

The Data Scientist's Guide to

Topological Data Analysis

Justin Skycak

David Cieslak, PhD, Aanalytics

Prof. Mark Behrens, Department of Mathematics

Submitted in partial fulfillment of the requirements of the
Glynn Honors Program at the University of Notre Dame, October 2017.

ACKNOWLEDGEMENTS.

I would like to extend a word of thanks to my advisors, David Cieslak and Mark Behrens. As my industry advisor, David helped me think intuitively from the perspective of a data scientist: cut through the details, and get to the story. I am thankful for the opportunities David has given me to present my work to a mixed technical and business-oriented audience at Aanalytics, and for the helpful feedback I have received from them.

As my faculty advisor, Mark helped me think precisely from the perspective of a mathematician: the devil is often in the details. I am thankful for his introducing me to topology during my first year at Notre Dame, and for his willingness to offer his personal time to expand my knowledge of Topological Data Analysis during my last.

Taken together, combining David and Mark's perspectives helped me explain Topological Data Analysis to an audience who needs to understand it intuitively enough to see its applications to the real world, and precisely enough to troubleshoot and quality-check their methodology.

ABSTRACT.

Topological Data Analysis, abbreviated TDA, is a suite of data analytic methods inspired by the mathematical field of algebraic topology. TDA is attractive yet elusive for most data scientists, since its potential as a data exploration tool is often communicated through esoteric terminology unfamiliar to non-mathematicians. The purpose of this guide is to bridge the communication gap between academia and industry, so that non-mathematician data scientists may add current TDA methods to their analytic toolkits and anticipate new developments in the field of TDA.

The guide begins with an overview of Mapper, a TDA algorithm which has recently transitioned from academia to industry with commercial success. We explain the Mapper algorithm, demo open-source software, and present a handful of its commercial use-cases (some of which are original). Then, we switch to persistent homology, a TDA method which has not yet broken through to industry but is supported by a growing body of academic work. We explain the intuition behind homotopy, approximation, homology, and persistence, and demo open-source persistent homology software. It is hoped that the data scientist reading this guide will be inspired to give Mapper a try in their future analytic work, and be on the lookout for future developments in persistent homology that push it from academia to industry.

TABLE OF CONTENTS.

1. Mapper	4
1.1. Algorithm	5
1.2. Software	11
1.3. Use-Cases	20
1.3.1. Ayasdi	21
1.3.2. Aanalytics	30
2. Persistent Homology	35
2.1. Homotopy	36
2.2. Approximation	43
2.3. Homology	46
2.4. Persistence	51
2.5. Software.	53

1. MAPPER.

Section 1 elaborates on three main points surrounding Mapper:

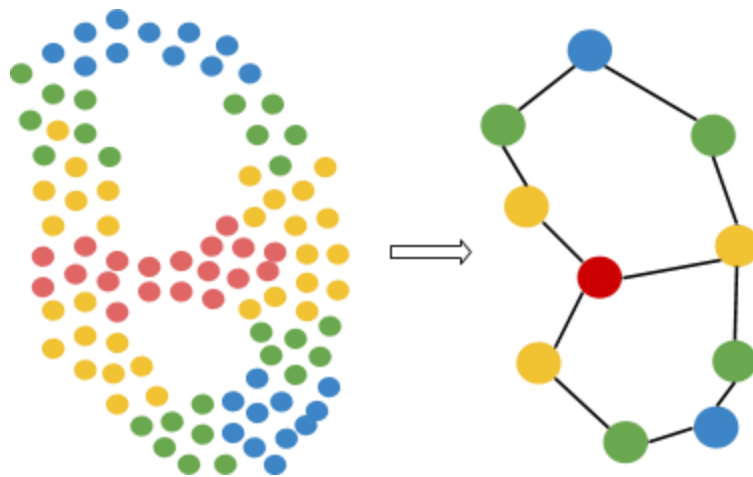
1. *Algorithm.* The Mapper algorithm maps high-dimensional data into smaller networks which retain the main topological features of the data and are easy to visualize.
2. *Software.* To run the Mapper algorithm on small to medium-size datasets, one can use the open source R package TDAmapper.
3. *Use-cases.* On a larger scale, Mapper has been used commercially by the company Ayasdi to forecast returns, detect fraud, aid in oil and gas exploration, plan ad campaigns, and discover biomarkers.

1.1. ALGORITHM.

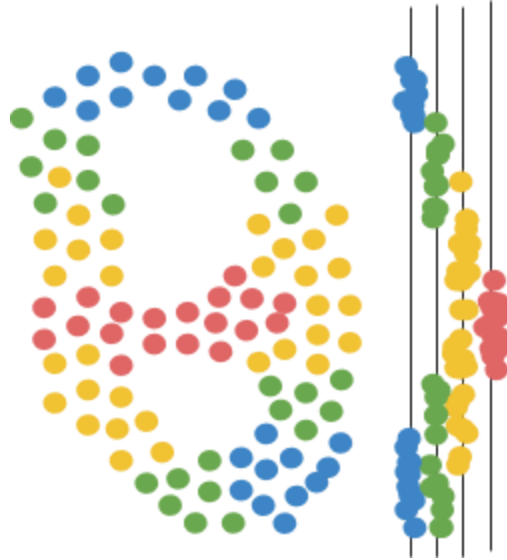
The Mapper algorithm (Singh et al. 2007) represents a data space's topology by converting it into a network. For example, suppose you have four classes of data: blue, green, yellow, and red. These classes might represent e.g. patient health or customer churn risk on a spectrum from favorable to unfavorable. If you could see in, say, 42 dimensions, you might notice that members of the same class tend to group together into clusters because they tend to have features in common. Maybe many of the sick patients have temperatures above 100 degrees Fahrenheit while most of the healthy ones have temperatures around 98, and maybe many of the high-risk churners have not signed up for the rewards program while most of the active customers have. You might also notice that, even though the data is 42-dimensional, there are a few distinct "paths" between favorable and unfavorable clusters, which may correspond to e.g. different treatment paths or customer journeys.

Unfortunately, we cannot see our data in a 42-dimensional space like we see objects in 3-dimensional space. However, using dimensionality-reduction algorithms, we can collapse the least important dimensions and focus on the ones that provide us with useful information, just like we can hold a paper flat in front of us to make it easier to read (here, we are reducing the dimensionality from 3 to 2). Mapper is one such dimensionality-algorithm, and it stands out from the rest because it preserves the topological features, such as paths, in our data.

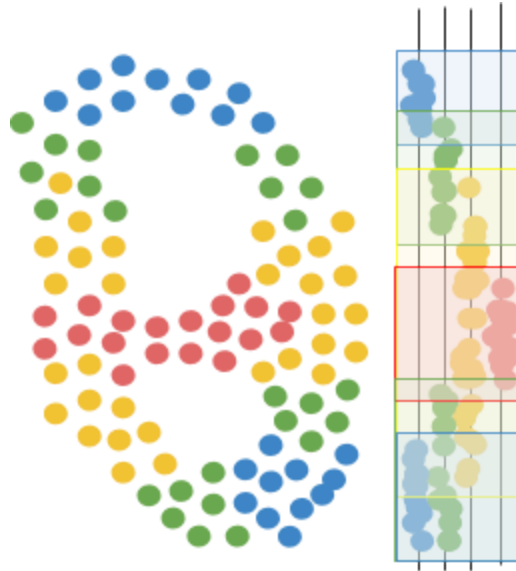
Mapper works by focusing our data through a lens, a particular key feature such as health or churn risk, and drawing a network, a 2-dimensional doodle that represents the overall shape of our data when seen through the lens. As an example, we'll walk through how the Mapper algorithm can algorithmically convert the dataset on the left to the network on the right:



First, Mapper focuses the data through the lens, a function which assigns a numerical value to each data point. The number can be a single feature of the data, like a patient's body temperature, or a combination of data features, like the total sum of phone calls, emails, and purchases made by a customer in the past month. To make this example easy to visualize, we'll choose a simple lens: vertical height.

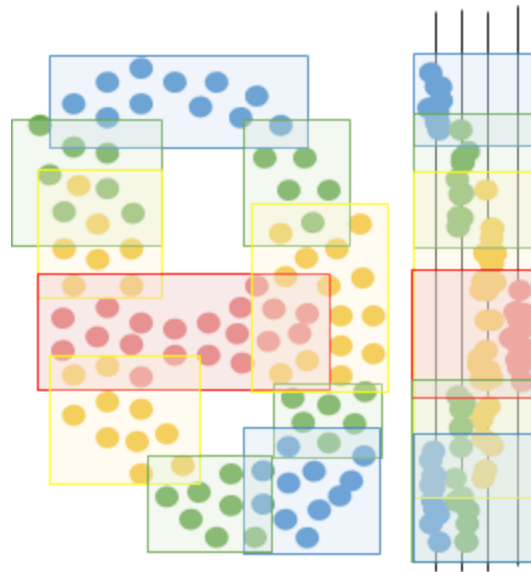


Next, Mapper uses a clustering algorithm to create a collection of overlapping clusters for the data, based on how far data points appear when seen through the lens. In other words, Mapper creates a “cover” for the mapped data. For example, a cover for a body temperature lens might consist of the intervals 90-96, 95-99, 98-101, 100-103, and 104-110 degrees Fahrenheit. Likewise, a cover for customer purchase total might consist of the intervals 0-10, 5-30, 20-50, 40-100, 75-150, 100-300, and 250-1000 dollars.



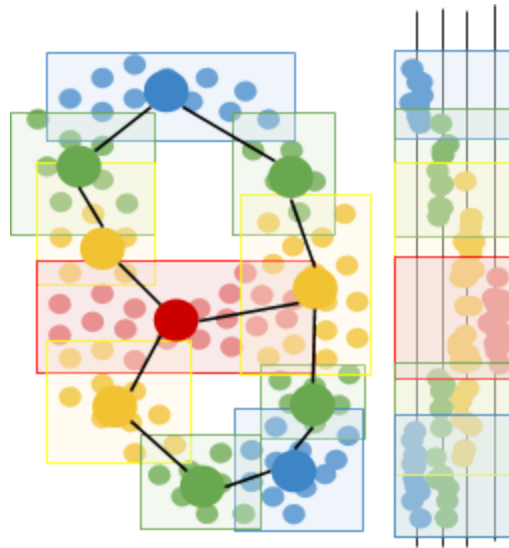
Then, Mapper runs another clustering algorithm *within* each original cluster, to separate each cluster into sub-clusters based on how far the data points actually are in the full data space (rather than just as seen through the lens). These sub-clusters represent different circumstances under which data points can be assigned the same value in the lens function. For example, two patients can both have the same high temperature of 101 degrees Fahrenheit, but one patient may be sick with a bacterial infection while the other may have the flu. On the basis of temperature alone, the two patients seem the same, but if variables from blood analysis are taken into account, the difference is clear. Likewise, two customers may have high churn risk due to a low number of purchases in the past year, but upon incorporating each customer's account balance and number of emails to the company, we might see that one customer is unhappy with the company whereas the other customer simply cannot afford

purchases anymore.



Finally, Mapper constructs a network by representing sub-clusters as nodes, and connecting nodes whose sub-clusters overlap (i.e. share data points). The nodes represent different segments of the population of data, segmented primarily by the lens metric and secondarily by all other factors. The connections or edges between nodes describe how the segments blend together, and can suggest potential paths for how data points may move through the data space. Knowledge of paths in the data space can be useful for businesses who want to learn how to engage e.g. their medium-activity customers and push them into the high-activity clusters over the next few months. Likewise, if a patient contracts a disease through a specific path, e.g. obesity via overeating, it may be more effective to treat the disease by trying to push them backwards along the same

path they came. For example, if a patient becomes obese due to some medication, it may be more effective to first try to counteract any other biomarkers that the medication pushes out of range, than to start by telling the patient to eat less and exercise more.

