1. What is an unsupervised learning approach? Why is it needed?

Unsupervised learning is important in the field of data science to reveal patterns that could have been missed.

Learn a bit about clustering problems in Machine Learning/ Artificial intelligence field of study.

In real life also, unsupervised learning could take into account

1. A person's personal strengths and weaknesses.
2. A new perspective that could take a look at the data at hand in a new light.

=====

Unsupervised learning is the training of an artificial intelligence (AI) algorithm using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance.

In unsupervised learning, an AI system may group unsorted information according to similarities and differences even though there are no categories provided. AI systems capable of unsupervised learning are often associated with generative learning models, although they may also use a retrieval-based approach (which is most often associated with supervised machine learning). Chatbots, self-driving cars, facial recognition programs, expert systems and robots are among the systems that may use either supervised or unsupervised learning approaches.

In unsupervised learning, an AI system is presented with unlabeled, categorized data and the system's algorithms act on the data without prior training. The output is dependent upon the coded algorithms. Subjecting a system to unsupervised learning is one way of testing AI.

Unsupervised learning algorithms can perform more complex processing tasks than supervised learning systems. However, unsupervised learning can be more unpredictable than the alternate model. While an unsupervised learning AI system might, for example, figure out on its own how to sort cats from dogs, it might also add unforeseen and undesired categories to deal with unusual breeds, creating clutter instead of order.

=====

This term encompasses all types of machine learning in which the result is unknown and there is no teacher to train the algorithm. In the case of unsupervised learning, the learning algorithm receives only the input data and is instructed to extract knowledge from this data.

There are basically two types of unsupervised learning:

1. Transformation of Records
2. Cluster method

Unsupervised Transformation

These are algorithms that generate a new representation of the data, which is easier to understand for humans or other machine learning algorithms than their original representation. One common application of the unsupervised transformation is the dimensionality reduction, which can be used to derive a composite representation of a few central features from a higher-dimensional representation of the data with many features. A common example of dimensionality reduction is the projection on two dimensions to better visualize data and thus better understand it. Another important and useful application for unsupervised transformation is finding parts or components that are the core of the data. An example of this is finding topics in a collection of text documents. The task is to find unknown topics that are mentioned in all documents. Here one tries to find out which topics occur in all documents. Such methods can be useful, for example, to follow discussions on topics such as elections, laws and pop stars.

Cluster Method

Clustering, on the other hand, divides records into separate groups with similar items. As an example, consider uploading images to a social network. To sort their pictures, the website might try to juxtapose pictures with the same person. However, the website does not know who is on which picture and how many different people are represented in their photo collection. A sensible approach would be to extract all faces and form groups with similar faces.

Challenges of unsupervised learning

The main problem with unsupervised learning is to evaluate if the algorithm has learned anything useful. Usually, unsupervised learning algorithms are applied to unlabeled data, so we do not know what the correct output should look like. That’s why it’s so hard to decide if a model is right. Therefore, unsupervised algorithms are often used in the exploration phase, where a data scientist wants to understand the data better and less as part of a large automated system. Another common application of unsupervised algorithms is preprocessing for supervised algorithms. A new representation of the data increases the learning accuracy of the monitored algorithm or reduces the memory and time overhead.

There are two types of algorithms commonly used in unsupervised learning:

k-Means

A k-means algorithm is a method of vector quantization that is also used for cluster analysis. In this case, a previously known number of k groups is formed from a set of similar objects. The algorithm is one of the most commonly used techniques for grouping objects, as it quickly finds the centers of the clusters. The algorithm prefers groups with low variance and similar size.

Apriori

The Apriori algorithm is a method for association analysis, a field of data mining. It serves to find meaningful and useful contexts in transaction-based databases, which are presented in the form of so-called association rules. A common application of the Apriori algorithm is shopping basket analysis. Items are products offered here and a purchase represents a transaction that contains the purchased items.

What is Unsupervised Learning? Unsupervised learning is a machine learning technique, where you do not need to supervise the model. Instead, you need to allow the model to work on its own to discover information. It mainly deals with the unlabelled data.

Unsupervised learning algorithms allows you to perform more complex processing tasks compared to supervised learning. Although, unsupervised learning can be more unpredictable compared with other natural learning methods.

Why Unsupervised Learning? Here, are prime reasons for using Unsupervised Learning:

Unsupervised machine learning finds all kind of unknown patterns in data. Unsupervised methods help you to find features which can be useful for categorization. It is taken place in real time, so all the input data to be analyzed and labeled in the presence of learners. It is easier to get unlabeled data from a computer than labeled data, which needs manual intervention.

Types of Unsupervised Learning Unsupervised learning problems further grouped into clustering and association problems.

Clustering

sample Cluster/group

Clustering is an important concept when it comes to unsupervised learning. It mainly deals with finding a structure or pattern in a collection of uncategorized data. Clustering algorithms will process your data and find natural clusters(groups) if they exist in the data. You can also modify how many clusters your algorithms should identify. It allows you to adjust the granularity of these groups.

There are different types of clustering you can utilize:

Exclusive (partitioning)

In this clustering method, Data are grouped in such a way that one data can belong to one cluster only.

Example: K-means

Agglomerative

In this clustering technique, every data is a cluster. The iterative unions between the two nearest clusters reduce the number of clusters.

Example: Hierarchical clustering

Overlapping

In this technique, fuzzy sets is used to cluster data. Each point may belong to two or more clusters with separate degrees of membership.

Probabilistic

This technique uses probability distribution to create the clusters

Example: Following keywords

- "man's shoe."
- "women's shoe."
- "women's glove."
- "man's glove."

can be clustered into two categories "shoe" and "glove" or "man" and "women."

Clustering Types

- Hierarchical clustering
- K-means clustering
- K-NN (k nearest neighbors)
- Principal Component Analysis
- Singular Value Decomposition
- Independent Component Analysis

Hierarchical Clustering:

Hierarchical clustering is an algorithm which builds a hierarchy of clusters. It begins with all the data which is assigned to a cluster of their own. Here, two close cluster are going to be in the same cluster. This algorithm ends when there is only one cluster left.

K-means Clustering

K means it is an iterative clustering algorithm which helps you to find the highest value for every iteration. Initially, the desired number of clusters are selected. In this clustering method, you need to cluster the data points into k groups. A larger k means smaller groups with more granularity in the same way. A lower k means larger groups with less granularity.

The output of the algorithm is a group of "labels." It assigns data point to one of the k groups. In k-means clustering, each group is defined by creating a centroid for each group. The centroids are like the heart of the cluster, which captures the points closest to them and adds them to the cluster.

K-mean clustering further defines two subgroups:

- Agglomerative clustering
- Dendrogram

Agglomerative clustering:

This type of K-means clustering starts with a fixed number of clusters. It allocates all data into the exact number of clusters. This clustering method does not require the number of clusters K as an input. Agglomeration process starts by forming each data as a single cluster.

This method uses some distance measure, reduces the number of clusters (one in each iteration) by merging process. Lastly, we have one big cluster that contains all the objects.

Dendrogram:

In the Dendrogram clustering method, each level will represent a possible cluster. The height of dendrogram shows the level of similarity between two join clusters. The closer to the bottom of the process they are more similar cluster which is finding of the group from dendrogram which is not natural and mostly subjective.

K- Nearest neighbors

K- nearest neighbour is the simplest of all machine learning classifiers. It differs from other machine learning techniques, in that it doesn't produce a model. It is a simple algorithm which stores all available cases and classifies new instances based on a similarity measure.

It works very well when there is a distance between examples. The learning speed is slow when the training set is large, and the distance calculation is nontrivial.

Principal Components Analysis:

In case you want a higher-dimensional space. You need to select a basis for that space and only the 200 most important scores of that basis. This base is known as a principal component. The subset you select constitute is a new space which is small in size compared to original space. It maintains as much of the complexity of data as possible.

Association

Association rules allow you to establish associations amongst data objects inside large databases. This unsupervised technique is about discovering interesting relationships between variables in large databases. For example, people that buy a new home most likely to buy new furniture.

Other Examples:

- A subgroup of cancer patients grouped by their gene expression measurements
- Groups of shopper based on their browsing and purchasing histories
- Movie group by the rating given by movies viewers

Supervised vs. Unsupervised Machine Learning

Parameters	Supervised machine learning technique	Unsupervised machine learning technique
Input Data	Algorithms are trained using labeled data.	Algorithms are used against data which is not labelled
Computational Complexity	Supervised learning is a simpler method.	Unsupervised learning is computationally complex
Accuracy	Highly accurate and trustworthy method.	Less accurate and trustworthy method.

Applications of unsupervised machine learning

Some applications of unsupervised machine learning techniques are:

- Clustering automatically split the dataset into groups base on their similarities
- Anomaly detection can discover unusual data points in your dataset. It is useful for finding fraudulent transactions
- Association mining identifies sets of items which often occur together in your dataset
- Latent variable models are widely used for data preprocessing. Like reducing the number of features in a dataset or decomposing the dataset into multiple components

Disadvantages of Unsupervised Learning

- You cannot get precise information regarding data sorting, and the output as data used in unsupervised learning is labeled and not known
- Less accuracy of the results is because the input data is not known and not labeled by people in advance. This means that the machine requires to do this itself.
- The spectral classes do not always correspond to informational classes.
- The user needs to spend time interpreting and label the classes which follow that classification.
- Spectral properties of classes can also change over time so you can't have the same class information while moving from one image to another.

Summary

- Unsupervised learning is a machine learning technique, where you do not need to supervise the model.
- Unsupervised machine learning helps you to finds all kind of unknown patterns in data.
- Clustering and Association are two types of Unsupervised learning.
- Four types of clustering methods are 1) Exclusive 2) Agglomerative 3) Overlapping 4) Probabilistic.
- Important clustering types are: 1)Hierarchical clustering 2) K-means clustering 3) K-NN 4) Principal Component Analysis 5) Singular Value Decomposition 6) Independent Component Analysis.
- Association rules allow you to establish associations amongst data objects inside large databases.
- In Supervised learning, Algorithms are trained using labelled data while in Unsupervised learning Algorithms are used against data which is not labelled.
- Anomaly detection can discover important data points in your dataset which is useful for finding fraudulent transactions.
- The biggest drawback of Unsupervised learning is that you cannot get precise information regarding data sorting.

2. What is clustering?

Let's suppose we give a child different objects to group. How does a child make a group? The child may group over the colour, over the shape, over the hardness or softness of the objects etc. The basic idea here is that the child tries to find out similarities and dissimilarities between different objects and then tries to make a group of similar objects. This is called **clustering**, the method of identifying similar instances and keeping them together. In Other words, clustering identifies homogeneous subgroups among the observations.

Clustering is an unsupervised approach which finds a structure/pattern in a collection of unlabeled data. A cluster is a collection of objects which are “similar” amongst themselves and are “dissimilar” to the objects belonging to a different cluster.

3. How do clustering and classification differ?

What is Clustering?

Basically, clustering involves grouping data with respect to their similarities. It is primarily concerned with distance measures and clustering algorithms which calculate the difference between data and divide them systematically.

For instance, students with similar learning styles are grouped together and are taught separately from those with differing learning approaches. In data mining, clustering is most commonly referred to as “unsupervised learning technic” as the grouping is based on a natural or inherent characteristic.

It is applied in several scientific fields such as information technology, biology, criminology, and medicine.

Characteristics of Clustering:

No Exact Definition

Clustering has no precise definition that is why there are various clustering algorithms or cluster models. Roughly speaking, the two kinds of clustering are hard and soft. Hard clustering is concerned with labeling an object as simply belonging to a cluster or not. In contrast, soft clustering or fuzzy clustering specifies the degree as to how something belongs to a certain group.

Difficult to be Evaluated

The validation or assessment of results from clustering analysis are often difficult to ascertain due to its inherent inexactness.

Unsupervised

As it is an unsupervised learning strategy, the analysis is merely based on current features; thus, no stringent regulation is needed.

What is Classification?

Classification entails assigning labels to existing situations or classes; hence, the term “classification”. For example, students exhibiting certain learning characteristics are classified as visual learners.

Classification is also known as “supervised learning technic” wherein machines learn from already labeled or classified data. It is highly applicable in pattern recognition, statistics, and biometrics.

Characteristics of Classification

Utilizes a “Classifier”

To analyze data, a classifier is a defined algorithm that concretely maps an information to a specific class. For example, a classification algorithm would train a model to identify whether a certain cell is malignant or benign.

Evaluated Through Common Metrics

The quality of a classification analysis is often assessed via precision and recall which are popular metric procedures. A classifier is evaluated regarding its accuracy and sensitivity in identifying the output.

Supervised

Classification is a supervised learning technic as it assigns previously determined identities based on comparable features. It deduces a function from a labeled training set.

Differences between Clustering and Classification

Supervision

The main difference is that clustering is unsupervised and is considered as “self-learning” whereas classification is supervised as it depends on predefined labels.

Use of Training Set

Clustering does not poignantly employ training sets, which are groups of instances employed to generate the groupings, while classification imperatively needs training sets to identify similar features.

Labeling

Clustering works with unlabeled data as it does not need training. On the other hand, classification deals with both unlabeled and labeled data in its processes.

Goal

Clustering groups objects with the aim to narrow down relations as well as learn novel information from hidden patterns while classification seeks to determine which explicit group a certain object belongs to.

Specifics

While classification does not specify what needs to be learned, clustering specifies the required improvement as it points out the differences by considering the similarities between data.

Phases

Generally, clustering only consists of a single phase (grouping) while classification has two stages, training (model learns from training data set) and testing (target class is predicted).

Boundary Conditions

Determining the boundary conditions is highly important in the classification process as compared to clustering. For instance, knowing the percentage range of “low” as compared to “moderate” and “high” is needed in establishing the classification.

Prediction

As compared to clustering, classification is more involved with prediction as it particularly aims to identity target classes. For instance, this may be applied in “facial key points detection” as it can be used in predicting whether a certain witness is lying or not.

Complexity

Since classification consists of more stages, deals with prediction, and involves degrees or levels, its’ nature is more complicated as compared to clustering which is mainly concerned with grouping similar attributes.

Number of Probable Algorithms

Clustering algorithms are mainly linear and nonlinear while classification consists of more algorithmic tools such as linear classifiers, neural networks, Kernel estimation, decision trees, and support vector machines.

Clustering	*Classification*
Unsupervised data	Supervised data
Does not highly value training sets	Does highly value training sets
Works solely with unlabeled data	Involves both unlabeled and labeled data
Aims to identify similarities among data	Aims to verify where a datum belongs to
Specifies required change	Does not specify required improvement
Has a single phase	Has two phases
Determining boundary conditions is not paramount	Identifying the boundary conditions is essential in executing the phases
Does not generally deal with prediction	Deals with prediction
Mainly employs two algorithms	Has a number of probable algorithms to use
Process is less complex	Process is more complex

4. What are the various applications of clustering?

What is Clustering? Clustering is the process of making a group of abstract objects into classes of similar objects.

Points to Remember

A cluster of data objects can be treated as one group.

While doing cluster analysis, we first partition the set of data into groups based on data similarity and then assign the labels to the groups.

The main advantage of clustering over classification is that, it is adaptable to changes and helps single out useful features that distinguish different groups.

Applications of Cluster Analysis Clustering analysis is broadly used in many applications such as market research, pattern recognition, data analysis, and image processing.

Clustering can also help marketers discover distinct groups in their customer base. And they can characterize their customer groups based on the purchasing patterns.

In the field of biology, it can be used to derive plant and animal taxonomies, categorize genes with similar functionalities and gain insight into structures inherent to populations.

Clustering also helps in identification of areas of similar land use in an earth observation database. It also helps in the identification of groups of houses in a city according to house type, value, and geographic location.

Clustering also helps in classifying documents on the web for information discovery.

Clustering is also used in outlier detection applications such as detection of credit card fraud.

As a data mining function, cluster analysis serves as a tool to gain insight into the distribution of data to observe characteristics of each cluster.

Requirements of Clustering in Data Mining The following points throw light on why clustering is required in data mining −

Scalability − We need highly scalable clustering algorithms to deal with large databases.

Ability to deal with different kinds of attributes − Algorithms should be capable to be applied on any kind of data such as interval-based (numerical) data, categorical, and binary data.

Discovery of clusters with attribute shape − The clustering algorithm should be capable of detecting clusters of arbitrary shape. They should not be bounded to only distance measures that tend to find spherical cluster of small sizes.

High dimensionality − The clustering algorithm should not only be able to handle low-dimensional data but also the high dimensional space.

Ability to deal with noisy data − Databases contain noisy, missing or erroneous data. Some algorithms are sensitive to such data and may lead to poor quality clusters.

Interpretability − The clustering results should be interpretable, comprehensible, and usable.

Applications of Clustering in different fields

Marketing : It can be used to characterize & discover customer segments for marketing purposes. Biology : It can be used for classification among different species of plants and animals. Libraries : It is used in clustering different books on the basis of topics and information. Insurance : It is used to acknowledge the customers, their policies and identifying the frauds. City Planning: It is used to make groups of houses and to study their values based on their geographical locations and other factors present. Earthquake studies: By learning the earthquake-affected areas we can determine the dangerous zones.

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.

Why Clustering ? Clustering is very much important as it determines the intrinsic grouping among the unlabeled data present. There are no criteria for a good clustering. It depends on the user, what is the criteria they may use which satisfy their need. For instance, we could be interested in finding representatives for homogeneous groups (data reduction), in finding “natural clusters” and describe their unknown properties (“natural” data types), in finding useful and suitable groupings (“useful” data classes) or in finding unusual data objects (outlier detection). This algorithm must make some assumptions which constitute the similarity of points and each assumption make different and equally valid clusters.

==========

Here are 7 examples of clustering algorithms in action.

1. Identifying Fake News Fake news is not a new phenomenon, but it is one that is becoming prolific.

What the problem is: Fake news is being created and spread at a rapid rate due to technology innovations such as social media. The issue gained attention recently during the 2016 US presidential campaign. During this campaign, the term Fake News was referenced an unprecedented number of times.

How clustering works: In a paper recently published by two computer science students at the University of California, Riverside, they are using clustering algorithms to identify fake news based on the content.

The way that the algorithm works is by taking in the content of the fake news article, the corpus, examining the words used and then clustering them. These clusters are what helps the algorithm determine which pieces are genuine and which are fake news. Certain words are found more commonly in sensationalized, click-bait articles. When you see a high percentage of specific terms in an article, it gives a higher probability of the material being fake news.

2. Spam filter You know the junk folder in your email inbox? It is the place where emails that have been identified as spam by the algorithm.

Many machine learning courses, such as Andrew Ng’s famed Coursera course, use the spam filter as an example of unsupervised learning and clustering.

What the problem is: Spam emails are at best an annoying part of modern day marketing techniques, and at worst, an example of people phishing for your personal data. To avoid getting these emails in your main inbox, email companies use algorithms. The purpose of these algorithms is to flag an email as spam correctly or not.

How clustering works: K-Means clustering techniques have proven to be an effective way of identifying spam. The way that it works is by looking at the different sections of the email (header, sender, and content). The data is then grouped together. These groups can then be classified to

identify which are spam. Including clustering in the classification process improves the accuracy of the filter to 97%. This is excellent news for people who want to be sure they're not missing out on your favorite newsletters and offers.

3. Marketing and Sales Personalization and targeting in marketing is big business.

This is achieved by looking at specific characteristics of a person and sharing campaigns with them that have been successful with other similar people.

What the problem is: If you are a business trying to get the best return on your marketing investment, it is crucial that you target people in the right way. If you get it wrong, you risk not making any sales, or worse, damaging your Customer trust.

How clustering works: Clustering algorithms are able to group together people with similar traits and likelihood to purchase. Once you have the groups, you can run tests on each group with different marketing copy that will help you better target your messaging to them in the future.

4. Classifying network traffic Imagine you want to understand the different types of traffic coming to your website. You are particularly interested in understanding which traffic is spam or coming from bots.

What the problem is: As more and more services begin to use APIs on your application, or as your website grows, it is important you know where the traffic is coming from. For example, you want to be able to block harmful traffic and double down on areas driving growth. However, it is hard to know which is which when it comes to classifying the traffic.

How clustering works: K-means clustering is used to group together characteristics of the traffic sources. When the clusters are created, you can then classify the traffic types. The process is faster and more accurate than the previous Autoclass method. By having precise information on traffic sources, you are able to grow your site and plan capacity effectively.

5. Identifying fraudulent or criminal activity In this scenario, we are going to focus on fraudulent taxi driver behavior. However, the technique has been used in multiple scenarios.

What is the problem: You need to look into fraudulent driving activity. The challenge is how do you identify what is true and which is false?

How clustering works: By analysing the GPS logs, the algorithm is able to group similar behaviors. Based on the characteristics of the groups you are then able to classify them into those that are real and which are fraudulent.

6. Document analysis There are many different reasons why you would want to run an analysis on a document. In this scenario, you want to be able to organize the documents quickly and efficiently.

