Contents

[https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68 2](#_Toc523956102)

[Types of Clustering 2](#_Toc523956103)

[Representative based algo - K-means: 2](#_Toc523956104)

[Algorithm 3](#_Toc523956105)

[Calculate no of clusters 3](#_Toc523956106)

[Elbow Method 3](#_Toc523956107)

[Silhouette method 4](#_Toc523956108)

[Challenges 4](#_Toc523956109)

[Assumptions of K mean Clustering 5](#_Toc523956110)

[Broken Assumptions effect 5](#_Toc523956111)

[Advantages 7](#_Toc523956112)

[Drawbacks 7](#_Toc523956113)

[Conclusion 7](#_Toc523956114)

[Hierarchical Clustering 7](#_Toc523956115)

[Advantages 8](#_Toc523956116)

[The main advantage of hierarchical clustering is that the clusters are not assumed to be globular. In addition, it scales well to larger datasets. 8](#_Toc523956117)

[Disadvantages 8](#_Toc523956118)

[DBSCAN - Density based clustering algorithm 8](#_Toc523956119)

[Advantage over K mean 10](#_Toc523956120)

[Weaknesses 10](#_Toc523956121)

[Mean Shift Clustering 10](#_Toc523956122)

[Expectation–Maximization (EM) Clustering using Gaussian Mixture Models (GMM). 12](#_Toc523956123)

[Clustering is defined by these three properties: 14](#_Toc523956124)

[Difference between K Means and Hierarchical clustering 14](#_Toc523956125)

[Trade off for Number of clulsters 15](#_Toc523956126)

https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68

# Types of Clustering

Broadly speaking, clustering can be divided into two subgroups :

● Hard Clustering: In hard clustering, each data point either belongs to a cluster

completely or not. For example, in the above example each customer is put into one

group out of the 10 groups.

● Soft Clustering : In soft clustering, instead of putting each data point into a separate

cluster, a probability or likelihood of that data point to be in those clusters is

assigned. For example, from the above scenario each costumer is assigned a

probability to be in either of 10 clusters of the retail store.

Expectation Maximization (EM) — Soft Clustering:

In order to understand the “soft clustering” approach let’s think about a particular situation where the k-means-algorithm’s weakness is well depicted. If we have a point that is equidistant between two clusters, then, depending on how the cluster centers are initialized, that point could be assigned to one or the other cluster. Soft clustering instead, using probability, can make that point as

“shared” between the two clusters. While k-mean is doing a deterministic assignment of the data,

EM is instead using a probabilistic approach .The estimation step finds the likelihood that a data “i” comes from a cluster “j”. The maximization step uses this “expected” model and maximizes it to find the data fits.

# Representative based algo - K-means:

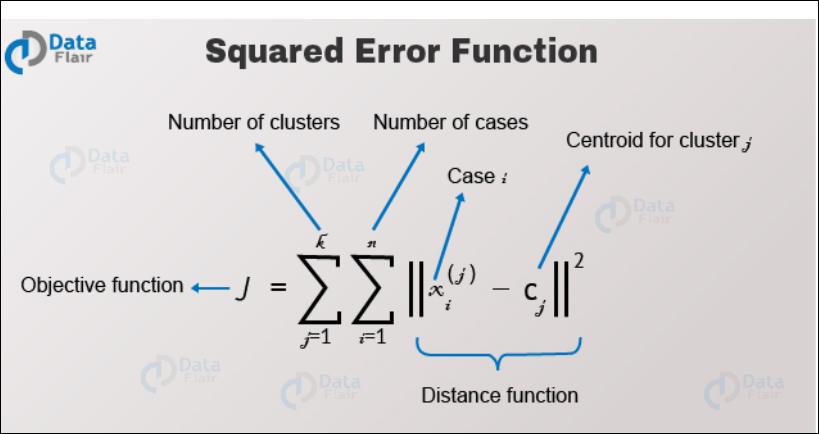
This algorithm takes a deterministic approach and it operates in two steps:

1-assign

2-optimize

## objective

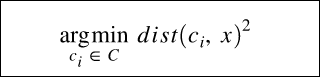
The objective of the K-means clustering is to minimize the [Euclidean distance](https://en.wikipedia.org/wiki/Euclidean_distance) that each point has from the centroid of the cluster. This is known as **intra-cluster variance** and can be minimized using the following squared error function – The SSE is defined as the sum of the squared distance between each member of the cluster and its centroid



## Algorithm

1. Data assigment step:

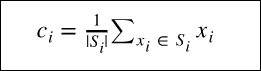
Each centroid defines one of the clusters. In this step, each data point is assigned to its nearest centroid, based on the squared Euclidean distance. More formally, if *ci* is the collection of centroids in set *C*, then each data point *x* is assigned to a cluster based on



where*dist*( *·*) is the standard (*L*2) Euclidean distance. Let the set of data point assignments for each *ith* cluster centroid be *Si*.