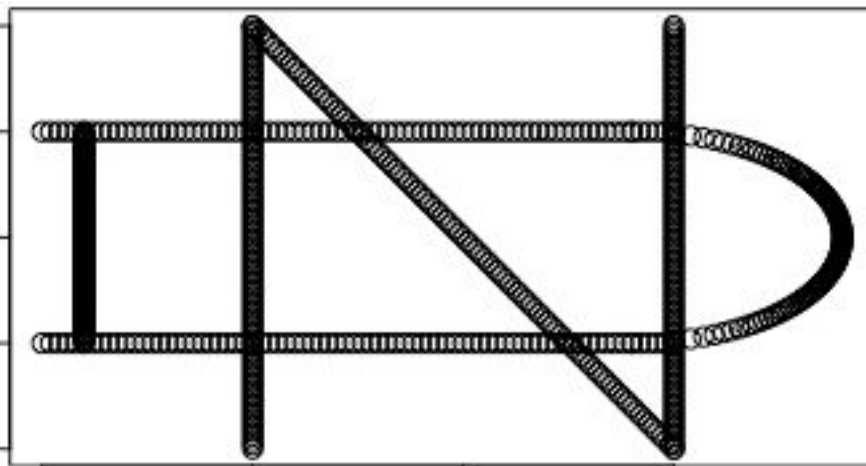


The resulting network also makes it easier to communicate the data visually, and gain exploratory insight from non-data-savvy domain experts who would not be able to interpret the data in numeric form.

1.2. SOFTWARE.

The Mapper algorithm has an open-source implementation in the TDAmapper package for R (Pearson et al. 2016). We'll begin its demonstration by creating a ND logo dataset and passing it into the Mapper function.

```
n_x <- c(rep(-.5,101),0.01*(-50:50),rep(.5,101))
n_y <- c(0.02*(-50:50),-0.02*(-50:50),0.02*(-50:50))
d_x <- c(rep(-0.9,101),0.02*(-50:25),0.02*(-50:25),
          0.4+sqrt(0.5^2-(0.01*(-50:50))^2))
d_y <- c(0.01*(-50:50),rep(0.5,76),rep(-0.5,76),
          0.01*(-50:50))
nd <- data.frame(x=c(n_x,d_x),y=c(n_y,d_y))
plot(nd)
```



TDAmapper's mapper function uses hierarchical clustering for the cover, which means that it accepts distances rather than points as input. This makes the mapper more flexible, since it is sometimes possible to compute distances or similarity scores even when the actual points are unknown - and if we do know the points, we can always compute distances between them. In this example, though, the practical implication of using distances is that we must convert the ND logo into a distance matrix before we pass it into the mapper.

We'll choose the rest of the parameters so that our lens projects points onto their x-coordinates, our primary clustering creates an image cover consisting of 10 intervals which overlap by 50%, and our secondary clustering separates each interval into up to 10 sub-clusters.

```
m <- mapper1D(  
  distance_matrix = dist(nd),  
  filter_values = c(n_x, d_x),  
  num_intervals = 10,  
  percent_overlap = 50,  
  num_bins_when_clustering = 10)
```

Then, we'll plot the topological network using R's igraph package.

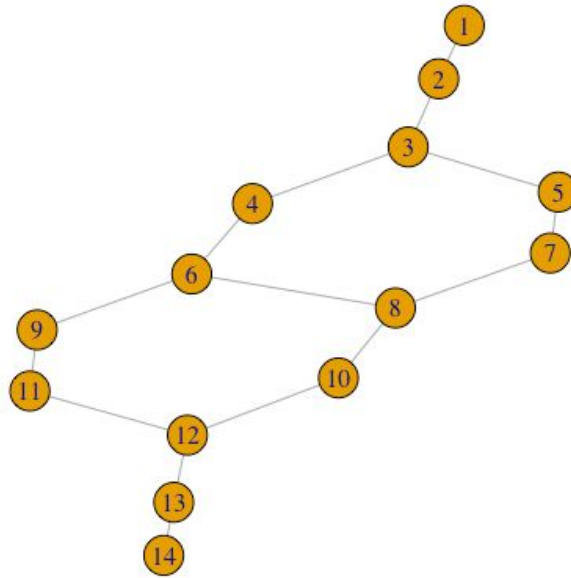
```

set.seed(0)

g <- graph.adjacency(m1x$adjacency, mode="undirected")

plot(g, layout = layout.auto(g))

```



Not only does TDMapper enable us to create the Mapper network, but it also tells us how the network was constructed at each step in the Mapper algorithm. For starters, we can find out which of the 10 image clusters a given node (sub-cluster) came from by looking at its level.

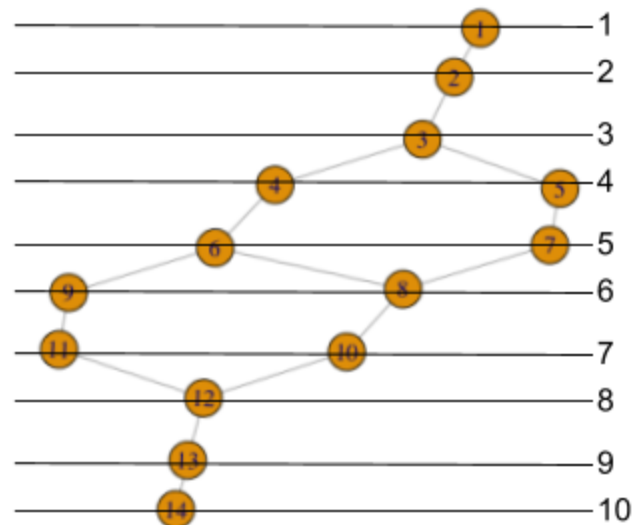
```

> m$level_of_vertex

1 2 3 4 4 5 5 6 6 7 7 8 9 10

```

This tells us that node 1 is a sub-cluster from the first (least) lens interval, and moreover, it is actually the full lens interval (i.e. the first lens interval did not split into separate sub-clusters). Similarly, node 2 is the full second lens interval, and node 3 is the full third lens interval. Nodes 4 and 5 are sub-clusters from the fourth lens interval, nodes 6 and 7 are sub-clusters from the fifth lens interval, and so on, up to node 14, which is a sub-cluster from the 10th lens interval.



We can display this same information, indexed by lens interval level rather than node number, as follows:

```
> m$vertices_in_level
[[1]] 1
[[2]] 2
```

```
[[3]] 3
[[4]] 4 5
[[5]] 6 7
[[6]] 8 9
[[7]] 10 11
[[8]] 12
[[9]] 13
[[10]] 14
```

We can also recover the indices of points in the original dataset, which comprise each node or sub-cluster.

```
> m$points_in_vertex
[[1]] 304 305 ... 498
[[2]] 1 2 ... 103 414 415 ... 430 490 491 ... 506
[[3]] 1 2 ... 121 423 424 ... 439 499 500 ... 515
[[4]] 104 105 ... 138 431 432 ... 448
[[5]] 507 508 ... 524
[[6]] 122 123 ... 155 440 441 ... 456
[[7]] 516 517 ... 532
[[8]] 139 140 ... 172 525 526 ... 541
[[9]] 449 450 ... 465
```



```

[[10]] 156 157 ... 190 533 534 ... 550
[[11]] 457 458 ... 474
[[12]] 173 174 ... 303 466 467 ... 480 542 543 ... 559 665
        656 657
[[13]] 191 192 ... 303 475 476 ... 480 551 552 ... 569 645
        646 ... 657
[[14]] 560 561 ... 654

```

With this information, we can color nodes according to their lens cluster level, and resize nodes according to how many data points they contain.

```

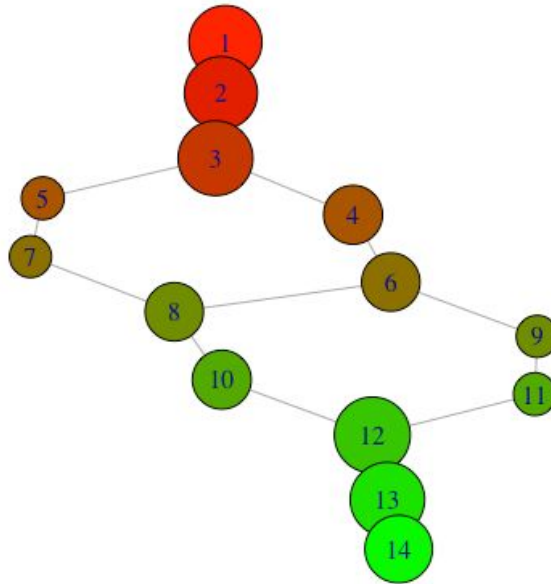
my_resolution = 100
my_palette = colorRampPalette(c('red','green'))
my_max = max(m$level_of_vertex, na.rm=TRUE)
my_vector = m$level_of_vertex / my_max
my_colors = my_palette(my_resolution)[as.numeric(cut(
        My_vector, breaks=my_resolution))]
g <- graph.adjacency(m$adjacency, mode="undirected")
vertex_size <- unlist(lapply(m1x$points_in_vertex,
        function(x) length(x)))
plot(g, layout = layout.auto(g1x),

```

```

vertex.size = 30*log(vertex_size)/
              max(log(vertex_size)),
vertex.color = my_colors)

```

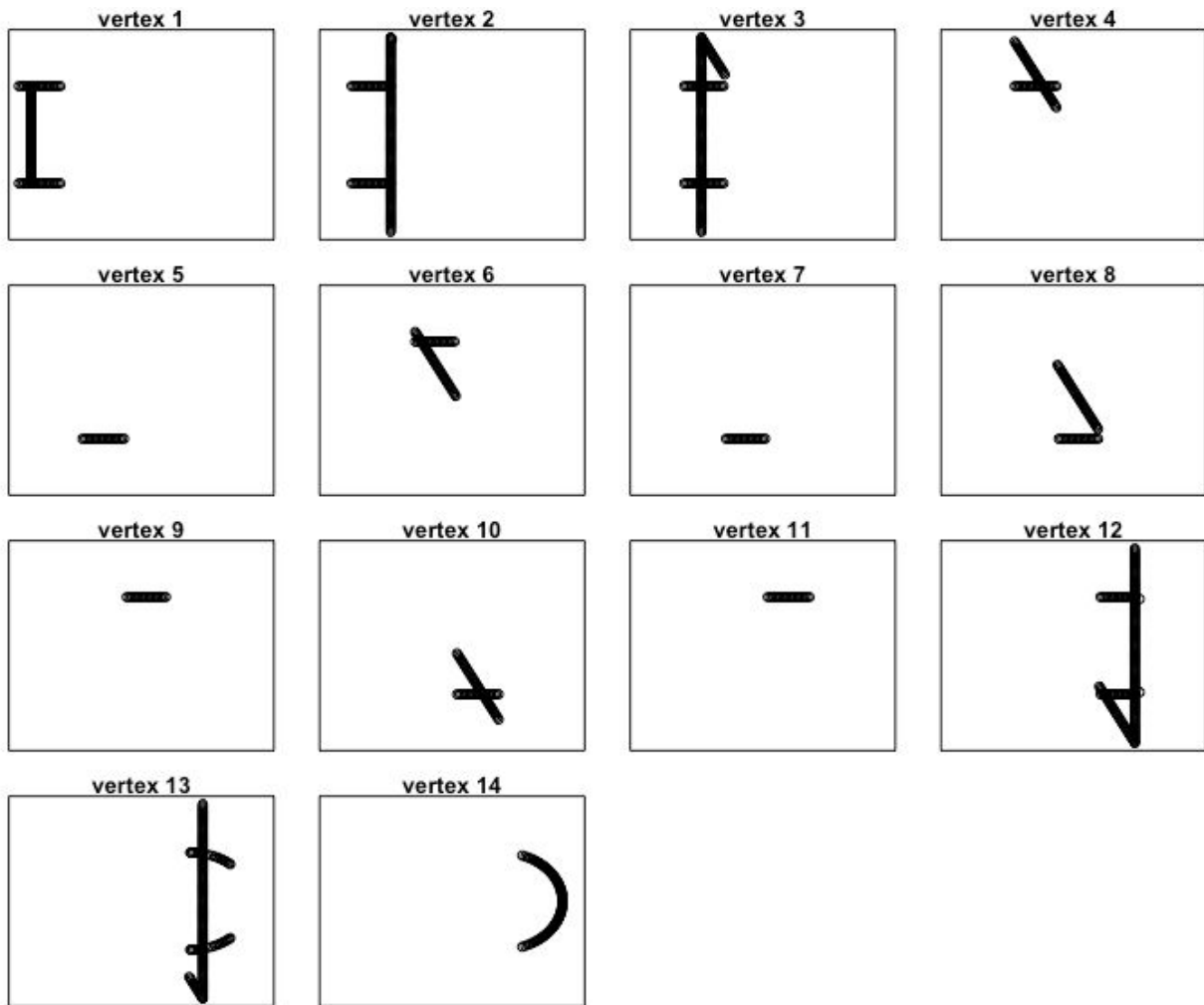


We can also look at what the nodes represent in the original plot:

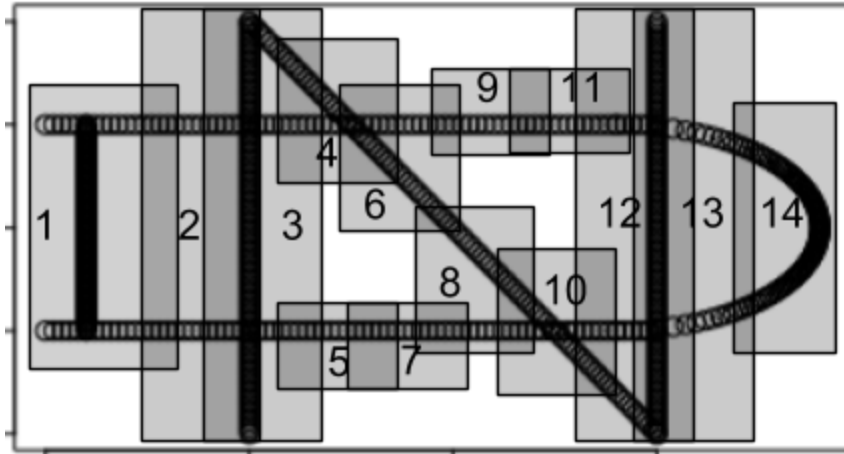
```

par(mfrow=c(4,4))
for(i in 1:length(m$points_in_vertex)){
  plot(nd[m$points_in_vertex[[i]],],
       xlim=c(-1,1),ylim=c(-1,1),
       xaxt='n',yaxt='n',
       main=paste('vertex',i))}

```



By identifying the vertices on the original plot, we can see how the output network got its shape.



For more in-depth analysis, one can also run a barrage of statistical tests to discover key factors contributing to differences between nodes. However, the relevant data must be passed manually from the mapper object to R's native statistical functions, as there are not yet off-the-shelf functions built into the TDAmapper's mapper object.

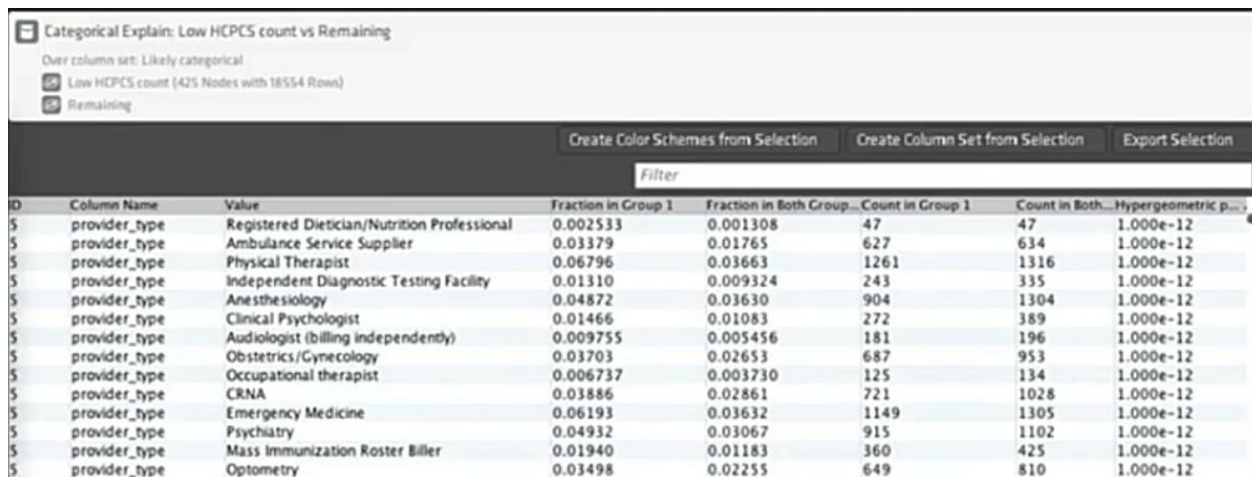
1.3. USE CASES.

Section 1.3 covers industry use-cases of Mapper at two companies:

1. *Ayasdi*, a commercial software company whose commercial Mapper software can be used to forecast returns, diagnose denied claims, detect fraud, identify oil wells and drilling machine failures, target campaign ads, and discover biomarkers.
2. *Aunalytics*, the data science software & consulting company at which the author is employed. Here, Mapper (via R's TDAmapper) provided granular insights on a location tracking dataset, and revealed insights in a sparse call-center dataset even though there was little cohesion in the resulting network.

1.3.1. AYASDI.

The commercial company Ayasdi developed commercial Mapper software and sells a subscription service to clients who wish to create topological network visualizations of their data. Their implementation is similar to R’s TDAmapper, except that it is heavily optimized to crunch large-scale datasets consisting in the millions of records. Furthermore, it has an “explain” function which automates the process of differentiating clusters via statistical testing. “Explain” works 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

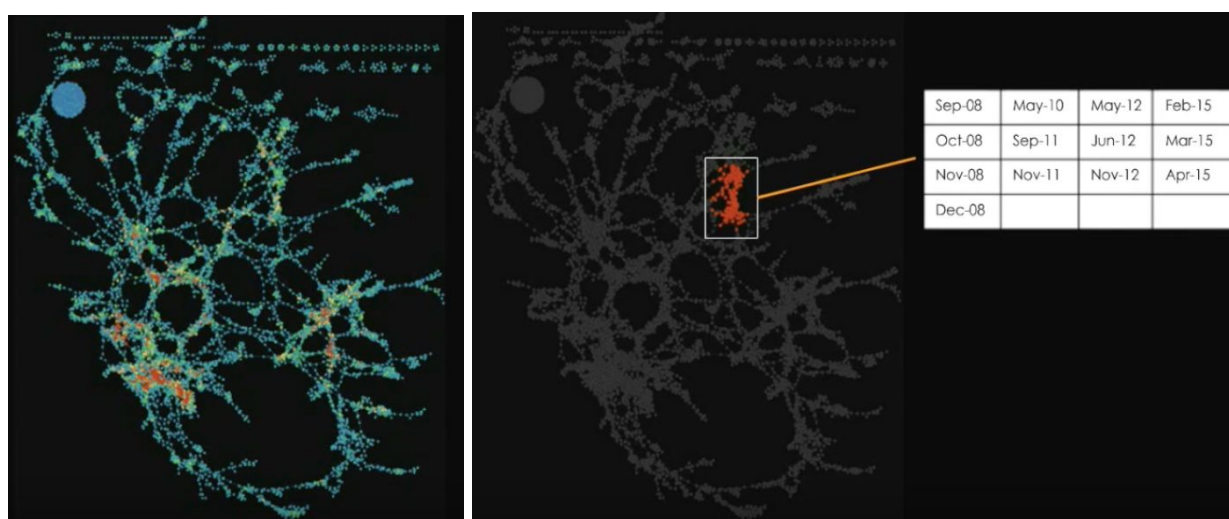
ID	Column Name	Value	Fraction in Group 1	Fraction in Both Group...	Count in Group 1	Count in Both...	Hypergeometric p...
\$	provider_type	Registered Dietician/Nutrition Professional	0.002533	0.001308	47	47	1.000e-12
\$	provider_type	Ambulance Service Supplier	0.03379	0.01765	627	634	1.000e-12
\$	provider_type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

In this section, we explore several commercial use-cases of Ayasdi’s software. 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.

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 (Roche et al. 2015). The nodes are colored by year.

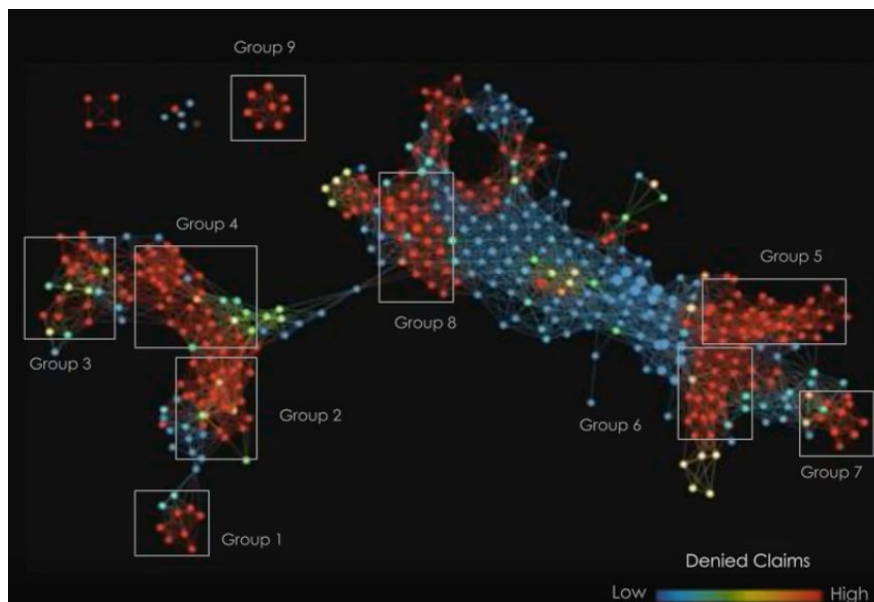


We see that the map is spread out over time, which indicates repeated patterns over time. For example, the group of highlighted nodes corresponds to high-volatility and high-stress conditions. This suggests the following strategy to forecast from an initial date: locate neighboring dates on the map, use their price trajectories to build a distribution of changes in price for each asset, and use mean or median for predictions. Then, individual predicted asset price-changes

can be aggregated to yield higher-level predictions, i.e. for each market sector.