What the problem is: Imagine you are limited in time and need to organize information held in documents quickly. To be able to complete this ask you need to: understand the theme of the text, compare it with other documents and classify it.

How clustering works: Hierarchical clustering has been used to solve this problem. The algorithm is able to look at the text and group it into different themes. Using this technique, you can cluster and organize similar documents quickly using the characteristics identified in the paragraph.

7. Fantasy Football and Sports Ok so up until this point we have looked into different business problems and how clustering algorithms have been applied to solve them.

   But now for the critical issues - fantasy football!

What is the problem: Who should you have in your team? Which players are going to perform best for your team and allow you to beat the competition? The challenge at the start of the season is that there is very little if any data available to help you identify the winning players.

How clustering works: When there is little performance data available to train your model on, you have an advantage for unsupervised learning. In this type of machine learning problem, you can find similar players using some of their characteristics. This has been done using K-Means clustering. Ultimately this means you can get a better team more quickly at the start of the year, giving you an advantage.

How will you use clustering algorithms? So there you have it, those were 7 innovative uses of clustering algorithms. As you can see, while the technique remains reasonably constant, you can apply it to many different scenarios.

Looking at the characteristics of different groups of data can help you make better predictions of behavior. In this scenario, the real value of the algorithms is to help you create the best possible groups of data.

Once you have a solid foundation of grouped data to work with, the opportunities become infinite.

5. How does clustering play a role in supervised learning?

Clustering tries to, well, cluster data in some space. The notion of what a cluster (like a group) is, is usually related to the notion of proximity: things that are closer to each other should be considered as belonging to the same cluster.

The interesting thing is that the very notion of proximity is highly dependent on the space in which proximity is computed and how that proximity is defined: are we comparing probability distributions? Should we use wasserstein distance? Are we comparing distributions to points? Should we use Mahalanobis distance? Are we comparing points in euclidian space? Does the L-p norm make sense? For which p?

All of these decisions we have to make, as to define what being “close” or “far” means, along with how we want to cluster (number of clusters, shape of clusters, the algorithms with which to find them), will lead us to different type of clusters.

Note that, in all I've said, I've never mentioned labels. Given the features (the space in which they “live”), a notion of proximity, and the algorithm, we can get to clusters, without using any labels as to which points should be grouped with which. Therefore, clustering is an unsupervised algorithm.

Clustering, however, requires us to define a bunch of hyperparameters, which I would call, maybe unofficially, some sort of supervision, as you are most likely injecting your knowledge into the pipeline, to get clusters that make sense.

Clustering algorithms aim to group data point by 1,2 or more features of the data (age and weight for example). Not only that but they tend to group with the help of a predefined set of functions (distance, density and so on).

They are unsupervised in the sense that you don't need pre-labeled data to use them. So this algorithm doesn't use a label that would be used in let's say a random forest classification algorithm. It gives you a score or a label.

If we go with the age/weight features, for a classification model to work you'd need a label of what you are trying to classify along with the featires, which is a very specific task because the aim is to approximate a function (accurately).

Clustering groups your data based on the features you supply it with and the predetermined function that the algorithm works with (distance, density etc.).

That being said, there are classification algorithms based on clustering (at least in python there is the KMeans Classifier where you can specify with function you want it to try to use to group and classify your data.

==========
Clustering is obviously an UNSUPERVISED task. Sometimes, It is also used to perform SEMI-SUPERVISED learning. but, Clustering is still unsupervised for his role, in there too.

IN clustering, what we do is; group similar looking data points together depending on some properties (similar properties). these groups are called clusters. We are not aware of our targets classes, when we do clustering. In fact, we assign classes after observing the “similar” features of the clusters. So, this is quite evident as an UNSUPERVISED task.

There are different approaches and factors of clustering, thus it can also be further classified. Following are the different types of clusters.

Based on Goals: Monothetic Polythetic

Based on overlaps: hard clustering soft clustering

Flat v/s Hierarchical: Flat

hierarchical :

Aglomerative Devisive

6. What are the requirements to be met by a clustering algorithm?

The primary requirements that should be met by a clustering algorithm are:

- It should be scalable

- It should be able to deal with attributes of different types;
- It should be able to discover arbitrary shape clusters;
- It should have an inbuilt ability to deal with noise and outliers;
- The clusters should not vary with the order of input records;
- It should be able to handle data of high dimensions.
- It should be easy to interpret and use.

7. Discuss the different approaches for clustering.

Approaches for Clustering:

The clustering approaches can be broadly divided into two categories: *Agglomerative* and *Divisive*.

Agglomerative: This approach first considers all the points as individual clusters and then finds out the similarity between two points, puts them into a cluster. Then it goes on finding similar points and clusters until there is only one cluster left i.e., all points belong to a big cluster. This is also called the bottom-up approach.

Divisive: It is opposite of the agglomerative approach. It first considers all the points to be part of one big cluster and in the subsequent steps tries to find out the points/ clusters which are least similar to each other and then breaks the bigger cluster into smaller ones. This continues until there are as many clusters as there are datapoints. This is also called the top-down approach.

8. What is WCSS?

THe objective function in k-Means measures sum of distances of observations from their cluster centroids, called Within-Cluster-Sum-of-Squares (WCSS). This is computed as

K-means is all about the analysis-of-variance paradigm. ANOVA - both uni- and multivariate - is based on the fact that the sum of squared deviations about the grand centroid is comprised of such scatter about the group centroids and the scatter of those centroids about the grand one: SStotal=SSwithin+SSbetween. So, if SSwithin is minimized then SSbetween is maximized.

SS of deviations of some points about their centroid (arithmetic mean) is known to be directly related to the overall squared euclidean distance between the points: the sum of squared deviations from centroid is equal to the sum of pairwise squared Euclidean distances divided by the number of points. (This is the direct extension of the trigonometric property of centroid. And this relation is exploited also in the double centering of distance matrix.)

Thus, saying "SSbetween for centroids (as points) is maximized" is alias to say "the (weighted) set of squared distances between the centroids is maximized".

Note: in SSbetween each centroid is weighted by the number of points Ni in that cluster i. That is, each centroid is counted Ni times. For example, with two centroids in the data, 1 and 2, SSbetween = N1*D1^2+N2*D2^2 where D1 and D2 are the deviations of the centroids from the

grand mean. That where word "weighted" in the former paragraph stems from.

Within Cluster Sum of Squares One measurement is Within Cluster Sum of Squares (WCSS), which measures the squared average distance of all the points within a cluster to the cluster centroid. To calculate WCSS, you first find the Euclidean distance (see figure below) between a given point and the centroid to which it is assigned. You then iterate this process for all points in the cluster, and then sum the values for the cluster and divide by the number of points.

9. Discuss the elbow method.

Means Clustering is an unsupervised machine learning algorithm which basically means we will just have input, not the corresponding output label. In this article, we will see it’s implementation using python.

K Means Clustering tries to cluster your data into clusters based on their similarity. In this algorithm, we have to specify the number of clusters (which is a hyperparameter) we want the data to be grouped into. Hyperparameters are the variables whose value need to be set before applying value to the dataset. Hyperparameters are adjustable parameters you choose to train a model that carries out the training process itself.

The K Means Algorithm is:

Choose a number of clusters “K” Randomly assign each point to Cluster Until cluster stop changing, repeat the following 1.For each cluster, compute the centroid of the cluster by taking the mean vector of the points in the cluster. 2.Assign each data point to the cluster for which the centroid is closest

Finding a K-value

There is no easy answer for choosing k value.One of the method is known as elbow method.First of all compute the sum of squared error(SSE) for some value of K.SSE is defined as the sum of the squared distance between centroid and each member of the cluster. Then plot a K against SSE graph.We will observe that as K increases SSE decreases as disortation will be small. So the idea of this algorithm is to choose the value of K at which the graph decrease abruptly.This sort of produces a “elbow effect” in the picture.

10. What is the significance of ‘K’ in K-Means and how is it calculated?

There is a popular method known as elbow method which is used to determine the optimal value of K to perform the K-Means Clustering Algorithm. The basic idea behind this method is that it plots the various values of cost with changing k. As the value of K increases, there will be fewer elements in the cluster. So average distortion will decrease. The lesser number of elements means closer to the centroid. So, the point where this distortion declines the most is the elbow point.

11. Discuss the step by step implementation of K-Means

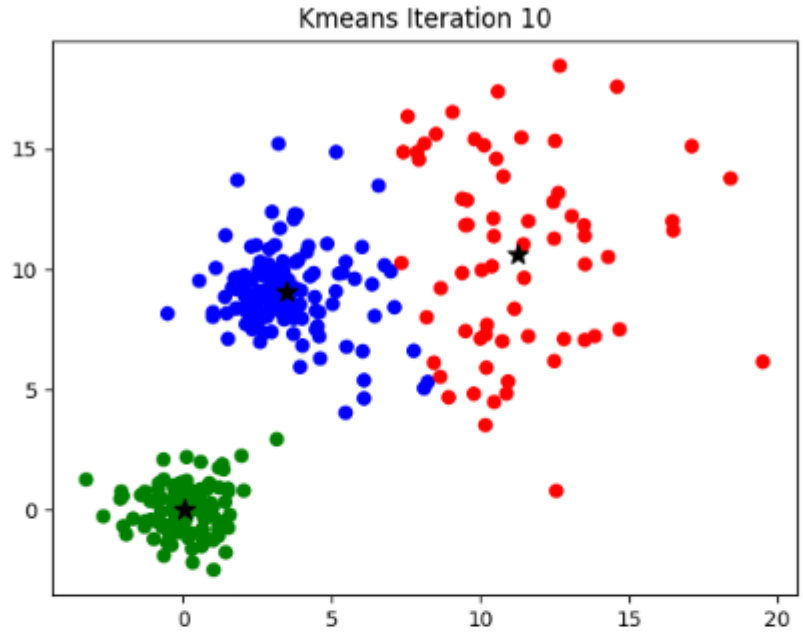
11. Discuss the step by step implementation of K-Means Clustering.

The Algorithm

K-means clustering is a good place to start exploring an unlabeled dataset. The K in K-Means denotes the number of clusters. This algorithm is bound to converge to a solution after some iterations. It has 4 basic steps:

1. Initialize Cluster Centroids (Choose those 3 books to start with)
2. Assign datapoints to Clusters (Place remaining the books one by one)
3. Update Cluster centroids (Start over with 3 different books)
4. Repeat step 2–3 until the stopping condition is met.

You don't have to start with 3 clusters initially, but 2–3 is generally a good place to start, and update later on.



Clustering with K=3

1. Initialize K & Centroids

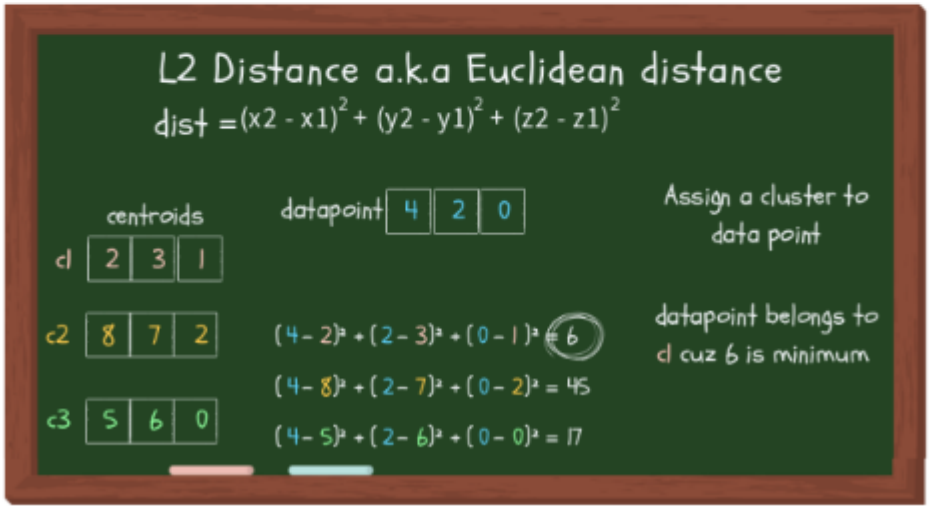
As a starting point, you tell your model how many clusters it should make. First the model picks up K, (let K = 3) datapoints from the dataset. These datapoints are called cluster centroids.

Now there are different ways you to initialize the centroids, you can either choose them at random — or sort the dataset, split it into K portions and pick one datapoint from each portion as a centriod.

2. Assigning Clusters to datapoints

From here on wards, the model performs calculations on it's own and assigns a cluster to each datapoint. Your model would calculate the distance between the datapoint & all the centroids, and will be assigned to the cluster with the nearest centroid. Again, there are different ways you can calculate this distance; all having their pros and cons. Usually we use the L2 distance.

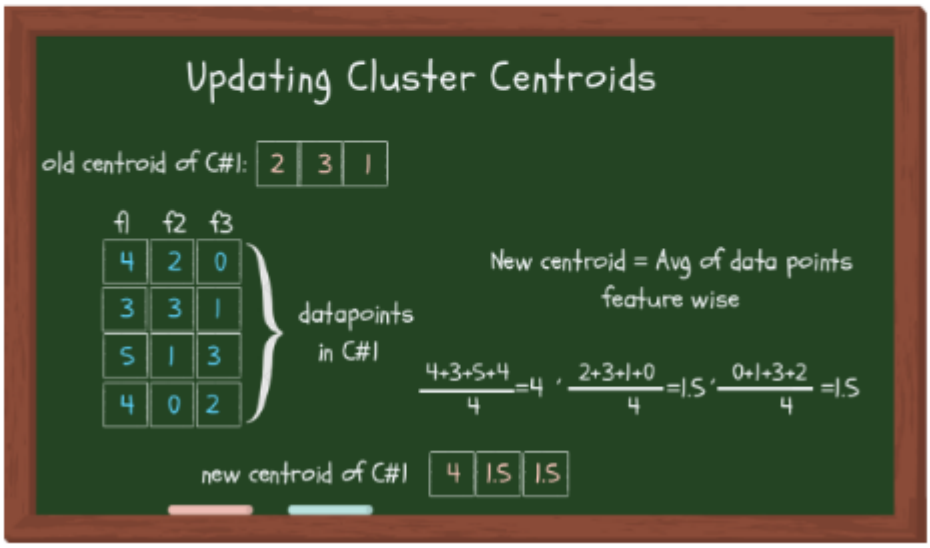
The picture below shows how to calculate the L2 distance between the centeroid and a datapoint. Every time a datapoint is assigned to a cluster the following steps are followed.



L2 or Euclidean distance

3. Updating Centroids

Because the initial centroids were chosen arbitrarily, your model the updates them with new cluster values. The new value might or might not occur in the dataset, in fact, it would be a coincidence if it does. This is because the updated cluster centorid is the average or the mean value of all the datapoints within that cluster.



Updating cluster centroids

Now if some other algo, like K-Mode, or K-Median was used, instead of taking the average value, mode and median would be taken respectively.

4. Stopping Criterion

Since step 2 and 3 would be performed iteratively, it would go on forever if we don't set a stopping criterion. The stopping criterion tells our algo when to stop updating the clusters. It is important to note that setting a stopping criterion would not necessarily return THE BEST clusters, but to make sure it returns reasonably good clusters, and more importantly at least **return** some clusters, we **need** to have a stopping criterion.

Like everything else, there are different ways to set the stopping criterion. You can even set multiple conditions that, if met, would stop the iteration and return the results. Some of the stopping conditions are:

1. The datapoints assigned to specific cluster remain the same (takes too much time)
2. Centroids remain the same (time consuming)
3. The distance of datapoints from their centroid is minimum (the thresh you've set)
4. Fixed number of iterations have reached (insufficient iterations → poor results, choose max iteration wisely)

Evaluating the cluster quality

The goal here isn't just to make clusters, but to make good, meaningful clusters. Quality clustering is when the datapoints within a cluster are close together, and afar from other clusters.

The two methods to measure the cluster quality are described below:

1. **Inertia:** Intuitively, inertia tells how far away the points within a cluster are. Therefore, a small of inertia is aimed for. The range of inertia's value starts from zero and goes up.
2. **Silhouette score:** Silhouette score tells how far away the datapoints in one cluster are, from the datapoints in another cluster. The range of silhouette score is from -1 to 1. Score should be closer to 1 than -1.

. . .

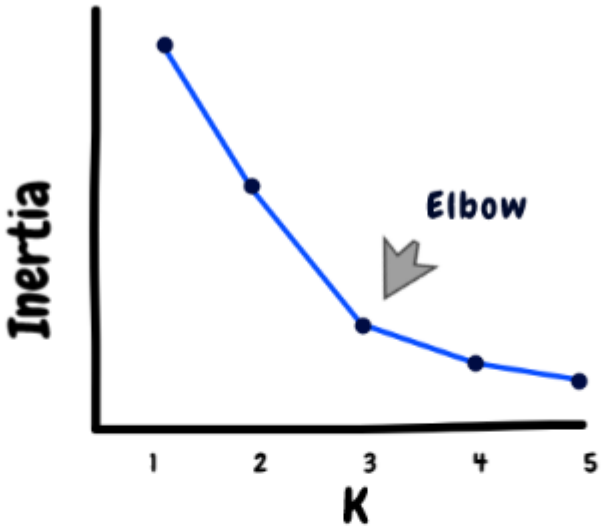
How many clusters?

You have to specify the number of clusters you want to make. There are a few methods available to choose the optimal number of K. The direct method is to just plot the datapoints and see if it gives you a hint. As you can see in the figure below, making 3 clusters seems like a good choice.

K=3 seems like a good choice

Other method is to use the value of inertia. The idea behind good clustering is having a small value of inertia, and small number of clusters.

The value of inertia decreases as the number of clusters increase. So, its a trade-off here. Rule of thumb: The elbow point in the inertia graph is a good choice because after that the change in the value of inertia isn't significant.



K=3 is the optimal choice

. . .

Naming your Clusters

When you've formed a cluster, you give a name it, and all the datapoints in that cluster are assigned this name as their label. Now your dataset has labels! You can perform testing using these labels. To find insights about your data, you can see what similarity do the datapoints within a cluster have, and how does it differ from other clusters.

Assigning a cluster to a new data point

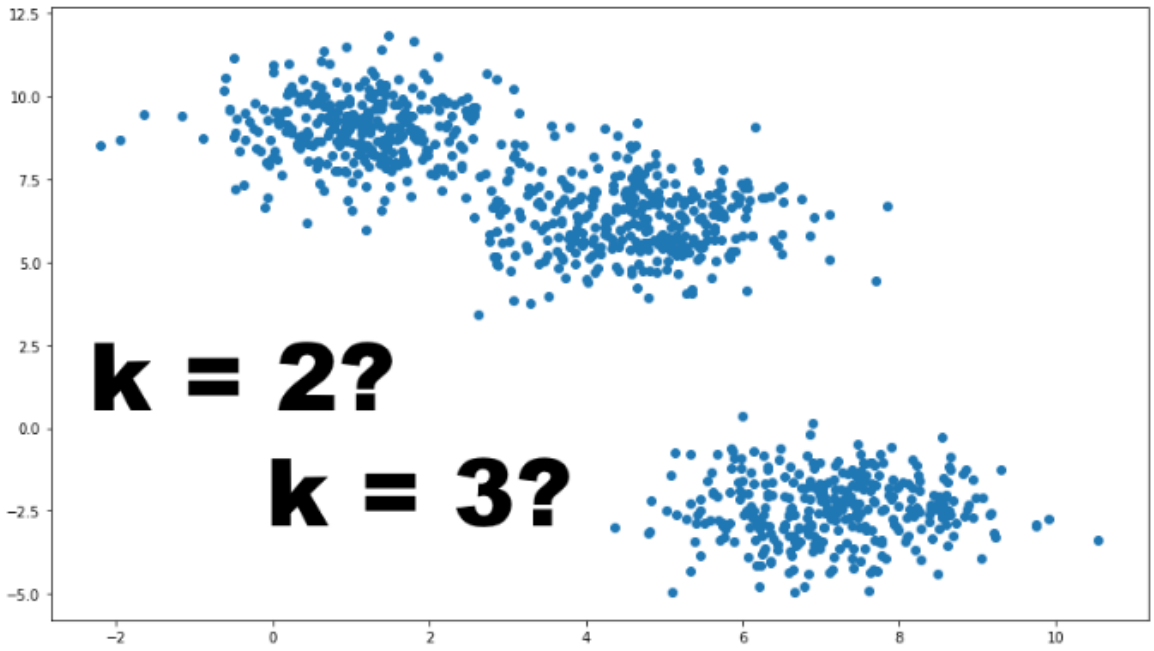
Once you've finalized your model, it can now assign a cluster to a new data point. The method of assigning cluster remains same, i.e., assigning it to the cluster with the closet centroid.

. . .

⚠ Warning!

It's important to preprocess your data before performing K-Means. You would have to convert your dataset into numerical values if it is not already, so that calculations can be performed. Also, applying feature reduction techniques would speed up the process, and also improve the results. These steps are important to follow because K-Means is sensitive to outliers, just like every other algo that uses average/mean values. Following these steps alleviate these issues.

