Introduction to the TDA package

Brittany T. Fasy and **Fabrizio Lecci**
In collaboration with the CMU TopStat Group

Abstract

We present a short tutorial and introduction to using the R package **TDA**, which provides some tools for Topological Data Analysis. In particular, it includes implementations of functions that, given some data, provide topological information about the underlying space, such as the distance function, the distance to a measure, the kNN density estimator, the kernel density estimator, and the kernel distance. The salient topological features of the sublevel sets (or superlevel sets) of these functions can be quantified with persistent homology. We provide an R interface for the efficient algorithms of the Dionysus C++ library, including a function for the persistent homology of the Rips filtration, and for the persistent homology of sublevel sets (or superlevel sets) of arbitrary functions evaluated over a grid of points. The significance of the features in the resulting persistence diagrams can be analyzed with functions that implement the methods discussed in Fasy, Lecci, Rinaldo, Wasserman, Balakrishnan, and Singh (2014), Chazal, Fasy, Lecci, Rinaldo, and Wasserman (2013) and Chazal, Fasy, Lecci, Michel, Rinaldo, and Wasserman (2014a). The R package **TDA** also includes the implementation of an algorithm for density clustering, which allows us to identify the spatial organization of the probability mass associated to a density function and visualize it by means of a dendrogram, the cluster tree.

Keywords: Topological Data Analysis, Persistent Homology, Density Clustering.

1. Introduction

Topological Data Analysis (TDA) refers to a collection of methods for finding topological structure in data (Carlsson 2009). Recent advances in computational topology have made it possible to actually compute topological invariants from data. The input of these procedures typically takes the form of a point cloud, regarded as possibly noisy observations from an unknown lower-dimensional set S whose interesting topological features were lost during sampling. The output is a collection of data summaries that are used to estimate the topological features of S.

In Section 2, we describe how to compute some widely studied functions that, starting from a point cloud, provide some topological information about the underlying space: the distance function (`distFct`), the distance to a measure function (`dtm`), the k Nearest Neighbor density estimator (`knnDE`), the kernel density estimator (`kde`), and the kernel distance (`kernelDist`).

One approach to TDA is persistent homology (Edelsbrunner and Harer 2010), a method for studying the homology at multiple scales simultaneously. More precisely, it provides a framework and efficient algorithms to quantify the evolution of the topology of a family of nested topological spaces. Given a real-valued function f, such as the ones described in Section 2, persistent homology describes how the topology of the lower level sets $\{x : f(x) \le t\}$

(or superlevel sets $\{x : f(x) \geq t\}$) changes as t increases from $-\infty$ to ∞ (or decreases from ∞ to $-\infty$). This information is encoded in the persistence diagram, a multiset of points in the plane, each corresponding to the birth and death of a homological feature that existed for some interval of t. In Section 3, we describe how to compute persistence diagrams of sublevel sets (or superlevel sets) of functions evaluated over a grid of points, using `gridDiag`, and how to compute the persistence diagram of the Rips filtration built on top of a point cloud, using `ripsDiag`. Our implementation relies on **Dionysus**, a `C++` library developed by Dmitriy Morozov and freely available at http://mrzv.org/software/dionysus/ .

One of the key challenges in persistent homology is to find a way to isolate the points of the persistence diagram representing the topological noise. In Fasy *et al.* (2014), Chazal *et al.* (2013), and Chazal *et al.* (2014a), we propose several statistical methods to construct confidence sets for persistence diagrams and other summary functions that allow us to separate topological signal from topological noise. Some of these methods are implemented and described in Section 3.

Finally, the **TDA** package provides the implementation of an algorithm for density clustering. This method allows us to identify and visualize the spatial organization of the data, without specific knowledge about the data generating mechanism and in particular without any a priori information about the number of clusters. In Section 4, we describe the function `clusterTree`, that, given a density estimator, encodes the hierarchy of the connected components of its superlevel sets into a dendrogram, the cluster tree. (Kpotufe and von Luxburg 2011; Kent 2013).

2. Distance Functions and Density Estimators

As a first toy example to using the **TDA** package, we show how to compute distance functions and density estimators over a grid of points. The setting is the typical one in TDA: a set of points $X = \{x_1, \ldots, x_n\}$ has been sampled from some distribution P supported on a manifold and we are interested in recovering the topological features of the underlying manifold by studying some functions of the data. The following code generates a sample of 400 points from the unit circle and constructs a grid of points over which we will evaluate the functions.

```
> library("TDA")
> X = circleUnif(400)
> Xlim=c(-1.6, 1.6);   Ylim=c(-1.7, 1.7);   by=0.065
> Xseq=seq(Xlim[1], Xlim[2], by=by)
> Yseq=seq(Ylim[1], Ylim[2], by=by)
> Grid=expand.grid(Xseq,Yseq)
```

The **TDA** package provides implementations of the following functions:

- The distance function is defined for each $y \in \mathbb{R}^d$ as $\Delta(y) = \inf_{x \in X} \|x - y\|_2$ and is computed for each point of the `Grid` with the following code:

```
> distance= distFct(X=X, Grid=Grid)
```

- Given a probability measure P, the distance to measure (DTM) is defined for each $y \in \mathbb{R}^d$ as

$$d_{m_0}(y) = \sqrt{\frac{1}{m_0} \int_0^{m_0} (G_y^{-1}(u))^2 du},$$

where $G_y(t) = P(\|X - y\| \leq t)$ and $0 < m_0 < 1$ is a smoothing parameter. For more details see Chazal, Cohen-Steiner, and Mérigot (2011).
Given $X = \{x_1, \ldots, x_n\}$, the empirical version of the DTM is

$$\hat{d}_{m_0}(y) = \sqrt{\frac{1}{k} \sum_{x_i \in N_k(y)} \|x_i - y\|^2},$$

where $k = \lceil m_0 n \rceil$ and $N_k(y)$ is the set containing the k nearest neighbors of y among $x_1, \ldots, x_n$. The DTM is computed for each point of the `Grid` with the following code:

```
> m0=0.1
> DTM=dtm(X=X, Grid=Grid, m0=m0)
```