2. Centroid update step:

In this step, the centroids are recomputed. This is done by taking the mean of all data points assigned to that centroid's cluster.



The algorithm iterates between steps one and two until a stopping criteria is met (i.e., no data points change clusters, the sum of the distances is minimized, or some maximum number of iterations is reached).

This algorithm is guaranteed to converge to a result. The result may be a local optimum (i.e. not necessarily the best possible outcome), meaning that assessing more than one run of the algorithm with randomized starting centroids may give a better outcome.

## Steps

K means is an iterative clustering algorithm that aims to find local maxima in each

iteration. This algorithm works in these 5 steps

1. Specify the desired number of clusters K :

2. Randomly assign each data point to a cluster

3. Compute cluster centroids

4. Re-assign each point to the closest cluster centroid ( This is done by computing the distance between centroid and each data point)

5. Re-compute cluster centroids by taking the mean of all the vector points in the group.

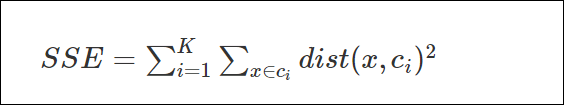
6. Repeat these steps for a set number of iterations or until the group centers don’t change much between iterations.

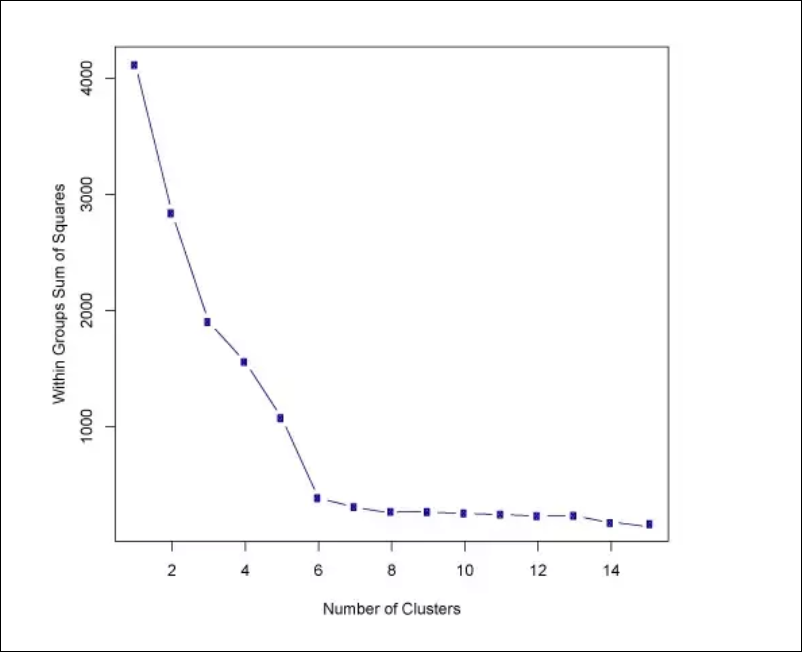
You can choose the number of clusters by visually inspecting your data points, but you will soon realize that there is a lot of ambiguity in this process for all except the simplest data sets. This is not always bad, because you are doing unsupervised learning and there's some inherent subjectivity in the labeling process. Here, having previous experience with that particular problem or something similar will help you choose the right value.

Calculate no of clusters

Elbow Method

Here, you draw a curve between WSS (within sum of squares) and the number of clusters. It is called elbow method because the curve looks like a human arm and the elbow point gives us the optimum number of clusters. As you can see that after the elbow point, there is a very slow change in the value of WSS, so you should take the elbow point value as the final number of clusters.

If you want some hint about the number of clusters that you should use, you can apply   
First of all, compute the sum of squared error (SSE) for some values of k (for example 2, 4, 6, 8, etc.). The SSE is defined as the sum of the squared distance between each member of the cluster and its centroid. Mathematically:  
  
  
 If you plot k against the SSE, you will see that *the error decreases as k gets larger*; this is because when the number of clusters increases, they should be smaller, so distortion is also smaller. The idea of the elbow method is to choose the k at which the SSE decreases abruptly. This produces an "elbow effect" in the graph, as you can see in the following picture:



In this case, k=6 is the value that the Elbow method has selected. Take into account that the Elbow method is an heuristic and, as such, it may or may not work well in your particular case. Sometimes, there are more than one elbow, or no elbow at all. In those situations you usually end up calculating the best k by evaluating how well k-means performs in the context of the particular clustering problem you are trying to solve.