Diagnosing Denied Claims

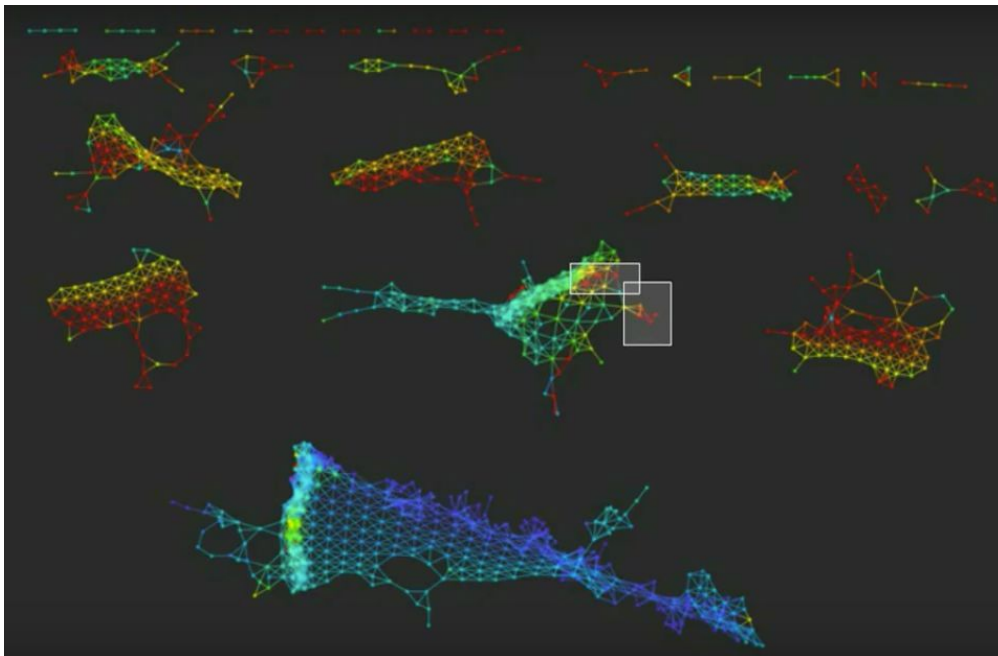
Simple denial patterns, consisting of only a few patterns, usually account for only a small portion of a denial backlog. However, Ayasdi's software has been used to find complex patterns in infusion and oncology medical necessity denials, accounting for up to 65% of the denial backlog ("Machine Intelligence," 2015). The following topological network was constructed by applying the Mapper algorithm to 5 million individual claims - its structure is determined by similarity between claims, and its nodes are colored according to how often the claims were accepted or denied on average.



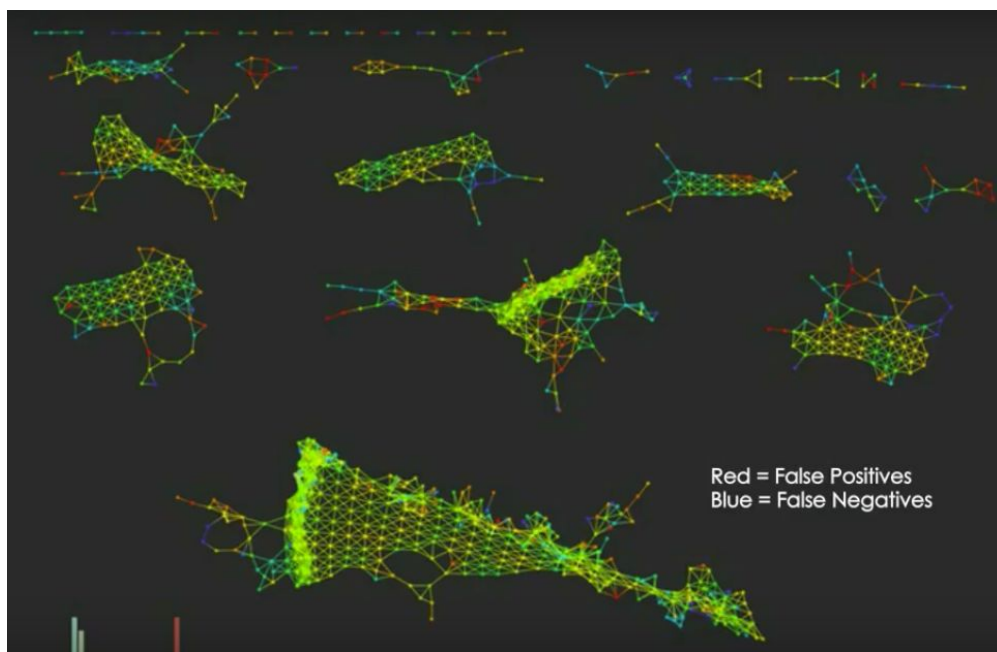
By locating several groups in the network and analyzing the group statistics, analysts were able to gain enough information to advise action pre-submission by modifying the final coding or supporting diagnosis, or at the point of care by seeking pre-authorization or reconsidering a procedure.

Detecting Fraud

The topological network below is based on the CMS public health claims dataset, which consists of over 9 million claims, 36 thousand providers, and 3600 unique codes (Rogers and Grahnen, 2015). The network structure is determined by similarity in how providers practice, while the node color is determined by medicare payment amount.

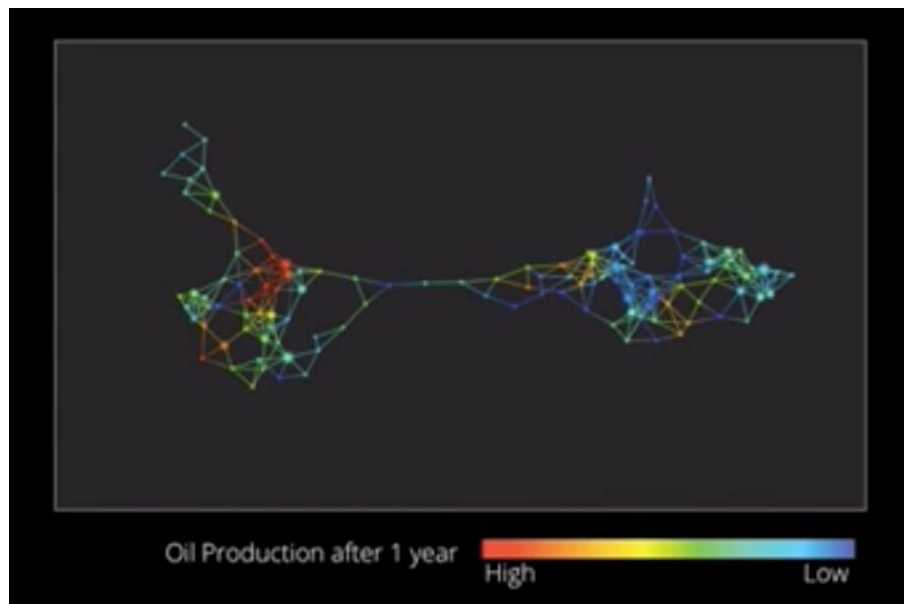


One can identify leads for investigation by looking for outlier providers who are getting paid abnormally much compared to other similar providers (two such groups are boxed in the network above). One can also improve detection models using this topological network: by recoloring the network nodes according to model performance (e.g. false positive rate), one can find groups for which the model performs poorly - and by running statistical tests to discover how these groups differ most significantly from the rest of the population, one can identify specific parameters which the model may have learned incorrectly.

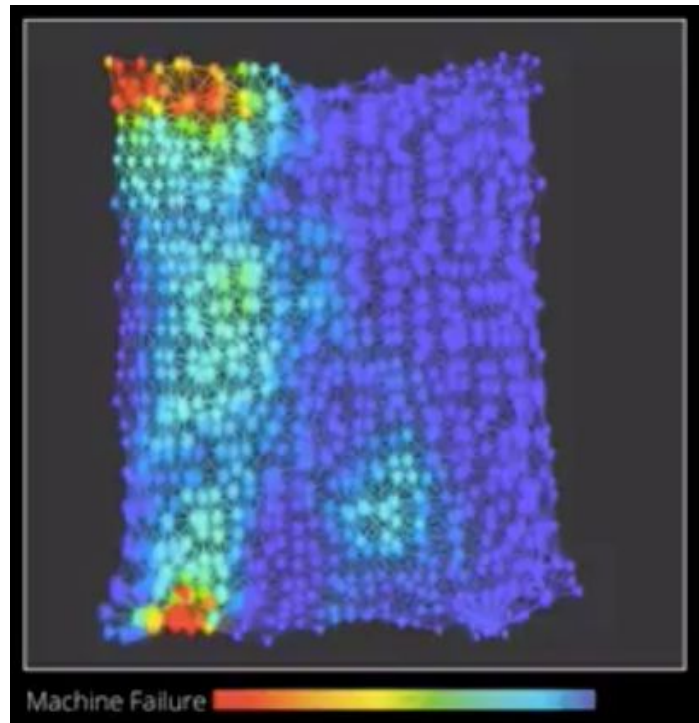


Oil and Gas Exploration

Below is an example of a topological network whose structure is based on drilling location, and whose color is based on the amount of oil recovered there (Parulekar and Johnson, “Analyzing Oil,” 2014). This information can be useful in identifying new locations most likely to be oil-rich.

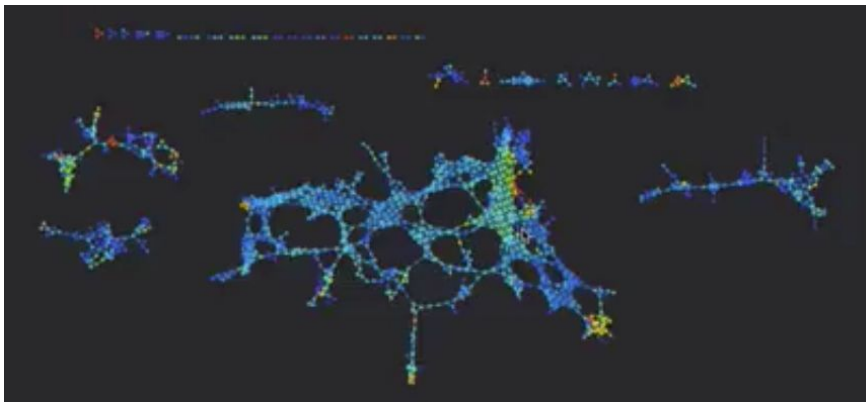
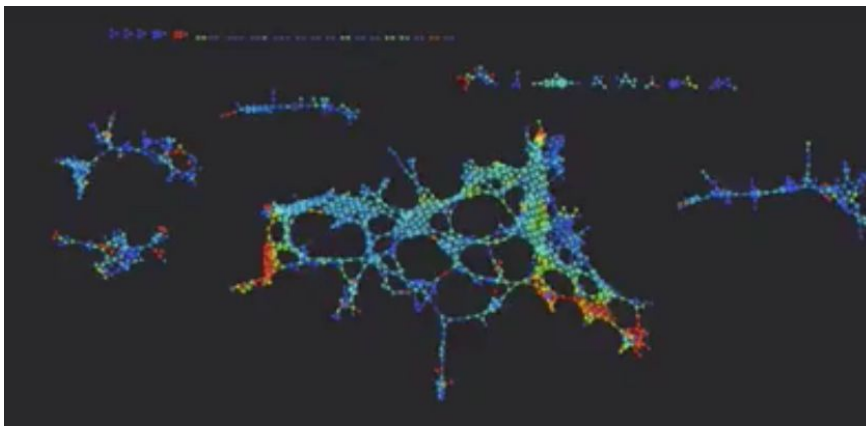
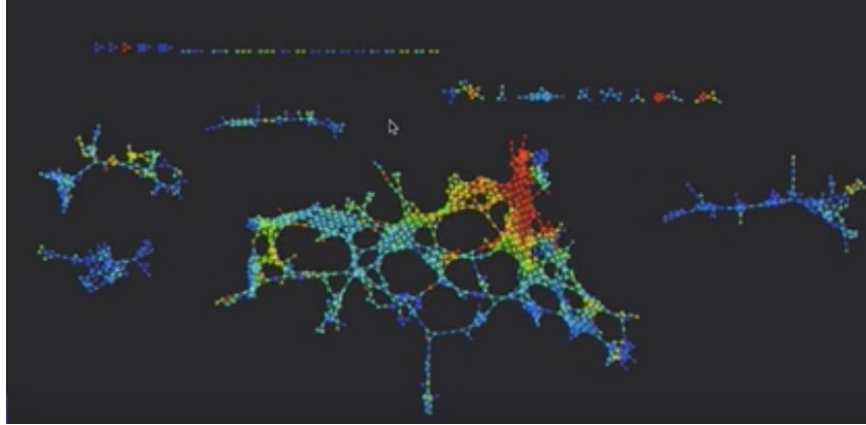


Topological networks can provide valuable information about the drilling equipment, as well. Below is a network whose structure is determined by a number of system state readout variables, and whose color determined by frequency of failure (red = high, blue = low). By better understanding the correlation between system status and failure frequency, one can anticipate critical events and avoid unnecessary replacements.



