performing *local* cuts through the HDBSCAN* hierarchy.

3.5.1 Cluster Stability

According to Hartigan's model, density-contour clusters of a given density $f(x)$ on $\mathbb{R}$ at a given density level λ are the maximal connected subsets of the level set defined as $\{x|f(x) \geq \lambda\}$ [28]. DBSCAN* estimates the density-contour clusters for a density threshold $\lambda = \frac{1}{\varepsilon}$ and unnormalized kNN estimate (for $k = m_{pts}$) of the density $f(x)$, given by $\frac{1}{d_{core}(x)}$.

HDBSCAN* produces all possible DBSCAN* solutions w.r.t. a given value of m_{pts}, and all thresholds $\lambda = \frac{1}{\varepsilon}$ in $[0,\infty)$. As the value of λ increases (i.e., ε decreases), clusters become smaller and smaller, until they disappear or break into sub-clusters; more prominent clusters "survive" longer after they appear, which is essentially the rationale behind *cluster lifetime* in classic hierarchical cluster analysis [23, 35]. In a traditional dendrogram, the length of the dendrogram scale along those hierarchical levels in which the cluster exists is the lifetime of the cluster.

In HDBSCAN*, a cluster can shrink, disappear or split into sub-clusters. A cluster is dissolved when it disappears by receding into singleton noise objects, or when the number of objects in the cluster falls below m_{clSize}, or when it splits into two or more prominent clusters due to removal of edges. Since a

data object belonging to a cluster can become noise at a density level which may be different from the density level at which the cluster splits or disappears, the stability of a cluster no longer depends on the lifetime of the cluster alone; it takes into account the contributions of each constituent data object.

Keeping this in mind, the contribution of an object x_p towards the density of cluster C_i is given by:

$$\lambda_{max}(x_p, C_i) - \lambda_{min}(C_i)$$

where $\lambda_{max}(x_p, C_i)$ is the maximum value of threshold λ at which the object x_p is part of the cluster C_i and $\lambda_{min}(C_i)$ is the density level at which the C_i came into existence.

For a HDBSCAN* hierarchy for a finite dataset $\mathbf{X}$, cluster labels and density thresholds associated with each hierarchical level, the *stability* of a cluster is defined as:

$$\begin{aligned} S(\mathbf{C}_i) &= \sum_{\mathbf{x}_j \in \mathbf{C}_i} \left(\lambda_{max}(x_p, \mathbf{C}_i) - \lambda_{min}(\mathbf{C}_i) \right) \\ \Rightarrow S(\mathbf{C}_i) &= \sum_{\mathbf{x}_j \in \mathbf{C}_i} \left(\frac{1}{\varepsilon_{min}(\mathbf{x}_j, \mathbf{C}_j)} - \frac{1}{\varepsilon_{max}(\mathbf{C}_i)} \right) \end{aligned} \tag{3.2}$$

3.5.2 Cluster Extraction as an Optimization Problem

Let $\mathbf{C}_2, \ldots, \mathbf{C}_k$ be the collection of all clusters in the simplified cluster hierarchy (tree) generated by HDBSCAN*, except the root $\mathbf{C}_1$, and let $S(\mathbf{C}_i)$ denote the stability value of each cluster. The problem of extracting a flat, non-overlapping clustering solution of the "most" prominent clusters (plus possibly noise) is formulated by Campello et al. [10] as an optimization problem with the objective of maximizing the overall aggregated stabilities of the extracted clusters, in the following way:

$$\max_{\delta_2, \ldots, \delta_k} \quad J = \sum_{i=2}^{k} \delta_i S(\mathbf{C}_i) \tag{3.3}$$

subject to

$$\begin{cases} \delta_i \in \{0, 1\}, & i = 2, \ldots, k \\ \sum_{j \in \mathbf{I}_h} \delta_j = 1, & \forall h \in \mathbf{L} \end{cases}$$

where $\delta_i (i = 2, \ldots, k)$ indicates whether cluster $\mathbf{C}_i$ is included in the flat solution ($\delta_i = 1$) or not ($\delta_i = 0$), $\mathbf{L} = \{h | \mathbf{C}_h$ is a leaf cluster$\}$ is the set of indexes of leaf clusters, and $\mathbf{I}_h = \{j | j \neq 1$ and $\mathbf{C}_j$ is ascendant of $\mathbf{C}_h (h$ included$)\}$ is the set of indexes of all clusters on the path from $\mathbf{C}_h$ to the root (excluded). The constraints prevent nested clusters on the same path to be selected.

To solve Problem (3.2), every node except the root is processed, starting from the leaves (bottom-up), deciding at each node $\mathbf{C}_i$ whether $\mathbf{C}_i$ or the nest-so-far selection of clusters in $\mathbf{C}_i$'s subtrees should be selected. To be able to make this decision locally at $\mathbf{C}_i$, the total stability of $\hat{S}(\mathbf{C}_i)$ of clusters selected in the subtree rooted at $\mathbf{C}_i$ is propagated and updated in the following, recursive way:

$$\hat{S}(\mathbf{C}_i) = \begin{cases} S(\mathbf{C}_i), & \text{if } \mathbf{C}_i \text{ is a leaf node} \\ \max\{S(\mathbf{C}_i), \hat{S}(\mathbf{C}_{il}) + \hat{S}(\mathbf{C}_{ir})\}, & \text{if } \mathbf{C}_i \text{is an internal node} \end{cases}$$

where $\mathbf{C}_{il}$ and $\mathbf{C}_{ir}$ are the left and right children of $\mathbf{C}_i$ (for the sake of simplicity, Campello et al. [10] discuss the case of binary trees).

Algorithm 3 gives the pseudo-code for finding the optimal solution to Problem (3.2).

Algorithm 3 Solution to Problem (3.1)

1. Initialize $\delta_2 = \ldots = \delta_k = 1$, and, for all leaf nodes, set $\hat{S}(\mathbf{C}_h) = S(\mathbf{C}_h)$.
2. Starting from the deepest levels, do bottom-up (except for the root):
   - 2.1 If $S(\mathbf{C}_i) \leq \hat{S}(\mathbf{C}_{il}) + \hat{S}(\mathbf{C}_{ir})$, set $S(\mathbf{C}_i) = \hat{S}(\mathbf{C}_{il}) + \hat{S}(\mathbf{C}_{ir})$ and set $\delta_i = 0$.
   - 2.2 Else: set $\hat{S}(\mathbf{C}_i) = S(\mathbf{C}_h)$ and set $\delta(\cdot) = 0$ for all clusters in $\mathbf{C}_i$'s subtrees.

Figure 3.7 illustrates a density function with cluster stabilities and excess of mass. The root splits into two child clusters – C_2 and C_3. C_2 has two denser regions, C_4 and C_5.

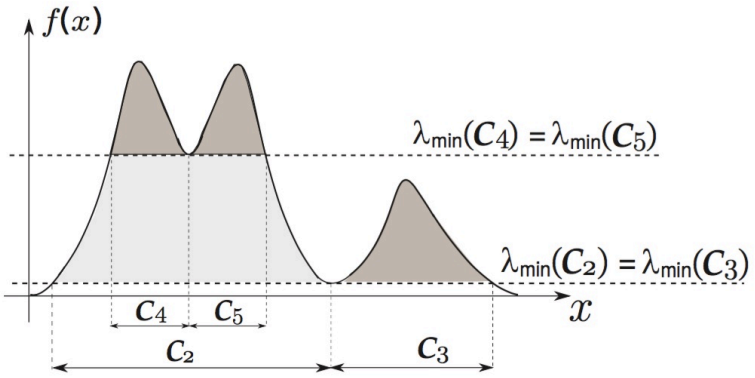


Figure 3.7: Taken from [10]: Illustration of a density function, clusters and excess of mass.

Figure 3.8 shows the example for optimal selection of clusters from a given cluster tree. For this tree, C_4 and C_5 are chosen as separate clusters instead of their parent C_2 since the sum of their stability is less than the stability of C_2. The child clusters of C_5 have lower stability values and are not prominent clusters. Similarly, the child clusters from C_3 are not as stable as their parent and as a result C_3 should be extracted. Note that C_3 is at a different density level than C_4 and C_5.

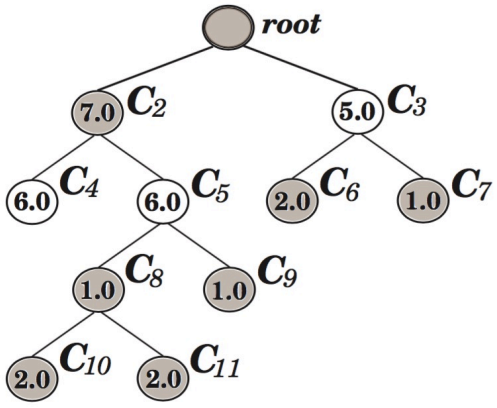


Figure 3.8: Taken from [10]: Illustration of the optimal selection of clusters from a given cluster tree.

Chapter 4

HDBSCAN* with kernel density estimates

HDBSCAN* is a hierarchical clustering algorithm based on density level sets [10]. It follows the approach of building a Minimum Spanning Tree (MST) in density space, and extracting the hierarchy from it, by removing edges in decreasing order of weights, which correspond to the Mutual Reachability Distance (MRD). The algorithm uses an unnormalized version of the k-nearest neighbor (kNN) density estimate. The original kNN density estimate [45] gives a density value for each object. HDBSCAN* uses the kNN distance, instead of the density value, to give a notion about the density of the object, i.e. for objects with lower kNN distance, the density would be high as neighboring objects are close by.

In Chapter 3, an overview of HDBSCAN* was presented. In this chapter, we will further explore HDBSCAN* to integrate other density estimates into it. Since HDBSCAN* already uses a version of the kNN kernel, it is a logical choice to incorporate the original kNN kernel density estimate into HDBSCAN*. Thus, we start by using the kNN kernel density estimate instead of its unnormalized form, kNN distance directly. We elaborate the problems we encountered, and then, finally, describe our new approach of building a MST in Euclidean Space. The edge weights of this MST are recalculated to be the inverse of the density estimate values. This is because, if we were to use density values for a spanning tree construction, we would obtain a Maximum Spanning Tree. By adopting the inverse of density value, we are still able to

use the concept of Minimum Spanning Tree in HDBSCAN*.

4.1 HDBSCAN* with the kNN density estimate

The kNN kernel proposed by Loftsgaarden [45] is given by Equation (2.5) in Chapter 2. For an object x, k nearest neighbors and r_k as the kNN distance between x and its k^{th} neighbor including x, we can derive the following relation from it

$$kNN(x) \propto \frac{1}{r_k}$$

This relationship is used in HDBSCAN* to represent the density of an object by the so called core distance of x, using the following unnormalized form of the kNN kernel:

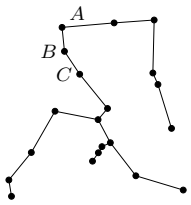
$$\text{unnormalized } kNN(x) = \frac{1}{r_k} = \frac{1}{d_{core}(x)} \tag{4.1}$$

As explained in Chapter 3, HDBSCAN* defines core distance and MRD as

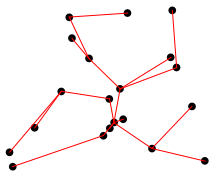
1. The **core distance** of an object is the k-nearest neighbor distance. Thus, the smaller the core distance, the denser the object.

2. **Mutual Reachability Distance** represents the connectivity between two objects. For any point on the line segment joining two objects, if we draw a circle of radius equal to the MRD of the two objects, the number of points encompassed within this hypersphere are at least k. By comparing the core distances and the distance between the objects, MRD is able to take into account density at the objects as well as how far apart they are.

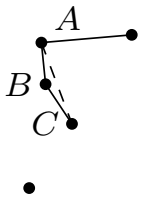
If the MST is build by solely considering the nearest neighbor distance or the second nearest neighbor distance, it would be the same as the Single Linkage tree [37]. This is because the MRD, in this case, is the distance between the objects and its nearest neighbor. For third nearest neighbors and onwards, the MST in Mutual Reachability Space is no longer the same as the single linkage one. The shortest path in Euclidean Space leads to a



(a) Single Linkage or MST in Euclidean Space



(b) Connections in MST for k=4



(c) The dashed line is the edge found in MST in Mutual Reachability space while the black edges are in the Euclidean Space MST.

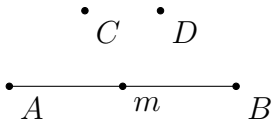
Figure 4.1: MSTs for 20 points in space, considering nearest neighbor 1 and 3, exclusive of point itself

more weighted path w.r.t MRD. As the number of objects to consider for calculation of core distance increases, objects farther and farther away from object of interest are considered for core distance calculation. Thus, core distance becomes larger than the Euclidean Distance.

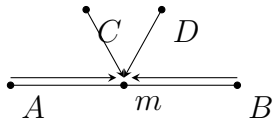
Consider 20 points distributed in two dimensional space. Figure 4.1a shows the Single Linkage MST. Using third nearest neighbor to calculate core distance, d_{core}, of each point and MRD, d_{mreach}, of each edge, the MST in Mutual Reachability Space does not have the connections (A,B) and (B,C). Instead, we can get (A,C) and (B,C) or (A,C) and (A,B). This is because

$$d_{mreach}(\mathbf{A},\mathbf{B}) = \max(d_{core}(\mathbf{A}), d_{core}(\mathbf{B}), d(\mathbf{A},\mathbf{B})) = d_{core}(\mathbf{B})$$
$$d_{mreach}(\mathbf{B},\mathbf{C}) = \max(d_{core}(\mathbf{B}), d_{core}(\mathbf{C}), d(\mathbf{B},\mathbf{C})) = d_{core}(\mathbf{B})$$
$$d_{mreach}(\mathbf{A},\mathbf{C}) = \max(d_{core}(\mathbf{A}), d_{core}(\mathbf{C}), d(\mathbf{A},\mathbf{C})) = d_{core}(\mathbf{A})$$

As $d_{core}(\mathbf{A}) < d_{core}(\mathbf{B})$, the longer edge in Euclidean Space has a lower MRD and would be selected in the density MST. Hence, the path directly connecting A and C is denser than going from A to C via B. Either of the edges (A,B) or (B,C) can be chosen.



(a) 4 objects in a data space. Let AB be the edge for which we are trying to estimate the density at midpoint, m.



(b) Arrows show the influence of objects A, B, C and D on the midpoint m.

Figure 4.2: Defining Connectivity as density estimate at midpoint

Now, we wish to use the kNN kernel instead of its unnormalized form, shown in Equation (4.1). This has the following challenges -

1. The value of density associated with each object is just the value of the kNN kernel for that object; it is no longer a distance. A low density value means that the region around the object is sparse, and a high density value implies, that there are numerous objects around our object of interest.

2. We have no notion of the density between two objects. The Mutual Reachability Distance cannot be used anymore, because distance by itself is not comparable to the inverse of density. They are on different scales. For example, if distance between two objects is 1.5 cm, and the density at the objects are 0.5 and 0.1 respectively, their inverse becomes 2 and 10. We cannot compare cm and inverse of density because there is no method of converting them into the same measurement units.

Hence, for using a kernel density estimate like the kNN kernel, we must define connectivity between two objects using something other than distance. Let us assume that if there are two objects A and B, the density at the midpoint of line segment joining A and B represents the density between the objects. While building the MST, we can use this value of inverse of density (to keep the spanning tree minimum) at midpoint. Figure 4.2 shows an illustration.

The naïve approach of using the kNN kernel would be to build the MST in density space – the edge weights are defined as the inverse of density at midpoint of the edge, and the self edge weights are the inverse of value of kNN kernel at the object itself. All other steps of HDBSCAN* would remain the

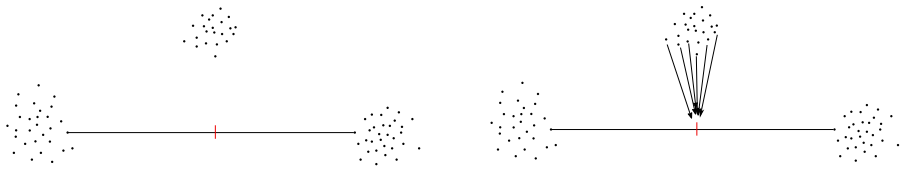


Figure 4.3: Density due to denser regions near midpoint : A third cluster influencing the density at midpoint (in red) on an edge connecting end points of two other clusters, resulting in an over estimation of the density connection between the two clusters.

same. A hierarchy would be extracted from the MST based on m_{clSize}, and an optional flat partition could be extracted from the hierarchy based on excess of mass and stability.

The problems with such as approach would be

1. Since the Minimum Spanning Tree is built in density space, the density at the midpoint of every edge in the complete graph has to be estimated. For an n point dataset, the complete graph has $n(n\text{-}1)/2$ edges, for each of which density has to be calculated, each time considering all n points to get the estimate.

2. Midpoint estimation is the simplest method for representing connectivity. However, it has one limitation when applied to a complete graph. For many edges, the density is very high at midpoint due to the presence of the clusters between the end points of the edge. Hence, if there is one or more clusters lying between the end points of the clusters we want to connect, as shown in Figure 4.3, then the density estimated at midpoint is very high since the points from the other clusters influence it.

If we tried following this methodology anyway to confirm its limitations, let us investigate the results for a $n = 500$ points in 2 dimensional dataset with 5 well separated clusters, as shown in Figure 4.4. To compare the results of HDBSCAN* with this naïve approach, called kNN-HDBSCAN* hereafter, the value for m_{pts} in HDBSCAN*, and k in kNN kernel are set to be the same. After building the hierarchy, a flat partition is obtained using stability, defined

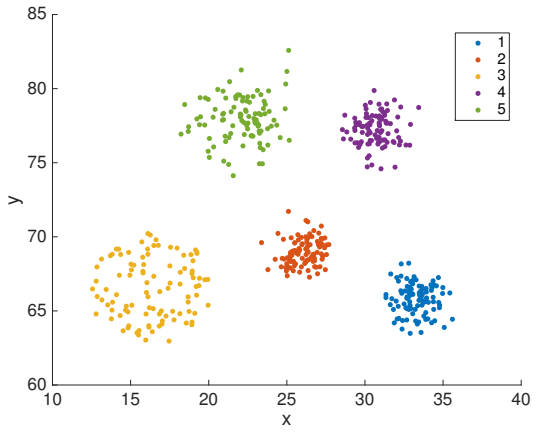


Figure 4.4: Sample dataset with 500 2D points in 5 clusters

by Campello et al. [11]. Adjusted Rand Index (ARI) [34] for the flat partitions from HDBSCAN* and kNN-HDBSCAN* for same k are compared against the available ground truth. The m_{clSize} is set to 10. We test over the range of $k \in [2, n/m_{clSize}]$, i.e. $k \in [2, 50]$.

The best ARI that can be obtained from HDBSCAN* for $k \in [2, 6]$ is 1. The best ARI from kNN-HDBSCAN* is 0.46 for $k = 31$. In terms of time taken, on average, HDBSCAN* could build the hierarchy and produce flat partition within 0.03 seconds, while kNN-HDBSCAN* took 1.58 seconds. It is evident that over the range of values tested, HDBSCAN* performs much better than using the kNN kernel, and estimating the density connectivity between two points A and B by the estimated density at the midpoint of the straight line connecting A and B. Further, computationally, it takes much more time because for each edge, the midpoint has to be found and its k nearest neighbor has to be located to calculate the density at the midpoint. Thus, for a complete graph, midpoints and their k nearest neighbors have to be computed for each of the $n(n-1)/2$ edges.

Figure 4.5 compares the ARI for each value of nearest neighbor. The ARI value from kNN kernel with midpoint density estimation is consistently less than the ARI value obtained from HDBSCAN* for the same value of k. To better understand the difference between the original HDBSCAN* and its version with the kNN kernel, for the same of $k = 4$, we compare the Minimum Spanning Trees obtained from HDBSCAN* and kNN-HDBSCAN*. These are

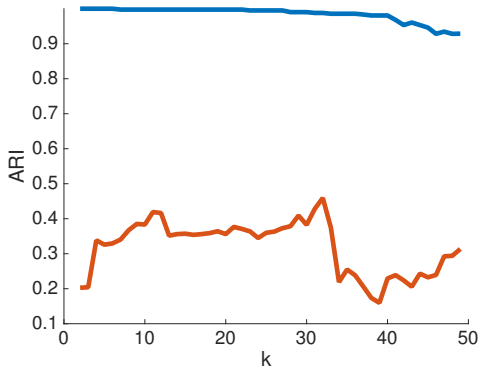
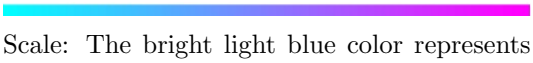
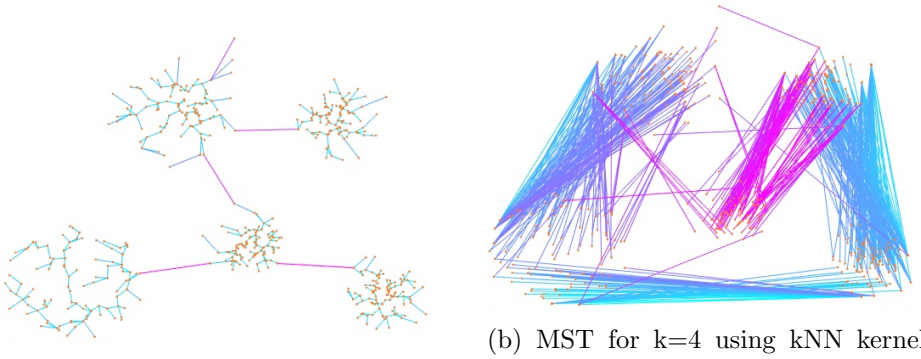


Figure 4.5: ARI values over the range of k nearest neighbors to consider: This figure shows the variation in ARI as we change the value of nearest neighbors to consider. The red line represents the result from kNN-HDBSCAN* and blue shows HDBSCAN*.



Scale: The bright light blue color represents dense edges whereas magenta shades indicate less dense edges.



(a) MST for k=4 using HDBSCAN*

(b) MST for k=4 using kNN kernel in HDBSCAN* and midpoint density estimation.

Figure 4.6: Minimum Spanning Trees for sample dataset (Figure 4.4) and $k = 4$.

shown in Figure 4.6a and 4.6b, respectively. As expected, for the kNN kernel MST, in Figure 4.6b, there are many edges connecting points of one cluster to another. Edges within the cluster are few. Also, these edges that connect points of different clusters are dense, the light blue edges are the ones with the highest values of edge weights. This can be explained as follows: when we estimate the density at the midpoint of an edge, it is affected by other points in space which may be closer to the midpoint and not close to the end points themselves.

Our comparison of the naïve approach with the original, theoretically, and

experimentally, confirms our doubts that building the Minimum Spanning Tree over the complete graph in density space using the kNN density estimate is time consuming and gives poor results. Thus, to generalize HDBSCAN* to use arbitrary kernels, our approach has to change.

4.2 HDBSCANk: the generalized approach

To avoid the problems discussed in the previous section, we propose the following solution: We build the Minimum Spanning Tree in Euclidean Space and then transform the edge weights to the inverse of density. Zahn [80] enlists the properties of MST for cluster analysis and Hartigan [29] studied the consistency of single linkage for high dimensional data. The advantages of using a MST in Euclidean Space are:

1. An MST in Euclidean Space is built by considering nearest neighbor distance. It follows the data itself, the placement of objects in terms of vicinity, as we connect close by objects. In two dimensions, the connections can be visualized using a plot.

2. It is easy to compute since it only requires a distance matrix.

3. Most importantly, the weights of the edges in the MST can be easily given values in density space. For a N object dataset, only (N-1) such calculations are required instead of the N(N-1)/2 edges that have to be processed in the complete graph.

4. The MST is general enough that it can be used for any kernel density estimate, i.e. once the spanning tree is built, any kernel density function can be applied to the edges. We do not have to recompute the tree if a different estimate is used. Such a property is highly useful when a comparative study with different kernels has to be conducted.

Thus, in our new approach, known as HDBSCAN kernel or HDBSCANk, we first compute the MST in Euclidean Space using Prim's [55]. Then, we recalculate the edge weights, such that they represent the inverse of the minimum

density along the edge. The connection between two objects stays in the tree; however, based on the value of density, the weight of the edge becomes higher or lower.

The parameter *method* would define the manner in which this representative of minimum density along the edge is calculated. The parameter *kernel* dictates the kernel density estimate to use. *bandwidth* represents the value of k for the kNN kernel and the smoothing parameter for other kernel density estimates, as elaborated upon in Chapter 2. Algorithm 4 shows the pseudo-code.

Algorithm 4 HDBSCANk Main Steps

Input: Dataset X, parameters $kernel, method, bandwidth$
Output: HDBSCANk hierarchy, (optional) Flat Partition

1. Compute the value of kernel denisty estimate for each object using given *kernel* and *bandwidth*.
2. Construct MST in Euclidean Space, $EMST$.
3. Update edge weights using given *bandwidth* for *kernel* and *method* and take the inverse of the estimate.
4. Extend the MST to obtain $EMST_{ext}$, by adding for each vertex a "self edge" with inverse of kernel density estimate at the corresponding object as weight.
5. Extract the HDBSCANk hierarchy as a dendrogram from MST_{ext}:
   - 4.1 For the root of the tree assign all objects the same label (single "cluster").
   - 4.2 Iteratively remove all edges from MST_{ext} in decreasing order of weights (in case of ties, edges must be removed simultaneously):
     - 4.2.1 Before each removal, set the dendrogram scale value of the current hierarchical level as he weight of the edge(s) to be removed.
     - 4.2.2 After each removal, assign labels to the connected component(s) that contain(s) the end vertex(-ices) of the removed edge(s), to obtain the next hierarchical level: assign a new cluster label to a component if it has at least one edge, else assign it a null label ("noise").

Optional Extract a flat partition from the HDBSCANk hierarchy using stability.

To evaluate the new approach to HDBSCAN* by computing an MST in

Euclidean Space first, we compare the following variants of HDBSCAN* using four two dimensional datasets summarized thereafter.

1. HDBSCAN*, the original algorithm,
2. MST-HDBSCAN*, HDBSCANk with unnormalized the kNN kernel and MRD, and
3. HDBSCANk with kNN kernel and the connectivity between objects still defined by the density at the midpoint of the edge connecting the points, but now restricted to edges in the Euclidean MST. Our *method* can, thus, be called 'Midpoint Estimation'.

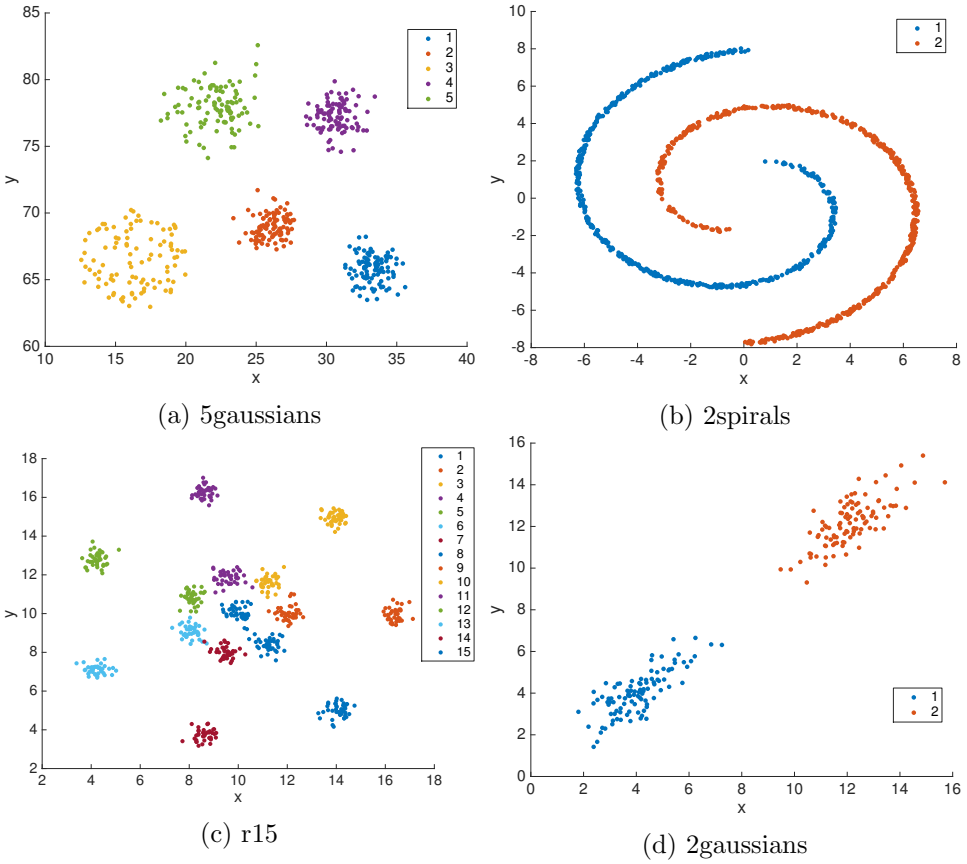
4.2.1 Experimental Results

Datasets

For these experiments, we report the performance of this algorithm on four artificially generated two dimensional datasets. Dataset 1, 5gaussians, has 500 two dimensional points, divided evenly into 5 Gaussian shaped clusters. Dataset 2, 2spirals, has 2 well separated spirals. It has 1000 points. Dataset 3, r15, has 15 Gaussian clusters, 7 of them are well separated and the other 8, form a flower like structure in the middle. Dataset 4, 2gaussians, is another artificially generated dataset with 200 2-dimensional points, well separated into 2 clusters. Datasets 2 and 3 were downloaded from the website, Joensuu Clustering Datasets [1] while the others were self-generated. Figure 4.7 shows the scatter plots and available ground truth for them.

Algorithmic Settings

After building the hierarchy, a flat partition is obtained using stability. The Adjusted Rand Index (ARI) for the flat partitions from HDBSCAN*, MST-HDBSCAN* and HDBSCANk with kNN kernel and Midpoint Estimation, for same k are compared against the available ground truth shown in Figure 4.7. The minimum cluster size (m_{clSize}) is set to 10. We test over the range of $k \in [2, n/m_{clSize}]$, where n is the number of objects in the dataset. A tabular summary of the datasets and the range of k tested is presented in Table 4.1.



(a) 5gaussians

(b) 2spirals

(c) r15

(d) 2gaussians

Figure 4.7: Two Dimensional Datasets

Dataset	No. of points	Dimension	No. of Clusters	k
5gaussians	500	2	5	[2,50]
2spirals	1000	2	2	[2,100]
r15	600	2	15	[2,60]
2gaussians	200	2	2	[2,20]

Table 4.1: Dataset summary and the range of k tested

Dataset	ARI		k	
	HDBSCAN*	MST-HDBSCAN*	HDBSCAN*	MST-HDBSCAN*
5gaussians	1	1	[2,6]	[2,6]
2spirals	1	1	[5,30]	[5,30]
r15	0.98	0.98	[2]	[2]
2gaussians	1	1	[2,20]	[2,20]

Table 4.2: Best Adjusted Rand Index and value for k for HDBSCAN* and MST-HDBSCAN*

Dataset	HDBSCAN*	MST-HDBSCAN*
5gaussians	0.03	0.03
2spirals	0.07	0.07
r15	0.03	0.03
2gaussians	0.01	0.01

Table 4.3: Computation times (in seconds) for the experiments in table 4.2 averaged over the range of k tested: This table shows the time taken in seconds for building the MST, extracting a hierarchy and a flat partition from it, averaged over the number of values for k, the number of nearest neighbors, considered.