One of the metrics that is commonly used to compare results across different values of K is the mean distance between data points and their cluster centroid. Since increasing the number of clusters will always reduce the distance to data points, increasing K will always decrease this metric, to the extreme of reaching zero when K is the same as the number of data points. Thus, this metric cannot be used as the sole target. Instead, mean distance to the centroid as a function of K is plotted and the "elbow point," where the rate of decrease sharply shifts, can be used to roughly determine K.

### Silhouette method

It plots cohesion and separation

### G-means

## Challenges

● The choice of how many clusters, which is probably the most important decision to make before running the algorithm. Depending on the distribution of the data, in fact, one could be inclined to select different number of clusters.

● The initial placement of the clusters. This can influence the final result as the algorithm can get stuck to a local minimum and therefore cluster the data in an odd way.

Possible resolution – Restart many times and use whichever results give the best final objective

## Assumptions of K mean Clustering

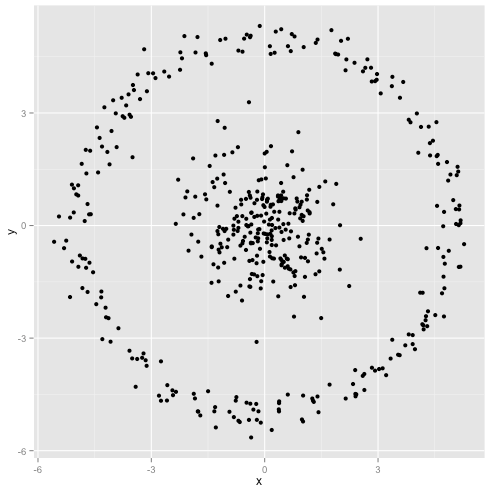
Most of them say that:

* k-means assumes the variance of the distribution of each attribute (variable) is spherical All clusters are spherical (i.i.d. Gaussian).
* all variables have the same variance(All axes have the same distribution and thus variance)
* the prior probability for all k clusters is the same, i.e., each cluster has roughly equal number of observations;

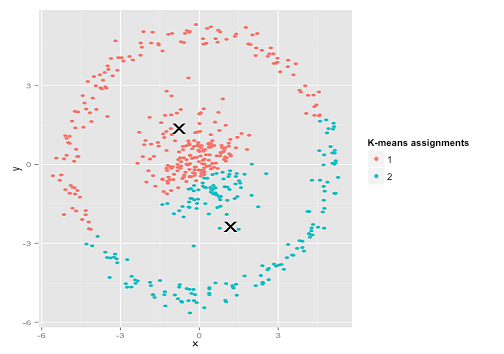
## Broken Assumptions effect

**Broken Assumption: Non-Spherical Data**

You argue that the k-means algorithm will work fine on non-spherical clusters. Non-spherical clusters like... these?

[](https://i.stack.imgur.com/g5Jb8.png)

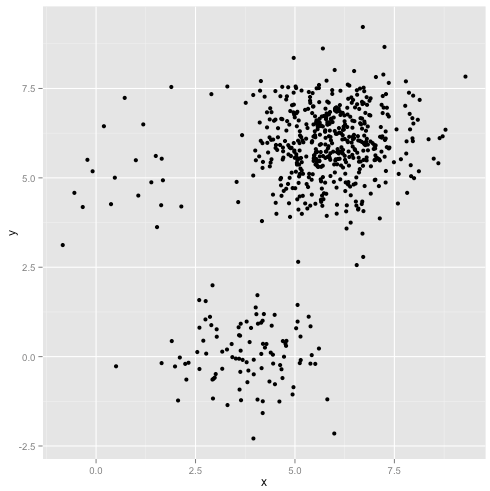
Maybe this isn't what you were expecting- but it's a perfectly reasonable way to construct clusters. Looking at this image, we humans immediately recognize two natural groups of points- there's no mistaking them. So let's see how k-means does: assignments are shown in color, imputed centers are shown as X's.



Well, that's not right. K-means was trying to fit a [square peg in a round hole](http://en.wikipedia.org/wiki/Square_peg_in_a_round_hole)- trying to find nice centers with neat spheres around them- and it failed. Yes, it's still minimizing t he within-cluster sum of squares- but just like in Anscombe's Quartet above, it's a Pyrrhic victory!

