DBSCAN Algorithm from Scratch in Python

Ryan Davidson · Follow
5 min read · Oct 21, 2020

22

DBSCAN (**D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise) is a popular **unsupervised** learning method utilized in model building and machine learning algorithms originally proposed by Ester et al in 1996. Before we go any further, we need to define what is "unsupervised" learning method. **Unsupervised** learning methods are when there is no clear objective or outcome we are seeking to find. Instead, we are clustering the data together based on the similarity of observations.

Definitions

- ε (epsilon): the radius of a neighborhood centered on a given point
- Core Point: a given point is considered a Core Point if there are at least *minPts* points within its ε neighborhood, including itself
- Border Point: a given point is considered a Borer Point if there fewer than *minPts* points within its ε neighborhood, including itself

- Noise: any point that is not a Core Point or Border Point
- Directly Density Reachable: a given point is Directly Density Reachable (ε Reachable) from another point if the second point is a core point, and the first point lies within the ε neighborhood of the second point
- Density Reachable: a given point is Density Reachable from another point if there is a chain of points, Directly Density Reachable from each other, that connects them
- Density Connected: A given point is Density Connected from another point if there is a third point from which both are Density Reachable — These points are said to be Connected Components

DBSCAN in a Nutshell

Given a set of points *P*, the radius of a neighborhood ε, and a minimum number of points *minPts:*

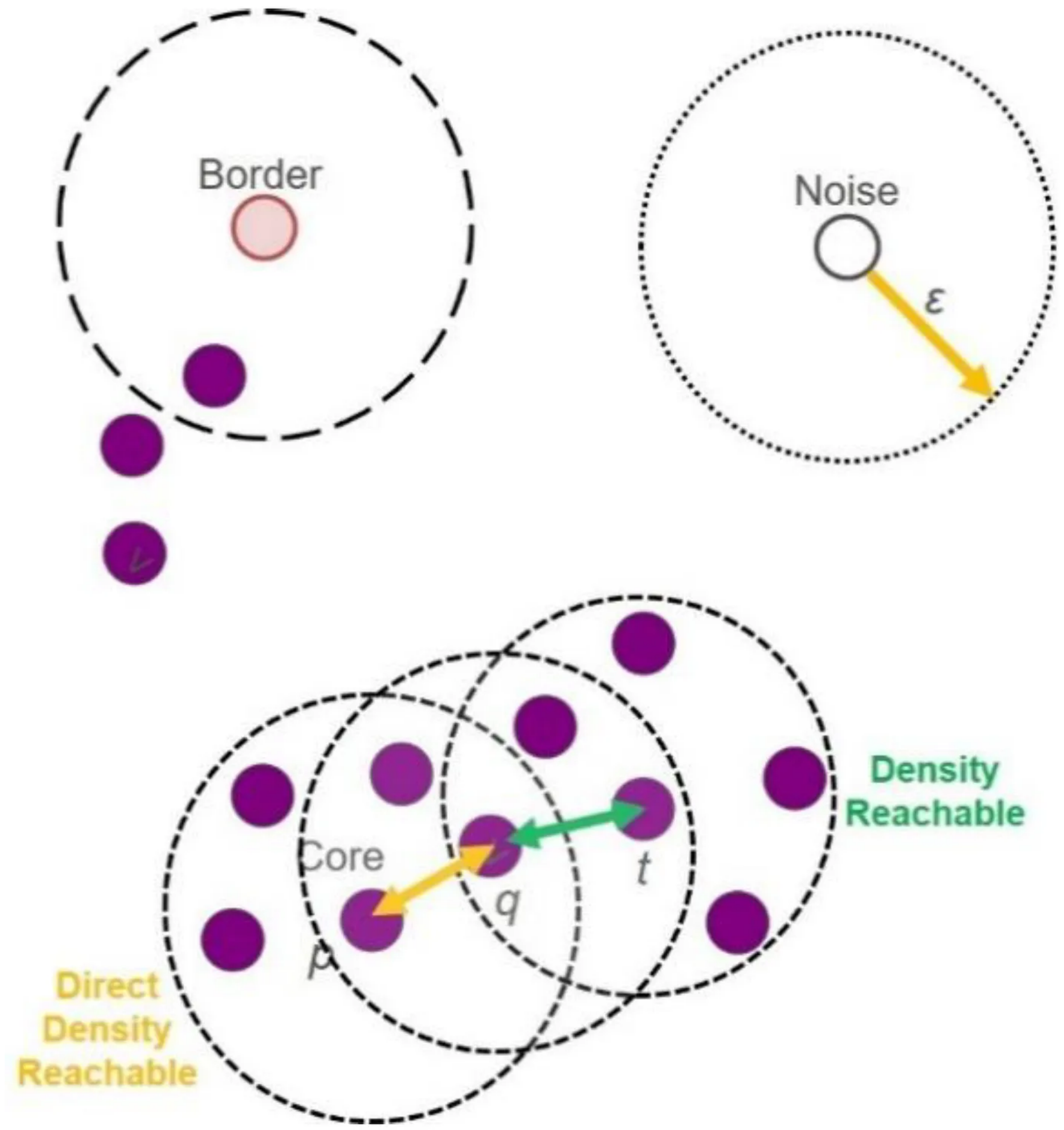
1. Find all points within the ε neighborhood of each point;
2. Identify Core Points with at least *minPts* neighbors;
3. Find all Connected Components of each core point — This Density Connected grouping of points is a cluster
4. Each Border Point is assigned to a cluster if that cluster is Density Reachable, otherwise, Border Point is considered Noise

