Member-only story

Spectral Clustering

Foundation and Application

William Fleshman · Follow
Published in Towards Data Science

Introduction

In this post, we'll cover the ins and outs of spectral clustering for graphs and other data. Clustering is one of the main tasks in unsupervised machine learning. The goal is to assign unlabeled data to groups, where similar data points hopefully get assigned to the same group.

Spectral clustering is a technique with roots in graph theory, where the approach is used to identify communities of nodes in a graph based on the edges connecting them. The method is flexible and allows us to cluster non graph data as well.

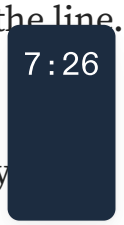
Spectral clustering uses information from the eigenvalues (spectrum) of special matrices built from the graph or the data set. We'll learn how to construct these matrices, interpret their spectrum, and use the eigenvectors to assign our data to clusters.

. . .

Eigenvectors and Eigenvalues

Critical to this discussion is the concept of eigenvalues and eigenvectors. For a matrix A, if there exists a vector x which isn't all 0's and a scalar λ such that $Ax = \lambda x$, then x is said to be an eigenvector of A with corresponding eigenvalue λ.

We can think of the matrix A as a function which maps vectors to new vectors. Most vectors will end up somewhere completely different when A is applied to them, but eigenvectors only change in magnitude. If you drew a line through the origin and the eigenvector, then after the mapping, the eigenvector would still land on the line. The amount which the vector is scaled along the line depends on λ.



We can easily find the eigenvalues and eigenvectors of a matrix using numpy in Python:

```python
import numpy as np

# a 2x2 matrix
A = np.array([[0,1],[-2,-3]])

# find eigenvalues and eigenvectors
vals, vecs = np.linalg.eig(A)

# print results
for i, value in enumerate(vals):
    print("Eigenvector:", vecs[:,i], ", Eigenvalue:", value)

# Eigenvector: [ 0.70710678 -0.70710678] , Eigenvalue: -1.0
# Eigenvector: [-0.4472136   0.89442719] , Eigenvalue: -2.0
```

find_eigen.py hosted with ❤ by GitHub view raw

Finding eigenvectors in Python.

Eigenvectors are an important part of linear algebra, because they help describe the dynamics of systems represented by matrices. There are numerous applications which utilize eigenvectors, and we'll use them directly here to perform spectral clustering.

. . .

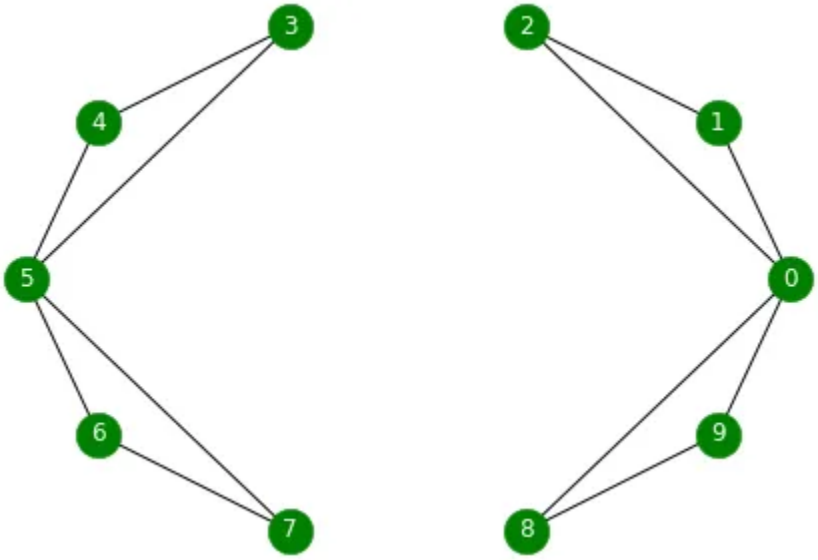
Graphs

Graphs are a natural way to represent many types of data. A graph is a set of nodes with a corresponding set of edges which connect the nodes. The edges may be directed or undirected and can even have weights associated with them.

A network of routers on the internet can easily be represented as a graph. The routers are the nodes, and the edges are the connections between pairs of routers.

Some routers might only allow traffic in one direction, so the edges could be directed to represent which direction traffic can flow. The weights on the edges could represent the bandwidth available along that edge. With this setup, we could then query the graph to find efficient paths for transmitting data from one router to another across the network.

Let's use the following undirected graph as a running example:



Graph with two disconnected components.

This graph has 10 nodes and 12 edges. It also has two connected components {0,1,2,8,9} and {3,4,5,6,7}. A connected component is a maximal subgraph of nodes which all have paths to the rest of the nodes in the subgraph.

Connected components seem important, if our task is to assign these nodes to communities or clusters. A simple idea would be to make each connected component its own cluster. This seems reasonable for our example graph, but it's possible that the entire graph might be connected, or that the connected components are very large. There could also be smaller structures within a connected component which are good candidates for communities. We'll shortly see the importance of this connected component idea for spectral clustering.

Adjacency Matrix

We can represent our example graph as an adjacency matrix, where the row and column indices represent the nodes, and the entries represent the absence or presence of an edge between the nodes. The adjacency matrix for our example graph looks like this:

In the matrix, we see that row 0, column 1 has a value of 1. That means that there is an edge connecting node 0 with node 1. If the edges were weighted, the weights of the edges would go in this matrix instead of just 1s and 0s. Since our graph is undirected, the entries for row i, col j will be equal to the entry at row j, col i. The last thing to notice is that the diagonal of this matrix is all 0, since none of our nodes have edges to themselves.

Degree Matrix

The degree of a node is how many edges connect to it. In a directed graph we could talk about in-degree and out-degree, but in this example we just have degree since the edges go both ways. Looking at our graph, we see that node 0 has degree 4, since it has 4 edges. We could also get the degree by taking the sum of the node's row in the adjacency matrix.