**Broken Assumption: Unevenly Sized Clusters**

What if the clusters have an uneven number of points- does that also break k-means clustering? Well, consider this set of clusters, of sizes 20, 100, 500. I've generated each from a multivariate Gaussian:

[](https://i.stack.imgur.com/WiH4T.png)

This looks like k-means could probably find those clusters, right? Everything seems to be generated into neat and tidy groups. So let's try k-means:



Ouch. What happened here is a bit subtler. In its quest to minimize the within-cluster sum of squares, the k-means algorithm gives more "weight" to larger clusters. In practice, that means it's happy to let that small cluster end up far away from any center, while it uses those centers to "split up" a much larger cluster

### **Sensitive to scale**

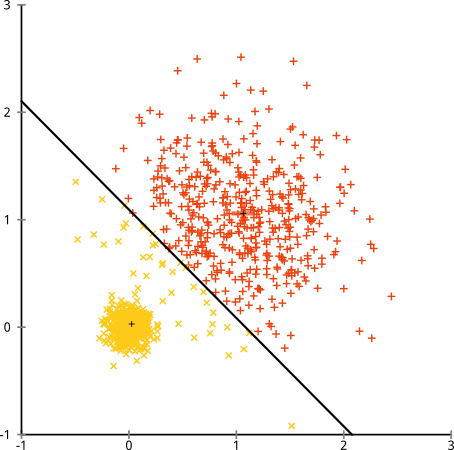
Rescaling your datasets will completely change results. While this itself is not bad, not realizing that you have to spend extra attention to scaling your data is bad. Scaling factors are extra d hidden parameters in k-means that "default" to 1 and thus are easily overlooked, yet have a major impact (but of course this applies to many other algorithms, too).

This is probably what you referred to as "all variables have the same variance". Except that ideally, you would also consider non-linear scaling when appropriate.

Also be aware that **it is only a heuristic to scale every axis to have unit variance**. This doesn't ensure that k-means works.

Scaling depends on the meaning of your data set. And if you have more than one cluster, you would want every cluster (independently) to have the same variance in every variable, too.

Here is a classic counterexample of data sets that k-means cannot cluster. Both axes are i.i.d. in each cluster, so it would be sufficient to do this in 1 dimension. But the clusters have varying variances, and k-means thus splits them incorrectly.



I don't think this counterexample for k-means is covered by your points:

* All clusters are spherical (i.i.d. Gaussian).
* All axes have the same distribution and thus variance.
* Both clusters have 500 elements each.

Yet, k-means still fails badly (and it gets worse if I increase the variance beyond 0.5 for the larger cluster) But: **it is not the algorithm that failed. It's the assumptions, which don't hold**. K-means is working perfectly, it's just optimizing the wrong criterion.

## Advantages

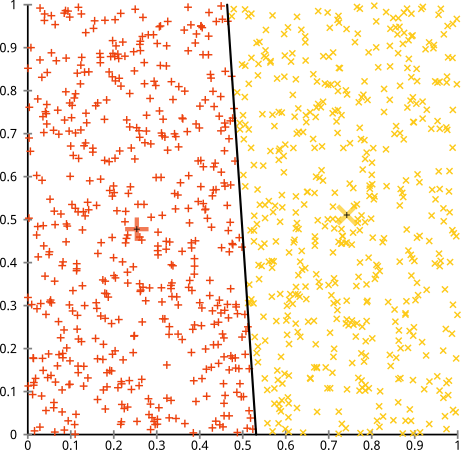
it’s pretty fast, as all we’re really doing is computing the distances between points and group centers; very few computations! It thus has a linear complexity O(n)

## Drawbacks

* Run k-means on uniform data, and you will still get clusters! It doesn't tell you when the data just does *not* cluster, and can take your research into a dead end this way.
* needs **linear separability** of the clusters
* need to specify the **number of clusters**
* It include every data point in the cluster no matter how far they are its nearest centroid.

