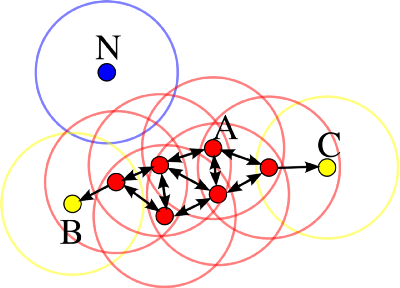
1. Minimum samples (“MinPts”): the fewest number of points required to form a cluster
2. ε (epsilon or “eps”): the maximum distance two points can be from one another while still belonging to the same cluster

DBSCAN stands for **Density-Based Spatial Clustering of Applications with Noise**. It is a density-based clustering algorithm. In other words, it clusters together data areas that are dense in the feature space.

The principle is quite simple. The model takes two parameters: epsilon and MinPoints. A point that has MinPoints around him within a radius of epsilon will be clustered with them. They are called core points. A point that is in the range of a core point but hasn't MinPoints points around him within epsilon range is called a border point. The border points are located at the edge of the clusters. Finally, a point that is not in the range of a core point is clustered alone and it is called a noisy point.



In the figure example shown above, red points represent core points, yellow ones represent border points and the blue one represents a noise point. By the end of the learning process, every point of the dataset will be either a core point, a border point, or a noise point.

There are some fairly well documented pros and cons of using DBSCAN.

Pros

* There is no need to specify the number of clusters, as opposed to k-means
* There is no prior on the shape of the clusters, they can be of any form
* Thanks to the noise data points idea, DBSCAN is robust to outliers

Cons

* *epsilon*and *MinPoints*are domain-dependent and can be hard to tune
* The distance metric used for finding neighbors can be hard to define – Used harversine because determines the [great-circle distance](https://en.wikipedia.org/wiki/Great-circle_distance) between two points on a [sphere](https://en.wikipedia.org/wiki/Sphere) given their [longitudes](https://en.wikipedia.org/wiki/Longitude) and [latitudes](https://en.wikipedia.org/wiki/Latitude). And as such is considered the preferred option for geographic points.
* Inappropriate for clusters with various densities, since *epsilon*is fixed (see [OPTICS](https://en.wikipedia.org/wiki/OPTICS_algorithm), an improved DBSCAN version fixing this issue)

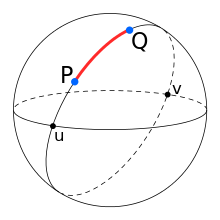
# Great circle:

# **Great-circle distance**

From Wikipedia, the free encyclopedia

[Jump to navigation](https://en.wikipedia.org/wiki/Great-circle_distance#mw-head)[Jump to search](https://en.wikipedia.org/wiki/Great-circle_distance#searchInput)

*This article is about shortest-distance on a sphere. For the shortest distance on an ellipsoid, see*[*geodesics on an ellipsoid*](https://en.wikipedia.org/wiki/Geodesics_on_an_ellipsoid)*.*

[](https://en.wikipedia.org/wiki/File:Illustration_of_great-circle_distance.svg)

A diagram illustrating great-circle distance (drawn in red) between two points on a sphere, P and Q. Two [nodal](https://en.wikipedia.org/wiki/Node)[[*disambiguation needed*](https://en.wikipedia.org/wiki/Wikipedia:WikiProject_Disambiguation/Fixing_links)] points, u and v, which are [antipodal](https://en.wikipedia.org/wiki/Antipodal_point), are also shown.

The **great-circle distance**, **orthodromic distance**, or **spherical distance** is the [distance](https://en.wikipedia.org/wiki/Distance) along a [great circle](https://en.wikipedia.org/wiki/Great_circle).

It is the shortest [distance](https://en.wikipedia.org/wiki/Distance) between two [points](https://en.wikipedia.org/wiki/Point_(geometry)) on the surface of a [sphere](https://en.wikipedia.org/wiki/Sphere), measured along the surface of the sphere (as opposed to a straight line through the sphere's interior). The distance between two points in [Euclidean space](https://en.wikipedia.org/wiki/Euclidean_space) is the length of a straight line between them, but on the sphere there are no straight lines. In [spaces with curvature](https://en.wikipedia.org/wiki/Manifold), straight lines are replaced by [geodesics](https://en.wikipedia.org/wiki/Geodesic). Geodesics on the sphere are circles on the sphere whose centers coincide with the center of the sphere, and are called 'great circles'.

The determination of the great-circle distance is part of the more general problem of [great-circle navigation](https://en.wikipedia.org/wiki/Great-circle_navigation), which also computes the azimuths at the end points and intermediate way-points.