The degree matrix is a diagonal matrix where the value at entry (i, i) is the degree of node i. Let's find the degree matrix for our example:

First, we took the sum across axis 1 (the rows) of our adjacency matrix, and then we put those values into a diagonal matrix. From the degree matrix, we can easily see that nodes 0 and 5 have 4 edges, while the rest of the nodes have only 2.

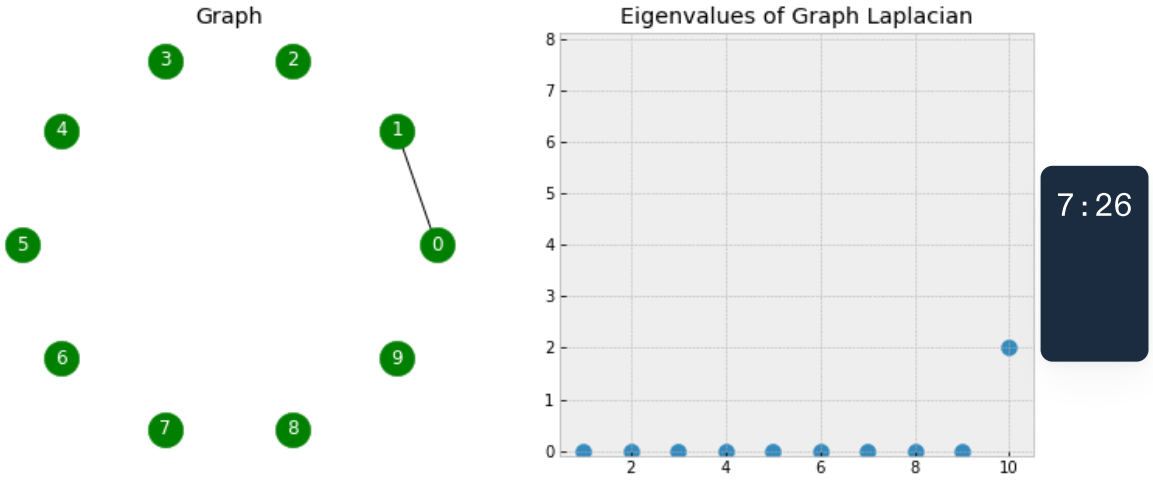
Graph Laplacian

Now we're going to calculate the Graph Laplacian. The Laplacian is just another matrix representation of a graph. It has several beautiful properties, which we will take advantage of for spectral clustering. To calculate the normal Laplacian (there are several variants), we just subtract the adjacency matrix from our degree matrix:

The Laplacian's diagonal is the degree of our nodes, and the off diagonal is the negative edge weights. This is the representation we are after for performing spectral clustering.

Eigenvalues of Graph Laplacian

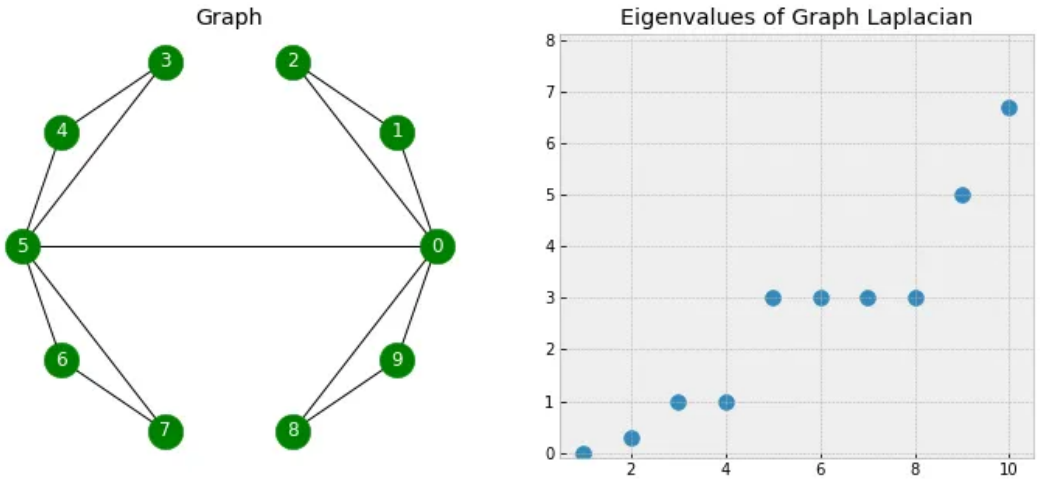
As mentioned, the Laplacian has some beautiful properties. To get a sense for this, let's examine the eigenvalues associated with the Laplacian as I add edges to our graph:



Eigenvalues of Graph Laplacian.

We see that when the graph is completely disconnected, all ten of our eigenvalues are 0. As we add edges, some of our eigenvalues increase. In fact, the number of 0 eigenvalues corresponds to the number of connected components in our graph!

Look closely as that final edge is added, connecting the two components into one. When this happens, all of the eigenvalues but one have been lifted:



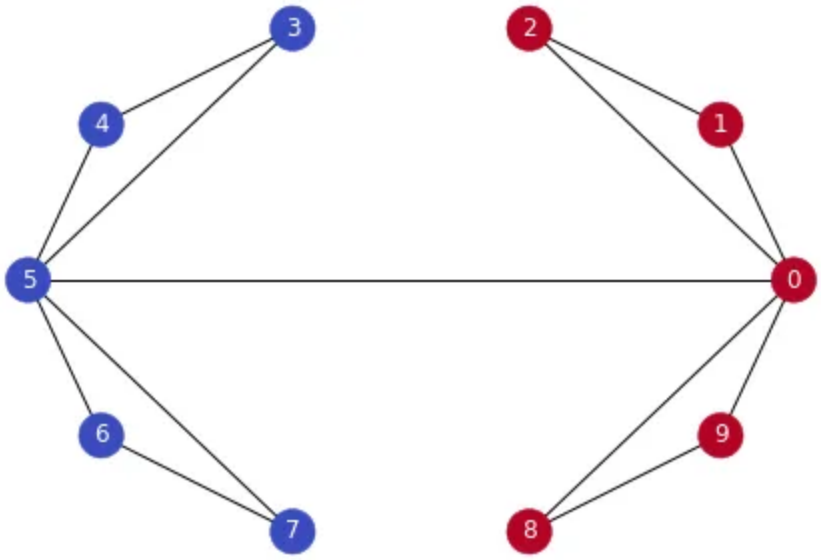
Number of 0-eigenvalues is number of connected components.

