Tutorial on the R package TDA

Jisu Kim
Brittany T. Fasy, Jisu Kim, Fabrizio Lecci, Clément Maria, Vincent Rouvreau

Abstract

I present a short tutorial and introduction to using the R package TDA, which provides tools for Topological Data Analysis. Given data, the salient topological features of underlying space can be quantified with persistent homology. TDA package provides a function for the persistent homology of the Rips filtration, and a function for the persistent homology of sublevel sets (or superlevel sets) of arbitrary functions evaluated over a grid of points. Some common choice of functions for the latter case, such as the distance function, the distance to a measure, the kNN density estimator, the kernel density estimator, and the kernel distance, are implemented in the TDA package. The R package TDA also provides a function for computing the confidence band that determines significance of the features in the resulting persistence diagrams.

Keywords: Topological Data Analysis, Persistent Homology.

1. Introduction

R(http://cran.r-project.org/) is a programming language for statistical computing and graphics.

R has several good properties: R has many packages for statistical computing. Also, R is easy to make (interactive) plots. R is a script language, and it is easy to use. But, R is slow. C or C++ stands on the opposite end: C or C++ also has many packages(or libraries). But, C or C++ is difficult to make plots. C or C++ is a compiler language, and is difficult to use. But, C or C++ is fast. In short, R has short development time but long execution time, and C or C++ has long development time but short execution time.

Several libraries are developed for Topological Data Analysis: for example, GUDHI(https://project.inria.fr/gudhi/software/), Dionysus(https://www.mrzv.org/software/dionysus/), and PHAT(https://code.google.com/p/phat/). They are all written in C++, since Topological Data Analysis is computationally heavy and R is not fast enough.

R package TDA(http://cran.r-project.org/web/packages/TDA/index.html) bridges between C++ libraries(GUDHI, Dionysus, PHAT) and R. TDA package provides an R interface for the efficient algorithms of the C++ libraries GUDHI, Dionysus and PHAT. So by using TDA package, short development time and short execution time can be both achieved.

R package TDA provides tools for Topological Data Analysis. You can compute several different things with TDA package: you can compute common distance functions and density estimators, the persistent homology of the Rips filtration, the persistent homology of sublevel sets of a function over a grid, the confidence band for the persistence diagram, and the cluster density trees for density clustering.

2. Setting up

Obviously, you should download R first. R of version at least 3.1.0 is recommended:
http://cran.r-project.org/bin/windows/base/ (for Windows)
http://cran.r-project.org/bin/macosx/ (for (Mac) OS X)

R is part of many Linux distributions, so you should check with your Linux package management system.

You can use whatever IDE that you would like to use (Rstudio, Eclipse, Emacs, Vim...). R itself also provides basic GUI or CUI. I personally use Rstudio:

http://www.rstudio.com/products/rstudio/download/

Before installing R package TDA, Four packages are needed to be installed: parallel, FNN, igraph, and scales. parallel is included when you install R, so you need to install FNN, igraph, and scales by yourself. You can install them by following code (or pushing 'Install R packages' button if you use Rstudio).

```r
#########################################################################
# installing required packages
#########################################################################
if (!require(package = "FNN")) {
  install.packages(pkgs = "FNN")
}
if (!require(package = "igraph")) {
  install.packages(pkgs = "igraph")
}
if (!require(package = "scales")) {
  install.packages(pkgs = "scales")
}
if (!require(package = "TDA")) {
  install.packages(pkgs = "TDA")
}
```  

After that, you can install R package TDA as in the following code (or pushing 'Install R packages' button if you use Rstudio).

```r
#########################################################################
# installing R package TDA
#########################################################################
if (!require(package = "TDA")) {
  install.packages(pkgs = "TDA")
}
```  

Once installation is done, R package TDA should be loaded as in the following code, before using the package functions.

```r
#########################################################################
# loading R package TDA
#########################################################################
library(package = "TDA")
```
3. Sample on manifolds, Distance Functions, and Density Estimators

3.1. Uniform Sample on manifolds

A set of $n$ points $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ has been sampled from some distribution $P$.

- $n$ sample from the uniform distribution on the circle in $\mathbb{R}^2$ with radius $r$.

```r
# uniform sample on the circle
circleSample <- circleUnif(n = 400, r = 1)
plot(circleSample)
```

- $n$ sample from the uniform distribution on the sphere $S^d$ in $\mathbb{R}^{d+1}$ with radius $r$.

```r
# uniform sample on the sphere
sphereSample <- sphereUnif(n = 10000, d = 2, r = 1)
if (!require(package = "rgl")) {
  install.packages(pkgs = "rgl")
}
library(rgl)
plot3d(sphereSample)
```

- $n$ sample from the uniform distribution on the torus in $\mathbb{R}^3$ with small radius $a$ and large radius $b$.

```r
# uniform sample on the torus
```
### Tutorial on the R package TDA

```r
torusSample <- torusUnif(n = 10000, a = 1.8, c = 5)
if (!require(package = "rgl")) {
  install.packages(pkgs = "rgl")
}
library(rgl)
plot3d(torusSample)
```