How to Determine the Optimal K for K-Means?



Introduction

The K-Means algorithm needs no introduction. It is simple and perhaps the most commonly used algorithm for clustering.

The basic idea behind k-means consists of defining k clusters such that total **within-cluster variation (or error) is minimum**.

I encourage you to check out the below articles for an in-depth explanation of different methods of clustering before proceeding further:

- *An Introduction to Clustering and different methods of Clustering*
- *A Beginner's Guide to Hierarchical Clustering and how to perform it in Python*

A cluster center is the representative of its cluster. The squared distance between each point and its cluster center is the required variation. The aim of k-means clustering is to find these k clusters and their centers while reducing the total error.

Quite an elegant algorithm. But there is a catch. How do you decide the number of clusters?

In this article, I will explain in detail two methods that can be useful to find this mysterious k in k-Means.

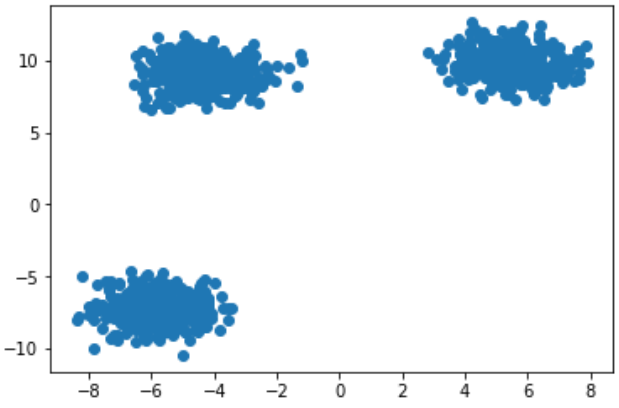
These methods are:

1. **The Elbow Method**
2. **The Silhouette Method**

We will use our own dataset generated by the code below for an illustration of the two methods:

This is how the data looks graphically:

This is how the data looks graphically:



Clearly, the dataset has 3 clusters. We will validate both our methods on this dataset.

The Elbow Method

This is probably the most well-known method for determining the optimal number of clusters. *It is also a bit naive in its approach.*

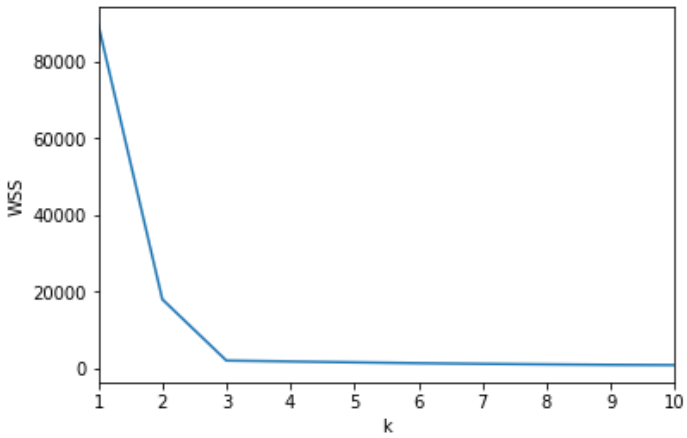
> *Calculate the **Within-Cluster-Sum of Squared** Errors (WSS) for **different values of k**, and choose the k for which WSS becomes first starts to diminish. In the plot of WSS-versus-k, this is visible as an **elbow.***

Within-Cluster-Sum of Squared Errors sounds a bit complex. Let's break it down:

- The Squared Error for each point is the square of the distance of the point from its representation i.e. its predicted cluster center.
- The WSS score is the sum of these Squared Errors for all the points.
- Any distance metric like the Euclidean Distance or the Manhattan Distance can be used.

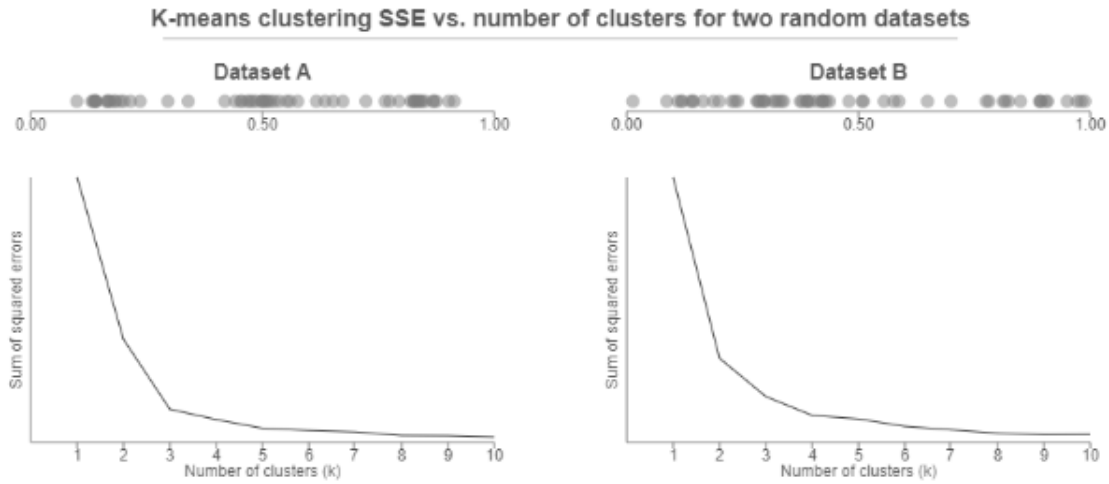
Let us implement this in Python using the ***sklearn*** library and our own function for calculating WSS for a range of values for k.

We obtain the following plot for WSS-vs-k for our dataset.



As expected, the **plot looks like an arm with a clear elbow at k = 3.**

Unfortunately, we do not always have such clearly clustered data. This means that the elbow may not be clear and sharp.



Source: bl.ocks.org/

For Dataset A, the elbow is clear at k = 3. However, this choice is ambiguous for Dataset B. We could choose k to be either 3 or 4.

In such an ambiguous case, we may use the Silhouette Method.

The Silhouette Method

> *The silhouette value measures how similar a point is to its own cluster (cohesion) compared to other clusters (separation).*
>
> *Source: Wikipedia*

The range of the Silhouette value is between +1 and -1. A **high value is desirable** and indicates that the point is placed in the correct cluster. If many points have a negative Silhouette value, it may indicate that we have created too many or too few clusters.

The Silhouette Value ***s(i)*** for each data point ***i*** is defined as follows:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \text{ if } |C_i| > 1$$

and

$$s(i) = 0, \text{ if } |C_i| = 1$$

Source: Wikipedia

Note: s(i) is defined to be equal to zero if ***i*** is the only point in the cluster. This is to prevent the number of clusters from increasing significantly with many single-point clusters.

Here, ***a(i)*** is the measure of similarity of the point ***i*** to its own cluster. It is measured as the average distance of ***i*** from other points in the cluster.

For each data point $i \in C_i$ (data point i in the cluster C_i), let

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i, j)$$

Source: Wikipedia

Similarly, ***b(i)*** is the measure of dissimilarity of ***i*** from points in other clusters.

For each data point $i \in C_i$, we now define

$$b(i) = \min_{i \neq j} \frac{1}{|C_j|} \sum_{j \in C_j} d(i, j)$$

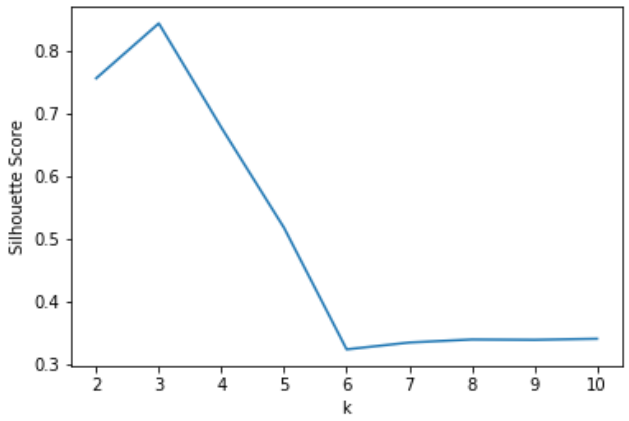
Source: Wikipedia

d(i, j) is the distance between points ***i*** and ***j***. Generally, **Euclidean Distance** is used as the distance metric.

The Silhouette score can be easily calculated in Python using the metrics module of the ***sklearn*** library.

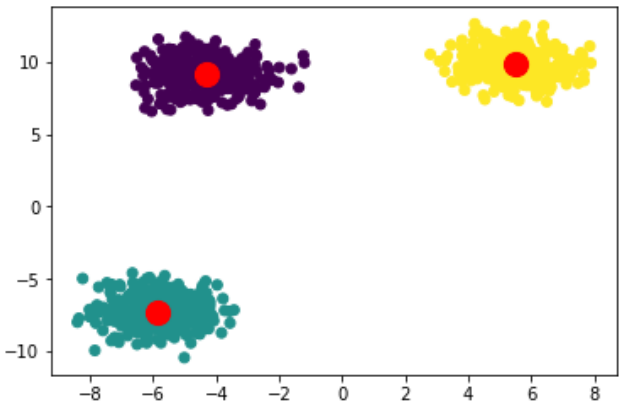
I mentioned before that a high Silhouette Score is desirable. The Silhouette Score reaches its ***global maximum at the optimal k***. This should ideally appear as a peak in the Silhouette Value-versus-k plot.

Here is the plot for our own dataset:



There is a clear peak at k = 3. Hence, it is optimal.

Finally, the data can be ***optimally*** clustered into 3 clusters as shown below.



End Notes

The Elbow Method is more of a decision rule, while the Silhouette is a metric used for validation while clustering. Thus, it can be used in combination with the Elbow Method.

Therefore, the Elbow Method and the Silhouette Method are not alternatives to each other for finding the optimal K. Rather they are tools to be used together for a more confident decision.

12. What are the challenges with K-Means?

Limitations of k-means are,

no overlap/nesting is possible between/within clusters. Sometimes a data-point could be exactly half-way from two adjacent clusters, when either you can assign that to either of those clusters or you can include data as to the probability for that point's association with each of those. Also as all the clusters are assumed to be spherical, there could be void space inbetween that has to be handled. This makes the iterative process to introduce more and more smaller spheres and could increase chances of errors when the actual grouping is forming an ellipse. So finally once we have built our ml model,

When new movies are released, we can categorize them to different clusters based on the other movies that are in the cluster. Based on a new customer's personal profile of behavior, we can make a suggestion for him/her.

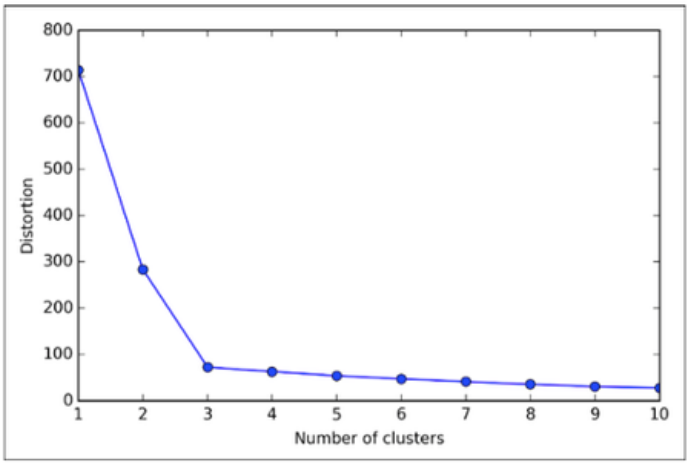
This is just one example. Analogously, this could be used in many consumer goods sales scenarios, where we classify goods by taking few parameters of the people for whom we built the model.

What are some of the issues with Clustering?

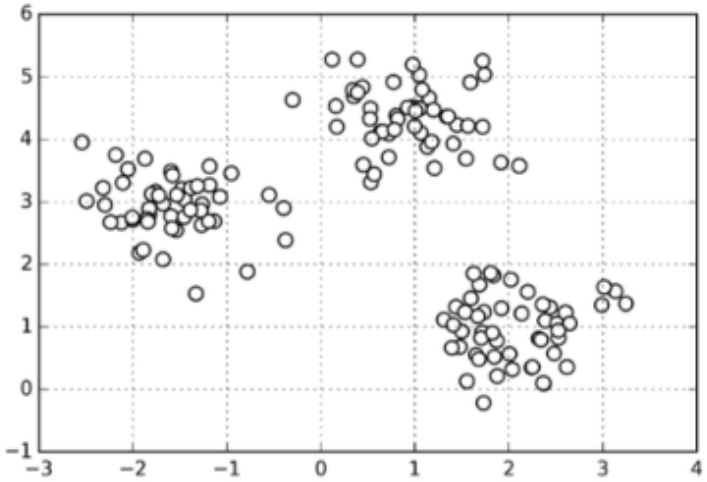
I wouldn't necessarily call most of them "issues" but rather "challenges". For example, *k*-means:

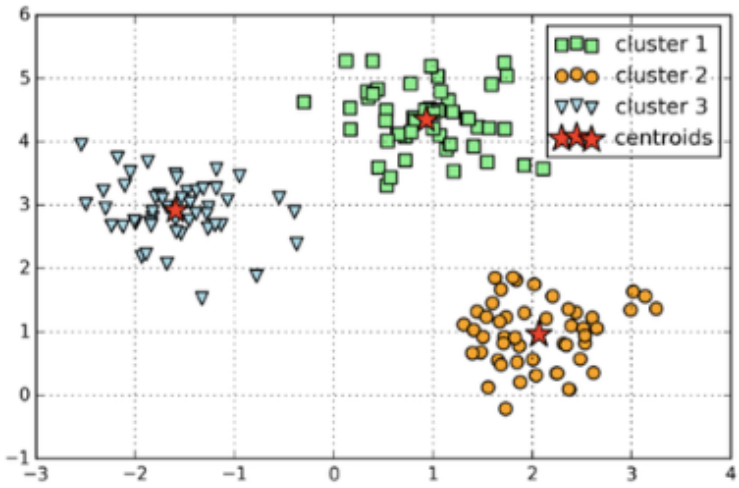
- The different results via *k*-means with distinct random initializations are definitely a problem. However, we could use *k*-means++ as an alternative, and if it's computationally feasible, we want to run your algorithm multiple times with different seeds and pick the one with e.g., lowest within cluster SSE (sum of squared errors)

- The number of clusters is (typically) not known a priori (that's basically the characteristic of unsupervised learning problems), but there are a few "performance" or "evaluation metrics one can use to infer a "satisfying" grouping against the value of K; this is also called the elbow method:



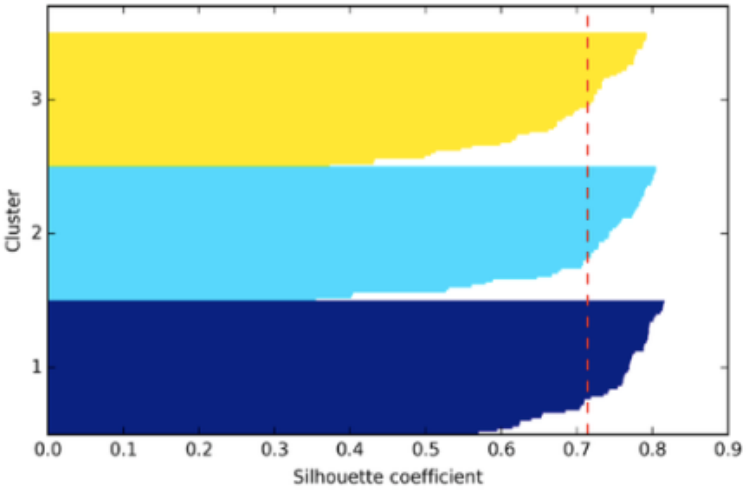
Here, it seems that k=3 would be a good pick. Let's have a look at the accompanying 2D dataset that I used to train the *k*-means algorithm and see if our intuition agrees:

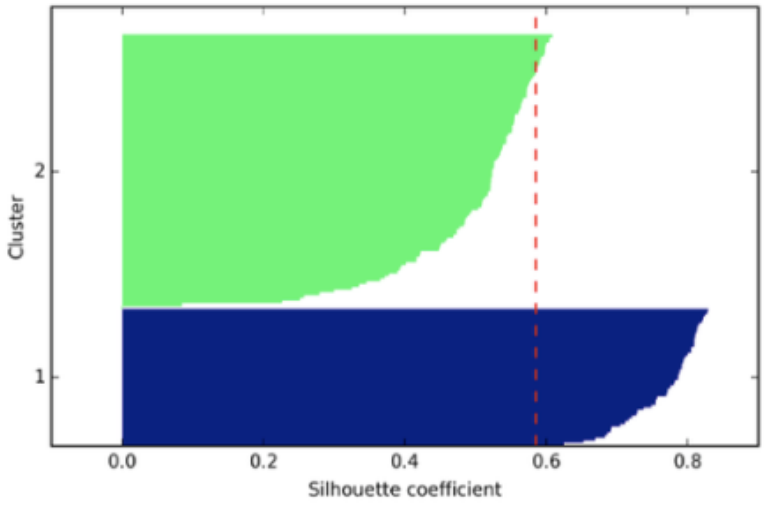




I'd say k=3 is definitely a reasonable pick. However, note that the "elbow" is typically not as clear as shown above. Moreover, note that in practice we normally work with higher-dimensional datasets so that we can't simply plot our data and double-check visually. (We could use unsupervised dimensionality reduction techniques though such as PCA). In fact, if we already knew that the 3 clusters belong to three different groups, this would be a classification task.

Anyway, there are other useful evaluation metrics such as the silhouette coefficient, which gives us some idea of the cluster sizes and shapes. Using the same dataset, let me give you a "good" silhouette plot (with k=3) and a not so decent one (k=2)





I would say that the biggest "shortcoming" in *k*-means may be that we assume that the groups come in spherical or globular shapes, which is rarely the case with "real-world" data. In contrast, I could think of choosing the "optimal" *k* as just another hyperparameter optimization procedure, which is also necessary for almost every supervised learning algorithm.

There are two challenges that need to handled wisely in order to get the most out of the k-means clustering algorithm:

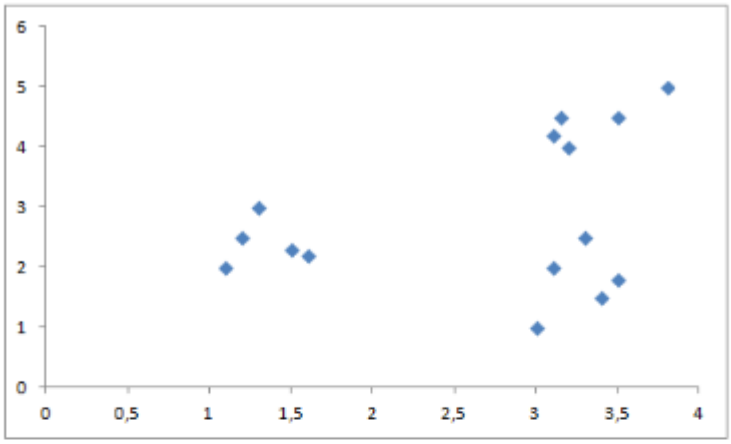
- Defining the number of clusters
- Determining the initial centroids

Defining the number of clusters

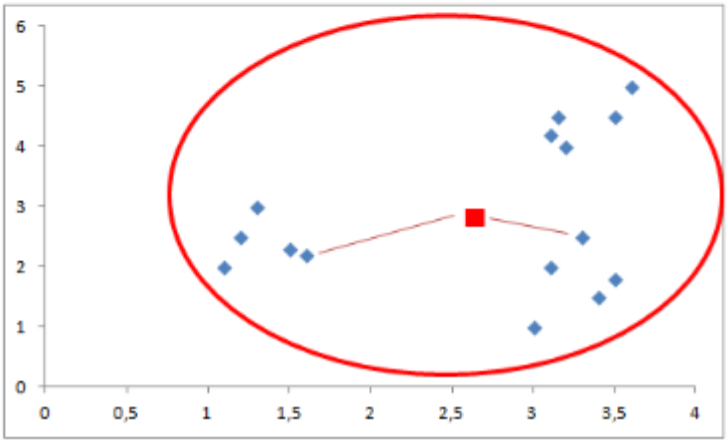
We need to declare the number of clusters before running k-means clustering algorithm. It is not capable of determining the optimum number of clusters. K-means partitions the data set into the number of clusters determined by us beforehand. Finding the optimum number of clusters is also a challenging task for us. We can't just look at the data set and find out how many partitions we should have.