Any given point may initially be considered noise and later revised to belong to a cluster, but once assigned to a cluster a point will never be assigned.

Okay if that was too much let's explain it in some simpler terms:

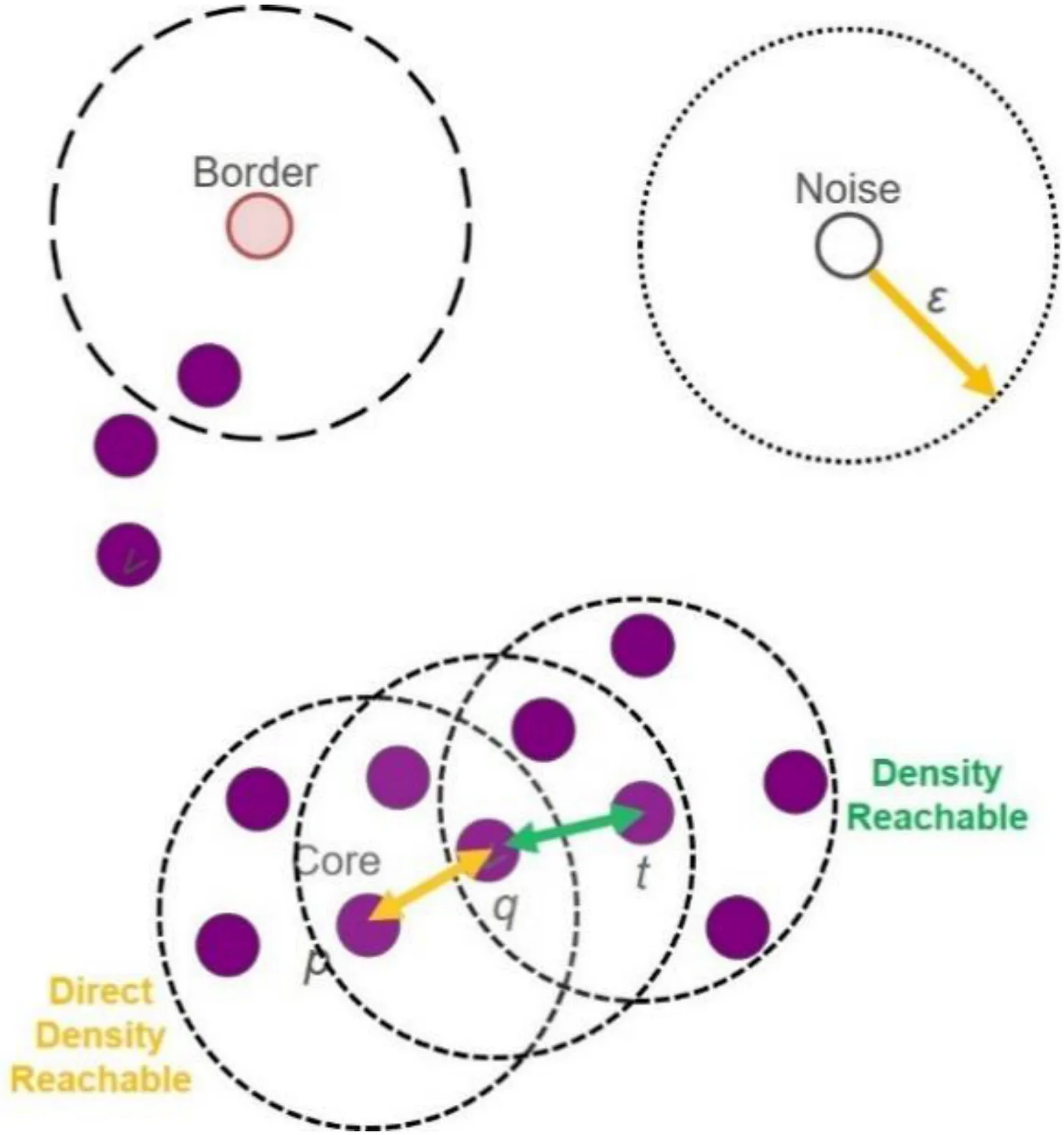
Given that **DBSCAN** is a **density-based clustering algorithm**, it does a great job of seeking areas in the data that have a high density of observations, versus areas of the data that are not very dense with observations. DBSCAN can sort data into clusters of varying shapes as well, another strong advantage. DBSCAN works as such:

- Divides the dataset into *n* dimensions
- For each point in the dataset, DBSCAN forms an *n-dimensional* shape around that data point and then counts how many data points fall within that shape.
- DBSCAN counts this shape as a *cluster.* DBSCAN iteratively expands the cluster, by going through each individual point within the cluster, and counting the number of other data points nearby. Take the graphic below for an example:



Going through the process step-by-step, DBSCAN will start by dividing the data into *n* dimensions. After DBSCAN has done so, it will start at a random point (in this case let's assume it was one of the purple points), and it will count how many other points are nearby. DBSCAN will continue this process until no other data points are nearby, and then it will look to form a second cluster.

As you may have noticed from the graphic, there are a couple of parameters and specifications that we need to give DBSCAN before it does its work.



Referring back to the graphic, the *epsilon* is the radius given to test the distance between data points. If a point falls within the *epsilon* distance of another point, those two points will be in the same cluster.

Furthermore, the *minimum number of points needed* is set to 4 in this scenario. When going through each data point, as long as DBSCAN finds 4 points within *epsilon* distance of each other, a cluster is formed.

Naftali Harris has created a great web-based visualization of running DBSCAN on a 2-dimensional dataset where you can set *epsilon* to higher and lower values. Try clicking on the “Smiley” dataset and hitting the GO button.

DBSCAN vs K-Means Clustering

DBSCAN is a popular clustering algorithm that is fundamentally very different from k-means.

- In k-means clustering, each cluster is represented by a centroid, and points are assigned to whichever centroid they are closest to. In DBSCAN, there are no centroids, and clusters are formed by linking nearby points to one another.
- k-means requires specifying the number of clusters, ‘k’. DBSCAN does not but does require specifying two parameters which influence the decision of whether two nearby points should be linked into the same cluster. These two parameters are a distance threshold, ε (epsilon), and “MinPts” (minimum number of points), to be explained.
- k-means runs over many iterations to converge on a good set of clusters, and cluster assignments can change on each iteration. DBSCAN makes only a single pass through the data, and once a point has been assigned to a particular cluster, it never changes.

