

# Topological Data Analysis

*Theory, Practice, Software, and Potential*



**Justin Skycak**

10.31.2016

AUNALYTICS EXPOSITORY PAPER

## CONTENTS

---

<b>[Introduction]</b>	TDA and ML ..... 2
<b>[Theory]</b>	A Mathematician’s View of TDA. .... 5
<b>[Practice]</b>	A Computer Scientist’s View of TDA ..... 16
<b>[Software]</b>	TDA Software ..... 26
<b>[Potential]</b>	What TDA Can Do For Us ..... 35

## TDA AND ML

---

### Summary

Machine Learning (ML) and Topological Data Analysis (TDA) are different approaches to data analysis, each of which has its own strengths and weaknesses relative to the other. Best results will be achieved by taking advantage of their complementarity.

### What are TDA and ML?

Machine Learning (ML) traditionally involves learning parameters to best fit a model to the data. Topological Data Analysis (TDA) involves summarizing important features of a dataset's "shape" arising from similarity between data points.

### Advantages of traditional ML over TDA

**Traditional ML has extensive software support.** ML arose from a community of programmers, while TDA arose from a community of mathematicians. Though the TDA software base is growing, current options are limited.

**Traditional ML has "proven" itself.** Traditional ML is solving the problems of computer vision, dialogue, and translation, and major software companies often have their own ML labs. TDA has yet to solve a "big" problem, and the only major player in commercial TDA is Ayasdi.

**Traditional ML is conceptually concrete.** TDA is conceptually abstract and can be hard to grasp without a background in higher mathematics. However, traditional ML requires a background in more concrete topics like optimization and probability.

### Advantages of TDA over Traditional ML

**TDA is general.** TDA requires only knowledge of distance between data points - therefore, it is coordinate-free and not limited by data types. (Distance can, but need not, be computed from coordinates.)

***TDA is robust.*** TDA uses distances to group points into local neighborhoods called open sets, and from there onward the analysis depends only on the collection of open sets. Any distance metric that is mathematically valid (positive-definite, symmetric, and subadditive) will yield the same open sets, and thus give rise to the same topological features. (However, different distance metrics will emphasize different features in visualizations.)

***TDA yields qualitative insights and suggests hypotheses.*** The key to good data analysis is asking the right questions. TDA reframes data in a way that makes it easy for humans to visualize and make inferences about it.

***TDA requires no parameter tuning.*** In TDA, analysis results don't depend on ad-hoc parameters, like the number of clusters in k-means clustering. Persistence barcodes allow us to view results at all levels of scale, on a single graph.

***TDA respects continuous data.*** While clustering methods from traditional ML tend to create spurious divisions in continuous data, TDA visualizations are able to capture continuous transitions between clusters.

***TDA does not incur projection loss.*** For high-dimensional data, dimensionality reduction is sometimes needed to make clustering feasible. When clustering is performed after dimensionality reduction, as is typical in traditional ML, clusters that should be distinct may overlap in lower-dimensional space. However, TDA keeps distinct clusters separate by performing dimensionality reduction and clustering simultaneously. By clustering small portions of the data into nodes and then connecting nodes which share data points, TDA can generate a higher-level network representation of the data which reduces complexity while maintaining dimensional separation.

## Embrace the “And”

Fortunately, ML and TDA are not mutually exclusive. One can use TDA to generate hypotheses which can be tested by ML and discover features which can fuel prediction algorithms. Likewise, one can use ML to create distance metrics for TDA visualizations. (For example, valid ML-inspired distance metrics include distance between leaf nodes from a decision tree, distance between leaf node sequences from a random forest, and distance between class probability vectors from a neural net classifier.)

## References

<https://www.ayasdi.com/blog/bigdata/how-tda-and-machine-learning-enhance-each-other/>

<https://www.ayasdi.com/resources/whitepaper/tda-and-machine-learning/>

## A MATHEMATICIAN'S VIEW OF TDA

---

### Summary

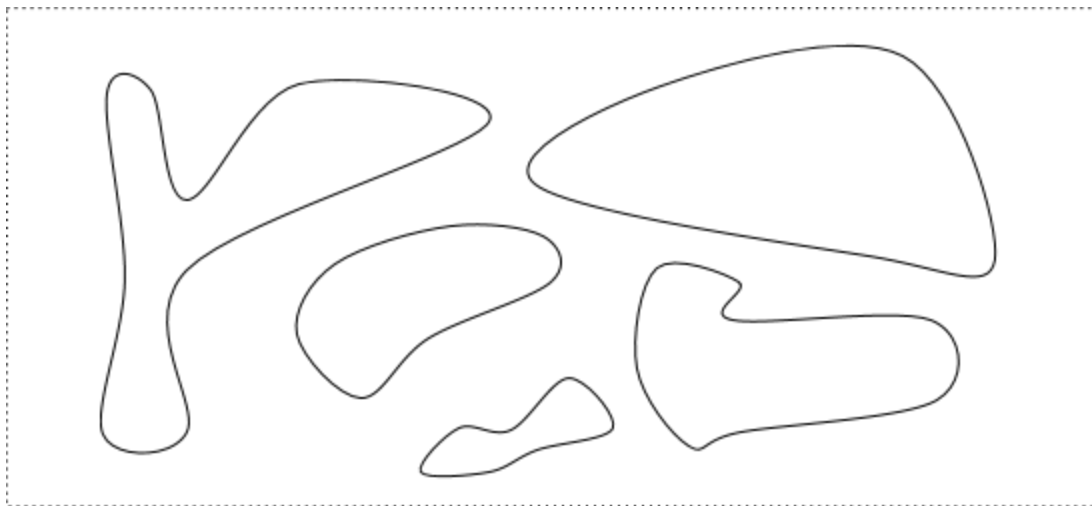
Algebraic topology aims to describe the connectivity of any arbitrary space. It does this by computing the homology, or number of “holes” in each dimension. In computational topology, datasets can be interpreted as samples taken from an underlying topological space, and for any given margin of error a topology can be constructed to approximate the underlying space. Persistence barcode plots show which topological features persist through many scales of the data, and can be used to calculate similarity between different spaces.

### Shapes, Loops, Holes, and Surfaces

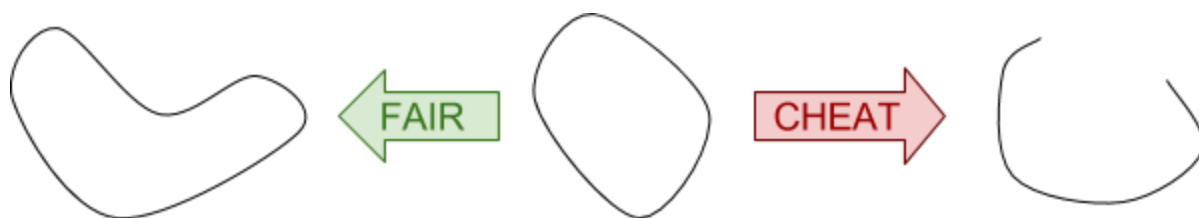
The ultimate goal of algebraic topology is to mathematically formalize information about a shape’s connectivity. Intuitively, “O” and “P” have the same connectivity, while “O” and “B” do not. This is because “O” and “P” can be deformed smoothly into one another, but deforming “B” into “O” would require tearing a loop.

Our intuition about connectivity seems to be based on loops: “O” and “P” have the same connectivity because they both have one loop, while “B” has different connectivity because it has two loops instead of one. Algebraic topology takes this idea of classifying shapes based on how many loops they have, and extends it to spaces of arbitrarily many dimensions.

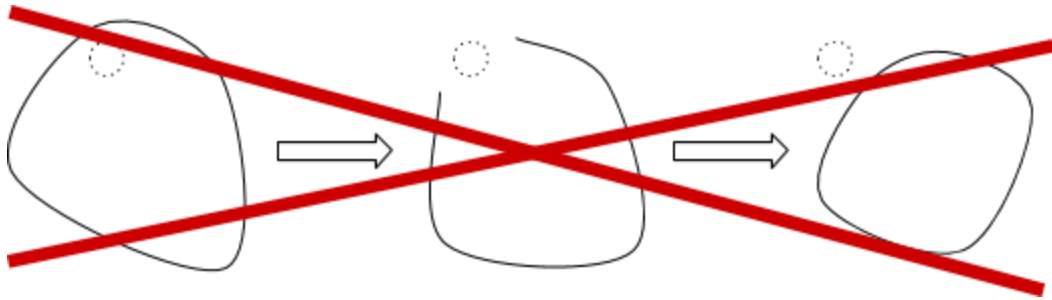
Let’s start off with a two-dimensional space, a plane. Immediately, we run into a problem we didn’t have with letters: we can draw infinitely many loops in this infinite sheet. How should we count them all?



So that we're not stuck counting loops for eternity, we'll say that loops are equivalent if they can be continuously deformed into each other. Given a loop, we can drag it across the plane, twist it around, stretch it, compress it, and so on, and it will still be the same loop. The only thing we aren't allowed to do is tear it. That's cheating.

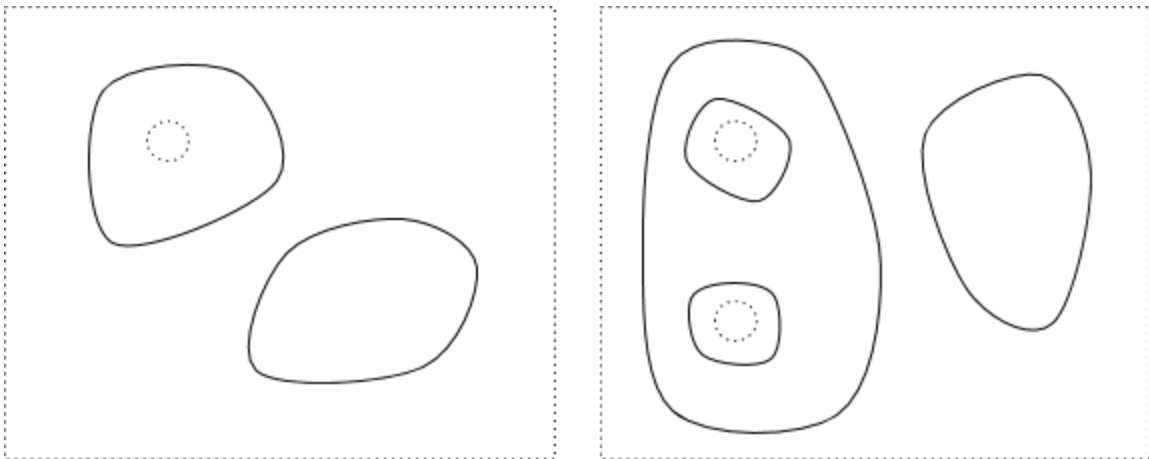


To count loops using this equivalence, we need to figure out how many *distinct* loops there are. What can prevent us from deforming one loop into another? A hole in the sheet will do it. If the hole is inside a loop, the only way to move it to the outside is to tear the loop, and that's cheating. Therefore, the loop with the hole inside is not the same as the loop with the hole outside.