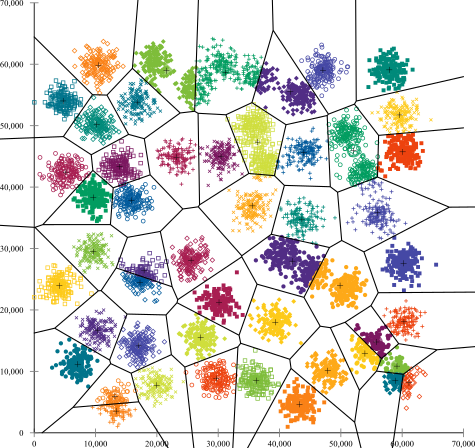
### **Clustering non-clustered data**

Run k-means on uniform data, and you will still get clusters! It doesn't tell you when the data just does not cluster, and can take your research into a dead end this way.



### **Even on perfect data sets, it can get stuck in a local minimum**

Below is the best of 10 runs of k-means on the classic A3 data set. This is a synthetic data set, designed for k-means. 50 clusters, each of Gaussian shape, reasonably well separated. Yet, it only with k-means++ and 100 iterations I did get the expected result... (below is 10 iterations of regular k-means, for illustration).



You'll quickly find many clusters in this data set, where k-means failed to find the correct structure. For example in the bottom right, a cluster was broken into three parts. But there is no way, k-means is going to move one of these centroids to an entirely different place of the data set - it's trapped in a local minimum (and this already was the best of 10 runs!)

And there are many of such local minima in this data set. Very often when you get two samples from the same cluster, it will get stuck in a minimum where this cluster remains split, and two other clusters merged instead. Not always, but very often. So you need a lot of iterations to have a lucky pick. With 100 iterations of k-means, I still counted 6 errors, and with 1000 iterations I got this down to 4 errors. K-means++ by the way it weights the random samples, works much better on this data set.

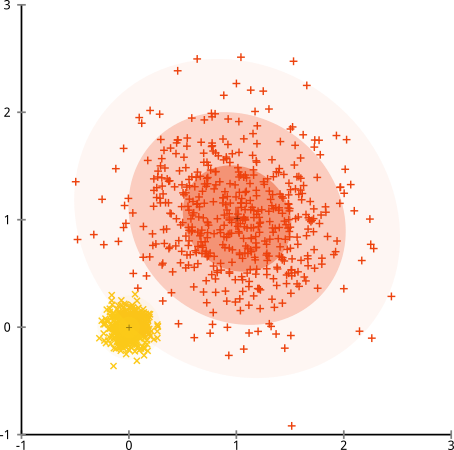
### **Means are continuous**

While you can run k-means on binary data (or one-hot encoded categorical data) the results will not be binary anymore. So you do get a result out, but you may be unable to interpret it in the end, because it has a different data type than your original data.

### **Hidden assumption: SSE is worth minimizing**

This is essentially already present in above answer, nicely demonstrated with linear regression. There are some use cases where k-means makes perfect sense. When Lloyd had to decode PCM signals, he did know the number of different tones, and least squared error minimizes the chance of decoding errors. And in color quantization of imaged, you do minimize color error when reducing the palette, too. But on your data, is the sum of squared deviations a meaningful criterion to minimize?

In above counterexample, the variance is not worth minimizing, because it depends on the cluster. Instead, a Gaussian Mixture Model should be fit to the data, as in the figure below:



(But this is not the ultimate method either. It's just as easy to construct data that does not satisfy the "mixture of k Gaussian distributions" assumptions, e.g., by adding a lot of background noise)

### **Too easy to use badly**

All in all, it's too easy to throw k-means on your data, and nevertheless get a result out (that is pretty much random, but you won't notice). I think it would be better to have a method which can fail if you haven't understood your data...

## Conclusion

Even though it works very well, K-Means clustering has its own issues. That include:

* If you run K-means on uniform data, you will get clusters
* Sensitive to scale due to its reliance on Euclidean distance.
* Even on perfect data sets, it can get stuck in a local minimum

K-Medians is another clustering algorithm related to K-Means, except instead of recomputing the group center points using the mean we use the median vector of the group.

## Why do we normalize data before performing K-means clustering? Why do we remove outliers before performing K-means clustering? How does it affect number of K-means cluster?

# Hierarchical Clustering

It cluster data with distance. It is greedy algo

**Bottom UP (Agglomerative)** – Data points are successively agglomerated into higher level clusters. Algorithm starts with individual data points. In each iteration , 2 clusters are selected that are deemed to be close as possible . These clusters are merged and replaced with newly created cluster. In the end, this algorithm

**Top Down –** It approach starting with all data points assigned in the same cluster and recursively performing splits till each data point is assigned a separate cluster.