The first eigenvalue is 0 because we only have one connected component (the whole graph is connected). The corresponding eigenvector will always have constant values (in this example all the values are close to 0.32).

The first nonzero eigenvalue is called the spectral gap. The spectral gap gives us some notion of the density of the graph. If this graph was densely connected (all

pairs of the 10 nodes had an edge), then the spectral gap would be 10.

The second eigenvalue is called the Fiedler value, and the corresponding vector is the Fiedler vector. The Fiedler value approximates the minimum graph cut needed to separate the graph into two connected components. Recall, that if our gra 7:26 already two connected components, then the Fiedler value would be 0. Each n the Fiedler vector gives us information about which side of the cut that node belongs. Let's color the nodes based on whether their entry in the Fielder vector is positive or not:



Nodes colored based on whether their entry in the Fiedler Vector is >0.

This simple trick has segmented our graph into two clusters! Why does this work? Remember that zero eigenvalues represent connected components. Eigenvalues near zero are telling us that there is almost a separation of two components. Here we have a single edge, that if it didn't exist, we'd have two separate components. So the second eigenvalue is small.

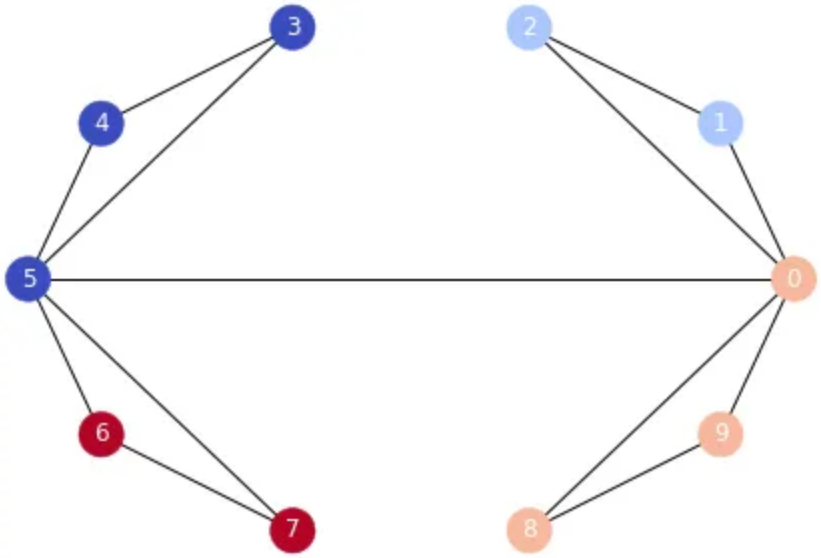
To summarize what we know so far: the first eigenvalue is 0 because we have one connected component. The second eigenvalue is near 0 because we're one edge away from having two connected components. We also saw that the vector associated with that value tells us how to separate the nodes into those approximately connected components.

You may have noticed that the next two eigenvalues are also pretty small. That tells us that we are "close" to having four separate connected components. In general, we

often look for the first large gap between eigenvalues in order to find the number of clusters expressed in our data. See the gap between eigenvalues four and five?

Having four eigenvalues before the gap indicates that there is likely four clusters. The vectors associated with the first three positive eigenvalues should give us information about which three cuts need to be made in the graph to assign ea node to one of the four approximated components. Let's build a matrix from three vectors and perform K-Means clustering to determine the assignments:

7:26



Spectral Clustering for 4 clusters.

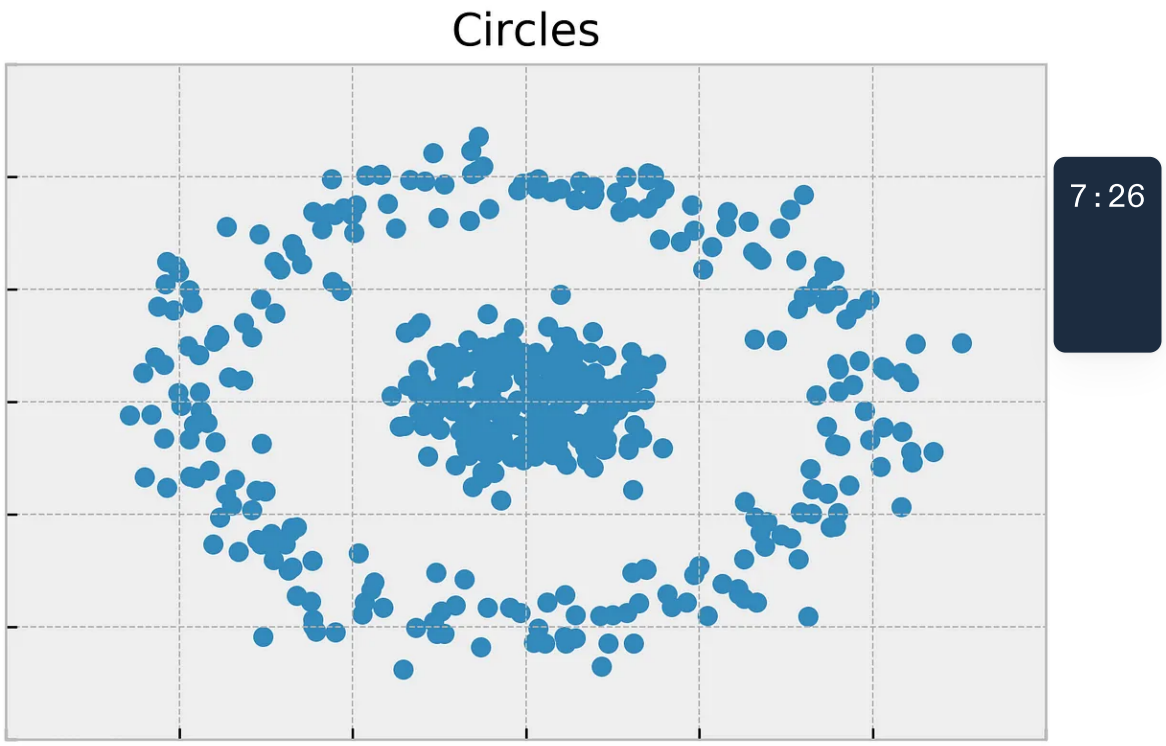
The graph has been segmented into the four quadrants, with nodes 0 and 5 arbitrarily assigned to one of their connected quadrants. That is really cool, and that is spectral clustering!