Comparison between HDBSCAN* and MST-HDBSCAN*

Table 4.2 shows the best ARI and the corresponding value(s) of k obtained for each dataset using HDBSCAN* and MST-HDBSCAN*. Table 4.3 shows the average time taken to run the algorithms per value of nearest neighbors. We find that building the Minimum Spanning Tree in Euclidean space, or in the density space first does not affect HDBSCAN*. We still obtain the same value for ARI, for the same range of k, in the same average time. Thus, using core distance and MRD is a robust approach.

Comparison of all 3 versions of HDBSCAN*

Table 4.4 shows the maximum ARI obtained from HDBSCANk, the value of k for which it was obtained and the average time taken to do all computations per value of k. The accuracy is close to what HDBSCAN* gets, as shown in Table 4.2.

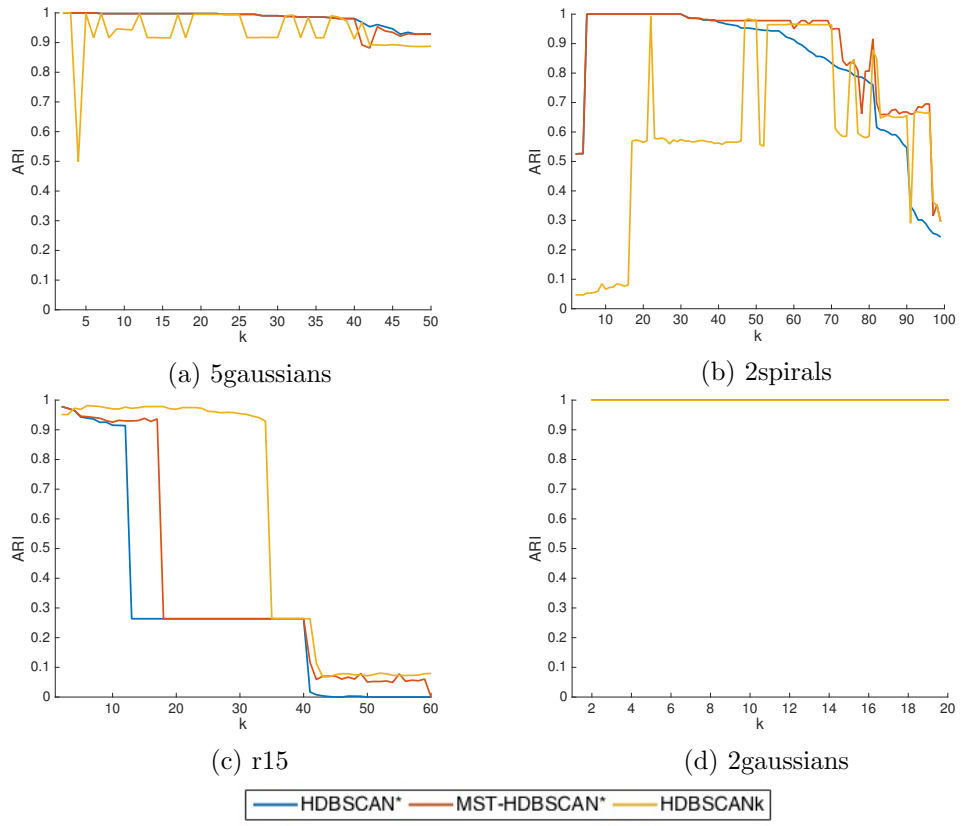
In Figure 4.8, we show the change in the ARI values with change in value of k for all 3 versions. Irrespective of dataset, HDBSCAN* and MST-

Dataset	ARI	k	Average Time (seconds)
5gaussians	1	[2,3]	0.03
2spirals	0.99	[22]	0.05
r15	0.98	[6,7]	0.03
2gaussians	1	[2,20]	0.01

Table 4.4: Results for HDBSCANk with kNN kernel and Midpoint Estimation : For each dataset, the best ARI that was achieved for the range of k tested, the corresponding value(s) of k for which this ARI was obtained and the average time taken per run of HDBSCANk is listed.

HDBSCAN* exhibit similar behavior. For a well separated dataset like 2gaussians, HDBSCANk with kNN kernel and Midpoint estimation, has the same variation of ARI with k. For r15, this combination of kernel and method is consistent in behavior with the other two versions, only the first drop in ARI happens much later for a higher value of k. 2spiral exhibits a special shape that HDBSCAN* can identify for a wide range of k. MST-HDBSCAN* can also get the same result for a smaller range of k. The kNN kernel can identify this special structure for two values of k. The value of ARI obtained from stability partition of the hierarchy for this dataset changes frequently with the change in value of k. On using HDBSCANk, on the 5gaussians dataset, we observe that the ARI value changes from one value of k to another but the ground truth structure of the dataset is still identified by HDBSCANk for some values of k. From this analysis we conclude that the kNN kernel with Midpoint Estimation is not a robust method.

By using our new approach of building the MST in Euclidean Space, and then recalculating the edge weights to be the inverse of density, we are able to come close to, if not replicate, the best results of HDBSCAN*, with both the unnormalized and original kNN kernel in HDBSCANk, as shown in Tables 4.2 and 4.4. The runtime in Tables 4.3, 4.4 is the same in most cases. This suggests that HDBSCANk is as fast as HDBSCAN*, at least for these datasets.



(a) 5gaussians

(b) 2spirals

(c) r15

(d) 2gaussians

Figure 4.8: ARI values over the range of k nearest neighbors to consider for HDBSCAN*, MST-HDBSCAN* and HDBSCANk.

4.3 Defining Connectivity between Objects: Edge Weight Estimation Methods

Hartigan defined a density-contour cluster as a subset $\mathbf{C} \subset R$ at density level λ, such that: (a) every object $x \in \mathbf{C}$ satisfies $f(x) \geq \lambda$, (b) C is connected, and (c) $\mathbf{C}$ is maximal, $f(x)$ being the density function defined for each x as a value proportional to the number of points per unit volume at x [28]. Hartigans model has two essential components: (a) density f defined at each object, and (b) density f defined along each path between the objects in the space. HDBSCAN* is based on Hartigans concept of density-based clustering, and accommodates these two components as (a) density f defined at each object, which is at least $1/\varepsilon$, and (b) density f defined on a path between the objects in the space, which essentially depends on the density between pairs of objects and which is equal to $1/d_{mreach}$.

As Hartigan noted in his paper on single linkage consistency [29], for determining density contours, one not only needs the objects which are connected but also the density on this connection. As we observed, there is no straightforward way of defining the connectivity when using general kernel density estimates. In this section, we suggest and discuss some possible connectivity methods.

1. **Midpoint Estimation** For the midpoint of each edge, in the MST in Euclidean Space, the kernel density estimate is calculated using all objects in the dataset.

2. **Midpoint Estimation using top contributors of density at vertices** In this approach, the density at the midpoint of each edge is calculated using a subset of the dataset. Consider an edge between A and B.

    - For a continuous kernel like the Normal kernel, the objects whose contribution make up approximately 85% of the density at A and B are identified. This percentage can vary with kernel. Then, these objects are used to calculate the density at the midpoint. For the

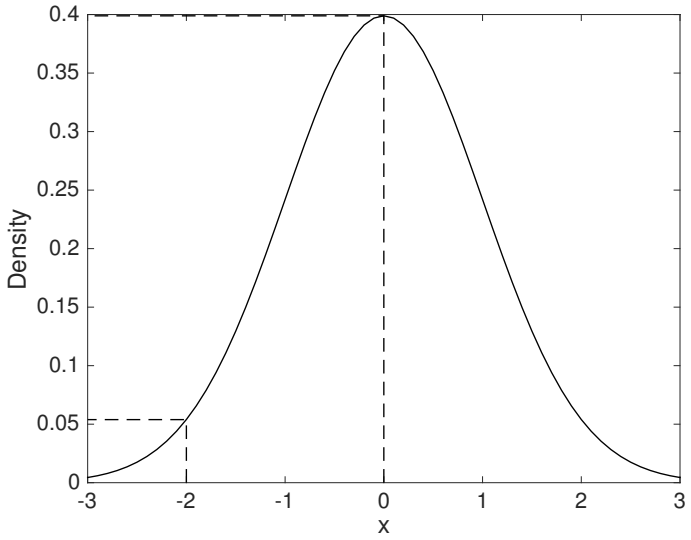


Figure 4.9: Contribution to density of 2-sigma region of a Normal distribution: The points within the one and two sigma regions of a Normal distribution with mean 0 and standard deviation 1, contribute to 0.0539*100/0.3989 = 13.5% density of at the mean. Thus, all points in the 2-sigma region contribute to most of the density of the object.

Normal kernel, if we observe the bell curve in one dimensional space, the two sigma region contributes almost 85% (86.5% to be exact) of density of the curve, as shown in Figure 4.9. Thus, instead of using all objects in space, only the most concentrated ones are considered.

- Kernels like the kNN and Epanechnikov kernels, have a property that objects beyond a certain distance (eg the kNN distance for the kNN kernel) do not have any influence on the density of an object of interest. Since the kernel is already truncated, all objects that have a non-zero contribution to density on A and B form the top contributors, and are used to evaluate the density at the midpoint.

The contribution of a distant object to density of a point is very low. By identifying the most influential points, we are able to get an approximate density estimate at midpoint which would not change much if all objects in space were considered, as in the case of Midpoint Estimation. Figure 4.10 shows the top contributors for an object in a data space.

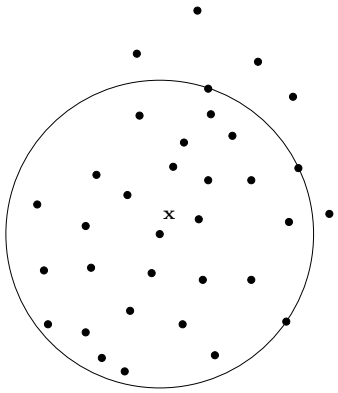


Figure 4.10: Top Contributors for an object, **x** : all objects within the circle contribute to 85% of the density of **x** for a continuous kernel. Alternatively, for truncated kernels, objects outside this circle have zero influence on density and need not be consider.

3. **Torque Rule Estimation** Given an edge connecting A and B, the length of line segment AB, and the density values at A and at B, if we assume a linear interpolation of density, there would be a point on the edge, such that the influence from A is same as the influence of B. If we think of density as a force, the forces exerted from A and B would be in balance at that point on AB. This point is known as the *torque point*, based on the equilibrium problems in Physics. The edge weight of the connection AB is then defined as the density on the torque point. As an illustration in one dimension (see Figure 4.11), consider two objects A and B that have associated density a and b respectively. They are d distance apart. To find the position d_1 from A, on the line joining AB where the density influence due to A is equal to that from B, calculate

$$ad_1 = b(d - d_1)$$
$$d_1 = \frac{bd}{a+b}$$

   Thus, the density estimated at point A+d_1 is the required density for the edge at the Torque point.

4. **Torque Rule Estimation using top contributors of density at vertices** Similar to Midpoint Estimation using the top contributors, in

this method, connectivity between two objects is defined as the density at the torque point due to the top contributors.

5. **Golden Search using top contributors of density** Golden Search, also known as the Golden Section Search [40], is used to find the maximum or minimum of a unimodal function. A unimodal function contains only one minimum or maximum on the interval $[a, b]$. For an edge AB, the coordinates of A and B form the interval, and the function we minimize is the value of the kernel density estimate due to the top contributors of A and B. We assume that, there would be only one point of minimum density for each edge.

4.4 Complexity of HDBSCANk

The time complexity of HDBSCANk depends heavily on the kernel and the edge weight estimation method that defines the connectivity between the objects. We first, summarize the complexity of steps that are invariant with respect to these two parameters, and then explain in detail, the complexity for each connectivity representative method and kernel, where necessary.

The main steps of HDBSCANk have been listed in Algorithm 4. For an d dimensional dataset with n objects:

- The kernel density estimate for an object is calculated in $\mathcal{O}(dn)$ time because of d dimensions and n objects considered that contribute to the density. For the whole dataset, this is calculated in $\mathcal{O}(dn^2)$ time.

- In Step 2, the Minimum Spanning Tree in Euclidean Space would be constructed from a complete graph. It would take $\mathcal{O}(dn^2)$ time to compute all nearest neighbor distances, and then $\mathcal{O}(n^2)$ to build the MST. If

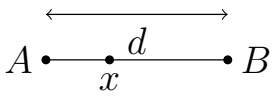


Figure 4.11: Locating the Torque Point: Consider two points A and B, d distance apart. We can find the point x on AB where influence of total density from A is balanced w.r.t influence from B.

Prim's algorithm [55] is used based on an ordinary list search, we could construct the MST in $\mathcal{O}(n^2 + m)$ time where $m = n(n-1)/2$ edges in the complete graph, so Step 2 runs in $\mathcal{O}(dn^2)$ time.

- In Step 3, we update the edge weights.
  - Using Midpoint Estimation or Torque Rule Estimation, for each edge in the MST, constant time is spent on calculating the coordinates of midpoint or torque point and $\mathcal{O}(dn)$ time to compute the K.D.E. value. Since this process is repeated $n-1$ times, once for each edge, it would take $\mathcal{O}(dn^2)$ time to perform updates to the edge weights by these two methods.

For methods that use top contributors of the vertices, these must be found first. In case of kernels like the Gaussian kernel, these can be computed while the density is calculated for each vertex in Step 1, by sorting the influence of each object on the vertex in descending order, and choosing the ones whose contribution sums upto 86.5% of the total density at the vertex object, i.e. those within two sigma region of the kernel. Computing this for every vertex takes $\mathcal{O}(n^2 \log n)$ time for sorting all influences, and $\mathcal{O}(n)$ comparisons to locate the contributors in the sorted list. In case of kernels like Epanechnikov and kNN kernel, no sorting is required and we can find all contributors by a linear search in $\mathcal{O}(n^2)$ time for all edges. Thus, depending on the kernel, top contributors can be found in $\mathcal{O}(n^2 \log n + n^2) = \mathcal{O}(n^2 \log n)$ for continuous kernels or $\mathcal{O}(n^2)$ time for truncated kernels.

- Midpoint Estimation and Torque Rule estimation using top contributors: Similar to the corresponding methods that use the whole dataset, for each edge, constant time is spent to calculate the midpoint or torque point, and then $\mathcal{O}(dn)$ time to compute K.D.E. at that point in the worst case, where subset of data is the dataset itself. Thus, these connectivity methods can take $\mathcal{O}(dn^2)$ to compute density of each edge, and $\mathcal{O}(n^2)$ or $\mathcal{O}(n^2 \log n)$ to find the top

contributors.

  - The time complexity of using Golden Search is $\mathcal{O}(\log \frac{1}{\epsilon})$ iterations of the function being minimized or maximized, converging to a ϵ-accurate solution. Since ϵ is a constant, in the worst case when all objects in the dataset are the top contributors, it would take $\mathcal{O}(dn^2)$ time to compute the density of all edges and $\mathcal{O}(n^2)$ using golden search and $\mathcal{O}(n^2 \log n)$ to find the top contributors.

- In Step 4, the MST is extended by adding self edges and this can be done in $\mathcal{O}(n)$ time. Before extracting the hierarchy, the $2n-1$ edges in the MST have to be sorted and this can be accomplished in $\mathcal{O}(n \log n)$ time.

- Extracting the hierarchy from the MST is the same as how the hierarchy is extracted in HDBSCAN*, in Step 5, and would take at most $\mathcal{O}(n^2)$ time, if every object has to be relabeled after an edge removal.

If we integrated a kernel like All-Points-Core-Distance kernel [50], the inverse of density of each object, given by $all_{pts}coredist$, is computed in $\mathcal{O}(dn^2)$ time in worst case. The formula is shown in Equation (2.7). The time for building the MST in Euclidean space remains the same. Updating the edge weights of the MST using the density of an edge, given by $d_{nmreach}$ (Equation (2.8)), takes $\mathcal{O}(dn^2)$ time. The time for rest of the steps also remains unchanged.

We conclude the total time complexity of HDBSCANk is $\mathcal{O}(dn^2 + n^2 \log n)$ when edge weight estimation methods with top contributors are used. Otherwise, it is $\mathcal{O}(dn^2)$ with arbitrary kernels, which is the same as the time complexity of HDBSCAN*. Table 4.5 summarizes the total time complexity of HDBSCANk.

In terms of space complexity, $\mathcal{O}(dn)$ space is required to store the dataset. In Steps 1 and 2, to compute the MST from a complete graph, requires $\mathcal{O}(n^2)$ space to store the distance matrix. The MST can be stored as a list of $n-1$ edges and n vertices. This would take $\mathcal{O}(n)$ space. Performing the updates in

Operation	Time Complexity
Calculate K.D.E. for all objects	$\mathcal{O}(dn^2)$
Find top contributors to density for each object	$\mathcal{O}(n^2 \log n)$
Construct MST in Euclidean Space from complete graph	$\mathcal{O}(dn^2)$
Update edge weights	$\mathcal{O}(dn^2)$
Sort edges in MST	$\mathcal{O}(n \log n)$
Build hierarchy from MST	$\mathcal{O}(n^2)$

Table 4.5: Time Complexity of HDBSCANk for a dataset $\mathbf{X}$ with d dimensions and n objects.

Step 3 does not increase the space. Extension of the MST adds n new edges to it, however, this too, does not affect the $\mathcal{O}(n)$ space to store the MST. For edge weight estimation methods that use top contributors, there can be at most n top contributors for each object. Thus, $\mathcal{O}(n^2)$ space is required to save the top contributors list for the dataset. While computing the hierarchy, in Step 5, at a particular level, only $\mathcal{O}(n)$ space is used. Hence, the total space complexity of HDBSCANk is $\mathcal{O}(n^2)$, the same as HDBSCAN*.

4.5 Conclusion

In this chapter we explored an extension of HDBSCAN* to use kernel density estimates. HDBSCAN* uses the unnormalized form of the kNN kernel and in our experiments, we replaced it with the actual kNN kernel. Testing this on a sample dataset, we concluded that the original HDBSCAN* approach had to be modified to incorporate arbitrary kernel density estimates. Also, using a kernel density estimate requires the exploration of methods that define connectivity between objects. Midpoint estimation is one such method where density connectivity between two objects is estimated as the density at the midpoint of the straight line joining these two objects. We proposed HDBSCANk, the version of HDBSCAN*, that can be used for arbitrary kernels and compared its results for some datasets with HDBSCAN* for both unnormalized and original kNN kernels. We also elaborated on new edge weight estimation methods that could be used to define the connectivity between objects.

Our experiments so far, are not enough to judge how useful kernel density

estimates are for hierarchical density based clustering because a) we only compared results of flat partitions that could be obtained from the hierarchy, and b) we do not have any dataset with a hierarchical ground truth. Thus, in the next chapter, we propose a hierarchical data generator that provides us with datasets which have a hierarchical ground truth. This will help evaluate our hierarchical algorithms more comprehensively, using a ground truth hierarchical structure. In chapter 6, using these datasets and others, we present a case study of different kernel density estimates and definitions of connectivity in a data space integrated into HDBSCANk, also studying the issue of bandwidth selection for kernels.

Chapter 5

Data Generator with Hierarchical Ground Truth

In real world applications of unsupervised techniques of machine learning, the availability of the true number of groups in the data, and information about which object belongs to which group is rare. This already makes cluster analysis results hard to evaluate. In addition, for the datasets that have ground truth, there is an absence of a hierarchical ground truth since a hierarchical ground truth involves multiple group assignments for data. To get a better idea of how good a hierarchical algorithm is, the current approach involves extraction of a flat partition from each Hierarchical Clustering Analysis (HCA) hierarchy and to compare these partitions using validation indexes like the Adjusted Rand Index [34], F-Measure [74], Silhouette Index [60] depending on availability of a flat partition ground truth labeling.

For document data, there are document collections that have been assigned categories that form a ground truth hierarchy. However, these must be generated using human intellect. Datasets like *Reuters RCV1* [42] and *oshumed* [31] are text document datasets whose hierarchical structure has been documented. But this is also rare. There is a need for a hierarchical data generator where there is minimum human intervention, and the data is simple enough that it can be used to test any hierarchical clustering algorithm, not just the ones specially proposed for document clustering.

In this chapter, we propose a hierarchical data generator that randomly samples data, based on the Gaussian sampling techniques and outputs the

data generated, as well a hierarchical representation for the same. The hierarchy corresponding to the dataset can be accepted as the correct solution, i.e. showing the underlying hierarchical properties of the data. Density-based clustering algorithms often model noise as a cluster and we follow the same representation of noise in the generator.

Our aim is to generate data that has a hierarchical ground truth. Thus, the points have to be generated such that they are divided into smaller clusters at subsequent levels. This would allow HCA algorithms that model noise as a cluster to easily test their validity against the underlying structure of the data.

There are two techniques in HCA:

1. Agglomerative or 'bottom up' in which each point is a singleton cluster at first. Each pair of clusters is merged as we move up the hierarchy based on some similarity criteria.

2. Divisive or 'top down' in which all points are initially part of one big cluster. At subsequent levels, splits are performed, dividing the data into smaller groups.

Our data generator uses the idea of divisive hierarchical clustering. We start without any data except the coordinates of the root cluster. A random number of points are sampled from around this root cluster. These are the cluster centers or *pseudo-centers* for that level. They are not part of the final dataset, only helper points. For every *pseudo-center*, the number of points it will have is assigned. A cluster may split into two or more *child clusters*. Thus, the number of child clusters a pseudo-center will have at the next level are calculated. By this calculation, the number of centers for the next level are available. Each of these are sampled from a Gaussian distribution with mean as their parent and a standard deviation smaller than any of the deviations at the previous level. For these *pseudo-centers* as well, the number of points around each center is determined. This iterative process continues till the number of points around a cluster drops below the threshold for that level, or no cluster has child clusters at the next level. Once all pseudo-centers have been found, we generate the

data as well a compact hierarchy for it, showing only the major levels at which clusters are split or die.

As mentioned in Chapter 3, the minimum cluster size is a user defined parameter in many practical applications of clustering that prevents the algorithms from finding very small clusters of objects. We incorporate the same parameter into our data generator to make it more consistent with hierarchical clustering approaches.

In the next section, we briefly describe the parameters of the generator. In section 5.3 we elaborate on the algorithm of the data generator. Section 5.4 shows some sample hierarchies and section 5.5 presents the ideas for future work.

5.1 Parameters

The user defined parameters for the data generator can be broadly divided into two categories:

1. Data descriptive: These parameters describe the properties of the dataset. These are:
   (a) the number of points it should have,
   (b) the dimensionality of the data
2. Hierarchy descriptive: These values define the hierarchical properties of the data. These are:
   (a) Branching factor: The maximum number of subclusters a parent cluster can have.
   (b) Minimum Cluster Size: When a cluster has minimum cluster size number of points, it cannot be split into child clusters.
   (c) Separation factor: This controls the degree of overlap between two adjacent clusters. The higher the separation factor, the smaller the probability of two clusters touching each other, or being in each other's region.

(d) Split probability decrease constant: A cluster can either die or split into child clusters. This split probability decrease constant or level constant value controls the decrease in probability of splitting and dying, as the level increases.

(e) Retention factor: Whenever a cluster splits into child clusters, a fraction of its total points are kept at the current level, and the rest are redistributed among the children. The retention factor makes sure that child clusters are truly embedded in a parent cluster (as opposed to only being joined at the level of the parent cluster).

We will elaborate on these parameters and their effect on the data generator in detail in the next section.

5.2 Data Generation with Hierarchical Ground Truth: The Approach

Given the number of points $numPoints$, and their dimensionality, d, we want to obtain two structures:

1. A $(numPoints \times d)$ matrix representing the dataset of points with their features.

2. The hierarchy for this dataset representing the different clusters a point can belong to depending on the level of the hierarchy.

Consider a hypothetical dataset with 1000 2-dimensional point. If the minimum cluster size parameter is set to 50, one of the possible hierarchies are shown in Figure 5.1. The root level has all 1000 points. At the next level, five clusters are obtained. Of these, the biggest clusters split into subclusters at further levels while those with 50 points do not split. Also, clusters that do not have more than $(2 \times 50 = 100)$ points, do not have child clusters at the next level. At every level, the clusters become more compact than the last level. Our aim is to generate data that exhibits these properties of the hierarchy.

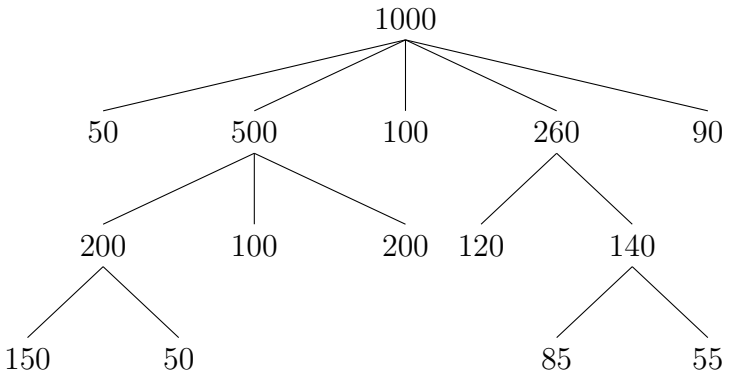


Figure 5.1: Sample hierarchy for 1000 points dataset : In this sample hierarchy, the 1000 points are divided into 5 clusters, two of which have further subclusters.

To generate data that has a hierarchical cluster structure, we need to be able to specify constraints on the hierarchical structure that data should have. Generating the dataset only requires the cluster centers, the standard deviation associated with that cluster and the number of points in the cluster. For building the hierarchical structure of the dataset, we need to know how many levels are present, how many clusters exist at each level, the standard deviation of each cluster, which clusters split into sub-clusters, and how many points are in each cluster.

To generate a dataset with hierarchical cluster structure, we start with the root of the hierarchy, which all data belongs to. In our generator, points for a cluster are sampled from Gaussian distributions, which are represented by a mean, which we call the *pseudo-center* of the cluster, and the 3-sigma region around the pseudo-center.

Pseudo-centers are points that act as cluster centers. They do not exist in the dataset as final objects but are used to generate the data itself. Points are sampled from within the 3-sigma region of the pseudo-centers. Consider a Gaussian distribution whose mean is the pseudo-center. The 3-sigma region of the pseudo-center is defined as the area within three standard deviations of the mean.

Statistically, if we sample a point from a Gaussian distribution, there is 68.27% chance it is sampled from within one standard deviation of the mean, 95.45% chance the sample lies within two standard deviations and a 99.73%

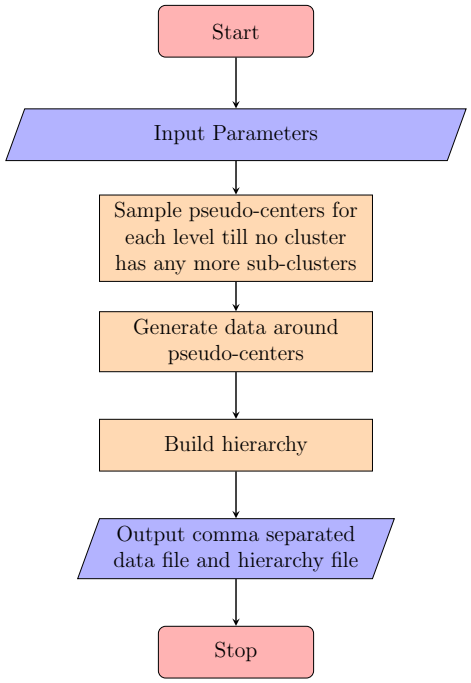


Figure 5.2: Flowchart depicting the major steps of the Data Generator

probability that the sample is within three standard deviation of the mean. The Gaussian distribution is a continuous distribution and by using the three sigma region, we limit our point generation to within this region. This also allows us to check if a point belongs to another pseudo-center's region.

Once we know all pseudo-centers, we generate data around them and build the final dataset and the corresponding hierarchy. These steps have been summarized in Figure 5.2.

5.2.1 Finding pseudo-centers

Before we elaborate on the procedure for finding the pseudo-centers and their properties, let us introduce some basic definitions related to the parameters of the data generator.

Definition 13 (Branching Factor) *The branching factor, denoted by $branchingFactor$, is the maximum number of child clusters a cluster can split into.*

Definition 14 (Minimum cluster size) *The minimum cluster size, denoted by $minClSize$, is the size of the smallest cluster that can exist without splitting.*

Definition 15 (Retention factor) *The retention factor, denoted by $retentionFactor$, is the fraction of the number of points in the cluster that are generated at the current level when the cluster splits, while the remaining points of the cluster will be generated from the description of the child clusters at lower levels.*

Our root node is defined by the origin in d-dimensional space. For example, for a two dimensional dataset, the root node has the pseudo-center coordinates (0,0). The standard deviation in each dimension is set to 1.

We determine the number of children the root node will have. This is decided by randomly selecting the number of splits that should happen at the root node. The number of splits, in general, is a number between 0 and $branchingFactor$, where 0 means that the cluster dies at current level, 1 implies it goes to the next level and a number greater than 1 indicates the number of children of the cluster. For the root node, we must ensure that there are at least 2 children. The number of splits depends on the following factors:

1. Total number of points in the dataset.

2. Number of points in the cluster: The more points in a cluster, the higher its probability to be split.

3. The level in the hierarchy, the root node being level 0. The higher the level, the lower the probability of a split. Thus, more splits happen near the root.

4. Minimum cluster size: When a cluster is split, each child cluster should have at least $minClSize$ number of points. For example, if $minClSize$ is 10 and we have 20 points in a cluster, the child clusters can be 10 and 10. However, if there were only 15 points, the cluster cannot be split into two. This acts like a stopping criterion.

5. Probability of split: This represents the chance that a cluster has to be split based on the number of points it has and the level.

6. Probability of staying: This is the property of a level. Every pseudo-center has a standard deviation associated with it. This standard deviation shrinks at every level so that levels near the root have more spread out clusters. With deep hierarchies, shrinking at many levels would lead to clusters concentrated at small regions. Thus, as we go to lower levels of the hierarchy, a cluster is more likely to die, avoiding generation of very deep hierarchies.

7. Retention factor: The $retentionFactor$ is a value between 0 and 1. When a cluster is split into child clusters, ($retentionFactor$ times number of points in the cluster) many points are generated at the current level and the remaining points are redistributed amongst the child clusters.

Let $k \in [2, branchingFactor]$ denote the number of children of the root node. Function **getLifeSentence** in Algorithm 5 is used to select this value using probability of split and stay. Note that there are multiple ways of defining the probability of split and the probability of stay. We present a generic way of using them. In a subsequent subsection, we will explain how we compute them in more detail.