> *K-means clustering tries to minimize distances within clusters. This distances are defined as within cluster sum of distances (WCSS).*

Let's see how WCSS changes with a different number of clusters. Assume we have the following very simple data set:



Let's be non-realistic and have one cluster:

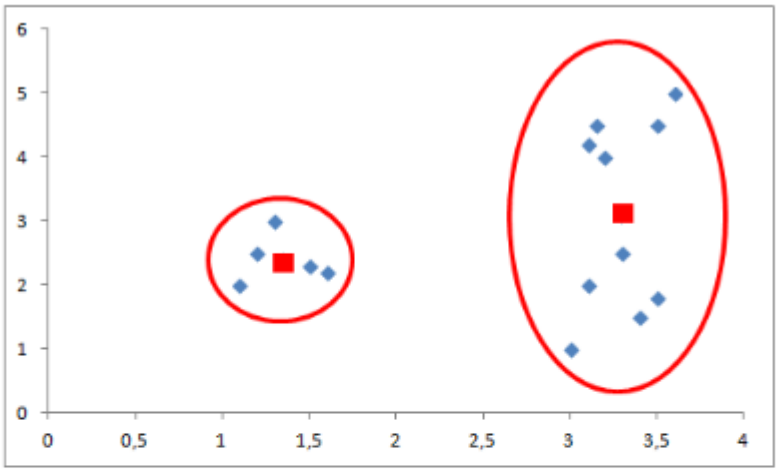


If we have one cluster, the centroid will be the red square. WCSS is calculated as:

$$\sum_{x_i \in c} (x_i - \bar{x})^2$$

The sum of squared distances between each data point and the mean (red square). As the number of clusters increases, the average distance between data points and centroids decreases and thus WCSS decreases.

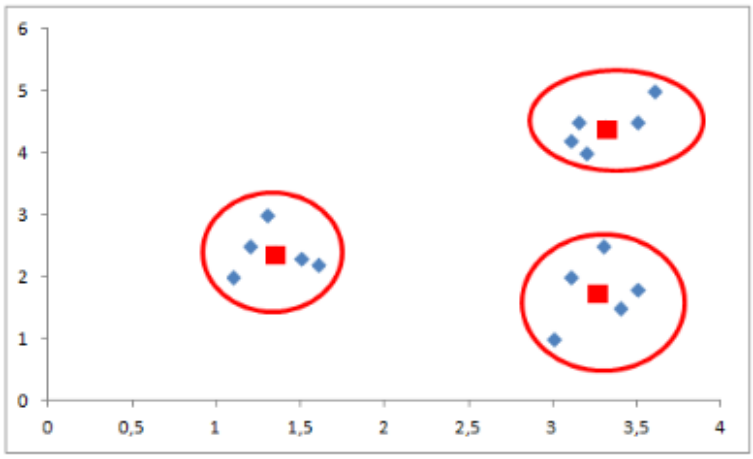
Let's see the case with two clusters:



As you can see the average distance between data points and centroids decreased. Please keep in mind that the number of distances are independent of the number of clusters. No matter how many clusters exist, we will calculate n distances where n is the number of data points. Therefore, we can only focus on the average distance.

WCSS further decreases with 3 clusters:

WCSS further decreases with 3 clusters:

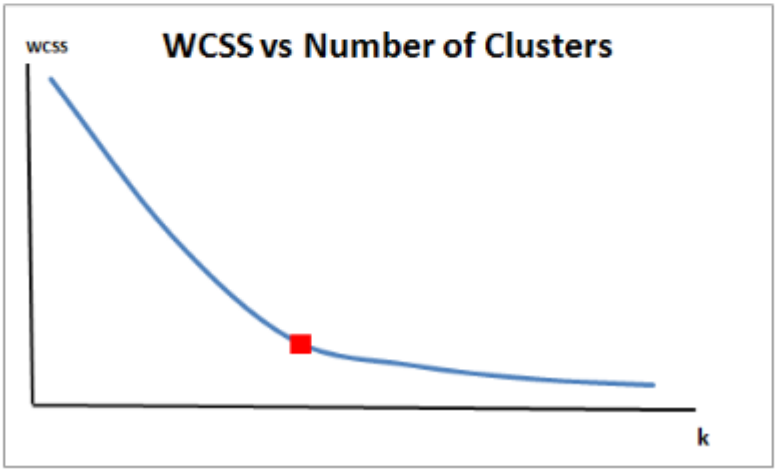


Until have, we have seen WCSS of 1,2 and 3 clusters which are calculated as:

- k=1 > WCSS = 35,16 units
- k=2 > WCSS = 20,91 units
- k=3 > WCSS = 2,95 units

How much further WCSS keeps decreasing? Well, it will eventually be zero. The highest number of clusters we can have is equal to the number of data points. Although not useful, we can have a separate cluster for each data point. Then the distance between a data point and its centroid becomes zero because the centroid of the data point is itself.

So why not have a cluster for each data point and make WCSS zero? Because, in that case, we will have a bigger problem which is overfitting. After some point, we will gain just a little in terms of WCSS by increasing the number of clusters. The graph below represents how WCSS changes as the number of clusters increases.



As you can see, after a certain point, WCSS does not decrease much. We should look for that sharp edge where the slope changed dramatically. That edge marked by red square is our optimum number of clusters.

If we choose k greater than that point, WCSS still decreases but it is not worth the risk of overfitting.

Determining the initial centroids

K-means is an iterative process. It is built on expectation-maximization algorithm. After the number of clusters is determined, it works by executing the following steps:

1. **Randomly select centroids (center of cluster) for each cluster.**
2. Calculate the distance of all data points to the centroids.
3. Assign data points to the closest cluster.
4. Find the new centroids of each cluster by taking the mean of all data points in the cluster.
5. Repeat steps 2,3 and 4 until all points converge and cluster centers stop moving.

We are now focusing on the first step. Depending on the underlying structure of the data set, different initial centroids may end up forming different clusters. In addition to that, selecting the centroids purely random may increase the running time so it takes more for the algorithm to converge.

Instead of selecting the initial centroids in a purely random manner, we may want to look for a smart way. That smart way is **k-means++**.

K-means++ ensures a smarter way to initialize clusters. As stated on wikipedia,

k-means++ *is an algorithm for choosing the initial values (or "seeds") for the k-means clustering algorithm.*

The difference between k-means and k-means++ is only selecting the initial centroids. The remaining steps are exactly the same. K-means++ chooses the first centroid uniformly at random from the data points in the dataset. Each following centroid is chosen from the remaining data points with probability proportional to its squared distance from the point's closest existing centroids.

Fortunately, k-means++ is implemented in scikit-learn. It is specified by "**init**" parameter of sklearn.cluster.KMeans. The default value is k-means++. The other option of init parameter is "random" which initializes centroids randomly.

13. Discuss the various improvements in K-Means.

Highlights • K-means clustering algorithm can be significantly improved by using a better initialization technique, and by repeating (re-starting) the algorithm.

• When the data has overlapping clusters, k-means can improve the results of the initialization technique.

• When the data has well separated clusters, the performance of k-means depends completely on the goodness of the initialization.

• Initialization using simple furthest point heuristic (Maxmin) reduces the clustering error of k-means from 15% to 6%, on average.

==========

-The clustering techniques are the most important part of the data analysis and k-means is the oldest and popular clustering technique used. The paper discusses the traditional K-means algorithm with advantages and disadvantages of it. It also includes researched on enhanced k-means proposed by various authors and it also includes the techniques to improve traditional K-means for better accuracy and efficiency. There are two area of concern for improving K-means;

1) is to select initial centroids and

2) by assigning data points to nearest cluster by using equations for calculating mean and distance between two data points.

The time complexity of the proposed K-means technique will be lesser that then the traditional one with increase in accuracy and efficiency. Keywords--K-means, Centroid based Clustering, enhanced K-means, Clustering, Pro and Cons of K-means

==========

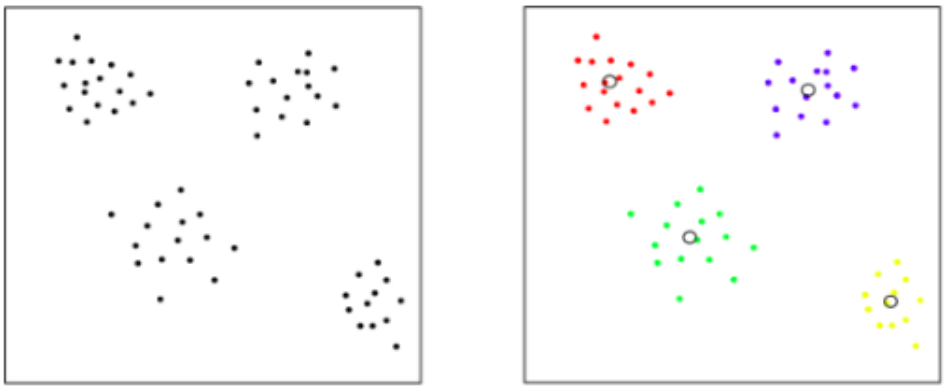
merging neighboring clusters if the resulting cluster's variance is below the threshold isolating elements that are "far" if a cluster's variance is above the threshold or moving some elements between neighboring clusters if it decreases the sum of squared errors (this evolution acts as a global optimization procedure, and prevents the bad consequences of initial assignment of cluster means you have in k-means)

To sum up, if you know the variance, you know how varied the clusters should be, so it's easier to e.g. detect outliers (which usually should be put into separate clusters).

==========

Clustering :

Clustering can be considered the most important unsupervised learning problem; so, as every other problem of this kind, it deals with finding a structure in a collection of unlabeled data. A loose definition of clustering could be “the process of organizing objects into groups whose members are similar in some way”. A cluster is therefore a collection of objects which are “similar” between them and are “dissimilar” to the objects belonging to other clusters. We can show this with a simple graphical example:



In this case we easily identify the 4 clusters into which the data can be divided; the similarity criterion is distance: two or more objects belong to the same cluster if they are “close” according to a given distance (in this case geometrical distance). This is called distance-based clustering. Another kind of clustering is conceptual clustering: two or more objects belong to the same cluster if this one defines a concept common to all that objects. In other words, objects are grouped according to their fit to descriptive concepts, not according to simple similarity measures.

The Goals of Clustering

So, the goal of clustering is to determine the intrinsic grouping in a set of unlabeled data. But how to decide what constitutes a good clustering? It can be shown that there is no absolute "best" criterion which would be independent of the final aim of the clustering. Consequently, it is the user which must supply this criterion, in such a way that the result of the clustering will suit their needs. For instance, we could be interested in finding representatives for homogeneous groups (data reduction), in finding "natural clusters" and describe their unknown properties ("natural" data types), in finding useful and suitable groupings ("useful" data classes) or in finding unusual data objects (outlier detection).

K-Means

K-Means is one of the most popular "clustering" algorithms. K-means stores k centroids that it uses to define clusters. A point is considered to be in a particular cluster if it is closer to that cluster's centroid than any other centroid.
K-Means finds the best centroids by alternating between (1) assigning data points to clusters based on the current centroids (2) choosing centroids (points which are the center of a cluster) based on the current assignment of data points to clusters.

How it works :

Input: k(the number of clusters),
D(a set of lift ratios)
Output: a set of k clusters
Method:
Arbitrarily choose k objects from D as the initial cluster centers;
Repeat:
1.(re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;
2. Update the cluster means, i.e., calculate the mean value of the objects for each cluster
Until no change;

The distance metric used to calculate similarity in step 1 is Euclidean distance

Euclidean distance

Euclidean distance is the most commonly used distance measure. Euclidean distance also called as simply distance. The usage of Euclidean distance measure is highly recommended when data is dense or continuous. Euclidean distance is the best proximity measure. The Euclidean distance between two points is the length of the path connecting them.The Pythagorean theorem gives this distance between two points. A generalized term for the Euclidean norm is the L2 norm or L2 distance.

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

Problem Statement :

To build a simple K-Means model for clustering the car data into different groups.

Data details

==========
1. Title: Car Evaluation
Database==========
The dataset is available at "http://archive.ics.uci.edu/ml/datasets/Car+Evaluation"
2. Sources:
(a) Creator: Marko Bohanec
(b) Donors: Marko Bohanec (marko.bohanec@ijs.si)
Blaz Zupan (blaz.zupan@ijs.si)
(c) Date: June, 19973. Past Usage:

The hierarchical decision model, from which this dataset is derived, was first presented in
M. Bohanec and V. Rajkovic: Knowledge acquisition and explanation for multi-attribute decision making. In 8th Intl Workshop on Expert Systems and their Applications, Avignon, France. pages 59-78, 1988.

Within machine-learning, this dataset was used for the evaluation of HINT (Hierarchy INduction Tool), which was proved to be able to completely reconstruct the original hierarchical model. This, together with a comparison with C4.5, is presented in B. Zupan, M. Bohanec, I. Bratko, J. Demsar: Machine learning by function decomposition. ICML-97, Nashville, TN. 1997 (to appear)

4. Relevant Information Paragraph:

Car Evaluation Database was derived from a simple hierarchical decision model originally developed for the demonstration of DEX(M. Bohanec, V. Rajkovic: Expert system for decision making. Sistemica 1(1), pp. 145-157, 1990.). The model evaluates cars according to the following concept structure:

CAR	car acceptability
. PRICE	overall price
. . buying	buying price
. . maint	price of the maintenance
. TECH	technical characteristics
. . COMFORT	comfort
. . . doors	number of doors
. . . persons	capacity in terms of persons to carry
. . . lug_boot	the size of luggage boot
. . safety	estimated safety of the car

Input attributes are printed in lowercase. Besides the target concept (CAR), the model includes three intermediate concepts: PRICE, TECH, COMFORT. Every concept is in the original model related to its lower level descendants by a set of examples (for these examples sets see http://www-ai.ijs.si/BlazZupan/car.html).

The Car Evaluation Database contains examples with the structural information removed, i.e., directly relates CAR to the six input attributes: buying, maint, doors, persons, lug_boot, safety.
Because of known underlying concept structure, this database may be particularly useful for testing constructive induction and structure discovery methods.

5. Number of Instances: 1728
(instances completely cover the attribute space)

6. Number of Attributes: 6

7. Attribute Values:

buying v-high, high, med, low
maint v-high, high, med, low
doors 2, 3, 4, 5-more
persons 2, 4, more
lug_boot small, med, big
safety low, med, high

8. Missing Attribute Values: none

9. Class Distribution (number of instances per class)

class	N	N[%]
unacc	1210	(70.023 %)
acc	384	(22.222 %)
good	69	(3.993 %)
v-good	65	(3.762 %)

Tools to be used :

Numpy,pandas,scikit-learn

Python Implementation with code :

Import necessary libraries

Import the necessary modules from specific libraries.

```
import os
import numpy as np
import pandas as pd
import numpy as np, pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

```
class     N       N[%]
--------------------
unacc     1210    (70.023 %)
acc       384     (22.222 %)
good      69      ( 3.993 %)
v-good    65      ( 3.762 %)
```

Tools to be used :

Numpy,pandas,scikit-learn

Python Implementation with code :

Import necessary libraries

Import the necessary modules from specific libraries.

```
import os
import numpy as np
import pandas as pd
import numpy as np, pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

Load the data set

Use pandas module to read the bike data from the file system. Check few records of the dataset.

```
data = pd.read_csv('data/car_quality/car.data',names=
['buying','maint','doors','persons','lug_boot','safety','class'])
data.head()
```

	buying	maint	doors	persons	lug_boot	safety	class
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

Check few information about the data set

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
buying      1728 non-null object
maint       1728 non-null object
doors       1728 non-null object
persons     1728 non-null object
lug_boot    1728 non-null object
safety      1728 non-null object
class       1728 non-null object
dtypes: object(7)
memory usage: 94.6+ KB
```

The train data set has 1728 rows and 7 columns.
There are no missing values in the dataset

Identify the target variable

```
data['class'],class_names = pd.factorize(data['class'])
```

The target variable is marked as class in the dataframe. The values are present in string format. However the algorithm requires the variables to be coded into its equivalent integer codes. We can convert the string categorical values into a integer code using factorize method of the pandas library.

Let's check the encoded values now.

```
print(class_names)
print(data['class'].unique())
```

```
Index([u'unacc', u'acc', u'vgood', u'good'], dtype='object')
[0 1 2 3]
```

As we can see the values has been encoded into 4 different numeric labels.

Identify the predictor variables and encode any string variables to equivalent integer codes

```
data['buying'],_ = pd.factorize(data['buying'])
data['maint'],_ = pd.factorize(data['maint'])
data['doors'],_ = pd.factorize(data['doors'])
data['persons'],_ = pd.factorize(data['persons'])
data['lug_boot'],_ = pd.factorize(data['lug_boot'])
data['safety'],_ = pd.factorize(data['safety'])
data.head()
```

```
buying              maint        doors       persons   lug_boot safety       class

0                   0            0               0              0
0                   0            0

1                   0            0               0              0
0                   1            0

2                   0            0               0              0
0                   2            0

3                   0            0               0              0
1                   0            0

4                   0            0               0              0
1                   1            0
```

Check the data types now :

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
buying      1728 non-null int64
maint       1728 non-null int64
doors       1728 non-null int64
persons     1728 non-null int64
lug_boot    1728 non-null int64
safety      1728 non-null int64
class       1728 non-null int64
dtypes: int64(7)
memory usage: 94.6 KB
```

Everything is now converted in integer form.

Select the predictor feature and select the target variable

In clustering there is no target variable as such. However clustering helps us to find a cluster which can be used as weak labels. These weak labels can bootstrap our supervised learning. This technique is widely used for semi-supervised learning.

We need not worry about it as of now. You can just consider that y value will be used to validate the accuracy of weak labeling going ahead.

```
X = data
y = data.iloc[:,-1]
```

Training / model fitting

```
model = KMeans(n_clusters=4,random_state=123)
model.fit(X)
```

Check few parameters of the learnt cluster (Model parameters study):

Check the cluster centroids or means :

It should be 4 vectors or a matrix with 4 rows since the number of clusters we have fitted is 4.

Let's check

```
model.cluster_centers_
```

```
[[1.76355748e+00 1.82429501e+00 8.37310195e-01 1.09327549e+00
1.05206074e+00 1.12581345e+00 8.45986985e-01 7.05422993e+00]
[1.72631579e+00 1.75438596e+00 2.50175439e+00 9.29824561e-01
9.64912281e-01 9.07017544e-01 3.19298246e-01 1.87719298e+00]
[5.00000000e-01 1.50000000e+00 1.50000000e+00 1.00000000e+00
1.00000000e+00 1.00000000e+00 1.94444444e-01 4.50000000e+00]
[2.18490566e+00 3.88679245e-01 4.98113208e-01 9.88679245e-01
9.84905660e-01 9.81132075e-01 2.45283019e-01 1.06581410e-14]]
```

Check the goodness of the cluster i.e. within sum of square of the model:

The KMeans algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares Inertia, or the within-cluster sum of squares criterion, can be recognized as a measure of how internally coherent clusters are.
The k-means algorithm divides a set of N samples X into K disjoint clusters C, each described by the mean j of the samples in the cluster. The means are commonly called the cluster "centroids".
The K-means algorithm aims to choose centroids that minimise the inertia, or within-cluster sum of squared criterion:

$$\sum_{i=0}^{n} \min_{\mu_j \in C} (||x_j - \mu_i||^2)$$

Inertia is not a normalized metric: we just know that lower values are better and zero is optimal. But in very high-dimensional spaces, Euclidean distances tend to become inflated (this is an instance of the so-called "curse of dimensionality"). Running a dimensionality reduction algorithm such as PCA prior to k-means clustering can alleviate this problem and speed up the computations.

```
model.inertia_
```

```
9447.907008065738
```

Lesser this number better is the model fit.

Check the quality of the weak classification by the model

```
labels = model.labels_
# check how many of the samples were correctly labeled
correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." %
(correct_labels, y.size))
```

```
Result: 456 out of 1728 samples were correctly labeled.
correct 0.26 classification
```

We have achieved a weak classification accuracy of 26% by our unsupervised model. Not Bad.