A plane with one hole, then, has two loops: one loop with the hole inside and another loop with the hole outside. The loop with the hole inside can be deformed into any other loop with the hole inside but not outside, and the loop with the hole outside can be deformed into any other loop with the hole outside but not inside. For a plane with two holes, we have four distinct loops: one around each hole, one around both holes, and one around neither hole.

(Technically, there is also a distinct loop which wraps twice around a hole, a distinct loop which wraps three times around a hole, and so on but we'll amend our definition of "loop" so that loops aren't allowed to cross themselves.)



Topologists have a word, *homotopy*, for the kind of non-tearing deformations we're imagining. They call the collection of distinct loops a *homotopy group*. Here are a few translations into topologist lingo:

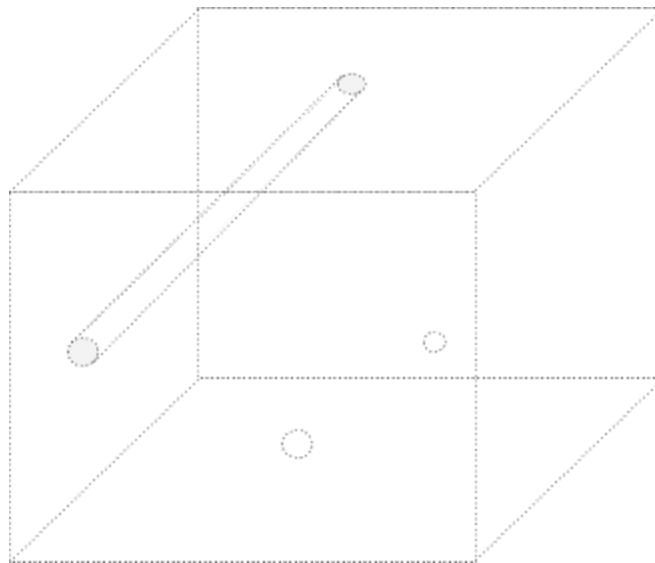
- "A sheet with no holes has one loop" → "An unpunctured plane has the homotopy



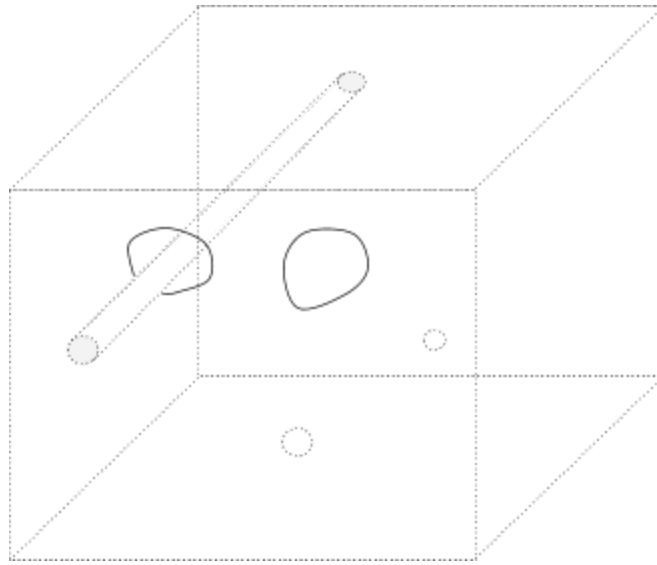
group consisting of loops around {no holes}"

- “A sheet with one hole has two loops” → “A once-punctured plane has the homotopy group consisting of loops around {no holes, hole 1}"
- “A sheet with two holes has four loops” → “A twice-punctured plane has the homotopy group consisting of loops around {no holes, hole 1 only, hole 2 only, both holes 1 and 2}"

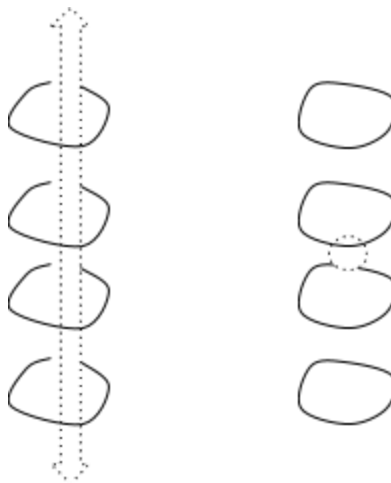
In higher-dimensional spaces, we need to count loops *in each dimension*. For example, consider a three-dimensional space with two points and a line removed. This looks like a cube of cheese that has two air bubbles and has been poked by a toothpick, infinitely enlarged.



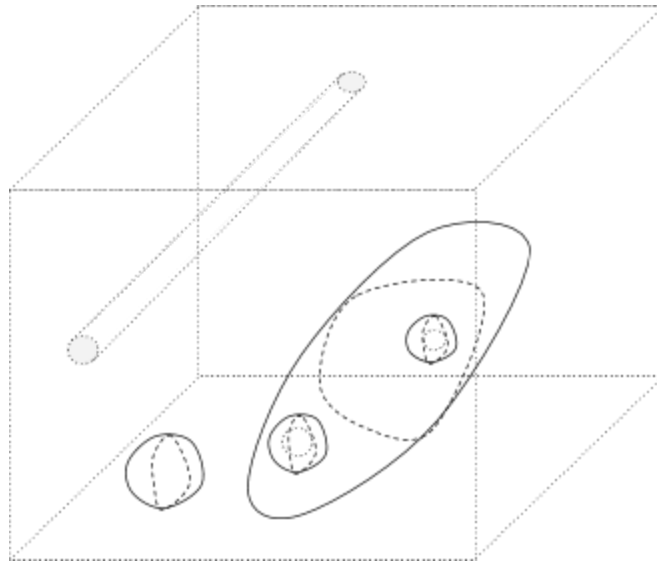
There are two one-dimensional loops: a circle with the line inside, and a circle with the line outside.



A circle containing the line is tethered to the line, but a circle containing the missing point can move above and below the point. Since circles can move through the missing points, the missing points do not affect the number of distinct one-dimensional loops.



There are four two-dimensional loops: a bubble with no points inside, a bubble with one point inside, a bubble with the other point inside, and a bubble with both points inside. The missing line does not affect the number of distinct two-dimensional loops because the line spans the entire space and consequently we cannot put the line inside of a bubble.



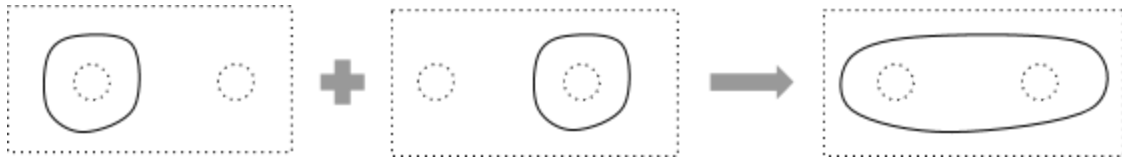
In topologist lingo, the homotopy groups of this shape are the loops with {line inside, line outside} in dimension 1 and the loops with {no points inside, one point inside, the other point inside, both points inside} in dimension 2.

In higher dimensions, it's helpful to think of loops as surfaces. A 1-dimensional loop (circle) is the surface of a 1-dimensional sphere (disk). A 2-dimensional loop (bubble) is the surface of a 2-dimensional sphere (solid sphere). In general, an  $n$ -dimensional loop is the surface of an  $n$ -dimensional sphere.

## Computational Issues and Simplicial Complexes

In higher dimensions, homotopy groups are hard to compute. One way forward is to reduce the number of loops we have to count for each space - specifically, we count only loops which can't be made from combinations of other loops. The groups consisting of these loops are called *homology groups*, and it's sometimes possible to apply clever tricks (beyond the scope of this paper) to compute a homology group even when we are unable to compute the homotopy group.

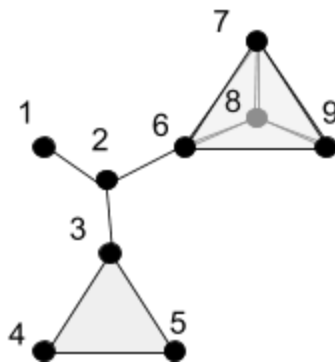
As a concrete example, consider again the three-dimensional space with two points and a line removed (i.e. the cheese cube on a toothpick with two air pockets). The bubble with both points inside is the combination of the bubble with one point inside and the bubble with the other point inside, so it does not belong in the homology group.



To compute the homology group for an  $N$ -dimensional space, one need only put an  $n$ -dimensional loop around each  $(N-n)$ -dimensional hole. Therefore, the  $n$ th Betti number, which topologists define as the number of independent surfaces in the  $n$ th dimension, is just one more than the number of  $(N-n)$ -dimensional holes.

However, though we can use clever tricks to compute more homology groups than homotopy groups, there is still no general method that we can use to compute the homology group for every space.

To solve this computational issue, we work with nice spaces where there is a general method for computation. One such family of spaces is the simplicial complex, whose homology can be calculated via linear algebra. A simplicial complex is a union of points (0-dimensional triangles), segments (1-dimensional triangles), 2-dimensional triangles, solid tetrahedrons (3-dimensional triangles), and higher-dimensional triangles.



Formally, a simplicial complex is represented by a pair  $(V, S)$ , a set of points  $V$  together with a set  $S$  containing subsets of  $V$ , such that any subset of a set in  $S$  is itself a set in  $S$ . An  $(n+1)$ -element subset in  $S$  is called an  $n$ -simplex and intuitively represents an  $n$ -dimensional triangle.

