Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *http://www.jstatsoft.org/*

DeBaCl: A Python Package for Interactive DEnsity-BAsed CLustering

Brian P. Kent
Carnegie Mellon University

Alessandro Rinaldo
Carnegie Mellon University

Timothy Verstynen
Carnegie Mellon University

Abstract

The level set tree approach of Hartigan (1975) provides a probabilistically based and highly interpretable encoding of the clustering behavior of a dataset by representing the hierarchy of data modes with the dendrogram of the level sets of a density estimator. This representation offers many advantages for exploratory analysis and clustering, especially for complex and high-dimensional data. Several R packages exist for level set tree estimation, but their practical usefulness is limited by computational inefficiency, lack of statistical justification, and absence of interactive graphical capabilities. To make it easier for practitioners to capture the advantages of level set trees, we have written the Python package **DeBaCl** for DEnsity-BAsed CLustering. In this article we illustrate how **DeBaCl**'s level set tree estimates can be used for difficult clustering tasks and interactive graphical data analysis. The package is intended to promote the practical use of level set trees through improvements in computational efficiency, flexible algorithms, and a high degree of user customization. In addition, we show how the level set tree methdology can be easily extended to deal with functional data.

Keywords: density-based clustering, level set tree, Python, interactive graphics, functional data analysis.

1. Introduction

Clustering is one of the most fundamental tasks in statistics and machine learning, and innumerable algorithms are available to practitioners. Some of the most popular methods, such

as K-means (MacQueen 1967; Lloyd 1982) and spectral clustering (Shi and Malik 2000), rely on the key operational assumption that there is one optimal partition of the data into K well-separated groups, where K is assumed to be known *a priori*. While effective in some cases, this flat or scale-free notion of clustering is inadequate when the data are very noisy or corrupted, or exhibit complex multimodal behavior and spatial heterogeneity, or simply when the value of K is unknown. In these cases, hierarchical clustering affords a more realistic and flexible framework in which the data are assumed to have multi-scale clustering features that can be captured by a hierarchy of nested subsets of the data, rather than a single partition. The knowledge of these subsets and their order of inclusions—typically depicted as a dendrogram—provides a great deal of information that goes beyond the original clustering task. In particular, it frees the practitioner from the requirement of knowing in advance the "right" number of clusters, provides a useful global summary of the entire dataset, and allows the practitioner to identify and focus on interesting sub-clusters at different levels of spatial resolution.

There are, of course, myriad algorithms for hierarchical clustering. However, in most cases their usage is advocated on the basis of heuristic arguments or computational ease, rather than well-founded theoretical guarantees. The high-density hierarchical clustering paradigm put forward by Hartigan (1975) is an exception. It is based on the simple but powerful definition of clusters as the maximal connected components of the super-level sets of the probability density specifying the data-generating distribution. This formalization has numerous advantages: (1) it provides a probabilistic notion of clustering that conforms to the inuition that clusters are the regions with largest probability to volume ratio; (2) it establishes a direct link between the clustering task and the fundamental problem of nonparametric density estimation; (3) it allows for a clear definition of clustering performance and consistency (Hartigan 1981) that is amenable to rigorous theoretical analysis and (4) as we show below, the dendrogram it produces is highly interpretable, offers a compact yet informative representation of a distribution, and can be interactively queried to extract and visualize subsets of data at desired resolutions. Though the notion of high-density clustering has been studied for quite

some time (Polonik 1995), recent theoretical advances have further demonstrated the flexibility and power of hierarhical density clustering XXX CITE LOTS OF PAPERS XXX. [bpk: which ones?]

This paper introduces the Python package **DeBaCl** for efficient and statistically-principled DEnsity-BAsed CLustering. **DeBaCl** is not the first implementation of level set tree estimation and clustering; the R packages **denpro** (Klemelä 2004), **gslclust** (Stuetzle and Nugent 2010), and **pdfCluster** (Azzalini and Menardi 2012) also produce various level set tree estimators. However, they tend to be too inefficient for most practical uses and rely on methods lacking rigorous theoretical justification. The popular nonparametric density-based clustering algorithm DBSCAN (Ester, Kriegel, and Xu 1996) is implemented in the R package **fpc** (Hennig 2013) and Python library **scikit-learn** (Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss, Dubourg, Vanderplas, Passos, Cournapeau, Brucher, Perrot, and Duchesnay 2011), but this method does not provide an estimate of the level set tree. [Tim: more specifics here about existing inefficiencies.]

DeBaCl handles much larger datasets than existing software, improves computational speed, and extends the utility of level set trees in three important ways: (1) it provides several novel visualization tools to improve the readability and interpetability of density cluster trees; (2) it offers a high degree of user customization; and (3) it implements several recent methodological advances. In particular, it enables construction of level set trees for arbitrary functions over a dataset, building on the idea that level set trees can be used even with data that lack a bona fide probability density fuction. **DeBaCl** also includes the first practical implementation of the recent, theoretically well-supported algorithm from Chaudhuri and Dasgupta (2010).

2. Level set trees

Suppose we have a collection of points $\mathbb{X}_n = \{x_1, \ldots, x_n\}$ in $\mathbb{R}^d$, which we model as i.i.d. draws from an unknown probability distribution with probability density function f (with respect to Lebesgue measure). Our goal is to identify and extract clusters of $\mathbb{X}_n$ without any

a priori knowledge about f or the number of clusters. Following the statistically-principled approach of Hartigan (1975), clusters can be identified as modes of f. For any threshold value $\lambda \geq 0$, the λ-*upper level set* of f is

$$L_\lambda(f) = \{x \in \mathbb{R}^d : f(x) \geq \lambda\}. \tag{1}$$

The connected components of $L_\lambda(f)$ are called the λ-clusters of f and *high-density clusters* are λ-clusters for any value of λ. It is easy to see that λ-clusters associated with larger values of λ are regions where the ratio of probability content to volume is higher. Also note that for a fixed value of λ, the corresponding set of clusters will typically not give a partition of $\{x : f(x) \geq 0\}$.

The level set tree is simply the set of all high-density clusters. This collection a tree because it has the following hierarchical property: for any two high-density clusters A and B, either A is a subset of B, B is a subset of A, or they are disjoint. This property allows us to visualize the level set tree with a dendrogram that shows all high-density clusters simultaneously and can be queried quickly and directly to obtain specific cluster assignments. Branching points of the dendrogram correspond to density levels where two or more modes of the pdf, i.e. new clusters, emerge. Each vertical line segment in the dendrogram represents the high-density clusters within a single pdf mode; these clusters are all subsets of the cluster at the level where the mode emerges. Line segments that do not branch are considered high-density modes, which we call the leaves of the tree. For simplicity, we tend to refer to the dendrogram as the level set tree itself.

Because f is unknown, the level set tree must be estimated from the data. Ideally, we would use the high-density clusters of a suitable density estimate $\widehat{f}$ to do this; for a well-behaved f and a large sample size, $\widehat{f}$ is close to f with high probability, so the level set tree for $\widehat{f}$ would be a good estimate for the level set tree of f (Chaudhuri and Dasgupta 2010). Unfortunately, this approach is not computationally feasible even for low-dimensional data because finding the upper level sets of $\widehat{f}$ requires evaluating the function on a dense mesh and identifying

λ-clusters requires a combinatorial search over all possible paths connecting any two points in the mesh.

Many methods have been proposed to overcome these computational obstacles. The first category includes techniques that remain faithful to the idea that clusters are regions of the sample space. Members of this family include histogram-based partitions (Klemelä 2004), binary tree partitions Klemelä (2005) (implemented in the R package **denpro**) and Delaunay triangulation partitions (Azzalini and Torelli 2007) (implemented in R package **pdfCluster**). These techniques tend to work well for low-dimension data, but suffer from the curse of dimensionality because partitioning the sample space requires an exponentially increasing number of cells or algorithmic complexity (Azzalini and Torelli 2007).

In contrast, another family of estimators produces high-density clusters of data points rather than sample space regions; this is the approach taken by our package. Conceptually, these methods estimate the level set tree of f by intersecting the level sets of f with the sample points $\mathbb{X}_n$ and then evaluating the connectivity of each set by graph theoretic means. This typically consists of the same three high-level steps: estimation of the probability density $\widehat{f}(x)$ from the data $\mathbb{X}^n$; construction of graph G that describes the similarity between each pair of data points, represented by graph vertices; and a search for connected components in a series of subgraphs G induced by removing nodes and/or edges of insufficient weight, relative to various density level values.

The variations within the latter category are found in the definition of the similarity graph G, the set of density levels over which to iterate, and the way in which G is restricted to a subgraph for a given density level λ. *Edge iteration* methods assign a weight to the edges of G based on the proximity of the incident vertices in feature space (Chaudhuri and Dasgupta 2010) or the value of $\widehat{f}(x)$ at the incident vertices (Wong and Lane 1983) or on a line segment connecting them (Stuetzle and Nugent 2010). For these procedures, the relevant density levels are the edge weights of G. Frequently, iteration over these levels is done by initializing G with an empty edge set and adding successively more heavily weighted edges, in the manner of

traditional single linkage clustering. In this family, the Chaudhuri and Dasgupta algorithm (which is a generalization of Wishart (1969)) is particularly interesting because the authors prove finite sample rates for convergence to the true level set tree (Chaudhuri and Dasgupta 2010). To the best of our knowledge, however, only Stuetzle and Nugent (2010) has a publicly available implementation, in the R package **gslclust**.