To summarize, we first took our graph and built an adjacency matrix. We then created the Graph Laplacian by subtracting the adjacency matrix from the degree matrix. The eigenvalues of the Laplacian indicated that there were four clusters. The vectors associated with those eigenvalues contain information on how to segment the nodes. Finally, we performed K-Means on those vectors in order to get the labels for the nodes. Next, we'll see how to do this for arbitrary data.

. . .

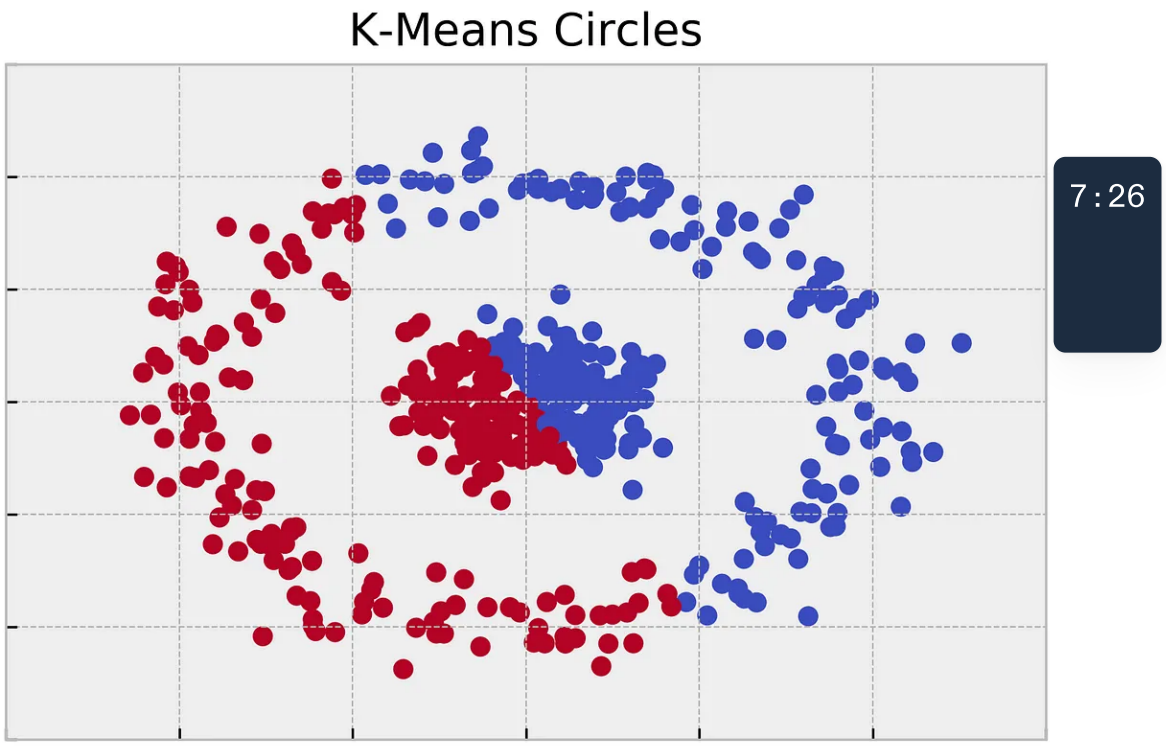
Spectral Clustering Arbitrary Data

Look at the data below. The points are drawn from two concentric circles with some noise added. We'd like an algorithm to be able to cluster these points into the two circles that generated them.



Circles Data.

This data isn't in the form of a graph. So first, let's just try a cookie cutter algorithm like K-Means. K-Means will find two centroids and label the points based on which centroid they are closest too. Here are the results:



K-Means Clustering of the Circles Data.

Obviously, K-Means wasn't going to work. It operates on euclidean distance, and it assumes that the clusters are roughly spherical. This data (and often real world data) breaks these assumptions. Let's try to tackle this with spectral clustering.