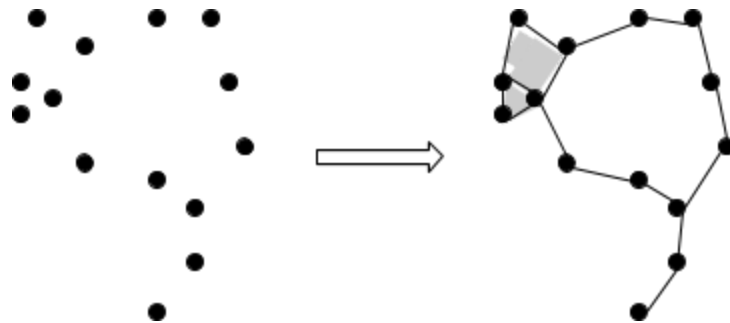
In the example complex above,  $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$  and  $S$  contains the following simplices:

- 0-simplices: 1, 2, 3, 4, 5, 6, 7, 8, 9
- 1-simplices: {1,2}, {2,3}, {2,6}, {3,4}, {3,5}, {4,5}, {6,7}, {6,8}, {6,9}, {7,8}, {7,9}, {8,9}
- 2-simplices: {3,4,5}, {6,7,8}, {6,7,9}, {6,8,9}, {7,8,9}
- 3-simplices: {6,7,8,9}

All together, then,  $S = \{1, 2, 3, 4, 5, 6, 7, 8, 9, \{1,2\}, \{2,3\}, \{2,6\}, \{3,4\}, \{3,5\}, \{4,5\}, \{6,7\}, \{6,8\}, \{6,9\}, \{7,8\}, \{7,9\}, \{8,9\}, \{3,4,5\}, \{6,7,8\}, \{6,7,9\}, \{6,8,9\}, \{7,8,9\}, \{6,7,8,9\}\}$ .

## Topological Approximation

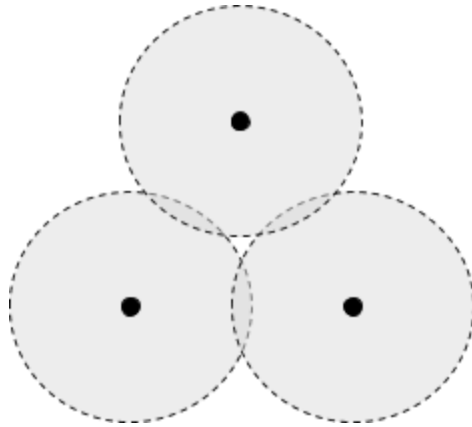
Real-world datasets are not explicit topological spaces. Rather, they are collections of points *sampled* from topological spaces. The goal of TDA is to analyze these point clouds to infer information about their underlying topological spaces. One can do this by using the point cloud to construct a simplicial complex which approximates the underlying topological space.



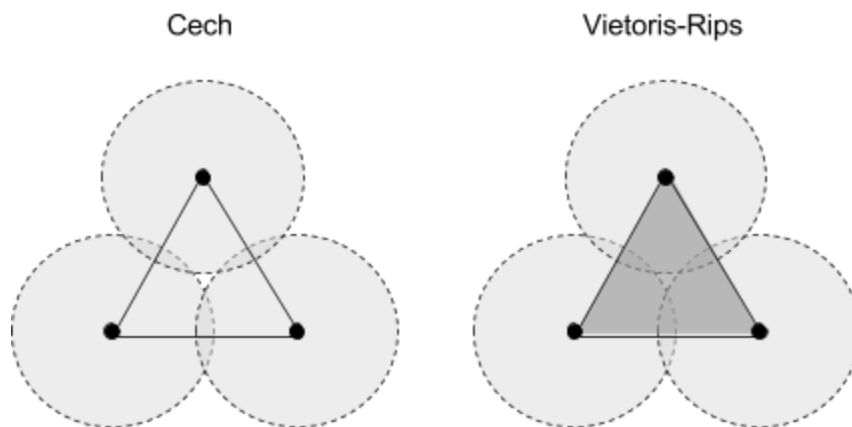
The main idea behind turning point clouds into simplicial complexes is to put epsilon-balls (error margins) around points and use the overlaps to determine the connections in the simplicial complex. The constructions generated using different values of epsilon will correspond to topological approximations of the point cloud at different levels of scale.

More precisely, one generates the Cech complex of a point cloud by adding an  $n$ -simplex whenever the intersection of all  $n$  balls is nonempty. A more computationally efficient method, the Vietoris-Rips complex, adds an  $n$ -simplex whenever the intersection of every *pair* in the  $n$  balls is nonempty, and contains the Cech complex as a subcomplex.

For example, consider these three points with pairwise-overlapping balls:



The associated Cech complex includes three points and three segments, but no triangle because the three balls don't all overlap together anywhere. However, The Vietoris-Rips complex contains the triangle in addition to the three points and segments, since each *pair* of balls overlaps.



There is a theorem stating that for any epsilon, there is a finite set of points such that the Cech complex (which is contained in the Vietoris-Rips complex) is homotopy equivalent to the full space. Therefore, in theory, our approximation should have the same topological features as the actual space provided we use enough points.

## Persistence Barcodes

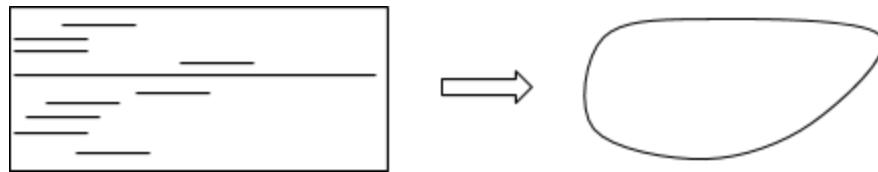
Our last question is, *which value of epsilon should we choose?*

This is a trick question.

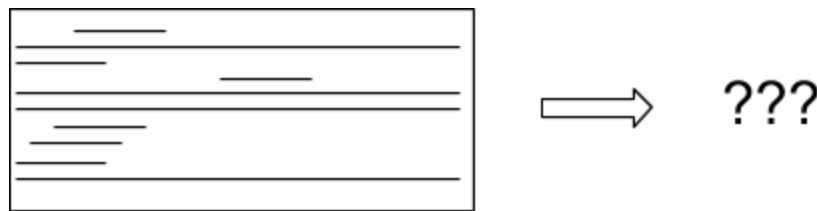
Since we're interested in topological features which persist across many scales of the data, we should use all values of epsilon. This is the idea behind persistent homology: we can figure out which features persist over the full range of scale by making a barcode plot that says whether a homology generator was detected at some value of epsilon.

On the horizontal axis we have values for epsilon, and on the vertical axis we have homology generators, loops which can potentially be in the homology group.

For example, this barcode plot with a single long bar tells us that the dataset has one loop which persists across many scales. Therefore, the dataset must be a circle.



However, in this barcode plot with many long bars, many loops means many possibilities for the space. Further analysis is needed to figure out which possibility is the right one.



Often, though, the question we seek to answer is not what the topological features of a dataset are, but whether two datasets have similar topology. We can quantify the degree to which two datasets share the same topology by comparing their persistence barcodes. Since barcodes are really just matrices, it is straightforward to compute a similarity index between them.

Persistence barcode distributions are currently an active area of research, and they can encode persistence of homology, but also be repurposed for persistence of centrality,

density, and other measures.

## References

<http://www.ams.org/images/carlsson-notes.pdf>

[http://www.dyinglovegrape.com/math/topology\\_data\\_1.php](http://www.dyinglovegrape.com/math/topology_data_1.php)

[http://www.dyinglovegrape.com/math/topology\\_data\\_2.php](http://www.dyinglovegrape.com/math/topology_data_2.php)

<https://www.math.utk.edu/~fernando/Students/GregClark/presentation.html#/>



## A COMPUTER SCIENTIST'S VIEW OF TDA

---

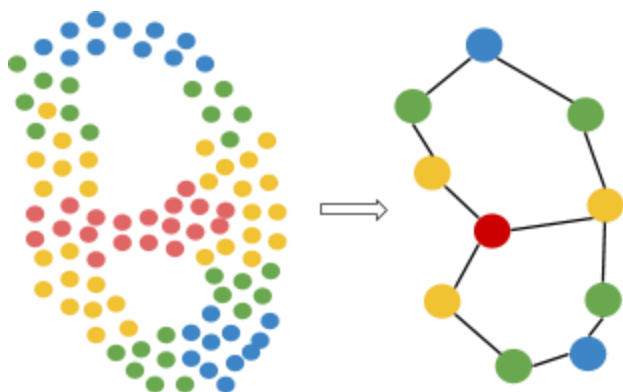
### Summary

While theoretical aspects of TDA make heavy use of persistence homology, TDA in practice tends toward the Mapper algorithm. This algorithm turns high-dimensional data into smaller networks which are easier to visualize and still retain the main topological features of the data. It has been used by Ayasdi to forecast returns, detect fraud, aid in oil and gas exploration, plan ad campaigns, and discover biomarkers.

### The Mapper Algorithm

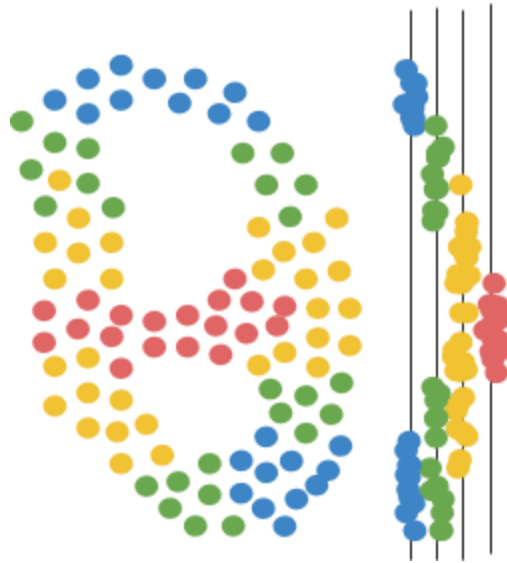
From a software standpoint, persistence homology is still in the works. There is an open-source R package, TDA, which has this functionality, but most other softwares are focused on the Mapper algorithm, which turns high-dimensional big data into smaller networks. Even Ayasdi, the main player in commercial TDA, has the Mapper algorithm at the core of its software.

The Mapper algorithm represents a space's topology by converting it into a network. For example:

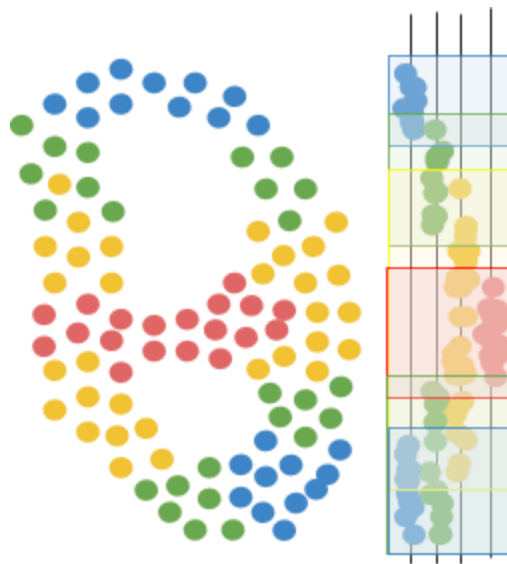


Here are the steps to yield the visualization above:

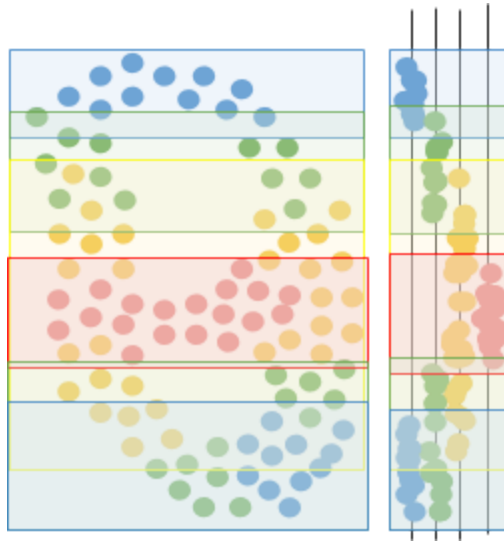
1. Apply some map (aka function, lens, filter) to the data. In this example, we use horizontal projection.