Once the total number of splits have been decided, we randomly assign the number of points to be generated by each child pseudo-center, while ensuring that each child cluster has at least $minClSize$ many points. The function **getBranches**, Algorithm 6, is used to perform this distribution. This function takes the number of points that are to be redistributed amongst child clusters and the number of child clusters as inputs. For every child cluster except the last child, a random number is generated between $minClSize$ and the maximum number of points that can assigned without violating $minClSize$. This number represents the number of points that would be in the child cluster. For example, suppose 80 points have to distributed amongst 4 clusters satisfying $minClSize = 15$. To ensure that there are at least 15 points in all other clusters, the first cluster ca have points between 15 and (80 - 15 * (4-0-1)) = 45. Then, there would be at least 45 points still left to divide amongst the 3 remaining clusters. The number of points to redistribute is updated. For

Algorithm 5 Procedure to find size of each split such that each split has at least $minClSize$ number of points

Require: $retentionFactor$, $minClSize$, $branchingFactor$

1: **function** GETLIFESENTENCE($points$, $level$)
2: $numberOfSplits \leftarrow 0$
3: $pointsToRedistribute = points - points * retentionFactor$
4: $probStay \leftarrow$probability of stay for $level$
5: **if** $pointsToRedistribute < minClSize * 2$ **then**
6: $prob \leftarrow$ some random number between 0 and 1
7: **if** $prob < probStay$ **then** ▷ Not enough points to split.
8: $numberOfSplits \leftarrow 1$ ▷ The cluster shrinks
9: **else**
10: $numberOfSplits \leftarrow 0$ ▷ The cluster dies
11: **end if**
12: **else**
13: $probSplit \leftarrow$probability of split at $level$ for $points$
14: $prob \leftarrow$ some random number between 0 and 1
15: **if** $prob < probSplit$ **then**
16: $min \leftarrow 2$
17: $max \leftarrow 2$
18: $var \leftarrow 3$ ▷ Maximum number of equal splits that are possible without violating $minClSize$
19: **for** $var <= branchingFactor$ **do**
20: **if** $pointsToRedistribute/var > minClSize$ **then**
21: $max = var$
22: **end if**
23: $var \leftarrow var + 1$
24: **end for**
25: $numberOfSplits \leftarrow$generate number between min and max.
26: **else**
27: **if** $prob < probStay$ **then**
28: $numberOfSplits \leftarrow 1$
29: **else**
30: $numberOfSplits \leftarrow 0$
31: **end if**
32: **end if**
33: **end if**
34: **return** $numberOfSplits$
35: **end function**

the last child cluster, the number of points still unassigned, is the number of points it will have.

Algorithm 6 Procedure to find size of each split such that each split has at least *minClSize* number of points

```
1: function getBranches(minClSize, points, splits)
2:     unassignedPoints ← points
3:     i ← 0
4:     nx ← array to store size of each split
5:     for i < splits do
6:         if i == splits − 1 then
                    ▷ finding size of last split so just assign remaining points
7:             nx[i] ← unassignedPoints
8:         else
                    ▷ Distribute points such that minClSize is never violated
9:             max ← unassignedPoints − (splits − i − 1) ∗ minClSize
10:            nx[i] ← generate number between minClSize and max
11:            unassigned ← unassignedPoints − nx[i]
12:        end if
13:        i ← i + 1
14:    end for
15:    return nx
16: end function
```

Definition 16 (Threshold Distance) *The threshold distance is the minimum distance between the clusters sampled from two different clusters at a particular level.*

Figure 5.3 illustrates how the threshold distance is used.

Since pseudo-centers sampled from the root node have the same parent, the threshold distance is not used. For simplicity, for the next level it is set to 1.0, and at every subsequent level, it is reduced by a predefined factor. If the factor is very low, there is a prominent distance between child pseudo-centers and the higher level clusters are very tightly packed. From our experiments with different values of this factor, we have found 0.8 to be a good choice.

Once the k pseudo-centers have been determined, the standard deviation of each center must be decided.

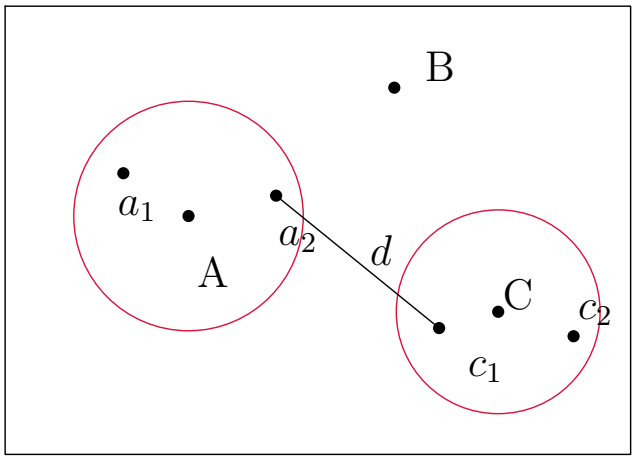


Figure 5.3: Role of Threshold Distance : Consider cluster centers A,B and C sampled from the origin. When cluster A splits into two child clusters a_1 and a_2 and C splits into child clusters c_1 and c_2, there is a minimum distance that must be maintained between the child clusters of A and C. This is the threshold distance, d.

Definition 17 (Separation factor) *The separation factor, denoted by $separationFactor$, is the extent of overlap between the three sigma regions of two adjacent clusters.*

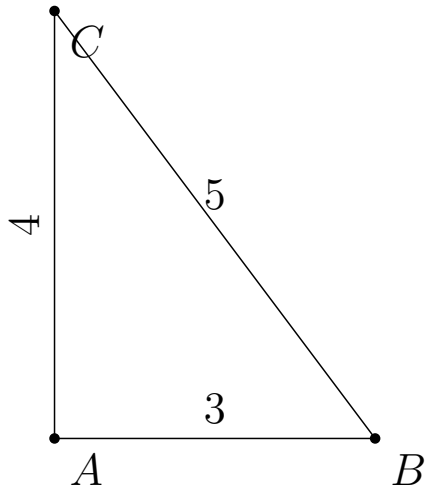
For each pseudo-cluster center, c, we find its nearest neighbor in the set C of all pseudo-centers obtained so far. We want the clusters around the pseudo-clusters to be distinguishable, hence, it is important to have a fixed method for deciding the standard deviation, instead of it being another randomly generated number. Using the separation factor and distance between c and the pseudo-centers closest to it, denoted by $nearestClusterDistance$, we define the standard deviation of the cluster as:

$$std_c = \frac{nearestClusterDistance_C}{separationFactor} \tag{5.1}$$

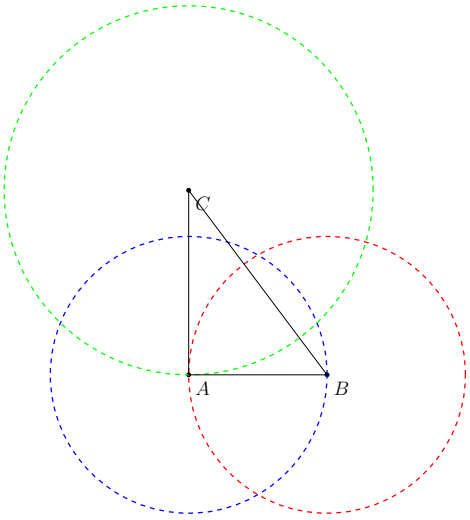
Figure 5.4 shows the effect of $separationFactor$ for three cluster centers in space.

Once the number of splits at first level, their respective pseudo-centers and standard deviations are defined, we keep track of the following for each level:

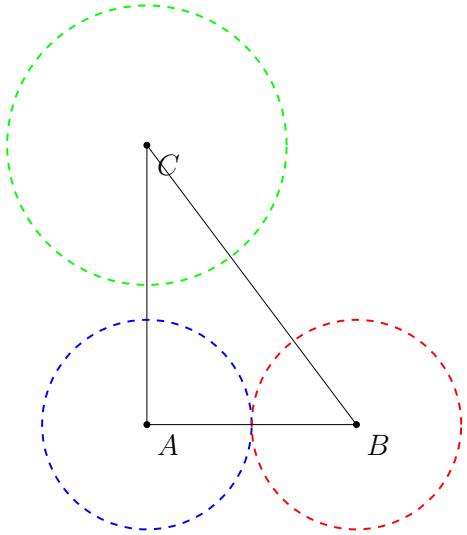
1. Number of points in each child cluster.



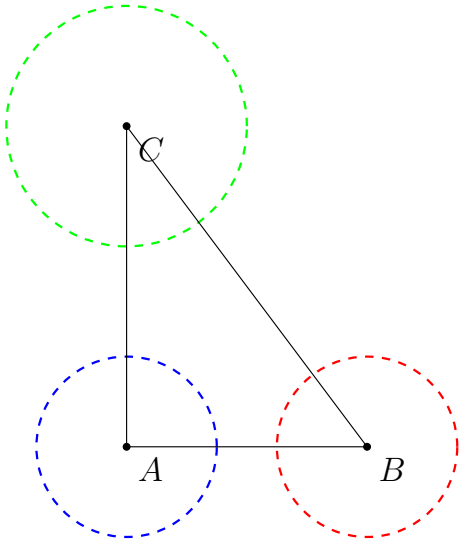
(a) Three points in space and the distance between them.



(b) *separationFactor*=3



(c) *separationFactor*=6



(d) *separationFactor*=8

Figure 5.4: Effect of *separationFactor* between three cluster centers in space : Consider three cluster centers, A, B and C, at some level of the hierarchy, such that B is the nearest neighbor of both A and C. If we were to draw a circle around each center with radius as three standard deviations around that center, we find that a *separationFactor* of 6 leads to touching three sigma regions. Below 6, the regions overlap and above 6, the separation starts to become more pronounced.

2. Number of splits for a parent cluster.

3. Clusters that cease to exist at the next level.

4. Clusters that died at earlier levels.

For all subsequent levels, we must decide the number of splits for each cluster center and, for clusters that die, move them to a list of final cluster centers.

At any level, three things can happen to a cluster:

1. It can split into child clusters.

2. It can stay the same, i.e. move another level down the hierarchy, and become more dense.

3. It can die, i.e. terminate at the current level, in which case its points are considered noise at the levels below the current level.

Definition 18 (Division Factor) *If a cluster shrinks, its standard deviation is reduced by the division factor, denoted by $divisionFactor$. The smaller the standard deviation, the more closely packed the points in the cluster will be. This ensures that as we move away from the root and down the tree, clusters have smaller standard deviations. The value of $divisionFactor$ is between (0,1).*

There are k clusters at level 1. For every cluster, using **getLifeSentence** function in Algorithm 5, we decide what happens to it at the next level. If the cluster ceases to exist after the current level, it is added to the final cluster center list along with its standard deviation. The standard deviation should be less than the lowest deviation at the previous level to ensure that lower level clusters are more dense. For clusters that shrink, i.e, the center and number of points remain the same, however, as it proceeds to the next level, we decrease its standard deviation by a predefined $divisionFactor$. If a cluster splits, the number of splits is calculated, $retentionFactor$ number of points are assigned to the cluster at the current level and the rest of the points are redistributed

amongst the children at the next level using Algorithm 6, **getBranches** function. A summary of these steps can be found in Algorithm 7, **getLevelSTDs**. This is a generic procedure that finds the standard deviation of the root as well. For the root, only lines 14-16 would be executed since the number of splits is guaranteed to be more than 1.

Now new pseudo-centers for child clusters must be sampled from the parent clusters that are split. Three conditions are imposed on this sampling:

1. The child cluster pseudo-centers should be sampled from within the 3-sigma region of the parent pseudo-center and it should not exist in the 3-sigma region of any clusters at the same level as the parent. Figure 5.5a shows an example of two cluster centers and the sampling region around them. If we sample a pseudo-center in the region common to the 3-sigma regions of A and B, shown by point C in Figure 5.5b , at the data generation step, the cluster could belong to either A or B. Thus, there would be ambiguity as to which parent cluster it belongs to. However, if child pseudo-centers for B is sampled from the non-overlapping region, there would be no question of which parent pseudo-center the points belong to at a higher level of the hierarchy, as shown in Figure 5.5c .

2. The child cluster is at least threshold distance for that level apart from pseudo-centers that are not its siblings. These may be pseudo-centers of clusters that have died, are shrinking, or sampled from other parents.

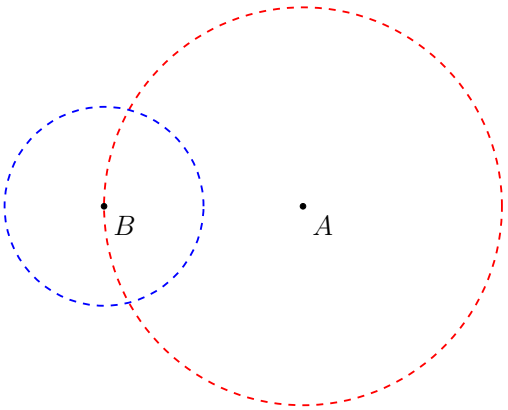
3. The child cluster pseudo-center is closer to its siblings than to other cluster's pseudo-centers. To ensure this, the maximum distance between a pseudo-center and its siblings is calculated. Also, the minimum distance between the pseudo-center and other pseudo-centers that are not its siblings is determined. If the maximum sibling distance is less than the minimum distance of the pseudo-center and non-sibling pseudo-centers, then the child cluster pseudo-center is closer to its siblings than other pseudo-cluster centers.

Once all next level cluster centers have been found, we save these values as

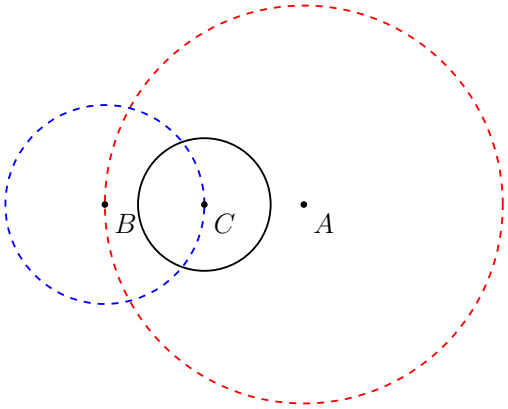
Algorithm 7 Defining Standard Deviation of Clusters

Require: *splitsAtLastLevel* : Splits at last level which gave birth to the current cluster centers.
separationFactor
Current cluster centers
Deviation of previous level clusters
longDead : an array that maintains which of the current cluster centers have died at earlier levels, 1 if cluster just died and 0 if it died earlier.
divisionFactor

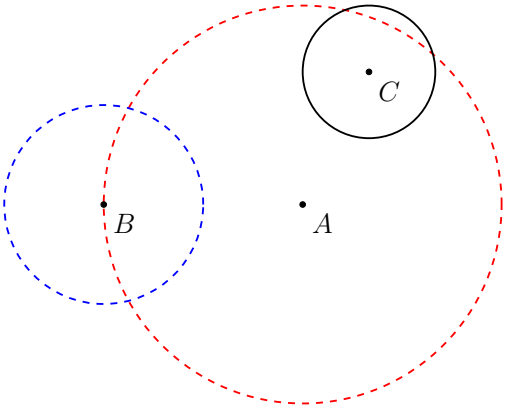
1: **function** GETLEVELSTDS
2: **for** every value, s, in array *splitsAtLastLevel* **do**
3: **if** $s == 0$ **then**
4: **if** $longDead[s] == 1$ **then**
5: Reduce its old standard deviation to a value less than the minimum value in deviations of previous level clusters.
6: **else**
7: Copy its standard deviation as it is.
8: **end if**
9: **end if**
10: **if** $s == 1$ **then**
11: Reduce its old standard deviation to a value less than the minimum value in deviations of previous level clusters.
12: **end if**
13: **if** $s > 1$ **then**
14: **for** each cluster that was formed due to the split **do**
15: $nearestNeighborDistance \leftarrow$ shortest distance between this cluster center and its siblings.
16: Deviation of this cluster is $nearestNeighborDistance$ / $separationFactor$.
17: **if** this deviation is bigger than the smallest deviation of previous level **then**
18: Deviation of this cluster is $nearestNeighborDistance$ / $divisonFactor$.
19: **end if**
20: **end for**
21: **end if**
22: **end for**
23: **return**
24: **end function**



(a) Three sigma regions of two cluster centers



(b) When child pseudo-cluster of A lies in overlapping region



(c) When child pseudo-cluster of A lies in non-overlapping region

Figure 5.5: Sampling of child cluster pseudo-centers : A and B are two cluster centers with their three sigma region boundaries in red and blue, respectively. To sample new pseudo-centers for cluster centers from A and B, if a point is sampled in the 3-sigma region of A or B, it is possible that it lies in the 3-sigma region of the other point as well. To avoid such points and to keep the cluster separate, we must check that a pseudo-center for a child cluster of a parent is not within the sampling region of any pseudo-center which is not its parent at the previous level.

representing that level for further processing. If some clusters are still alive, the next level is processed. Otherwise, data generation is started.

5.2.1.1 Probability of Splitting, Dying and Shrinking

The probabilities of staying and splitting helps the generator decide if a cluster should split into child clusters, stay as it is, or be considered as noise from the next level onwards. The situations where these are utilized are as follows

1. If a cluster has at least $2 * minClSize$ points
    - it can split into child clusters that would have at least $minClSize$ points.
    - it can stay or die (based on the probability of stay).
2. If a cluster has less then $2 * minClSize$ points
    - it can stay or die (based on the probability of stay).

Probability of split

The probability of split is calculated for a cluster at level h to decide if it should split to generate child cluster at level $h + 1$.

Definition 19 (Split probability decrease constant) *The split probability decrease constant, denoted by levelConstant, controls the rate of decay for the probability of split and stay with level in the hierarchy.*

The probability of splitting of a cluster depends on the *levelConstant* and two other factors:

1. The probability of splitting at a particular level. For deeper levels of the hierarchy, this probability is very low. Also, note that a cluster splits only if it stays.
2. The number of points in the cluster. A cluster with 500 points should be more likely to split than a cluster with 100 points. The more objects a cluster has, the higher is its probability to split.

Let h be the current level, $numPoints$ be the total number of points in the dataset, n be the number of points in the current cluster that may split. We define p as the fraction points in the cluster.

$$p = \frac{n}{numPoints} \tag{5.2}$$

We can use an exponential decay function with some value for $levelConstant$ to compute the probability of a split, combing the two factors:

$$P(split) = e^{-levelConstant*h/p} \tag{5.3}$$

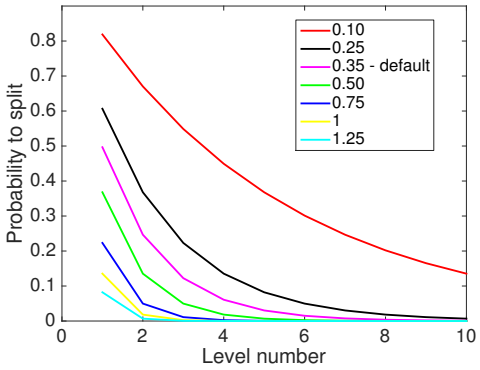
The higher the value of p, i.e. the bigger the size of the current cluster, the smaller would be the value of $levelConstant*h/p$ and, hence, the larger would be the probability of a split. Similarly, the smaller the value of p, the higher the value of $levelConstant * h/p$ and the smaller $P(split)$.

Let us assume that we have a 1000 point dataset and a cluster with 500 points. Figure 5.6a examines the behavior of the function for different values of $levelConstant$. By default $levelConstant = 0.35$ as the hierarchies produced with it were 5 levels, inclusive of the root, on an average, and hence, were easy to visualize. This value can be changed by the user. The higher the value, the smaller the hierarchies would be due to fewer numbers of splits.

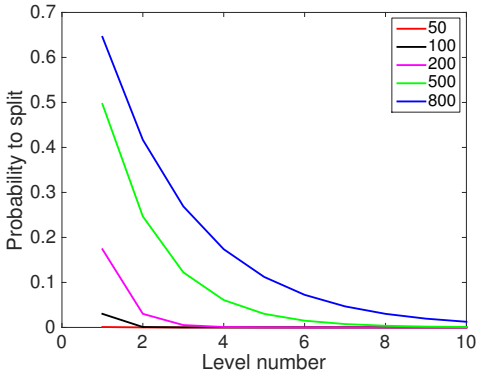
Next, we want to analyze how the probability of split changes with the number of points in the cluster. Figure 5.6b shows the respective curves when we vary the number of points in the cluster in a 1000 point dataset from 50 to 800. As expected, the probability of splitting a large cluster is high for any level. As the size of a cluster decreases and level number increases, the probability of split tends to be 0.

Probability of stay

The probability of staying and splitting are independent of each other. Only if a cluster is big enough to split and is able to stay at a particular level does the probability of stay come into play. We want it to have the following property: It should decrease as we go to lower levels. Thus, it is a function of the level



(a) The size of cluster was set to 500. As the value of *levelConstant* and current level increase, the probability of split also decreases. The higher the *levelConstant*, the lower the probability of split at a level.



(b) For the default value of *levelConstant*, the larger the number of points in the cluster, the larger is the probability of split at any level.

Figure 5.6: Probability of split at different levels of the hierarchy, for different values of split probability decrease constant, *levelConstant* and number of points in a cluster for a 1000 point dataset.

number and for the root level, it should be 1. The higher the initial probability to stay, the more levels we will have in the tree.

To reduce the number of user defined parameters, we have modeled the probability of stay similar to the probability of split using $levelConstant$ and the current level number, h:

$$P(stay) = e^{levelConstant*h} \tag{5.4}$$

The probability of stay is independent of the number of points in the cluster and the number of points in the dataset. It is function of the level number. As with probability of split, we observe the behavior of probability of stay for different levels in Figure 5.7 .

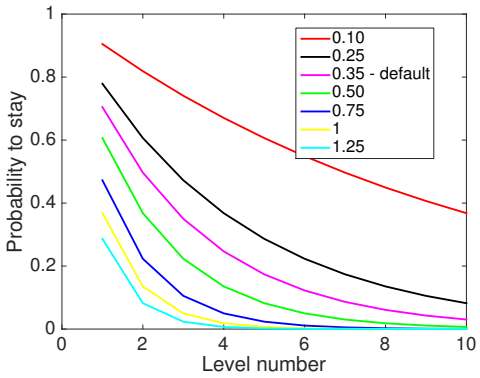
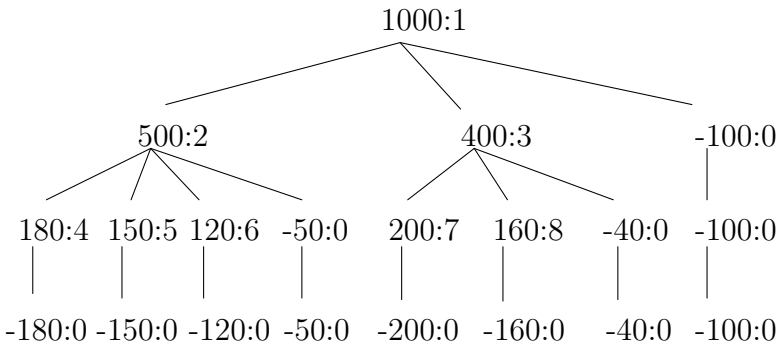


Figure 5.7: Probability of stay for a cluster at different levels of a hierarchy, for different values of split probability decrease constant, $levelConstant$: As the value of $levelConstant$ and current level increase, the probability of stay also decreases. The higher the $levelConstant$, the lower the probability of stay at a level and the smaller will be the overall hierarchy.

5.2.2 Data Generation Process

Once all pseudo-centers for clusters have been found, we generate the data. The pseudo-center of each cluster has a deviation associated with it and the number of points in its cluster. We simply generate the respective number of points from within the three sigma region of each center. Fox example, consider a 400 point cluster that splits into 3 child clusters. The 3 clusters



(a) The points kept at a level when a cluster splits are shown collectively as a child of the split cluster at next level with a negative sign, meaning they are noise at this level. Similarly, the points in all "dead" clusters are also shown with a negative sign at subsequent levels.

pseudo-center	No. of points in cluster	Standard Deviation	Indices of points in final Dataset	Last Alive Level
c_0	100	sd_0	900-999	0 (root)
c_1	50	sd_1	450-499	1
c_2	40	sd_2	860-899	1
c_3	180	sd_3	0-179	2
c_4	150	sd_4	180-329	2
c_5	120	sd_5	330-449	2
c_6	200	sd_6	500-699	2
c_7	160	sd_7	700-859	2

(b) Pseudo-centers description for clusters and the corresponding indexes of points in the generated data (obtained from the data generation step). Indexing starts from 0.

Figure 5.8: Data Generation for 1000 points and minimum cluster size of 100, $retentionFactor$ of 0.10, $branchingFactor$ of 5

are assigned the data in the range [0,400) to facilitate a fast build up of the hierarchy. This is shown in Figure 5.8.

5.2.3 Ground Truth Hierarchy Generation

Hierarchy generation is the process of assigning labels to each object for different levels of the tree. We simply assign labels [0,k] at each level where 0 is for noise and k is the total number of pseudo-centers for clusters generated.

In the final hierarchy, each level represents a cluster assignments for the points. The root level is populated first with all points assigned 1. Iteratively, we look at all other levels, assigning 0 to the clusters that have died and a label to those that exist at current level. Algorithm 8 shows the pseudo code

to build the final hierarchy. The generated hierarchy for Figure 5.8 is shown in Figure 5.9.

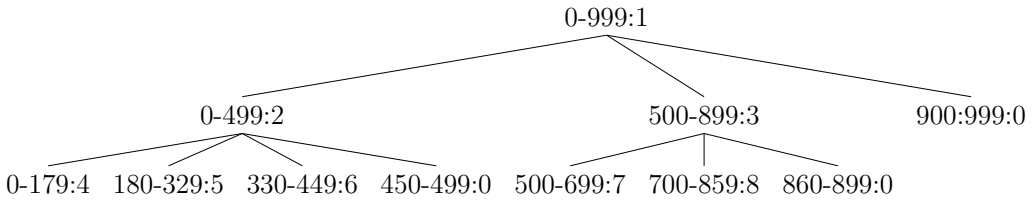


Figure 5.9: The Final Hierarchy for the Data Hierarchy for 5.8 : The format is index range : cluster number. It implies that the labels of the points in that index are the cluster number.

Algorithm 8 Pseudo code to build the final hierarchy file given all pseudo-centers of clusters and the data hierarchy

```
1:  function HIERARCHYBUILDER
2:      hierarchy ← matrix of size levels × numPoints + 1
3:      clusterNumber ← 2
4:      for level l in the hierarchy do
5:          hierarchy[l][0] ← thresholdDistance
6:          if l == lastLevel then
7:              Label all points as noise
8:              Set density level as 0.
9:          end if
10:         if l == 0 then                                  ▷ Root node
11:             Label all points as 1, belonging to one cluster.
12:             Set density level as 1 or the initial threshold distance.
13:         end if
14:         if l ≠ 0 or l ≠ lastLevel then
15:             for cluster, c present at this level do
16:                 if c died at a level before or at l then
17:                     Assign all its points cluster label 0.
18:                 else
19:                     Assign all its points label clusterNumber
20:                 end if
21:                 clusterNumber = clusterNumber + 1
22:             end for
23:         end if
24:     end for
25:     return
26: end function
```

5.3 Sample Hierarchies

In this section we present some sample datasets and their underlying hierarchies. We use the default settings listed in Table 5.1 and vary one of the major parameters - the branching factor, or the separation factor, or the retention factor - one at a time, to show the different hierarchies that can be obtained for the same number of points and dimensionality.

The split probability decrease constant was set to 0.35 as the hierarchies produced with it were 5 levels, inclusive of the root, on an average, and hence, were easy to visualize. The minimum cluster size is set to 30 for a 1000 points dataset to get deep enough hierarchies and to have enough points to redistribute amongst child clusters.

Parameter	Default setting
Number of points in dataset	1000
Dimensionality of data	2
Branching factor	5
Minimum Cluster Size	30
Separation Factor	3
Split probability decrease constant	0.35
Retention Factor	0.10

Table 5.1: Default setting for the Data Generator

5.3.1 Varying the Branching Factor

For these experiments, we vary the branching factor, keeping all other parameters constant. The *branchingFactor* defines the maximum number of splits that a cluster can split into. The effects of the branching factor can be summarized as

- The higher the number of possible splits, the larger the number of subclusters. With more subclusters, points are more evenly distributed between them, and the only way a hierarchy would be deep is if a cluster shrinks over multiple levels. Since the probability of split favors large

clusters, the chances of a split after the split of a cluster with large number of points would be low. This can be observed in Figure 5.10c and its corresponding hierarchical ground truth. The root node split into 9 clusters and none of them split into subclusters at deeper levels of the hierarchy.

- The branching factor only specifies the range of possible splits for a cluster. For instance, $branchingFactor = 10$ can lead to a split into 1, 2, 3, 4, 5, 6, 7, 8, 9, or 10 clusters while a $branchingFactor = 5$ would only give 1, 2, 3, 4 or 5 clusters. Thus, $branchingFactor = 5$ is just another case for $branchingFactor = 10$. An example is Figure 5.10e which was produced by setting the branching factor to 15 but the result was similar to branching factor of 10.

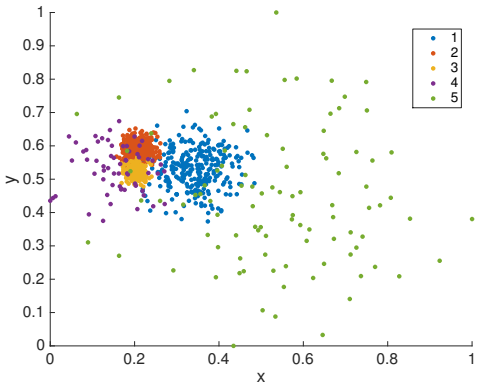
Figure 5.10 shows some datasets and their corresponding hierarchical ground truths for varying branching factors.

5.3.2 Varying the Separation Factor

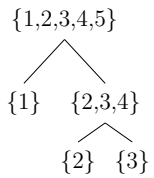
The $separationFactor$ defines how much overlap there should be between two adjacent clusters. The effects of separation factor are summarized as

- The smaller the value of this parameter, the more overlap there can be at a particular level. Figure 5.11 shows some sample datasets with different separation factor values for some two level hierarchy.

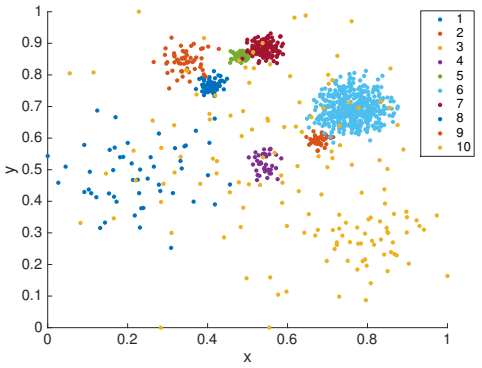
- Since clusters generally exists for more than one level, the standard deviation that they have when they are born decreases by the $divisionFactor$ at every level. Higher values of $separationFactor$, lead to further apart clusters, that will shrink more quickly at subsequent levels. Contrasting between two datasets generated for the same $separationFactor = 3$ but different number of levels, see Figure 5.12, we find that for the deeper hierarchy (Figure 5.12b), we already have very small densely packed clusters in blue. Based on this reasoning, it is safe to have a small separation factor and the default is set to 3.