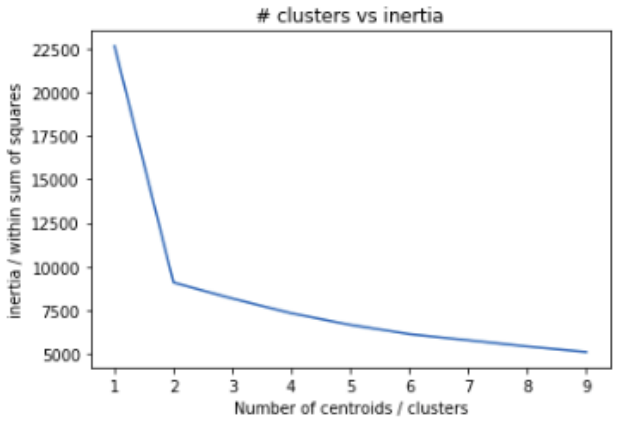
How do we know value of optimal value of K

The value of k basically means that in how many clusters you can separate out the most homogeneous data. Choosing a large value of K will lead to greater amount of execution time. Selecting the small value of K might or might not give a good fit. There is no such guaranteed way to find the best value of K. However we can run few experiments and find the value of each model's inertia and plot it on a graph. This plot is also known as elbow plot. We basically try to find the value of k from where there is major shift in value of inertia.

```
from mpl_toolkits.mplot3d import Axes3D

def elbow_plot(data, maxK=40, seed_centroids=None):
    """
        parameters:
        - data: pandas DataFrame (data to be fitted)
        - maxK (default = 10): integer (maximum number of clusters
with which to run k-means)
        - seed_centroids (default = None ): float (initial value of
centroids for k-means)
    """
    sse = {}
    for k in range(1, maxK):
        print("k: ", k)
        if seed_centroids is not None:
            seeds = seed_centroids.head(k)
            kmeans = KMeans(n_clusters=k, max_iter=500, n_init=100,
random_state=0, init=np.reshape(seeds, (k,1))).fit(data)
            data["clusters"] = kmeans.labels_
        else:
            kmeans = KMeans(n_clusters=k, max_iter=300, n_init=100,
random_state=0).fit(data)
            data["clusters"] = kmeans.labels_
        # Inertia: Sum of distances of samples to their closest
cluster center
        sse[k] = kmeans.inertia_
    plt.figure()
    plt.plot(list(sse.keys()), list(sse.values()))
    plt.show()
    return

elbow_plot(X, maxK=10)
```



By the plot we can see that there is a kink at k=2. Hence k =2 can be considered as good number of cluster to cluster this data.
Let's check the inertia and weak classification with number of cluster = 2.

```
model = KMeans(n_clusters=2,random_state=123)
model.fit(X)
labels = model.labels_
# check how many of the samples were correctly labeled
correct_labels = sum(y == labels)

print("Result: %d out of %d samples were correctly labeled." %
(correct_labels, y.size))
print("correct %.02f classification " %
(correct_labels/float(y.size)))
```

```
Result: 869 out of 1728 samples were correctly labeled.
correct 0.50 classification
```

Woah !!! our weak labeling is as good as a weak supervised classification model.

Pros and Cons

Time Complexity

K-means is linear in the number of data objects i.e. O(n), where n is the number of data objects. The time complexity of most of the hierarchical clustering algorithms is quadratic i.e. O(n2). Therefore, for the same amount of data, hierarchical clustering will take quadratic amount of time. Imagine clustering 1 million records?

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

K-means starts with a random choice of cluster centers, therefore it may yield different clustering results on different runs of the algorithm. Thus, the results may not be repeatable and lack consistency. However, with hierarchical clustering, you will most definitely get the same clustering results.

Off course, K-means clustering requires prior knowledge of K (or number of clusters), whereas in hierarchical clustering you can stop at whatever level (or clusters) you wish.

Hierarchical Clustering:

14. Discuss the agglomerative and divisive clustering approaches.

Agglomerative Hierarchical clustering Technique: In this technique, initially each data point is considered as an individual cluster. At each iteration, the similar clusters merge with other clusters until one cluster or K clusters are formed.

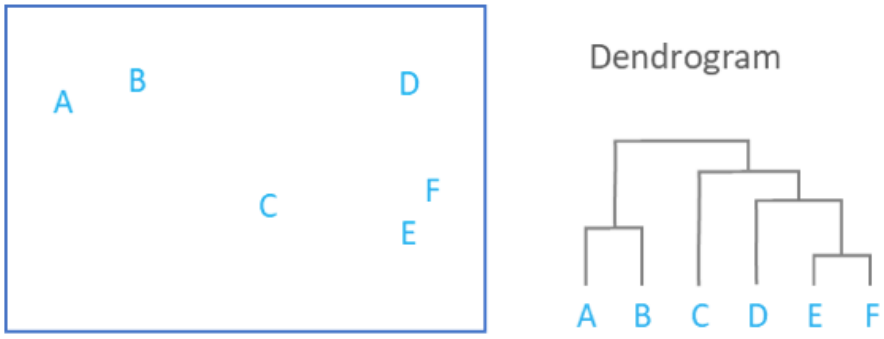
2. Divisive Hierarchical clustering Technique: Since the Divisive Hierarchical clustering Technique is not much used in the real world, I'll give a brief of the Divisive Hierarchical clustering Technique. In simple words, we can say that the Divisive Hierarchical clustering is exactly the opposite of the Agglomerative Hierarchical clustering. In Divisive Hierarchical clustering, we consider all the data points as a single cluster and in each iteration, we separate the data points from the cluster which are not similar. Each data point which is separated is considered as an individual cluster. In the end, we'll be left with n clusters.

Agglomerative clustering (Bottom-up approach): Each sample is treated as a single cluster and then successively merge (or agglomerate) pairs of clusters until all clusters have been merged into a single cluster.

Divisive clustering (top-down): A single cluster of all the samples is portioned recursively into two least similar clusters until there is one cluster for each observation. The divisive clustering algorithm is exactly the reverse of Agglomerative clustering

15. What are dendrograms?

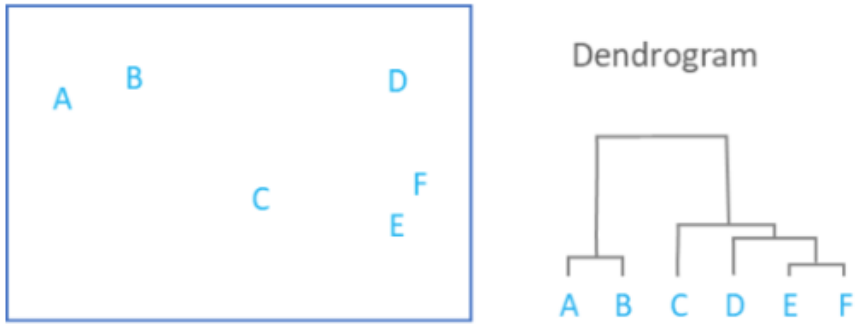
A *dendrogram* is a diagram that shows the hierarchical relationship between objects. It is most commonly created as an output from *hierarchical clustering*. The main use of a dendrogram is to work out the best way to allocate objects to clusters. The dendrogram below shows the hierarchical clustering of six *observations* shown on the *scatterplot* to the left. (Dendrogram is often miswritten as dendogram.)



How to read a dendrogram

The key to interpreting a dendrogram is to focus on the height at which any two objects are joined together. In the example above, we can see that E and F are most similar, as the height of the link that joins them together is the smallest. The next two most similar objects are A and B.

In the dendrogram above, the height of the dendrogram indicates the order in which the clusters were joined. A more informative dendrogram can be created where the heights reflect the distance between the clusters as is shown below. In this case, the dendrogram shows us that the big difference between clusters is between the cluster of A and B versus that of C, D, E, and F.

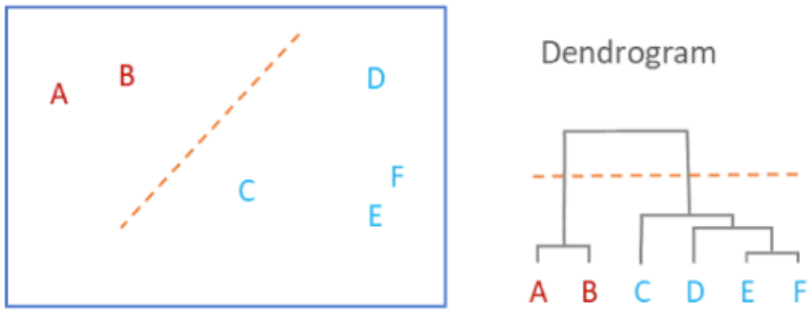


It is important to appreciate that the dendrogram is a summary of the distance matrix, and, as occurs with most summaries, information is lost. For example, the dendrogram suggests that C and D are much closer to each other than is C to B, but the original data (shown in the scatterplot), shows us that this is not true. To use some jargon, a dendrogram is only accurate when data satisfies the *ultrametric tree inequality*, and this is unlikely for any real-world data.

The consequence of the information loss is that the dendrograms are most accurate at the bottom, showing which items are very similar.

Allocating observations to clusters

Observations are allocated to clusters by drawing a horizontal line through the dendrogram. Observations that are joined together below the line are in clusters. In the example below, we have two clusters. One cluster combines A and B, and a second cluster combining C, D, E, and F.



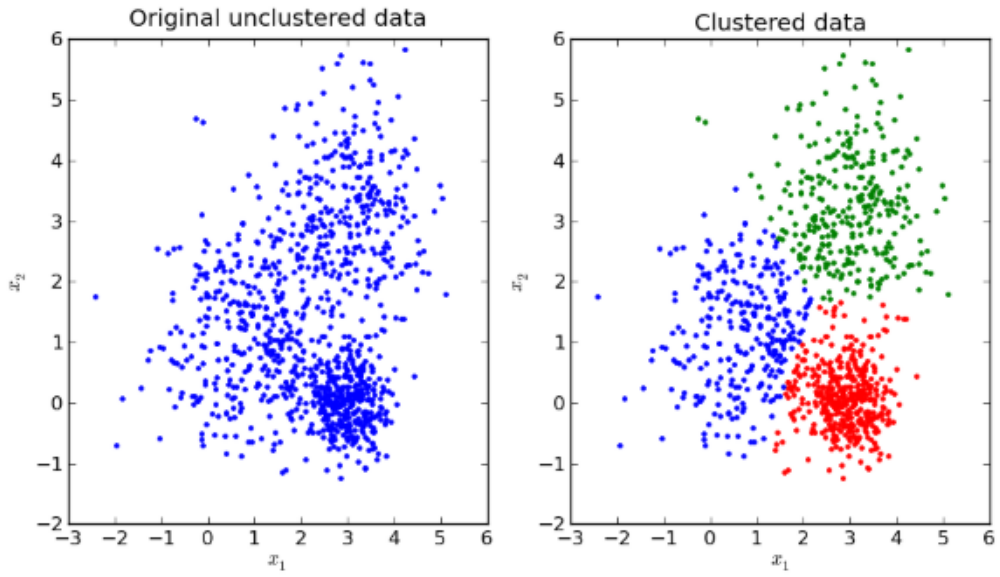
Dendrograms cannot tell you how many clusters you should have

A common mistake people make when reading dendrograms is to assume that the shape of the dendrogram gives a clue as to how many clusters exist. In the example above, the (incorrect) interpretation is that the dendrogram shows that there are two clusters, as the distance between the clusters (the vertical segments of the dendrogram) are highest between two and three clusters.

Interpretation of this kind is justified only when the ultrametric tree inequality holds, which, as mentioned above, is very rare. In general, it is a mistake to use dendrograms as a tool for determining the number of clusters in data. Where there is an obviously "correct" number of clusters this will often be evident in a dendrogram. However, dendrograms often suggest a correct number of clusters when there is no real evidence to support the conclusion.

16. Discuss the Hierarchical clustering in detail.

Clustering is basically a technique that groups similar data points such that the points in the same group are more similar to each other than the points in the other groups. The group of similar data points is called a **Cluster.**



Differences between Clustering and Classification/Regression models:

In classification and regression models, we are given a data set(D) which contains data points(Xi) and class labels(Yi). Where, Yi's belong to {0,1} or {0,1,2,...,n) for Classification models and Yi's belong to real values for regression models.

When comes to clustering, we're provided with a data set that contains only data points(Xi). Here we're **not** provided with the class labels(Yi).

Hierarchical clustering Technique:

Hierarchical clustering is one of the popular and easy to understand clustering technique. This clustering technique is divided into two types:

1. Agglomerative
2. Divisive

Agglomerative Hierarchical clustering Technique: In this technique, initially each data point is considered as an individual cluster. At each iteration, the similar clusters merge with other clusters until one cluster or K clusters are formed.

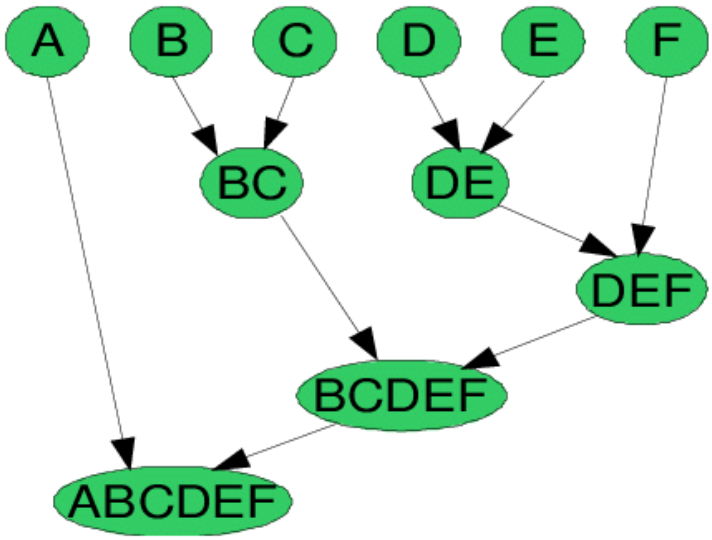
The basic algorithm of Agglomerative is straight forward.

- Compute the proximity matrix
- Let each data point be a cluster
- Repeat: Merge the two closest clusters and update the proximity matrix
- Until only a single cluster remains

Key operation is the computation of the proximity of two clusters

To understand better let's see a pictorial representation of the Agglomerative Hierarchical clustering Technique. Lets say we have six data points {A,B,C,D,E,F}.

- Step- 1: In the initial step, we calculate the proximity of individual points and consider all the six data points as individual clusters as shown in the image below.

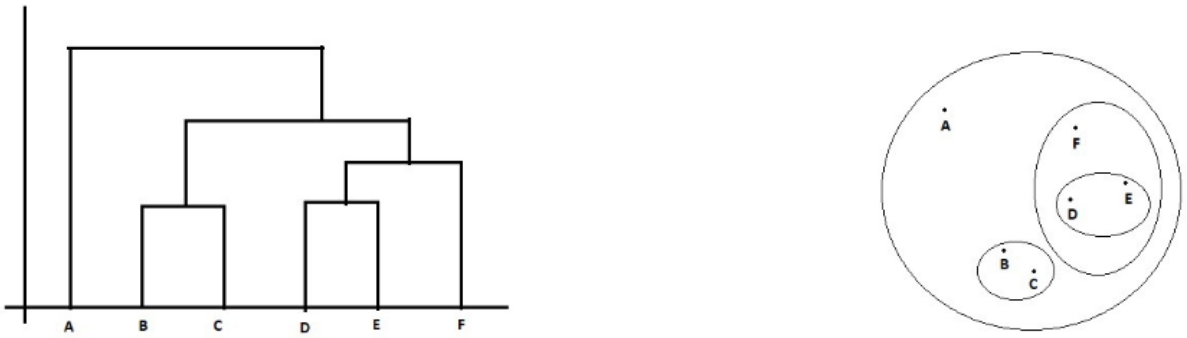


Agglomerative Hierarchical Clustering Technique

- Step- 2: In step two, similar clusters are merged together and formed as a single cluster. Let's consider B,C, and D,E are similar clusters that are merged in step two. Now, we're left with four clusters which are A, BC, DE, F.
- Step- 3: We again calculate the proximity of new clusters and merge the similar clusters to form new clusters A, BC, DEF.
- Step- 4: Calculate the proximity of the new clusters. The clusters DEF and BC are similar and merged together to form a new cluster. We're now left with two clusters A, BCDEF.
- Step- 5: Finally, all the clusters are merged together and form a single cluster.

The Hierarchical clustering Technique can be visualized using a **Dendrogram.**

A **Dendrogram** is a tree-like diagram that records the sequences of merges or splits.



Dendrogram representation

2. Divisive Hierarchical clustering Technique: Since the Divisive Hierarchical clustering Technique is not much used in the real world, I'll give a brief of the Divisive Hierarchical clustering Technique.

In simple words, we can say that the Divisive Hierarchical clustering is exactly the opposite of the **Agglomerative Hierarchical clustering.** In Divisive Hierarchical clustering, we consider all the data points as a single cluster and in each iteration, we separate the data points from the cluster which are not similar. Each data point which is separated is considered as an individual cluster. In the end, we'll be left with n clusters.

As we're dividing the single clusters into n clusters, it is named as **Divisive Hierarchical clustering.**

So, we've discussed the two types of the Hierarchical clustering Technique.

But wait!! we're still left with the **important part** of Hierarchical clustering.

"HOW DO WE CALCULATE THE SIMILARITY BETWEEN TWO CLUSTERS???"

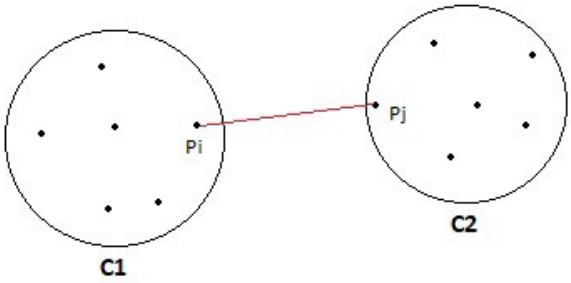
Calculating the similarity between two clusters is important to merge or divide the clusters. There are certain approaches which are used to calculate the similarity between two clusters:

- MIN
- MAX
- Group Average
- Distance Between Centroids
- Ward's Method
- **MIN:** Also known as single linkage algorithm can be defined as the similarity of two clusters C1 and C2 is equal to the **minimum** of the similarity between points Pi and Pj such that Pi belongs to C1 and Pj belongs to C2.

Mathematically this can be written as,

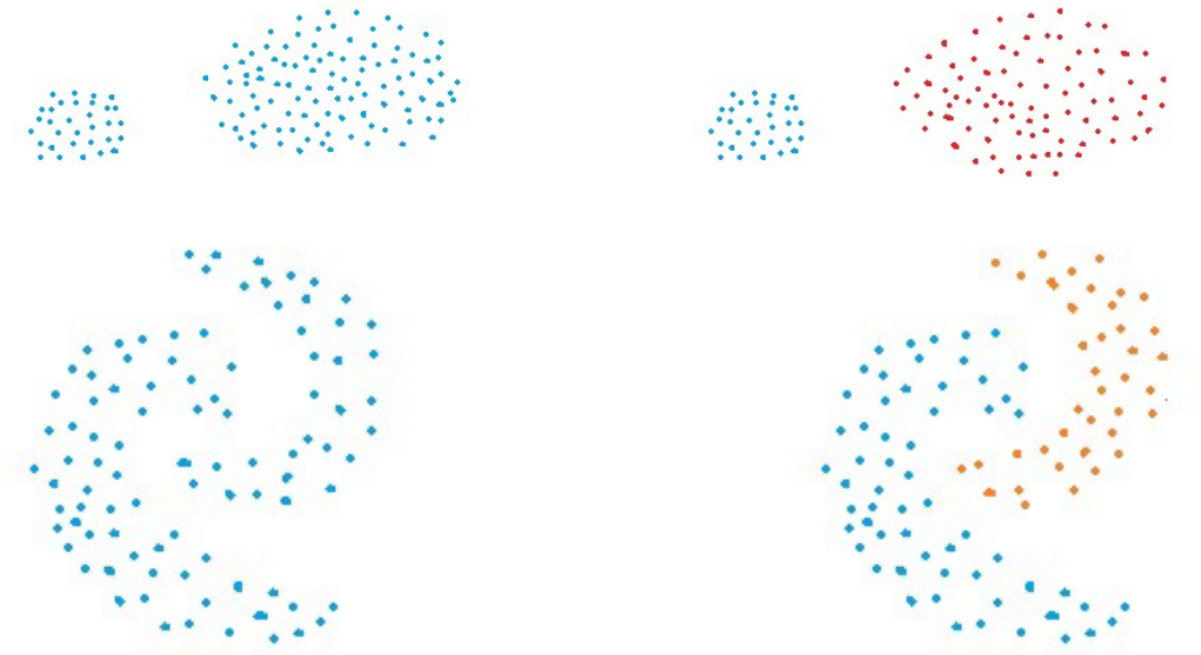
$$Sim(C1,C2) = \text{Min } Sim(Pi,Pj) \text{ such that } Pi \in C1 \ \& \ Pj \in C2$$

In simple words, pick the two closest points such that one point lies in cluster one and the other point lies in cluster 2 and take their similarity and declare it as the similarity between two clusters.



Pros of MIN:

- This approach can separate non-elliptical shapes as long as the gap between two clusters is not small.



Original data vs Clustered data using MIN approach

Cons of MIN:

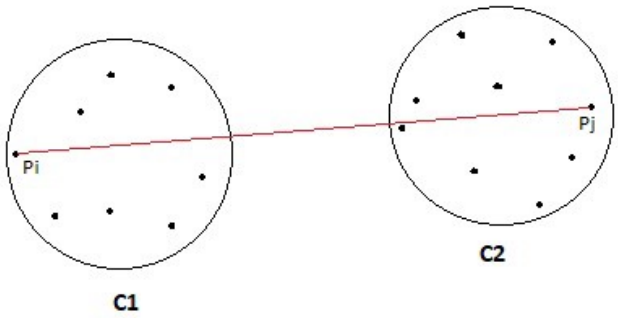
- MIN approach cannot separate clusters properly if there is noise between clusters.