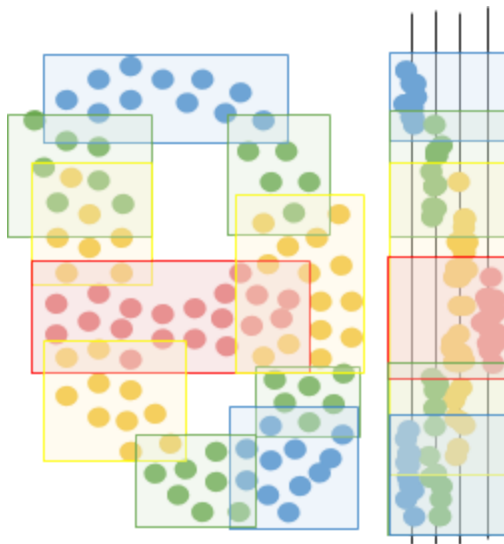
2. Use hierarchical clustering to create a cover (collection of overlapping clusters) for the map image. Hierarchical clustering is used because we are not really working with points, but distances between points.



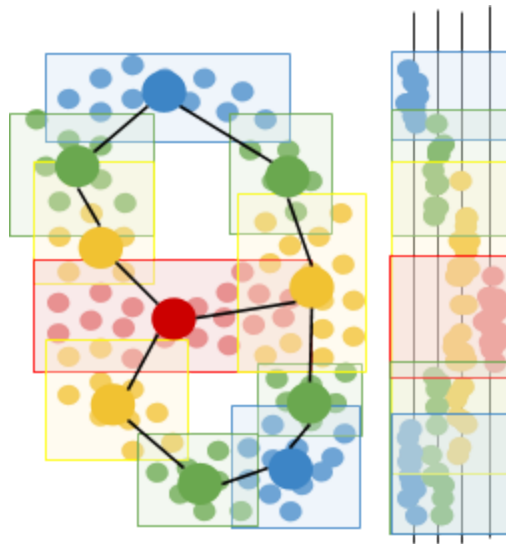
3. Take the inverse image of the map cover to get a cover for the data.



4. Run clustering algorithm on each cluster in the data cover to see if it can be separated (and if so, separate it).

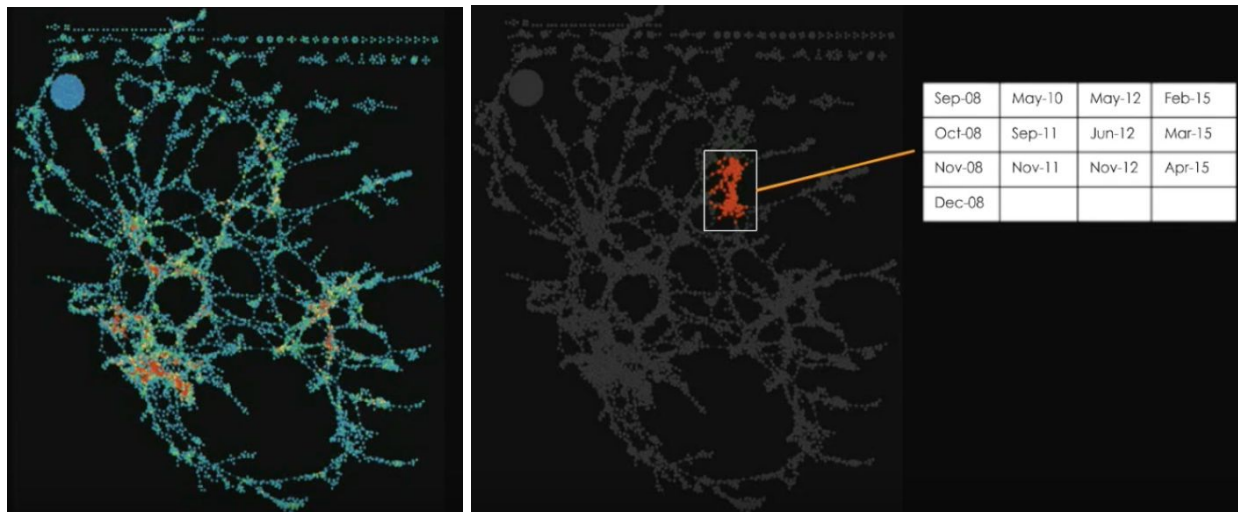


5. Represent data clusters as nodes, and connect nodes whose clusters overlap (i.e. share data points).



### Use Case: Forecasting Returns

Below is a network that Ayasdi software generated by applying the Mapper algorithm to over 300 market and economic variables, sampled over 25 years. The nodes are colored by year.



We see that the map is spread out over time, which indicates repeated patterns over time. For example, the group of highlighted nodes corresponds to high-volatility and high-stress conditions.

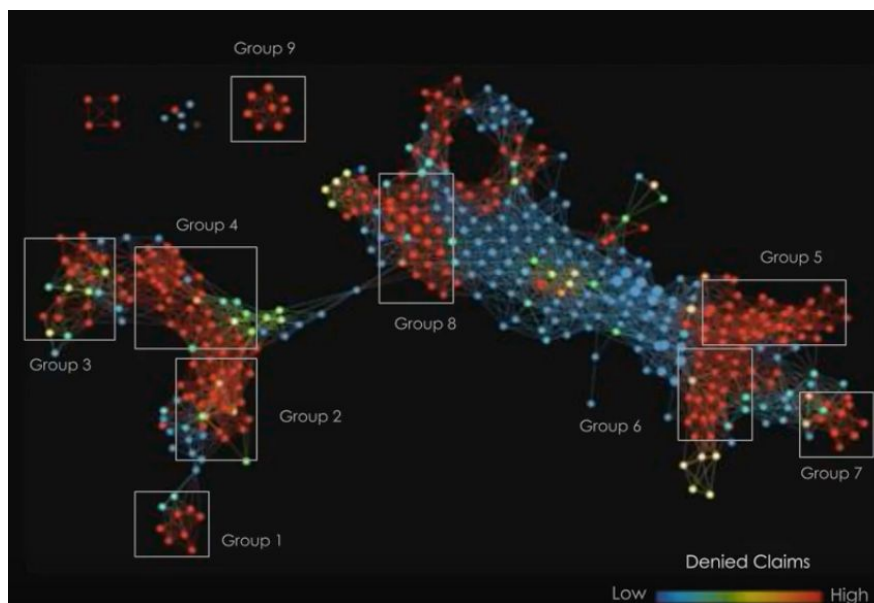
This suggests the following strategy to forecast from an initial date: locate neighboring

dates on the map, use their price trajectories to build a distribution of changes in price for each asset, and use mean or median for predictions. Then, the predicted asset price-changes can be used to come up with higher-level predictions, i.e. for each market sector.

### Use Case: Diagnosing Denied Claims

Pivot tables can only find denial patterns consisting of a few factors, and these simple denial patterns usually account for a minority of a denial backlog. Ayasdi's software has been used to find complex patterns in infusion/oncology medical necessity denials, accounting for up to 65% of denial backlog.

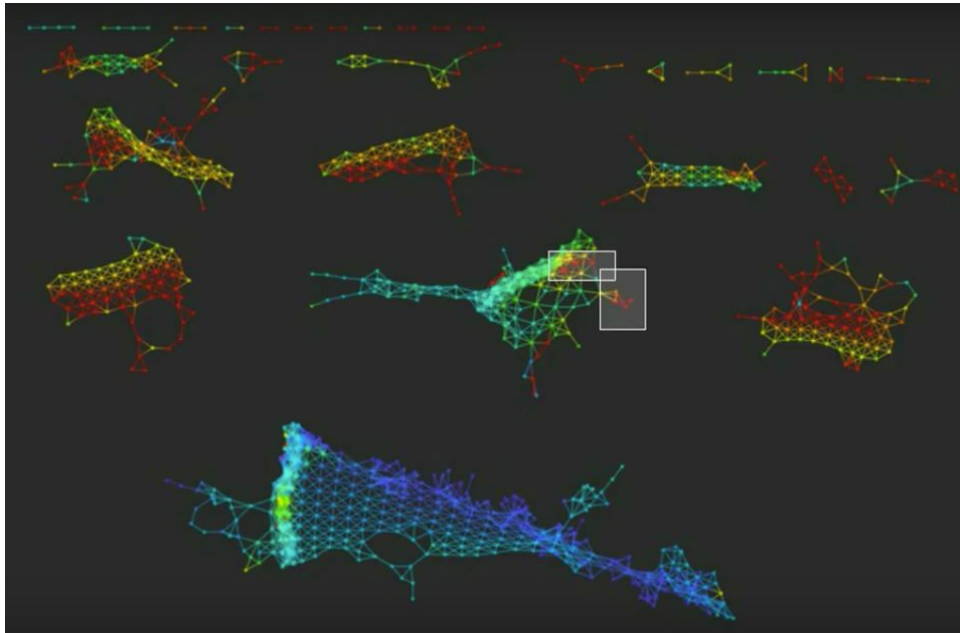
The following topological network was constructed by applying the Mapper algorithm to 5 million individual claims. The network's structure is determined by similarity between claims, and nodes are colored according to how often the claims were accepted or denied on average.



By locating several groups in the network and analyzing the group statistics, analysts were able to gain enough information to advise action pre-submission (i.e. modifying the final coding or supporting diagnosis) or at the point of care (i.e. seeking pre-authorization or reconsidering a procedure).

