**Top Three Clustering Algorithms You Should Know Instead of K-means Clustering**

**A comprehensive guide to industry leading clustering techniques**



Photo by [Mel Poole](https://unsplash.com/@melpoole?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText) on [Unsplash](https://unsplash.com/s/photos/clusters?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText)

K-means clustering is arguably one of the most commonly used clustering techniques in the world of data science (anecdotally speaking), and for good reason. It’s simple to understand, easy to implement, and is computationally…

However, there are several limitations of k-means clustering which hinders its ability to be a strong clustering technique:

* **K-means clustering assumes that the data points are distributed in a spherical shape**, which may not always be the case in real-world data sets. This can lead to suboptimal cluster assignments and poor performance on non-spherical data.
* **K-means clustering requires the user to specify the number of clusters in advance**, which can be difficult to do accurately in many cases. If the number of clusters is not specified correctly, the algorithm may not be able to identify the underlying structure of the data.
* **K-means clustering is sensitive to the presence of outliers and noise in the data**, which can cause the clusters to be distorted or split into multiple clusters.
* **K-means clustering is not well-suited for data sets with uneven cluster sizes** **or non-linearly separable data**, as it may be unable to identify the underlying structure of the data in these cases.

And so in this article, I wanted to talk about three clustering techniques that you should know as alternatives to k-means clustering:

1. DBSCAN
2. Hierarchical Clustering
3. Spectral Clustering

Enjoying this article? [**Subscribe and become a member today**](https://terenceshin.medium.com/membership)to never miss another article on data science guides, tricks and tips, life lessons, and more!

**1. DBSCAN**

**What is DBSCAN?**

DBSCAN is a clustering algorithm that groups data points into clusters based on the **density of the points**.

The algorithm works by identifying points that are in high-density regions of the data and expanding those clusters to include all points that are nearby. Points that are not in high-density regions and are not close to any other points are considered noise and are not included in any clusters.

This means that DBSCAN can automatically identify the number of clusters in a dataset, unlike other clustering algorithms that require the number of clusters to be specified in advance. DBSCAN is useful for data that has a lot of noise or for data that doesn’t have well-defined clusters.

**How DBSCAN works**

The mathematical details of how DBScan works can be somewhat complex, but the basic idea is as follows.

1. Given a dataset of points in space, the algorithm first defines a distance measure (generally the Euclidean distance) that determines how close two points are to each other. This distance measure is typically based on the , which is the straight-line distance between two points in space.
2. Once the distance measure has been defined, the algorithm then uses this measure to identify clusters in the dataset. It does this by starting with a random point in the dataset, and then calculating the distance between that point and all the other points in the dataset. If the distance between two points is less than a specified threshold (known as the “eps” parameter), then the algorithm considers those two points to be part of the same cluster.
3. The algorithm then repeats this process for each point in the dataset, and iteratively builds up clusters by adding points that are within the specified distance of each other. Once all the points have been processed, the algorithm will have identified all the clusters in the dataset.

**Why DBSCAN is better than K-means Clustering**

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm that is often considered to be superior to k-means clustering in many situations. This is because DBSCAN has several advantages over k-means clustering, including:

* **DBSCAN does not require the user to specify the number of clusters in advance**, which makes it well-suited for data sets where the number of clusters is not known. In contrast, k-means clustering requires the number of clusters to be specified in advance, which can be difficult to do accurately in many cases.
* **DBSCAN can handle data sets with varying densities and cluster sizes**, as it groups data points into clusters based on density rather than using a fixed number of clusters. In contrast, k-means clustering assumes that the data points are distributed in a spherical shape, which may not always be the case in real-world data sets.
* **DBSCAN can identify clusters with arbitrary shapes**, as it does not impose any constraints on the shape of the clusters. In contrast, k-means clustering assumes that the data points are distributed in spherical clusters, which can limit its ability to identify clusters with complex shapes.
* **DBSCAN is robust to the presence of noise and outliers in the data**, as it can identify clusters even if they are surrounded by points that are not part of the cluster. In contrast, k-means clustering is sensitive to noise and outliers, and they can cause the clusters to be distorted or split into multiple clusters.

Overall, DBSCAN is useful when the data has a lot of noise or when the number of clusters is not known in advance. Unlike other clustering algorithms, which require the number of clusters to be specified, DBSCAN can automatically identify the number of clusters in a dataset. This makes it a good choice for data that doesn’t have well-defined clusters or when the structure of the data is not known. DBSCAN is also less sensitive to the shape of the clusters than other algorithms, so it can identify clusters that are not circular or spherical.

**Example of DBSCAN**

Practically speaking, imagine that you have a dataset containing the locations of different shops in a city. You want to use DBScan to identify clusters of shops in the city. The algorithm would then identify clusters of shops in the city based on the density of shops in different areas. For example, if there is a high concentration of shops in a particular neighborhood, the algorithm might identify that neighborhood as a cluster. It would also identify any areas of the city where there are very few shops as “noise” that does not belong to any cluster.

Below is some starting code to set up DBSCAN in practice.

# Import library and create instance of model  
from sklearn.cluster import DBSCAN  
  
dbscan = DBSCAN(eps=0.5, min\_samples=5)  
  
# Fit the DBSCAN model to our data by calling the `fit` method  
dbscan.fit(customer\_locations)  
  
# Access the clusters by using the `labels\_` attribute  
clusters = dbscan.labels\_

The **clusters** variable contains a list of values, where the value represents what cluster each index number is in. By joining this to the original data, you can see which data points are associated with which clusters.