- **MAX:** Also known as the complete linkage algorithm, this is exactly opposite to the **MIN** approach. The similarity of two clusters C1 and C2 is equal to the **maximum** of the similarity between points Pi and Pj such that Pi belongs to C1 and Pj belongs to C2.

Mathematically this can be written as,

$Sim(C1,C2) = Max\ Sim(Pi,Pj)$ such that $Pi \in C1$ & $Pj \in C2$

In simple words, pick the two farthest points such that one point lies in cluster one and the other point lies in cluster 2 and take their similarity and declare it as the similarity between two clusters.



Pros of MAX:

- MAX approach does well in separating clusters if there is noise between clusters.

Original data vs Clustered data using MAX approach

Cons of Max:

- Max approach is biased towards globular clusters.
- Max approach tends to break large clusters.

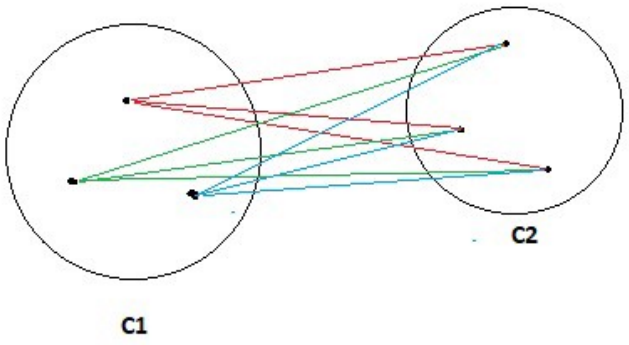
Original data vs Clustered data using MAX approach

- **Group Average:** Take all the pairs of points and compute their similarities and calculate the average of the similarities.

Mathematically this can be written as,

$$sim(C1,C2) = \sum sim(Pi, Pj)/|C1|*|C2|$$

where, $Pi \in C1$ & $Pj \in C2$

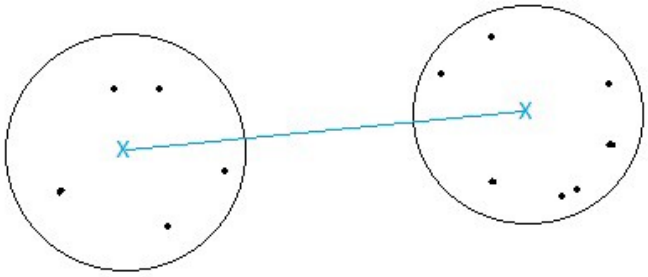


Pros of Group Average:

- The group Average approach does well in separating clusters if there is noise between clusters.

Cons of Group Average:

- The group Average approach is biased towards globular clusters.
- **Distance between centroids:** Compute the centroids of two clusters C1 & C2 and take the similarity between the two centroids as the similarity between two clusters. This is a less popular technique in the real world.



- **Ward's Method:** This approach of calculating the similarity between two clusters is exactly the same as Group Average except that Ward's method calculates the sum of the square of the distances Pi and PJ.

Mathematically this can be written as,

$$sim(C1,C2) = \sum (dist(Pi, Pj))^2 / |C1| * |C2|$$

Pros of Ward's method:

- Ward's method approach also does well in separating clusters if there is noise between clusters.

Cons of Ward's method:

- Ward's method approach is also biased towards globular clusters.

Space and Time Complexity of Hierarchical clustering Technique:

Space complexity: The space required for the Hierarchical clustering Technique is very high when the number of data points are high as we need to store the similarity matrix in the RAM. The space complexity is the order of the square of n.

Space complexity = $O(n^2)$ where n is the number of data points.

Time complexity: Since we've to perform n iterations and in each iteration, we need to update the similarity matrix and restore the matrix, the time complexity is also very high. The time complexity is the order of cube of n.

Time complexity = $O(n^3)$ where n is the number of data points.

Limitations of Hierarchical clustering Technique:

1. There is no mathematical objective for Hierarchical clustering.
2. All the approaches to calculate the similarity between clusters has its own disadvantages.
3. High space and time complexity for Hierarchical clustering. Hence this clustering algorithm cannot be used when we have huge data.

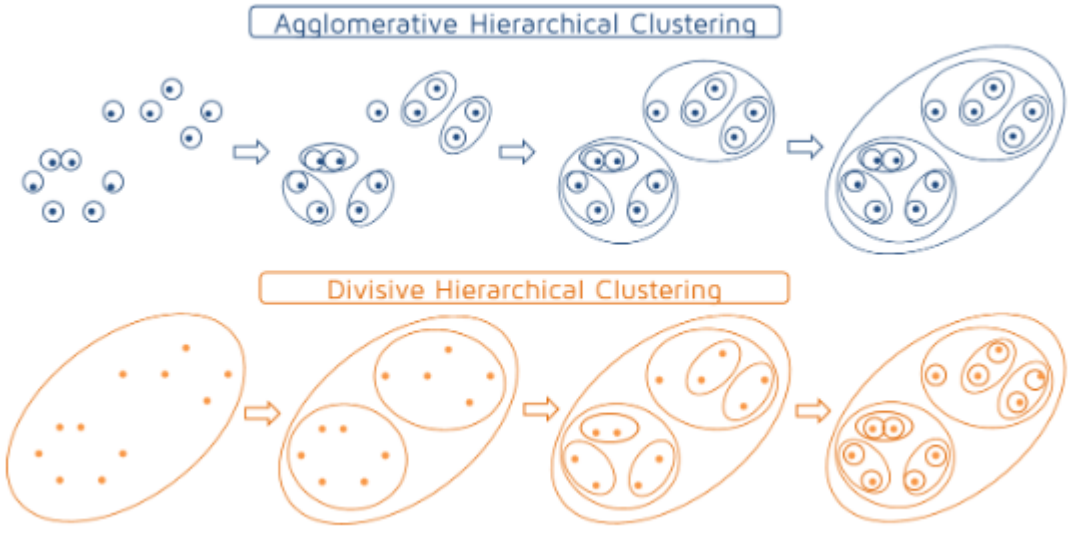
Clustering is an algorithm that groups similar objects into groups called *clusters*. The endpoint is a set of clusters, where each cluster is distinct from each other cluster, and the objects within each cluster are broadly similar to each other.

OK now let's try to understand hierarchical clustering by a real-life example.

If you think about the file arrangement in your personal computer, you will know that it is a hierarchy. Your hard disk is divided into various drives. Each drive contains various folders, an opening that reveals more folders until a point. **Hierarchical clustering** also uses the same approaches where it uses clusters instead of folders.

Types of Clustering

1. **Agglomerative clustering** (Bottom-up approach): Each sample is treated as a single cluster and then successively merge (or *agglomerate*) pairs of clusters until all clusters have been merged into a single cluster.
2. **Divisive clustering (top-down)**: A single cluster of all the samples is portioned recursively into two least similar clusters until there is one cluster for each observation. The divisive clustering algorithm is exactly the reverse of Agglomerative clustering.



Comparison between agglomerative and divisive algorithms

> *Scikit-learn implements only the agglomerative clustering.*

- Compared to K-means, agglomerative algorithms are more cumbersome and do not scale well to large datasets.
- Agglomerative algorithms are more suitable for statistical studies.
- These algorithms do offer the advantage of creating a complete range of nested cluster solutions.

Linkage measures

Before any clustering is performed, it is required to determine the proximity matrix containing the distance between each point using a distance function. Then, the matrix is updated to display the distance between each cluster. The following three methods differ in how the distance between each cluster is measured.

Complete Linkage

For each pair of clusters, the algorithm computes and merges them to minimize the maximum distance between the clusters (in other words, the distance of the farthest elements)

$$\forall\, C_i, C_j \;\; L_{ij} = max\{d(x_a, x_b)\, \forall\, x_a \in C_i \; and \; x_b \in \; C_j\}$$

Average Linkage

It's similar to complete linkage, but in this case, the algorithm uses the average distance between the pairs of clusters

$$\forall\, C_i, C_j \;\; L_{ij} = \frac{1}{|C_i||C_j|} \sum_{x_a \in C_i} \sum_{x_b \in C_j} d(x_a, x_b)$$

Ward's Linkage

In this method, all clusters are considered and the algorithm computes the sum of squared distances within the clusters and merges them to minimize it. From a statistical viewpoint, the process of agglomeration leads to a reduction in the variance of each resulting cluster

$$\forall\, C_i, C_j \;\; L_{ij} = \sum_{x_a \in C_i} \sum_{x_b \in C_j} \|x_a - x_b\|^2$$

Measures of distance (similarity)

We define affinity, a metric function of two arguments with the same dimensionality as of dataset. The most common metrics (also supported by scikit-learn) are

Euclidean or L2

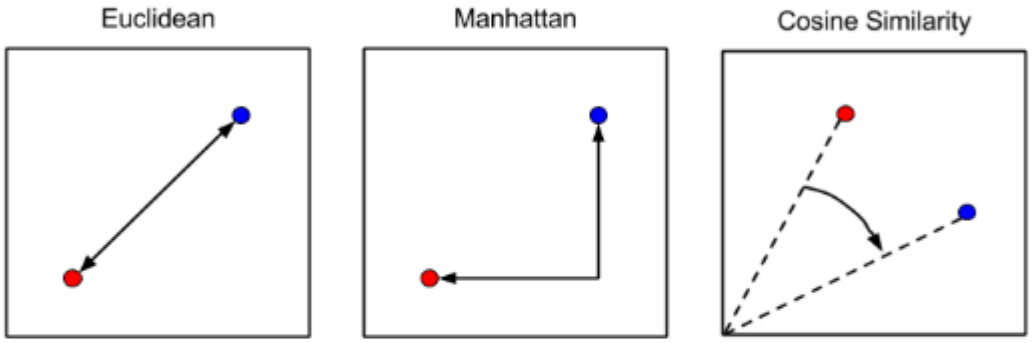
Euclidean distance is the "ordinary" straight-line distance between two points in Euclidean space. With this distance, Euclidean space becomes a metric space

$$d_{Euclidean}(\bar{x}_1, \bar{x}_2) = \|\bar{x}_1 - \bar{x}_2\|_2 = \sqrt{\sum_i (x_1^i - x_2^i)^2}$$

Manhattan or L1

Similar to Euclidean, but the distance is calculated by summing the absolute value of the difference between the dimensions. For eg., Manhattan distance implies moving straight, first along one axis and then along with the other.

$$d_{Manhattan}(\bar{x}_1, \bar{x}_2) = \|\bar{x}_1 - \bar{x}_2\|_1 = \sum_i |x_1^i - x_2^i|$$



Cosine

Cosine distance reduces noise by taking the shape of the variables, more than their values, into account. It tends to associate observations that have the same maximum and minimum variables, regardless of their effective value.

$$d_{Cosine}(\bar{x}_1, \bar{x}_2) = 1 - \frac{\bar{x}_1 \cdot \bar{x}_2}{\|\bar{x}_1\|_2 \|\bar{x}_2\|_2}$$

> *Note: The Ward's linkage supports only the Euclidean distance.*

Working

1. Compute the distance matrix between each input data point.
2. Consider each data point as a cluster.
3. Merge two closest clusters and update the distance matrix.
4. Repeat step third until one single cluster remains.

```
1  from sklearn.cluster import AgglomerativeClustering
2
3  hc = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='ward')
4  y_hc = hc.fit_predict(your_data)
```

hcalgo.py hosted with ❤ by GitHub view raw

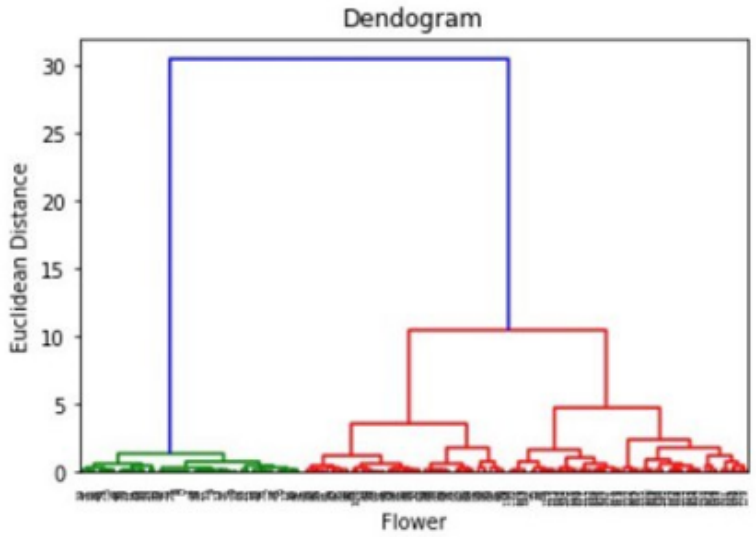
Dendrograms

To better understand the **agglomeration** process, it's useful to introduce a graphical method called a dendrogram, which shows in a static way how the aggregations are performed, starting from the bottom (where all samples are separated) till the top (where the linkage is complete)

```
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method='ward'))

plt.title('Dendrogram')
plt.xlabel('Flower')
plt.ylabel('Euclidean Distance')
plt.show()
```

dendogram.py hosted with ❤ by GitHub view raw



- We get three clusters by our dendrogram as shown in the figure above
- Height in dendrogram at which two clusters are merged represents the distance between two clusters in data space.

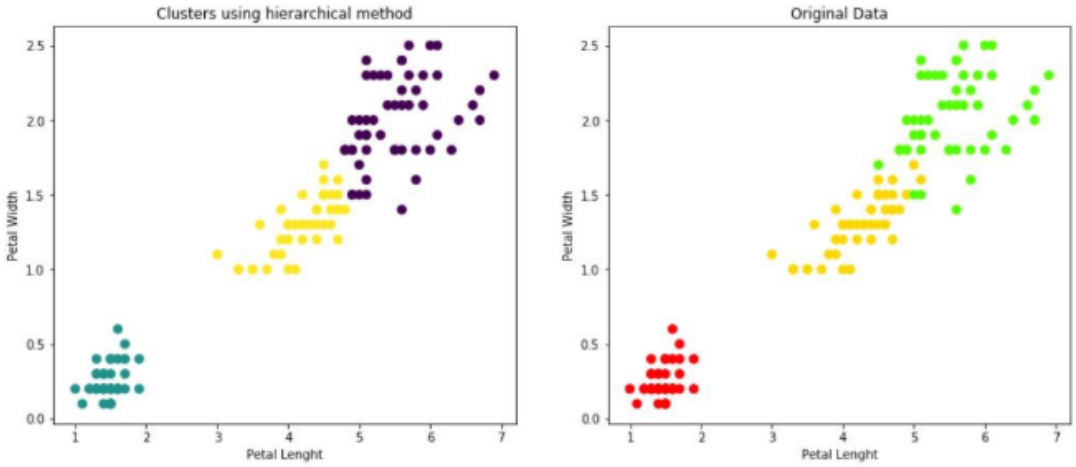
Comparing with original labels

Let us compare how this Hierarchical clustering work on our famous iris dataset

```
fig = plt.figure(figsize=(15,6))
plt.subplot(121)
plt.scatter(X[:, 0], X[:, 1], c=y_hc, s=50, cmap='viridis')
plt.title('Clusters using hierarchical method')
plt.ylabel('Petal Width')
plt.xlabel('Petal Lenght')

plt.subplot(122)
plt.scatter(X[:, 0], X[:, 1], c=df.iloc[:,5].values, s=50, cmap='prism')
plt.title('Original Data')
plt.ylabel('Petal Width')
plt.xlabel('Petal Lenght')
plt.show()
```

compare.py hosted with ❤ by GitHub view raw



Space and Time complexity

Space: it requires $O(N^2)$ space for storing the distance matrix.

Time: $O(N^2Log(N))$ in most cases.

Disadvantages of Hierarchical Clustering

- HC is computationally expensive $O(N^2Log(N))$ hence is not recommended on huge datasets whereas k-means using linear time
- It is sensitive to noise and outliers.
- We have to define the number of clusters like k-means clustering algorithm

17. Discuss the various linkage methods for clustering.

Clustering tries to find structure in data by creating groupings of data with similar characteristics. The most famous clustering algorithm is likely K-means, but there are a large number of ways to cluster observations. Hierarchical clustering is an alternative class of clustering algorithms that produce 1 to n clusters, where n is the number of observations in the data set. As you go down the hierarchy from 1 cluster (contains all the data) to n clusters (each observation is its own cluster), the clusters become more and more similar (almost always).

There are two types of hierarchical clustering: divisive (top-down) and agglomerative (bottom-up).

Divisive Divisive hierarchical clustering works by starting with 1 cluster containing the entire data set. The observation with the highest average dissimilarity (farthest from the cluster by some metric) is reassigned to its own cluster. Any observations in the old cluster closer to the new cluster are assigned to the new cluster. This process repeats with the largest cluster until each observation is its own cluster.

Agglomerative

Agglomerative clustering starts with each observation as its own cluster. The two closest clusters are joined into one cluster. The next closest clusters are grouped together and this process continues until there is only one cluster containing the entire data set.

What does it mean to be close?

In the section above, I neglected to define what “close” means. There are a variety of possible metrics, but I will list the 4 most popular: single-linkage, complete-linkage, average-linkage, and centroid-linkage.

Single-Linkage

Single-linkage (nearest neighbor) is the shortest distance between a pair of observations in two clusters. It can sometimes produce clusters where observations in different clusters are closer together than to observations within their own clusters. These clusters can appear spread-out.

Complete-Linkage

Complete-linkage (farthest neighbor) is where distance is measured between the farthest pair of observations in two clusters. This method usually produces tighter clusters than single-linkage, but these tight clusters can end up very close together. Along with average-linkage, it is one of the more popular distance metrics.

Average-Linkage

Average-linkage is where the distance between each pair of observations in each cluster are added up and divided by the number of pairs to get an average inter-cluster distance. Average-linkage and complete-linkage are the two most popular distance metrics in hierarchical clustering.

Centroid-Linkage

Centroid-linkage is the distance between the centroids of two clusters. As the centroids move with new observations, it is possible that the smaller clusters are more similar to the new larger cluster than to their individual clusters causing an inversion in the dendrogram. This problem doesn’t arise in the other linkage methods because the clusters being merged will always be more similar to themselves than to the new larger cluster

18. Discuss the differences between K-Means and Hierarchical clustering.

Difference between K Means and Hierarchical clustering

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(n2).

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.

K Means is found to work well when the shape of the clusters is hyper spherical (like circle in 2D, sphere in 3D).

K Means clustering requires prior knowledge of K i.e. no. of clusters you want to divide your data into. But, you can stop at whatever number of clusters you find appropriate in hierarchical clustering by interpreting the dendrogram

==========

The most important difference is the hierarchy. Actually, there are two different approaches that fall under this name: top-down and bottom-up.

In top-down hierarchical clustering, we divide the data into 2 clusters (using k-means with k=2 , for example). Then, for each cluster, we can repeat this process, until all the clusters are too small or too similar for further clustering to make sense, or until we reach a preset number of clusters.

In bottom-up hierarchical clustering, we start with each data item having its own cluster. We then look for the two items that are most similar, and combine them in a larger cluster. We keep repeating until all the clusters we have left are too dissimilar to be gathered together, or until we reach a preset number of clusters.

In k-means clustering, we try to identify the best way to divide the data into k sets simultaneously. A good approach is to take k items from the data set as initial cluster representatives, assign all items to the cluster whose representative is closest, and then calculate the cluster mean as the new representative, until it converges (all clusters stay the same).

==========

Difference between Hierarchical and Partitional Clustering

In conclusion, the main differences between Hierarchical and Partitional Clustering are that each cluster starts as individual clusters or singletons. With every iteration, the closest clusters get merged. This process repeats until one single cluster remains for Hierarchical clustering.

An example of Hierarchical clustering is the Two-Step clustering method.

Whereas, Partitional clustering requires the analyst to define K number of clusters before running the algorithm and objects closest to the clusters are grouped. With every iteration, the distance of the clusters shifts. This process continues until there is no more movement in the centroid of each cluster or until the stopping criterion is met.

An example of Partitional clustering is the K-Means clustering method.

==========

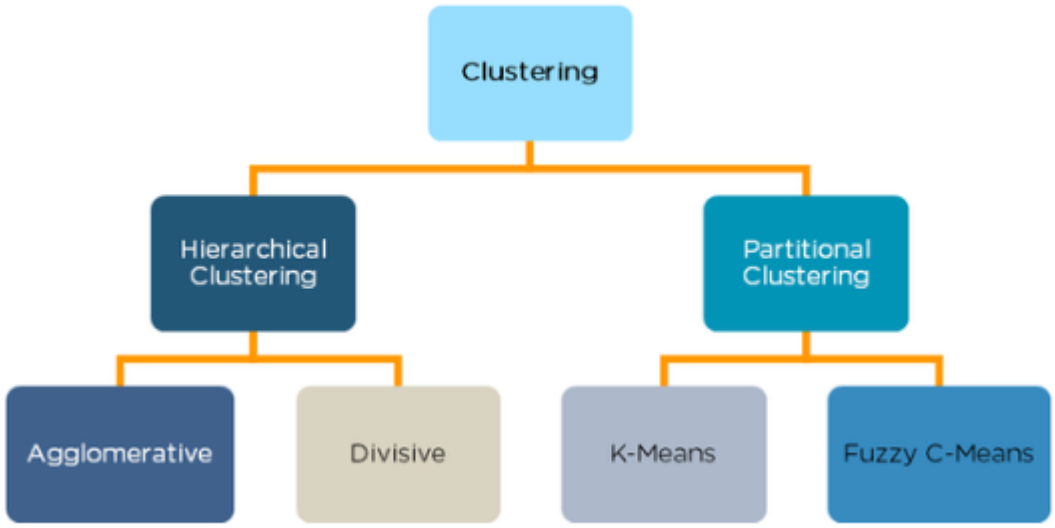
I would say hierarchical clustering is usually preferable, as it is both more flexible and has fewer hidden assumptions about the distribution of the underlying data.