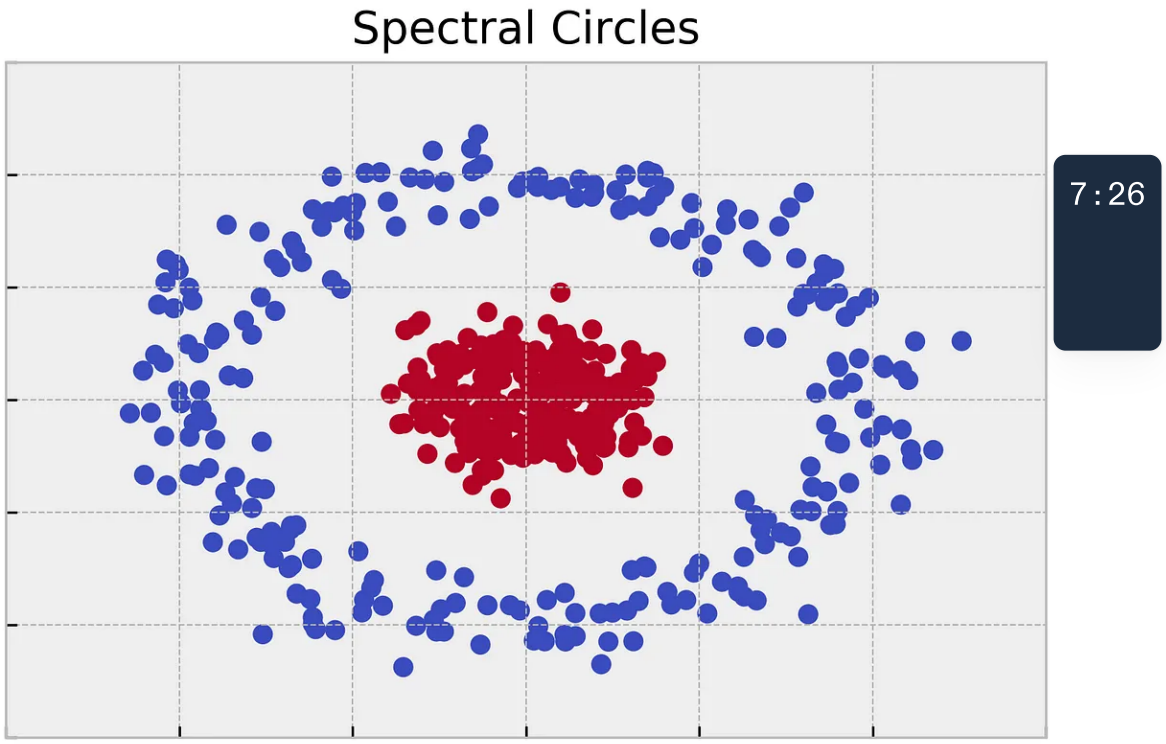
Nearest Neighbors Graph

There are a couple of ways to treat our data as a graph. The easiest way is to construct a k-nearest neighbors graph. A k-nearest neighbors graph treats every data point as a node in a graph. An edge is then drawn from each node to its k nearest neighbors in the original space. Generally, the algorithm isn't too sensitive of the choice of k. Smaller numbers like 5 or 10 usually work pretty well.

Look at the picture of the data again, and imagine that each point is connected to its 5 closest neighbors. Any point in the outer ring should be able to follow a path along the ring, but there won't be any paths into the inner circle. It's pretty easy to see that this graph will have two connected components: the outer ring and the inner circle.

Since we're only separating this data into two components, we should be able to use our Fiedler vector trick from before. Here's the code I used to perform spectral clustering on this data:

And here are the results:



Spectral Clustering of the Circles Data.

Other Approaches

The nearest neighbor graph is a nice approach, but it relies on the fact that “close” points should belong in the same cluster. Depending on your data, that might not be true. A more general approach is to construct an affinity matrix. An affinity matrix is just like an adjacency matrix, except the value for a pair of points expresses how similar those points are to each other. If pairs of points are very dissimilar then the affinity should be 0. If the points are identical, then the affinity might be 1. In this way, the affinity acts like the weights for the edges on our graph.

How to decide on what it means for two data points to be similar is one of the most important questions in machine learning. Often domain knowledge is the best way to construct a similarity measure. If you have access to domain experts, ask them this question.

There are also entire fields dedicated to learning how to construct similarity metrics directly from data. For example, if you have some labeled data, you can train a classifier to predict whether two inputs are similar or not based on if they have the same label. This classifier can then be used to assign affinity to pairs of unlabeled points.

. . .

Conclusion

We've covered the theory and application of spectral clustering for both grap
arbitrary data. Spectral clustering is a flexible approach for finding clusters v
your data doesn't meet the requirements of other common algorithms.

7:26

First, we formed a graph between our data points. The edges of the graph capture the similarities between the points. The eigenvalues of the Graph Laplacian can then be used to find the best number of clusters, and the eigenvectors can be used to find the actual cluster labels.

I hope you enjoyed this post and find spectral clustering useful in your work or exploration.

See you next time!

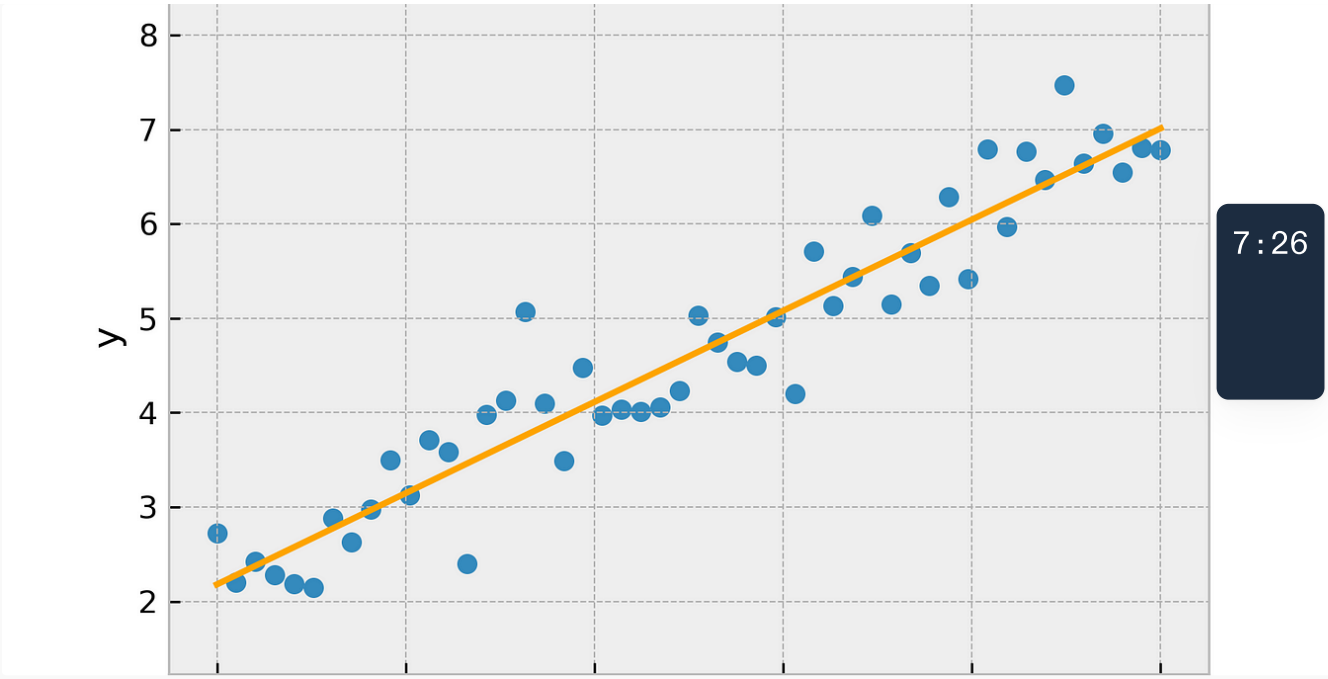
Follow

Written by William Fleshman

878 Followers · Writer for Towards Data Science

Machine learning writ large!