My Approach to the DBSCAN Algorithm

I like the language of trees for describing cluster growth in DBSCAN. It starts with an arbitrary seed point which has at least MinPts points nearby within a distance or "radius" of ε. We do a breadth-first search along each of these nearby points. For a given nearby point, we check how many points *it* has within its radius. If it has fewer than MinPts neighbors, this point becomes a *leaf*–we don't continue to grow the cluster from it. If it *does* have at least MinPts, however, then it's a *branch,* and we add all of its neighbors to the FIFO ("First In, First Out") queue of our breadth-first search.

Once the breadth-first search is complete, we are done with that cluster and

There is one other novel aspect of DBSCAN which affects the algorithm. If a point has fewer than MinPts neighbors, ***AND it's not a leaf node of another cluster***, then it's labeled as a "Noise" point that doesn't belong to any cluster.

Noise points are identified as part of the process of selecting a new seed if a particular seed point does not have enough neighbors, then it is labeled as a Noise point. This label is often temporary, however–these Noise points are often picked up by some cluster as a leaf node.

To understand this, I feel like it is best to just jump headfirst into the code:

```
import numpy

def dbscan(D, eps, MinPts):
    '''
    Cluster the dataset `D` using the DBSCAN algorithm.

    dbscan takes a dataset `D` (a list of vectors), a threshold distance
    `eps`, and a required number of points `MinPts`.

    It will return a list of cluster labels. The label -1 means noise, and then
    the clusters are numbered starting from 1.
    '''

    # This list will hold the final cluster assignment for each point in D.
    # There are two reserved values:
    #    -1 - Indicates a noise point
    #     0 - Means the point hasn't been considered yet.
    # Initially all labels are 0.
    labels = [0]*len(D)

    # C is the ID of the current cluster.
    C = 0

    # This outer loop is just responsible for picking new seed points--a point
    # from which to grow a new cluster.
    # Once a valid seed point is found, a new cluster is created, and the
    # cluster growth is all handled by the 'expandCluster' routine.

    # For each point P in the Dataset D...
    # ('P' is the index of the datapoint, rather than the datapoint itself.)
    for P in range(0, len(D)):

        # Only points that have not already been claimed can be picked as new
        # seed points.
        # If the point's label is not 0, continue to the next point.
        if not (labels[P] == 0):
           continue

        # Find all of P's neighboring points.
        NeighborPts = region_query(D, P, eps)

        # If the number is below MinPts, this point is noise.
        # This is the only condition under which a point is labeled
        # NOISE--when it's not a valid seed point. A NOISE point may later
        # be picked up by another cluster as a boundary point (this is the only
        # be picked up by another cluster as a boundary point (this is the only
        # condition under which a cluster label can change--from NOISE to
        # something else).
        if len(NeighborPts) < MinPts:
            labels[P] = -1
        # Otherwise, if there are at least MinPts nearby, use this point as the
        # seed for a new cluster.
        else:
           C += 1
           grow_cluster(D, labels, P, NeighborPts, C, eps, MinPts)

    # All data has been clustered!
    return labels


def grow_cluster(D, labels, P, NeighborPts, C, eps, MinPts):
    '''
    Grow a new cluster with label `C` from the seed point `P`.

    This function searches through the dataset to find all points that belong
    to this new cluster. When this function returns, cluster `C` is complete.

    Parameters:
      `D`      - The dataset (a list of vectors)
      `labels` - List storing the cluster labels for all dataset points
      `P`      - Index of the seed point for this new cluster
      `NeighborPts` - All of the neighbors of `P`
      `C`      - The label for this new cluster.
      `eps`    - Threshold distance
      `MinPts` - Minimum required number of neighbors
    '''

    # Assign the cluster label to the seed point.
    labels[P] = C

    # Look at each neighbor of P (neighbors are referred to as Pn).
    # NeighborPts will be used as a FIFO queue of points to search--that is, it
    # will grow as we discover new branch points for the cluster. The FIFO
    # behavior is accomplished by using a while-loop rather than a for-loop.
    # In NeighborPts, the points are represented by their index in the original
    # dataset.
    i = 0
    while i < len(NeighborPts):

        # Get the next point from the queue.
        Pn = NeighborPts[i]

        # If Pn was labelled NOISE during the seed search, then we
        # know it's not a branch point (it doesn't have enough neighbors), so
        # make it a leaf point of cluster C and move on.
        if labels[Pn] == -1:
           labels[Pn] = C

        # Otherwise, if Pn isn't already claimed, claim it as part of C.
        elif labels[Pn] == 0:
            # Add Pn to cluster C (Assign cluster label C).
            labels[Pn] = C

            # Find all the neighbors of Pn
            PnNeighborPts = region_query(D, Pn, eps)

            # If Pn has at least MinPts neighbors, it's a branch point!
            # Add all of its neighbors to the FIFO queue to be searched.
            if len(PnNeighborPts) >= MinPts:
                NeighborPts = NeighborPts + PnNeighborPts
            # If Pn *doesn't* have enough neighbors, then it's a leaf point.
            # Don't queue up it's neighbors as expansion points.
            #else:
                # Do nothing
                #NeighborPts = NeighborPts

        # Advance to the next point in the FIFO queue.
        i += 1

    # We've finished growing cluster C!


def region_query(D, P, eps):
    '''
    Find all points in dataset `D` within distance `eps` of point `P`.

    This function calculates the distance between a point P and every other
    point in the dataset, and then returns only those points which are within a
    threshold distance `eps`.
    '''
    neighbors = []

    # For each point in the dataset...
    for Pn in range(0, len(D)):

        # If the distance is below the threshold, add it to the neighbors list.
        if numpy.linalg.norm(D[P] - D[Pn]) < eps:
           neighbors.append(Pn)

    return neighbors
```