Campaign Ad Targeting

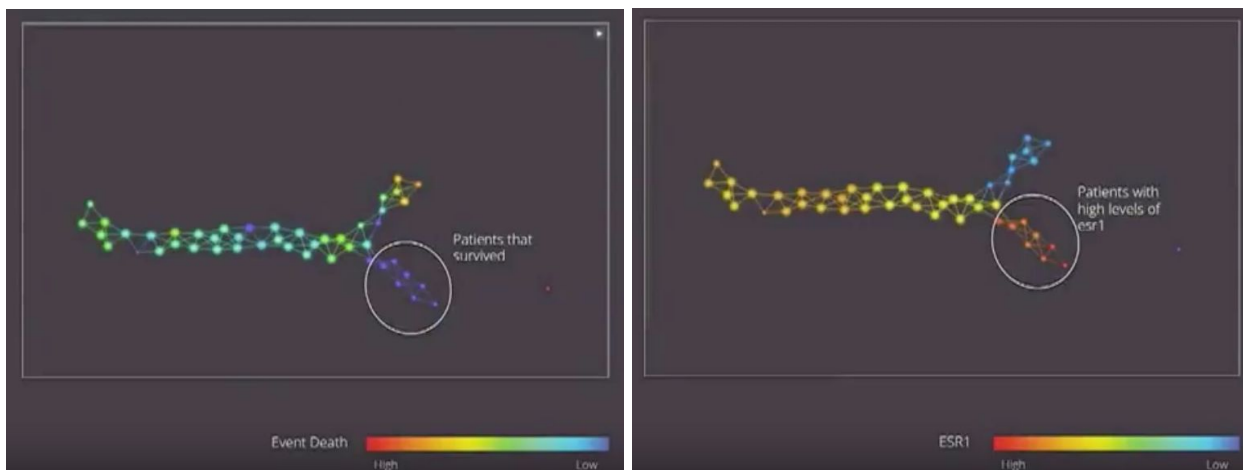
Based on data on 37,000 Twitter users who tweeted about Chris Christie, a topological network structured by account similarity and colored by word frequency can be used to identify niche conversations that are good targets for campaign ads (Parulekar and Johnson, “Campaign Planning,” 2014). Shown below (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.

Biomarker Discovery

Below is a topological network generated by data from 272 breast cancer patients, where the structure is based on similarity in genes expressed by patients (Parulekar and Johnson, "Ayasdi Cure," 2014). 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*.



1.3.2. AUNALYTICS.

Aunalytics is the data science software & consulting company at which the author is employed. Here, Mapper (via R's TDAmapper) outperformed hierarchical clustering in providing granular insights on a location tracking dataset, and detected call-center teams which took abnormally long times to accept calls even when there was little cohesion in the network.

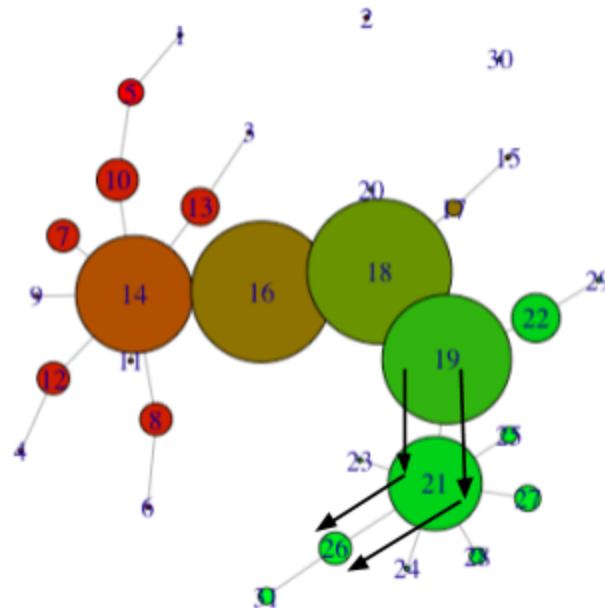
Segmentation via Location Tracking

A location tracking dataset from an Aunalytics digital media client included a count of visits to different location categories. In attempt to segment the user base, the author originally performed hierarchical clustering on visit profiles within each category. The highest degree of segmentation was observed within the "Recreation and Leisure" category, which consisted of the following subcategories: Stadiums/Arenas, Recreation Centers, Swimming Pool, Athletic Fields, Baseball, Basketball, Football, Soccer, Tennis, Running, Golf, Gym and Fitness Centers, Outdoors.

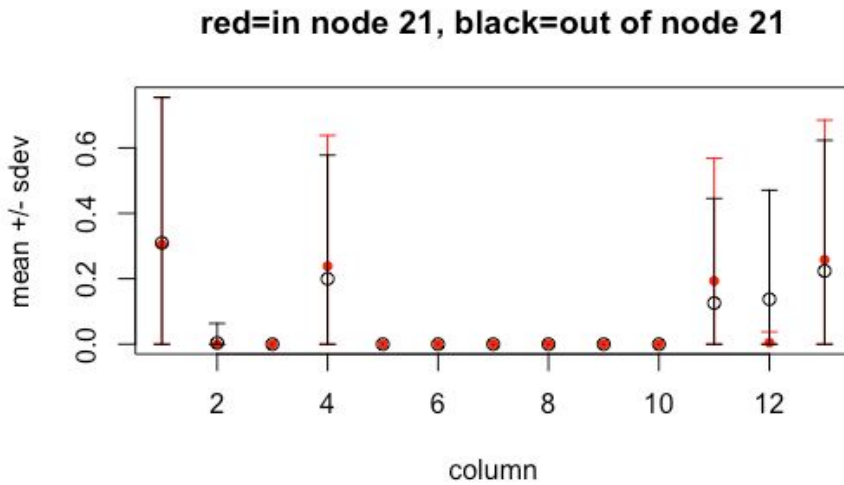
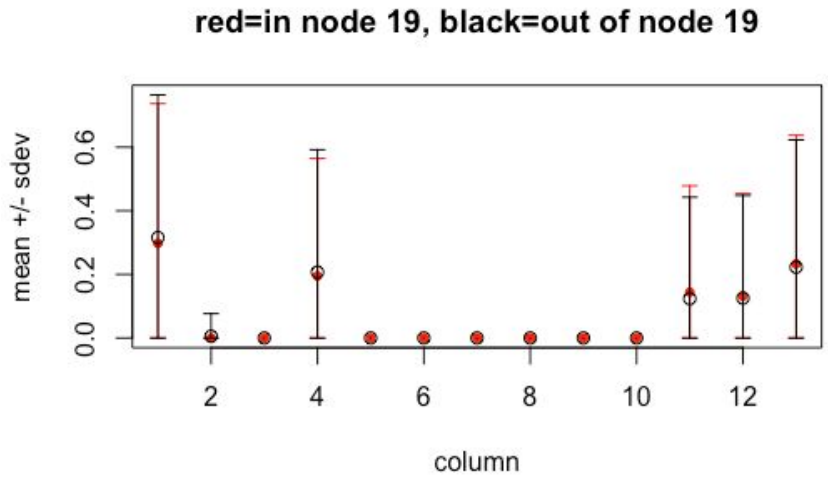
To perform hierarchical clustering, the author created a dataset whose rows consisted of visit frequency (in %) for each of 13 subcategories above, and then computed and sorted a Euclidean distance matrix via dendrogram. The resulting visualization revealed 6 clusters.

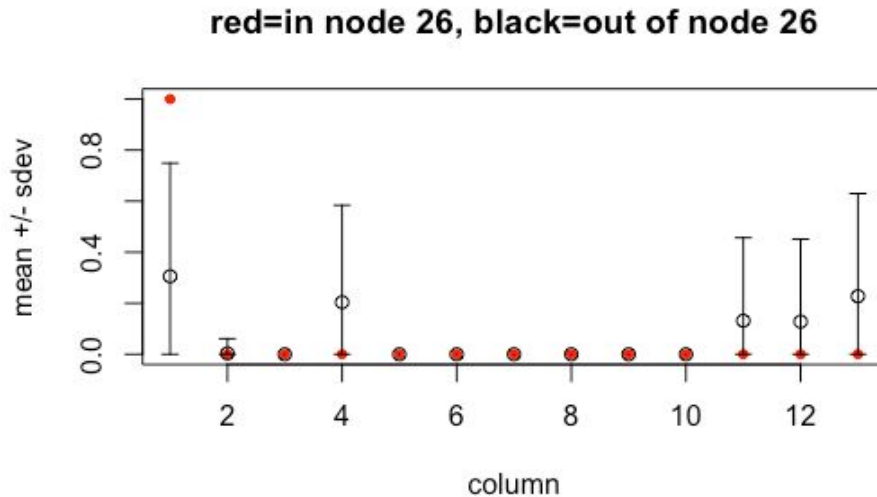


Using the Mapper algorithm, however, revealed many more clusters. Moreover, it revealed paths by which clusters were connected.



For example, we will inspect 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.



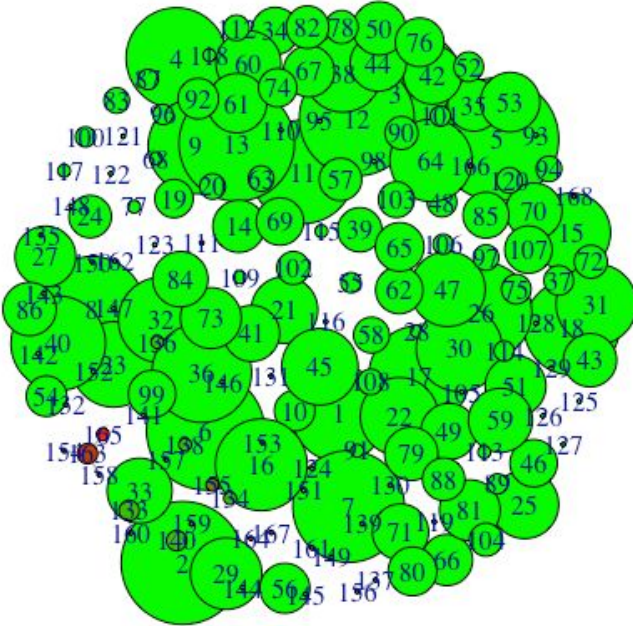


Node 19 has a normal profile, but node 21 has a low average in column corresponding to gym and fitness centers (column 12). Node 26 has a low average here as well, and also has low averages in columns corresponding to athletic fields, golf, and outdoors (columns 4, 11, 13). However, node 26 has a high average in stadiums and arenas (column 1). We conclude that, for this example, the Mapper algorithm revealed much finer granularity than hierarchical clustering.

Call Center

Mapper 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 tech team from a particular location, are associated with abnormally long times to accept calls. This example demonstrates how Mapper can reveal insights from data even when the data is sparse and non-cohesive and the resulting network does not appear to contain any clear paths between nodes.