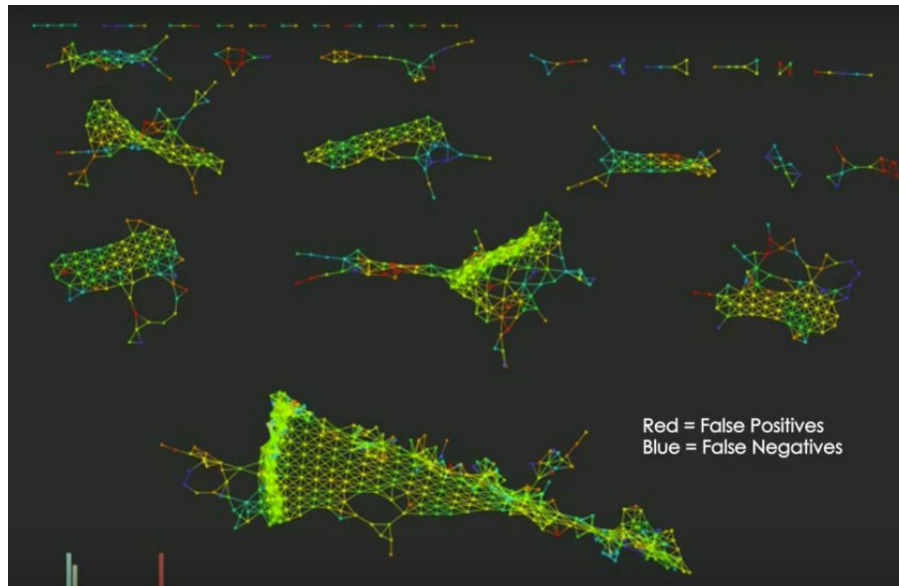
## Use Case: Detecting Fraud

The topological network below is based on the CMS public health claims dataset of over 9 million claims, 36 thousand providers, and 3600 unique codes. The network structure is determined by similarity in how providers practice, while the node color is determined by medicare payment amount.



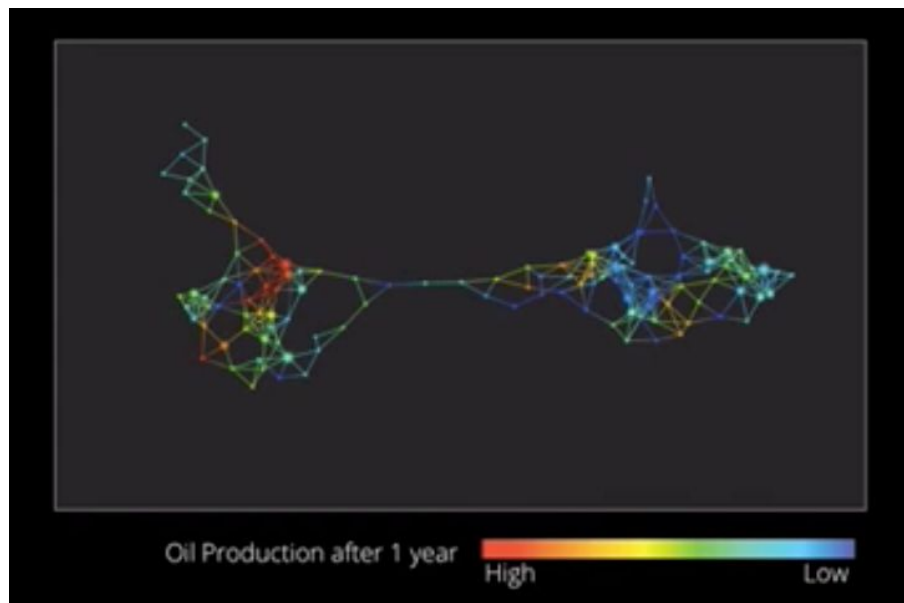
One can identify leads for investigation by looking for outlier providers who are getting paid abnormally much compared to other similar providers. Two such groups are boxed in the network above.

One can also improve detection models using this topological network. By recoloring the network nodes according to model performance (e.g. false positive rate), one can find groups for which the model performs poorly. By running statistical tests to discover how these groups differ most significantly from the rest of the population, one can identify parameters which the model may have learned incorrectly.



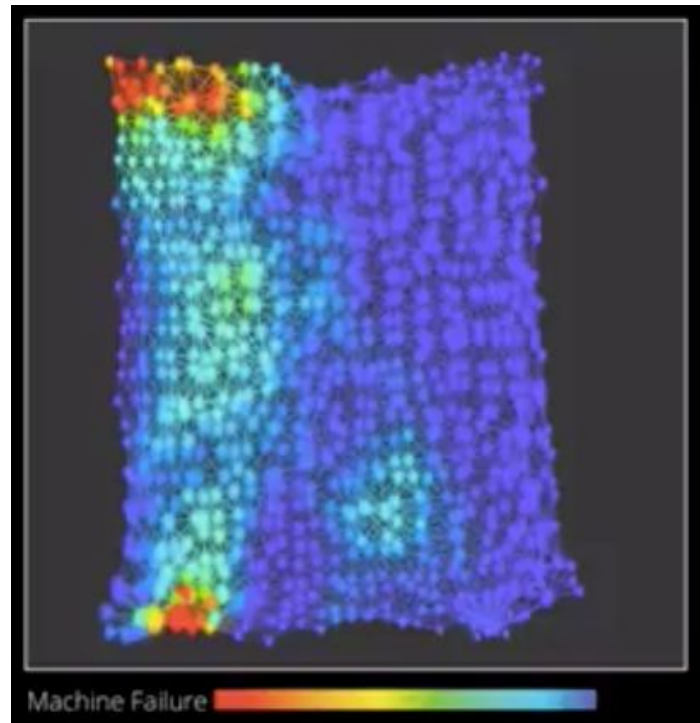
### Use Case: Oil and Gas Exploration

Below is an example of a topological network whose structure is based on a drilling location, and whose color is based on the amount of oil recovered there. This information can be useful in identifying new locations most likely to be oil-rich.



Topological networks can provide valuable information about the drilling equipment, as

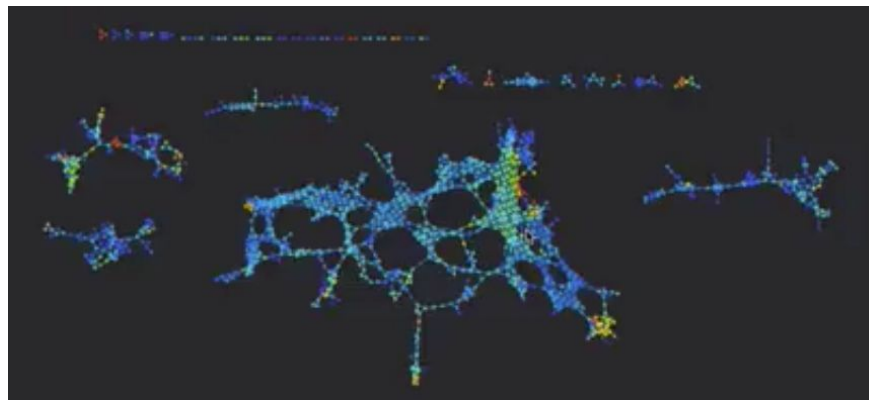
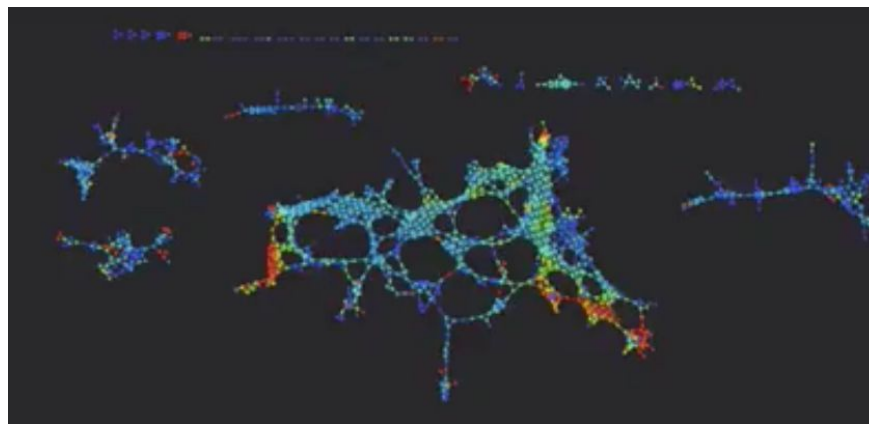
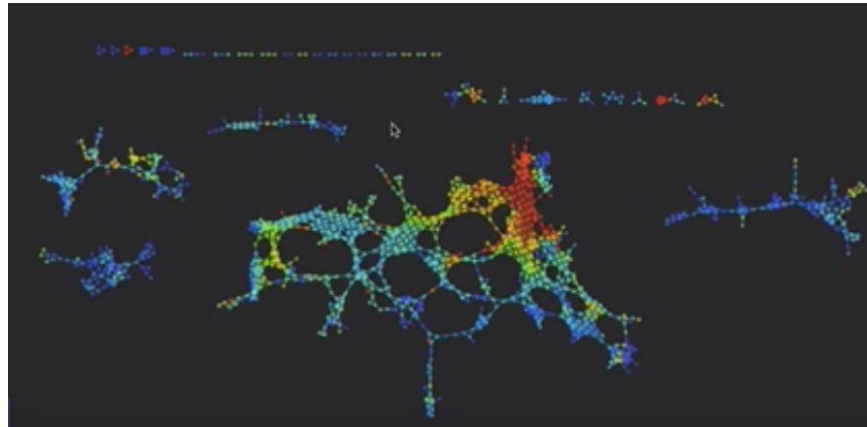
well. Below is a network whose structure is determined by of readout variables in system states, and whose color determined by frequency of failure (red = high, blue = low). By better understanding the correlation between system status and failure frequency, one can anticipate critical events and avoid unnecessary replacements.



### Use Case: Campaign Planning

Based on data on 37,000 Twitter users who tweeted about Chris Christie, a topological network structured by account similarity and colored by word frequency can be used to identify niche conversations that are good targets for ads. Shown below (top to bottom) are colorings corresponding to “scandal,” “traffic,” and “Governor.”