1 hosted with ❤ by GitHub view raw

Above is a working implementation within Python. Please note that this emphasis in the implementation of the algorithm… the distance calculations, for example, could be optimized significantly.

You can also find this code along with a validation python file on GitHub here.

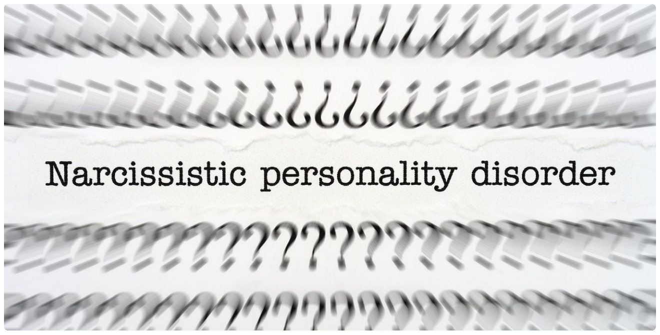
Written by Ryan Davidson

Follow

9 Followers

Data Science student, geologist, and avid disc golfer

More from Ryan Davidson



Ryan Davidson

Millennial's: More Narcissistic than Other Generations?

Brief History Lesson in Greek Mythology:

6 min read · Feb 26, 2020

11

Ryan Davidson

The End is in Sight or is it Just the Beginning?

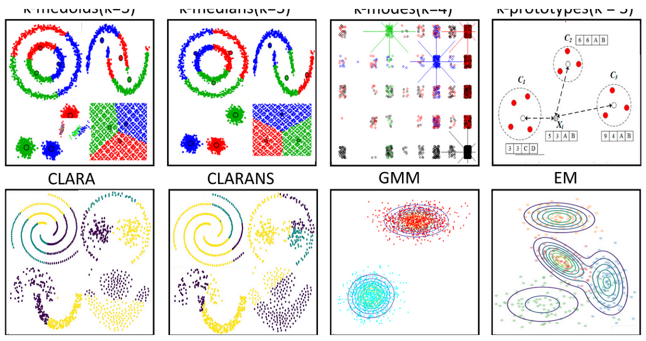
From no experience in being able to write one line of Python code to work on a cross-...