- The k Nearest Neighbor density estimator, for each $y \in \mathbb{R}^d$, is defined by

$$\hat{\delta}_k(y) = \frac{k}{n \ v_d \ r_k^d(y)},$$

where v_n is the volume of the Euclidean d dimensional unit ball and $r_k^d(x)$ is the Euclidean distance form point x to its kth closest neighbor among the points of X. It is computed for each point of the `Grid` with the following code:

```
> k=60
> kNN=knnDE(X=X, Grid=Grid, k=k)
```

- The Gaussian Kernel Density Estimator (KDE), for each $y \in \mathbb{R}^d$, is defined by

$$\hat{p}_h(y) = \frac{1}{n(\sqrt{2\pi}h)^d} \sum_{i=1}^n \exp\left(\frac{-\|y - x_i\|_2^2}{2h^2}\right).$$

where h is a smoothing parameter. It is computed for each point of the `Grid` with the following code:

```
> h=0.3
> KDE= kde(X=X, Grid=Grid, h=h)
```

- The Kernel distance estimator, for each $y \in \mathbb{R}^d$, is defined by

$$\hat{\kappa}_h(y) = \sqrt{\frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n K_h(x_i, x_j) + K_h(y, y) - 2\frac{1}{n} \sum_{i=1}^n K_h(y, x_i)},$$

where $K_h(x, y) = \exp\left(\frac{-\|x-y\|_2^2}{2h^2}\right)$ is the Gaussian Kernel with smoothing parameter h. The Kernel distance is computed for each point of the `Grid` with the following code:

```
> h=0.3
> Kdist= kernelDist(X=X, Grid=Grid, h=h)
```

For this 2 dimensional example, we can visualize the functions using `persp` form the **graphics** package. For example the following code produces the KDE plot in Figure 1:

```
> persp(Xseq,Yseq,matrix(KDE,ncol=length(Yseq), nrow=length(Xseq)),
+ xlab="",ylab="",zlab="",theta=-20,phi=35, ltheta=50, col=2, border=NA,
+ main="KDE", d=0.5, scale=FALSE, expand=3, shade=0.9)
```
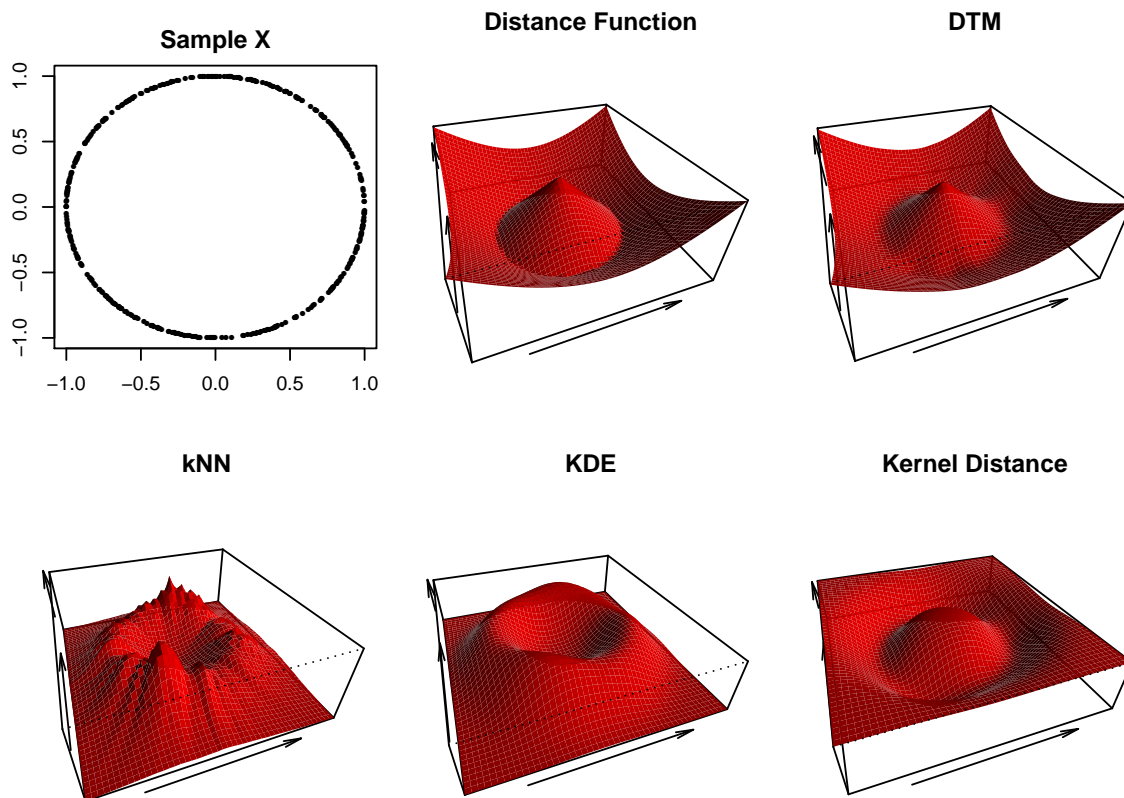

Figure 1: distance functions and density estimators evaluated over a grid of points.