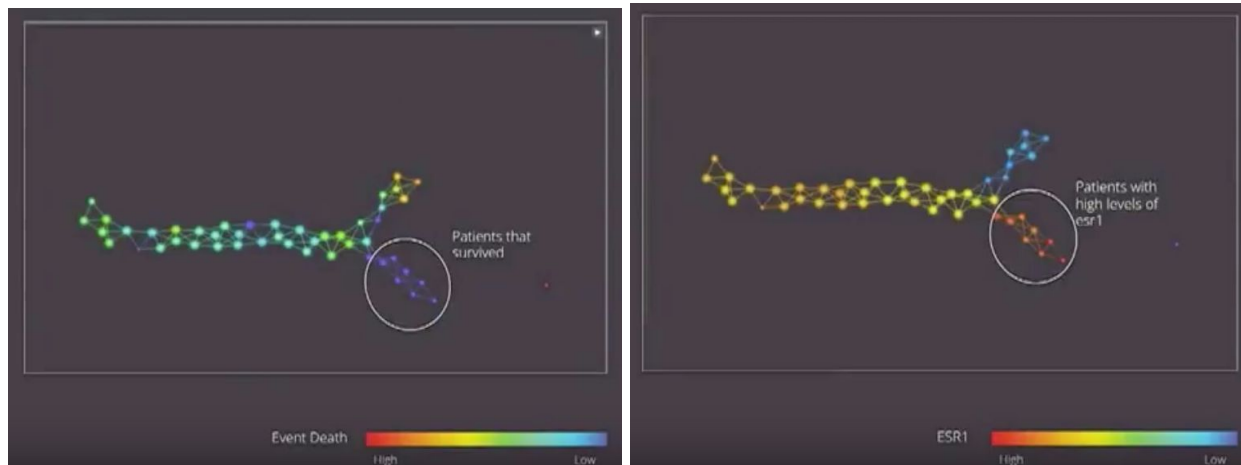




One can also investigate an individual group to see what other words differentiate the group from other groups. This gives more specific insight into the content of the discussion.

## Use Case: Biomarker Discovery

Below is a topological network generated by data from 272 breast cancer patients. The structure is based on similarity in genes expressed by patients. The left graph is colored by death (red = high, blue = low), while the right graph is colored by esr1 level (red = high, blue = low). We can see that the flare of patients who survived corresponds to the flare of patients with high levels of esr1.



## References

<https://research.math.osu.edu/tgda/mapperPBG.pdf>

<http://topology.cs.wisc.edu/Bak.pdf>

<https://www.ayasdi.com/company/events/forecasting-returns-using-machine-intelligence/>

<https://www.ayasdi.com/company/events/machine-intelligence-for-denied-claims/>

<https://www.ayasdi.com/company/events/recognizing-the-shape-of-fraud-using-machine-intelligence-for-fraud-waste-and-abuse/>

<https://www.ayasdi.com/company/events/webinar-oil-and-gas-exploration-with-ayasdi/>

<https://www.ayasdi.com/company/events/webinar-campaign-planning-with-social-media-intelligence/>

<https://www.ayasdi.com/company/events/webinar-biomarker-drug-target-discovery-with-ayasdi/>

## TDA SOFTWARE

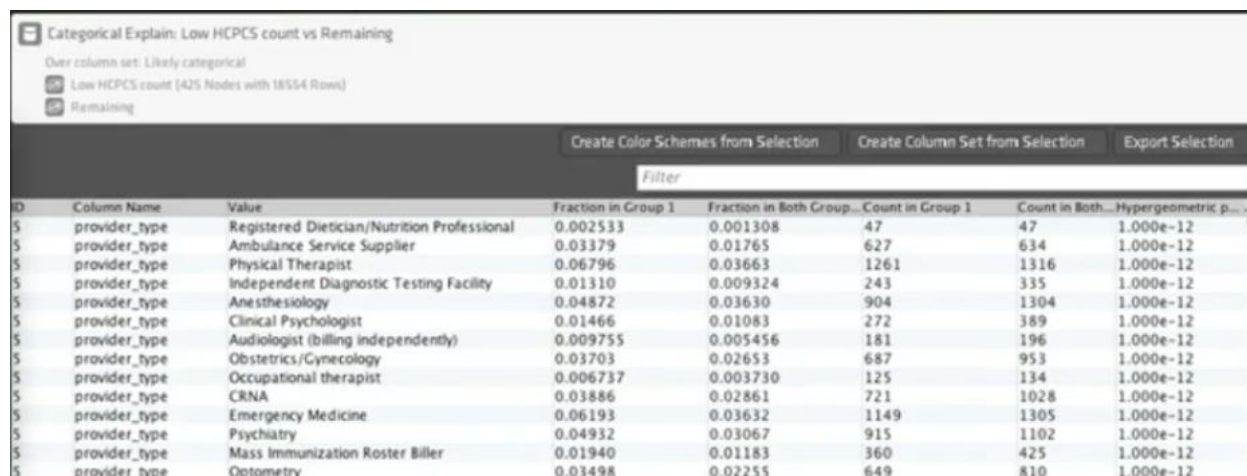
### Summary

The market for commercial TDA is dominated by Ayasdi, which focuses on the Mapper algorithm. There is an open-source R package TDAmapper which implements the Mapper algorithm, and another open-source R package TDA with persistent homology capabilities. There is also a Python module, KeplerMapper, which implements the Mapper algorithm, but it requires data points rather than distances as input.

### Ayasdi

As shown in the use-cases of the previous section, Ayasdi software uses the Mapper algorithm to create topological network visualizations of complex datasets. Many of the use-cases involve coloring the nodes of the network, visually identifying clusters, and figuring out what separates interesting clusters from the rest of the data. Ayasdi provides a graphical interface with off-the-shelf methods for these capabilities.

Ayasdi also has an “explain” function which automates this last step of differentiating clusters. It does this by running a barrage of statistical tests against a selected group and ranking the selected group’s most significant differences from the rest of the data.



Categorical Explain: Low HCPCS count vs Remaining

Over column set: Likely categorical

- Low HCPCS count (425 Nodes with 18554 Rows)
- Remaining

Create Color Schemes from Selection   Create Column Set from Selection   Export Selection

Filter

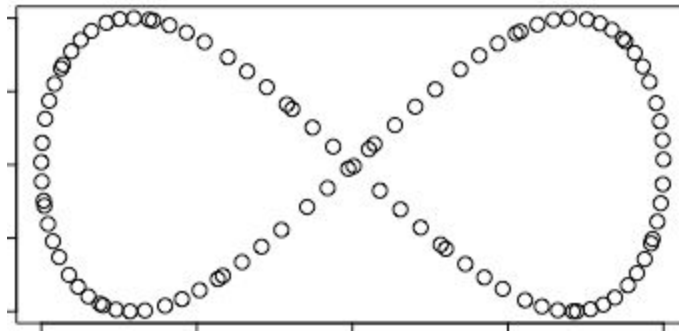
ID	Column Name	Value	Fraction in Group 1	Fraction in Both Group...	Count in Group 1	Count in Both...	Hypergeometric p...
5	provider_type	Registered Dietician/Nutrition Professional	0.002533	0.001308	47	47	1.000e-12
5	provider_type	Ambulance Service Supplier	0.03379	0.01765	627	634	1.000e-12
5	provider_type	Physical Therapist	0.06796	0.03663	1261	1316	1.000e-12
5	provider_type	Independent Diagnostic Testing Facility	0.01310	0.009324	243	335	1.000e-12
5	provider_type	Anesthesiology	0.04872	0.03630	904	1304	1.000e-12
5	provider_type	Clinical Psychologist	0.01466	0.01083	272	389	1.000e-12
5	provider_type	Audiologist (billing independently)	0.009755	0.005456	181	196	1.000e-12
5	provider_type	Obstetrics/Gynecology	0.03703	0.02653	687	953	1.000e-12
5	provider_type	Occupational therapist	0.006737	0.003730	125	134	1.000e-12
5	provider_type	CRNA	0.03886	0.02861	721	1028	1.000e-12
5	provider_type	Emergency Medicine	0.06193	0.03632	1149	1305	1.000e-12
5	provider_type	Psychiatry	0.04932	0.03067	915	1102	1.000e-12
5	provider_type	Mass Immunization Roster Biller	0.01940	0.01183	360	425	1.000e-12
5	provider_type	Optometry	0.03498	0.02255	649	810	1.000e-12

Due to the commercial nature of Ayasdi’s software, we cannot explore it in great detail.

## R package TDAmapper

TDAmapper is an R implementation of the Mapper algorithm. We'll demonstrate it on a figure-8:

```
x_coord <- 2*cos(0.5*(1:100))
y_coord <- sin(1:100)
figure_8 <- data.frame( x=x_coord, y=y_coord )
plot(figure_8)
```

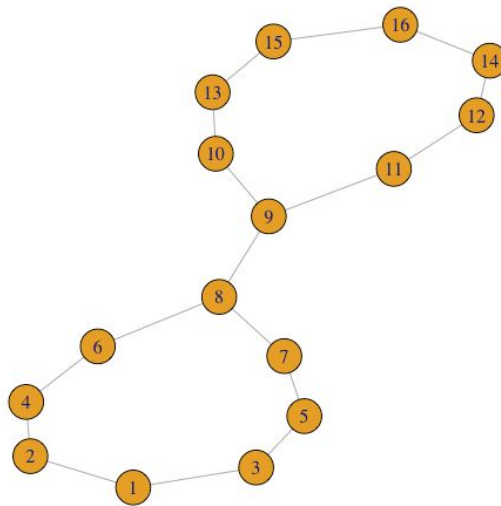


Since the mapper function operates on distances rather than points, we need to convert the figure-8 into a distance matrix before we pass it to the mapper. We'll choose the rest of the parameters so that our filter projects points vertically onto their x-coordinates, our image cover consists of 10 clusters (intervals, since we're working in one dimension) which overlap by 50%, and each interval can be separated into up to 10 sub-clusters.

```
m1x <- mapper1D(
  distance_matrix = dist(figure_8),
  filter_values = x_coord,
  num_intervals = 10,
  percent_overlap = 50,
  num_bins_when_clustering = 10)
```

Then, we'll plot the topological network using the igraph package:

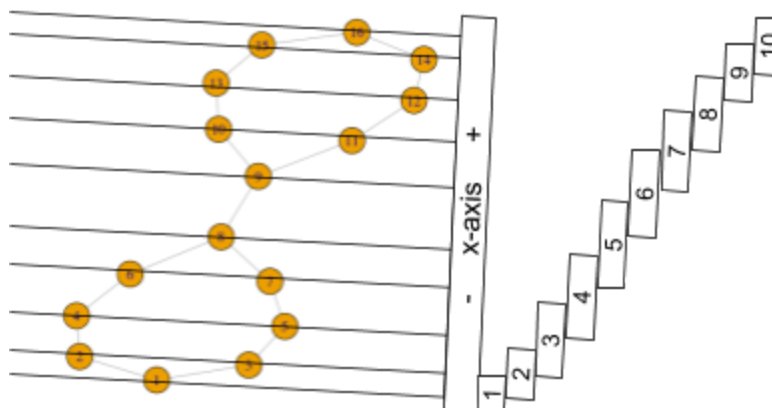
```
g1x <- graph.adjacency(m1x$adjacency, mode="undirected")
plot(g1x, layout = layout.auto(g1x) )
```



We can find out which of the 10 image clusters a given node corresponds to by looking at its “level”:

```
> mlx$level_of_vertex
[1] 1 2 2 3 3 4 4 5 6 7 7 8 8 9 9 10
```

This tells us that node 1 came from the first (least) image interval, nodes 2 and 3 came from the second image interval, ..., and node 16 came from the 10th (greatest) image interval.