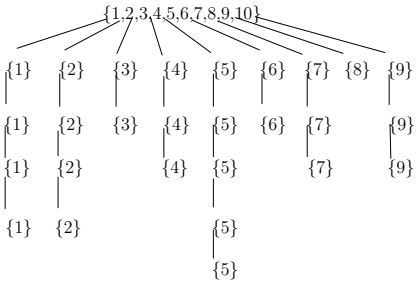
(a) Branching Factor = 5



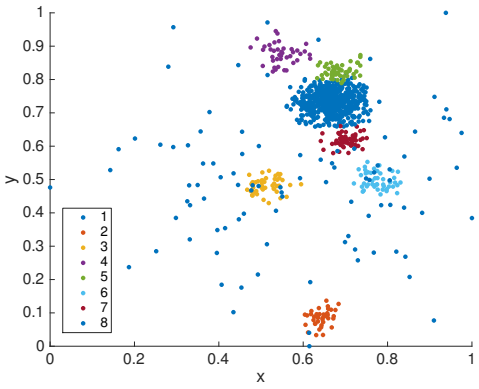
(b) Corresponding hierarchy for 5.10a



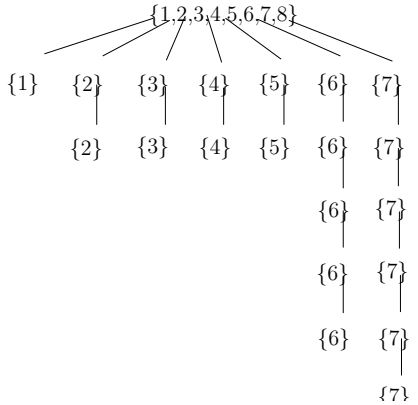
(c) Branching Factor = 10



(d) Corresponding hierarchy for 5.10c

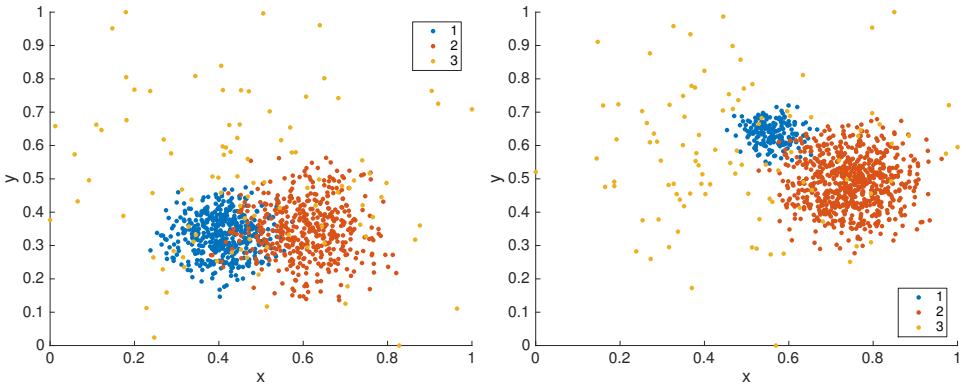


(e) Branching Factor = 15



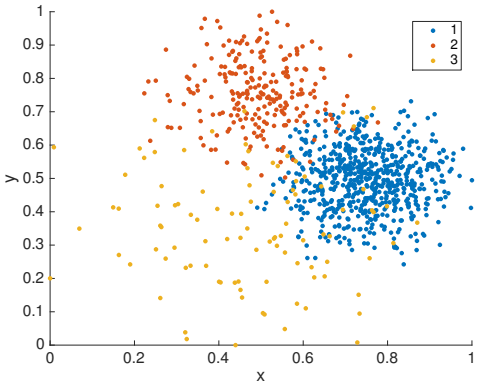
(f) Corresponding hierarchy for 5.10e

Figure 5.10: Effect of Branching Factor : Generated 1000 point datasets in 2 dimensions with minimum cluster size 30, *rententionFactor* of 0.10 and separation factor of 3. We observe that $branchingFactor = 10$ can lead to numerous splits at root levels leading to few or no splits at further levels. Note that the labels so not represent the actual labels of clusters in the hierarchy. These numbers are only for visualization.

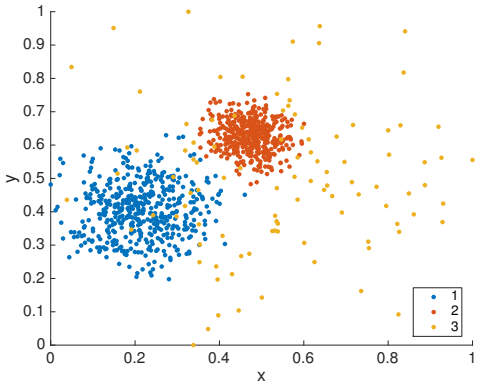


(a) Separation Factor = 2.5

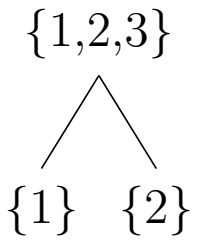
(b) Separation Factor = 3



(c) Separation Factor = 3.5

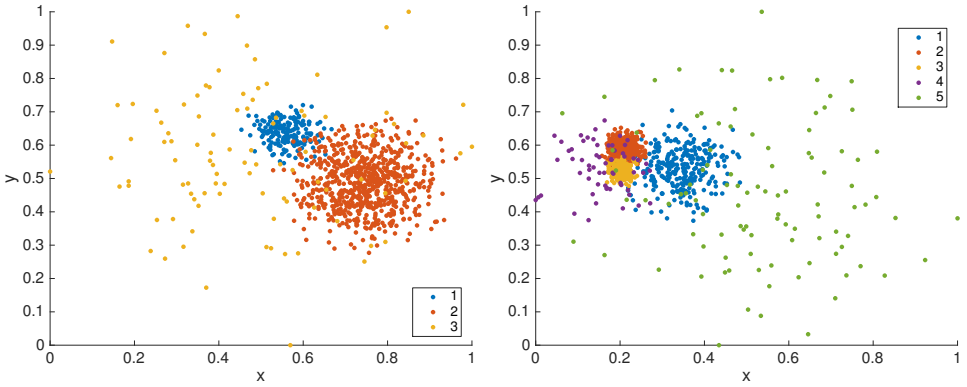


(d) Separation Factor = 4



(e) Corresponding hierarchy for all the datasets

Figure 5.11: Generated 1000 point datasets with minimum cluster size 30, $rententionFactor$ of 0.10 and $branchingFactor$ of 5. For the same structure of datasets with different separation factor values, higher separation factors generate datasets that overlap less, (a) having the most overlap between the two clusters and $separationFactor = 2.5$ while (d) has the least overlap amongst these datasest with $separationFactor = 4$.



(a) Short hierarchy example (b) Deeper hierarchy example

Figure 5.12: Generated 1000 point datasets in 2 dimensions with *separationFactor* of 3, minimum cluster size 30, *rententionFactor* of 0.10 and *branchingFactor* of 5. As clusters shrink at deeper levels of the hierarchy, their standard deviation reduces. This implies that if we already start with a small value of standard deviation to avoid overlap, shrinking would cause the clusters to be concentrated in very small regions due to low standard deviations.

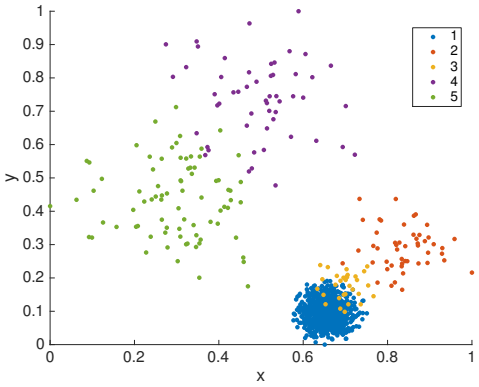
5.3.3 Varying of Retention Factor

The retention factor defines the fraction of points that stay at a level – become noise – when a cluster splits. We vary the retention factor from {0,0.1,0.2}, allowing 0 points, 10% of the points and 20% of the points to be retained at the parent level. Some observations about the retention factor are:

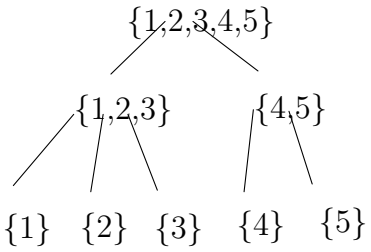
- If no points are retained at the parent level, the nested clusters typically would be surrounded by less dense regions.
- The higher the retention factor, the lower the number of points to distribute between child clusters. Thus, there must be a balance between the amount of points retained at a parent and the number of points in the child cluster.

5.4 Summary and Future Work

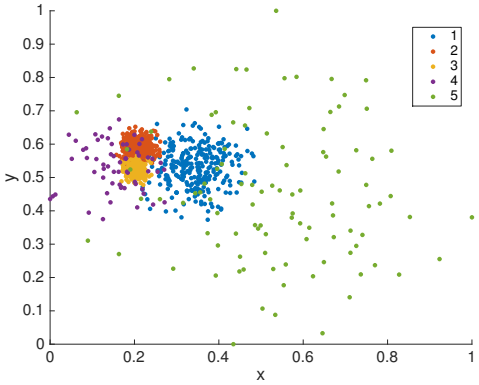
In this chapter, we proposed a novel data generator that produces data with hierarchical cluster structure. We used a method of sampling Gaussian distributions for the generation of data. Also, we assumed that the Gaussians were



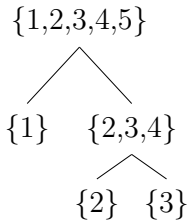
(a) Retention Factor = 0



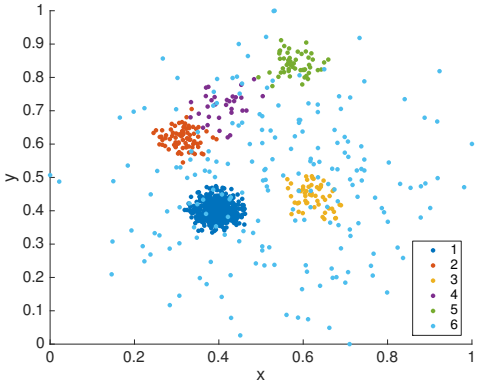
(b) Corresponding hierarchy for 5.13a



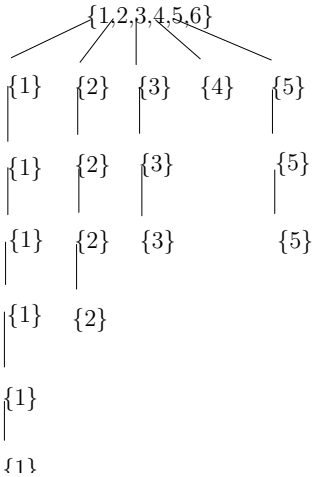
(c) Retention Factor = 0.10



(d) Corresponding hierarchy for 5.13c



(e) Retention Factor = 0.20



(f) Corresponding hierarchy for 5.13e

Figure 5.13: Effect of Retention Factor : Generated 1000 point datasets in 2 dimensions with minimum cluster size 30, *branchingFactor* of 0.10 and separation factor of 3. The *rentionFactor* can be seen as the amount of noise that should be between child clusters or the number of less-dense points that help connect child clusters. Note that the labels so not represent the actual labels of clusters in the hierarchy. These numbers are only for visualization.

isotropic, i.e. same standard deviation in each dimension. Each dimension was independent of the other and data was uncorrelated between dimensions. Using probabilities of split and stay, we were able to ensure that there were more splits for large clusters. However, there is always the small chance that in spite of a large probability to split/stay, a big cluster dies or a small cluster splits. By using the minimum cluster size parameter, we ensure that no cluster has fewer points than the minimum cluster size value. Also, we presented numerous hierarchies and datasets that can be obtained from the generator.

In the future, we would like to extend our work to anisotropic Gaussians, allowing dependence between the dimensions and we would use a covariance matrix to record the standard deviation as well the degree of correlation between the data in each dimension. Mahalanobis distance [47] would be used to check if a point is within the three sigma region of a parent. The result would be elliptical clusters, instead of spherical shaped-clusters.

The data generator models noise as a cluster, i.e. all points that become noise are given the label 0 irrespective of which level they are at and which cluster they initially belonged to. Thus, the datasets generated can be used by algorithms that represent noise as one cluster. Other HCA algorithms like Single Linkage [37], on the other hand, give a unique label to every noise object. Every noise object belongs to its own singleton cluster. These algorithms would not be able to use the datasets generated. Thus, in the future, we would like to have an implementation of noise that can be used by both kinds of datasets and not just limited to density-based clusters.

Chapter 6

Case Study of Integrating Kernel Density Estimates into Hierarchical Clustering

The most important user defined parameter for HDBSCAN* is m_{pts} which is the number of nearest neighbors to consider (including the point itself) while estimating the density using the unnormalized kNN kernel [12]. For our extended version of HDBSCAN* with kernels, HDBSCANk, we introduced the parameter, *kernel*, which is the kernel density estimate the user wants the algorithm to use for its calculations, the connectivity method, *method*, that we elaborated on in Chapter 4 and the smoothing parameter, *bandwidth*, which defines the bandwidth of the kernel in each dimension. In the case of the kNN kernel, *bandwidth* is simply the value of k. Our preliminary experiments using one *kernel* and *method* left some questions unanswered in Section 4.2. Testing using one kernel does not say much about a generalized algorithm that can be used with any kernel. Comparing results of only flat partitions does not tell us how good a hierarchical clustering algorithm is in discovering complete ground truth hierarchies. Similarly, just evaluating one bandwidth value does not tell much about the robustness of a technique.

To find answers to all these questions, we undertook a detailed case study. As examples of kernels with fixed bandwidth parameter, we integrated the Gaussian kernel and the Epanechnikov kernel into our framework, HDBSCANk. We performed an exhaustive search to find the best bandwidth based on op-

timizing internal cluster validation measures, DBCV [51] and SI [60]. These results were compared with the best performance we can get by using statistical bandwidth estimators like Silverman's Rule of thumb [66], and cross validation using Maximum Likelihood [18, 25] mentioned in Section 2.2.1.1. We also integrated the All-Points-Core-Distance kernel into our algorithm, HDBSCANk. This kernel was recently proposed by Moulavi [50]. Lastly, all kernel results were evaluated against the original HDBSCAN* as the baseline algorithm.

We used a variety of datasets – two dimensional datasets downloaded from Joensuu Clustering Datasets [1] and two self generated two dimensional dataset, real datasets downloaded from the UCI repository [43], artificially generated high dimensional datasets using Handl and Knowles [27] data generator and hierarchical datasets obtained from our data generator with hierarchical ground truth proposed in the Chapter 5.

6.1 Cluster Validation Measures

In this section we explain the cluster validation measures used in the experiments.

6.1.1 External Validation Measure

External cluster validation measures are used to evaluate the extent to which cluster labels match externally available ground truth labeling.

Adjusted Rand Index (ARI)

The Rand Index [56] is a measure of similarity between two data clusterings. Given a dataset with n objects $\mathbf{S} = \{o_1, \ldots, o_n\}$, and two labellings of $\mathbf{S}$: $\mathbf{X} = \{x_1, \ldots, x_n\}$ and $\mathbf{Y} = \{y_1, \ldots, y_n\}$, where x_i and y_i are the possible cluster assignments for the object o_i, we can calculate

- a - the number of pairs of objects that have the same label in both $\mathbf{X}$ and $\mathbf{Y}$.

- b - the number of pairs of objects that have the different label in $\mathbf{X}$ as well as $\mathbf{Y}$.
- c - the number of pairs of objects that have the same label in both $\mathbf{X}$ but different labels in $\mathbf{Y}$.
- d - the number of pairs of objects that have the different label in $\mathbf{X}$ but same labels in $\mathbf{Y}$.

Using a, b, c, d, the Rand index is defined as:

$$R = \frac{a+b}{a+b+c+d}$$

The Rand Index is between 0 and 1 and can give high values even for random groupings of objects. Also, the expected value of the Rand Index of two random partitions does not take a constant value (say zero). The Adjusted Rand Index (ARI) proposed by Hubert and Arabie [34] assumes the generalized hypergeometric distribution as a model of randomness and overcomes these shortcomings of the Rand Index.

The ARI is in the range -1 to 1, giving 1 for a perfect match between the two clusterings and -1 if the index is less than the expected value. Hubert and Arabie [34] proposed the general form for adjustment of an index by chance, for an index with a constant expected value, is given by

$$AdjutedRandIndex = \frac{Index - ExpectedIndex}{MaxIndex - ExpectedIndex}$$

Using Rand Index, ARI is defined as:

$$ARI = \frac{a - \frac{(a+c)(a+b)}{a+b+c+d}}{\frac{2a+b+c}{2} - \frac{(a+c)(a+b)}{a+b+c+d}} \tag{6.1}$$

where a, b, c, d are defined as above.

Hierarchy Agreement Index (HAI)

The Hierarchy Agreement Index (HAI) [36] is based on the Rand Index [56] and compares the hierarchical structure of two hierarchies. Its values ranges from 0 (dissimilar hierarchies) to 1 (same hierarchical structure). The relatedness

of each pair of objects in the dataset under two input hierarchies is compared and the results are aggregated to achieve a single unified measure of hierarchy correspondence.

To perform this evaluation for a n object dataset, a *hierarchy distance*, $d_{\mathcal{H}}(a, b)$ between two elements a and b in cluster hierarchy $\mathcal{H}$ is defined. Let $n_o^{a,b}$ be the smallest node in the hierarchy containing both a and b, and n_{oD} be the total number of objects at node n_o. Then, $size(n_o) = |\frac{n_{oD}}{n}|$ represents the proportion of total objects in n_o as compared to the dataset size n.

$$d_{\mathcal{H}}(a, b) = \begin{cases} 0 & \text{if } n^{a,b} \text{ is a leaf node} \\ size(n^{a,b}) & \text{otherwise} \end{cases}$$

Two elements that lie in the same leaf node are maximally close under that hierarchy. Finally, HAI is given by:

$$HAI(\mathcal{H}_1, \mathcal{H}_2) = 1 - \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} |d_{\mathcal{H}_1}(x_i, x_j) - d_{\mathcal{H}_2}(x_i, x_j)| \tag{6.2}$$

The distance $d_{\mathcal{H}}$ is independent of the specific structure of the hierarchy because it is unaffected by the number of intermediate nodes between x_i , x_j and n^{x_i,x_j}.

6.1.2 Internal Validation Measures

These measures are used to measure the goodness of a clustering structure using only the data itself.

Density Based Cluster Validation (DBCV)

DBCV is a relative validation index for density-based, arbitrary shaped clusters. Let $\mathcal{C} = \{C_1, \ldots C_k\}$ be the clustering solution being evaluated that assigns each object to one of the k clusters. The relative density connection between two objects is assessed using the following formulation of the density based distance, The all-points-core-distance (inverse of the density) of an object $\mathbf{x}$, belonging to cluster C_j with respect to all other $n_j 1$ objects in C_j is defined as:

$$all_{pts}coredist(\mathbf{o}) = \left(\frac{\sum_{x_i \in C_j, x_i \neq \mathbf{x}} \left(\frac{1}{d(\mathbf{x}, x_i)} \right)^d}{n_j - 1} \right)^{-1/d} \tag{6.3}$$

The original space in which the clustering solution is embedded is transformed to the density space using the all points core distance to represent density of an object and the Mutual Reachability Distance [41], between two objects x_i and x_j to define the connectivity between them.

$$d_{mreach}(\mathbf{x}_i, \mathbf{x}_j) = \max(d_{core}(\mathbf{x}_i), d_{core}(\mathbf{x}_j), d(\mathbf{x}_i, \mathbf{x}_j)) \tag{6.4}$$

Using $all_{pts}coredist$ and d_{mreach}, a Minimum Spanning Tree is built in Mutual Reachability Space. From this MST, individual MSTs for each cluster are derived, capturing both the density and shape of the cluster. The edges in the MSTs are used to identify the regions of low density within the clusters, while the edges in the full MST are used to identify regions of high density between pairs of clusters, representing the density separation between clusters.

The density sparseness of a cluster (D_{Sp}) is the maximum edge weight value in its corresponding MST, considering only its internal vertices, i.e., vertices that have a degree greater than one. Considering the edges connecting two clusters, the density separation of the two clusters (D_{Sep}) is given by the minimum d_{mreach} between the internal vertices of the clusters, thus, capturing, the region with lowest density between the clusters.

A validation index V_C is computed for each cluster using its density sparseness, D_{Sp}, and density separation, D_{Sep} w.r.t the cluster nearest to it.

The denominator is a normalization term that ensures V_C is in the range [-1,1] and the *min* allows the closest neighbor cluster to be selected for calculations.

$$V_C(C_i) = \frac{\min_{j \neq i}(D_{Sep}(C_i, C_j)) - D_{Sp}(C_i)}{\max(\min_{j \neq i}(D_{Sep}(C_i, C_j)), D_{Sp}(C_i)} \tag{6.5}$$

DBCV is the combination of quality of all clusters, define as:

$$DBCV(\mathcal{C}) = \sum_{C \in \mathcal{C}} \frac{|C_i|}{|\mathbf{X}|} V_C(C_i) \tag{6.6}$$

By multiplying the quality of each cluster by the percentage of objects it contains, noise is implicitly considered. A clustering solution with high amount of noise is penalized, receiving a small score, even if its clusters have a good individual score according to Equation 6.5.

Silhouette Index (SI)

The Silhouette Index is an internal validation measure that compares how similar an object is to its own cluster compared to other clusters [60]. The Silhouette Index for a single object ranges from -1 to 1, where a high value indicates that the object fits well into its own cluster and fits poorly into neighboring clusters. If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters, or there may be no structure in the data.

Consider an n object dataset $\mathbf{X} = \{x_1, \ldots x_n\}$ and a clustering solution $\mathbf{C} = \{C_1, \ldots, C_k\}$ that separated the objects in $\mathbf{X}$ into k clusters. For every object $x \in \mathbf{C_i}$, let a_x be the average dissimilarity of x with all other objects within the same cluster C_i.

Let b_x be the lowest average dissimilarity of x to any other cluster of which x is not a member. Thus, from the set $\mathbf{C}$, we select the cluster C_j that is closest to x and $C_i \neq C_j$. The silhouette value of x is defined by

$$s_x = \begin{cases} 1 - \frac{a_x}{b_x} & \text{if } a_x < b_x \\ 0 & \textbf{if } a_x = b_x \\ \frac{a_x}{b_x} - 1 & \text{if } a_x > b_x \end{cases} \tag{6.7}$$

The Silhouette index of the dataset $\mathbf{X}$ is the average silhouette for all objects.

$$SI(\mathbf{X}) = \frac{1}{n} \sum_{i=1}^{n} s_{x_i} \tag{6.8}$$

6.2 Experimental Setup

For the case study described in this chapter, we normalized the datasets to $[0, 1]$ in each dimension. Consider a n object dataset $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2} \ldots \mathbf{x_n}\}$ in d dimensional space. The value of each object in dimension j is $(\mathbf{x}_{1j}, \mathbf{x}_{2j}, \ldots, \mathbf{x}_{nj})$. By calculating the minimum and maximum value of objects in dimension j, the normalized value z_{ij} of object $\mathbf{x}_i$ in j dimension is given by

$$z_{ij} = \frac{\mathbf{x}_{ij} - \min(\mathbf{x}_{\cdot j})}{\max(\mathbf{x}_{\cdot j}) - \min(\mathbf{x}_{\cdot j})} \tag{6.9}$$

- **Kernel**

  We use the All-Points-Core-Distance [51], Gaussian and Epanechnikov kernels for our experiments.

- **Edge Weight Estimation Methods**

  The following edge estimation methods, proposed in Section 4.3 to define the connectivity between a pair of objects, are compared in the experiments.

  1. Midpoint Estimation, hereafter referred to as **M1**, estimates the density for an edge as the density at the midpoint due to the whole dataset.
  2. Midpoint Estimation using top contributors, hereafter referred to as **M2**, estimates the density for an edge as the density at the midpoint due to a subset of the dataset.
  3. Torque Rule Estimation using top contributors, hereafter referred to as **M3**, estimates the density for an edge as the density at the torque point due to a subset of the dataset.
  4. Golden Search using top contributors, hereafter referred to as **M4**, estimates the density for an edge as the density at the point where the density estimate is minimum, calculated by performing golden search and minimizing the K.D.E. value.

  Since Interval Estimation and Torque Rule Estimation consistently gave results that were no better than Midpoint Estimation for most datasets, we do not analyze them in this chapter.

- **Calculation of Bandwidth**

  1. We used two common heuristics to estimate bandwidths – Silverman's Rule of thumb [66] and Maximum Likelihood [18, 25]. These were calculated using the functions available in R and MATLAB, respectively. Their respective equations can be found in Chapter 2, Equation (2.3) and Equation (2.4).

2. We also performed a small experiment with three datasets of dimensionality $\{2, 10, 64\}$ and found that the bandwidth range of $[0.001, 0.200]$ with increments of 0.001, is sufficient to find an optimal partition for any dataset. The small experiment was as follows: Consider a value $h \in [0.001, 0.200]$. Using h as the bandwidth in every dimension for some kernel in HDBSCANk, we obtain a hierarchy and flat partitions from it. On evaluating all partitions from the hierarchies built for different values of $h \in [0.001, 0.200]$, we found that the value of h that gave the best possible flat partition was in this range. This is one way of performing an exhaustive search for the optimal bandwidth.

- **Partition Extraction** For datasets that did not have a hierarchical ground truth, we used the following partition techniques:

  1. Stability partition: This is the default flat partitioning technique used by HDBSCAN* which is an instance of FOSC [11].

  2. All partitions: In practical scenarios, ground truth is not available. Often, an internal validation index is used to decide which parameter gives the best result. Based on this practice and to obtain a sense of the "potential" of a hierarchy, i.e., what could be the best possible extracted partition, from each hierarchy, we extracted and evaluated all possible partitions. This can be achieved as follows: Consider a simple cluster tree for the hierarchy, as shown in Figure 6.1. The set of all possible extractions from this tree are all combinations of clusters such that each object has only one cluster assignment. Thus, parent clusters cannot be combined with their child clusters. For this sample tree, all possible partitions are : $\{\mathbf{C_1}, \mathbf{C_2}\}$ and $\{\mathbf{C_1}, \mathbf{C_3}, \mathbf{C_4}\}$.

- **Internal Validation**

  For each bandwidth and selected partition extraction method, we report three types of values -

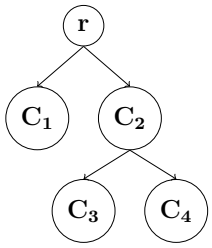


Figure 6.1: Sample Cluster Tree

1. The maximum ARI that can be obtained from the hierarchies. This represents the potential of the hierarchy generated.
2. The ARI value of the bandwidth that has the highest value of DBCV.
3. The ARI value of the bandwidth that has the highest value of SI.

- For datasets with hierarchical ground truth, the hierarchy for each combination of *kernel*, *method*, *bandwidth* was compared with the given hierarchical ground truth using HAI.
- **Minimum Cluster Size** Both HDBSCAN* and HDNSCANk have the parameter, m_{clSize}, for hierarchy simplification. In HDBSCAN*, m_{clSize} is set to be the same value as m_{pts}, though this can be varied. For consistency in hierarchies extracted from both algorithms, we decided to fix the value of m_{clSize}. Also, HDBSCANk does not have a m_{pts} parameter. Only when using the kNN kernel $bandwidth = k = m_{pts}$; for other kernels, *bandwidth* is a small decimal number and it cannot be used to set the value of m_{clSize}.

A summary of the parameters of the case study is listed in Table 6.1.

6.2.1 Case Study Steps

The aim of our study is to observe the behavior of the kernels and methods for edge estimation in order to get insight into their performance. Based on the type of available ground truth, our experiments can be summarized as:

1. Case 1 - Available ground truth is hierarchical: In this case, the hierarchy obtained from HDBSCANk and HDBSCAN* are compared with the

Table 6.1: Experimental Parameters

Parameter	Settings
Kernel	Normal Epanechnikov All-Points-Core-Distance Kernel
Internal Validation Index	DBCV SI
Bandwidth	Silverman's Rule of Thumb Bandwidth Maximum Likelihood Bandwidth Exhaustive search in range [0.001, 0.200]
Edge Weight Estimation Methods	**M1** Midpoint estimation **M2** Midpoint estimation using top contributors **M3** Torque Rule using top contributors **M4** Golden search using top contributors

given ground truth using HAI. For the exhaustive search, the bandwidth with the maximum HAI and the corresponding HAI value are reported.

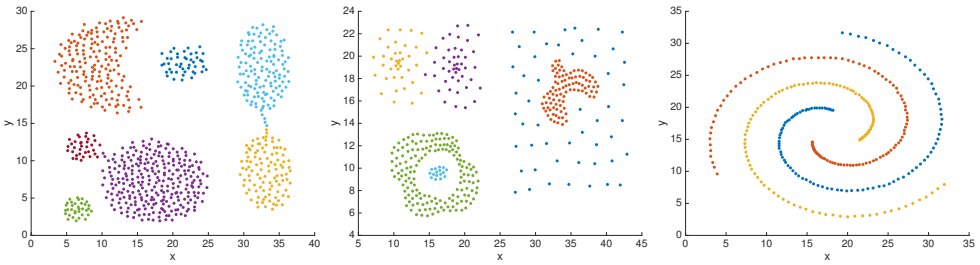
2. Case 2 - Available ground truth is a single labeling of the objects: In this case, for the stability partition, we report the ARI. When all partitions are evaluated, we report the bandwidth for the hierarchy that obtained the maximum ARI, the bandwidth with maximum DBCV value and the bandwidth with maximum SI value. The ARI of the partitions with best DBCV and the best SI are reported as well.

6.3 Results

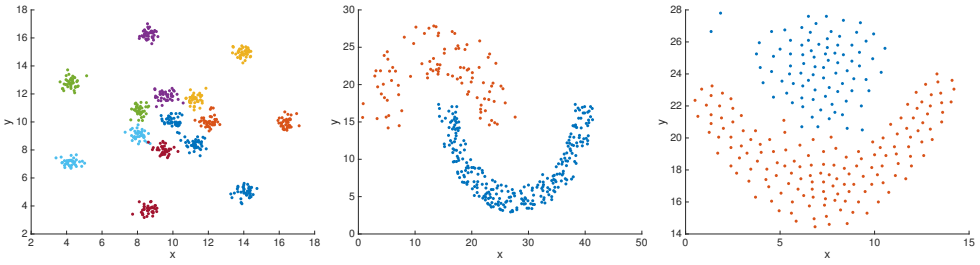
This section is organized as follows. We have 4 major categories of datasets: two dimensional datasets, real datasets, high dimensional datasets and datasets with hierarchical ground truth. For each category, we present an aggregate result which summarizes the general behavior of all the datasets together. Next, we illustrate the results of a few datasets in more detail.

6.3.1 Results on Two dimensional Clustering Data Sets

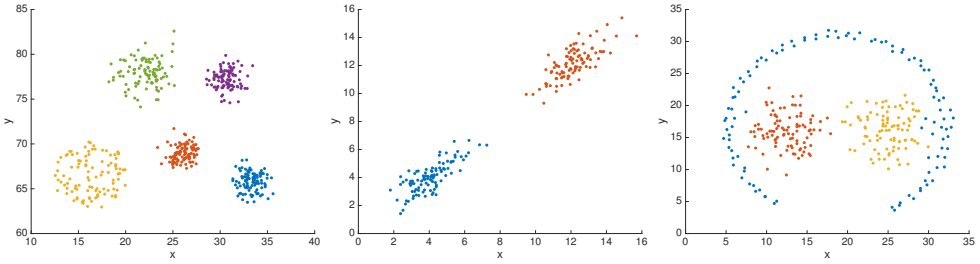
A summary of the two dimensional datasets is given in Table 6.2 . The plots of these datasets are shown in Figure 6.2 .