2.1. Bootstrap Confidence Bands

We can construct a $(1 - \alpha)$ confidence band for a function using the bootstrap algorithm, which we briefly describe using the kernel density estimator:

1. Given a sample $X = \{x_1, \ldots, x_n\}$, compute the kernel density estimator $\hat{p}_h$;

2. Draw $X^* = \{x_1^*, \ldots, x_n^*\}$ from $X = \{x_1, \ldots, x_n\}$ (with replacement), and compute $\theta^* = \sqrt{n}\|\hat{p}_h^*(x) - \hat{p}_h(x)\|_\infty$, where $\hat{p}_h^*$ is the density estimator computed using X^*;

3. Repeat the previous step B times to obtain $\theta_1^*, \ldots, \theta_B^*$;

4. Compute $q_\alpha = \inf \left\{ q : \frac{1}{B} \sum_{j=1}^{B} I(\theta_j^* \geq q) \leq \alpha \right\}$;

5. The $(1 - \alpha)$ confidence band for $\mathbb{E}[\hat{p}_h]$ is $\left[\hat{p}_h - \frac{q_\alpha}{\sqrt{n}} , \hat{p}_h + \frac{q_\alpha}{\sqrt{n}} \right]$.

Fasy *et al.* (2014) and Chazal *et al.* (2014a) prove the validity of the bootstrap algorithm for kernel density estimators, distance to measure, and kernel distance, and use it in the framework of persistent homology. The bootstrap algorithm is implemented in the function `bootstrapBand`, which provides the option of parallelizing the algorithm (`parallel=TRUE`) using the package **parallel**. The following code computes a 90% confidence band for $\mathbb{E}[\hat{p}_h]$, showed in Figure 2.

```
> band=bootstrapBand(X=X, FUN=kde, Grid=Grid, B=100,
+                    parallel=F, alpha=0.1, h=h)
```
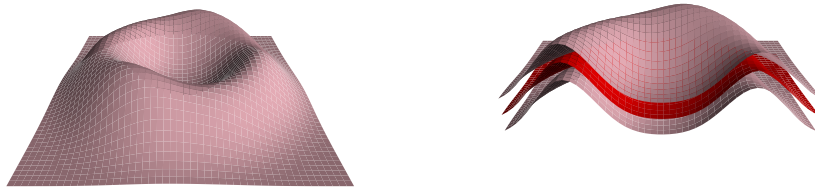

Figure 2: the 90% confidence band for $\mathbb{E}[\hat{p}_h]$ has the form $[\ell, u] = [\hat{p}_h - q_\alpha/\sqrt{n} , \hat{p}_h + q_\alpha/\sqrt{n}]$. The plot on the right shows a section of the functions: the red surface is the KDE $\hat{p}_h$; the pink surfaces are ℓ and u.

3. Persistent Homology

We provide an informal description of the implemented methods of persistent homology. We assume the reader is familiar with the basic concepts and, for a rigorous exposition, we refer to the textbook Edelsbrunner and Harer (2010).

3.1. Persistent Homology Over a Grid

In this section, we describe how to use the `gridDiag` function to compute the persistent homology of sublevel (and superlevel) sets of the functions described in Section 2. The function `gridDiag` evaluates a given real valued function over a triangulated grid, constructs a filtration of simplices using the values of the function, and computes the persistent homology of the filtration. `gridDiag` currently supports grids in dimension 1,2, and 3. Future releases of the **TDA** package will extend its functionalities.

The following code computes the persistent homology of the superlevel sets (`sublevel=FALSE`) of the kernel density estimator (`FUN=kde, h=0.3`) using the point cloud stored in the matrix `X` from the previous example. The same code would work for the other functions defined in Section 2 (it is sufficient to replace `kde` and its smoothing parameter `h` with another function and its parameters). The function `gridDiag` returns an object of the class `"diagram"`. The other inputs are the features of the grid over which the `kde` is evaluated (`Xlim`, `Ylim` and `by`), the smoothing parameter `h`, and a logical variable that indicates whether a progress bar should be printed (`printStatus`).

```
> Diag = gridDiag( X=X, FUN=kde, h=0.3, Xlim=Xlim, Ylim=Ylim, by=by,
+                  sublevel=F, printStatus=F )
```

We plot the data and the the diagram, using the function `plot`, implemented as a standard S3 method for objects of the class `"diagram"`. The following command produces the third plot in Figure 3.

```
> plot(Diag, band=2*band$width, main="KDE Diagram")
```
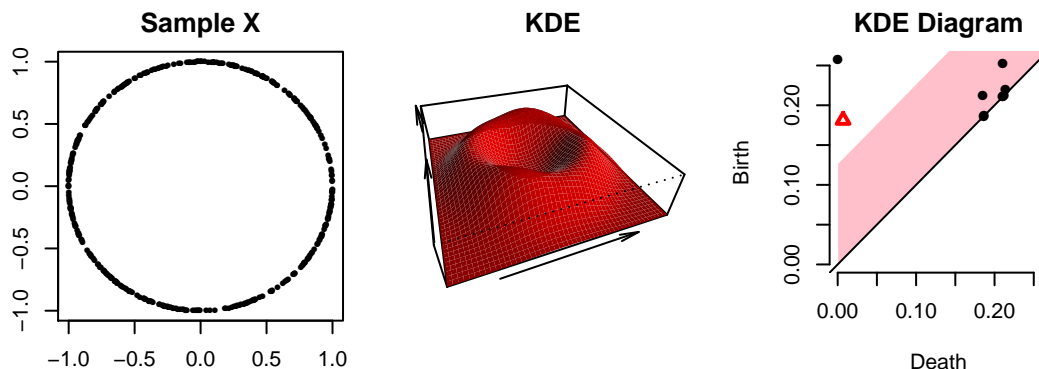

Figure 3: The plot on the right shows the persistence diagram of the superlevel sets of the KDE. Black points represent connected components and red triangles represent loops. The features are born at high levels of the density and die at lower levels. The pink 90% confidence band separates significant features from noise.