Point iteration methods construct similarity graph G so the vertex for observation x_i is weighted according to $\widehat{f}(x_i)$, but the edges are unweighted. In the simplest form, there is an edge between the vertices for observations x_i and x_j if the distance between x_i and x_j is smaller than some threshold value, or if x_i and x_j are among each other's k-closest neighbors (Kpotufe and Luxburg 2011; Maier, Hein, and von Luxburg 2009). A more complicated version places an edge (x_i, x_j) in G if the amount of probability mass that would be needed to fill the valleys along a line segment between x_i and x_j is smaller than a user-specified threshold (Menardi and Azzalini 2013). The method is available in the R package **pdfCluster**.

3. Implementation

The default level set tree algorithm in **DeBaCl** is described in Algorithm 1, based on the method proposed by Kpotufe and von Luxburg (2011) and Maier, Hein, and von Luxburg (2009). For a sample with n observations in $\mathbb{R}^d$, the k-nearest neighbor (kNN) density estimate is:

$$\widehat{f}(x_j) = \frac{k}{n \cdot v_d \cdot r_k^d(x_j)} \tag{2}$$

where v_d is the volume of the Euclidean unit ball in $\mathbb{R}^d$ and $r_k(x_j)$ is the Euclidean distance from point x_j to its k'th closest neighbor. The process of computing subgraphs and finding connected components of those subgraphs is implemented with the **igraph** package (Csardi and Nepusz 2006). Our package also depends on the **NumPy** and **SciPy** packages for basic computation (Jones, Oliphant, and Peterson 2001) and the **Matplotlib** package for plotting (Hunter 2007). We use this algorithm because it is straightforward and fast; although the naïve version does require computation of all $\binom{n}{2}$ pairwise distances, the procedure can be substan-

Algorithm 1: DeBaCl level set tree estimation procedure

Input: $\{x_1, \ldots, x_n\}$, k, γ
Output: $\widehat{\mathcal{T}}$, a hierarchy of subsets of $\{x_1, \ldots, x_n\}$

$G \leftarrow k$-nearest neighbor similarity graph on $\{x_1, \ldots, x_n\}$;
$\widehat{f}(\cdot) \leftarrow k$-nearest neighbor density estimate based on $\{x_1, \ldots, x_n\}$;
for $j \leftarrow 1$ **to** n **do**
  $\lambda_j \leftarrow \widehat{f}(x_j)$;
  $L_{\lambda_j} \leftarrow \{x_i : \widehat{f}(x_i) \geq \lambda_j\}$;
  $G_j \leftarrow$ subgraph of G induced by L_j;
  Find the connected components of G_{λ_j};
$\widehat{\mathcal{T}} \leftarrow$ dendrogram of connected components of graphs $G_1, \ldots, G_n$, ordered by inclusions;
$\widehat{\mathcal{T}} \leftarrow$ remove components of size smaller than γ;
return $\widehat{\mathcal{T}}$

tially shortened by estimating connected components on a sparse grid of density levels. The implementation of this algorithm is novel in its own right (to the best of our knowledge), and **DeBaCl** includes several other new visualization and methodological tools.

3.1. Visualization tools

Our level set tree plots increase the amount of information contained in a tree visualization and greatly improve interpretability relative to existing software. Suppose a sample of 2,000 observations in $\mathbb{R}^2$ from a mixture of three Gaussian distributions (Figure 1a). The traditional level set tree is illustrated in Figure 1b and the **DeBaCl** version in Figure 1c. A plot based only on the mathematical definition of a level set tree conveys the structure of the mode hierarchy and indicates the density levels where each tree node begins and ends, but does not indicate how many points are in each branch or visually associate the branches with a particular subset of data. In the proposed software package, level set trees are plotted to emphasize the empirical mass in each branch (i.e. the fraction of data points in the associated cluster): tree branches are sorted from left-to-right by decreasing empirical mass, branch widths are proportional to empirical mass, and the white space around the branches is proportional to empirical mass. For matching tree nodes to the data, branches can be colored to correspond to

high-density data clusters (Figures 1c and 1d). Clicking on a tree branch produces a banner that indicates the start and end levels of the associated high-density cluster as well as its empirical mass. [bpk:remember to add this banner to the plot in figure 1c.]

The level set tree plot is an excellent tool for interactive exploratory data analysis, because it acts as a handle for identifying and plotting spatially coherent, high-density subsets of data. The full power of this feature can be seen clearly with the more complex data of Section 4.

3.2. Alternate scales

By construction the nodes of a level set tree are indexed by density levels λ, which determine the scale of the vertical axis in a plot of the tree. While it does convey the parent-child relationships in the tree, interpretability of the λ scale is limited by the fact that it depends on the height of the density estimate $\widehat{f}$. It is not clear, for example, whether $\lambda = 1$ would be a low- or a high-density threshold; this depends on the particular distribution.

To remove the scale dependence we can instead index level set tree nodes based on the probability content of upper level sets. Specifically, let α be a number between 0 and 1 and define

$$\lambda_\alpha = \sup\left\{\lambda\colon \int_{x\in L_\lambda(f)} f(x)dx \geq \alpha\right\} \tag{3}$$

to be the value of λ for which the upper level set of f has probability content no smaller than α (Rinaldo, Singh, Nugent, and Wasserman 2012). The map $\alpha \mapsto \lambda_\alpha$ gives a monotonically decreasing one-to-one correspondence between values of α in $[0, 1]$ and values of λ in $[0, \max_x f(x)]$. In particular, $\lambda_1 = 0$ and $\lambda_0 = \max_x f(x)$. For an empirical level set tree, set λ_α to the α-quantile of $\{\widehat{f}(x_i)\}_{i=1}^n$. Expressing the height of the tree in terms of α instead of λ does not change the topology (i.e. number and ordering of the branches) of the tree; the re-indexed tree is a deformation of the original tree in which some of its nodes are stretched out and others are compressed.

α-indexing is more interpretable and useful for several reasons. The α level of the tree indexes