Through any two points on a sphere that are not [antipodal points](https://en.wikipedia.org/wiki/Antipodal_point) (directly opposite each other), there is a unique great circle. The two points separate the great circle into two arcs. The length of the shorter arc is the great-circle distance between the points. A great circle endowed with such a distance is called a [Riemannian circle](https://en.wikipedia.org/wiki/Riemannian_circle) in [Riemannian geometry](https://en.wikipedia.org/wiki/Riemannian_geometry).

Between antipodal points, there are infinitely many great circles, and all great circle arcs between antipodal points have a length of half the [circumference](https://en.wikipedia.org/wiki/Circumference) of the circle, or {\displaystyle \pi r}, where *r* is the [radius](https://en.wikipedia.org/wiki/Radius) of the sphere.

The [Earth](https://en.wikipedia.org/wiki/Earth) [is nearly spherical](https://en.wikipedia.org/wiki/Earth_radius), so great-circle distance formulas give the distance between points on the [surface of the Earth correct to within about 0.5%](https://en.wikipedia.org/wiki/Arc_length#Arcs_of_great_circles_on_the_Earth).[[1]](https://en.wikipedia.org/wiki/Great-circle_distance#cite_note-1)

The [vertex](https://en.wiktionary.org/wiki/vertex) is the highest-latitude point on a great circle.

The pros, i’m happy with but I’ve sought to address some of the cons in my approach.

Epsilon – link

Min-Points – user variable – an element of knowing the data.

Why not K-means?

## Clustering algorithms: k-means and DBSCAN

The k-means algorithm is likely the most common clustering algorithm. But for spatial data, the DBSCAN algorithm is far superior. Why?

The k-means algorithm groups N observations (i.e., rows in an array of coordinates) into k clusters. However, k-means is not an ideal algorithm for latitude-longitude spatial data because it minimizes variance, not geodetic distance. There is substantial distortion at latitudes far from the equator, like those of this data set. The algorithm would still “work” but its results are poor and there isn’t much that can be done to improve them.

Instead, let’s use an algorithm that works better with arbitrary distances: scikit-learn’s implementation of the [DBSCAN](http://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html) algorithm. DBSCAN clusters a spatial data set based on two parameters: a physical distance from each point, and a minimum cluster size. This method works much better for spatial latitude-longitude data.

Calculating epsilon value using kneed –

I wanted a way to calculate the epsilon because it seemed so arbitrary at first I thought it was unlikely that a user would stumble or guess the optimum value. Remember, this is being made for the average analyst, not an ODMr, not a DS and not a mathematician.

I ended up using NearestNeighbors. The use of this model is linked to the min\_pts set by the user, so in some ways, that element of calculating the eps is still in part left to the user. However, I figured the user would be better placed to choose this value than the eps.

This technique calculates the average distance between each point and its k nearest neighbors, where k = the MinPts value you selected. The average k-distances are then plotted in ascending order on a k-distance graph. You’ll find the optimal value for ε at the point of maximum curvature (i.e. where the graph has the greatest slope).

This…

# **A Systematic Method for Tuning the eps Value**

Since the **eps** figure is proportional to the expected number of neighbours discovered, we can use the nearest neighbours to reach a fair estimation for **eps**. Let us compute the nearest neighbours.

Distance variation at the 10th neighbour

Note that in the nearest neighbour calculation, the point itself will appear as the first nearest neighbour. So we seek the 11 nearest neighbours. We sort the distance to the 10th nearest neighbour and plot the distance variation. As we can see, the elbow point appears somewhere in between **0.1** and **0.3**. Quite what we were expecting isn’t it? I choose the 10th neighbour considering the fact that I pick 10 as **min\_samples** value for clustering. I hope it makes sense so far.

# **KneeLocator to Detect Elbow Point**

[Ville Satopaa](https://ieeexplore.ieee.org/author/37704602300) et al. presented the paper “[**Finding a “Kneedle” in a Haystack: Detecting Knee Points in System Behavior**](https://doi.org/10.1109/ICDCSW.2011.20)” in the year 2011. In this article, for the purpose of detecting the **elbow point** (or **knee point**), I will be using their python library [**kneed**](https://pypi.org/project/kneed/). We can use the #

The plot of Knee Point

We can see that the detected knee point by this method is at distance 0.178. Now we can use this value as our **eps** to see how our new clustering would look like.

DBSCAN with Auto-detected Eps

We can see that we have a reasonable estimate of the actual clustering. This is usually good enough for research work. If non-existence of out-liers is an intuitive assumption for the scenario, one can simply use the computed nearest neighbours to re-assign the outlier points (named as cluster--1) to detected clusters.

# **Limitations**

There are a few implicit assumptions in this approach.