More from William Fleshman and Towards Data Science



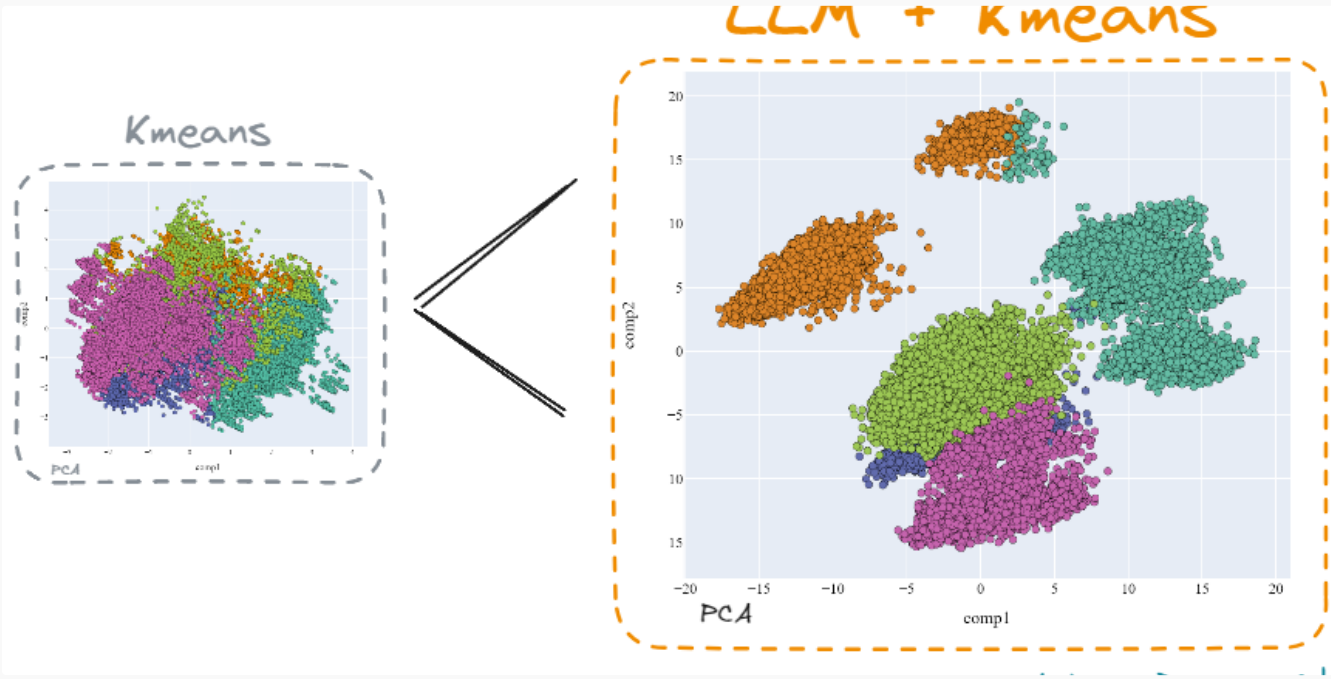
William Fleshman in Towards Data Science

Linear Regression

A unification of Maximum Likelihood Estimation and minimizing the sum of squares.

· 7 min read · Feb 10, 2019

355 4



Damian Gil in Towards Data Science

Mastering Customer Segmentation with LLM

Unlock advanced customer segmentation techniques using LLMs, and improve your clustering models with advanced techniques

23 min read · Sep 26

2.5K 22

7 : 26

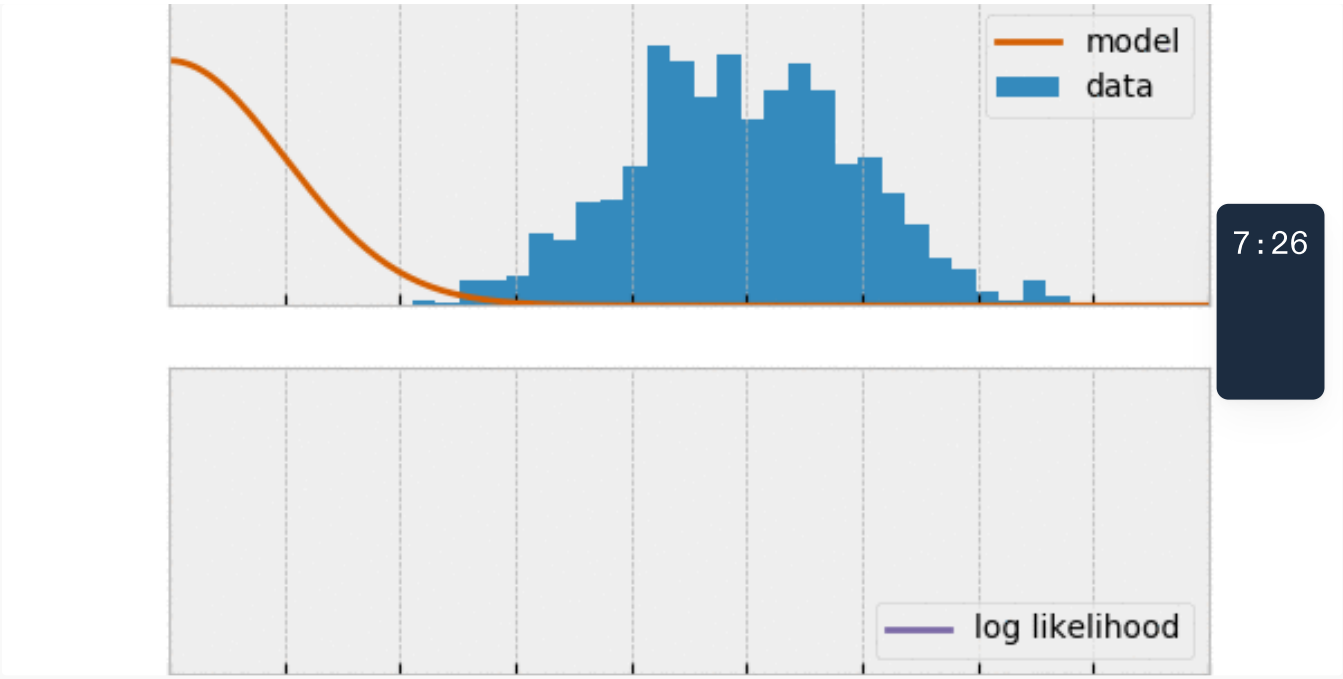
Giuseppe Scalamogna in Towards Data Science