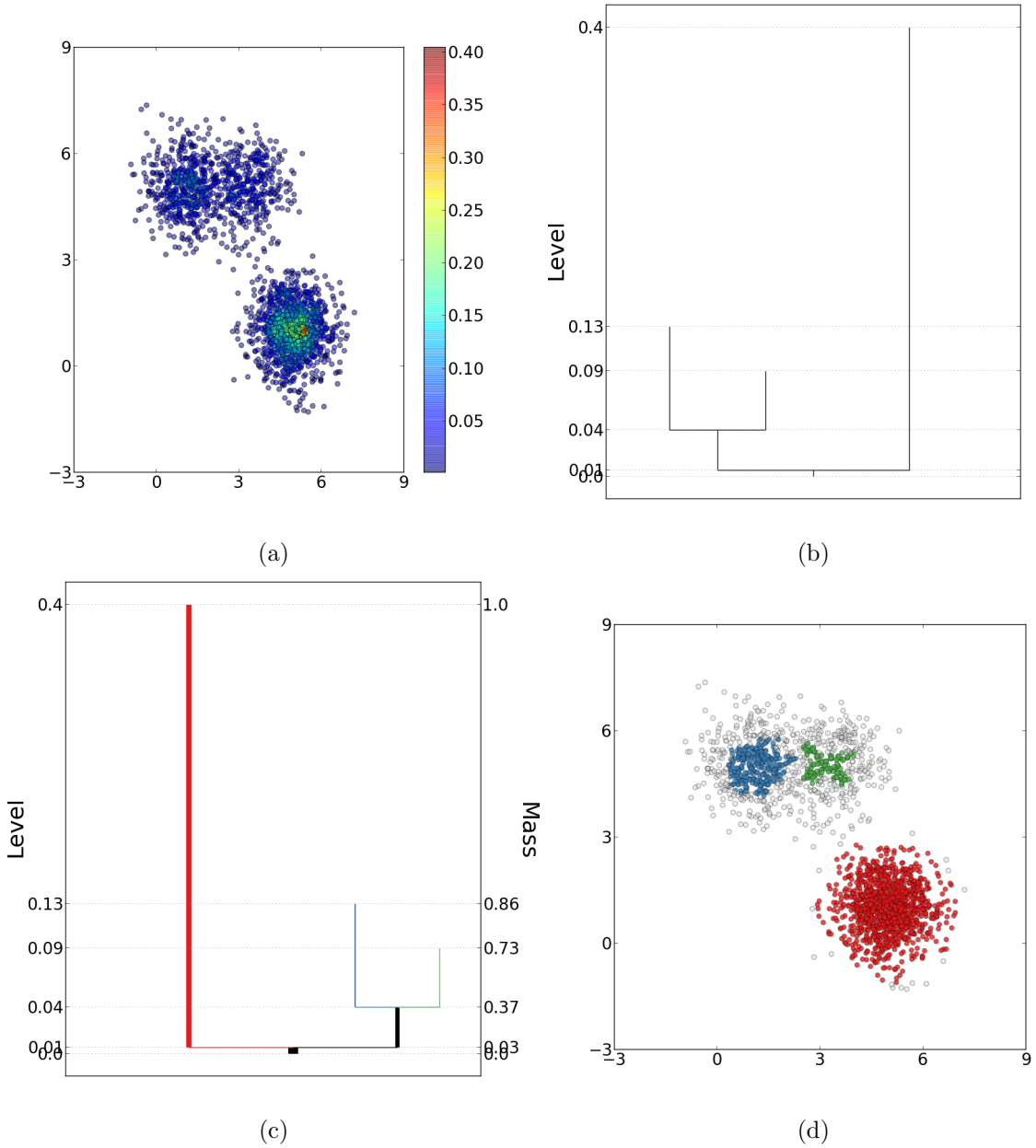


Figure 1: Level set tree plots and cluster labeling for a simple simulation. A level set tree is constructed from a sample of 2,000 observations is drawn from a mix of three Gaussians in $\mathbb{R}^2$. a) The kNN density estimator evaluated on the data. b) A plot of the tree based only on the mathematical definition of level set trees. c) New level set tree plot, from **DeBaCl**. The branches are ordered from left by decreasing mass, colored to match the cluster labels in d, and a second vertical axis is added that indicates that fraction of background mass at each critical density level. d) Cluster labels from the *all-mode* labeling technique, where each leaf of the level set tree is designated as a cluster.

clusters corresponding to the $1 - \alpha$ fraction of "most clusterable" data points; in particular, larger α values yield more compact and well-separated clusters, while smaller values can be used for de-noising and outlier removal. Because α is always between 0 and 1, scaling by probability content also enables comparisons of level set trees arising from data sets drawn from different pdfs, possibly in spaces of different dimensions. Finally, the α-index is more effective than λ-indexing in representing regions of large probability content but low density and is less affected by small fluctuations in density estimates.

A common (incorrect) intuition when looking at an α-indexed level set tree plot is to interpret the height of the branches as the size of the corresponding cluster. To facilitate this intuition, we introduce the κ-indexed level set tree, where the height of each branch is proportional to the mass of the corresponding cluster. For the population κ-tree, suppose each density level set is composed of a finite number of components C_k:

$$L_\lambda(f) = \bigcup_k C_k. \tag{4}$$

Let the excess mass for density level λ and region $C \in \mathbb{R}^d$ be

$$\mathcal{E}(\lambda, C) = \int_C f(x)dx - \lambda \int_C dx. \tag{5}$$

Assume for simplicity that the support is connected, i.e. $L_0(f) = C_0$ and that the root node has index 0. Let parent_i be the parent of node i, and kids_i be the child nodes of i.

Imagine λ increasing from 0 to the maximum of f. Each node i of a level set tree has a birth level λ_i' where its high-density cluster first appears and a death level λ_i'' where its cluster splits or vanishes. The true κ-tree can be defined recursively by associating with each node i two

numbers κ_i' and κ_i'' such that $\kappa_i' - \kappa_i''$ is the *salient mass* of node i. Specifically, set

$$\begin{aligned}
\kappa_0' &= 1, \\
\kappa_i' &= \kappa_{\text{parent}_i}'', \\
\kappa_i'' &= \kappa_i' - \mathcal{E}(\lambda_i', C_i) + \sum_{j \in \text{kids}_i} \mathcal{E}(\lambda_j', C_j).
\end{aligned}$$

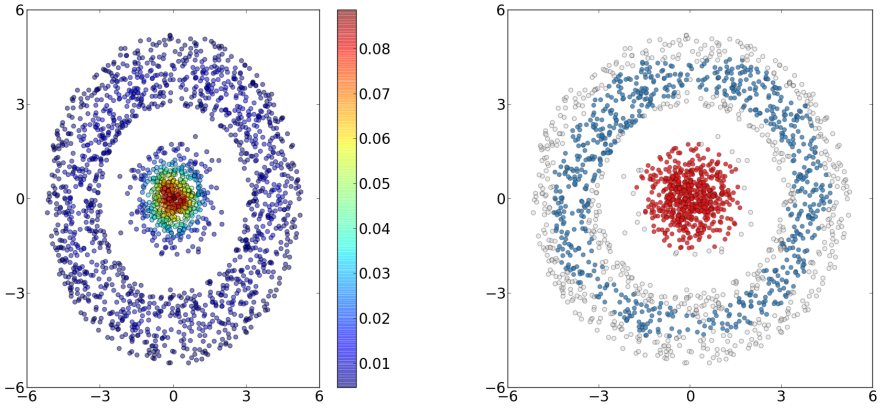
Note that switching from the λ to α level set tree index did not change the topology of the tree, switching to the κ index *does* change the topology. In practice we subtract the above quantities from 1 to get an increasing scale that matches the λ and κ scales. To estimate the κ-tree, let m_i be the fraction of data contained in the cluster for tree node i at birth (i.e. at λ_i'). Again, define the estimated tree recursively:

$$\begin{aligned}
\widehat{\kappa}_0' &= 1, \\
\widehat{\kappa}_i' &= \widehat{\kappa}_{\text{parent}_i}'', \\
\widehat{\kappa}_i'' &= \widehat{\kappa}_i' - m_i + \sum_{j \in \text{kids}_i} m_j
\end{aligned}$$

and subtract all values from 1 to invert the scale.

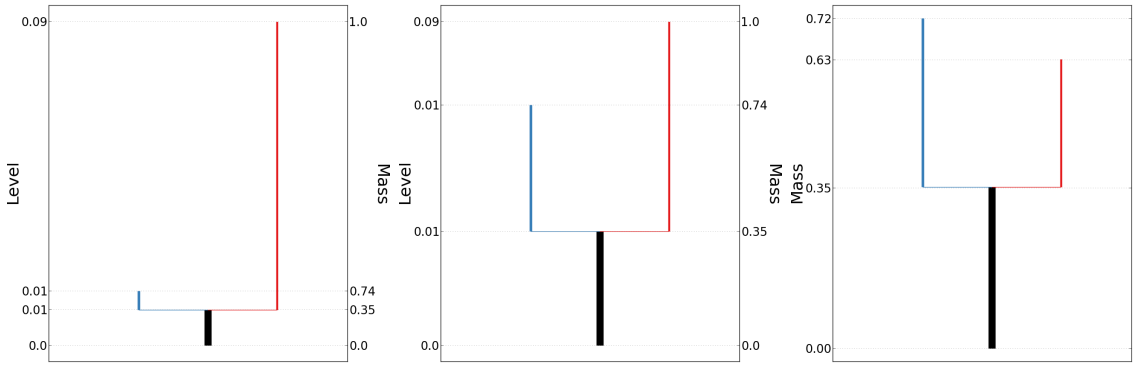
Figure 3 illustrates the differences between the three types of indexing for the "crater" example in Figure 2. This example consists of a central Gaussian with high density and low mass surrounded by a ring with high mass but uniformly low density. The λ-scale tree (Figure 3a) correctly indicates the heights of the modes of $\widehat{f}$, but tends to produce the incorrect intuition that the ring (blue node and blue points in Figure 2b) is small. The α-scale plot (Figure 3b) ameliorates this problem by indexing node heights to the quantiles of $\widehat{f}$. The blue node appears at $\alpha = 0.35$, when 65% of the data remains in the upper level set, and vanishes at $\alpha = 0.74$, when only 26% of the data remains in the upper level set. It is tempting to say that this means the blue node contains $0.74 - 0.35 = 0.39$ of the mass but this is incorrect because some of the difference in mass is due to the red node. This interpretation is precisely the design of the κ-tree, however, where we can say that the blue node contains $0.72 - 0.35 = 0.37$

of the data.



(a) (b)

Figure 2: [Tim: un-squish subfig (a)]



(a) (b) (c)

Figure 3

3.3. Cluster retrieval options

Clustering problems are typically ill-defined because the concept of a cluster is vague at best. As a result, we view the estimated level set tree as the most complete answer to a clustering task, but some applications require that each data point be assigned to a single cluster label. Much of the work on level set trees ignores this phase of a clustering application or assumes that labels will be assigned according to the connected components at a chosen λ (density) or

α (mass) level, which **DeBaCl** accomodates through the *upper set clustering* option. Rather than choosing a single density level, a practitioner might prefer to specify the number of clusters K (as with K-means). One way (of many) that this can be done is to find the first $K-1$ splits in the level set tree and identify each of the children from these splits as a cluster, known in **DeBaCl** as the *first-K clustering* technique. A third, preferred, option, avoids the choice of λ, α, or K altogether and treats each leaf of the level set tree as a separate cluster (Azzalini and Torelli 2007). We call this the *all-mode clustering* method. Use of these labeling options is illustrated in Section 4.