With k-Means clustering, you need to have a sense ahead-of-time what your desired number of clusters is (this is the 'k' value). Also, k-means will often give unintuitive results if (a) your data is not well-separated into sphere-like clusters, (b) you pick a 'k' not well-suited to the shape of your data, i.e. you pick a value too high or too low, or (c) you have weird initial values for your cluster centroids (one strategy is to run a bunch of k-means algorithms with random starting centroids and take some common clustering result as the final result).

In contrast, hierarchical clustering has fewer assumptions about the distribution of your data - the only requirement (which k-means also shares) is that a distance can be calculated each pair of data points. Hierarchical clustering typically 'joins' nearby points into a cluster, and then successively adds nearby points to the nearest group. You end up with a 'dendrogram', or a sort of connectivity plot. You can use that plot to decide after the fact of how many clusters your data has, by cutting the dendrogram at different heights. Of course, if you need to pre-decide how many clusters you want (based on some sort of business need) you can do that too. Hierarchical clustering can be more computationally expensive but usually produces more intuitive results.

What are the types of Clustering?

Clustering can be divided into two subgroups. They are,



Hierarchical vs K-Means Clustering

Comparison Between K-Means & Hierarchical Clustering

As we have seen in the above section, the results of both the clustering are almost similar to the same dataset. It may be possible that when we have a very large dataset, the shape of clusters may differ a little. However, along with many similarities, these two techniques have some differences also. The below table shows the comparison between K-Means and Hierarchical clustering algorithms based on our implementations.

	K-Means Clustering	Hierarchical Clustering
Category	Centroid based, partition-based	Hierarchical, Agglomerative
Method to find the optimal number of clusters	The Elbow method using WCSS	Dendrogram
Directional approach	Not any, the only centroid is considered to form clusters	Top-down, bottom-up
Python Library	sklearn – KMeans	sklearn-AgglomerativeClustering

3. Comparisons on K –Means and Hierarchical Clustering

Properties	**K –Means**	**Hierarchical Clustering**
Definition	K Means Clustering generates a specific number of disjoint, flat (non-hierarchical) Clusters.	Hierarchical Clustering method construct a hierarchy of Clustering, not just a single partition of objects.
Clustering Criteria	It is well suited to generating globular Cluster.	Use a distance matrix as Clustering Criteria. A termination Condition can be used .Example –A number of Clusters.

Performance	The performance of K- mean algorithm is better than Hierarchical Clustering Algorithm.	Hierarchical Clustering Algorithm performance is less as compare to K- mean algorithm.
Category Data	K- Means can be used in categorical data is first converted into numeric by assigning rank.	Hierarchical algorithm was adopted for categorical data, and due to its complexity a new approach for assigning rank value to each categorical attribute.
Sensitive To Noise	K-Means is very sensitive to noise in the dataset.	It is less sensitive to noise in the dataset.
Cluster	There are always K.	The number of Clusters k is not required as an input.
Execution Time	K -mean algorithm also increases its time of execution.	Hierarchical algorithm its performance is better.
Quality	K-Means algorithms Shows less quality.	Hierarchical algorithm shows more quality.

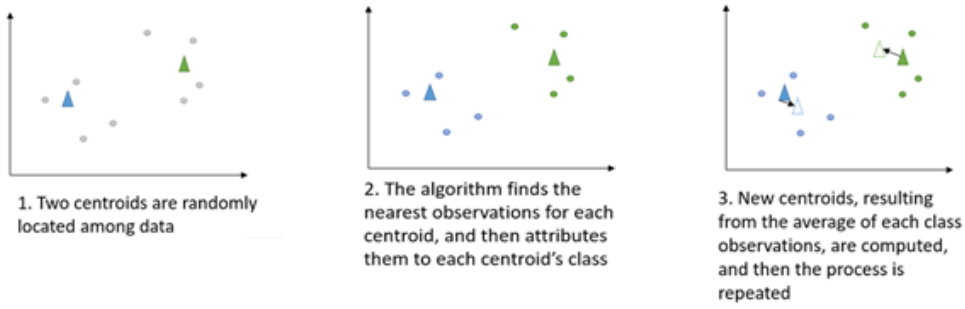
Quality	K-Means algorithms Shows less quality.	Hierarchical algorithm shows more quality.
Data Set	k -mean algorithm is good for large dataset.	Hierarchical is good for small datasets.

K-means

The first step of this algorithm is creating, among our unlabeled observations, *c* new observations, randomly located, called 'centroids'. The number of centroids will be representative of the number of output classes (which, remember, we do not know). Now, an iterative process will start, made of two steps:

- First, for each centroid, the algorithm finds the nearest points (in terms of distance that is usually computed as Euclidean distance) to that centroid, and assigns them to its category;
- Second, for each category (represented by one centroid), the algorithm computes the average of all the points which has been attributed to that class. The output of this computation will be the new centroid for that class.

Every time the process is reiterated, some observations, initially classified together with one centroid, might be redirected to another one. Furthermore, after several reiterations, the change in centroids' location should be less and less important since the initial random centroids are converging to the real ones. This process ends when there is no more change in centroids' position.



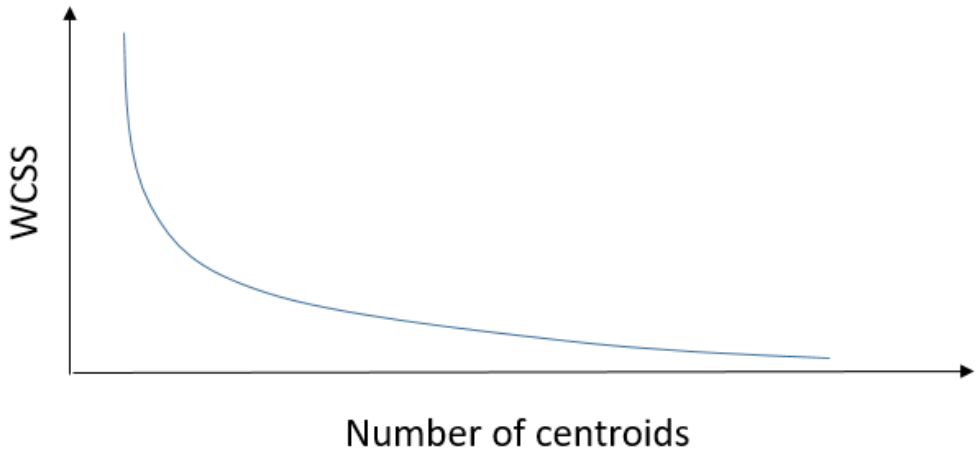
Now, how can we decide the number of centroids?

There are many methods that could be employed for this task. However, in this article, I'm going to explain and use the so-called 'Elbow method'. The idea is that what we would like to observe within our clusters is a low level of variation, which is measured with the within-cluster sum of squares (WCSS):

$$WCSS = \sum_{xi \in c} (x_i - \bar{x})^2$$

And it is intuitive to understand that, the higher the number of centroids, the lower the WCSS. In particular, if we have as many centroids as the number of our observations, each WCSS will be equal to zero. However, if we remember the law of parsimony, we know that setting the highest number possible of centroids would be inconsistent.

The idea is picking that number of centroids after which the reduction in WCSS is irrelevant. The relation I've just described can be represented with the following graph:



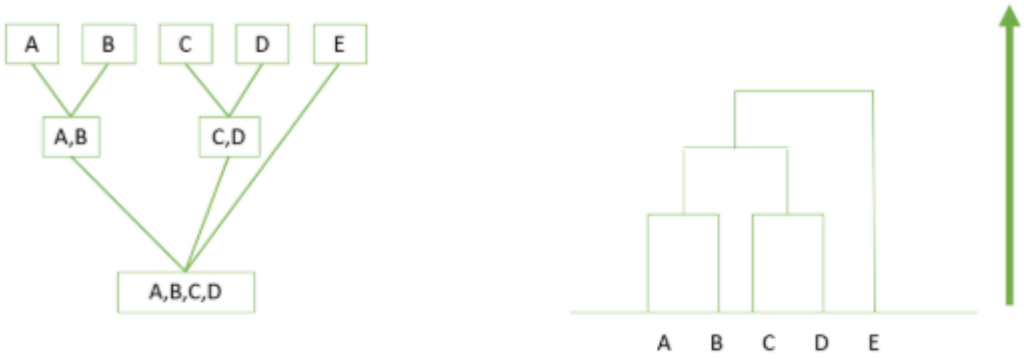
The idea is that, if the plot is an arm, the elbow of the arm is the optimal number of centroids.

Hierarchical Clustering

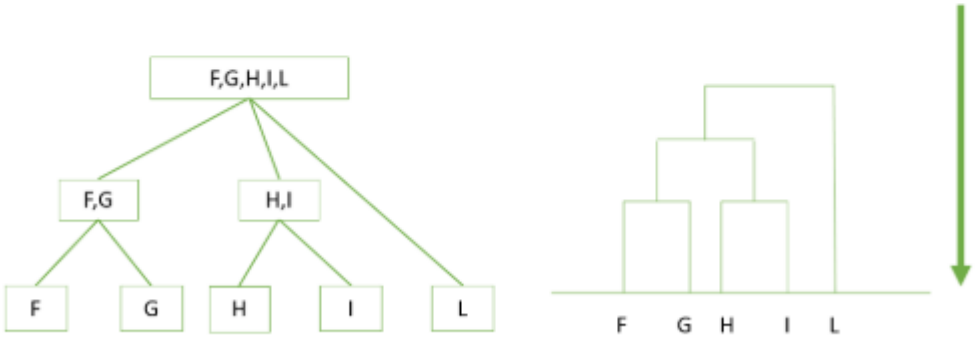
This algorithm can use two different techniques:

- Agglomerative
- Divisive

Those latter are based on the same ground idea, yet work in the opposite way: being K the number of clusters (which can be set exactly like in K-means) and n the number of data points, with $n > K$, agglomerative HC starts from n clusters, then aggregates data until it obtains K clusters; divisive HC, on the other hand, starts from just one cluster and then splits it depending, again, on similarities, until it obtains K clusters. Note that, when I say similarities, I'm referring to the distance among data points, which can be computed in different ways (I will dwell on this later on).



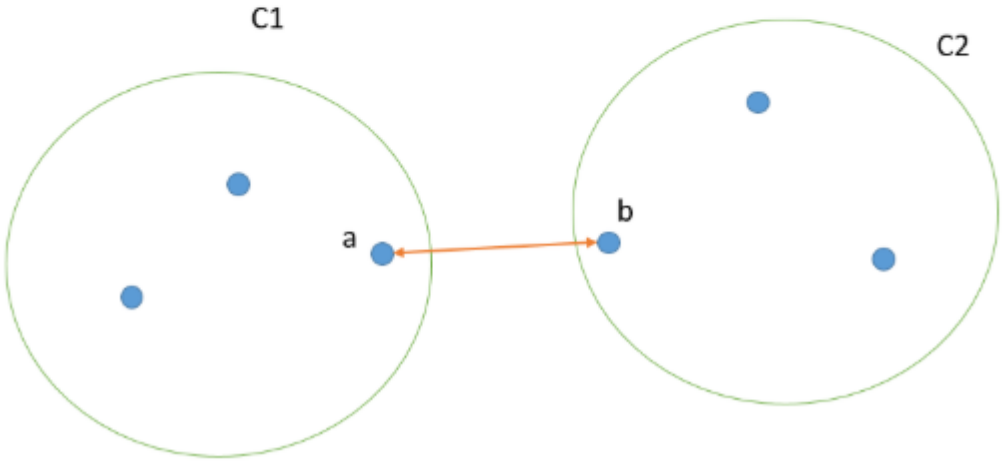
Agglomerative HC



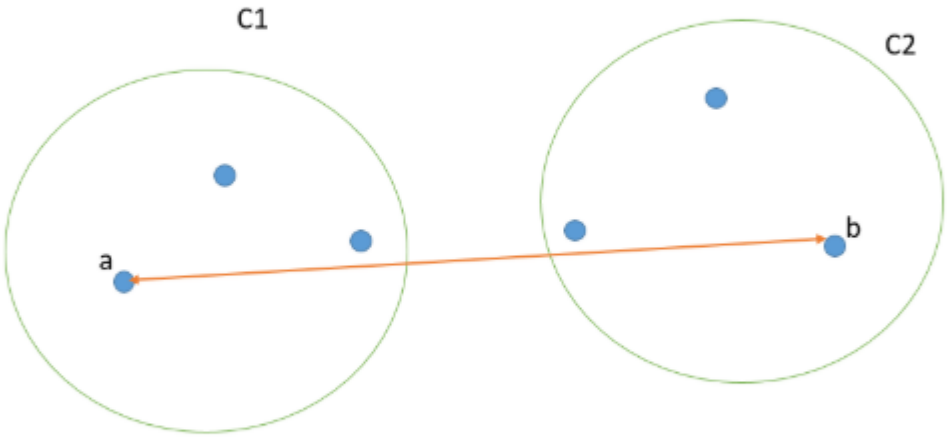
Divisive HC

As anticipated, the key element of discrimination here is similarity among data points. In mathematical terms, similarity mainly refers to distance, and it can be computed with different approaches. Here, I will propose three of them:

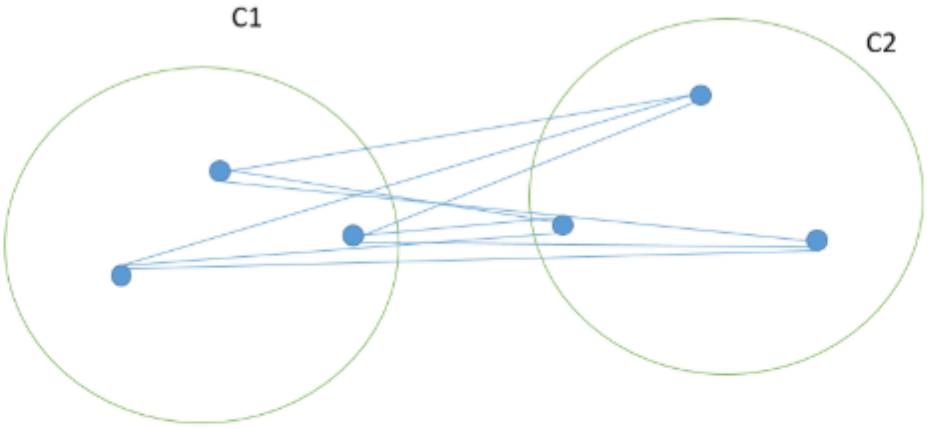
- Min: it states that, given two clusters C1 and C2, the similarity between them is equal to the minimum of similarity (translated: distance) between point a and b, such that a belongs to C1 and b belongs to C2.



- Max: it states that, given two clusters C1 and C2, the similarity between them is equal to the maximum of similarity between point a and b, such that a belongs to C1 and b belongs to C2.



- Average: it takes all the pairs of points, compute their similarities and then calculate the average of the similarities. That latter is the similarity between the clusters C1 and C2.



So, to recap, both the algorithms look for similarities among data and both use the same approaches to decide the number of clusters. Choosing the one rather than the other really depends on the kind of task you're facing.

DBSCAN

19. Discuss the basic terms used in DBSCAN.

DBSCAN(Density Based Spatial Clustering of Apllications with Noise)

It is an unsupervised machine learning algorithm.This algorithm defines clusters as continuous regions of high density.

Some definitions first:

Epsilon: This is also called eps. This is the distance till which we look for the neighbouring points.

Min_points: The minimum number of points specified by the user.

Core Points: If the number of points inside the *eps radius* of a point is greater than or equal to the *min_points* then it's called a core point.

Border Points: If the number of points inside the *eps radius* of a point is less than the *min_points* and it lies within the *eps radius* region of a core point, it's called a border point.

Noise: A point which is neither a core nor a border point is a noise point.

20. Discuss the step by step implementation of DBSCAN.

Density-based spatial clustering of applications with noise (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.

Parameters:

The DBSCAN algorithm basically requires 2 parameters:

> ***eps***: *specifies how close points should be to each other to be considered a part of a cluster. 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.*

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 $\geq D + 1$. 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.*

Why should we use DBSCAN?

The DBSCAN algorithm should be used to find associations and structures in data that are hard to find manually but that can be relevant and useful to find patterns and predict trends.

Clustering methods are usually used in biology, medicine, social sciences, archaeology, marketing, characters recognition, management systems and so on.

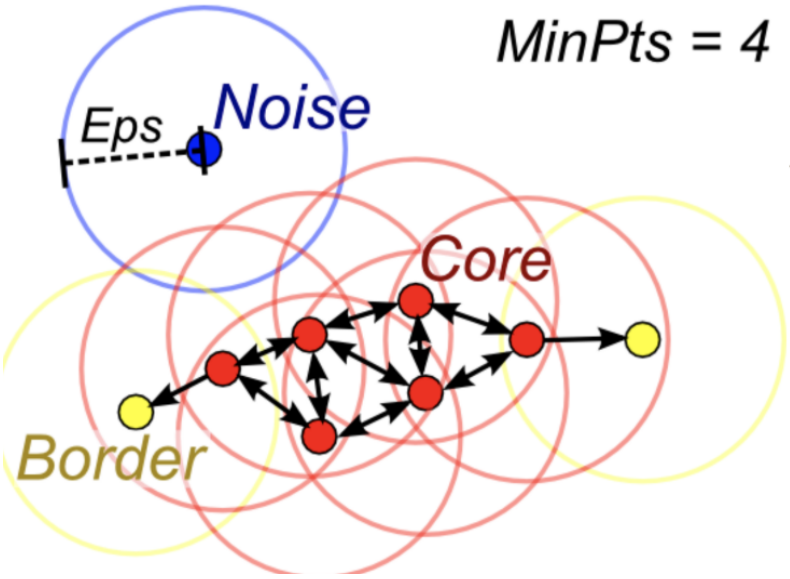
Let's think in a practical use of DBSCAN. Suppose we have an e-commerce and we want to improve our sales by recommending relevant products to our customers. We don't know exactly what our customers are looking for but based on a data set we can predict and recommend a relevant product to a specific customer. We can apply the DBSCAN to our data set (based on the e-commerce database) and find clusters based on the products that the users have bought. Using this clusters we can find similarities between customers, for example, the customer A have bought 1 pen, 1 book and 1 scissors and the customer B have bought 1 book and 1 scissors, then we can recommend 1 pen to the customer B. This is just a little example of use of DBSCAN, but it can be used in a lot of applications in several areas.

How can we easily implement it?

As I already wrote (tip: don't believe in everything I write) the DBSCAN is a well-known algorithm, therefore, you don't need to worry about implement it yourself. You can use one of the libraries/packages that can be found on the internet. Here is a list of links that you can find the DBSCAN implementation: Matlab, R, R, Python, Python.

Back to **DBSCAN. DBSCAN** is a clustering method that is used in machine learning to separate clusters of high density from clusters of low density. 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:



Red: Core Points

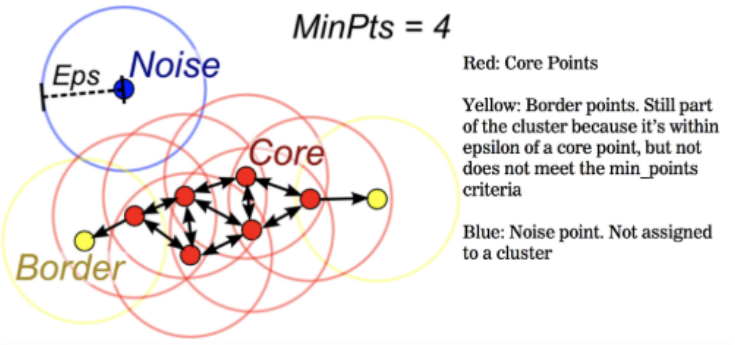
Yellow: Border points. Still part of the cluster because it's within epsilon of a core point, but not does not meet the min_points criteria

Blue: Noise point. Not assigned to a cluster

Going through the aforementioned 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 lets assume it was one of the red 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 parameters and specifications that we need to give DBSCAN before it does its work. The two parameters we need to specify are as such:

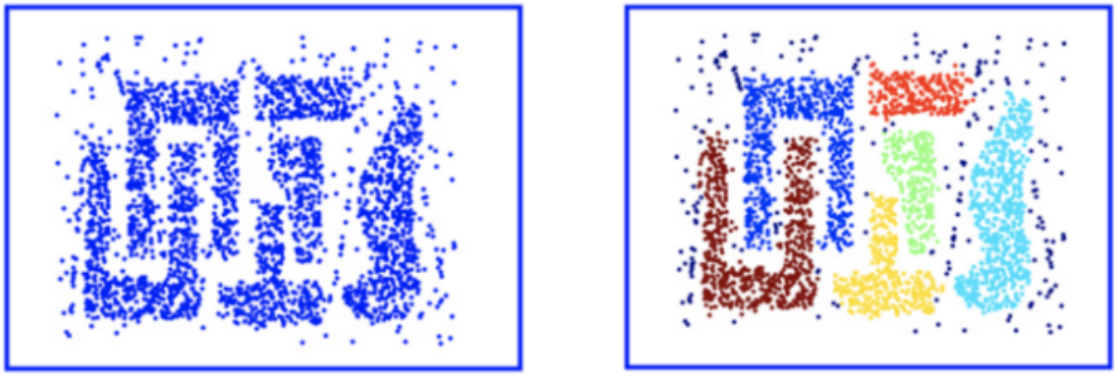
> *What is the minimum number of data points needed to determine a single cluster?*
>
> *How far away can one point be from the next point within the same cluster?*



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.