The option `band=2*band$width` produces a pink confidence band for the persistence diagram, using the confidence band constructed for the corresponding kernel density estimator in the previous section. The features above the band can be interpreted as representing significant homological features, while points in the band are not significantly different from noise. The validity of the bootstrap confidence band for persistence diagrams of KDE, DTM, and Kernel Distance derive from the *Stability Theorem* (Chazal, de Silva, Glisse, and Oudot 2012) and is discussed in detail in Fasy *et al.* (2014) and Chazal *et al.* (2014a).

The function `plot` for the class `"diagram"` provide the options of rotating the diagram (`rotated=TRUE`), drawing the barcode in place of the diagram (`barcode=TRUE`), as well as other standard graphical options:

```
> par(mfrow=c(1,2), mai=c(0.8,0.8,0.3,0.1))
> plot(Diag, rotated=T, band=band$width, main="Rotated Diagram")
> plot(Diag, barcode=T,  main="Barcode")
```
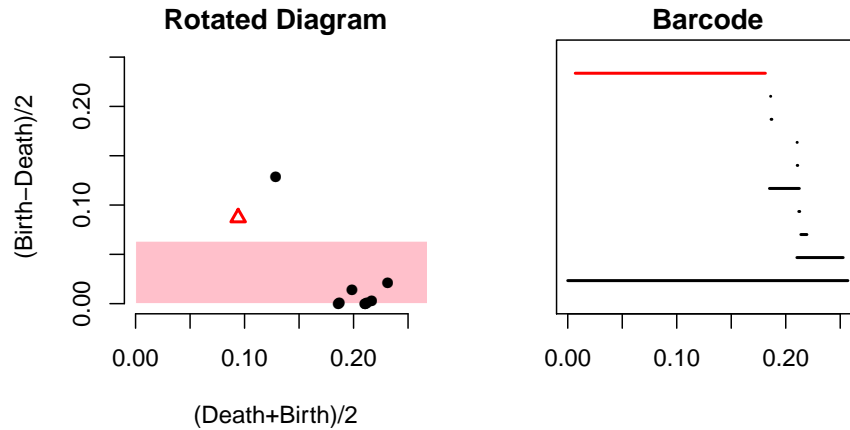

Figure 4: Rotated Persistence Diagram and Barcode

3.2. Rips Diagrams

The *Vietoris-Rips complex* $R(X, \varepsilon)$ consists of simplices with vertices in X and diameter at most ε. In other words, a simplex σ is included in the complex if each pair of vertices in σ is at most ε apart. The sequence of Rips complexes obtained by gradually increasing the radius ε creates a filtration.

The `ripsDiag` function computes the persistence diagram of the Rips filtration built on top of a point cloud. This function is an R wrapper of the function "rips-pairwise" of the **Dionysus** C++ library.

The following code generates 60 points from two circles:

```
> Circle1 = circleUnif(60)
> Circle2 = circleUnif(60, r=2) +3
> Circles=rbind(Circle1,Circle2)
```

We specify the limit of the Rips filtration and the max dimension of the homological features we are interested in (0 for components, 1 for loops, 2 for voids...):

```
> maxscale=5            # limit of the filtration
> maxdimension=1        # components and loops
```

and we generate the persistence diagram:

```
> Diag=ripsDiag(X=Circles, maxdimension, maxscale, printStatus=F)
```

Alternatively, using the option `dist="arbitrary"` in `ripsDiag()`, the input X can be an $n \times n$ matrix of distances. This option is useful when the user wants to consider a Rips filtration

constructed using an arbitrary distance.

Finally we plot the data and the the diagram:

```
> par(mfrow=c(1,2), mai=c(0.8,0.8,0.3,0.3))
> plot(Circles, pch=16, xlab="",ylab="")
> plot(Diag)
```
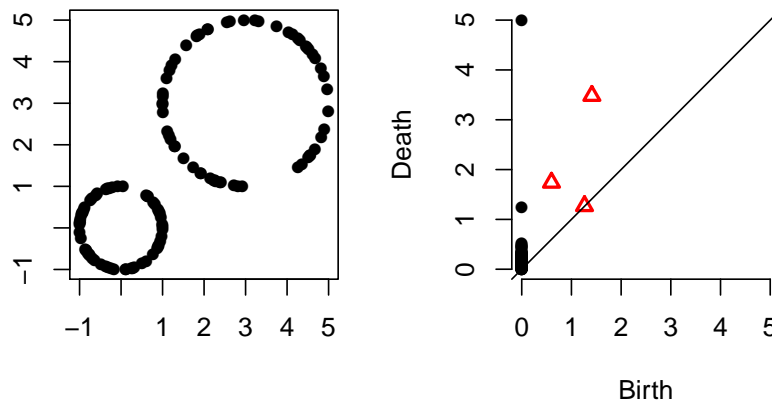

Figure 5: Rips persistence diagram. Black points represent connected components and red triangles represent loops.

3.3. Bottleneck and Wasserstein Distances

Standard metrics for measuring the distance between two persistence diagrams are the bottleneck distance and the pth Wasserstein distance (Edelsbrunner and Harer 2010). The **TDA** package includes the functions `bottleneck` and `wasserstein`, which are R wrappers of the functions "bottleneck_distance" and "wasserstein_distance" of the **Dionysus** C++ library .