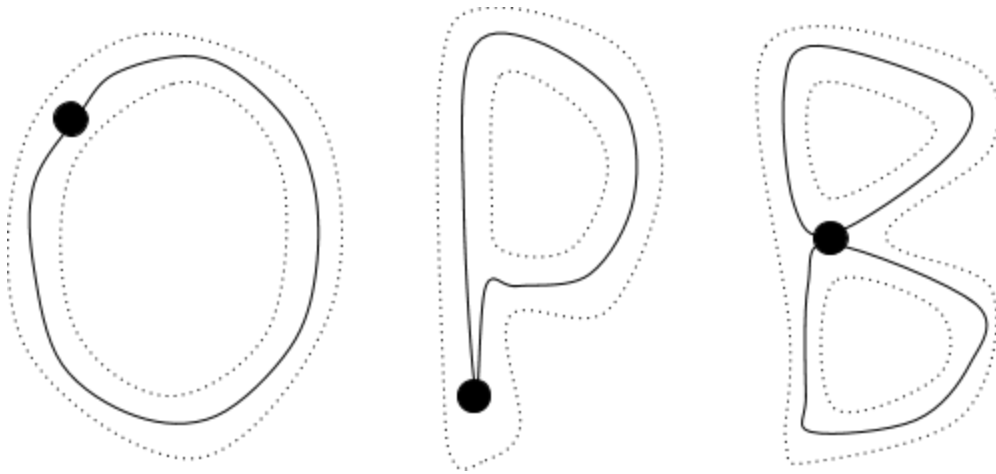
2. PERSISTENT HOMOLOGY.

Section 2 elaborates on four main points surrounding Persistent Homology:

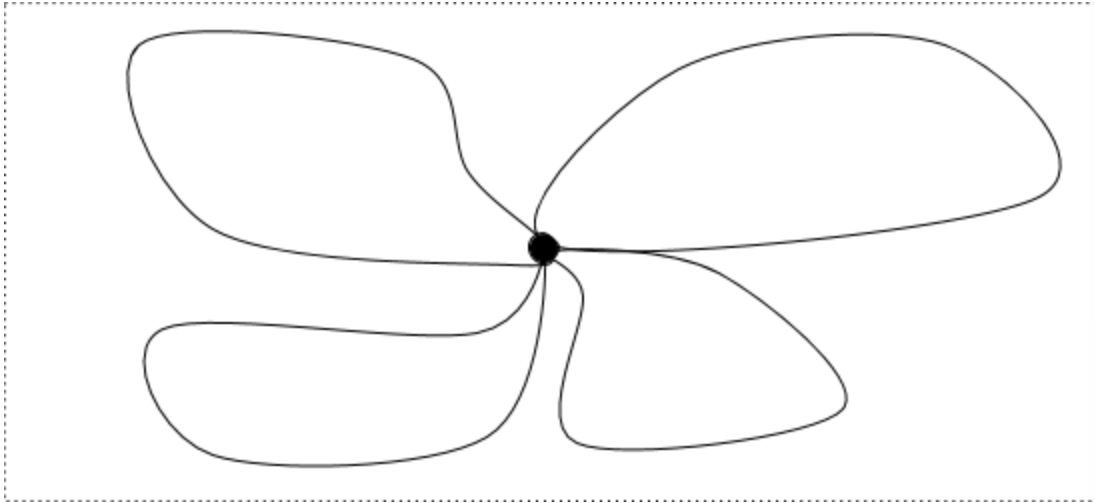
1. *Homotopy*. Algebraic topology aims to describe the connectivity of any arbitrary space. It does this by computing the homotopy, or number of “loops” in each dimension.
2. *Approximation*. In computational topology, datasets can be interpreted as samples taken from an underlying topological space, and for any given margin of error a topology can be constructed to approximate the underlying space.
3. *Homology*. Homotopy groups are extremely difficult to compute in high dimensions. Homology is a similar concept which can be easier to compute.
4. *Persistence*. Persistence barcode plots show which topological features persist through many scales of the data, and can be used to calculate similarity between different spaces.
5. *Software*. To compute persistent homology of small to medium-size datasets, one can use the open source R package TDA.

2.1. HOMOTOPY.

To describe a shape's connectivity, you can count the number of "loops" in the shape, starting from an initial point called the basepoint. For example, the letters "O" and "P" are said to have the same connectivity because they each have one loop, whereas the letters "O" and "B" are said to have different connectivity because "O" has one loop and "B" has two loops. Algebraic topology takes this idea of classifying shapes based on how many loops they have, and extends it to spaces of arbitrarily many dimensions (Carlsson, 2009).



For example, 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?

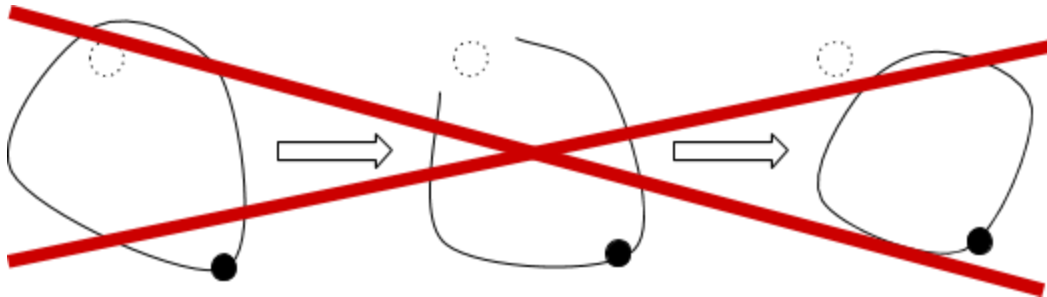


So that we're not stuck counting loops for eternity, we say that loops are equivalent if they can be continuously deformed into each other. Given a loop, we can drag it across the plane, twist it around, stretch it, compress it, and so on, and it will still be the same loop. The only thing we aren't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

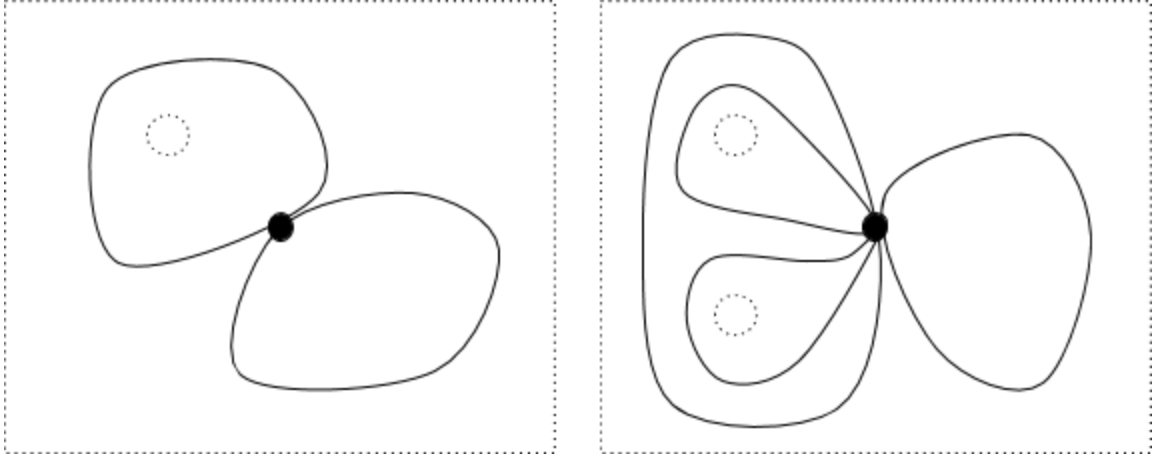


To count loops using this equivalence, we need to figure out how many *distinct* loops there are. In the plane above, it's easy -- there is only one distinct loop. However, if there is a hole in the sheet, the loop with the hole inside is not

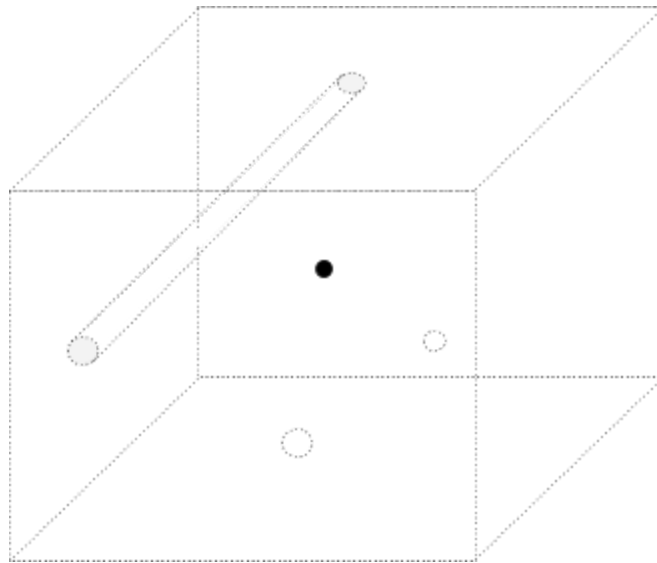
the same as the loop with the hole outside: if the hole is inside a loop, the only way to move it to the outside is to tear the loop, and that's cheating.



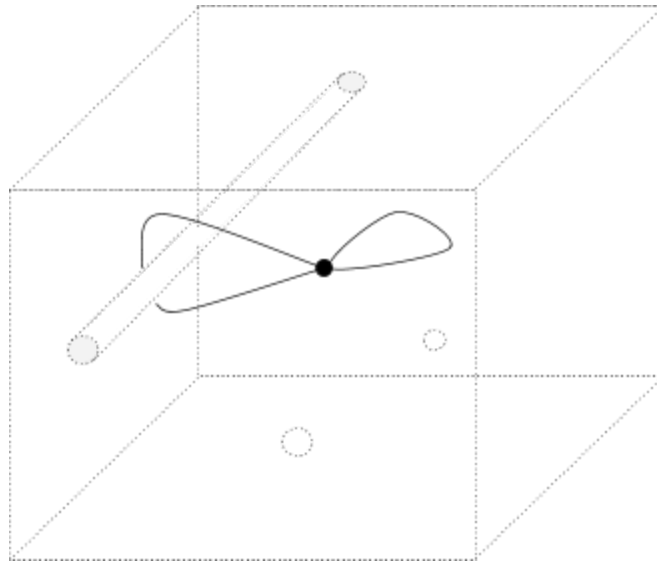
A plane with one hole, then, has two loops: one loop with the hole inside and another loop with the hole outside. The loop with the hole inside can be deformed into any other loop with the hole inside, and the loop with the hole outside can be deformed into any other loop with the hole outside. For a plane with two holes, we have four distinct loops: one around each hole, one around both holes, and one around neither hole. Topologists have a word, *homotopy*, for the kind of non-tearing deformations we're imagining, and they call the collection of distinct loops a *homotopy group*.



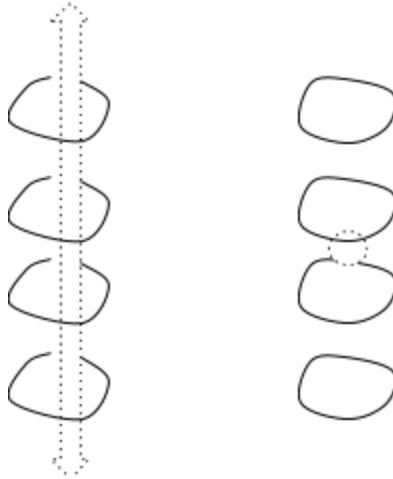
In higher-dimensional spaces, we need to count loops *in each dimension*. For example, consider a three-dimensional space with two points and a line removed. This looks like a cube of cheese that has two air bubbles and has been poked by a toothpick, infinitely enlarged.