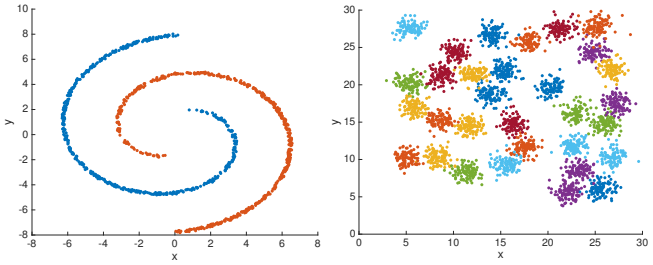
(a) aggregation (b) compound (c) spiral



(d) r15 (e) jain (f) flame



(g) 5gaussians (h) 2gaussians (i) pathbased



(j) 2spirals (k) d31

Figure 6.2: Two Dimensional Datasets

Dataset Name	No. of points	No. of Clusters
aggregation	788	7
compound	399	6
spiral	312	3
r15	600	15
jain	373	2
flame	240	2
5gaussians	500	5
2gaussians	200	2
pathbased	300	3
2spirals	1000	2
d31	3100	31

Table 6.2: 2 dimensional datasets Summary

For datasets with less than 1000 points, a minimum cluster size of 10 was used; for 2spirals and d31, minimum cluster size was set to 50.

The *potential* of an algorithm is the best ARI value we could get if we knew the right parameters to use (to obtain this best result). For HDBSCAN*, this potential is found if we test every value of m_{pts} and from every hierarchy that is produced, extract all possible partitions. Since there are no parameters for the All-Points-Core-Distance kernel, the search for the best partition to determine "potential" is limited to a single hierarchy. By comparing these partitions with the available ground truth, we can determine the maximum value of ARI that is achievable, hence, recording the parameters that lead to this best possible result. We ran this experiment on all 11 datasets. Figure 6.3 shows the bar plot comparing all these techniques after aggregating the ARIs of all datasets. We found that

- Our baseline HDBSCAN*, on an average, also performed well, giving close to 0.95 ARI.
- The All-Points-Core-Distance Kernel could give about 0.8 ARI.
- In terms of potential of the Normal kernel, if all partitions were evaluated for every hierarchy, a maximum ARI of close to 0.95 could be obtained with Method M3 (Torque Rule estimation using top contributors).

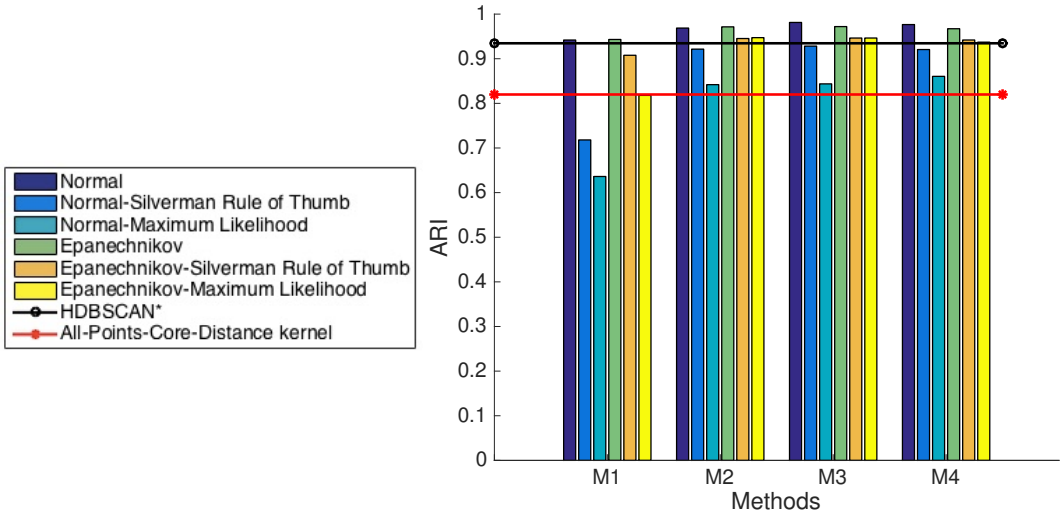


Figure 6.3: Maximum ARI that can be achieved on an average for Two Dimensional Datasets

- The Epanechnikov kernel was also not far behind, with ARI close to 0.95 when used with the edge estimation method M2, Midpoint Estimation using top contributors.
- The simple edge estimation method of Midpoint Estimation, M1, was outperformed by all other kernel-edge weight estimation method combinations.
- The plug-in bandwidths performed better with the Epanechnikov kernel than with the Normal kernel. Silverman's rule of thumb was better than Maximum Likelihood estimation of bandwidth.

Based on this average behavior, we can conclude that HDBSCANk is capable of finding the ground truth. It can perform better than HDBSCAN* and the All-Points-Core-Distance kernel. However, there is still the problem of finding the right bandwidth. Simply using a precomputed bandwidth with Epanechnikov kernel can give ARI close to the potential ARI.

In practical applications of unsupervised machine learning algorithms, the ground truth of the datasets is not available. If there were enough resources available in terms of computation power and time, one could still use internal validation measure to select the best parameters. For both HDBSCAN* and

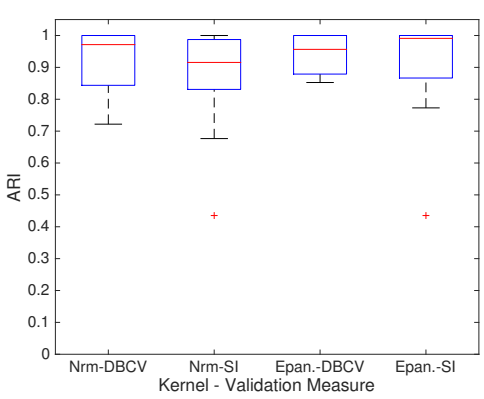
HDBSCANk, this process would be similar to what we did to find the potential of the algorithms. In the same setting as above, we introduced interval validation measures – DBCV and SI – to help us find the best parameter. We then compare the ARI value from this 'best' partition to what we are capable of achieving.

We compared how a kernel would perform if the resources were available, and determine how the two internal validation measures compare to each other on this task. We use box plots to show the variation of ARI of partitions selected by optimizing DBCV and SI. The results are shown in Figure 6.4.

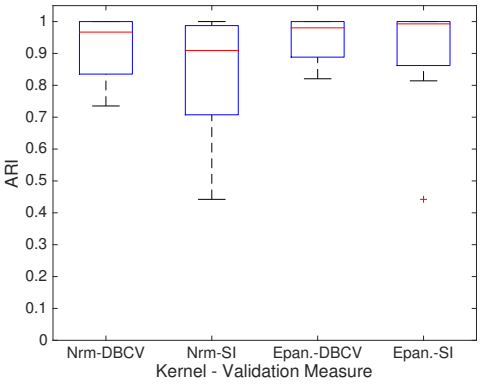
We can summarize these results as follows:

- Among the kernels, Epanechnikov shows the least variation in ARIs obtained by choosing the best partition with SI. With SI, the median value of ARI that can be obtained for this kernel by exhaustively searching for the best bandwidth is close to 1. See Figures
- The Normal kernel with DBCV has consistency small Interquartile range, irrespective of edge estimation methods.
- The Normal kernel can identify a better partition when used in combination with DBCV as compared to SI.
- The outliers on the plots, Figures 6.4a, 6.4b, 6.4c and 6.4d correspond in all cases to the d31 dataset with DBCV and the spiral with SI.
- HDBSCAN* and All-Points-Core-Distance kernel, Figure 6.4e, have a long box plots, meaning the results vary a lot. Usually, as in the case of HDBSCAN-DBCV combination, it is more probable to get a partition with ARI less than 0.75 than one with higher ARI.

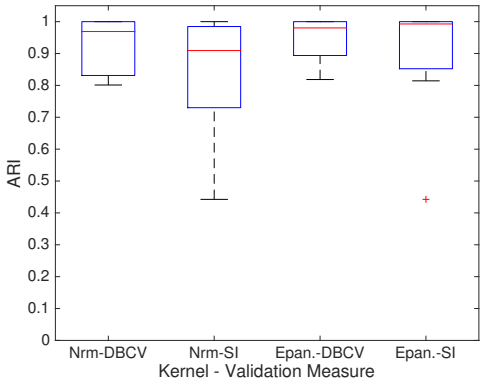
From these box plots we conclude that the edge estimation methods are more robust when used with SI or DBCV as compared to applying the same validation measures for selection of best partition with HDBSCAN* or the All-Points-Core-Distance kernel does not always give a high ARI. Also, with edge



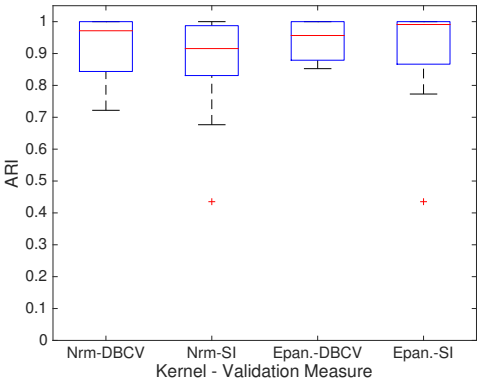
(a) M1 - Midpoint Estimation



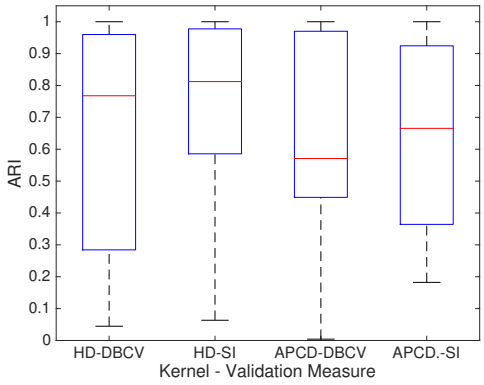
(b) M2 - Midpoint Estimation using top contributors



(c) M3 - Torque Estimation using top contributors



(d) M4 - Golden Search on top contributors



(e) HDBSCAN and All-Points-Core-Distance Kernel

Figure 6.4: Performance of Validation Measures - Density-based cluster validation (DBCV) and Silhouette Index (SI) - with HDBSCAN*(HD) and HDBSCANk with the All-Points-Core-Distance kernel (APCD), the Normal kernel (Nrm) and the Epanechnikov (Epan) kernel.

weight estimation methods that use top contributors, the partitions selected with best DBCV and best SI were high ARI partitions.

Though the box plots indicate which internal validation index can be useful, we find that doing such an extensive analysis of every possible partition is time consuming and with an algorithm that is already $\mathcal{O}(dn^2 + n^2 \log n)$ in d dimensions, it would not be feasible to undertake such an analysis in many practical applications. Thus, we focus the rest of the experimental analysis on evaluating the performance of all algorithms using the partitions we can get from the hierarchies by the default extraction method based on stability.

On an average, the maximum ARI achievable with HDBSCAN* (Table 6.3) by evaluating all partitions for all hierarchies built for different values of m_{pts}, was 0.93. When evaluating only the stability partition, the ARI value of 0.922 was obtained. Using internal validation measures, DBCV and SI, produced stability partitions with ARI of 0.78 and 0.82, on an average.

For the Normal kernel (Figure 6.5), on an average for all two dimensional datasets, the maximum ARI achieving by evaluating all partitions from hierarchies of bandwidths [0.001, 0.200], was about 1 with M3. If only stability partitions were evaluated for the same hierarchies, the maximum ARI value achievable was about 0.95 with M2. For all methods, the best ARI from stability partitions was close to that obtained by evaluating all partitions. Now, looking at what we could achieve, using the internal validation measure DBCV gave the ARI value of close to 0.8 with M2-M4; with SI, the average ARI of 0.7 could be obtained. The heuristic bandwidth's stability partitions showed similar behavior. Thus, using internal validation measures or heuristic bandwidths with the Normal kernel did not find stability partitions that were better than HDBSCAN* or the All-Points-Core-Distance kernel. Note that datasets like d31 and spiral, scatter plots shown in Figures 6.2k and 6.2c respectively, were identified to be outliers in the Figure 6.4a-6.4d. Their ARI values would affect

Potential	Best ARI from Stability Partition	Validation Measure used for Selection	
		SI	DBCV
0.93	0.92	0.78	0.82

Table 6.3: Different ARIs from HDBSCAN*

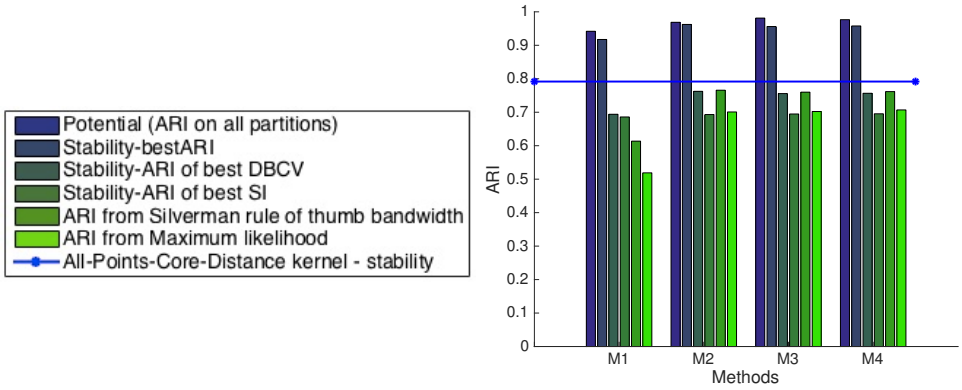


Figure 6.5: Maximum ARI from Stability Partitions averaged over all 11 datasets: HDBSCANk with Normal Kernel

the mean result for the potential plots as well the plots for the ARI values obtained from the stability partitions as well.

A similar analysis with the Epanechnikov kernel (Figure 6.6) was conducted. Though the potential of the hierarchies was found to be close to the ARI value of 1 with edge weight estimation methods M2-M4, and the highest achievable ARI from stability partitions was also close to 1, the internal validation measures could not select a partition with such values of ARI. The heuristics bandwidth estimators were able to perform better than the All-Points-Core-Distance kernel for all edge weight estimation methods, Silverman's rule of thumb being more accurate. Using either of the heuristic bandwidths with methods M2-M4, the ARI achieved was more that what HDBSCAN* or All-Points-Core-Distance kernel can achieve. Thus, with the

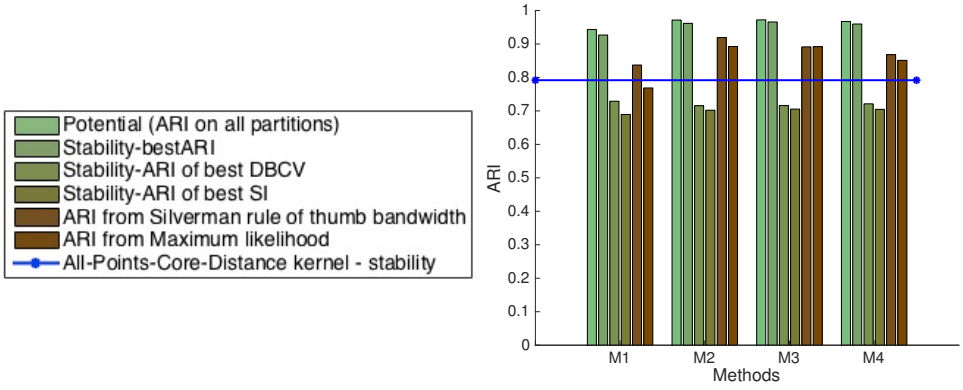


Figure 6.6: Maximum ARI from Stability Partitions averaged over all 11 datasets: HDBSCANk with Epanechnikov Kernel

two dimensional datasets,the Epanechnikov kernel did better than the Normal kernel. On an average for the 11 datasets, using the internal validation indexes selects partitions with lower ARI value. The heuristic bandwidth estimators perform better than DBCV or SI, their stability partitions giving more ARI than the All-Points-Core-Distance kernel. We got the ARI of 0.9 with M2 and DBCV with the Epanechnikov kernel, which is more than the All-Points-Core-Distance kernel result (ARI close to 0.8) and the ARI of best DBCV and SI partitions from HDBSCAN* (Table 6.3).

Our aggregate plots do not tell us much about the individual datasets. The datasets in this section can be broadly divided into three categories:

1. Datasets with approximately the same density of each cluster – 5gaussians, 2gaussians, 2spirals;

2. Datasets in which some clusters are more dense than others – aggregation, compound, r15, jain, or some clusters seem to form a bigger cluster – d31 and r15, however they are still globular enough to be identified as different clusters at some density level. The pathbased dataset is another dataset where one cluster is less dense than the other two clusters;

3. Datasets where denseness of a cluster is concentrated at one end of it. The best example for this is spiral dataset. The three spirals have more points where they are close and less at the tails. The flame dataset, which comes from fuzzy clustering, also falls in this category as it is not so easy to separate.

We now analyze one representative dataset from each category and also present a study of the robustness of the bandwidth of HDBSCANk and m_{pts} for HDBSCAN*.

6.3.1.1 2spirals Dataset

The 2spirals dataset, Figure 6.2j, is a 1000 point dataset with 2 spirals that have uniform density along the arms of the spirals. Figure 6.7 shows the maximum ARI we can get using HDBSCAN*, HDBSCANk with the Normal

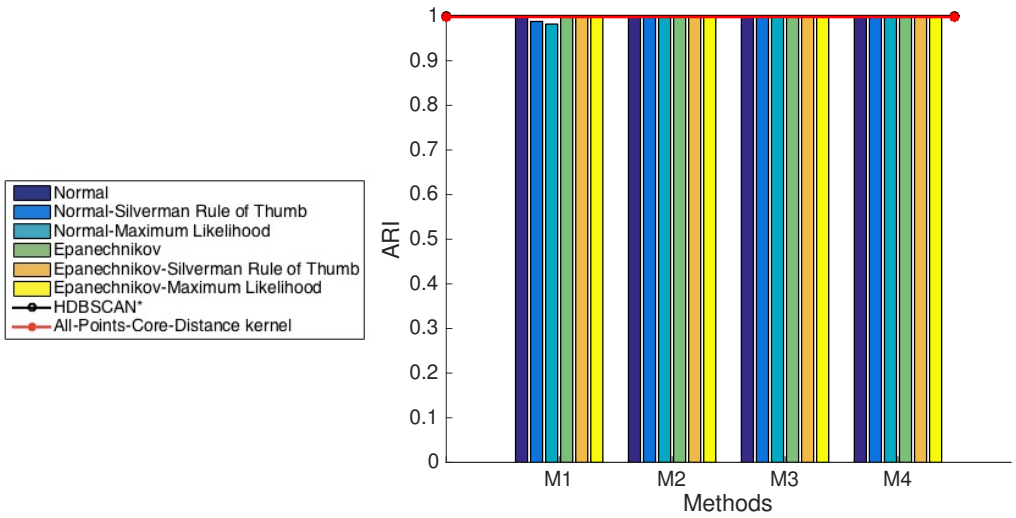


Figure 6.7: Maximum ARI that can be achieved on the 2spirals Dataset

and Epanechnikov kernels with different edge estimation methods and the All-Points-Core-Distance kernel. Irrespective of kernel, edge estimation method and algorithm used (HDBSCAN* or HDBSCANk), this dataset can be easily separated into two clusters. On evaluating all partitions from hierarchies of heuristic bandwidths, a partition with ARI value of 1 can be found.

With HDBSCAN*, irrespective of internal validation measure used, the ARI of 1 could be achieved from amongst the stability partitions. The All-Points-Core-Distance kernel could also produce the stability partition with the ARI value of 1.

As we observed in Figure 6.7, the Epanechnikov kernel is capable of finding the ground truth of this dataset when all partitions are evaluated. As shown in Figure 6.8, on comparing only the stability partitions with the ground truth, the ARI value of 1 is still achievable. Evaluating the range of bandwidths from $[0.001, 0.200]$ and selecting the partition with maximum SI did not give good ARI results. That is to be expected because SI is known to work better with globular clusters than arbitrary shaped clusters. Using DBCV for selection as well as the heuristic bandwidth estimators gave the ARI of 1.

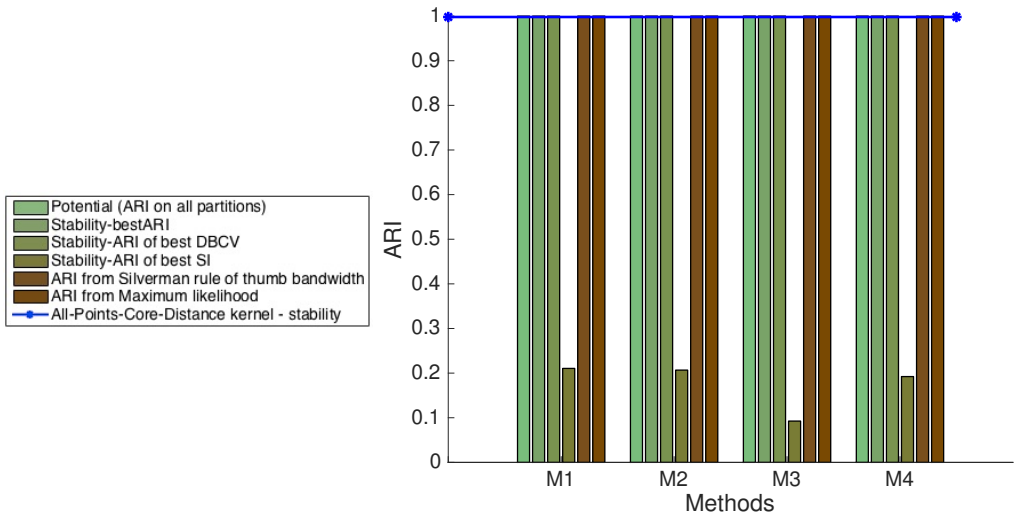


Figure 6.8: 2spirals Dataset : maximum ARI from stability partitions the Epanechnikov Kernel in HDBSCANk

Robustness of Bandwidth

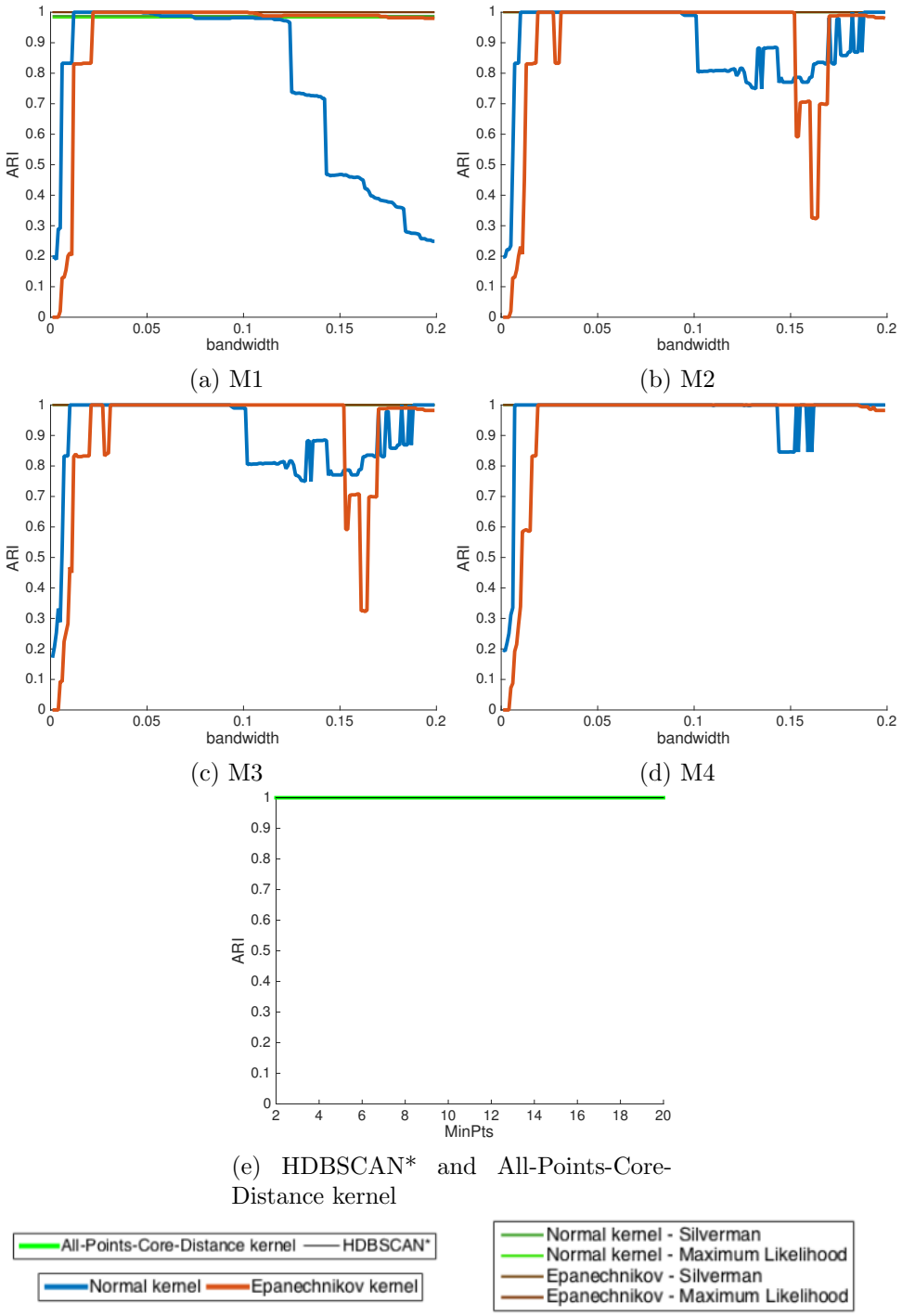
Figures 6.9a - 6.9d shows the change in ARI with change in bandwidth of HDBSCANk using the Normal and the Epanechnikov kernel. The Epanechnikov kernel showed less change in ARI with increasing bandwidth. Also, for M3, it was able to give ARI of 1 for all bandwidths after 0.021. The Normal kernel also had a number of bandwidths for which the ground truth could be identified. The heuristic bandwidths gave ARI of 1, irrespective of kernel and method.

Figure 6.9e shows change in ARI with change in m_{pts} for HDBSCAN*. Of the 19 m_{pts} tested, all gave an ARI of 1. As mentioned earlier, the All-Points-Core-Distance kernel also gives ARI of 1 with stability partition.

Thus, with both the Normal and the Epanechnikov kernel in HDBSCANk, we can perform as well as HDBSCAN* and HDBSCANk with All-Points-core-Distance kernel.

6.3.1.2 d31 Dataset

This dataset is composed of 31 Gaussian clusters of 100 points each, as shown in Figure 6.2k. Figure 6.10 shows the ARI we can obtain if we evaluated all possible partitions from every hierarchy. With HDBSCAN*, the maximum



(a) M1

(b) M2

(c) M3

(d) M4

(e) HDBSCAN* and All-Points-Core-Distance kernel

Figure 6.9: 2spirals dataset: Change in ARI as bandwidth of kernel of HDBSCANk and m_{pts} of HDBSCAN* is varied

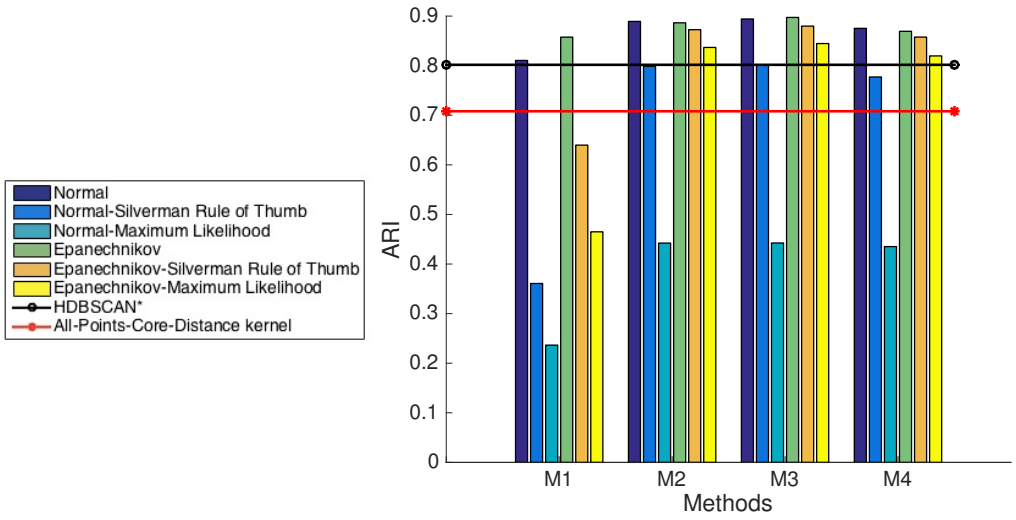


Figure 6.10: Maximum ARI that can be achieved on the d31 Dataset

achievable ARI was 0.8 while from the HDBSCANk with All-Points-Core-Distance kernel hierarchy, a partition with ARI of 0.7 could be extracted. The exhaustive search with the Normal and Epanechnikov kernels could produce partitions with maximum ARI of close to 0.9 with methods M2-M4. Evaluating all partitions from hierarchies built with the Normal kernel using these heuristic bandwidth estimators gave an ARI value that was smaller than the best ARI we can get by exhaustively searching for the bandwidth. The bandwidth estimators performed better with the Epanechnikov kernel, the best partition from the HDBSCANk hierarchy using Silverman's rule of thumb giving an ARI of almost

Using HDBSCAN* with internal validation measures for selecting the best stability partition for the hierarchies for different m_{pts}, the ARI of partition with maximum DBCV was 0.33 while with SI, ARI of 0.8 was obtained. The stability partition from All-Points-Core-Distance kernel gave ARI of 0.7. Now we compare these results with those from the stability partitions from the Epanechnikov kernel.