We generate two persistence diagrams of the Rips filtrations built on top of the two (separate) circles of the previous example,

```
> Diag1=ripsDiag(Circle1,maxdimension=1, maxscale=5, printStatus=F)
> Diag2=ripsDiag(Circle2,maxdimension=1, maxscale=5, printStatus=F)
```

and we compute the bottleneck distance and the 2nd Wasserstein distance between the two diagrams. In the following code, the option `dimension=1` specifies that the distances between diagrams are computed using only one dimensional features (loops).

```
> print( bottleneck(Diag1, Diag2, dimension=1) )

[1] 1.035405

> print( wasserstein(Diag1, Diag2, p=2, dimension=1) )

[1] 1.397373
```

3.4. Landscapes and Silhouettes

Persistence landscapes and silhouettes are real-valued functions that further summarize the information contained in a persistence diagram. They have been introduced and studied in Bubenik (2012), Chazal *et al.* (2013), and Chazal, Fasy, Lecci, Michel, Rinaldo, and Wasserman (2014b). We briefly introduce the two functions.

Landscape. The persistence landscape is a collection of continuous, piecewise linear functions $\lambda \colon \mathbb{Z}^+ \times \mathbb{R} \to \mathbb{R}$ that summarizes a persistence diagram. To define the landscape, consider the set of functions created by tenting each each point $p = (x, y) = \left(\frac{b+d}{2}, \frac{d-b}{2}\right)$ representing a birth-death pair (b, d) in the persistence diagram D as follows:

$$\Lambda_p(t) = \begin{cases} t - x + y & t \in [x - y, x] \\ x + y - t & t \in (x, x + y] \\ 0 & \text{otherwise} \end{cases} = \begin{cases} t - b & t \in [b, \frac{b+d}{2}] \\ d - t & t \in (\frac{b+d}{2}, d] \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

We obtain an arrangement of piecewise linear curves by overlaying the graphs of the functions $\{\Lambda_p\}_p$; see Figure 6. The persistence landscape of D is a summary of this arrangement. Formally, the persistence landscape of D is the collection of functions

$$\lambda(k, t) = k\max_p \Lambda_p(t), \quad t \in [0, T], k \in \mathbb{N}, \tag{2}$$

where $k$max is the kth largest value in the set; in particular, 1max is the usual maximum function.

Silhouette. Consider a persistence diagram with N off diagonal points. For every $0 < p < \infty$ we define the power-weighted silhouette

$$\phi^{(p)}(t) = \frac{\sum_{j=1}^{N} |d_j - b_j|^p \Lambda_j(t)}{\sum_{j=1}^{N} |d_j - b_j|^p}.$$

The value p can be thought of as a trade-off parameter between uniformly treating all pairs in the persistence diagram and considering only the most persistent pairs. Specifically, when p is small, $\phi^{(p)}(t)$ is dominated by the effect of low persistence features. Conversely, when p is large, $\phi^{(p)}(t)$ is dominated by the most persistent features; see Figure 6 (right).

The landscape and silhouette functions can be evaluated over a one-dimensional grid of points `tseq` using the functions `landscape` and `silhouette`. In the following code, we use the persistence diagram from Figure 5 to construct the corresponding landscape and silhouette for one-dimensional features (`dimension=1`). The option `KK=1` specifies that we are interested in the 1st landscape function, and `p=1` is the power of the weights in the definition of the silhouette function.

```
> tseq=seq(0,maxscale, length=1000)    #domain
> Land= landscape(Diag, dimension=1, KK=1, tseq)
> Sil=silhouette(Diag, p=1,   dimension=1, tseq)
```

The functions `landscape` and `silhouette` return real valued vectors, which can be simply plotted with `plot(tseq, Land, type="l"); plot(tseq, Sil, type="l")`. See Figure 7.
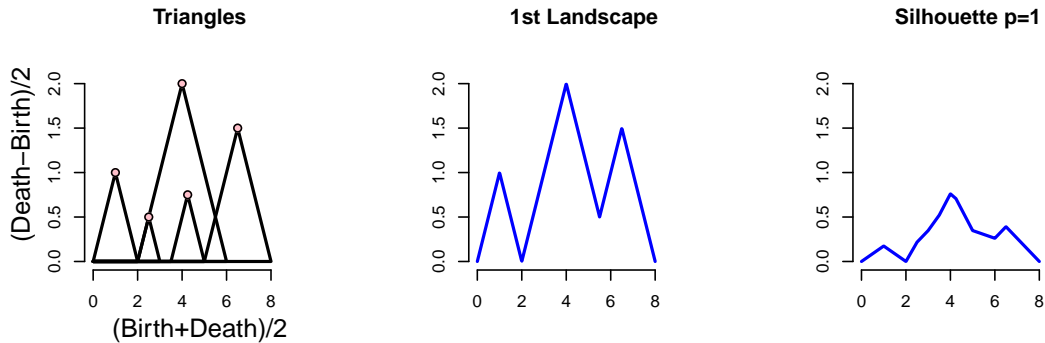
Figure 6: Left: we use the rotated axes to represent a persistence diagram D. A feature $(b, d) \in D$ is represented by the point $(\frac{b+d}{2}, \frac{d-b}{2})$ (pink). In words, the x-coordinate is the average parameter value over which the feature exists, and the y-coordinate is the half-life of the feature. Middle: the blue curve is the landscape $\lambda(1, \cdot)$. Right: the blue curve is the silhouette $\phi^{(1)}(\cdot)$.
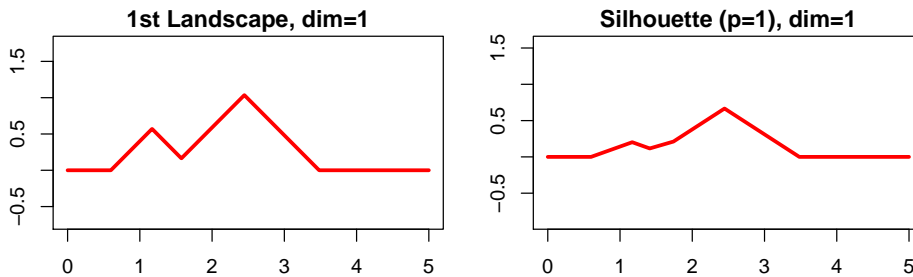

Figure 7: Landscape and Silhouette of the one-dimensional features of the diagram of Figure 5.