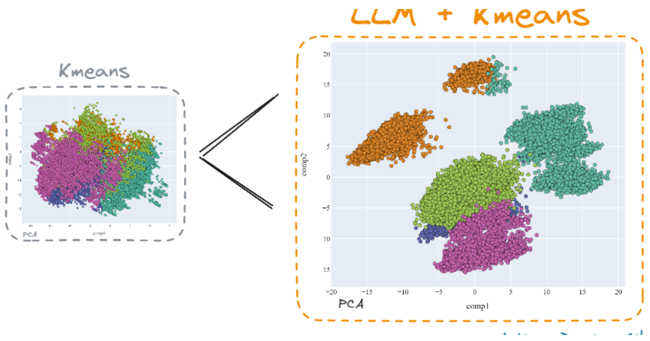
5 min read · Mar 1, 2021

5

See all from Ryan Davidson

Recommended from Medium





Hazal Gültekin

Clustering Algorithms

Clustering is an unsupervised machine learning task. Using a clustering algorithm...

4 min read · Aug 22

50

Damian Gil in Towards Data Science

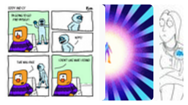
Mastering Customer Segmentation with LLM

Unlock advanced customer segmentation techniques using LLMs, and improve your...

23 min read · Sep 26

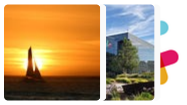
2.8K 24

Lists



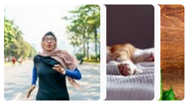
Staff Picks

468 stories · 337 saves



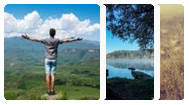
Stories to Help You Level-Up at Work

19 stories · 239 saves



Self-Improvement 101

20 stories · 685 saves



Productivity 101

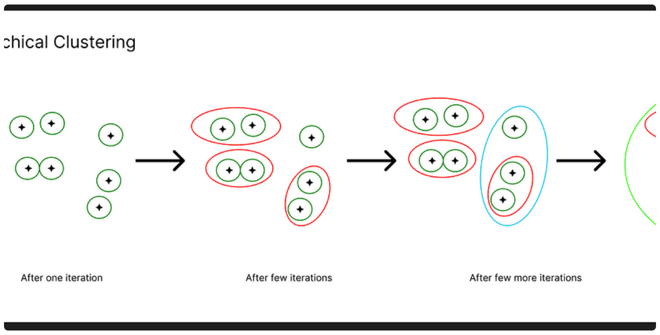
20 stories · 623 saves

Shruti Dhumne

Mean-Shift Clustering: A Powerful Technique for Data Analysis with...

Introduction

3 min read · Jun 1



Chandra Prakash Bathula in MLearning.ai

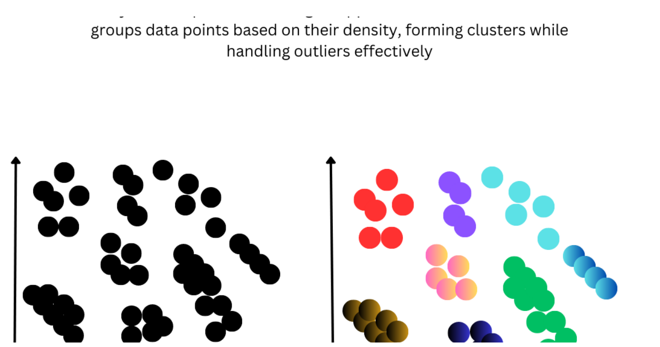
Everything to know about Hierarchical Clustering;...

Hierarchical Clustering.

16 min read · Jun 25

1

56


groups data points based on their density, forming clusters while
handling outliers effectively

YashwanthReddyGoduguchintha

Density-Based Spatial Clustering of Applications with Noise (DBSCAN…

The DBSCAN algorithm works by defining two parameters, epsilon (ε) and minimum points…

4 min read · Apr 11

Jayaramganesh

DBSCAN Clustering

DBSCAN

3 min read · Jul 20

See more recommendations