On evaluating all partitions from the hierarchies in bandwidth range [0.001 ,0.200], we found that the Epanechnikov kernel had the potential of giving ARI of close to 0.9 with M3 (from Figure 6.10). On evaluating only the stability partitions, the best ARI achievable is the same – about 0.87 with M3, as

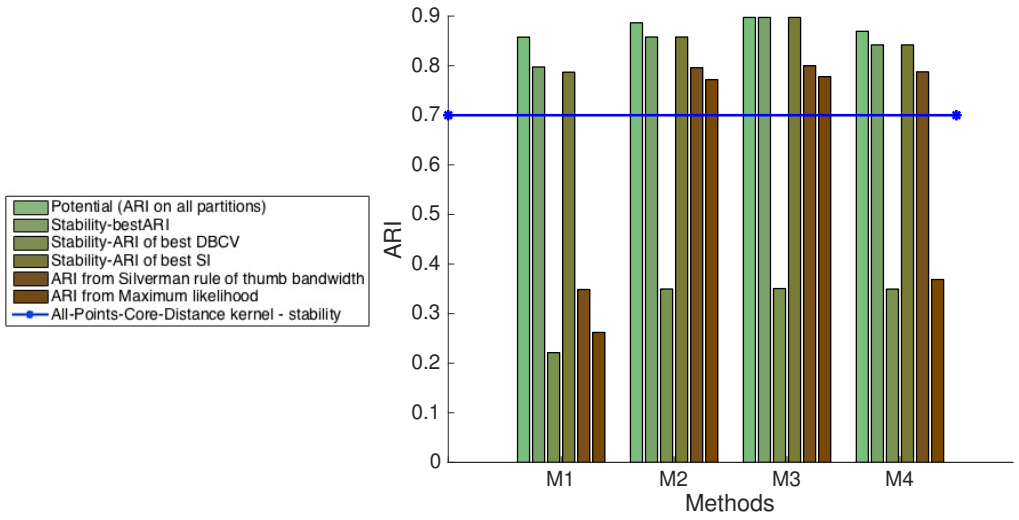


Figure 6.11: d31 Dataset : ARI using HDBSCANk with Epanechnikov Kernel

shown in Figure 6.11. All four edge estimation methods are capable of producing a stability partition with ARI of about 0.85. Using the stability partition with best SI gave better ARIs than optimizing the DBCV values here because DBCV prefers some of these clusters to be combined while SI prefers them to be separated. The ARI value obtained with SI and DBCV was 0.87 (with M3) and 0.35 (M2-M4), respectively. The stability partitions extracted from hierarchies constructed based on Silverman's Rule of thumb heuristic bandwidth estimator performed well, giving ARI value of approximately 0.8 with edge weight estimation methods M2-M4. On the other hand, cross validation using Maximum Likelihood also gave ARI close to 0.8 with methods M2 and M3.

Robustness of the bandwidths

HDBSCANk with the Epanechnikov kernels is capable of finding stability partitions with better ARI than HDBSCAN* or All-Points-Core-Distance kernel. Of the 200 bandwidths tested for the Epanechnikov kernels, there was only one 'best' bandwidth for every kernel-edge weight estimation combination. Thus, predicting the correct bandwidth-kernel-method combination is important to obtain good results. The heuristic bandwidth estimators with the Epanechnikov kernel with edge eight estimation methods M2-M4 will give stability

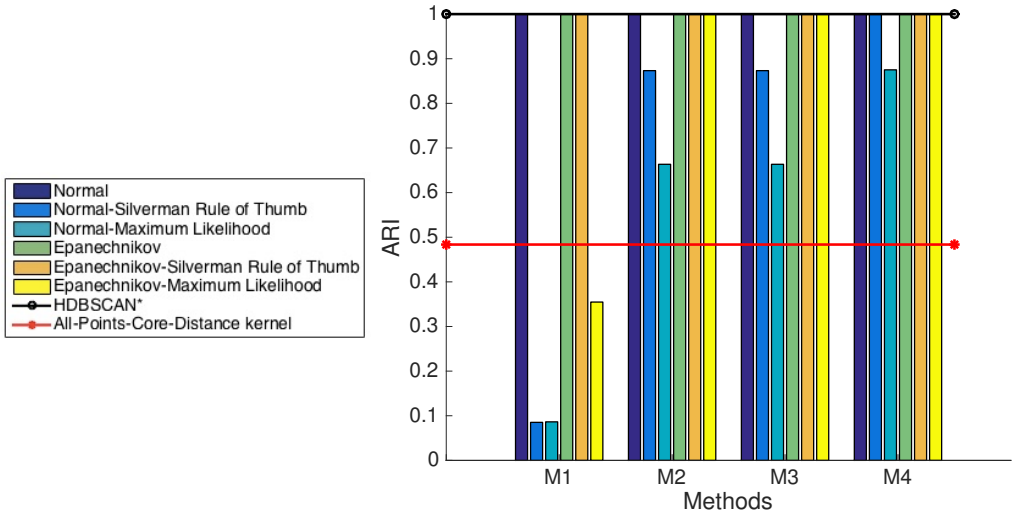


Figure 6.12: Maximum ARI that can be achieved on the spiral Dataset

partitions of close to the baseline algorithms: HDBSCAN* and HDBSCANk with the All-Points-Core-Distance kernel.

6.3.1.3 Spiral Dataset

The spiral dataset (Figure 6.2c) is similar to the 2spirals dataset, except that all three spirals are denser at the place where they are closest together and sparser at their tails. Figure 6.12 shows the maximum ARI we can get by evaluating all partitions for every algorithm. HDBSCAN* can identify the underlying structure with $m_{pts} = 2, 3, 4$. With the All-Points-Core-Distance kernel, we achieved an ARI of approximately 0.5. Both the Normal and the Epanechnikov kernels in HDBSCANk are capable of identifying the ground truth when giving the right parameters. The edge weight estimation methods, that used the top contributors, work with both kernels when conducting an exhaustive search for the best bandwidth. The heuristic bandwidths, especially Silverman's rule of thumb, was found capable of an ARI of 1 with Epanechnikov kernel, irrespective of the method for estimating the connectivity between objects. However, using them with the Normal kernel and M1 gave low ARIs of 0.1.

With HDBSCAN*, irrespective of interval validation measure used to select the best stability partition over the range of m_{pts} tested, an ARI of 1 could

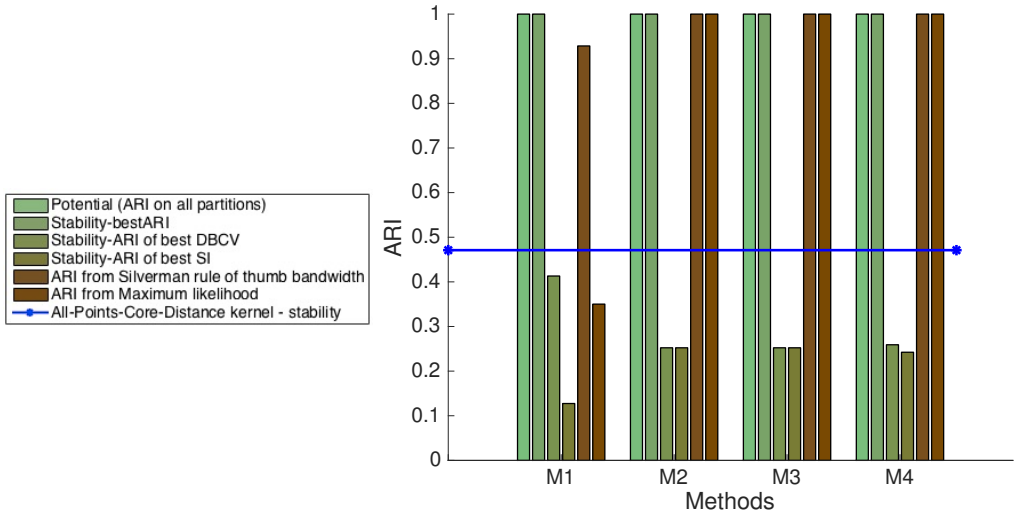


Figure 6.13: spiral Dataset : maximum ARI from stability partitions using the Epanechnikov Kernel

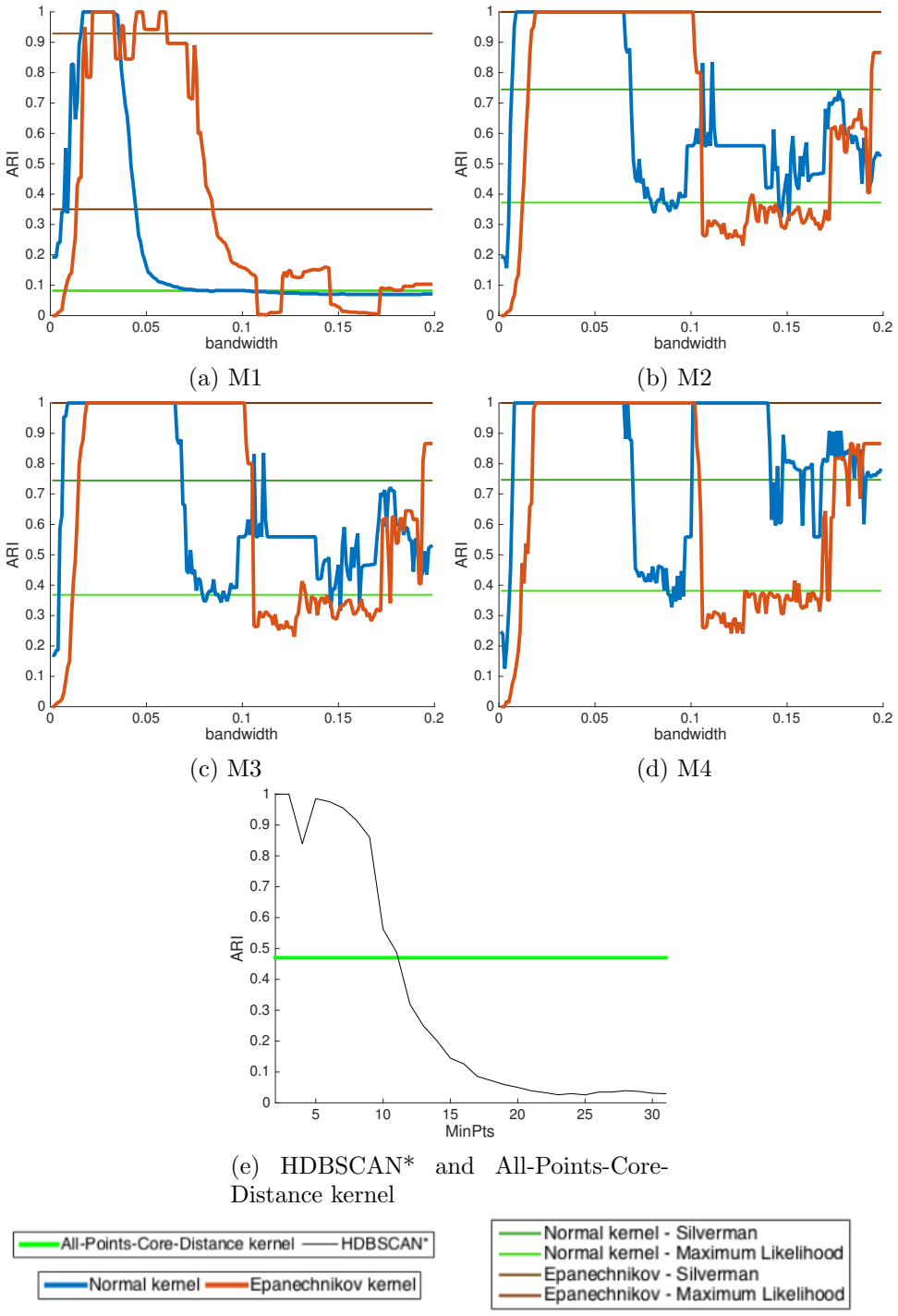
be achieved. The stability partition for All-Points-Core-Distance kernel gave ARI of 0.49.

The Epanechnikov kernel, on evaluating all partitions, was capable of ARI of 1 as shown in Figure 6.12. Figure 6.13 compares the different ARI values we get from the stability partitions of hierarchies built with Epanechnikov kernel. The maximum ARI of the stability partitions of the hierarchies for bandwidths [0.001,0.200] was 1. Thus, the stability partitions can identify the ground truth. Both internal validation measures, DBCV and SI, for selecting the best stability partition from the hierarchies built for the range of bandwidths did not produce partitions with high ARI values. However, the stability partitions from hierarchies built with Epanechnikov kernel and bandwidths estimated from Silverman's rule of thumb and Maximum Likelihood gave ARI values of 1 each with all top contributors edge estimation methods, i.e. M2-M4.

Thus, with the Epanechnikov kernel in HDBSCANk, we could get better results than the All-Points-Core-Distance kernel and HDBSCAN*.

Robustness of bandwidth

Figures 6.14a-6.14d show the change in ARI with change in bandwidth of HDBSCANk using the Normal and the Epanechnikov kernels. There are very few bandwidths with M1 whose stability partitions are capable of finding the



(a) M1

(b) M2

(c) M3

(d) M4

(e) HDBSCAN* and All-Points-Core-Distance kernel

Figure 6.14: spirals dataset: Change in ARI as bandwidth of kernel of HDBSCANk and m_{pts} of HDBSCAN* is varied

ground truth. On the other hand, with edge weight estimation methods M2-M4, smaller bandwidths in range 0.02 to 0.100 are capable of finding the underlying structure in the dataset. A wider range is available with the Epanechnikov kernel. The heuristic bandwidths gave ARI of 1 with the Epanechnikov kernel for methods M2-M4. With the Normal kernel and Silverman's rule of thumb, gave ARI of 0.75. When the stability partitions built from hierarchy of Maximum Likelihood heuristic bandwidth and the Normal kernel was compared with ground truth, ARI of approximately 0.4 was obtained. For HDBSCAN*, of the 30 m_{pts} evaluated, only two m_{pts} were found whose stability partition gave the ARI value of 1.

Thus, extracting stability partitions from hierarchies using Epanechnikov kernel has a wider range of bandwidths that pertain to the underlying ground truth as compared to the Normal kernel and HDBSCAN* m_{pts} parameter. Both the Normal and Epanechnikov kernels are capable of doing better than the All-Points-Core-Distance kernel integrated into HDBSCANk.

6.3.2 Results on High Dimensional Artificial Data sets

In this section, we test HDBSCANk and HDBSCAN* with higher dimensional datasets, varying the number of clusters and the dimensionality, summarized in Table 6.4. These datasets were generated using the ellipsoid generator by Handl and Knowles [27]. According to their description of the generator, there may be some degree of overlap between the ellipsoids. A minimum cluster size of 50 was used.

We first present the average best ARI we can get for the 16 datasets generated if we evaluated all partitions from every hierarchy. Next, we take a closer look at the 32 dimensional datasets and how the performance of the algorithms changed with different number of clusters. Next, for 12 cluster datasets, we

Parameter	Range
Dimensionality	4,12,32, 64
No. of Clusters	10, 15, 20

Table 6.4: Parameters for Ellipsoid Data Generation

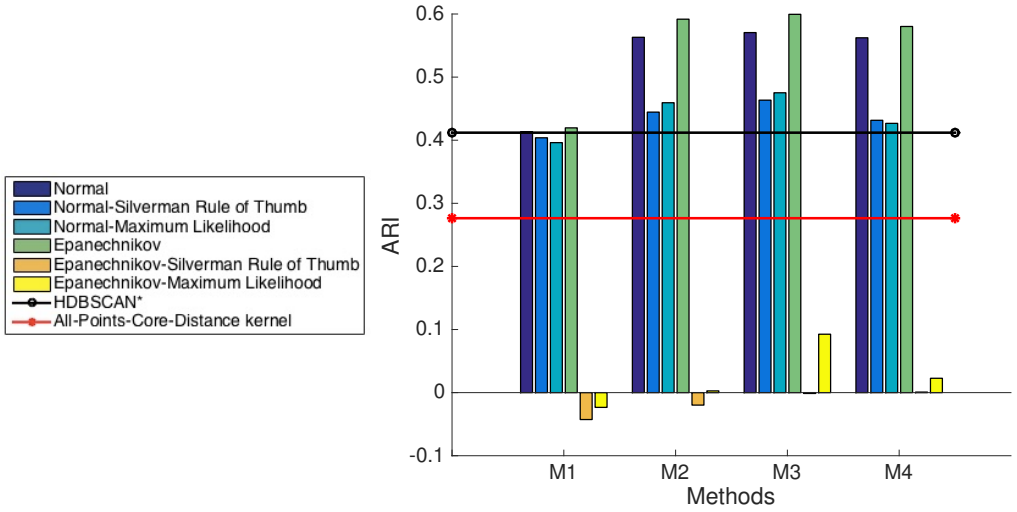


Figure 6.15: Maximum ARI that can be achieved on the High Dimensionality Datasets on an average

comment on their separability as dimensionality increased. Lastly, for the 32 dimensional dataset with 15 clusters, the robustness of the bandwidth of HDBSCANk and m_{pts} parameter of HDBSCAN* respectively are presented.

Potential to Identify Clusters in Higher Dimensions

Figure 6.15 shows the average best ARI that can be obtained by evaluating all possible partitions from hierarchies to identify and separate the clusters in higher dimensions. With HDBSCAN*, the highest achievable ARI was found to be approximately 0.4 on an average. HDBSCANk with the All-Points-Core-Distance kernel built a hierarchy whose best flat partition had an ARI of 0.28. Performing an exhaustive search for the bandwidth with the Normal kernel gave a partition with ARI value 0.55 with M3. The best partition from the Epanechnikov kernel was with edge weight estimation method M3 and ARI value of 0.58; method M2 giving ARI of 0.57 being the next best. While the heuristic bandwidths had performed well with the Epanechnikov kernel in two dimensional datasets, they did poorly in this setting this time. However, they performed better with the Normal kernel, giving ARI values of 0.47.

Thus, in terms of potential, an exhaustive search for the bandwidth with the Normal and the Epanechnikov kernels in HDBSCANk can give better results than HDBSCAN* and the All-Points-core-Distance kernel.

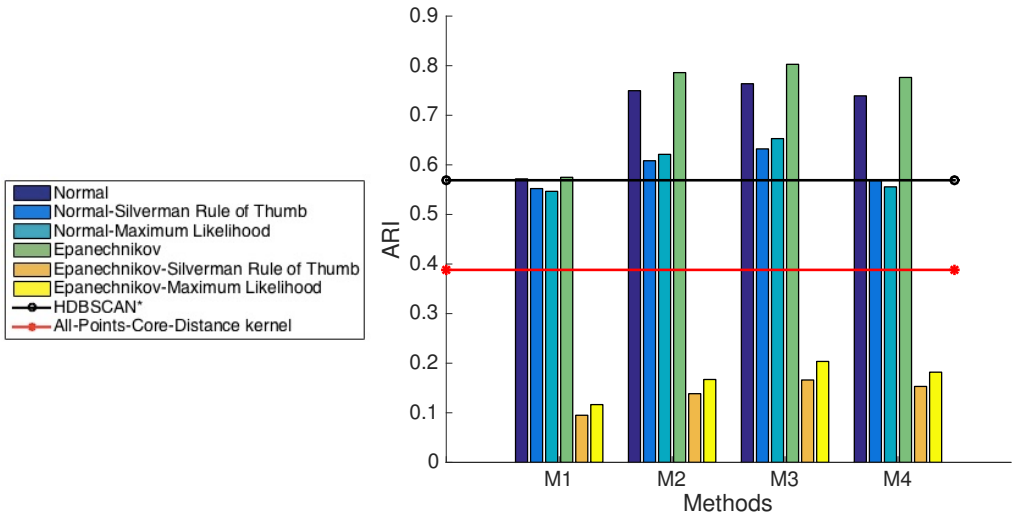


Figure 6.16: Maximum ARI that can be achieved on the three 32 Dimensionality Datasets on an average

Varying Number of Clusters

We present results for the 32-dimensional datasets with different numbers of clusters in the ground truth.

The bar plot in Figure 6.16 summarizes the results for the maximum ARI possible for these datasets if all partitions were extracted from every hierarchy. The maximum ARI from the All-Points-Core-Distance kernel was 0.4 on average for the three datasets while form the HDBSCAN* hierarchies a partition with the an ARI value of about 0.55. On an average, the Normal kernel results in an ARI of 0.75 with M2 and M3. The Epanechnikov kernel with exhaustive search for the best bandwidth could partially identify the ground truth clusters, giving an ARI value of approximately 0.80 with edge weight estimation methods M2 and M3. The heuristic bandwidths worked better with the Normal kernel than the Epanechnikov kernel. They gave better partitions than those obtained from HDBSCAN* hierarchy for edge weight estimation methods M2 and M3 with the Normal kernel.

HDBSCAN*, on evaluation of only the stability partitions gave ARI value of 0.45 when DBCV was used for internal validation and 0.55 with SI. The All-Points-Core-Distance kernel integrated into HDBSCANk, on an average for the 32 dimensional datasets, produced stability partitions with ARI of

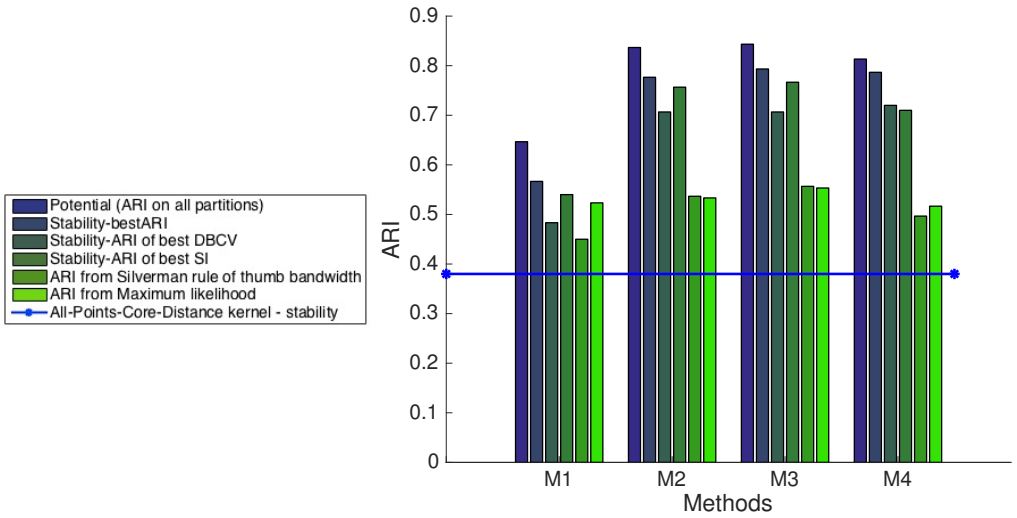


Figure 6.17: Average Maximum ARI from Stability Partition for 32 dimension Datasets

approximately 0.4.

The potential is what we could get if we evaluated all partitions against the available ground truth. For the Normal kernel, Figure 6.17, it was approximately 0.75 with M3. On evaluating only the stability partitions, HDBSCANk with this kernel was capable of getting an ARI value of 0.79 with edge weight estimation method M3. Using DBCV for selecting the best stability partition from the hierarchies of range of bandwidth tested gave ARI of 0.72 with M4 and SI produced a stability partition of ARI value 0.76 with M3. Using heuristic bandwidth estimators did not give as good results but the average from All-Points-core-Distance kernel could be improved to 0.55 with M3 using either bandwidths.

Figure 6.18 compares the best ARI that can be obtained on evaluating all stability partitions from hierarchies obtained with the Normal and the Epanechnikov kernels, and using M3. Also compared are the ARI for stability partition from the All-Points-Core-Distance kernel hierarchy and the best ARI that can be obtained amongst all stability partitions of HDBSCAN* for all possible m_{pts}. We found that as the number of clusters increases, it becomes harder to find the ground truth. The decrease in maximum ARI from the Normal and the Epanechnikov kernels with change in number of clusters is slower than the drop in ARI values for HDBSCAN* and the All-Points-Core-

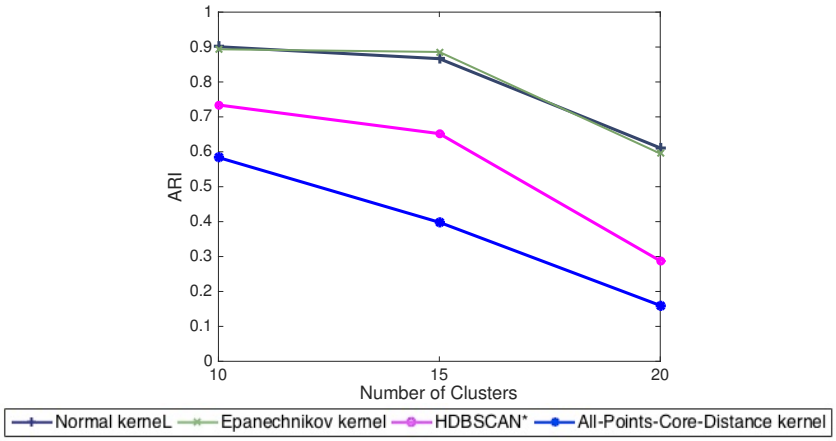


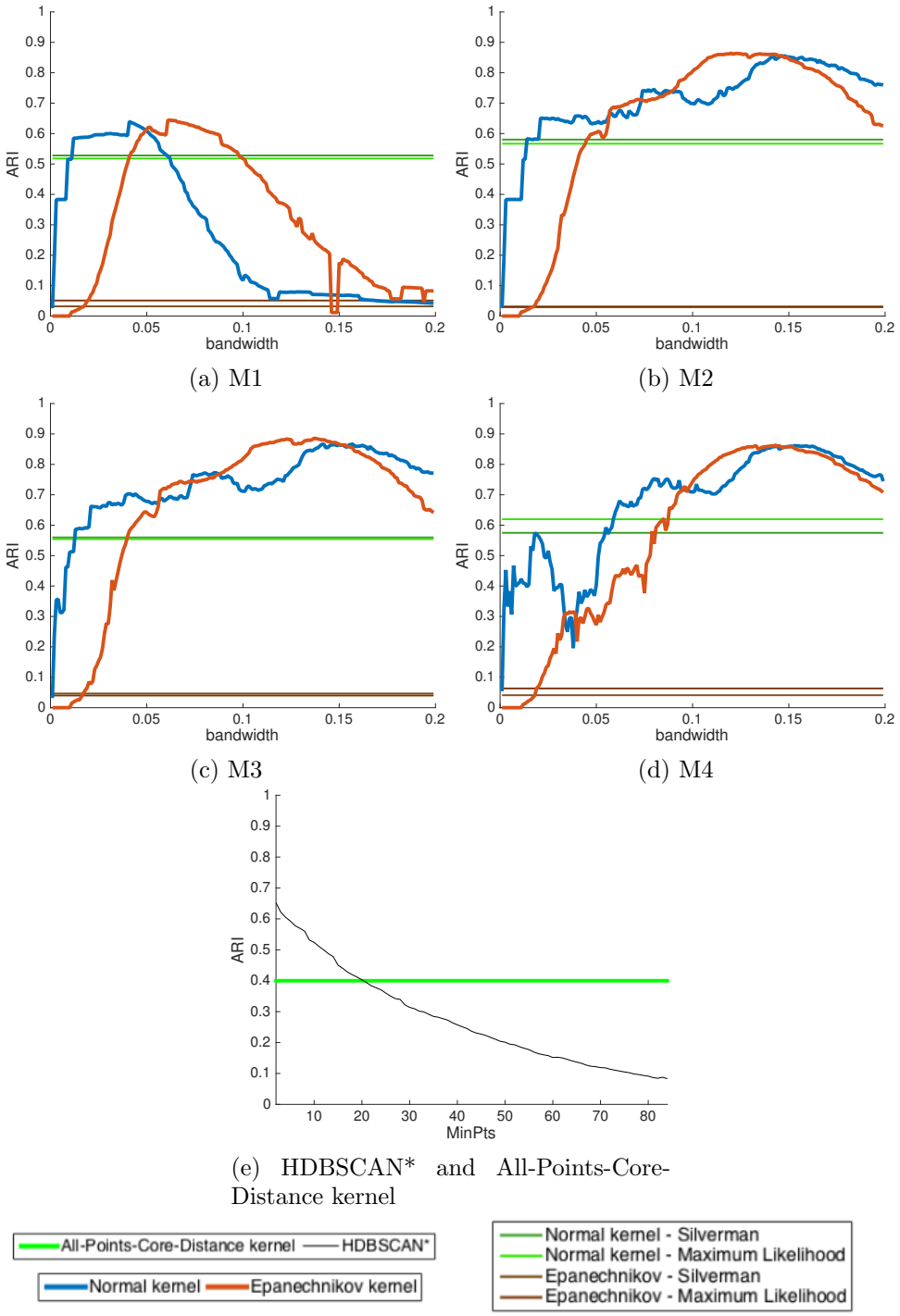
Figure 6.18: Change in maximum ARI from Stability Partitions by varying number of clusters for 32 dimensional Datasets: For HDBSCANk with the Normal and the Epanechnikov kernels, ARI for M3 are shown.

Distance kernel. However, there was only one value of the m_{pts} and *bandwidth* that gave the presented value of ARI for each setting.

In summary, for the 32 dimensions, using HDBSCANk, we can improve upon the results from HDBSCAN* and the All-Points-Core-Distance kernel if we perform an exhaustive search for bandwidth or use one of the heuristics for estimating the bandwidth and using them with the Normal kernel. Any of the edge estimation methods with top contributors can be used.

Robustness of a bandwidth for a high dimensional dataset

For the 32 dimensional dataset with 15 clusters, Figures 6.19a-6.19d depict the change in ARI value as bandwidth changes for the edge estimation methods. All edge weight estimation methods except M1 produce stability partitions with high ARI values for both kernels. The heuristic bandwidths give an ARI of approximately 0.5 with the normal kernel and 0 with the Epanechnikov kernel. Figure 6.19e shows the variation in ARI value for different m_pts. We see that the maximum achievable ARI for this dataset with HDBSCAN* is about 0.65 and this can only be achieved with $m_{pts} = 2$. The All-Points-core-Distance kernel gives a small ARI value of 0.4. Thus, looking at these figures, we conclude that if any bandwidth in the range [0.01,0.15] is used with HDBSCANk with the Normal or the Epanechnikov kernel and edge weight es-



(a) M1

(b) M2

(c) M3

(d) M4

(e) HDBSCAN* and All-Points-Core-Distance kernel

Figure 6.19: 32 dimensional dataset with 15 clusters: Change in ARI as bandwidth of kernel of HDBSCANk and m_{pts} of HDBSCAN* is varied

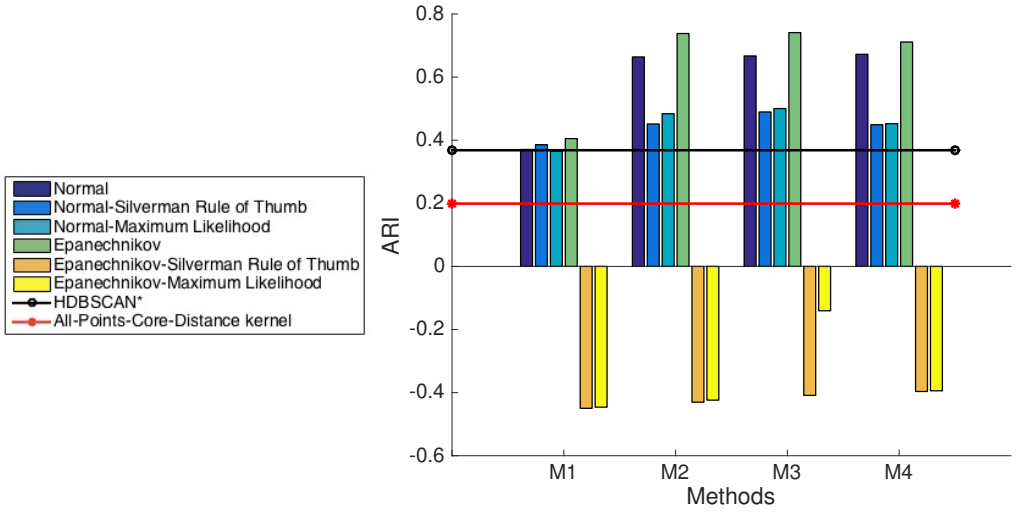


Figure 6.20: Maximum ARI that can be achieved on the High Dimensionality Datasets with 20 clusters

timation methods M2-M4, the stability partition obtained would have an ARI value higher than what HDBSCAN* or the All-Points-core-Distance kernel integrated into HDBSCANk cam give on comparison with the ground truth.