3.5. Confidence Bands for Landscapes and Silhouettes

Recent results in Chazal *et al.* (2013) and Chazal *et al.* (2014b) show how to construct confidence bands for landscapes and silhouettes, using a bootstrap algorithm (multiplier bootstrap). This strategy is useful in the following scenario. We have a very large dataset with N points. There is a diagram D and landscape λ corresponding to some filtration built on the data. When N is large, computing D is prohibitive. Instead, we draw n subsamples, each of size m. We compute a diagram and landscape for each subsample yielding landscapes $\lambda_1, \ldots, \lambda_n$. (Assuming m is much smaller than N, these subsamples are essentially independent and identically distributed.) Then we compute $\frac{1}{n} \sum_i \lambda_i$, an estimate of $\mathbb{E}(\lambda_i)$, which can be regarded as an approximation of λ. The function `multipBootstrap` uses the landscapes $\lambda_1, \ldots, \lambda_n$ to construct a confidence band for $\mathbb{E}(\lambda_i)$. The same strategy is valid for silhouette functions. We illustrate the method with a simple example.

First we sample N points from two circles:

```
> N=4000
> XX1 = circleUnif(N/2)
> XX2 = circleUnif(N/2, r=2) +3
```

```
> X=rbind(XX1,XX2)
```

Then we specify the number of subsamples n, the subsample size m, and we create the objects that will store the n diagrams and landscapes:

```
> m=80          # subsample size
> n=10          # we will compute n landscapes using subsamples of size m
> tseq=seq(0,maxscale, length=500)       #domain of landscapes
> Diags=list()                           #here we store n Rips diags
> Lands=matrix(0,nrow=n, ncol=length(tseq)) #here we store n landscapes
```

For n times, we subsample from the large point cloud, compute n Rips diagrams and the corresponding 1st landscape functions (KK=1), using 1 dimensional features (`dimension=1`):

```
> for (i in 1:n){
+         subX=X[sample(1:N,m),]
+         Diags[[i]]=ripsDiag(subX,maxdimension=1, maxscale=5)
+         Lands[i,]=landscape(Diags[[i]], dimension=1, KK=1, tseq )
+ }
```

Finally we use the n landscapes to construct a 95% confidence band for the mean landscape

```
> bootLand=multipBootstrap(Lands,B=100,alpha=0.05, parallel=F)
```

which is plotted by the following code. See Figure 8.

```
> plot(tseq, bootLand$mean, main="Mean Landscape with 95% band")
> polygon(c(tseq, rev(tseq)),
+         c(bootLand$band[,1],rev(bootLand$band[,2])), col="pink")
> lines(tseq, bootLand$mean, lwd=2, col=2)
```

3.6. Selection of Smoothing Parameters

An unsolved problem in topological inference is how to choose the smoothing parameters, for example h for KDE and m for DTM.

Chazal *et al.* (2014a) suggest the following method, that we describe here for the kernel density estimator, but works also for the kernel distance and the distance to measure.

Let $\ell_1(h), \ell_2(h), \ldots,$ be the lifetimes of the features of a persistence diagram at scale h. Let $q_\alpha(h)/\sqrt{n}$ be the width of the confidence band for the kernel density estimator at scale h, as described in Section 2.1. We define two quantities that measure the amount of significant information at level h:

- The number of significant features, $N(h) = \# \left\{ i : \ell(i) > 2\frac{q_\alpha(h)}{\sqrt{n}} \right\}$;

- The total significant persistence, $S(h) = \sum_i \left[\ell_i - 2\frac{q_\alpha(h)}{\sqrt{n}} \right]_+$.

Figure 8: 95% confidence band for the mean landscape function.

These measures are small when h is small since $q_\alpha(h)$ is large. On the other hand, they are small when h is large since then all the features of the KDE are smoothed out. Thus we have a kind of topological bias-variance tradeoff. We choose h to maximize $N(h)$ or $S(h)$.

The method is implemented in the function `maxPersistence`, as shown in the following toy example. First, we sample 1600 point from two circles (plus some clutter noise) and we specify the limits of the grid over which the KDE will be evaluated:

```
> XX1 = circleUnif(600)
> XX2 = circleUnif(1000, r=1.5) +2.5
> noise=cbind(runif(80, -2,5),runif(80, -2,5))
> X=rbind(XX1,XX2, noise)
> # Grid limits
> Xlim=c(-2,5)
> Ylim=c(-2,5)
> by=0.2
```

Then we specify a sequence of smoothing parameters among which we will select the optimal one, the number of bootstrap iterations and the level of the confidence bands to be computed:

```
> parametersKDE=seq(0.1,0.6,by=0.05)
> B=50        # number of bootstrap iterations. Should be large.
> alpha=0.1   # level of the confidence bands
```