New ChatGPT Prompt Engineering Technique: Program Simulation

A potentially novel technique for turning a ChatGPT prompt into a mini-app.

9 min read · Sep 3

1.7K 17

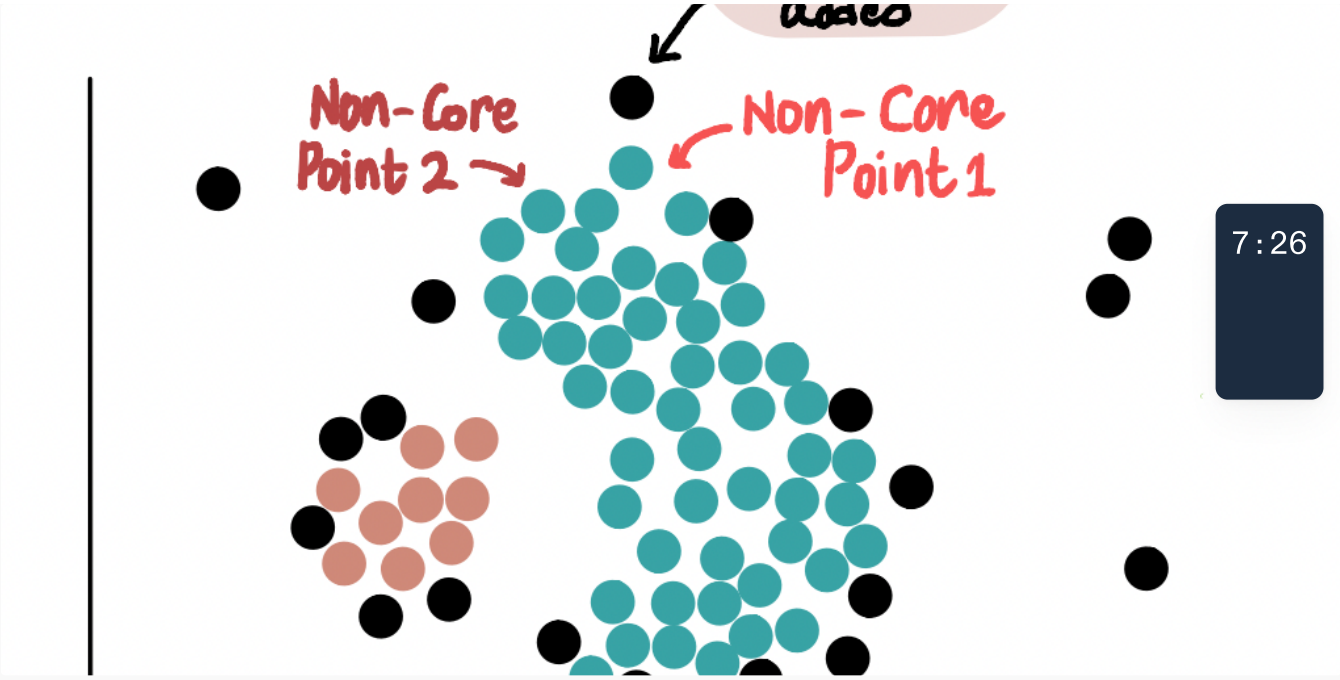


William Fleshman in Towards Data Science

Maximum Likelihood Estimation

Fundamentals of Machine Learning (Part 2)