**Algorithm**

1. We begin by treating each data point as a single cluster i.e if there are X data points in our dataset then we have X clusters. We then select a distance metric that measures the distance between two clusters. As an example we will use average linkage which defines the distance between two clusters to be the average distance between data points in the first cluster and data points in the second cluster.
2. On each iteration we combine two clusters into one. The two clusters to be combined are selected as those with the smallest average linkage. I.e according to our selected distance metric, these two clusters have the smallest distance between each other and therefore are the most similar and should be combined.
3. Step 2 is repeated until we reach the root of the tree i.e we only have one cluster which contains all data points. In this way we can select how many clusters we want in the end, simply by choosing when to stop combining the clusters i.e when we stop building the tree!

For termination max or min threshold is used.

dendrogram - The order of merging naturally creates a tree like structure illustrating the relation between different clusters which is called dendrogram.

With the use of a dendrogram you are able to see when and what gets merged; so you can choose where to stop the merging of clusters and keep the most relevant clusters. The height in the dendrogram at which two clusters are merged represents the distance between two clusters in the data space.

The **decision of the no. of clusters** that can best depict different groups can be chosen by observing the dendrogram. The best choice of the no. of clusters is the no. of vertical lines in the dendrogram cut by a horizontal line that can transverse the maximum distance vertically without intersecting a cluster.

## Advantages

## The main advantage of hierarchical clustering is that the clusters are not assumed to be globular. In addition, it scales well to larger datasets.

## Disadvantages

Much like K-Means, the user must choose the number of clusters (i.e. the level of the hierarchy to "keep" after the algorithm completes).

# DBSCAN - Density based clustering algorithm

**Why Density based**

* The major problem of K mean was that it assumes a specific shape where as DBSCAN works well with non spherical data also
* It does not force every data point into a cluster — if a data point lies outside of the specified starting parameters, it will be classified as "noise" and not included in any resulting cluster.
* DBSCAN does not require the analyst to select the number of clusters a priori — the algorithm determines this based on the parameters it's given.

DBSCAN is a well-known data clustering algorithm that is commonly used in data mining and machine learning.

Based on a set of points (let’s think in a bidimensional space as exemplified in the figure), DBSCAN groups together points that are close to each other based on a distance measurement (usually Euclidean distance) and a minimum number of points. It also marks as outliers the points that are in low-density regions.

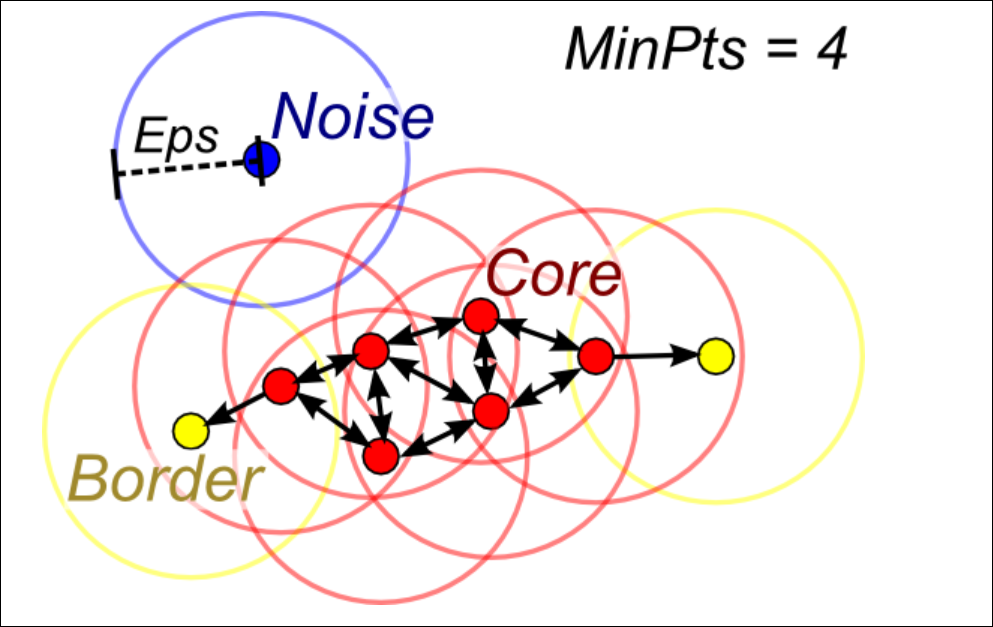
Density of data point is defined by the number of points that lie within a radius of eps of that point

Parameters:

The DBSCAN algorithm basically requires 2 parameters:

**eps**: the minimum distance between two points. It means that if the distance between two points is lower or equal to this value (eps), these points are considered neighbors.