**Check out** [**Saturn Cloud**](https://saturncloud.io/?utm_source=Medium+&utm_medium=TDS&utm_campaign=Terence+Shin&utm_term=top-5-clustering-algorithms) **if you want to build your first clustering model using the code above!**

**2. Hierarchical Clustering**

**What is Hierarchical Clustering?**

Hierarchical clustering is a method of cluster analysis that is used to group similar objects into clusters based on their similarity. It is a type of clustering algorithm that creates a **hierarchy of clusters**, with each cluster being divided into smaller sub-clusters until all objects in the dataset are assigned to a cluster.

**How Hierarchical Clustering works**

Imagine that you have a dataset containing the heights and weights of different people. You want to use hierarchical clustering to group the people into clusters based on their height and weight.

1. You would first need to calculate the distance between all pairs of people in the dataset. Once you have calculated the distances between all pairs of people, you would then use a hierarchical clustering algorithm to group the people into clusters.
2. The algorithm would start by treating each person as a separate cluster, and then it would iteratively merge the closest pairs of clusters until all the people are grouped into a single hierarchy of clusters. For example, the algorithm might first merge the two people who are closest to each other, and then merge that cluster with the next closest cluster, and so on, until all the people are grouped into a single hierarchy of clusters.

**Why Hierarchical Clustering is better than K-means Clustering**

Hierarchical clustering is a good choice when the goal is to produce a tree-like visualization of the clusters, called a dendrogram. This can be useful for exploring the relationships between the clusters and for identifying clusters that are nested within other clusters. Hierarchical clustering is also a good choice when the number of samples is small, because it does not require the number of clusters to be specified in advance like some other algorithms do. Additionally, hierarchical clustering is less sensitive to outliers than other algorithms, so it can be a good choice for data that has a few outlying points.

There are several other reasons why hierarchical clustering is better than k-means:

* **Hierarchical clustering also does not require the user to specify the number of clusters in advance**.
* **Hierarchical clustering can also handle data sets with varying densities and cluster sizes**, as it groups data points into clusters based on similarity rather than using a fixed number of clusters.
* Hierarchical clustering produces a hierarchy of clusters, **which can be useful for visualizing the structure of the data and identifying relationships between clusters**.
* Hierarchical clustering is also robust to the presence of noise and outliers in the data, as it can identify clusters even if they are surrounded by points that are not part of the cluster.

**3. Spectral Clustering**

**What is Spectral Clustering?**

Spectral clustering is a clustering algorithm that uses the eigenvectors of a similarity matrix to identify clusters. The similarity matrix is constructed using a kernel function, which measures the similarity between pairs of points in the data. The eigenvectors of the similarity matrix are then used to transform the data into a new space where the clusters are more easily separable. Spectral clustering is useful when the clusters have a non-linear shape, and it can handle noisy data better than k-means.

**Why Spectral Clustering is better than K-means Clustering**

Spectral clustering is a good choice when the data is not well-separated and the clusters have a complex, non-linear structure. Unlike other clustering algorithms that only consider the distances between points, spectral clustering also takes into account the relationship between points, which can make it more effective at identifying clusters that have a more complex shape.

Spectral clustering is also less sensitive to the initial configuration of the clusters, so it can produce more stable results than other algorithms. Additionally, spectral clustering is able to handle large datasets more efficiently than other algorithms, so it can be a good choice when working with very large datasets.

Several other reasons why Spectral clustering is better than K-means include the following:

* **Spectral clustering does not require the user to specify the number of clusters in advance.**
* **Spectral clustering can handle data sets with complex or non-linear patterns**, as it uses the eigenvectors of a similarity matrix to identify clusters.
* **Spectral clustering is robust to the presence of noise and outliers** in the data, as it can identify clusters even if they are surrounded by points that are not part of the cluster.
* **Spectral clustering can identify clusters with arbitrary shapes**, as it does not impose any constraints on the shape of the clusters.

**Example of Spectral Clustering**

To use Spectral clustering in Python, you can use the following code as a starting point to build a Spectral Cluster model:

# import library  
from sklearn.cluster import SpectralClustering  
  
# create instance of model and fit to data  
model = SpectralClustering()  
model.fit(data)  
  
# access model labels  
clusters = model.labels\_

Again, the **clusters** variable contains a list of values, where the value represents what cluster each index number is in. By joining this to the original data, you can see which data points are associated with which clusters.

**Comparing the Three Algorithms**

Both DBSCAN and spectral clustering are density-based clustering algorithms, which means they identify clusters by finding groups of points that are densely packed together. However, there are some key differences between the two algorithms that can make one more appropriate to use than the other in certain situations.

DBSCAN is better suited to data that has well-defined clusters and is relatively free of noise. It is also good at identifying clusters that have a consistent density throughout, meaning that the points in the cluster are about the same distance apart from each other. This makes it a good choice for data that has a clear structure and is easy to visualize.

On the other hand, spectral clustering is better suited to data that has a more complex, non-linear structure and may not have well-defined clusters. It is also less sensitive to the initial configuration of the clusters and can handle large datasets more efficiently, so it is a good choice for data that is more challenging to cluster.

Hierarchical clustering is unique in the sense that it produces a tree-like visualization of the clusters, called a dendrogram. This makes it a good choice for exploring the relationships between the clusters and for identifying clusters that are nested within other clusters.

In comparison to DBSCAN and spectral clustering, hierarchical clustering is a slower algorithm and is not as effective at identifying clusters that have a complex, non-linear structure. It is also not as good at identifying clusters that have a consistent density throughout, so it may not be the best choice for data that has well-defined clusters. However, it can be a useful tool for exploring the structure of a dataset and for identifying clusters that are nested within other clusters.