Varying Dimensionality for 20 clusters

Figure 6.20 elaborates on the performance of the algorithms on datasets with 20 clusters when varying the dimensionality. HDBSCAN* and the All-Points-Core-Distance kernel, on average, did not perform well (maximum ARI from partitions was 0.4 and 0.2 respectively). The Normal kernel had the maximum potential of an ARI value of 0.7 with edge weight estimation methods M2-M3. For the Epanechnikov kernel, checking all bandwidths in the predefined range, a maximum ARI of about 0.75 with edge weight estimation methods, M2 and M3, can be achieved. The heuristic bandwidths with the Normal kernel produced partitions that had better ARI values than both HDBSCAN* and HDBSCANk with All-Points-Core-Distance kernel. With the Epanechnikov kernel, on the other hand, the partitions did not confirm to the ground truth, giving ARI values in negative.

For edge estimation method M3, Figure 6.21 shows the change in ARI that can be obtained for the datasets with 20 clusters. With the Normal kernel, there is a drop in ARI from 0.6 in 32 dimensions to 0.45 in 64 dimensions while

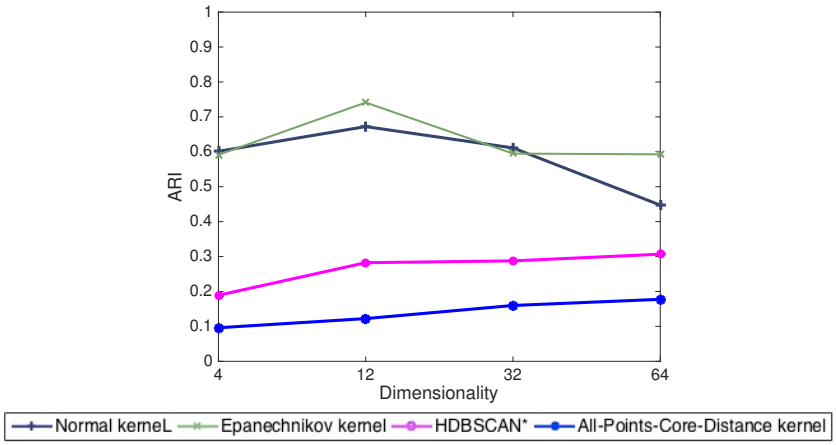


Figure 6.21: Change in maximum ARI from Stability Partitions by varying number of clusters for 32 dimensional Datasets. For HDBSCANk with the Normal and the Epanechnikov kernels, ARI for M3 are shown.

Dataset	No. of objects	Dimensions	No. of clusters
yeast	1484	7	10
iris	150	4	3
seeds	210	7	3
wine	178	13	3
glass	214	9	6
ecoli	151	5	3
leaf	340	16	3

Table 6.5: UCI Data sets (only numeric attributes)

the Epanechnikov kernel remains almost the same in ARI for the two higher dimensions. Interestingly, the ARI of the stability partition from the All-Points-Core-Distance kernel steadily increases with increase in dimensionality. HDBSCAN* exhibits a similar behavior. This may be due to the way the data is generated by Handl and Knowles's [27] data generator. As dimensionality increases and the number of clusters increases, the data becomes more and more well separated.

6.3.3 Results on UCI Data Sets

We used seven datasets from the UCI repository [43] and they are summarized in Table 6.5. We present the average results over all datasets first, and then elaborate on the dataset *iris*, as well the dataset *leaf*. A minimum cluster size of 10 was used.

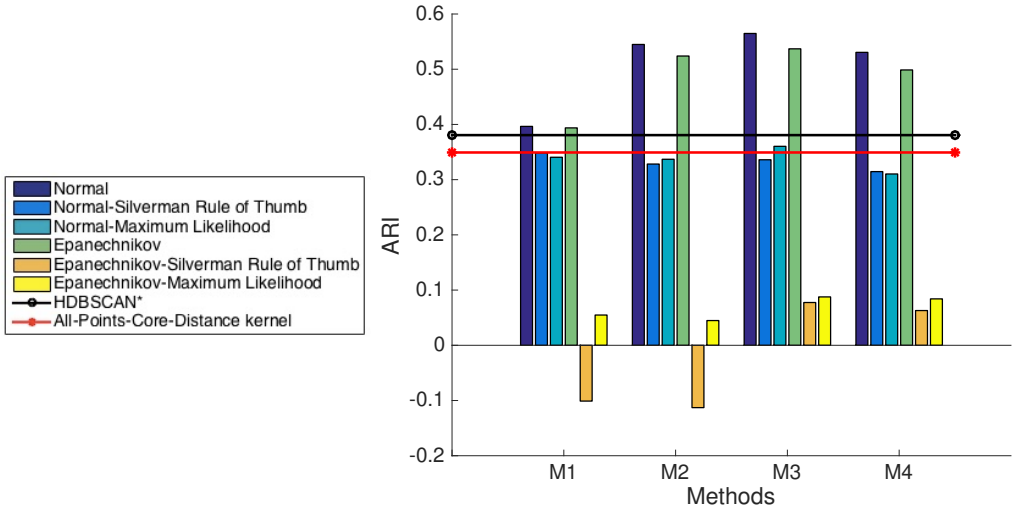


Figure 6.22: Maximum ARI that can be achieved on UCI Datasets over all datasets

Looking at Figure 6.22, we can see that HDBSCAN* and the All-Points-Core-Distance kernel showed consistent behavior as last section, giving ARI values of 0.4 and 0.3, on average. An average maximum ARI (over all 7 datasets) of about 0.55 using the Normal kernel and M2 could be obtained. Similar to the high-dimensional datasets in the previous section, the bandwidths obtained from heuristic bandwidth estimators did not perform well with the Epanechnikov kernel.

HDBSCAN*, on an average for the datasets with 20 clusters produced stability partitions with ARI 0.26 when DBCV was used for internal validation and 0.29 when the stability partition with highest SI value was reported. The stability partitions from All-Points-Core-Distance kernel gave an ARI of 0.27.

Figure 6.23 presents the results for the Epanechnikov kernel and the ARI of stability partitions obtained in combination with different edge estimation techniques and heuristic bandwidths. The best stability partition was never as accurate as the partition that could be obtained by evaluating all possible partitions. The partition with the maximum SI was as good as a random partition, on an average, with the ARI value of 0.3. Since UCI datasets ranged in dimensionality from 4-16, we start to see that the heuristic bandwidth already perform worse than the interval validation measures here. Both DBCV and SI produce partitions with ARI close to 0.3-0.4, the best being 0.37 with

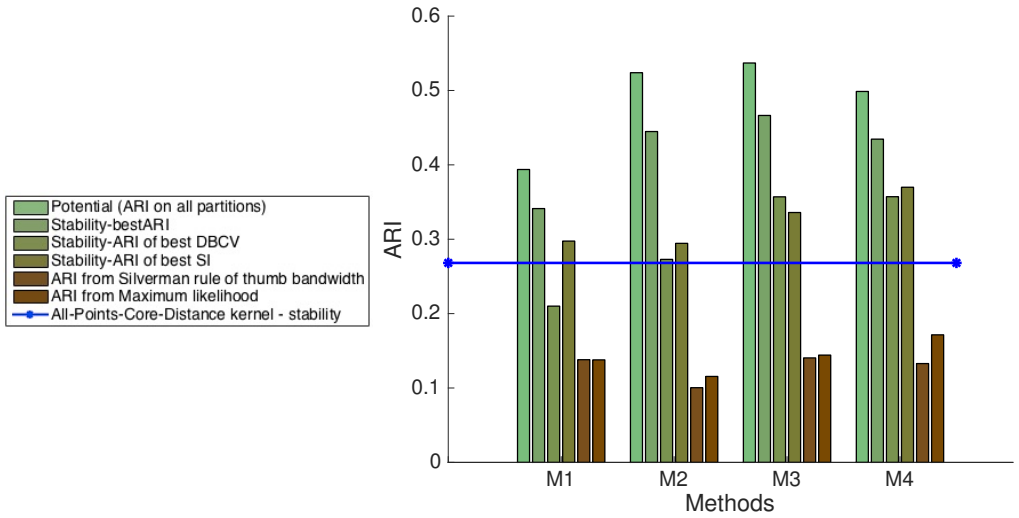


Figure 6.23: Maximum ARI that can be achieved on UCI Datasets using the Epanechnikov kernel in HDBSCANk on an average over all datasets

SI and M4.

Iris

The iris dataset consists of 4 dimensions describing 3 kinds of flowers – *iris setosa*, *iris virginica* and *iris versicolor* – with the two overlapping classes. Figure 6.24 summarizes the ARI values we would get if we had the resources and the ground truth to exhaustively search for the partition with the maximum ARI. HDBSCAN* can achieve an ARI value of 0.71; the All-Points-Core-Distance kernel can achieve an ARI of 0.7 and using M2, M3 or M4, ARI values of almost 0.85 with the Normal as well as the Epanechnikov kernels can be achieved. The best partitions from hierarchies using heuristic bandwidths have an ARI value of approximately 0.6.

HDBSCAN*, irrespective of which internal validation measuer was optimized to choose the best stability partition, an ARI value of 0.57 was always achieved for this dataset. HDBSCANk with the All-Points-Core-Distance kernel also resulted in a stability partition of an ARI of 0.56.

For the Normal kernel and evaluation of all stability partitions, see Figure 6.25, an ARI of 0.75 with M4 when exhaustively looking for the bandwidth can be achieved. The bandwidth estimated with Silverman's rule of thumb and Maximum Likelihood can result in partitions with an ARI close to 0.56.

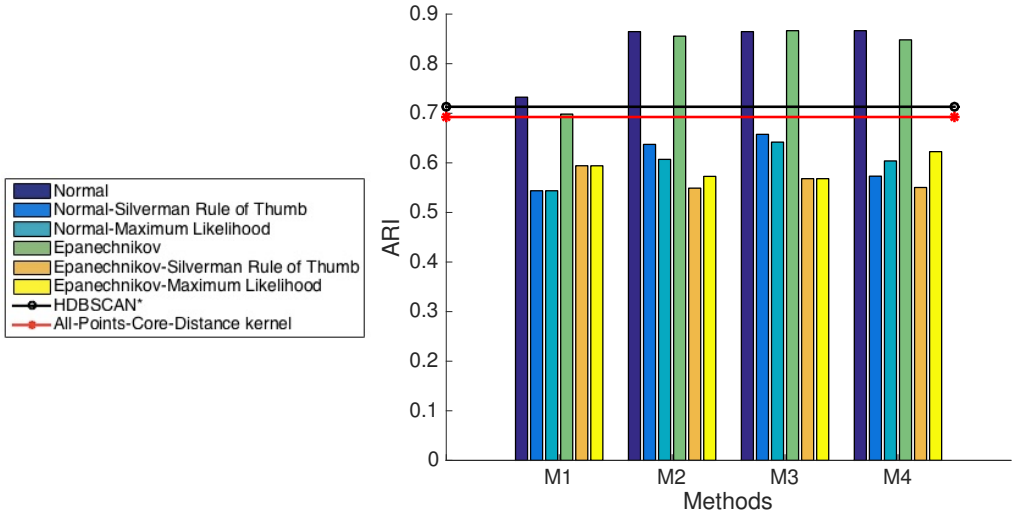


Figure 6.24: Maximum ARI that can be achieved on *iris* Dataset

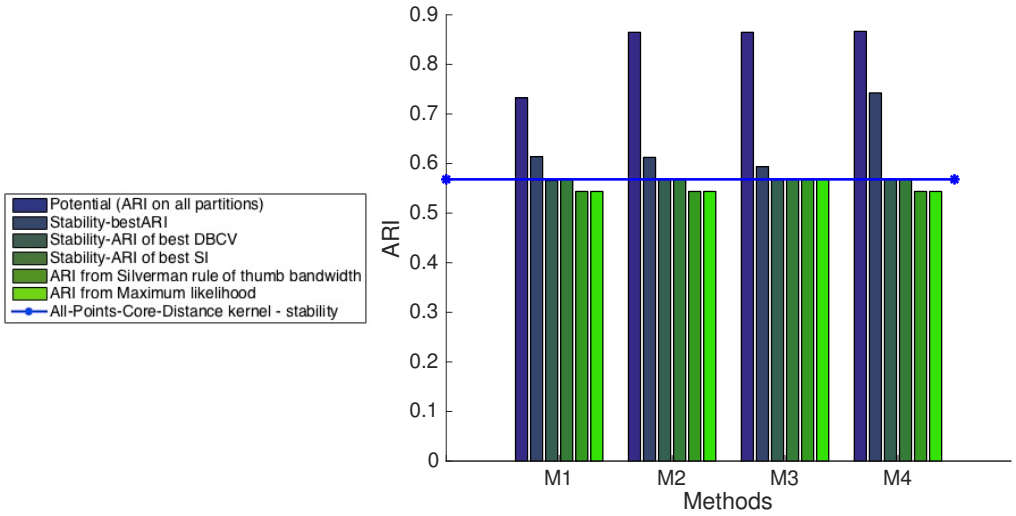
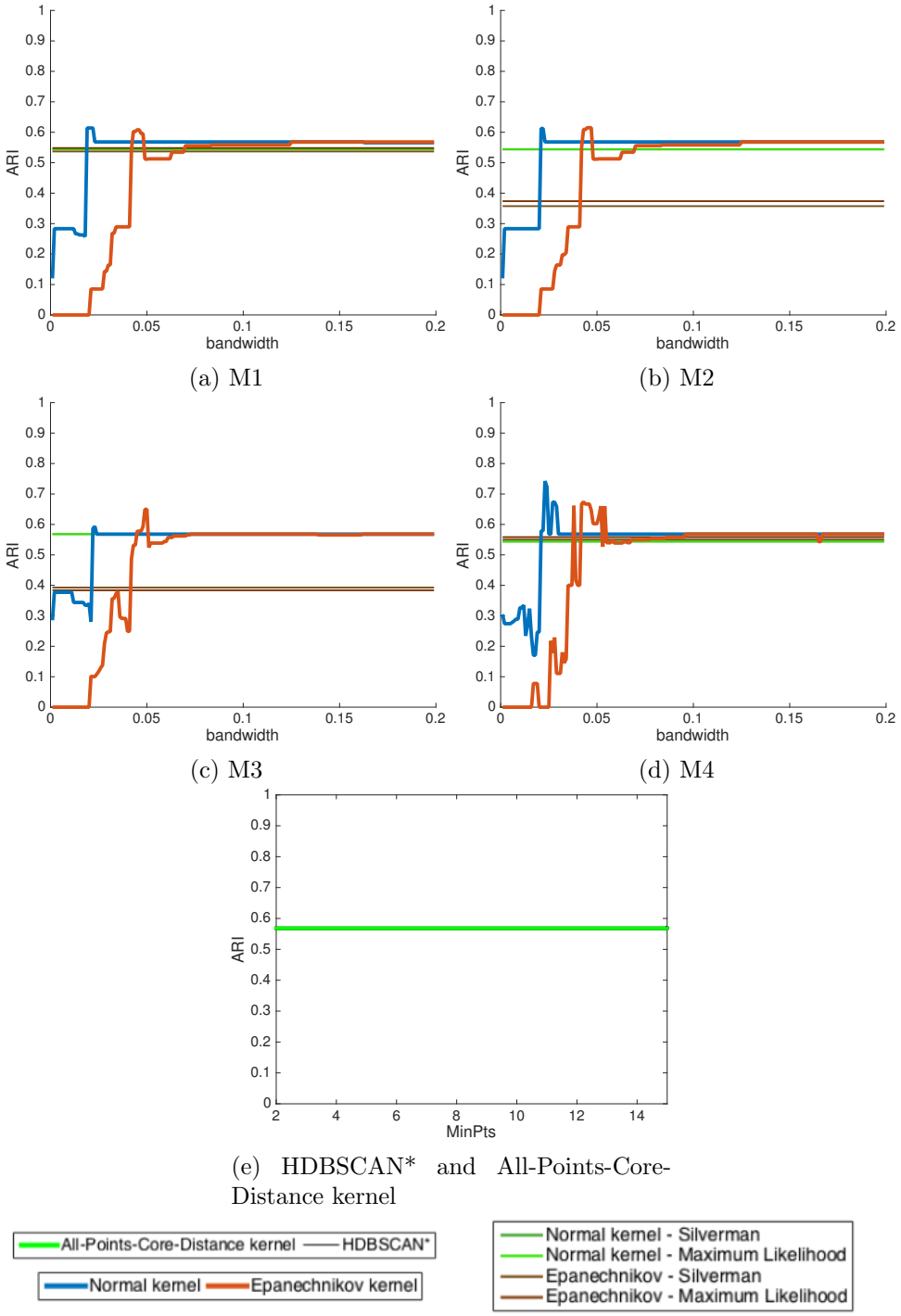


Figure 6.25: Maximum ARI that can be obtained from Stability Partitions using HDBSCANk with the Normal kernel - *iris* Dataset

Robustness of Bandwidth for *iris*

Figure 6.26 shows the change in ARI of stability partitions obtained form different bandwidths in HDBSCANk and m_{pts} in HDBSCAN*. The All-Points-Core-Distance kernel in HDBSCANk and HDBSCAN* gave the ARI values of 0.56. HDBSCANk with the Normal and Epanechnikov kernels, on the other hand, was capable of getting ARI of 0.75 and 0.7, which are higher than the ARI from both HDBSCAN* and the All-Points-Core-Distance kernel. The ARI value of 0.56, though, is consistently achieved by all.



(a) M1

(b) M2

(c) M3

(d) M4

(e) HDBSCAN* and All-Points-Core-Distance kernel

Figure 6.26: iris dataset: Change in ARI as bandwidth of kernel of HDBSCANk and m_{pts} of HDBSCAN* is varied

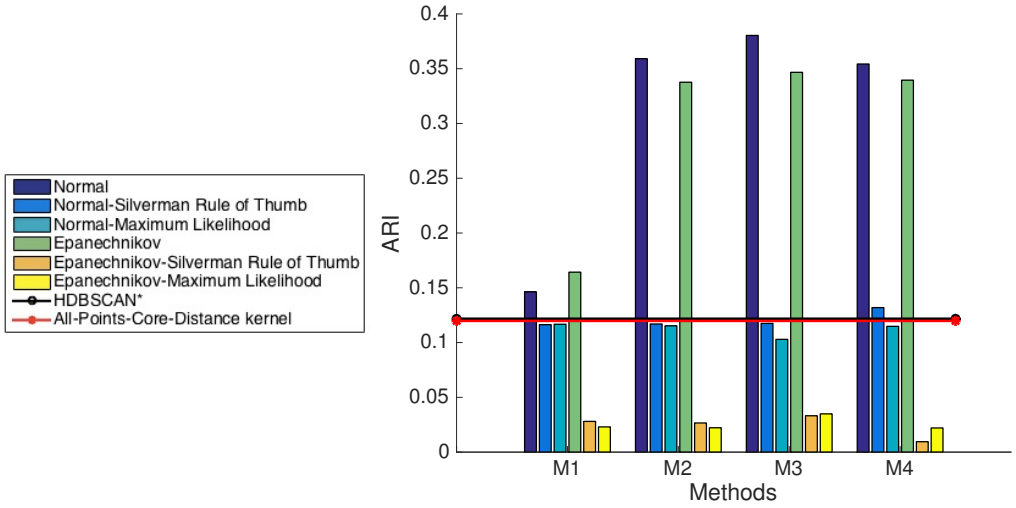


Figure 6.27: Maximum ARI that can be achieved on *leaf* Dataset

Leaf

The leaf dataset has 16 dimensions and 340 objects with 3 clusters. Separating the classes is a challenge. Figure 6.27 summarizes the results of potential of the techniques. The All-Points-Core-Distance kernel and HDBSCAN* gave the same highest achievable ARI of 0.12. The Normal kernel with M3 was able to obtain an ARI value of 0.37 and no algorithm could get a higher ARI than this. The heuristic bandwidths have an ARI of zero with the Epanechnikov kernel and an ARI of 0.1 with the Normal kernel.

With stability extraction type, the best ARIs obtained were even worse, reaching a high of at most 0.3. The All-Points-Core-Distance kernel gave ARI of 0.013 while partitions selected by optimizing both DBCV and SI failed to improve the ARI. The stability partitions from hierarchies using heuristic bandwidth estimators were better with the Normal kernel and came to close to the maximum ARI possible. Hence, we omit those charts here.

6.3.4 Results on Synthetic Hierarchical Data Sets

The main purpose for proposing the data generator in Chapter 5 was to be able to evaluate hierarchies produced by hierarchal clustering algorithms against an underlying hierarchical ground truth. The data generator has a number of parameters. In this section, we conducted three different studies, varying

Parameters	Values
Number of Points	1000
Minimum Cluster Size	30
Retention Factor	0.10
Dimensionality	2, **5**, 10, 20
Branching Factor	3, **5**, 7, 10
Separation Factor	2.5, **3**, 3.5, 4

Table 6.6: Parameter Settings for Hierarchical Datasets. The bold numbers represent the default values.

- the dimensionality of the data,
- the separation between the clusters (determined by $separationFactor$),
- the maximum number of children a cluster could have (controlled by $branchingFactor$),

Table 6.6 lists the values that were varied when generating datasets from the data generator with hierarchical ground truth.

In all experiments, we compared the hierarchies obtained with the different algorithms with the "ground truth hierarchy" as constructed by the generator using HAI [36], as defined in Chapter 5.

6.3.4.1 Varying Dimensionality of Data

On an average over all datasets generated, see Figure 6.28, we get at an ARI value of at least 0.78 irrespective of which method or kernel we use. HDBSCAN* and the All-Points-Core-Distance kernel also produce hierarchies with HAI of approximately 0.8. The Normal kernel with exhaustive search and edge weight estimation methods M2, M3 and M4 is able to identify hierarchies that have HAI of approximately 0.85 when compared with the underlying ground truth of the data structure. The Epanechnikov kernel with edge weight estimation method M2 did slightly better than all other kernel-method combinations. The heuristic bandwidth estimators do not outperform this HAI.

Figure 6.29 shows the change in value of HAI for each method. It is inferred that the performance of HDBSCAN* and All-Points-Core-Distance ker-

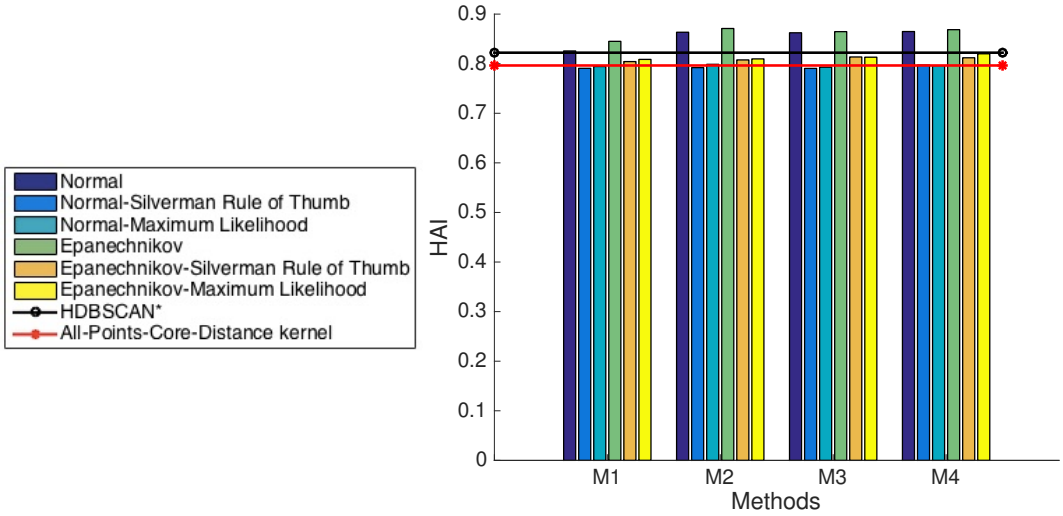


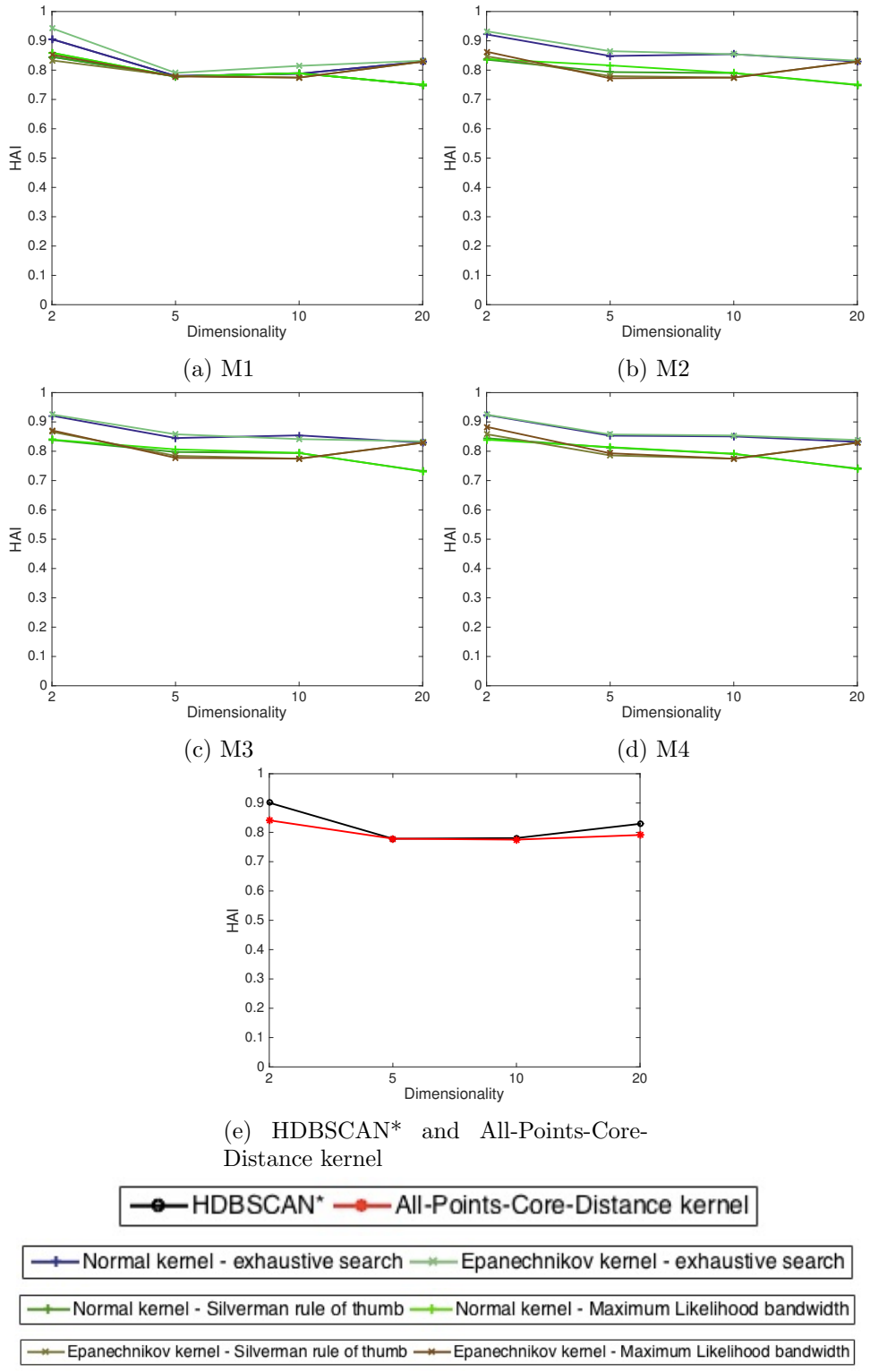
Figure 6.28: Average maximum HAI over all dataset for variable dimensionality of data

nel did not change much with dimensionality while the exhaustive search for bandwidth with the Normal and the Epanechnikov kernels started to produce hierarchies that had lower HAI values as dimensionality increased. This is exactly the observation we had when we compared the flat partitions from these algorithms in high dimensional datasets. The bandwidth calculated using Maximum Likelihood has the minimum ARI amongst all techniques when used in combination with the Normal kernel. With the Epanechnikov kernel, it started to perform better with increase in dimensionality from 10 to 20 for all edge weight estimation methods.

6.3.4.2 Varying Separation Between Clusters

The separation factor in the data generator defines the spread of a cluster as a function of its distance from its nearest cluster. Larger values of this factor, lead to a smaller spread of a cluster and more separated clusters. We now study how HDBSCANk is able to extract the hierarchy from the dataset as the clusters are more or less separated.

As the separation between clusters became more pronounced, it should be easier to identify clusters and subclusters, we explained in Section 5.5.3.2. This behavior is shown by all algorithms. In Figure 6.30, we show the increasing value of HAI for different edge weight estimation techniques for the Normal



(a) M1

(b) M2

(c) M3

(d) M4

(e) HDBSCAN* and All-Points-Core-Distance kernel

Figure 6.29: Trend of maximum HAI as Dimensionality increases for the Normal and the Epanechnikov kernels, HDBSCAN* and All-Points-Core-Distance kernel

and the Epanechnikov kernels, HDBSCAN* and the All-Points-Core-Distance kernel. The hierarchies showed an improvement in HAI value as separation factor increased. Performing an exhaustive search for the best bandwidth with Epanechnikov kernel and the Normal kernel produced hierarchies with HAI value over 0.9 using edge estimation methods M2, M3 and M4. The HAI from hierarchy built with Silverman's rule of thumb saw a drop in HAI value with increasing separation factor with the Epanechnikov kernel, however, the HAI value of approximately 0.8 could still be achieved. The bandwidth estimated using Maximum Likelihood heuristic performed better with the Normal kernel.

6.3.4.3 Varying Branching Factor

In this last experiment, see Figure 6.31, we varied the number of child clusters that a cluster could have. The behavior of the algorithms was consistent with the last two experiments – HDBSCAN* and the All-Points-Core-Distance kernel both produced hierarchies with the HAI of about 0.83. We were able to obtain an HAI value of approximately 0.85 on average; the Normal kernel did the best with edge estimation methods M2, M3 and M4, using an exhaustive search for bandwidth over all possible partitions; similar evaluations with the Epanechnikov kernel produced an hierarchy that had HAI value of approximately 0.9 with M2. Of heuristic bandwidth estimators, cross validation with Maximum Likelihood gave hierarchies with HAI of 0.81 with the Normal kernel and 0.83 with the Epanechnikov kernel. Silverman's rule of thumb bandwidth performed better with the Normal kernel, giving hierarchies of HAI 0.83 with M1 and with the Epanechnikov kernel, approximately 0.8 with all methods.