Note that each of these methods assigns only a fraction of points to clusters (the *foreground* points), while leaving low-density observations (*background* points) unlableled. Assigning the background points to clusters can be done with any classification algorithm, and **DeBaCl** includes a handful of simple options, including a k-nearest neighbor classifer, for the task.

3.4. Chaudhuri and Dasgupta algorithm

Chaudhuri and Dasgupta (2010) introduce an algorithm for estimating a level set tree that is particularly notable because the authors prove finite-sample convergence rates (where consistency is in the sense of Hartigan (1981)). The algorithm is a generalization of single linkage, reproduced here for convenience in Algorithm 2. To translate this program into a practical

Algorithm 2: Chaudhuri-Dasgupta level set tree estimation procedure (Chaudhuri and Dasgupta 2010).

Input: $\{x_1, \ldots, x_n\}$, k, α
Output: $\widehat{\mathcal{T}}$, a hierarchy of subsets of $\{x_1, \ldots, x_n\}$

$r_k(x_i) \leftarrow$ distance to the k'th neighbor of x_i;
for $r \leftarrow 0$ **to** ∞ **do**
  $G_r \leftarrow$ graph with vertices $\{x_i : r_k(x_i) \leq r\}$ and edges $\{(x_i, x_j) : \|x_i - x_j\| \leq \alpha r\}$;
  Find the connected components of G_{λ_r};
$\widehat{\mathcal{T}} \leftarrow$ dendrogram of connected components of graphs G_r, ordered by inclusions;
return $\widehat{\mathcal{T}}$

implementation, we must find a finite set of values for r such that the graph G_r can only

change at these values. When $\alpha = 1$, the only values of r where the graph can change are the edge lengths in the graph $e_{ij} = \|x_i - x_j\|$ for all i and j. Let r take on each value of e_{ij} in descending order; in each iteration remove vertices and edges with larger k-neighbor radius and edge length, respectively.

When $\alpha \neq 1$, the situation is trickier. First, note that including r values where the graph *does not* change is not a problem, since the original formulation of the method includes all values of $r \in \mathbb{R}^{+0}$. Clearly, the vertex set can still change at any edge length e_{ij}. The edge set can only change at values where $r = e_{ij}/\alpha$ for some i, j. Suppose $e_{u,v}$ and $e_{r,s}$ are consecutive values in a descending ordered list of edge lengths. Let $r = e/\alpha$, where $e_{u,v} < e < e_{r,s}$. Then the edge set $E = \{(x_i, x_j) : \|x_i - x_j\| \leq \alpha r = e\}$ does not change as r decreases until $r = e_{u,v}/\alpha$, where the threshold of αr now excludes edge (x_u, x_v). Thus, by letting r iterate over the values in $\bigcup_{i,j}\{e_{ij}, \frac{e_{ij}}{\alpha}\}$, we capture all possible changes in G_r.

In practice, starting with a complete graph and removing one edge at a time is extremely slow because this requires $2 * \binom{n}{2}$ connected component searches. The **DeBaCl** implementation includes an option to initialize the algorithm at the k-nearest neighbor graph instead, which is a substantially faster approximation to the Chaudhuri-Dasgupta method. This shortcut is still dramatically slower than **DeBaCl**'s geometric tree algorithm, which is one reason why we prefer the latter. Future development efforts will focus on improvements in the speed of the Chaudhuri-Dasgupta procedure.

3.5. Pseudo-densities for functional data

The level set tree estimation procedure in Algorithm 1 can be extended to work with data sampled from non-Euclidean spaces that do not admit a well-defined pdf. The lack of a density function seems at first blush to be an insurmountable problem for the level set tree framework because the trees are traditionally defined on the levels of a pdf. In this case, however, level set trees can be built based on the levels of a pseudo-density estimate that measures the similarity of observations and the overall connectivity of the sample space. Pseudo-densities cannot be

used to compute probabilities as in Euclidean spaces, but are proportional to the statistical expectations of estimates of the form $\widehat{f}$, which remain well-defined random quantities (Ferraty and Vieu 2006).

Random functions, for example, may have well-defined probability distributions that cannot be represented by pdfs (Billingsley 2012). To build level set trees for this type of data, **DeBaCl** accepts very general functions for $\widehat{f}$, including pseudo-densities, although the user must compute the pairwise distances. The package includes a utility function for evaluating a k-nearest neighbor pseudo-density estimator on the data based on the pairwise distances. Specifically, equation 2 is modified by expunging the term v^d and setting d arbitrarily to 1. An application is shown in Section 4.

3.6. User customization

One advantage of **DeBaCl** over existing cluster tree software is that **DeBaCl** is intended to be easily modified by the user. As described above, two major algorithm types are offered, as well as the ability to use pseudo-densities for functional data. In addition, the package allows a high degree of customization in the type of similarity graph, *data ordering function* (density, pseudo-density, or arbitrary function), pruning function, cluster labeling scheme, and background point classifier. In effect, the only fixed aspect of **DeBaCl** is that clusters are defined for every level to be connected components of a geometric graph.

4. Usage

4.1. Basic Example

[Get more information from Tim on what exactly the fiber tracks are] In this section we walk through the density-based clustering analysis of 10,000 fiber tracks mapped in a human brain with diffusion-weighted imaging. For this analysis we use only the subcortical endpoint of

each fiber track, which is in $\mathbb{R}^3$. Despite this straightforward context of finite, low-dimensional data, the clustering problem is somewhat challenging because the data are known to have complicated non-convex spatial patterns. For this paper we add the **DeBaCl** package to the Python path at run time, but this can be done in a more persistent manner for repeated use. The **NumPy** library is also needed for this example, and we assume the dataset is located in the working directory. Here we use our preferred algorithm, the geometric level set tree, in module **geom_tree**

```
## Import DeBaCl package
import sys
sys.path.append('/home/brian/Projects/debacl/DeBaCl/')

from debacl import geom_tree as gtree
from debacl import utils as utl

## Import other Python libraries
import numpy as np

## Load the data
X = np.loadtxt('0187_endpoints.csv', delimiter=',')
n, p = X.shape
```

The next step is to define parameters for construction and pruning of the level set tree, as well as general plot aesthetics. For this example we set the density and connectivity smoothness parameter k to be 1% of n, or 100. The pruning parameter γ is set to be $0.05n$ or 500. Tree branches with fewer points than this will be merged into larger sibling branches. For the sake of speed, we use a small subsample in this example.

```
## Downsample
n_samp = 5000
ix = np.random.choice(range(n), size=n_samp, replace=False)
```

```
X = X[ix, :]
n, p = X.shape

## Set level set tree parameters
p_k = 0.01
p_gamma = 0.05

k = int(p_k * n)
gamma = int(p_gamma * n)

## Set plotting parameters
utl.setPlotParams()
```

For straightforward cases like this one, the level set tree can be constructed and pruned with a single convenience function. Each of the density estimation, connectivity estimation, level set tree construction, and pruning steps can be done individually for more customization of parameters and method options (see below). Note the `print` function is overloaded to show a summary of the tree.

```
## Build the level set tree with the all-in-one function
tree = gtree.geomTree(X, k, gamma, n_grid=None, verbose=False)
print tree
```

```
     alpha1  alpha2  children   lambda1   lambda2 parent  size
key
0    0.0000  0.0044    [1, 2]  0.000000  0.000004   None  5000
1    0.0044  0.1394  [21, 22]  0.000004  0.000495      0  2971
2    0.0044  0.0664  [11, 12]  0.000004  0.000107      0  2007
11   0.0664  0.3946  [27, 28]  0.000107  0.004922      2  1424
12   0.0664  0.3718        []  0.000107  0.004428      2   303
21   0.1394  0.9978        []  0.000495  0.054786      1  1436
```

```
22   0.1394  0.9972        []  0.000495  0.054162      1   834
27   0.3946  0.4016  [29, 30]  0.004922  0.005057     11   870
28   0.3946  0.9992        []  0.004922  0.058173     11   363
29   0.4016  0.9998        []  0.005057  0.061124     27   432
30   0.4016  0.9274        []  0.005057  0.033544     27   431
```

The next step in the typical usage is to assign cluster labels to a set of foreground data points, with the function `GeomTree.getClusterLabels`. There are many ways to do this with a level set tree, and the choice of method is specified with the `method` argument. When the correct number of clusters K is known, the `first-k` option retrieves the first K disjoint clusters that appear when λ is increased from 0. Alternately, the `upper-set` option cuts the tree at a single level, which is useful if the goal is to include or exclude a certain fraction of the data from the upper level set. Here we use this function with α set to 0.05, which removes the 5% of the observations with the lowest estimated density (i.e. outliers) and clusters the remainder. Finally, the `all-mode` option returns a foreground cluster for each leaf of the level set tree, which avoids the need to specify either K, λ, or α.