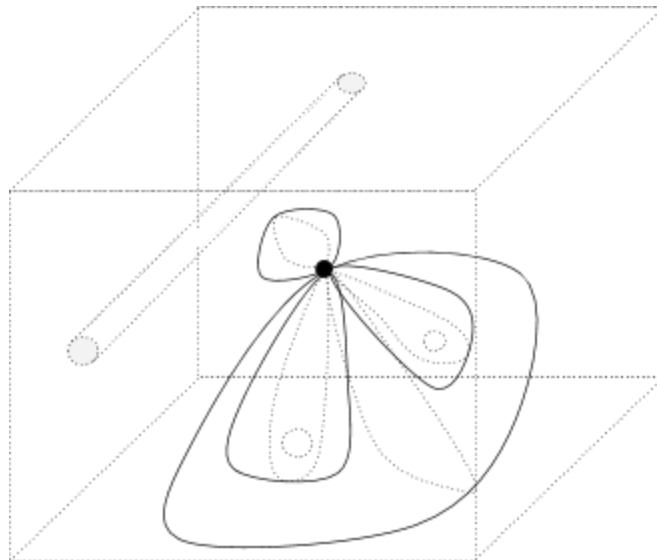
There are two one-dimensional loops: a circle with the line inside, and a circle with the line outside.



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.



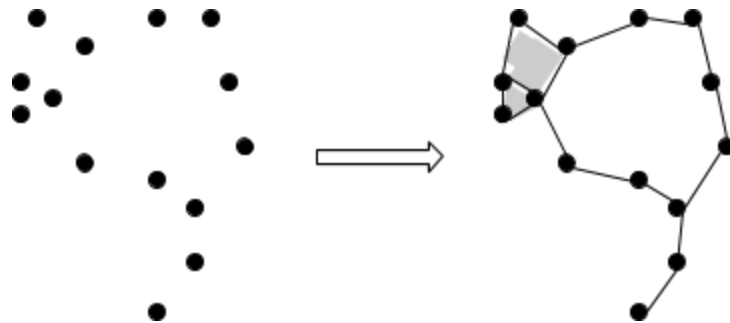
There are four two-dimensional loops: a bubble with no points inside, a bubble with one point inside, a bubble with the other point inside, and a bubble with both points inside. The missing line does not affect the number of distinct two-dimensional loops because the line spans the entire space and consequently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



In 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.

2.2. APPROXIMATION.

Real-world datasets are not explicit topological spaces. Rather, they are collections of points *sampled* from topological spaces, and the goal of topological data analysis is to analyze these point clouds and 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 (Carlsson, 2009).

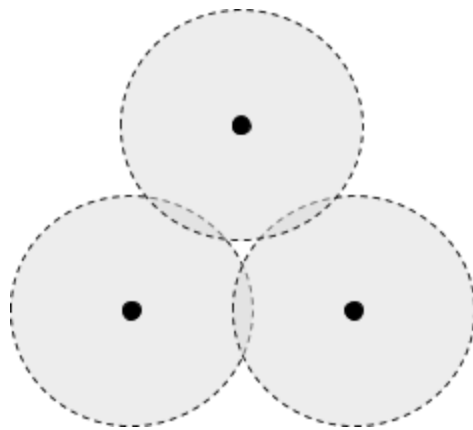


The main idea behind turning point clouds into simplicial complexes is to put epsilon-balls, or error margins, around points and use the overlaps to determine the connections in the simplicial complex. The constructions generated using different values of epsilon will correspond to topological approximations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

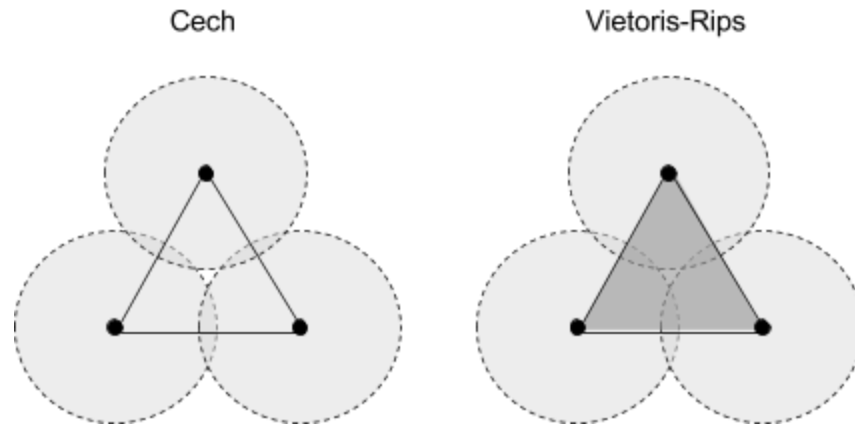
For example, one generates the Čech complex of a point cloud by adding an n -simplex whenever the intersection of all n balls is nonempty. A more

computationally efficient method, which generates the Vietoris-Rips complex, adds an n -simplex whenever the intersection of every *pair* in the n balls is nonempty, and contains the Čech complex as a subcomplex.

To see how these complexes are constructed, first consider these three points with pairwise-overlapping balls.



The associated Čech 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 is homotopy-equivalent to the full space (this also applies to the Vietoris-Rips complex, since it contains the Cech complex). Therefore, in theory, our approximation should have the same topological features as the actual space, provided we use enough points.

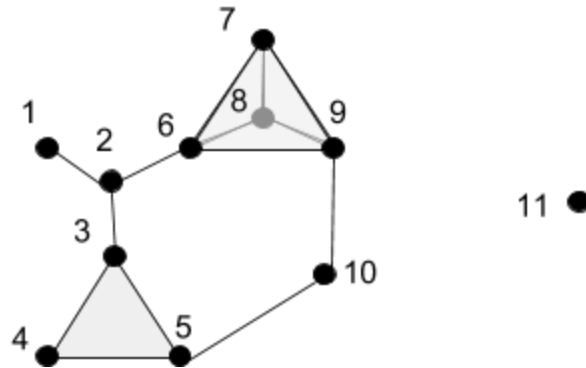
2.3. HOMOLOGY.

Unfortunately, homotopy groups are extremely difficult to compute in high dimensions. However, there is a similar concept, homology, which can be calculated on simplicial complexes via linear algebra (Carlsson, 2009). Like homotopy, homology also counts the number of loops of each dimension in a space, where loops are allowed to shift along the boundary of a higher-dimensional component on the space.

In simplicial complexes, the lowest-level components are points, followed by segments, and then triangles, and then solid tetrahedrons, and so on. You can think of an n th level component as an n -dimensional triangle, or more formally, an n -simplex.

Level-0 component	point	0-dimensional triangle	0-simplex
Level-1 component	segment	1-dimensional triangle	1-simplex
Level-2 component	triangle	2-dimensional triangle	2-simplex
Level-3 component	tetrahedron	3-dimensional triangle	3-simplex
...
Level- n component		N -dimensional triangle	n -simplex

For example, consider the following simplicial complex:

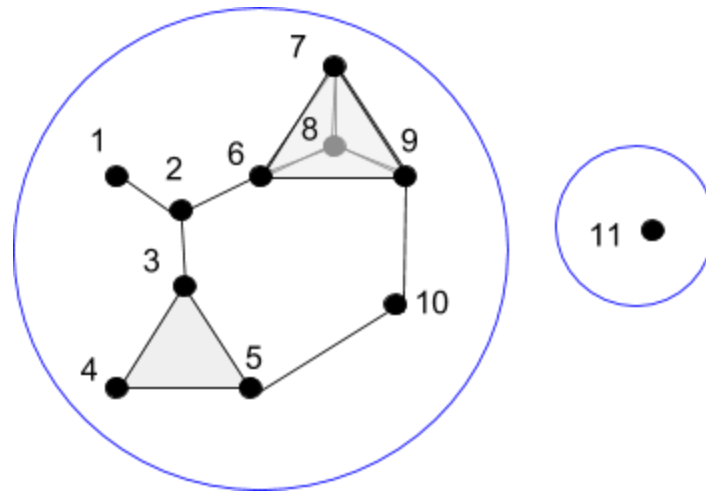


This complex contains the following simplices:

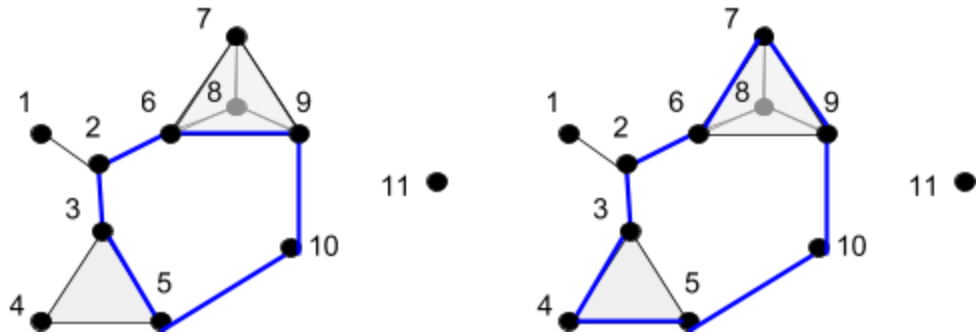
- 0-simplices: {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}, {10}, {11}
- 1-simplices: {1,2}, {2,3}, {2,6}, {3,4}, {3,5}, {4,5}, {5,10}, {6,7}, {6,8}, {6,9}, {7,8}, {7,9}, {8,9}, {9,10}
- 2-simplices: {3,4,5}, {6,7,8}, {6,7,9}, {6,8,9}, {7,8,9}

The 0th homology of this complex consists of those 0-dimensional loops along 0-simplices, which cannot be deformed into one another by shifting along the boundary of a 1-simplex. Put more simply, it consists of those points which cannot be shifted to one another along an edge. Since points {1} through {10} are all connected by paths through edges, they are all viewed as the same in first homology -- but since {11} is not connected to any other point by an edge, it is different. Thus, the 0th homology of the above complex consists of two loops, which intuitively represent “point islands”: points {0} through {10} inhabit the

first island, while point {11} is the lone inhabitant of the second island.

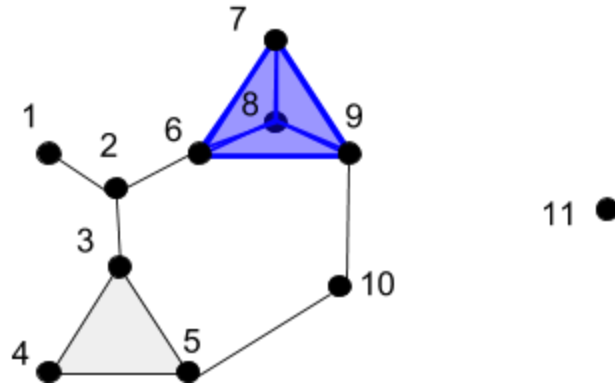


The 1st homology of this complex consists of those 1-dimensional loops along 1-simplices, which cannot be deformed into one another by shifting along the boundary of a 2-simplex. Put more simply, it consists of those edge loops which cannot be shifted to one another along triangles. For example, the two edge loops below are the same because $\{3,5\}$ can be shifted to $\{3,4\}$ and $\{4,5\}$ on the triangle $\{3,4,5\}$, and $\{6,9\}$ can be shifted to $\{6,7\}$ and $\{7,9\}$ on the triangle $\{6,7,9\}$.



It turns out that, for this complex, there is no other edge loop that is different from the inner edge loop. Therefore, the 1st homology of the above complex consists of one loop, which intuitively represents the “donut hole” in the complex.

The 2nd homology is the last homology for this complex, because it contains no simplices beyond the 2nd dimension. It consists of those 2-dimensional loops along 2-simplices, which cannot be deformed into one another by shifting along the boundary of a 3-simplex. Put more simply, it consists of those closed (think “inflatable”) surfaces which cannot be stretched into to one another along solid tetrahedrons. The 2-simplices $\{3,4,5\}$, $\{6,7,8\}$, $\{6,7,9\}$, $\{6,8,9\}$, and $\{7,8,9\}$ together form the surface of a tetrahedron, and the solid tetrahedron itself is not included in our complex. Therefore, the 2nd homology of the example complex consists of one loop, the surface of the tetrahedron.