**minPoints**: the minimum number of points to form a dense region. For example, if we set the minPoints parameter as 5, then we need at least 5 points to form a dense region or a cluster.



The [image](https://i.stack.imgur.com/5aikc.png) above shows how DBSCAN parameters work. The circles from each data point represent the epsilon distance. For a data point to be considered "core" and therefore "clusterable," it needs to have the minimum number of data points fall within its epsilon radius (in this example, that number is four including the data point itself). "Border" points are also in the core clusters and form their edges. The blue data point has neither the minimum number of data points in its neighborhood nor does it fall within a core data point's neighborhood and is thus labeled as "noise."

**Parameter estimation:**

The parameter estimation is a problem for every data mining task. To choose good parameters we need to understand how they are used and have at least a basic previous knowledge about the data set that will be used.

**eps**: if the eps value chosen is too small, a large part of the data will not be clustered. It will be considered outliers because don’t satisfy the number of points to create a dense region. On the other hand, if the value that was chosen is too high, clusters will merge and the majority of objects will be in the same cluster. The eps should be chosen based on the distance of the dataset (we can use a k-distance graph to find it), but in general small eps values are preferable.

**minPoints**: As a general rule, a minimum minPoints can be derived from a number of dimensions (D) in the data set, as minPoints ≥ D + 1. Larger values are usually better for data sets with noise and will form more significant clusters. The minimum value for the minPoints must be 3, but the larger the data set, the larger the minPoints value that should be chosen.

You can find more about parameter estimation [here](https://en.wikipedia.org/wiki/DBSCAN#Parameter_estimation).

## Advantage over K mean

It can discover cluster if arbitrary shape

It does not require number of clusters as an input parameter

Weaknesses**:**

The user must tune the hyperparameters 'epsilon' and ‘**minPoints**,' which define the density of clusters. DBSCAN is quite sensitive to these hyperparameters.

# Mean Shift Clustering

* To explain mean-shift we will consider a set of points in two-dimensional space like the above illustration. We begin with a circular sliding window centered at a point C (randomly selected) and having radius r as the kernel. Mean shift is a hill climbing algorithm which involves shifting this kernel iteratively to a higher density region on each step until convergence.
* At every iteration the sliding window is shifted towards regions of higher density by shifting the center point to the mean of the points within the window (hence the name). The density within the sliding window is proportional to the number of points inside it. Naturally, by shifting to the mean of the points in the window it will gradually move towards areas of higher point density.
* We continue shifting the sliding window according to the mean until there is no direction at which a shift can accommodate more points inside the kernel. Check out the graphic above; we keep moving the circle until we no longer are increasing the density (i.e number of points in the window).
* This process of steps 1 to 3 is done with many sliding windows until all points lie within a window. When multiple sliding windows overlap the window containing the most points is preserved. The data points are then clustered according to the sliding window in which they reside.

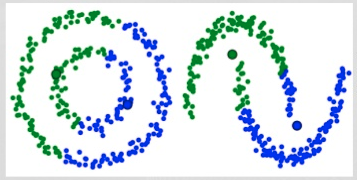
**Pros and Cons**

In contrast to K-means clustering there is no need to select the number of clusters as mean-shift automatically discovers this. That’s a massive advantage. The fact that the cluster centers converge towards the points of maximum density is also quite desirable as it is quite intuitive to understand and fits well in a naturally data-driven sense. The drawback is that the selection of the window size/radius “r” can be non-trivial.

# Expectation–Maximization (EM) Clustering using Gaussian Mixture Models (GMM).

It is a type of soft clustering.

One of the major drawbacks of K-Means is its naive use of the mean value for the cluster center. We can see why this isn’t the best way of doing things by looking at the image below. On the left hand side it looks quite obvious to the human eye that there are two circular clusters with different radius’ centered at the same mean. K-Means can’t handle this because the mean values of the clusters are a very close together. K-Means also fails in cases where the clusters are not circular, again as a result of using the mean as cluster center.



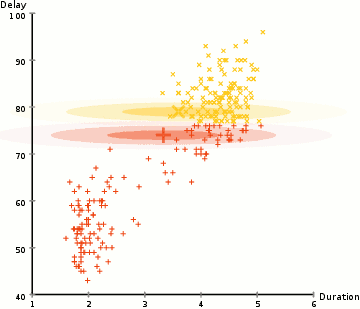
Two failure cases for K-Means

Gaussian Mixture Models (GMMs) give us more flexibility than K-Means. With GMMs we assume that the data points are Gaussian distributed; this is a less restrictive assumption than saying they are circular by using the mean.