1. Densities across all the clusters are the same.
2. Cluster sizes or standard deviations are the same.

These assumptions are implied when we consider the same neighbour level for knee computation. However, in the original data, we can clearly see that the densities are not the same. This is the main reason why we observe a few outliers even though the points are distributed using a fixed standard deviation when we create blobs. Moreover, fixing these is beyond the scope of this article.

### Why ball\_tree - **1.6.4.3. Ball Tree**

To address the inefficiencies of KD Trees in higher dimensions, the ball tree data structure was developed. Where KD trees partition data along Cartesian axes, ball trees partition data in a series of nesting hyper-spheres. This makes tree construction more costly than that of the KD tree, but results in a data structure which can be very efficient on highly structured data, even in very high dimensions.

A ball tree recursively divides the data into nodes defined by a centroid C and radius r, such that each point in the node lies within the hyper-sphere defined by r and C. The number of candidate points for a neighbor search is reduced through use of the triangle inequality:

|x+y|≤|x|+|y|

With this setup, a single distance calculation between a test point and the centroid is sufficient to determine a lower and upper bound on the distance to all points within the node. Because of the spherical geometry of the ball tree nodes, it can out-perform a KD-tree in high dimensions, though the actual performance is highly dependent on the structure of the training data. In scikit-learn, ball-tree-based neighbors searches are specified using the keyword algorithm = 'ball\_tree', and are computed using the class **[BallTree](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.BallTree.html" \l "sklearn.neighbors.BallTree" \o "sklearn.neighbors.BallTree)**. Alternatively, the user can work with the **[BallTree](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.BallTree.html" \l "sklearn.neighbors.BallTree" \o "sklearn.neighbors.BallTree)** class directly.

RADIANS - .fit() required the coordinates to be in radians when using the haversine method.

AND SOME STUFF ABOUT RADIANS

It only makes sense to convert an angle (ø) to a distance (d) when the distance in question is on the circumference of a circle or on the surface of a sphere. When that is the case, use the equation ø = d/r – where r is the radius of the circle or sphere. This gives a value in radians, which is easy to convert to degrees. If you know the angle in degrees and want to find the arc length, convert the angle to radians and then use the converse expression: d = ø • r. To get distance in English units, you must express the radius in English units. Similarly, you must express the radius in metric units to get the distance in kilometers, meters, centimeters or millimeters.

## Measuring Angles in Radians

A radian is an angular measurement based on the length of the radius of a circle or sphere. The radius is a line drawn from the center of the circle to a point A on its circumference or on its perimeter if it is a sphere. When a radial line moves from point A to another point B on the circumference, it traces an arc of length d while, at the same time, scribing an angle ø at the center point of the circle.

By definition, one radian is the angle you scribe when the length of the arc from point A to point B equals the length of the radius. In general, you determine the magnitude of any angle ø in radians by dividing the arc length traced by the radian lines between two points by the radius. This is the mathematical expression: ø (radians) = d/r. For this expression to work, you must express arc length and radius in the sa

[RELATED](https://sciencing.com/convert-latitude-longtitude-feet-2724.html)

### **[How to Convert Latitude & Longtitude Into Feet](https://sciencing.com/convert-latitude-longtitude-feet-2724.html)**

**Updated March 13, 2018**

*By Chris Deziel*

It only makes sense to convert an angle (ø) to a distance (d) when the distance in question is on the circumference of a circle or on the surface of a sphere. When that is the case, use the equation ø = d/r – where r is the radius of the circle or sphere. This gives a value in radians, which is easy to convert to degrees. If you know the angle in degrees and want to find the arc length, convert the angle to radians and then use the converse expression: d = ø • r. To get distance in English units, you must express the radius in English units. Similarly, you must express the radius in metric units to get the distance in kilometers, meters, centimeters or millimeters.

## Measuring Angles in Radians

A radian is an angular measurement based on the length of the radius of a circle or sphere. The radius is a line drawn from the center of the circle to a point A on its circumference or on its perimeter if it is a sphere. When a radial line moves from point A to another point B on the circumference, it traces an arc of length d while, at the same time, scribing an angle ø at the center point of the circle.

By definition, one radian is the angle you scribe when the length of the arc from point A to point B equals the length of the radius. In general, you determine the magnitude of any angle ø in radians by dividing the arc length traced by the radian lines between two points by the radius. This is the mathematical expression: ø (radians) = d/r. For this expression to work, you must express arc length and radius in the same units.

**02:2612:50**

For example, suppose you want to determine the angle of the arc traced by radial lines extending from the center of the earth to San Francisco and to New York. These two cities are 2,572 miles (4,139 kilometers) apart, and the equatorial radius of the earth is 3,963 miles (6378 kilometers). We can find the angle using either the metric or English units, as long as we use them consistently: 2,572 miles/3,963 miles = 4,139 km/6,378 km = 0.649 radians.