We can display this same information, indexed by level, as follows:

```

> mlx$vertices_in_level
[[1]]
[1] 1

[[2]]
[1] 2 3

[[3]]
[1] 4 5

[[4]]
[1] 6 7

[[5]]
[1] 8

[[6]]
[1] 9

[[7]]
[1] 10 11

[[8]]
[1] 12 13

[[9]]
[1] 14 15

[[10]]
[1] 16

```

We can also recover the indices of the data points which comprise each vertex:

```

> mlx$points_in_vertex
[[1]]
[1] 5 6 7 8 18 19 20 30 31 32 33 43 44 45 55 56 57 58 68 69
70 80 81 82 83 93 94 95 96

```

```
[[2]]
```

```
[1] 5 17 30 42 55 67 80 93
```

```
[[3]]
```

```
[1] 8 21 33 46 58 71 83 96
```

```
[[4]]
```

```
[1] 4 17 29 42 54 67 92
```

```
[[5]]
```

```
[1] 21 34 46 59 71 84
```

```
[[6]]
```

```
[1] 4 16 29 54 79 92
```

```
[[7]]
```

```
[1] 9 34 59 72 84 97
```

```
[[8]]
```

```
[1] 3 9 16 22 41 47 66 72 79 85 91 97
```

```
[[9]]
```

```
[1] 3 22 28 35 41 47 53 60 66 78 85 91
```

```
[[10]]
```

```
[1] 10 35 48 60 73 98
```

```
[[11]]
```

```
[1] 15 28 40 53 78
```

```
[[12]]
```

```
[1] 2 15 27 40 65 90
```

```
[[13]]
```

```
[1] 10 23 48 61 73 86 98
```

```

[[14]]
[1]  2 14 27 39 52 65 77 90

[[15]]
[1] 11 23 36 49 61 74 86 99

[[16]]
[1]  1 11 12 13 14 24 25 26 36 37 38 39 49 50 51
52 62 63 64 74 75 76 77 87 88 89 99 100

```

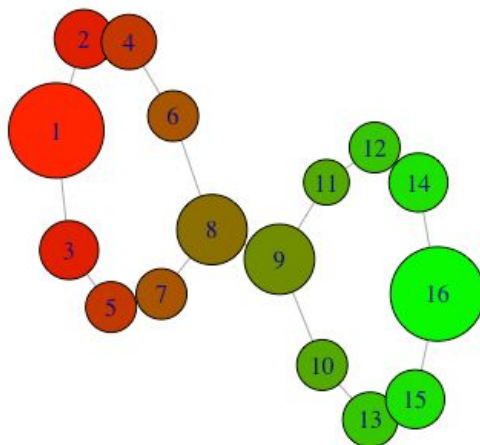
With this information, one can color and resize nodes:

```

my_resolution = 100
my_palette = colorRampPalette(c('red','green'))
my_max = max(m1x$level_of_vertex, na.rm=TRUE)
my_vector = m1x$level_of_vertex / my_max
my_colors = my_palette(my_resolution)[as.numeric(cut(my_vector,
                                                    breaks=my_resolution))]

glx <- graph.adjacency(m1x$adjacency, mode="undirected")
plot(glx, layout = layout.auto(glx),
     vertex_size = 50*log(vertex_size)/max(log(vertex_size)),
     vertex.color = my_colors )

```





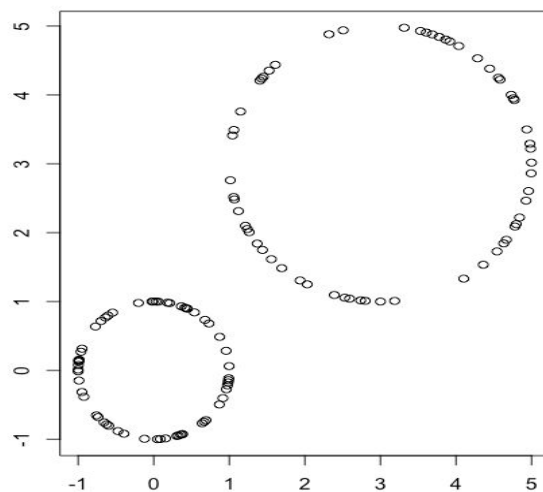
We can also run a barrage of statistical tests just like Ayasdi (but this must be hand-coded, as there are not yet off-the-shelf functions for these capabilities). TDAmapper also has a two-dimensional mapper, and graph interactivity is in development.

## R package TDA

Another R package, TDA, has persistent homology capabilities. Here, we will demonstrate on two circles:

```
Circle1 <- circleUnif(n = 60)
Circle2 <- circleUnif(n = 60, r = 2) + 3
Circles <- rbind(Circle1, Circle2)

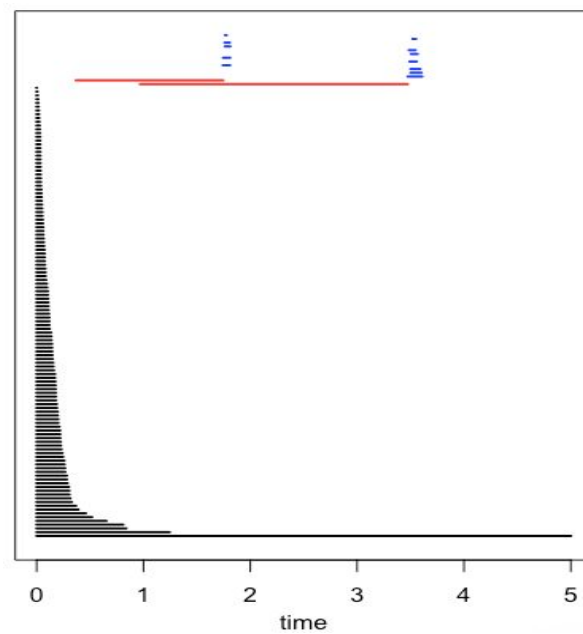
plot(Circles, xlab="", ylab="")
```



We'll create a barcode diagram for loops in dimensions 0, 1, and 2:

```
Diag <- ripsDiag(X = Circles, maxdimension = 2, maxscale = 5,
  library = "GUDHI", printProgress = FALSE)

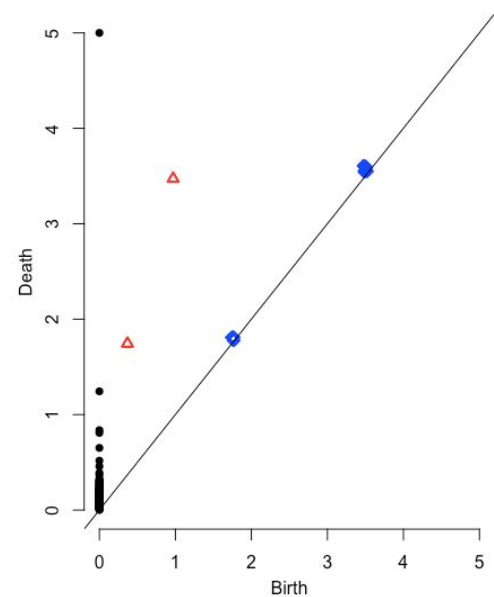
plot(Diag[["diagram"]], barcode = TRUE, main = "Barcode")
```



The zero-dimensional loops (connected components) are colored black, the one-dimensional loops (normal loops) red, and the two-dimensional loops (bubbles) blue.

This same information can be represented more compactly with a birth-death diagram:

```
plot(Diag[["diagram"]])
```



There are also functions for calculating the Bottleneck and Wasserstein distances, which measure dissimilarity between homology diagrams. Below, we calculate these distances between each individual circle.

```
Diag1 <- ripsDiag(X = Circle1, maxdimension = 1, maxscale = 5)
Diag2 <- ripsDiag(X = Circle2, maxdimension = 1, maxscale = 5)

> print(bottleneck(Diag1 = Diag1[["diagram"]],
  Diag2 = Diag2[["diagram"]], dimension = 1))

[1] 1.250809

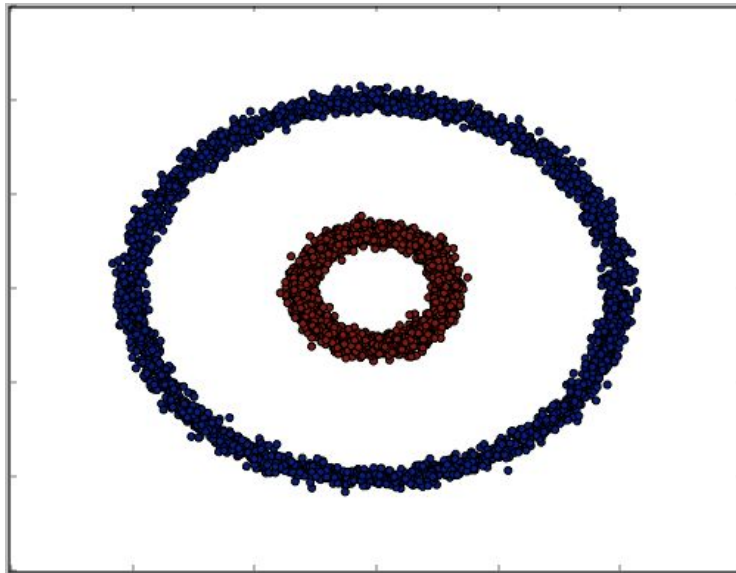
> print(wasserstein(Diag1 = Diag1[["diagram"]],
  Diag2 = Diag2[["diagram"]], p = 2, dimension = 1))

[1] 2.036909
```

## Python module KeplerMapper

KeplerMapper is an Python implementation of the Mapper algorithm. Like the R implementation TDAmapper, KeplerMapper works by feeding data into a mapper object and then visualizing the mapper. However, whereas TDAmapper accepts distances as the input to the mapper, KeplerMapper requires that the input consist of actual data points. Also, KeplerMapper uses d3.js to make its plots interactive. We'll demonstrate on two circles:

```
from sklearn import datasets
import matplotlib.pyplot as plt
data, labels = datasets.make_circles(n_samples=5000,
  noise=0.03,
                                     factor=.3)
plt.scatter(data[:,0], data[:,1], c=labels)
```



First, we create the mapper object and a dictionary to store the nodes, edges, and meta-information of the simplicial complex:

```
import km
mapper = km.KeplerMapper(verbose=1)
projected_data = mapper.fit_transform(data, projection=[0,1])
complex = mapper.map(projected_data, data, nr_cubes = 10)
```

Then, we can play with an interactive visualization of the complex:

```
mapper.visualize(complex, path_html="viz.html",
                 title="two_circles")
```



## References

<https://cran.r-project.org/web/packages/TDAmapper/TDAmapper.pdf>

<https://github.com/paultpearson/TDAmapper>

<http://www.kaisataipale.net/blog/2016/01/25/tdamapper-in-r/>

<https://cran.r-project.org/web/packages/TDA/vignettes/article.pdf>

[http://www.win.tue.nl/SoCG2015/wp-content/uploads/tutorials/150623\\_presentation\\_2.pdf](http://www.win.tue.nl/SoCG2015/wp-content/uploads/tutorials/150623_presentation_2.pdf)

<https://github.com/MLWave/kepler-mapper>

## TDA POTENTIAL

---

### Summary

Via R's TDAmapper, TDA was able to provide more granular information on a location tracking dataset than our previous hierarchical clustering methods. It was also able to detect teams with abnormally long accept times in a call center dataset.