#### 3.2. Distance Functions, and Density Estimators

We compute distance functions and density estimators over a grid of points. Suppose a set of points $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ has been sampled from some distribution $P$. 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.

```r
# uniform sample on the circle, and grid of points
X <- circleUnif(n = 400, r = 1)
Xlim <- c(-1.6, 1.6)
Ylim <- c(-1.7, 1.7)
by <- 0.065
Xseq <- seq(from = Xlim[1], to = Xlim[2], by = by)
Yseq <- seq(from = Ylim[1], to = Ylim[2], by = by)
Grid <- expand.grid(Xseq, Yseq)

• The distance function is defined for each $y \in \mathbb{R}^d$ as $\Delta(y) = \inf_{x \in X} \|x - y\|_2$.
```

```r
# distance function
distance <- distFct(X = X, Grid = Grid)
par(mfrow = c(1,2))
plot(X, xlab = "", ylab = "", main = "Sample X")
persp(x = Xseq, y = Yseq,
     z = matrix(distance, nrow = length(Xseq), ncol = length(Yseq)),
     xlab = "", ylab = "", zlab = "", theta = -20, phi = 35, scale = FALSE,
     expand = 3, col = "red", border = NA, ltheta = 50, shade = 0.5,
     main = "Distance Function")
```
• 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. The DTM can be seen as a smoothed version of the distance function. For more details see 7.

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

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

where $v_d$ is the volume of the Euclidean $d$ dimensional unit ball and $r_k^d(x)$ is the Euclidean distance form point $x$ to its $k$th closest neighbor among the points of $X$.
• The Gaussian Kernel Density Estimator (KDE), for each \( y \in \mathbb{R}^d \), is defined as

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

where \( h \) is a smoothing parameter.
The Kernel distance estimator, for each \( y \in \mathbb{R}^d \), is defined as

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

where \( K_h(x, y) = \exp\left(-\frac{\|x-y\|^2}{2h^2}\right) \) is the Gaussian Kernel with smoothing parameter \( h \).

# kernel distance
# kernelDist(X = X, Grid = Grid, h = h)

```r
h <- 0.3
Kdist <- kernelDist(X = X, Grid = Grid, h = h)
par(mfrow = c(1,2))
plot(X, xlab = "", ylab = "", main = "Sample X")
persp(x = Xseq, y = Yseq,
      z = matrix(Kdist, nrow = length(Xseq), ncol = length(Yseq)),
      xlab = "", ylab = "", zlab = "", theta = -20, phi = 35, scale = FALSE,
      expand = 3, col = "red", border = NA, ltheta = 50, shade = 0.5,
      main = "Kernel Distance")```
4. Persistent Homology

4.1. Persistent Homology Over a Grid

`gridDiag` function computes the persistent homology of sublevel (and superlevel) sets of the functions. The function `gridDiag` evaluates a given real valued function over a triangulated grid (in arbitrary dimension), constructs a filtration of simplices using the values of the function, and computes the persistent homology of the filtration. The user can choose to compute persistence diagrams using either the `Dionysus` library (`library = "Dionysus"`) or the `PHAT` library (`library = "PHAT"`).

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 other inputs are the features of the grid over which the `kde` is evaluated (`lim` and `by`), and a logical variable that indicates whether a progress bar should be printed (`printProgress`).

```r
Diag <- gridDiag(X = X, FUN = kde, lim = cbind(Xlim, Ylim), by = by,
                 sublevel = FALSE, library = "Dionysus", printProgress = FALSE, h = 0.3)
```

The function `plot` plots persistence diagram for objects of the class "diagram". 8th line of the following command produces the third of the following plot.

```r
par(mfrow = c(1,3))
plot(X, main = "Sample X")
persp(x = Xseq, y = Yseq,
```
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).

```r
# other options for plotting persistence diagram
par(mfrow = c(1,3))
plot(Diag[["diagram"]], main = "KDE Diagram")
plot(Diag[["diagram"]], rotated = TRUE, main = "Rotated Diagram")
plot(Diag[["diagram"]], barcode = TRUE, main = "Barcode")
```

### 4.2. Rips Diagrams

The Vietoris-Rips complex $R(X, \varepsilon)$ consists of simplices with vertices in $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ and diameter at most $\varepsilon$. The `ripsDiag` function computes the persistence diagram of the Rips filtration built on top of a point cloud. The user can choose to compute the Rips persistence diagram using either the C++ library GUDHI (library = "GUDHI"), or Dionysus (library = "Dionysus")

The following code generates 60 points from two circles, as in the following figure:
We specify the limit of the Rips filtration (maxscale = 5) and the max dimension (maxdimension = 1) of the homological features we are interested in (0 for components, 1 for loops, 2 for voids, etc.). Then we plot the data and the diagram.
4.3. Bottleneck and Wasserstein Distances

\[
\text{Diag1 } \leftarrow \text{ripsDiag}(X = \text{Circle1}, \text{maxdimension} = 1, \text{maxscale} = 5) \\
\text{Diag2 } \leftarrow \text{ripsDiag}(X = \text{Circle2}, \text{maxdimension} = 1, \text{maxscale} = 5) \\
\]

\[
\text{print(bottleneck(Diag1 = Diag1["diagram"], Diag2 = Diag2["diagram"], dimension = 1))} \\
\text{## [1] 1.175885} \\
\text{print(wasserstein(Diag1 = Diag1["diagram"], Diag2 = Diag2["diagram"], p = 2, dimension = 1))} \\
\text{## [1] 1.764362} \\
\]

4.4. Landscapes and Silhouettes
tseq <- seq(from = 0, to = 5, length = 1000)  # domain
Land <- landscape(Diag = Diag["diagram"], dimension = 1, KK = 1, tseq = tseq)
plot(tseq, Land, type = "l", main = "1st Landscape, dim=1", ylab = "", asp = 1,
col = "red", lwd = 3)

1st Landscape, dim=1

Affiliation:
Firstname Lastname
Affiliation
Address, Country
E-mail: name@address
URL: http://link/to/webpage/