The function `maxPersistence` can be parallelized (`parallel=TRUE`) and a progress bar can be printed (`printProgress=TRUE`):

```
> maxKDE=maxPersistence(kde, parametersKDE, X, Xlim, Ylim, Zlim=NA,
+                       by=by, sublevel=F, B=B, alpha=alpha,
+                       parallel=TRUE, printProgress=TRUE)
```

```
Progress: 0.09  0.18  0.27  0.36  0.45  0.55  0.64  0.73  0.82  0.91  1
```

The S3 methods `summary` and `plot` are implemented for the class `"maxPersistence"`. We can display the values of the parameters that maximize the two criteria:

```
> print(summary(maxKDE))

Call:
maxPersistence(FUN = kde, parameters = parametersKDE, X = X,
    Xlim = Xlim, Ylim = Ylim, Zlim = NA, by = by, sublevel = F,
    B = B, alpha = alpha, parallel = TRUE, printProgress = TRUE)

The number of significant features is maximized by
[1] 0.25 0.30 0.35

The total significant persistence is maximized by
[1] 0.15
```

and produce the summary plot of Figure 9:

```
> par(mfrow=c(1,2), mai=c(0.8,0.8,0.35,0.3))
> plot(X, pch=16, cex=0.5, main="Two Circles")
> plot(maxKDE, main="Max Persistence - KDE")
```
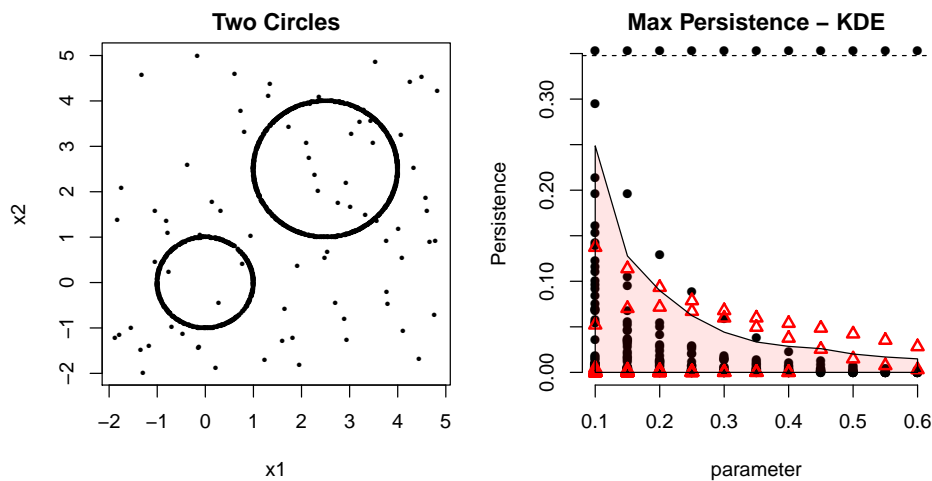

Figure 9: Max Persistence Method for the selection of smoothing parameters. For each value of the smoothing parameter we display the persistence of the corresponding homological features, along with a (pink) confidence band that separates the statistically significant features from the topological noise.

4. Density Clustering

The last example of this vignette illustrates the use of the function `clusterTree`, which is an implementation of Algorithm 1 in Kent, Rinaldo, and Verstynen (2013).

First, we briefly describe the task of density clustering; we defer the reader to Kent (2013) for a more rigorous and complete description. Let f be the density of the probability distribution P generating the observed sample $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^D$. For a threshold value $\lambda > 0$, the corresponding super level set of f is $L_f(\lambda) := \mathrm{cl}(\{x \in \mathbb{R}^D : f(x) > \lambda\})$, and its D-dimensional subsets are called high-density regions. The high-density clusters of P are the maximal connected subsets of $L_f(\lambda)$. By considering all the level sets simultaneously (from $\lambda = 0$ to $\lambda = \infty$), we can record the evolution and the hierarchy of the high-density clusters of P. This naturally leads to the notion of the cluster density tree of P (see, e.g., Hartigan (1981)), defined as the collection of sets $T := \{L_f(\lambda), \lambda \geq 0\}$, which satisfies the tree property: $A, B \in T$ implies that $A \subset B$ or $B \subset A$ or $A \cap B = \emptyset$. We will refer to this construction as the λ-tree. Alternatively, Kent *et al.* (2013) introduced the κ-*tree*, which facilitates the interpretation of the tree by precisely encoding the probability content of each tree branch rather than the density level. Cluster trees are particularly useful for high dimensional data, whose spatial organization is difficult to represent.

We illustrate the strategy with a simple example. First we generate a 2D point cloud from three (not so well) separated clusters (see top left plot of Figure 10):

```
> X1=cbind(rnorm(300,1,.8),rnorm(300,5,0.8))
> X2=cbind(rnorm(300,3.5,.8),rnorm(300,5,0.8))
> X3=cbind(rnorm(300,6,1),rnorm(300,1,1))
> XX=rbind(X1,X2,X3)
```

Then we use the function `clusterTree` to compute cluster trees using the k Nearest Neighbors density estimator (`k=100` nearest neighbors) and the Gaussian kernel density estimator, with smoothing parameter `h`.

```
> Tree=clusterTree(XX, k=100, density="knn", printStatus=F)
> TreeKDE=clusterTree(XX, k=100, h=0.3, density="kde", printStatus=F)
```

Note that, even when `kde` is used to estimate the density, we have to provide the option `k=100`, so that the algorithm can compute the connected components at each level of the density using a k Nearest Neighbors graph.