### Segmentation via Location Tracking

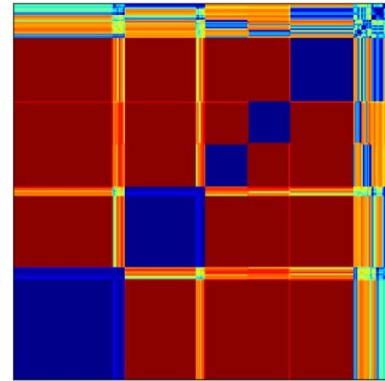
A location tracking dataset we analyzed included a count of visits to different location categories. In attempt to segment the user base, we performed hierarchical clustering on visit profiles within each category. We saw the highest degree of segmentation within the “Recreation and Leisure” category, which consisted of the following subcategories:

Recreation And Leisure	(400000000)	
>Stadiums/Arenas	(401000000)	(col 1)
>Recreation Centers	(402000000)	(col 2)
>Swimming Pool	(403000000)	(col 3)
>Athletic Fields	(404000000)	(col 4)
>Baseball	(405000000)	(col 5)
>Basketball	(406000000)	(col 6)
>Football	(407000000)	(col 7)
>Soccer	(408000000)	(col 8)
>Tennis	(409000000)	(col 9)
>Running	(410000000)	(col 10)
>Golf	(411000000)	(col 11)
>Gym And Fitness Centers	(412000000)	(col 12)
>Outdoors	(413000000)	(col 13)

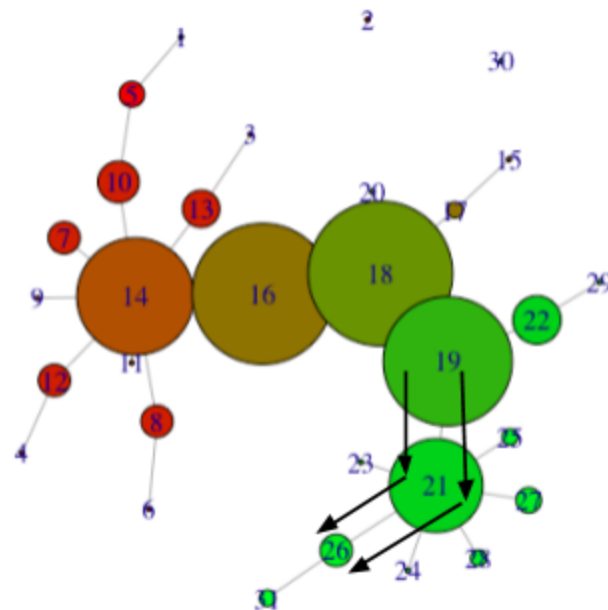
To perform hierarchical clustering, we created a dataset whose rows consisted of visit frequency (in %) for each of 13 subcategories above. Then, we computed the Euclidean distance matrix and sorted it via dendrogram. The resulting visualization revealed 6 clusters:

Recreation Category	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
Athletic Fields	1%	1%	1%	99%	1%	49%
Golf	0%	95%	1%	1%	0%	7%
Gym and Fitness	1%	1%	1%	0%	97%	6%
Outdoors	1%	1%	96%	0%	1%	13%
Recreation Centers	0%	0%	0%	0%	0%	7%
Stadiums and Arenas	97%	1%	2%	0%	1%	17%
Swimming Pools	0%	0%	0%	0%	0%	7%

High-end sports players/fans   Golfers   Hikers Campers   Recreational sports players   Gym rats   Everyone else

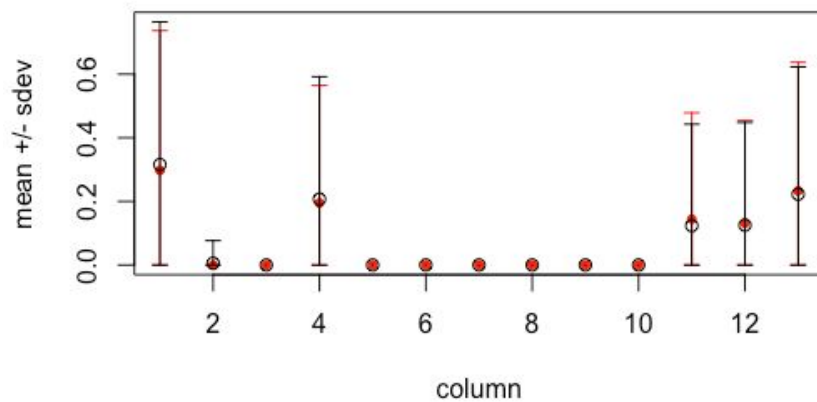


Using the Mapper algorithm, however, we see many more clusters. Moreover, we see the paths by which they are connected:

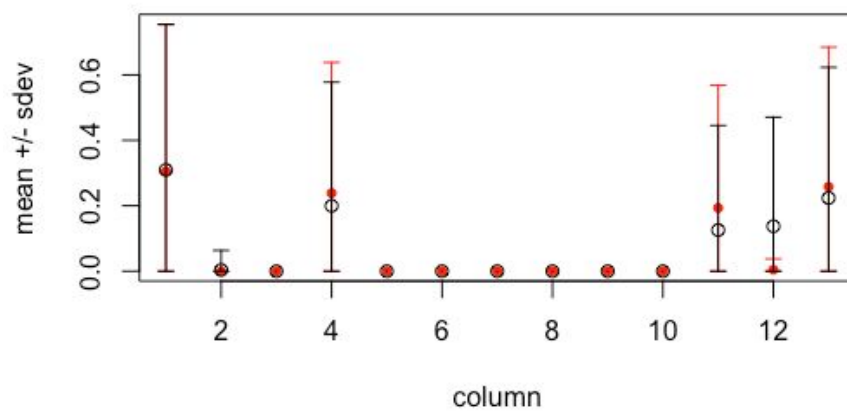


For the sake of example, we'll look more closely at the distinguishing characteristics of the high-visit flare consisting of nodes 19, 21, and 26. Below are graphs of column means for the in-node and out-of-node populations:

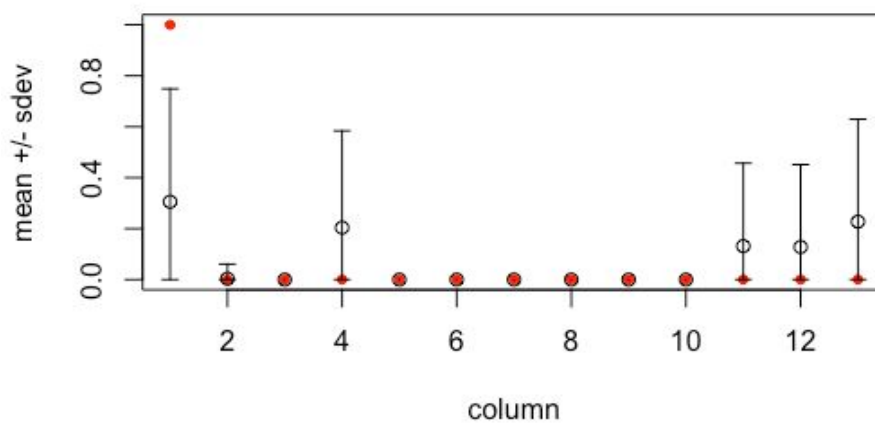
red=in node 19, black=out of node 19



red=in node 21, black=out of node 21



red=in node 26, black=out of node 26



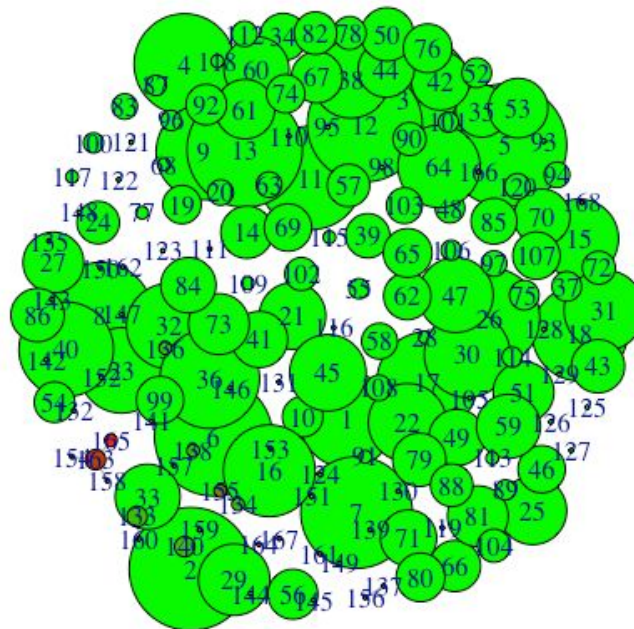


Node 19 has a normal profile, but node 21 has a low average in column corresponding to gym and fitness centers (col 12). Node 26 has a low average here as well, and also has low averages in columns corresponding to athletic fields, golf, and outdoors (cols 4, 11, 13). However, node 26 has a high average in stadiums and arenas (col 1).

We conclude that, for this example, the Mapper algorithm can reveal much finer granularity than hierarchical clustering.

## Call Center

TDA was also used to investigate a 10,000-record sample of call center data. The initial goal was to find trends over time, but after a week of little success this goal was replaced with an anomaly detection approach. The topological network below was structured by comparing each call's queue, team name, and location name, and colored by the amount of time needed to accept the call (green = short, red = long).



Ideally, calls should be accepted quickly. However, we can see that clusters 163 and 165, corresponding to the South Bend tech team, are associated with abnormally long accept times.