Specify additional arguments for each method by keyword argument; the `getClusterLabels` method parses them intelligently. For all of the labeling methods the function returns two objects. The first is an $m \times 2$ matrix, where m is the number of points in the foreground set. The first column is the index of an observation in the full data matrix, and the second column is the cluster label. The second object is a list of the tree nodes that are foreground clusters. This is useful for coloring level set tree nodes to match observations plotted in feature space.

```
uc_k, nodes_k = tree.getClusterLabels(method='first-k', k=3)
uc_lambda, nodes_lambda = tree.getClusterLabels(method='upper-set', threshold=0.05,
     scale='lambda')
uc_mode, nodes_mode = tree.getClusterLabels(method='all-mode')
```

This tree summary table is much easier to understand in plot form, using the `GeomTree.plot` method. The key decision is which values to use as the primary vertical scale of the plot, which is controlled by the `form` parameter. When form is set to *lambda*, the vertical scale of the level set tree plot matches the scale of the estimated density. When form is set to *alpha*, the vertical scale represents the fraction of data that is excluded from the upper level set at each density level. Lastly, setting form to `kappa` creates a tree plot where the height of each node is proportional to the mass of the node minus the mass of its child nodes. See Section 3.2 for more detail.

The three plot forms are shown below for the foreground clusters based on *first-K* clustering with K set to 3. The plotting function returns a tuple with several objects, but only the first is useful for static plotting. Also note the use of the `nodes_k` object to color the tree nodes that correspond to foreground clusters. These plots are shown in Figure 4, along with a plot of the observations in feature space. The foreground points are colored to match the tree colors by using the `plotForeground` function from the **DeBaCl utils** module.

```
## Plot the level set tree with three different vertical scales, colored by the
    first-K clustering
fig = tree.plot(form='lambda', width='mass', color_nodes=nodes_k)[0]
fig.savefig('../figures/endpt_tree_lambda.png')

fig = tree.plot(form='alpha', width='mass', color_nodes=nodes_k)[0]
fig.savefig('../figures/endpt_tree_alpha.png')

fig = tree.plot(form='kappa', width='mass', color_nodes=nodes_k)[0]
fig.savefig('../figures/endpt_tree_kappa.png')



## Plot the foreground points from the first-K labeling
fig, ax = utl.plotForeground(X, uc_k, fg_alpha=0.6, bg_alpha=0.4, edge_alpha=0.3, s
    =22)
```

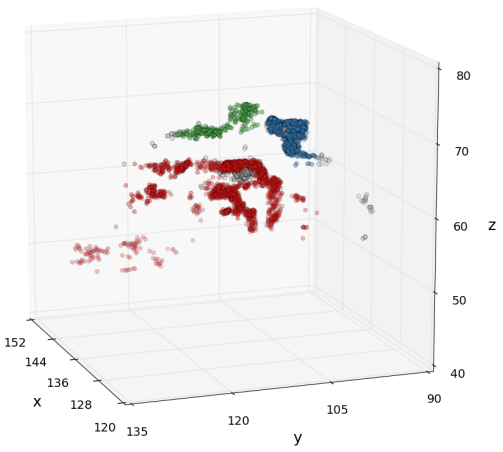
```
ax.elev = 14; ax.azim=160 # adjust the camera angle
fig.savefig('../figures/endpt_firstK_fg.png', bbox_inches='tight')
```

A level set tree plot is also useful as a scaffold for interactive exploration of spatially coherent subsets of data, either by selecting individual nodes of the tree or by retreiving high-density clusters at a selected density or mass level. These interactive tools are illustrated in Figure 5. [Tim: elaborate about walking through the branches of the tree.]

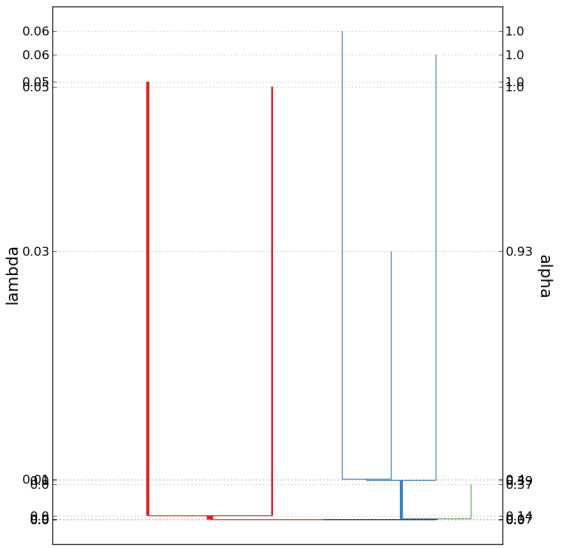
```
tool1 = gtree.ComponentGUI(tree, X, form='alpha', output=['scatter'], size=18,
    width='mass')
tool1.show()

tool2 = gtree.ClusterGUI(tree, X, form='alpha', width='mass', size=18)
tool2.show()
```

The final step of our standard data analysis is to assign background points to a foreground cluster. **DeBaCl**'s **utils** module includes several very simple classifiers for this task, although more sophisticated methods have been proposed (Azzalini and Torelli 2007). For this example we again use the 'first-K' cluster labeling output, and assign background points with a k-nearest neighbor classifier. The observations are plotted a final time, with all points assigned to a cluster (Figure 6).

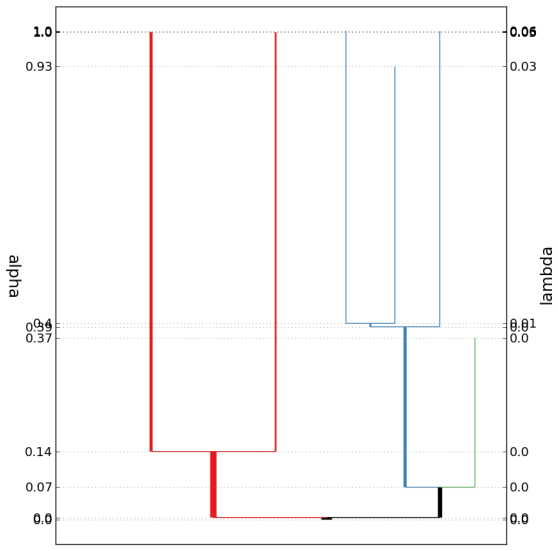
```
## Assign background points with a simple kNN classifier
segment = utl.assignBackgroundPoints(X, uc_k, method='knn', k=k)

## Plot all observations, colored by cluster
fig, ax = utl.plotForeground(X, segment, fg_alpha=0.6, bg_alpha=0.4, edge_alpha
    =0.3, s=22)
ax.elev = 14; ax.azim=160
fig.savefig('../figures/endpt_firstK_segment.png', bbox_inches='tight')
```



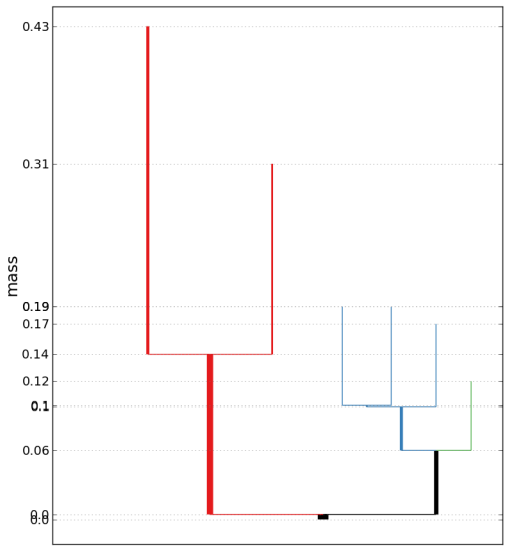
(a) Fiber endpoint data, colored by *first-K* foreground cluster