That way, we have two parameters to describe the shape of the clusters: the mean and the standard deviation!

Taking an example in two dimensions, this means that the clusters can take any kind of elliptical shape (since we have standard deviation in both the x and y directions). Thus, each Gaussian distribution is assigned to a single cluster.

In order to find the parameters of the Gaussian for each cluster (e.g the mean and standard deviation) we will use an optimization algorithm called Expectation–Maximization (EM). Take a look at the graphic below as an illustration of the Gaussians being fitted to the clusters. Then we can proceed on to the process of Expectation–Maximization clustering using GMMs.



EM Clustering using GMMs

* We begin by selecting the number of clusters (like K-Means does) and randomly initializing the Gaussian distribution parameters for each cluster. One can try to provide a good guesstimate for the initial parameters by taking a quick look at the data too. Though note, as can be seen in the graphic above, this isn’t 100% necessary as the Gaussians start our as very poor but are quickly optimized.
* Given these Gaussian distributions for each cluster, compute the probability that each data point belongs to a particular cluster. The closer a point is to the Gaussian’s center, the more likely it belongs to that cluster. This should make intuitive sense since with a Gaussian distribution we are assuming that most of the data lies closer to the center of the cluster.
* Based on these probabilities, we compute a new set of parameters for the Gaussian distributions such that we maximize the probabilities of data points within the clusters. We compute these new parameters using a weighted sum of the data point positions, where the weights are the probabilities of the data point belonging in that particular cluster. To explain this in a visual manner we can take a look at the graphic above, in particular the yellow cluster as an example. The distribution starts off randomly on the first iteration, but we can see that most of the yellow points are to the right of that distribution. When we compute a sum weighted by the probabilities, even though there are some points near the center, most of them are on the right. Thus naturally the distribution’s mean is shifted closer to those set of points. We can also see that most of the points are “top-right to bottom-left”. Therefore the standard deviation changes to create an ellipse that is more fitted to these points, in order to maximize the sum weighted by the probabilities.
* Steps 2 and 3 are repeated iteratively until convergence, where the distributions don’t change much from iteration to iteration.

There are really 2 key advantages to using GMMs. Firstly GMMs are a lot more **flexible** in terms of **cluster covariance** than K-Means; due to the standard deviation parameter, the clusters can take on any ellipse shape, rather than being restricted to circles. K-Means is actually a special case of GMM in which each cluster’s covariance along all dimensions approaches 0. Secondly, since GMMs use probabilities, they can have multiple clusters per data point. So if a data point is in the middle of two overlapping clusters, we can simply define its class by saying it belongs X-percent to class 1 and Y-percent to class 2. I.e GMMs support **mixed** **membership**.

# Clustering is defined by these three properties:

● Richness: for any assignment of objects into clusters, there is a distance matrix such that the clustering

scheme returns that clustering.

● Scale-invariance: scaling distances by a positive value does not change the clustering.

● Consistency: shrinking intra-cluster distances and expanding inter-cluster distances does not change the final clustering.

# Difference between K Means and Hierarchical clustering

**Prior Knowledge of K** - Use hierarchical when we have no information about number of clusters

Hierarchical clustering can’t handle big data well but K Means clustering can. This is

because the time complexity of K Means is linear i.e. O(n) while that of hierarchical

clustering is quadratic i.e. O(n 2 )

**Shape of Clusters**  
K-means works well when the shape of clusters are hyper-spherical  (or circular in 2 dimensions). If the natural clusters occurring in the dataset are non-spherical then probably K-means is not a good choice.

**Repeatability** In K Means clustering, since we start with random choice of clusters, the results

produced by running the algorithm multiple times might differ. While results are

reproducible in Hierarchical clustering.

# Trade off for Number of clulsters

The more you draw clusters, the better your separation measures (BSS) will be. If you have one million customers, and you lay out one million clusters, then your measure of separation

will be at the top. But, your measure of compactness (WSS) will come out empty.

So, to optimize your clustering algorithm you need to minimize

the WSS and maximize the BSS.

You start to understand the trade off. Here is a way to solve it mathematically: lay down this

trade off on a simple graph and take your decision from there.

● The more clusters you have, the more compactness you leave behind.

● The less clusters you have, the worse your separation measure will be.

In addition, your K-means algorithm results depend on where the centroid starts. As the centroids start in different places, they might also end up in different centres. That means the same K-means algorithm, re-run, might give a different answer! To solve this issue you should run it several times.