Robustness of Bandwidth for an hierarchical dataset

Figure 6.32 shows the effect of change of bandwidth of HDBSCANk and m_{pts} of HDBSCAN* on the hierarchical structure of the dataset. There are values in Figures 6.32a, 6.32b and 6.32d where there is a peak that shows the maximum value of HAI that was obatined. However, in general, for both HDBSCANk (irrespective of the kernel and edge weight estimation method) and HDBSCAN*

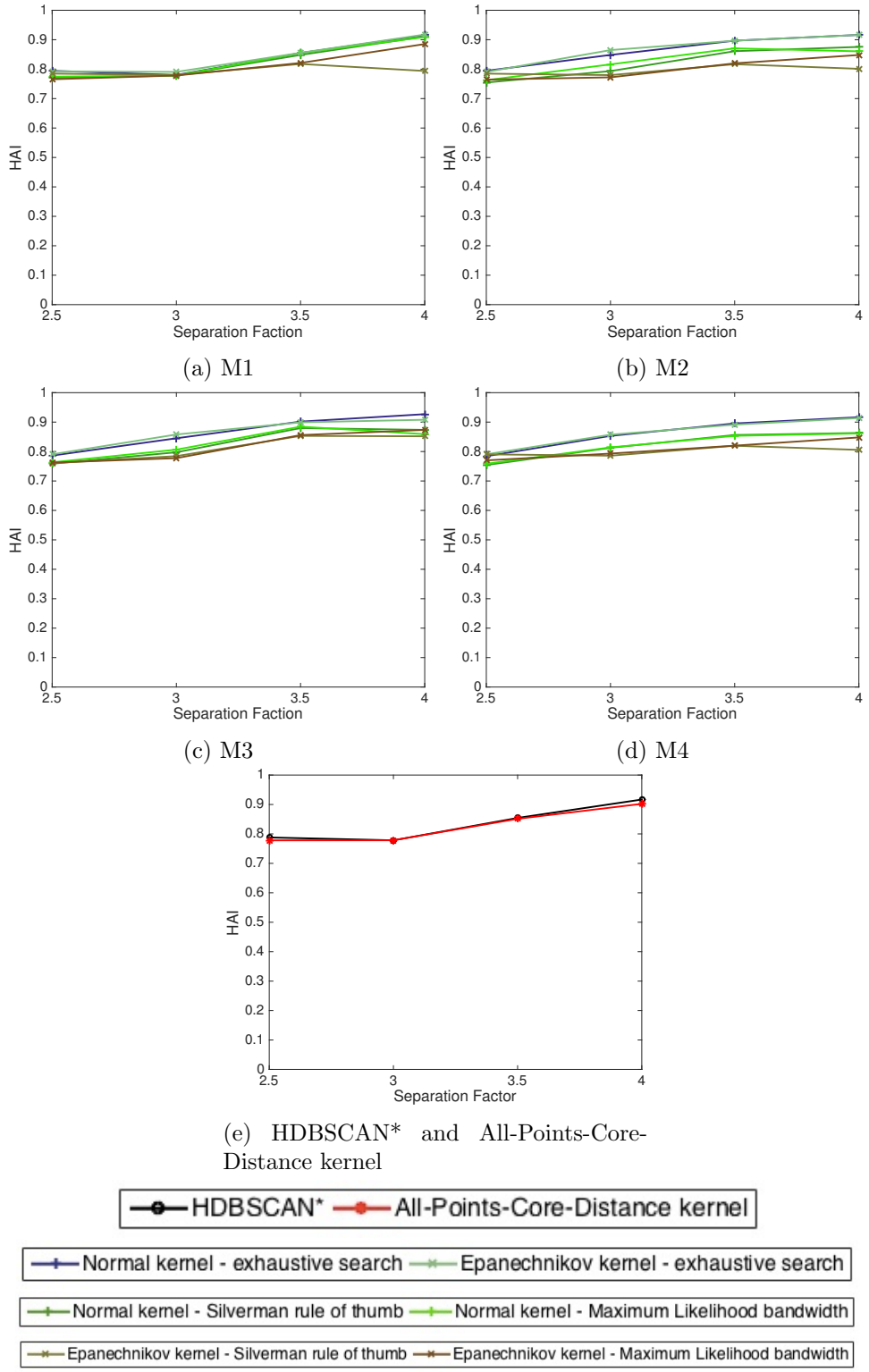


Figure 6.30: Trend of maximum HAI as Separation factor increases for the Normal and the Epanechnikov kernels, HDBSCAN* and All-Points-Core-Distance kernel

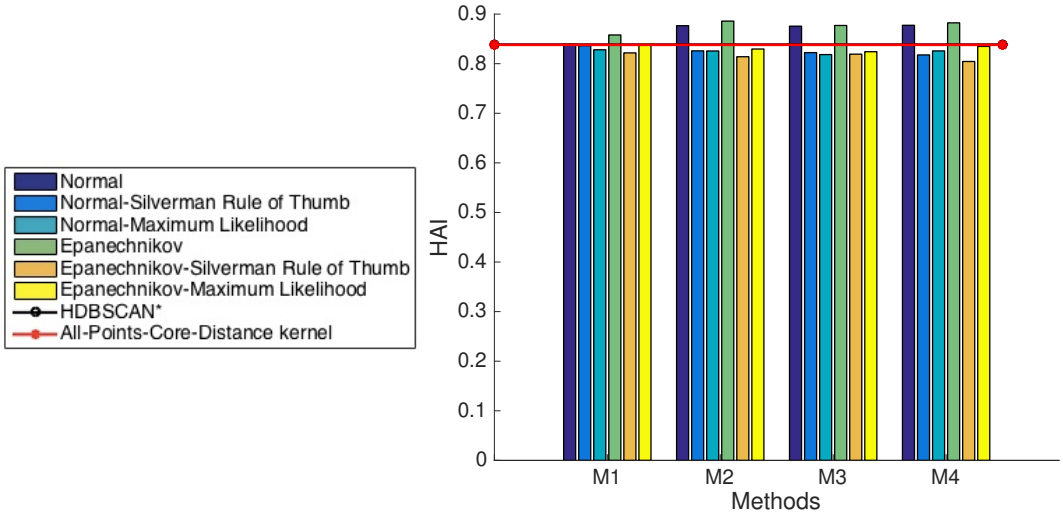


Figure 6.31: Average maximum over all dataset HAI for variable number of child clusters of data

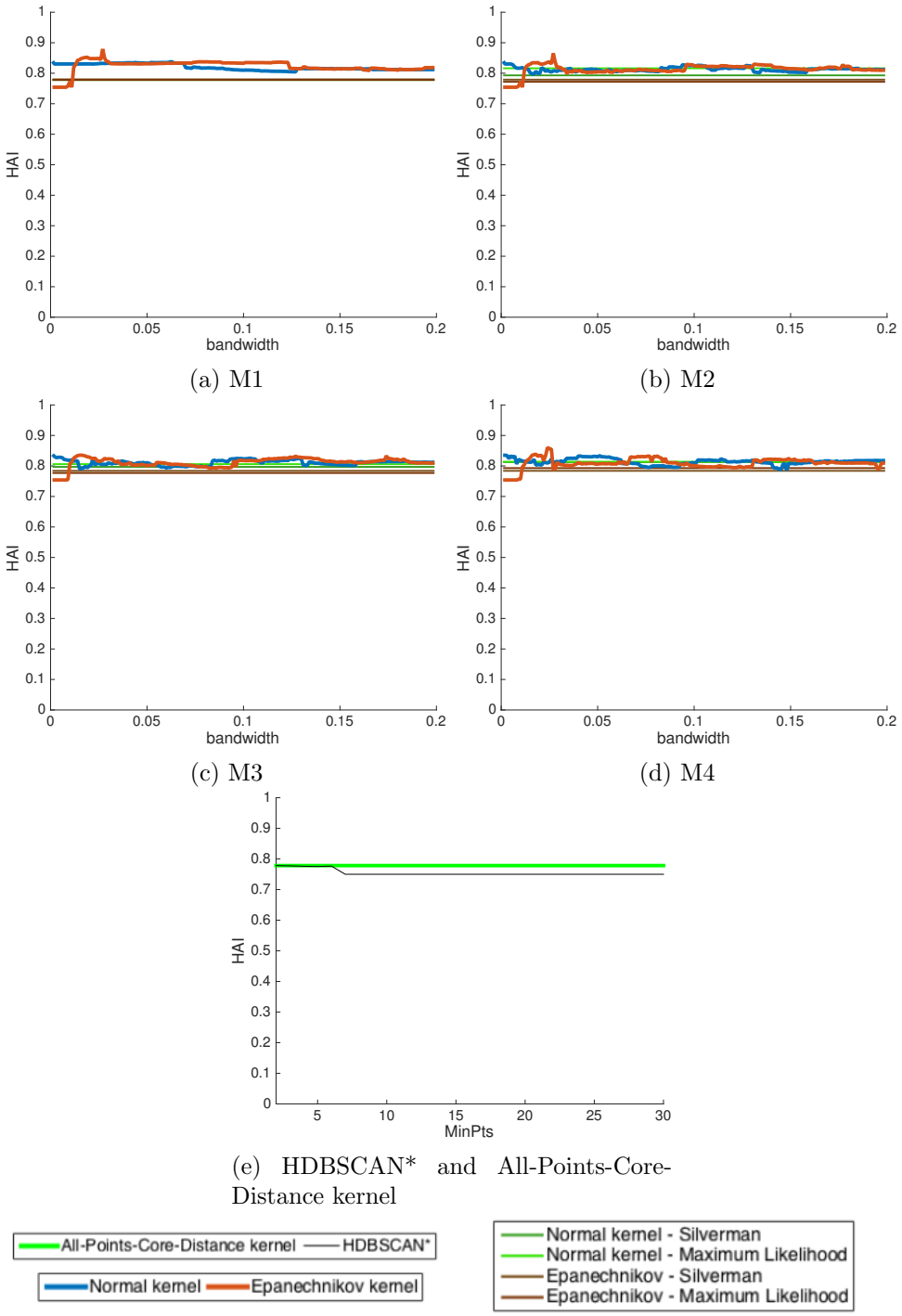
(Figure 6.32e), all bandwidths and m_{pts} can identify the underlying hierarchical structure well. Thus, the bandwidth is robust in an hierarchical setting.

6.4 Discussion

Our case study was divided into two major parts: in the first part, we analyzed the flat partitions that we can get from hierarchies build using HDBSCAN* and HDBSCANk, and in the second part, we compared our hierarchies to the available hierarchical ground truths. We exhaustively looked for the 'best' bandwidth, using various selection methods and also tested how well statistical bandwidth estimations did in a hierarchical clustering setting.

For datasets from whose hierarchy which we extracted flat partitions :

- While in two dimensions, the heuristic bandwidth estimators worked better with the Epanechnikov kernel, they did not perform well with that kernel in higher dimensions.
- The Normal kernel in general performed well if we could exhaustively search for the bandwidth in [0.001, 0.200] range.
- In terms of edge weight estimation techniques, we found that methods



(a) M1

(b) M2

(c) M3

(d) M4

(e) HDBSCAN* and All-Points-Core-Distance kernel

Figure 6.32: Dimensionality 5, separation factor 3 dataset: Change in HAI as bandwidth of kernel of HDBSCANk and m_{pts} of HDBSCAN* is varied

which underestimated the density – using only the top contributors of the objects they connected – were capable of performing better than simple methods of estimating density at the midpoint.

- We optimized internal validation measures, DBCV and SI, to select the 'best' partition and found that they were not always successful. For two dimensional datasets, they did well. While selecting the best stability partition from the hierarchies in higher dimensions, they could achieve an ARI of about 0.6 on an average. Further, selecting a partition that optimized DBCV or SI value performed better than choosing the stability partition of hierarchy built from a heuristic bandwidth.

- With increase in dimensionality, flat partitions from the Normal kernel in combination with the heuristic bandwidths started to perform better than using the Epanechnikov kernel.

- The All-Points-Core-Distance kernel integrated into HDBSCANk was outperformed by both the Normal and Epanechnikov kernels with edge weight estimation methods M2, M3 and M4 in higher dimensions.

For datasets whose hierarchical structure was available:

- When we performed the analysis of directly comparing hierarchies with ground truth for datasets generated from our hierarchical data generator, we found that the obtained cluster hierarchies did indeed correspond well to the ground truth structure, irrespective of dimensionality and overlap between clusters.

- With increase in dimensionality, Silverman's rule of thumb performed better than the Maximum Likelihood estimation bandwidth for the Normal kernel. With the Epanechnikov kernel, Maximum Likelihood estimation produced hierarchies that adhered to ground truth better as dimensionality increased. These were as good as the exhaustive search for bandwidth for all edge weight estimation methods tested.

- Increasing the separation between clusters helped built hierarchies that were better able to identify the underlying structure. Here, exhaustive search for bandwidth performed the best with either kernels and connectivity definitions.

- For variable number of clusters, we found HDBSCANk an exhaustive search for bandwidth showed the Epanechnikov and the Normal kernels to be robust.

Overall, we can say that it is possible to find better partitions than those obtained by HDBSCAN* or with the All-Points-Core-Distance kernel. In a practical scenario, exhaustive search over all bandwidths, comparing with the ground truth, is not an option. The partitions selected by optimization of interval cluster validation measures, DBCV and SI, were found to be, in general, better than the partitions from heuristic bandwidth estimators, especially in high dimensional datasets. However, using the edge estimation methods with top contributors and performing an exhaustive search for the bandwidth using an internal measure are time-intensive processes. The heuristic bandwidths hierarchies are not guaranteed to give good flat partitions. Thus, to trade time with higher value of ARI, using HDBSCAN* or the All-Points-Core-Distance kernel in HDBSCANk are better options.

In hierarchical settings, exhaustively searching for the 'best' bandwidth the Normal and Epanechnikov kernels could give better hierarchies than HDBSCAN* hierarchies. Also, the hierarchies obtained using the statistical bandwidth estimators gave high HAI values of about 0.8, meaning they can identify the underlying hierarchical structure to a good extent.

6.5 Summary

In this chapter, we presented results from our case study and tried answering the questions we had in the beginning. This case study analyzed our algorithms as well as internal validation measures, statistical bandwidth calculators, flat partitions and hierarchical datasets. We found that using K.D.E. in the hi-

erarchical density-based clustering setting lead to good results and we could improve on the baseline algorithm, HDBSCAN*, that we compared with. In the final chapter, we present the final conclusion and future work in the next chapter.

Chapter 7

Conclusion and Future Work

Density-based clustering has largely used distance to estimate density [21, 24, 2, 35]. As Azzalini [3] noted in 2007, we now have the computational power to use density estimates in clustering. In 1975, when Hartigan [28] proposed his ideas of density-based clustering, it was not possible to use computationally expensive methods, but now it is. There is extensive research around density-based clustering methods with kernels [2, 8, 17, 32, 49, 70] but they are algorithms that find a flat partition. To the best of our knowledge, there is no hierarchical algorithm that uses general density estimates.

This thesis was a first step towards integration of kernel density estimates into density-based hierarchical clustering. We started with the simple idea of replacing distance with density in an existing hierarchical density-based algorithm. The density between two objects was defined as the value of density at the midpoint of the edge connecting them. However, the midpoint and the density at the midpoint of each edge in the complete graph had to be estimated before a Minimum Spanning Tree could be extracted. Thus, it was computationally expensive. Further, the technique was prone to overestimation of density at the midpoint due to presence of objects closer to the midpoint than the end object vertices of the edge.

This led us to propose HDBSCANk, a framework for integrating arbitrary density estimates into hierarchical clustering. We found a number of kernel density estimates to use but no simple way to define the connectivity between objects. We, thus, proposed some possible definitions of connectivity by esti-

mating the edge weight between two connected objects by various methods.

We set out on this case study with many questions in mind

1. Can integration of an arbitrary kernel density estimate into hierarchical clustering do better than the kNN approach?

2. Does one kernel always outperform others?

3. Is one of our edge estimation methods always better than the others?

4. Since we do not have ground truth in practical applications, can we use an internal validation measure to predict the best parameter to use?

5. There are statistical methods for estimating bandwidths of kernels. Is it feasible to use them?

6. And most importantly, are we capable of identifying the hierarchical structure of a dataset?

To answer all these questions, we conducted an extensive experimental evaluation. We performed an exhaustive search for the best bandwidth for the selected kernels, and compared the results with bandwidths estimated by heuristics commonly used in statistics. We also used interval validation measures, DBCV [51] and SI [60], to choose the partition that optimized their value and checked how the value of ARI [34] of these partitions fared as compared to the partitions with the maximum ARI. We repeated this process for the Normal kernel as well as the Epanechnikov kernel with all edge weight estimation methods. Similar set-up results from the All-Points-Core-Distance kernel [50] and HDBSCAN* were computed.

There are many datasets with flat ground truths to test clustering algorithms but only few document datasets for which a hierarchical ground truth structure are available. To make the hierarchical comparison of this case study possible, we proposed a data generator that produces datasets with hierarchical ground truth.

Since it is time intensive to extract all partitions in real world applications, we compared those results with the stability partition technique [11].

Its results were found to be not far off from the potential if all partitions were extracted. These experiments were performed on 11 two-dimensional datasets taken from Joensuu Clustering Datasets [1], 16 high dimensional datasets generated with the data generator by Handl and Knowles [27] and 7 datasets from the UCI repository [43].

Producing datasets with the data generator presented in Chapter 5, we used HAI [36] to compare the clustering hierarchies with the ground truth hierarchies and found that all methods, kernels and heuristic bandwidths did well.

Our final recommendation for edge density connectivity estimation is to use Torque Rule estimation with top contributors, Golden search with the top contributors. Since the Epanechnikov kernel was found to be as good as the Normal kernel always, we suggest using it over the Normal kernel for density estimation since it is faster to compute. If a flat partition is needed, we do not recommend the heuristic bandwidth estimators unless the data is known to be well separated and low dimensional. In hierarchical settings, however, these heuristics are recommended, though with the exhaustive search for bandwidth, one can get better clustering hierarchies.

We found HDBSCAN* and the All-Points-Core-Distance kernel to be faster than other kernel density estimates. Though the Normal and the Epanechnikov kernels can do better than HDBSCAN* and the All-Points-Core-Distance kernel with edge weight estimation methods that use top contributors, they require the value of the bandwidth that cannot always be selected using a heuristic. Some datasets worked better with DBCV while partitions with higher ARI value were selected on optimizing the SI value. In general, they could find bandwidths that gave better ARI values than using heuristic bandwidths, especially in high dimensions.

7.1 Future Work

In this thesis, we proposed a general framework in which any kernel can be used for hierarchical density-based clustering. We tested on well known kernels

in statistics, and our research has opened up many venues for further research. Our algorithm is time intensive with continuous kernels like the Normal kernel, and the best edge weight estimation method to define connectivity between objects require extra computations before density of an edge in the Minimum Spanning Tree can be updated. Thus, we need to find ways to make either the computations faster or use different kernels.

There has recently been research focused on reducing the time to compute density estimates [57, 65, 81]. Raykar and et al. [57] propose a technique that reduces the estimation for one dimension from $\mathcal{O}(mn)$ time to $\mathcal{O}(n+m)$ where n is the number of objects for which density has to be estimated and m is the number of objects that influence density for each object. Zheng et al. [81] proposed randomized and deterministic algorithms for kernel density estimation on large datasets.

New kernels can also be proposed, like the parameterless All-Points-core-Distance kernel [50] that was integrated into HDBSCANk.

A different direction for future research could be to propose a new heuristic to calculate the bandwidth. We will need estimators that are able to estimate bandwidth for multidimensional data. For example, Duong and Hazelton [19] proposed a cross validation technique for multivariate kernel density estimation and Chacon and Duong [13] proposed a another bandwidth estimator. It would be interesting to see how these estimators perform in HDBSCANk.

Our research utilized numerical datasets. Real data, however, is not always just numerical. It can possess categorical attributes as well as binary attributes. Kernels have been proposed for handling these special feature types [5, 26]. Clustering datasets with different kinds of data is another direction to look into.

Bibliography

[1] Joensuu clustering datasets. http://http://cs.joensuu.fi/sipu/datasets/. Accessed: 2015-05-15.

[2] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: ordering points to identify the clustering structure. In *ACM Sigmod Record*, volume 28, pages 49–60. ACM, 1999.

[3] Adelchi Azzalini and Nicola Torelli. Clustering via nonparametric density estimation. *Statistics and Computing*, 17(1):71–80, 2007.

[4] Jeffrey D Banfield and Adrian E Raftery. Model-based gaussian and non-gaussian clustering. *Biometrics*, pages 803–821, 1993.

[5] Adrian W Bowman. A note on consistency of the kernel method for the analysis of categorical data. *Biometrika*, 67(3):682–684, 1980.

[6] Adrian W Bowman. An alternative method of cross-validation for the smoothing of density estimates. *Biometrika*, 71(2):353–360, 1984.

[7] Adrian W Bowman and Adelchi Azzalini. *Applied smoothing techniques for data analysis: The kernel approach with S-Plus illustrations: The kernel approach with S-Plus illustrations*. OUP Oxford, 1997.

[8] Joseph Kent Bryan. *Classification and clustering using density estimation*. PhD thesis, 1971.

[9] Ricardo JGB Campello, Davoud Moulavi, and Joerg Sander. A simpler and more accurate auto-hds framework for clustering and visualization of biological data. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 9(6):1850–1852, 2012.

[10] Ricardo JGB Campello, Davoud Moulavi, and Jörg Sander. Density-based clustering based on hierarchical density estimates. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 160–172. Springer, 2013.

[11] Ricardo JGB Campello, Davoud Moulavi, Arthur Zimek, and Jörg Sander. A framework for semi-supervised and unsupervised optimal extraction of clusters from hierarchies. *Data Mining and Knowledge Discovery*, 27(3):344–371, 2013.

[12] Ricardo JGB Campello, Davoud Moulavi, Arthur Zimek, and Jörg Sander. Hierarchical density estimates for data clustering, visualization, and outlier detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 10(1):5, 2015.

[13] José E Chacón and Tarn Duong. Bandwidth selection for multivariate density derivative estimation, with applications to clustering and bump hunting. Technical report, 2012.

[14] Probal Chaudhuri and James S Marron. Sizer for exploration of structures in curves. *Journal of the American Statistical Association*, 94(447):807–823, 1999.

[15] Probal Chaudhuri and James Steven Marron. Scale space view of curve estimation. *Annals of Statistics*, pages 408–428, 2000.

[16] Antonio Cuevas, Manuel Febrero, and Ricardo Fraiman. Estimating the number of clusters. *Canadian Journal of Statistics*, 28(2):367–382, 2000.

[17] Antonio Cuevas, Manuel Febrero, and Ricardo Fraiman. Cluster analysis: a further approach based on density estimation. *Computational Statistics & Data Analysis*, 36(4):441–459, 2001.

[18] Robert P. W. Duin. On the choice of smoothing parameters for parzen estimators of probability density functions. *IEEE Transactions on Computers*, (11):1175–1179, 1976.

[19] Tarn Duong and Martin L Hazelton. Cross-validation bandwidth matrices for multivariate kernel density estimation. *Scandinavian Journal of Statistics*, 32(3):485–506, 2005.

[20] Levent Ertöz, Michael Steinbach, and Vipin Kumar. Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *SDM*, pages 47–58. SIAM, 2003.

[21] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, volume 96, pages 226–231, 1996.

[22] M. A. T. Figueiredo and A. K. Jain. Unsupervised learning of finite mixture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):381–396, March 2002.

[23] Ana LN Fred and Anubhav K Jain. Combining multiple clusterings using evidence accumulation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(6):835–850, 2005.

[24] Gunjan Gupta, Alexander Liu, and Joydeep Ghosh. Automated hierarchical density shaving: a robust automated clustering and visualization framework for large biological data sets. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 7(2):223–237, 2010.

[25] Hermans J. Habbema, J.D.F. and K. Van Den Broek. A stepwise discriminant analysis program using density estimation. *Compstat 1974: Proceedings in Computational Statistics*, pages 101–110, 1974.

[26] Peter Hall. On nonparametric multivariate binary discrimination. *Biometrika*, 68(1):287–294, 1981.

[27] Julia Handl and Joshua Knowles. Cluster generators for large high-dimensional data sets with large numbers of clusters. http://personalpages.manchester.ac.uk/mbs/julia.handl/generators.pdf, 2005.

[28] John A Hartigan. Clustering algorithms. 1975.

[29] John A Hartigan. Consistency of single linkage for high-density clusters. *Journal of the American Statistical Association*, 76(374):388–394, 1981.

[30] Michel Herbin, Noël Bonnet, and Philippe Vautrot. Estimation of the number of clusters and influence zones. *Pattern Recognition Letters*, 22(14):1557–1568, 2001.

[31] William Hersh, Chris Buckley, TJ Leone, and David Hickam. Ohsumed: An interactive retrieval evaluation and new large test collection for research. In *SIGIR94*, pages 192–201. Springer, 1994.

[32] Alexander Hinneburg and Hans-Henning Gabriel. Denclue 2.0: Fast clustering based on kernel density estimation. In *Advances in Intelligent Data Analysis VII*, pages 70–80. Springer, 2007.

[33] Alexander Hinneburg and Daniel A Keim. A general approach to clustering in large databases with noise. *Knowledge and Information Systems*, 5(4):387–415, 2003.

[34] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[35] Anil K Jain and Richard C Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[36] David M Johnson, Caiming Xiong, Jing Gao, and Jason J Corso. Comprehensive cross-hierarchy cluster agreement evaluation. In *AAAI (Late-Breaking Developments)*, 2013.

[37] Stephen C Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

[38] M Chris Jones, James S Marron, and Simon J Sheather. A brief survey of bandwidth selection for density estimation. *Journal of the American Statistical Association*, 91(433):401–407, 1996.

[39] MC Jones and RF Kappenman. On a class of kernel density estimate bandwidth selectors. *Scandinavian Journal of Statistics*, pages 337–349, 1992.

[40] Jack Kiefer. Sequential minimax search for a maximum. *Proceedings of the American mathematical society*, 4(3):502–506, 1953.

[41] L. Lelis and J. Sander. Semi-supervised density-based clustering. In *Ninth IEEE International Conference on Data Mining*, pages 842–847, Dec 2009.

[42] David D Lewis, Yiming Yang, Tony G Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *The Journal of Machine Learning Research*, 5:361–397, 2004.

[43] M. Lichman. UCI machine learning repository, 2013.

[44] Clive R Loader. Bandwidth selection: classical or plug-in? *Annals of Statistics*, pages 415–438, 1999.

[45] Don O Loftsgaarden, Charles P Quesenberry, et al. A nonparametric estimate of a multivariate density function. *The Annals of Mathematical Statistics*, 36(3):1049–1051, 1965.

[46] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[47] Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55, 1936.

[48] Pavan Kumar Mallapragada, Rong Jin, and Anil Jain. Non-parametric mixture models for clustering. In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 334–343. Springer, 2010.

[49] Giovanna Menardi and Adelchi Azzalini. An advancement in clustering via nonparametric density estimation. *Statistics and Computing*, 24(5):753–767, 2014.

[50] Davoud Moulavi. *Finding, Evaluating and Exploring Clustering Alternatives Unsupervised and Semi-supervised*. PhD thesis, Department of Computing Science, University of Alberta, 2014. unpublished thesis.

[51] Davoud Moulavi, Pablo A Jaskowiak, Ricardo JGB Campello, Arthur Zimek, Jörg Sander, et al. Density-based clustering validation. In *Proceedings of the 14th SIAM International Conference on Data Mining (SDM), Philadelphia, PA*, pages 839–847, 2014.

[52] Francis O Oyegue and Sunday M Ogbonmwan. The efficiency of product multivariate kernels. In *Proceedings of the World Congress on Engineering and Computer Science*, volume 2, 2014.

[53] Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

[54] Yaling Pei and Osmar Zaïane. A synthetic data generator for clustering and outlier analysis. Technical report, 2006.

[55] Robert Clay Prim. Shortest connection networks and some generalizations. *Bell system technical journal*, 36(6):1389–1401, 1957.

[56] William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

[57] Vikas C Raykar, Ramani Duraiswami, and Linda H Zhao. Fast computation of kernel estimators. *Journal of Computational and Graphical Statistics*, 19(1):205–220, 2010.

[58] Stephen J Roberts. Parametric and non-parametric unsupervised cluster analysis. *Pattern Recognition*, 30(2):261–272, 1997.

[59] Murray Rosenblatt et al. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3):832–837, 1956.

[60] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[61] Mats Rudemo. Empirical choice of histograms and kernel density estimators. *Scandinavian Journal of Statistics*, pages 65–78, 1982.

[62] Bernhard Schlkopf, John Platt, and Thomas Hofmann. *Map-Reduce for Machine Learning on Multicore*, pages 281–288. MIT Press, 2007.

[63] David W Scott and George R Terrell. Biased and unbiased cross-validation in density estimation. *Journal of the American Statistical Association*, 82(400):1131–1146, 1987.

[64] W Scott David. Multivariate density estimation. theory, practice and visualization, 1992.

[65] Matineh Shaker, Jonas Nordhaug Myhre, and Deniz Erdogmus. Computationally efficient exact calculation of kernel density derivatives. *Journal of Signal Processing Systems*, 81(3):321–332, 2015.

[66] Bernard W Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.

[67] Jeffrey S Simonoff. *Smoothing methods in statistics*. Springer Science & Business Media, 2012.

[68] Werner Stuetzle. Estimating the cluster tree of a density by analyzing the minimal spanning tree of a sample. *Journal of classification*, 20(1):025–047, 2003.

[69] Werner Stuetzle and Rebecca Nugent. A generalized single linkage method for estimating the cluster tree of a density. *Journal of Computational and Graphical Statistics*, 2012.

[70] Andrej Taliun, Michael H Böhlen, and Arturas Mazeika. Core: Non-parametric clustering of large numeric databases. In *SDM*, pages 14–25. SIAM, 2009.

[71] George R Terrell and David W Scott. Variable kernel density estimation. *The Annals of Statistics*, pages 1236–1265, 1992.

[72] Thanh N Tran, Ron Wehrens, and Lutgarde MC Buydens. Knn-kernel density-based clustering for high-dimensional multivariate data. *Computational Statistics & Data Analysis*, 51(2):513–525, 2006.

[73] Berwin A Turlach et al. *Bandwidth selection in kernel density estimation: A review*. Université catholique de Louvain, 1993.

[74] Cornelis Joost Van Rijsbergen. Foundation of evaluation. *Journal of Documentation*, 30(4):365–373, 1974.

[75] Xiao-Feng Wang and Yifan Xu. Fast clustering using adaptive density peak detection. *Statistical methods in medical research*, page 0962280215609948, 2015.

[76] Larry Wasserman. *All of nonparametric statistics*. Springer Science & Business Media, 2006.

[77] David Wishart. Mode analysis: A generalization of nearest neighbor which reduces chaining effects. *Numerical Taxonomy*, 76(17):282–311, 1969.

[78] M Anthony Wong and Christian Schaak. Using the kth nearest neighbor clustering procedure to determine the number of subpopulations. 1982.

[79] Lei Xu and Michael I Jordan. On convergence properties of the em algorithm for gaussian mixtures. *Neural computation*, 8(1):129–151, 1996.

[80] Charles T Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *Computers, IEEE Transactions on*, 100(1):68–86, 1971.

[81] Yan Zheng, Jeffrey Jestes, Jeff M Phillips, and Feifei Li. Quality and efficiency for kernel density estimates in large data. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 433–444. ACM, 2013.