(b) Lambda scale

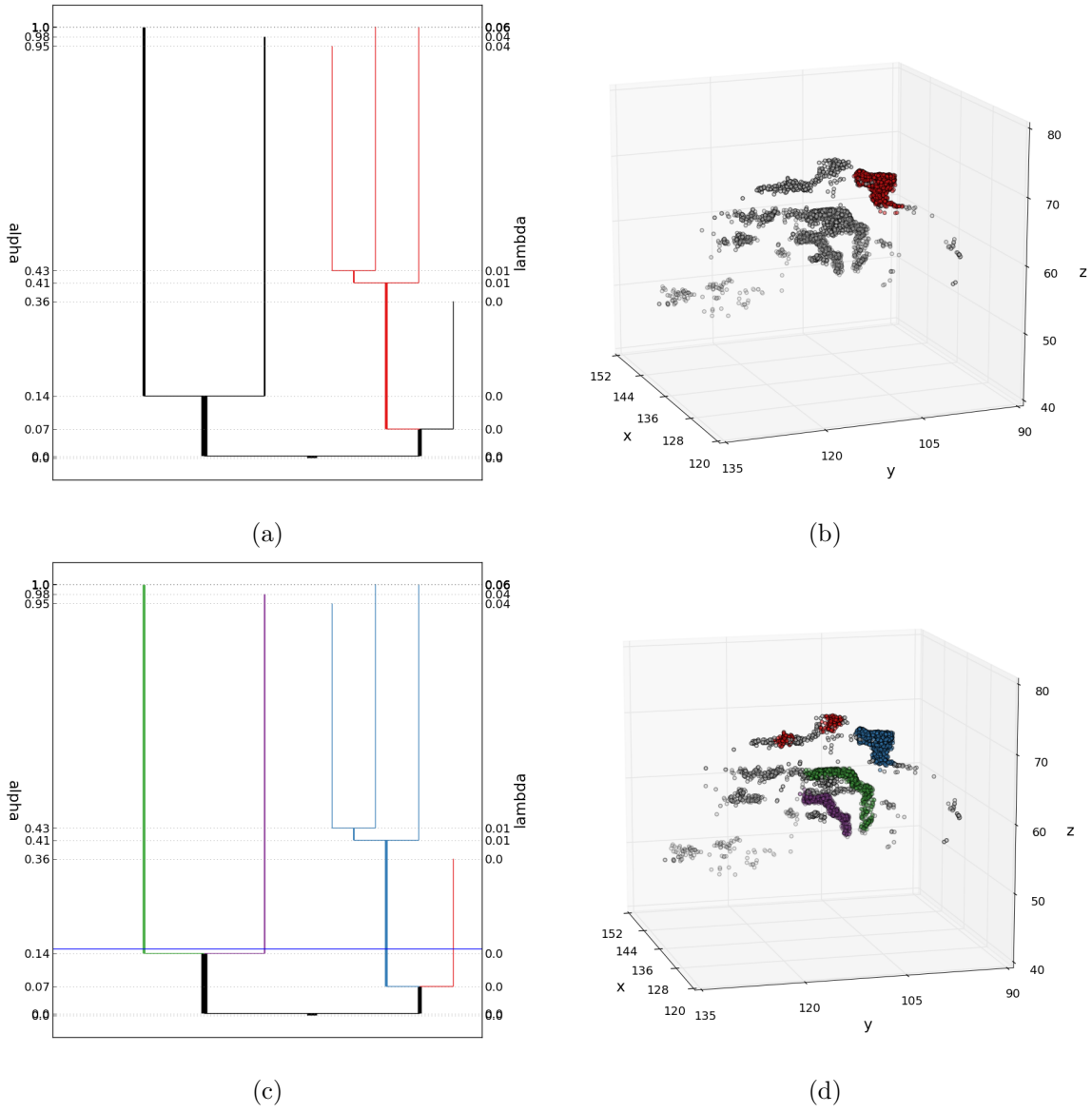


(c) Alpha scale



(d) Kappa scale

Figure 4: First-K clustering results with different vertical scales and the clusters in feature space.



(a) (b)

(c) (d)

Figure 5: The level set tree can be used as a scaffold for interactive exploration of data subsets or upper level set clusters.

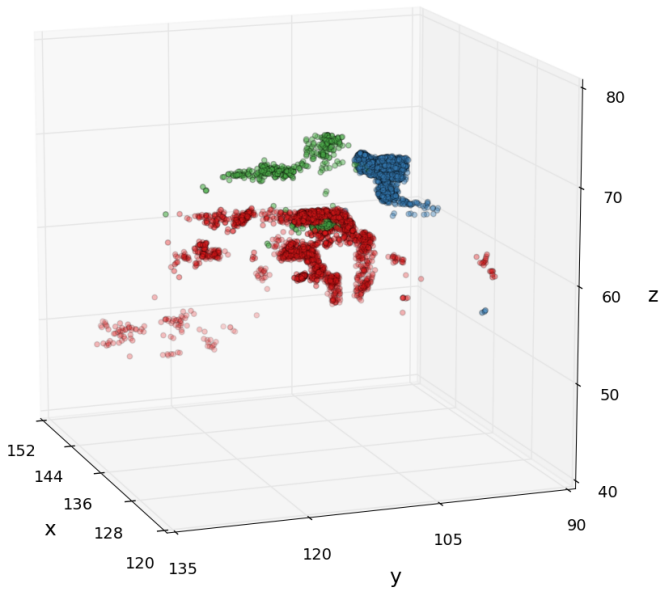


Figure 6: Endpoint data, with background points assigned to the *first-K* foreground clusters with a k-nearest neighbor classifier.

For more control, the level set tree can be estimated by doing the similarity graph, density estimate, tree construction, and pruning in separate steps. Here we use methods in **DeBaCl**'s **utils** module to build a k-nearest neighbor similarity graph `W`, a k-nearest neighbor density estimate `fhat`, a grid of density levels `levels`, and the background observation sets at each density level (`bg_sets`). The `constructTree` method of the **geom_tree** module puts the pieces together to make the tree and the `prune` function removes tree leaf nodes that are small and likely due to random noise.

```
## Similarity graph and density estimate
W, k_radius = utl.knnGraph(X, k, self_edge=False)
fhat = utl.knnDensity(k_radius, n, p, k)


## Tree construction and pruning
bg_sets, levels = utl.constructDensityGrid(fhat, mode='mass', n_grid=None)
tree = gtree.constructTree(W, levels, bg_sets, mode='density', verbose=False)
```

```
tree.prune(method='size-merge', gamma=gamma)
print tree
```

```
     alpha1  alpha2  children   lambda1   lambda2 parent  size
key
0    0.0000  0.0044    [1, 2]  0.000000  0.000004   None  5000
1    0.0044  0.1394  [21, 22]  0.000004  0.000495      0  2971
2    0.0044  0.0664  [11, 12]  0.000004  0.000107      0  2007
11   0.0664  0.3946  [27, 28]  0.000107  0.004922      2  1424
12   0.0664  0.3718        []  0.000107  0.004428      2   303
21   0.1394  0.9978        []  0.000495  0.054786      1  1436
22   0.1394  0.9972        []  0.000495  0.054162      1   834
27   0.3946  0.4016  [29, 30]  0.004922  0.005057     11   870
28   0.3946  0.9992        []  0.004922  0.058173     11   363
29   0.4016  0.9998        []  0.005057  0.061124     27   432
30   0.4016  0.9274        []  0.005057  0.033544     27   431
```

In the definition of density levels and background sets, the `constructDensityGrid` allows the user to specify the `n_grid` parameter to speed up the algorithm by computing the upper level set and connectivity for only a subset of density levels. The `mode` parameter determines whether the grid of density levels is based on evenly-sized blocks of observations (`mode='mass'`) or density levels (`mode='levels'`); we generally prefer the 'mass' mode for our own analyses.

The `mode` parameter of the tree construction function is usually set to be 'density', which treats the underlying function `fhat` as a density or pseudo-density function, with a floor value of 0. This algorithm can be applied to arbitrary functions that do not have a floor value, in which case the mode should be set to 'general'.

4.2. Extension: The Chaudhuri-Dasgupta Tree

Usage of the Chaudhuri-Dasgupta algorithm is similar to the standalone `geomTree` function. First load the **DeBaCl** module **cd_tree** (labeled here for brevity as **cdt**) and the utility functions in **utils**, as well as the data.

```
## Import DeBaCl package
import sys
sys.path.append('/home/brian/Projects/debacl/DeBaCl/')

from debacl import cd_tree as cdt
from debacl import utils as utl

## Import other Python libraries
import numpy as np

## Load the data
X = np.loadtxt('0187_endpoints.csv', delimiter=',')
n, p = X.shape
```

Because the straightforward implementation of the Chaudhuri-Dasgupta algorithm is extremely slow, we use a random subset of only 200 observations (out of the total of 10,000). The smoothing parameter is set to be 2.5% of n, or 5. The pruning parameter is 5% of n, or 10. The pruning parameter is slightly less important for the Chaudhuri-Dasgupta algorithm because the α parameter acts to ensure robustness of the clusters.

```
## Downsample
n_samp = 200
ix = np.random.choice(range(n), size=n_samp, replace=False)
X = X[ix, :]
n, p = X.shape
```

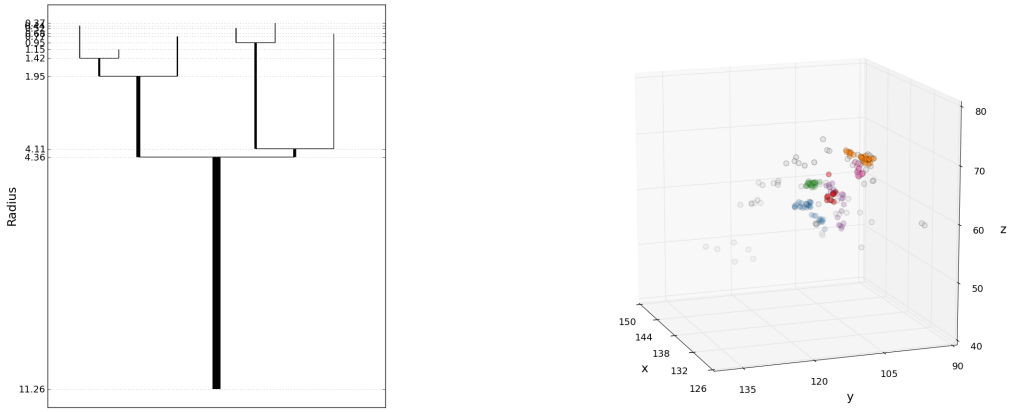
```
## Set level set tree parameters
p_k = 0.025
p_gamma = 0.05
k = int(p_k * n)
gamma = int(p_gamma * n)

## Set plotting parameters
utl.setPlotParams()
```