· 9 min read · Feb 3, 2019



Shreya Rao in Towards Data Science

DBSCAN Clustering: Break It Down For Me

An accessible introduction to a powerful algorithm

· 6 min read · Nov 29, 2022

405 4

YashwanthReddyGoduguchintha

Hierarchical clustering

Hierarchical clustering:-

3 min read · Apr 8

1

7 : 26

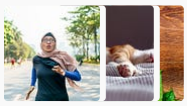
Lists

Staff Picks

467 stories · 328 saves

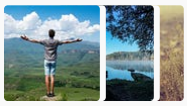
Stories to Help You Level-Up at Work

19 stories · 237 saves



Self-Improvement 101

20 stories · 659 saves



Productivity 101

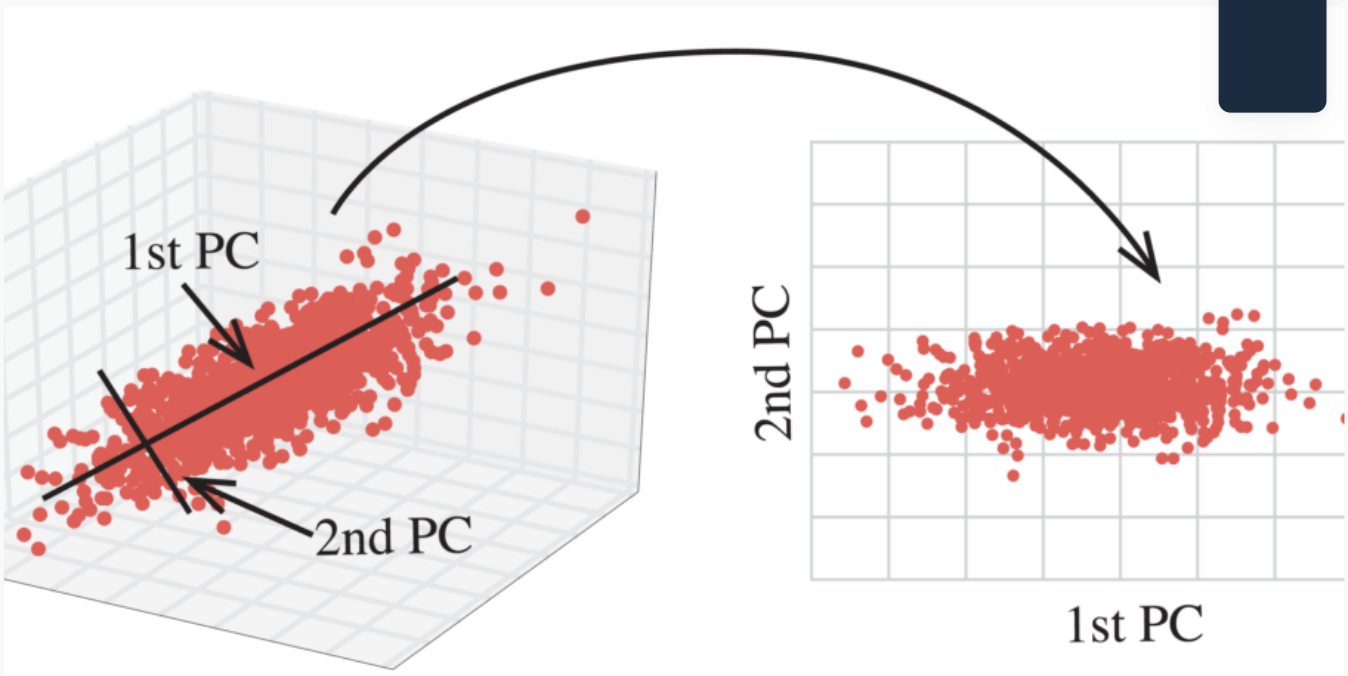
20 stories · 611 saves

Sruthy Nath

Unveiling Data Patterns with Gaussian Mixture Models (GMM) for Density Estimation

In the vast sea of data, lies the hidden treasure of insights and patterns waiting to be discovered. One way to unlock this treasure trove...

3 min read · Aug 16



Tushar Babbar in AlliedOffsets

Unveiling the Power of PCA: Turbocharge Your Data Science with Dimensionality Reduction!

This post will help you understand the importance of dimensionality reduction using the most popular technique, PCA.

6 min read · Jun 14

7

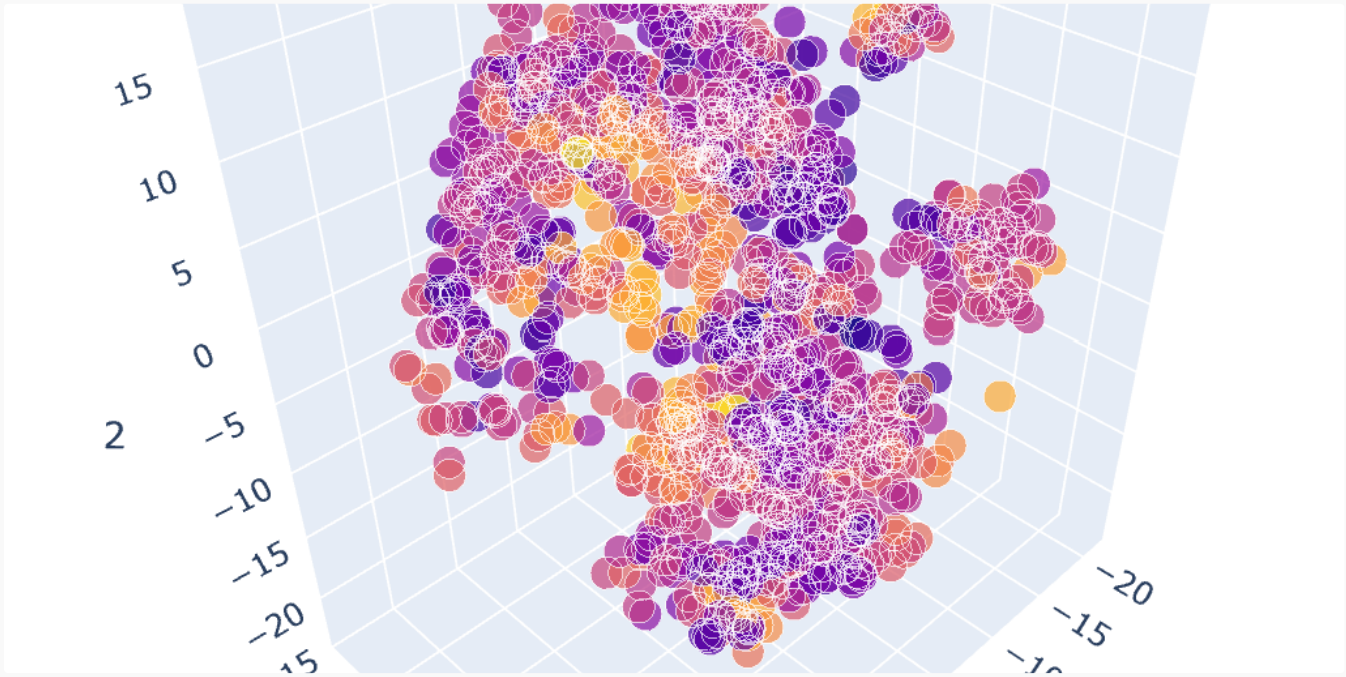
Yannis Poulakis

Is Silhouette the Right Clustering Evaluation Metric for You?

While popular, simple experiments prove that Silhouette index may not be the index to evaluate every clustering solution.

4 min read · Jul 4

17



Marlon Hamm

Music Clustering

Dimensionality reduction and clustering of your spotify Liked Song history

7 min read · Apr 19

3

7 : 26

See more recommendations