The `"clusterTree"` objects `Tree` and `TreeKDE` contain information about both the λ-tree and the κ-tree. The function `plot` for objects of the class `"clusterTree"` produces the plots in Figure 10.

```
> plot(Tree, type="lambda", main="lambda Tree (knn)")
> plot(Tree, type="kappa", main="kappa Tree (knn)")
> plot(TreeKDE, type="lambda", main="lambda Tree (kde)")
> plot(TreeKDE, type="kappa", main="kappa Tree (kde)")
```
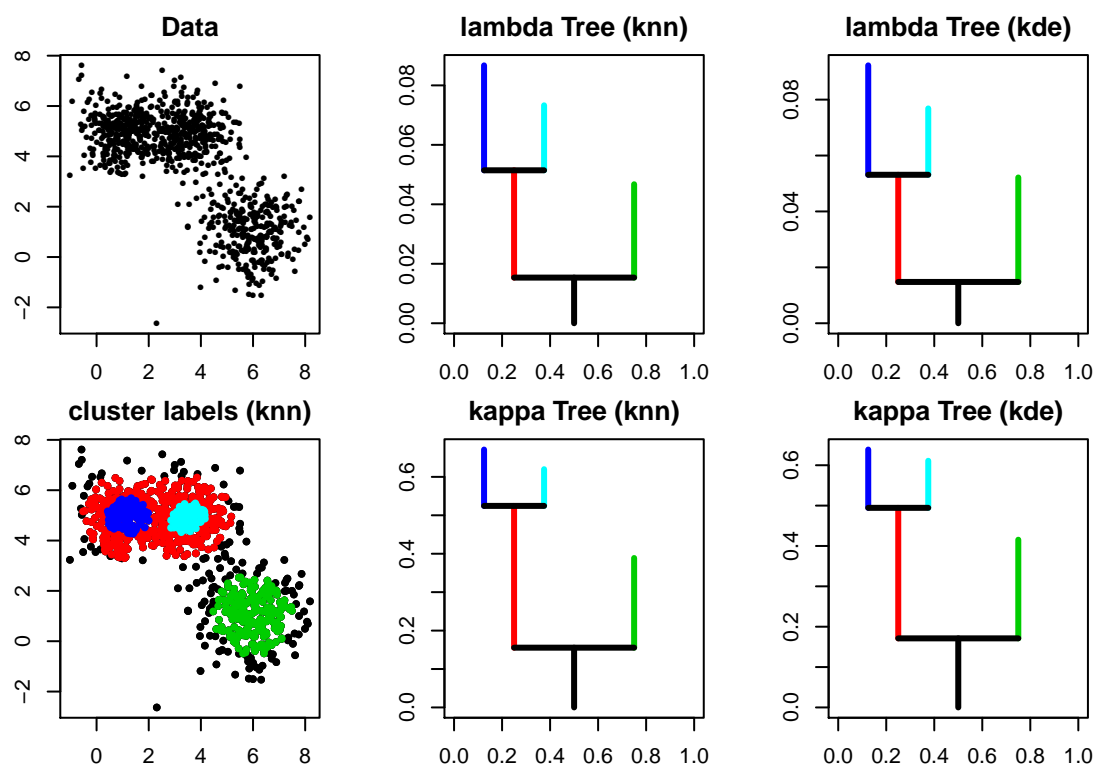
Figure 10: The lambda trees and kappa trees of the k Nearest Neighbor density estimator and the kernel density estimator.

References

Bubenik P (2012). "Statistical topological data analysis using persistence landscapes." *arXiv preprint arXiv:1207.6437*.

Carlsson G (2009). "Topology and data." *Bulletin of the American Mathematical Society*, **46**(2), 255–308.

Chazal F, Cohen-Steiner D, Mérigot Q (2011). "Geometric inference for probability measures." *Foundations of Computational Mathematics*, **11**(6), 733–751.

Chazal F, de Silva V, Glisse M, Oudot S (2012). "The structure and stability of persistence modules." *arXiv preprint arXiv:1207.3674*.

Chazal F, Fasy BT, Lecci F, Michel B, Rinaldo A, Wasserman L (2014a). "Robust Topological Inference: Distance-To-a-Measure and Kernel Distance." *Technical Report*.

Chazal F, Fasy BT, Lecci F, Michel B, Rinaldo A, Wasserman L (2014b). "Subsampling Methods for Persistent Homology." *arXiv preprint arXiv:1406.1901*.

Chazal F, Fasy BT, Lecci F, Rinaldo A, Wasserman L (2013). "Stochastic convergence of persistence landscapes and silhouettes." *arXiv preprint arXiv:1312.0308*.

Edelsbrunner H, Harer J (2010). *Computational topology: an introduction.* American Mathematical Society.

Fasy BT, Lecci F, Rinaldo A, Wasserman L, Balakrishnan S, Singh A (2014). "Confidence Sets For Persistence Diagrams." *To appear in The Annals of Statistics.*

Hartigan JA (1981). "Consistency of single linkage for high-density clusters." *Journal of the American Statistical Association*, **76**(374), 388–394.

Kent B (2013). *Level Set Trees for Applied Statistics.* Ph.D. thesis, Department of Statistics, Carnegie Mellon University.

Kent BP, Rinaldo A, Verstynen T (2013). "DeBaCl: A Python Package for Interactive DEnsity-BAsed CLustering." *arXiv preprint arXiv:1307.8136.*

Kpotufe S, von Luxburg U (2011). "Pruning nearest neighbor cluster trees." In *International Conference on Machine Learning (ICML).*

Affiliation:

Brittany T. Fasy
Computer Science Department
Tulane University
Email: brittany.fasy@alumni.duke.edu

Fabrizio Lecci
Department of Statistics
Carnegie Mellon University
Email: lecci@cmu.edu

TopStat Group
Carnegie Mellon University
Website: http://www.stat.cmu.edu/topstat/
Email: topstat@stat.cmu.edu