The straightforward implementation of the Chaudhuri-Dasgupta algorithm starts with a complete graph and removes one edge a time, which is extremely slow. The `start` parameter of the `cdTree` function allows for shortcuts. These are approximations to the method, but are necessary to make the algorithm remotely useful in practice. Currently, the only implemented shortcut is to start with a k-nearest neighbor graph.

```
## Construct the level set tree estimate
tree = cdt.cdTree(X, k, alpha=1.4, start='knn', verbose=False)
tree.prune(method='size-merge', gamma=gamma)
```

As with the geometric tree, we can print a summary of the tree, plot the tree, retrieve foreground cluster labels, and plot the foreground clusters. This is illustrated below for the 'all-mode' labeling method.

```
## Print/make output
print tree

fig = tree.plot()
fig.savefig('../figures/cd_tree.png')

uc, nodes = tree.getClusterLabels(method='all-mode')
```



(a) (b)

Figure 7: The Chaudhuri-Dasgupta tree for the fiber track endpoint data, downsampled from 10,000 to 200 observations to make computation feasible. Foreground clusters based on all-mode clustering are shown on the right.

```
fig, ax = utl.plotForeground(X, uc, fg_alpha=0.6, bg_alpha=0.4, edge_alpha=0.3, s
    =60)
ax.elev = 14; ax.azim=160
fig.savefig('../figures/cd_allmode.png')
```

key	children	parent	r1	r2	size
0	[5, 6]	None	11.261404	4.357185	200
5	[11, 12]	0	4.357185	1.951872	99
6	[7, 8]	0	4.357185	4.110413	74
7	[]	6	4.110413	0.682422	13
8	[25, 26]	6	4.110413	0.951856	61
11	[]	5	1.951872	0.766786	33
12	[15, 16]	5	1.951872	1.416684	51
15	[]	12	1.416684	0.444463	24
16	[]	12	1.416684	1.154000	19
25	[]	8	0.951856	0.517182	27
26	[]	8	0.951856	0.365553	20

4.3. Extension: Functional Data

Nothing in the process of estimating a level set tree requires $\widehat{f}$ to be a bona fide probability density function, and the **DeBaCl** package allows us to use this fact to use level set trees for much more complicated datasets. To illustrate we use the phoneme dataset from Ferraty and Vieu (2006), which contains 400 observations of each of five short speech patterns. Each observation is recorded on a regular grid of 150 frequencies, but we treat this as an approximation of a continuous function over a set frequencies defined on an interval of $\mathbb{R}^1$. Because the observations are random curves they do not have bona fide density functions, but we can still construct a sample level set tree by estimating a pseudo-density function that measures the proximity of each curve to its neighbors.

To start we load the **DeBaCl** modules and the data, which have been pre-smoothed for this example with cubic splines. The true class of each observation is in the last column of the raw data object. The curves for each phoneme are shown in Figure 8.

```
## Import DeBaCl package
import sys
sys.path.append('/home/brian/Projects/debacl/DeBaCl/')

from debacl import geom_tree as gtree
from debacl import utils as utl

## Import other Python libraries
import numpy as np
import scipy.spatial.distance as spdist
import matplotlib.pyplot as plt

## Set plotting parameters
utl.setPlotParams()

## Load data
```

```
speech = np.loadtxt('smooth_phoneme.csv', delimiter=',')
phoneme = speech[:, -1].astype(np.int)
speech = speech[:, :-1]

n, p = speech.shape
```

```
## Plot the curves, separated by true phoneme
fig, ax = plt.subplots(3, 2, sharex=True, sharey=True)
ax = ax.flatten()
ax[-2].set_xlabel('frequencies')
ax[-1].set_xlabel('frequencies')

for g in np.unique(phoneme):
        ix = np.where(phoneme == g)[0]
        for j in ix:
                ax[g].plot(speech[j, :], c='black', alpha=0.15)

fig.savefig('../figures/phoneme_data.png')
```

For functional data we need typically need to define a non-standard distance, precluding the use of the all-in-one function `GeomTree.geomTree` or even the utility function `utils.knnGraph`. First the bandwith and tree pruning parameters are set to be $0.01n$. In the second step all pairwise distances are computed in order to find the k-nearest neighbors for each observation. For simplicity we use Euclidean distance between a pair of curves (which happens to work well), but this is not generally the best distance for functions. Next, the adjacency matrix for a symmetric k-nearest neighbor graph is constructed, which is no different than the finite-dimensional case. Finally the pseudo-density estimator is built by using the finite-dimenisonal k-nearest neighbor density estimator with the dimension set (incorrectly) to 1. This function does not integrate to 1, but the function induces an ordering on the observations (from smallest to largest k-neighbor radius) that is invariant to the dimension.

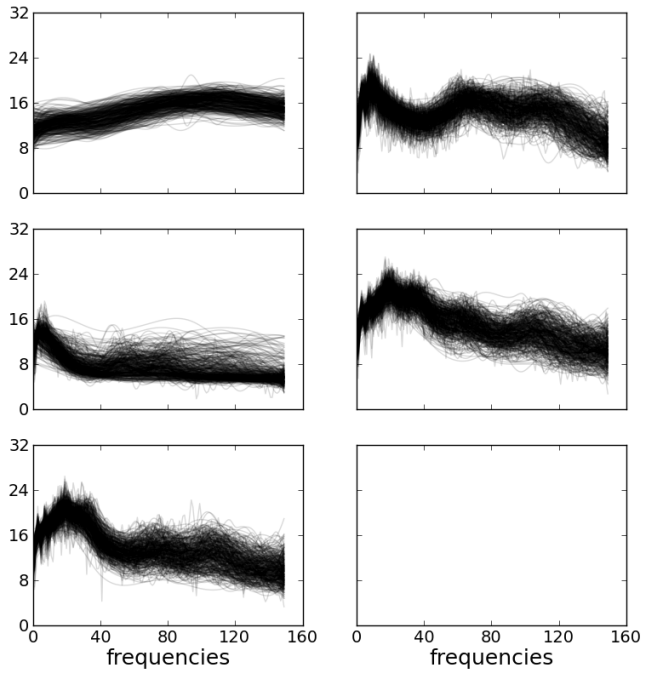


Figure 8: Smoothed waveforms for spoken phonemes, separated by true phoneme.

This ordering is all that is needed for the final step of building the level set tree.

```
## Bandwidth and pruning parameters
p_k = 0.01
p_gamma = 0.01
k = int(p_k * n)
gamma = int(p_gamma * tree.n)

## Find all pairwise distances and the indices of each point's k-nearest neighbors
D = spdist.squareform(spdist.pdist(speech, metric='euclidean'))
rank = np.argsort(D, axis=1)
ix_nbr = rank[:, 0:k]
ix_row = np.tile(np.arange(n), (k, 1)).T

## Construct the similarity graph adjacency matrix
W = np.zeros(D.shape, dtype=np.bool)
```

```
W[ix_row, ix_nbr] = True
W = np.logical_or(W, W.T)
np.fill_diagonal(W, False)

## Compute a pseudo-density estimate and evaluate at each observation
k_nbr = ix_nbr[:, -1]
r_k = D[np.arange(n), k_nbr]
dens = utl.knnDensity(r_k, n, p=1, k=k)

## Build the level set tree
bg_sets, levels = utl.constructDensityGrid(dens, mode='mass', n_grid=None)
tree = dcl.makeLevelSetTree(W, levels, bg_sets, mode='density', verbose=False)
tree.mergeBySize(gamma)
print tree.getSummary()
```

```
     alpha1  alpha2 children   lambda1   lambda2 parent  size
key
0    0.0000  0.2660   [1, 2]  0.000000  0.000261   None  2000
1    0.2660  0.3435   [3, 4]  0.000261  0.000275      0  1125
2    0.2660  1.0000       []  0.000261  0.000938      0   343
3    0.3435  0.4905   [5, 6]  0.000275  0.000307      1   565
4    0.3435  0.7705       []  0.000275  0.000426      1   413
5    0.4905  0.9920       []  0.000307  0.000808      3   391
6    0.4905  0.7110       []  0.000307  0.000382      3    85
```

Once the level set tree is constructed we can plot it and retrieve cluster labels as with finite-dimensional data. In this case we choose the all-mode cluster labeling which produces four clusters that match the true phoneme groups reasonably well.

```
## Retrieve cluster labels
uc, leaves = tree.allModeCluster()

## Level set tree plot
fig = tree.plot(width_mode='mass', color_nodes=leaves)[0]
fig.savefig('../figures/phoneme_tree.png')

## Plot the curves, colored by foreground cluster
palette = plutl.Palette()
fig, ax = plt.subplots()
ax.set_xlabel("frequency index")

for c in np.unique(uc[:,1]):
    ix = np.where(uc[:,1] == c)[0]
    ix_clust = uc[ix, 0]

    for i in ix_clust:
        ax.plot(speech[i,:], c=np.append(palette.colorset[c], 0.25))

fig.savefig('../figures/phoneme_allMode.png')
```