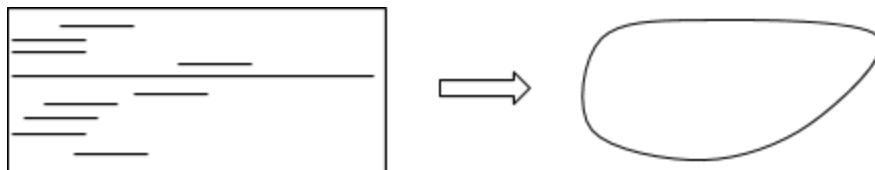
The number of components in the n th homology is called the n th Betti number, and by comparing the Betti numbers of different spaces, we can gain an idea of how topologically similar or different they are. For example, our complex had three nontrivial homologies, giving rise to Betti numbers $(2, 1, 1)$. If we found two other datasets, constructed complexes on them, computed their homologies, and found that they had Betti numbers $(2, 2, 1)$ and $(1, 5, 3)$, then we would interpret the process generating the data of our example $(2, 1, 1)$ complex as more similar to the process generating the data of the $(2, 2, 1)$ complex, than the process generating the data of the $(1, 5, 3)$ complex.

Measuring topological similarities between spaces by comparing their Betti numbers is just the tip of the iceberg. We have lots of mathematical machinery (e.g. probability and calculus) to analyze transformations between points, and Betti numbers give us a way to interpret entire spaces at points. This opens the door to studying not only relationships between parameters in a system, but also relationships between systems with completely different parameters.

2.4. PERSISTENCE.

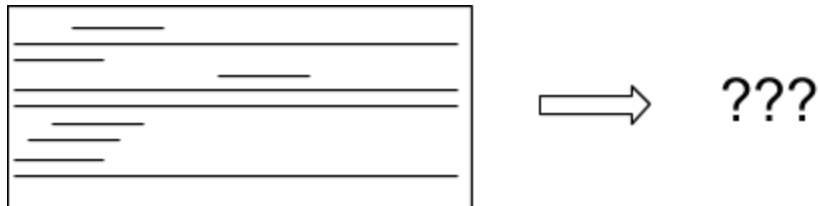
When we're interested in topological features which persist across many scales of the data, we need consider all values of epsilon for the simplicial complex we construct on our data. This is the idea behind persistent homology: we can figure out which topological features persist over the full range of scale by making a plot that says whether a particular homology component was detected at some value of epsilon (Carlsson, 2009).

Barcode plots have values of epsilon on the horizontal axis, and a list of homology components on the vertical axis. Each row, then, corresponds to a homology component, and is shaded at values of epsilon where the homology component appears. For example, if the barcode plot below represents first homology, then a single long bar tells us that the dataset has a main loop which persists across many scales. Therefore, the dataset must look something like a circle.



In a barcode plot with many long bars, many loops means many possibilities for the space. In these situations, we cannot always get a clear idea of

what our space “looks” like, but we can still quantify the degree to which two datasets share the same topology by computing a similarity index between their persistence barcodes.

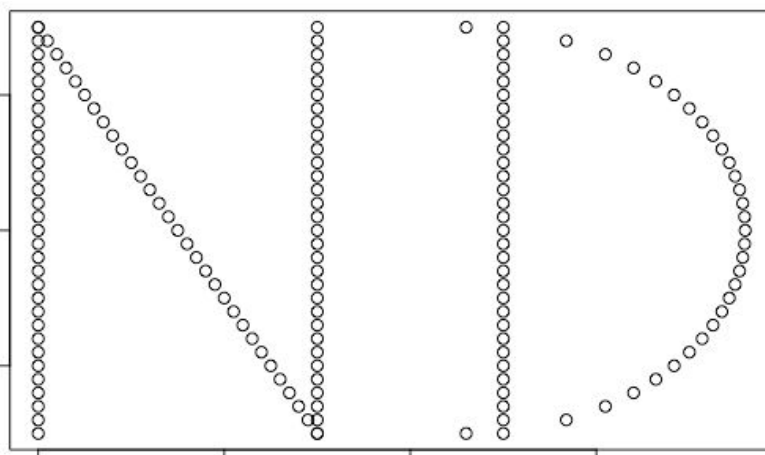


Persistence barcode distributions are currently an active area of research, and they can encode persistence of not only homology components, but also centrality, density, and other network measures.

2.5. SOFTWARE.

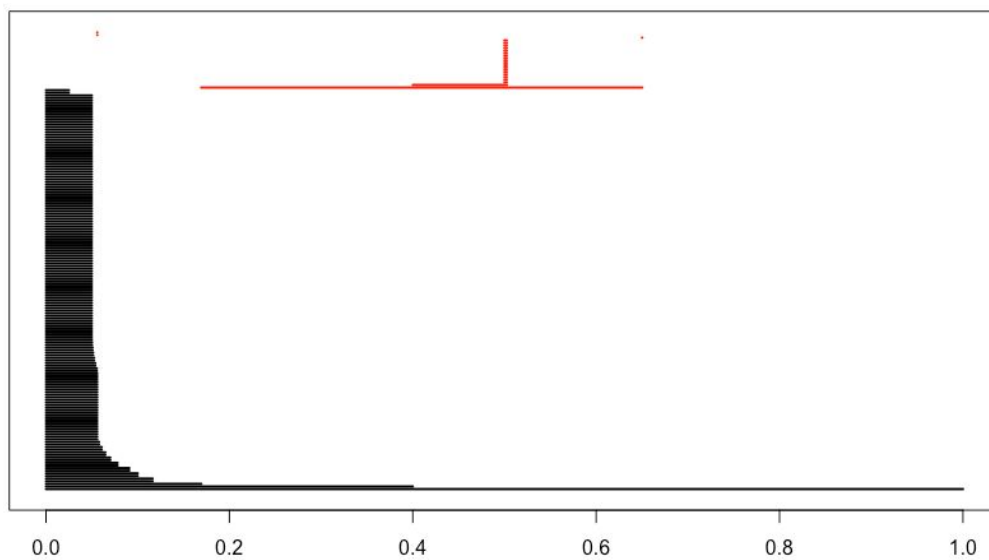
An R package named TDA (Fasy et al. 2014) has persistent homology capabilities, which we will demonstrate on another ND logo dataset.

```
n_x <- c(rep(-1, 31), 0.005*seq(-200, -50, 5), rep(-.25, 31))
n_y <- c(0.01*seq(-75, 75, 5), -0.01*seq(-75, 75, 5),
        0.01*seq(-75, 75, 5))
d_x <- c(rep(.25, 31),
        0.15+sqrt(0.75^2-(0.01*seq(-75, 75, 5))^2))
d_y <- c(0.01*seq(-75, 75, 5), 0.01*seq(-75, 75, 5))
nd <- data.frame(x=c(n_x, d_x), y=c(n_y, d_y))
plot(nd)
```



We'll create a barcode diagram to display the dataset's persistent homology in dimensions 0 and 1, for epsilon ranging from 0 to 1. The first homology components are colored black, while the second homology components are colored red.

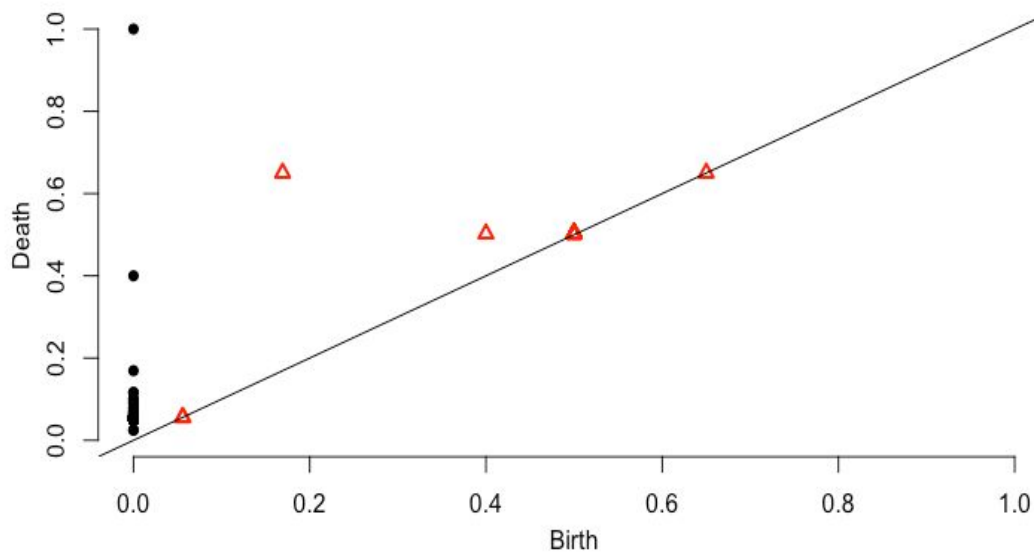
```
Diag <- ripsDiag(X = nd, maxdimension = 2, maxscale = 1,  
  library = "GUDHI", printProgress = FALSE)  
plot(Diag[["diagram"]], barcode = TRUE, main = "Barcode")
```



In first homology we see one component which persists the whole way, capturing the N and D together, and another component which persists about halfway, capturing the separation between the N and the D. In second homology, we see one component which persists halfway, which captures the hole in the D.

Birth-death diagrams are also used to display the same information as persistence barcodes:

```
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 the N and the D in the logo.

```
n <- data.frame(x = n_x, y = n_y)
d <- data.frame(x = d_x, y = d_y)
DiagN <- ripsDiag(X = n, maxdimension = 1, maxscale = 1)
DiagD <- ripsDiag(X = d, maxdimension = 1, maxscale = 1)
```



```
> print(bottleneck(Diag1 = DiagN[["diagram"]],  
                  Diag2 = DiagD[["diagram"]], dimension = 1))  
  
0.2404992  
  
> print(wasserstein(Diag1 = DiagN[["diagram"]],  
                  Diag2 = DiagD[["diagram"]], p = 2, dimension = 1))  
  
0.05783988
```

BIBLIOGRAPHY.

Carlsson, Gunnar. "Topology and data." *Bulletin of the American Mathematical Society* 46.2 (2009): 255-308.

Fasy, Brittany, Jisu Kim, Fabrizio Lecci, Clement Maria, and Vincent Rouvreau. "Introduction to the R Package TDA." *CRAN*. 2014.

"Machine Intelligence for Denials Reduction." *Ayasdi Resources*. 2015.

Parulekar, Sanjna, and Alexis Johnson. "Analyzing Oil & Gas Data with Ayasdi." *Ayasdi Resources*. 2014.

Parulekar, Sanjna, and Alexis Johnson. "Ayasdi Cure: Turning Data into Therapies" *Ayasdi Resources*. 2014.

Parulekar, Sanjna, and Alexis Johnson. "Campaign Planning with Social Media Intelligence." *Ayasdi Resources*. 2014.

Pearson, Paul, Daniel Muellner, and Gurjeet Singh. "Analyze High-Dimensional Data Using Discrete Morse Theory." *CRAN*. 2016.

Roche, Terry, Tim Grant, Patrick Rogers, and Mukund Ramachandran. "Predicting the Future: Forecasting Returns using Machine Intelligence." *Ayasdi Resources*. 2015.

Rogers, Patrick, and Johan Grahnen. "Recognizing the Shape of Fraud: Improve FWA discovery with Machine Intelligence." *Ayasdi Resources*. 2015.

Singh, Gurjeet, Facundo Mémoli, and Gunnar E. Carlsson. "Topological methods for the analysis of high dimensional data sets and 3d object recognition." *SPBG*. 2007.