5. Conclusion

The Python package **DeBaCl** for hierarchical density-based clustering provides a highly usable implementation of level set tree estimation and clustering. It improves on existing software through computational efficiency and a high-degree of modularity and customization. Namely, **DeBaCl**:

- offers the first known implementation of the theoretically well-supported Chaudhuri-Dasgupta level set tree algorithm;

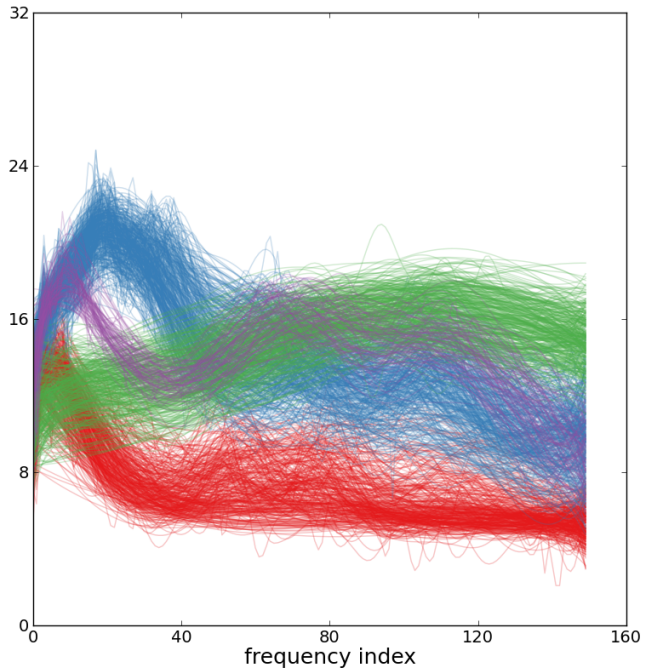


Figure 9: All-mode foreground clusters for the smoothed phoneme data. [Tim: this is a good result. Mention somewhere that this result suggests the method would work well for spike-sorting in electrophysiological data.]

- allows for very general data ordering functions, which are typically probability density estimates but could also be pseudo-density estimates for infinite-dimensional functional data or even arbitrary functions;
- accepts any similarity graph, density estimator, pruning function, cluster labeling scheme, and background point assignment classifier;
- includes the all-mode cluster labeling scheme, which does not require an *a priori* choice of the number of clusters;
- incorporates the λ, α, and κ vertical scales for plotting level set trees, as well as other plotting tweaks to make level set tree plots more interpretable and usable;
- and finally, includes interactive GUI tools for selecting coherent data subsets or high-density clusters based on the level set tree.

The **DeBaCl** package and user manual is available at `https://github.com/CoAxLab/DeBaCl`. The project remains under active development; the focus for the next version will be on improvements in computational efficiency, particularly for the Chaudhuri-Dasgupta algorithm.

Acknowledgments

This research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-10-2-0022. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implies, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for the Government purposes notwithstanding any copyright notation herein. This research was also supported by NSF CAREER grant DMS 114967.

References

Azzalini A, Menardi G (2012). "Clustering via Nonparametric Density Estimation : the R Package pdfCluster." *Technical Report 1*, University of Padua. URL http://cran.r-project.org/web/packages/pdfCluster/index.html.

Azzalini A, Torelli N (2007). "Clustering via nonparametric density estimation." *Statistics and Computing*, **17**(1), 71–80. ISSN 0960-3174. doi:10.1007/s11222-006-9010-y. URL http://www.springerlink.com/index/10.1007/s11222-006-9010-y.

Billingsley P (2012). *Probability and Measure*. Wiley. ISBN 978-1118122372.

Chaudhuri K, Dasgupta S (2010). "Rates of convergence for the cluster tree." In *Advances in Neural Information Processing Systems 23*, pp. 343–351. Vancouver, BC.

Csardi G, Nepusz T (2006). "The igraph Software Package for Complex Network Research." *InterJournal*, **1695**(38).

Ester M, Kriegel Hp, Xu X (1996). "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise." In *Knowledge Discovery and Data Mining*, pp. 226–231.

Ferraty F, Vieu P (2006). *Nonparametric Functional Data Analysis*. Springer. ISBN 9780387303697.

Hartigan J (1975). *Clustering Algorithms*. John Wiley & Sons. ISBN 978-0471356455.

Hartigan JA (1981). "Consistency of Single Linkage for High-Density Clusters." *Journal of the American Statistical Association*, **76**(374), 388–394.

Hennig C (2013). "fpc: Flexible procedures for clustering." URL http://cran.r-project.org/package=fpc.

Hunter JD (2007). "Matplotlib: A 2D Graphics Environment." *Computing in Science & Engineering*, **9**(3), 90–95. URL http://matplotlib.org/.

Jones E, Oliphant T, Peterson P (2001). "SciPy: Open Source Scientific Tools for Python." URL http://www.scipy.org/.

Klemelä J (2004). "Visualization of Multivariate Density Estimates With Level Set Trees." *Journal of Computational and Graphical Statistics*, **13**(3), 599–620. ISSN 1061-8600. doi:10.1198/106186004X2642. URL http://www.tandfonline.com/doi/abs/10.1198/106186004X2642.

Klemelä J (2005). "Algorithms for Manipulation of Level Sets of Nonparametric Density Estimates." *Computational Statistics*, **20**(2), 349–368. doi:10.1007/BF02789708.

Kpotufe S, Luxburg UV (2011). "Pruning nearest neighbor cluster trees." *Proceedings of the 28th International Conference on Machine Learning*, **105**, 225–232.

Lloyd S (1982). "Least squares quantization in PCM." *IEEE Transactions on Information Theory*, **28**(2), 129–137. ISSN 0018-9448. doi:10.1109/TIT.1982.1056489.

MacQueen J (1967). "Some Methods for Classification and And Analysis of Multivariate Observations." *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, **1**, 281–297.

Maier M, Hein M, von Luxburg U (2009). "Optimal construction of k-nearest-neighbor graphs for identifying noisy clusters." *Theoretical Computer Science*, **410**(19), 1749–1764. ISSN 03043975. doi:10.1016/j.tcs.2009.01.009.

Menardi G, Azzalini A (2013). "An Advancement in Clustering via Nonparametric Density Estimation." *Statistics and Computing*. ISSN 0960-3174. doi:10.1007/s11222-013-9400-x. URL http://link.springer.com/10.1007/s11222-013-9400-x.

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

Polonik W (1995). "Measuring Mass Concentrations and Estimating Density Contour Clusters - An Excess Mass Approach." *The Annals of Statistics*, **23**(3), 855–881.

Rinaldo A, Singh A, Nugent R, Wasserman L (2012). "Stability of Density-Based Clustering." *Journal of Machine Learning Research*, **13**, 905–948. arXiv:1011.2771v1.

Shi J, Malik J (2000). "Normalized Cuts and Image Segmentation." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22**(8), 888–905.

Stuetzle W, Nugent R (2010). "A Generalized Single Linkage Method for Estimating the Cluster Tree of a Density." *Journal of Computational and Graphical Statistics*, **19**(2), 397–418. ISSN 1061-8600. doi:10.1198/jcgs.2009.07049. URL http://www.tandfonline.com/doi/abs/10.1198/jcgs.2009.07049.

Wishart D (1969). "Mode analysis: a generalization of nearest neighbor which reduces chaining effects." In AJ Cole (ed.), *Proceedings of the Colloquium on Numerical Taxonomy held in the University of St. Andrews*, pp. 282–308.

Wong A, Lane T (1983). "A kth Nearest Neighbour Clustering Procedure." *Journal of the Royal Statistical Society: Series B*, **45**(3), 362–368.

Affiliation:

Brian P. Kent

Department of Statistics

Carnegie Mellon University

Baker Hall 132

Pittsburgh, PA 15213

E-mail: bpkent@andrew.cmu.edu

URL: http://www.brianpkent.com

Alessandro Rinaldo

Department of Statistics

Carnegie Mellon University

Baker Hall 132

Pittsburgh, PA 15213

E-mail: arinaldo@cmu.edu

URL: http://www.stat.cmu.edu/~arinaldo/

Timothy Verstynen

Department of Psychology & Center for the Neural Basis of Cognition

Carnegie Mellon University

Baker Hall 340U

Pittsburgh, PA 15213

E-mail: timothyv@andrew.cmu.edu

URL: http://www.psy.cmu.edu/~coaxlab/

Journal of Statistical Software http://www.jstatsoft.org/

published by the American Statistical Association http://www.amstat.org/

Volume VV, Issue II *Submitted:* yyyy-mm-dd

MMMMMM YYYY *Accepted:* yyyy-mm-dd