You will also notice that the Blue point in the graphic is not contained within any cluster. DBSCAN does *NOT* necessarily categorize every data point, and is therefore terrific with handling outliers in the dataset. Lets examine the graphic below:



The left image depicts a more traditional clustering method that does not account for multi-dimensionality. Whereas the right image shows how DBSCAN can contort the data into different shapes and dimensions in order to find similar clusters.

The left image depicts a more traditional clustering method, such as K-Means, that does not account for multi-dimensionality. Whereas the right

Means, that does not account for multi-dimensionality. Whereas the right image shows how DBSCAN can contort the data into different shapes and dimensions in order to find similar clusters. We also notice in the right image, that the points along the outer edge of the dataset are not classified, suggesting they are outliers amongst the data.

Advantages of DBSCAN:

- Is great at separating clusters of high density versus clusters of low density within a given dataset.
- Is great with handling outliers within the dataset.

Disadvantages of DBSCAN:

- Does not work well when dealing with clusters of varying densities. While DBSCAN is great at separating *high* density clusters from *low* density clusters, DBSCAN struggles with clusters of similar density.
- Struggles with high dimensionality data. I know, this entire article I have stated how DBSCAN is great at contorting the data into different dimensions and shapes. However, DBSCAN can only go so far, if given data with too many dimensions, DBSCAN suffers

DBSCAN Implementation in Python

> *1. Assigning the data as our X values*

```
# setting up our data to cluster
X = data


# WE MUST SCALE AND STARDARDIZE THE DATA
X = StandardScaler().fit_transform(X)
```

Remember, since this is unsupervised learning, we do not have any clear y values to set.

> *2. Instantiating our DBSCAN Model. In the code below, epsilon = 3 and min_samples is the minimum number of points needed to constitute a cluster.*

```
# setting up DBSCAN
dbscan = DBSCAN(eps = 3, min_samples = 4)

# fitting model
model = dbscan.fit(X)
```

> *3. Storing the labels formed by the DBSCAN*

```
labels = model.labels_
```

> *4. Identifying which points make up our "core points"*

```
import numpy as np
from sklearn import metrics
```

```
# identifying the core samples
core_samples = np.zeros_like(labels, dtype = bool)

core_samples[dbscan.core_sample_indices_] = True
print(core_samples)
```

> *5. Calculating the number of clusters*

```
# declare the number of clusters:
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_clusters_
```

> *6. Computing the Silhouette Score*

```
print("Silhouette Coefficient: %0.3f" % metrics.silhouette_score(X, labels))
```

Metrics for Measuring DBSCAN's Performance:

Silhouette Score: The silhouette score is calculated utilizing the mean intra- cluster distance between points, AND the mean nearest-cluster distance. For instance, a cluster with a lot of data points very close to each other (high density) AND is far away from the next nearest cluster (suggesting the cluster is very unique in comparison to the next closest), will have a strong silhouette score. A silhouette score ranges from -1 to 1, with -1 being the worst score possible and 1 being the best score. Silhouette scores of 0 suggest overlapping clusters.

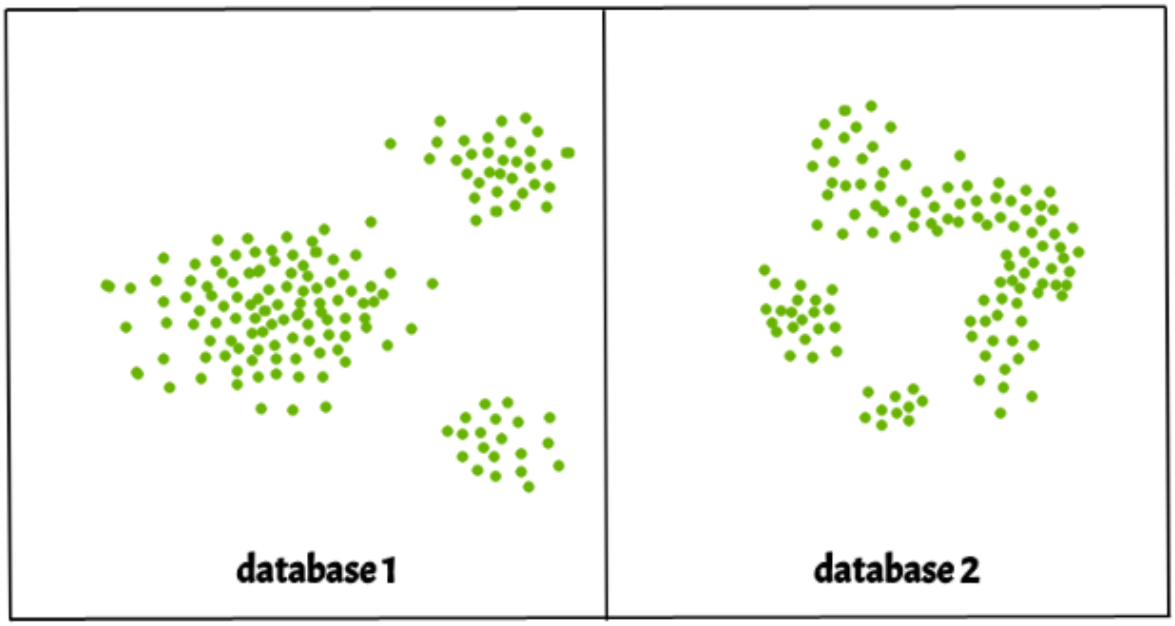
Inertia: Inertia measures the internal cluster sum of squares (sum of squares is the sum of all residuals). Inertia is utilized to measure how related clusters are amongst themselves, the lower the inertia score the better. **HOWEVER**, it is important to note that inertia heavily relies on the assumption that the clusters are convex (of spherical shape). DBSCAN does not necessarily divide data into spherical clusters, therefore inertia is not a good metric to use for evaluating DBSCAN models (*which is why I did not include inertia in the code above*). Inertia is more often used in other clustering methods, such as K-means clustering.

DBSCAN Clustering in ML | Density based clustering

Clustering analysis or simply Clustering is basically an Unsupervised learning method that divides the data points into a number of specific batches or groups, such that the data points in the same groups have similar properties and data points in different groups have different properties in some sense. It comprises of many different methods based on different evolution.
E.g. K-Means (distance between points), Affinity propagation (graph distance), Mean-shift (distance between points), DBSCAN (distance between nearest points), Gaussian mixtures (Mahalanobis distance to centers), Spectral clustering (graph distance) etc.

Fundamentally, all clustering methods use the same approach i.e. first we calculate similarities and then we use it to cluster the data points into groups or batches. Here we will focus on **Density-based spatial clustering of applications with noise** (DBSCAN) clustering method.

Clusters are dense regions in the data space, separated by regions of the lower density of points. The ***DBSCAN algorithm*** is based on this intuitive notion of "clusters" and "noise". The key idea is that for each point of a cluster, the neighborhood of a given radius has to contain at least a minimum number of points.



Why DBSCAN ?

Partitioning methods (K-means, PAM clustering) and hierarchical clustering work for finding spherical-shaped clusters or convex clusters. In other words, they are suitable only for compact and well-separated clusters. Moreover, they are also severely affected by the presence of noise and outliers in the data.

Real life data may contain irregularities, like –

i) Clusters can be of arbitrary shape such as those shown in the figure below.

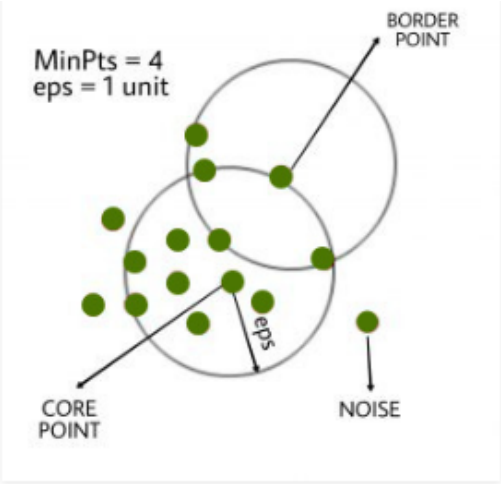
ii) Data may contain noise.

The figure below shows a data set containing nonconvex clusters and outliers/noises. Given such data, k-means algorithm has difficulties for identifying these clusters with arbitrary shapes.

DBSCAN algorithm requires two parameters –

1. **eps** : It defines the neighborhood around a data point i.e. if the distance between two points is lower or equal to 'eps' then they are considered as neighbors. If the eps value is chosen too small then large part of the data will be considered as outliers. If it is chosen very large then the clusters will merge and majority of the data points will be in the same clusters. One way to find the eps value is based on the ***k-distance graph***.
2. **MinPts**: Minimum number of neighbors (data points) within eps radius. Larger the dataset, the larger value of MinPts must be chosen. As a general rule, the minimum MinPts can be derived from the number of dimensions D in the dataset as, `MinPts >= D+1`. The minimum value of MinPts must be chosen at least 3.

> ***In this algorithm, we have 3 types of data points.***
>
> ***Core Point***: *A point is a core point if it has more than MinPts points within eps.*
> ***Border Point***: *A point which has fewer than MinPts within eps but it is in the neighborhood of a core point.*
> ***Noise or outlier***: *A point which is not a core point or border point.*

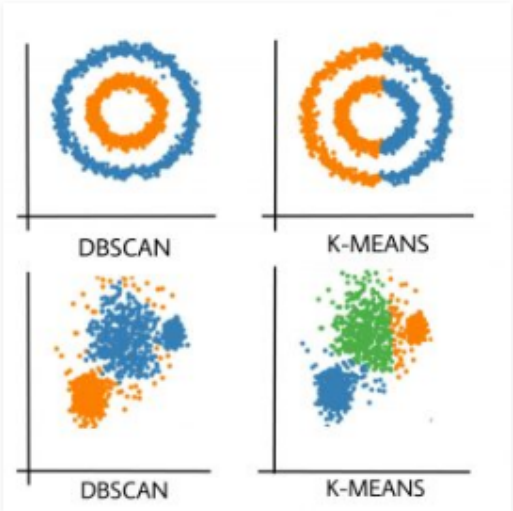


DBSCAN algorithm can be abstracted in the following steps –

1. Find all the neighbor points within eps and identify the core points or visited with more than MinPts neighbors.
2. For each core point if it is not already assigned to a cluster, create a new cluster.
3. Find recursively all its density connected points and assign them to the same cluster as the core point.
   A point *a* and *b* are said to be density connected if there exist a point *c* which has a sufficient number of points in its neighbors and both the points *a* and *b* are within the *eps distance*. This is a chaining process. So, if *b* is neighbor of *c*, *c* is neighbor of *d*, *d* is neighbor of *e*, which in turn is neighbor of *a* implies that *b* is neighbor of *a*.
4. Iterate through the remaining unvisited points in the dataset. Those points that do not belong to any cluster are noise.

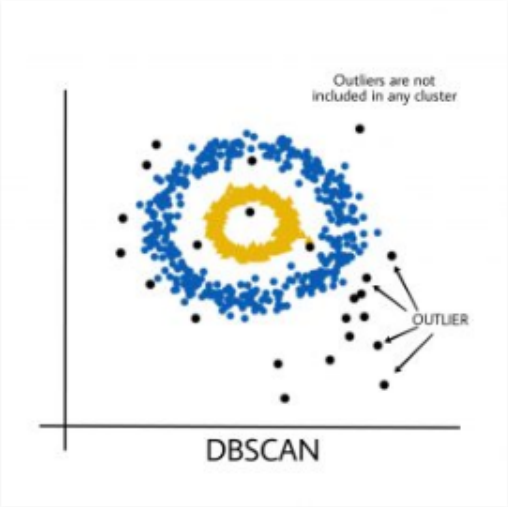
Disadvantage Of K-MEANS:

1. K-Means forms spherical clusters only. This algorithm fails when data is not spherical (i.e. same variance in all directions).



2. K-Means algorithm is sensitive towards outlier. Outliers can skew the clusters in K-Means in very large extent.

large extent.



3. K-Means algorithm requires one to specify the number of clusters a priory etc.

Basically, DBSCAN algorithm overcomes all the above-mentioned drawbacks of K-Means algorithm. DBSCAN algorithm identifies the dense region by grouping together data points that are closed to each other based on distance measurement.

1. Definitions within DBSCAN Algorithm:

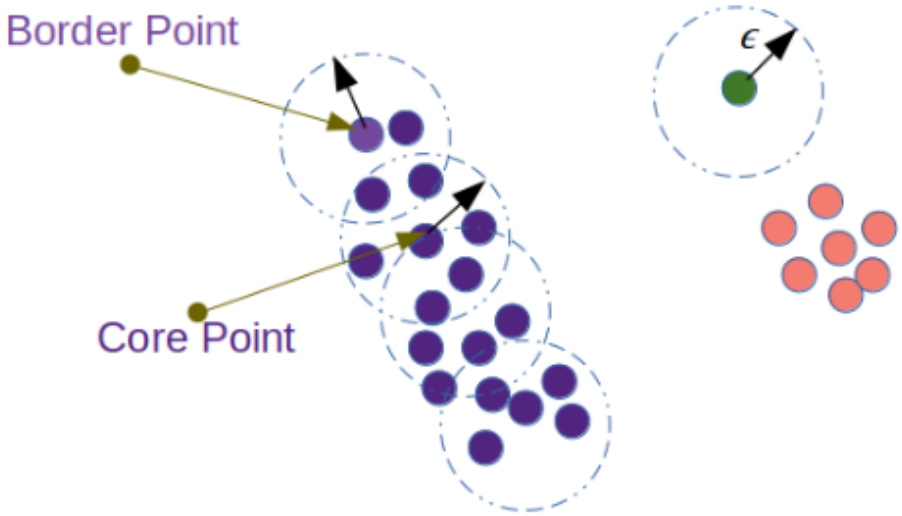
To understand DBSCAN in more detail, let's dive into it. ***The main concept of DBSCAN algorithm is to locate regions of high density that are separated from one another by regions of low density.*** So, how do we measure density of a region ? Below are the 2 steps —

- Density at a point P: Number of points within a circle of Radius *Eps (ϵ)* from point P.
- Dense Region: For each point in the cluster, the circle with radius ϵ contains at least minimum number of points (*MinPts*).

The Epsilon neighborhood of a point P in the database *D* is defined as (following the definition from Ester et.al.)

$$N(p) = \{q \in D \mid dist(p, q) \leq \epsilon\} \ldots . (1)$$

Following the definition of dense region, a point can be classified as a ***Core Point if*** $|N(p)| \geq MinPts$. The Core Points, as the name suggests, lie usually within the interior of a cluster. *A* ***Border Point*** *has fewer than MinPts within its ϵ-neighborhood (N), but it lies in the neighborhood of another core point.* ***Noise*** *is any data point that is neither core nor border point.* See the picture below for better understanding.



$$N_{Eps}(p) = \{q \in D \text{ such that } dist(p, q) \leq \epsilon\}$$

ϵ = 1 unit, MinPts = 7

Core and Border Points in a Database D. Green data point is Noise. (Source: Author, Real Reference [2])

One problem of this approach could be — ϵ-neighborhood of the border points contain significantly less number of points than the core points. Since *MinPts* is a parameter in the algorithm, setting it to a low value to include the border points in the cluster can cause problem to eliminate the noise.

Here comes the concept of *density-reachable and density-connected points*.

Directly Density Reachable: Data-point *a* is directly density reachable from a point *b* if —

1. $|N(b)| \geq MinPts$; *i.e. b is a core point.*
2. $a \in N(b)$ *i.e. a is in the epsilon neighborhood of b.*

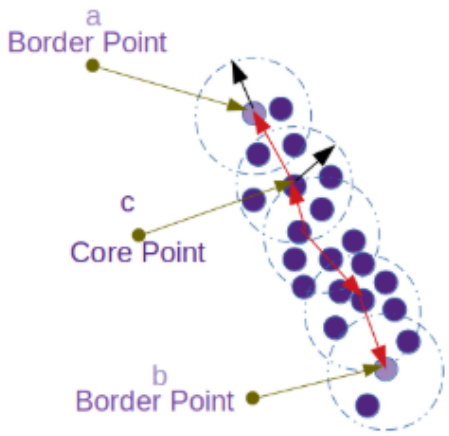
Considering a border point and a core point, we can understand that notion of directly density reachable is not symmetric, because even though the core point falls in the epsilon neighborhood of border point, the border point doesn't have enough *MinPts,* and thus fail to satisfy both conditions.

Density Reachable: Point *a* is density reachable from a point b with respect to ϵ and *MinPts*, if —

Density reachable is transitive in nature but, just like direct density reachable, it is not symmetric.

Density Connected: There can be cases when 2 border points will belong to the same cluster but they don't share a specific core point, then we say that they are density connected if, there exists a common core point, from which these borders points are density reachable. As you can understand that density connectivity is symmetric. Definition from the Ester et.al. paper is given below —

"A point ***a*** *is density connected to a point* ***b*** *with respect to* ϵ *and MinPts, if there is a point* ***c*** *such that, both* ***a*** *and* ***b*** *are density reachable from* ***c*** *w.r.t. to* ϵ *and MinPts."*



a, b are Density Reachable from a core point c.

a, b are called Density Connected points.

Two border points a, b are density connected through the core point c.

. . .

2. Steps of DBSCAN Algorithm:

With the definitions above, we can go through the steps of DBSCAN algorithm as below —

1. The algorithm starts with an arbitrary point which has not been visited and its neighborhood information is retrieved from the ϵ parameter.
2. If this point contains *MinPts* within ϵ neighborhood, cluster formation starts. Otherwise the point is labeled as noise. This point can be later found within the ϵ neighborhood of a different point and, thus can be made a part of the cluster. Concept of density reachable and density connected points are important here.
3. If a point is found to be a core point then the points within the ϵ neighborhood is also part of the cluster. So all the points found within ϵ neighborhood are added, along with their own ϵ neighborhood, if they are also core points.
4. The above process continues until the density-connected cluster is completely found.
5. The process restarts with a new point which can be a part of a new cluster or labeled as noise.

From the definitions and algorithm steps above, you can guess *two of the biggest drawbacks of DBSCAN algorithm*.

- If the database has data points that form clusters of varying density, then DBSCAN fails to cluster the data points well, since the clustering depends on ϵ and *MinPts* parameter, they cannot be chosen separately for all clusters.
- If the data and features are not so well understood by a domain expert then, setting up ϵ and *MinPts* could be tricky and, may need comparisons for several iterations with different values of ϵ and *MinPts*.

==========

Here's a list of advantages of KMeans and DBScan:

KMeans is much faster than DBScan

DBScan doesn't need number of clusters

Here's a list of disadvantages of KMeans and DBScan:

K-means need the number of clusters hidden in the dataset

DBScan doesn't work well over clusters with different densities

DBScan needs a careful selection of its parameters

==========

What is the difference between K-MEAN and density based clustering algorithm (DBSCAN)? Density based clustering algorithm has played a vital role in finding non linear shapes structure based on the density. Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is most widely used density based algorithm. It uses the concept of density reachability and density connectivity. On the other hand, K-means (MacQueen, 1967) is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k centroids, one for each cluster. These centroids shoud be placed in a cunning way because of different location causes different result.

k-means clustering is a method of vector quantization originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

The problem is computationally difficult (NP-hard); however, there are efficient heuristic algorithms that are commonly employed and converge quickly to a local optimum. These are usually similar to the expectation-maximization algorithm for mixtures of Gaussian distributions via an iterative refinement approach employed by both algorithms. Additionally, they both use cluster centers to model the data; however, k-means clustering tends to find clusters of comparable spatial extent, while the expectation-maximization mechanism allows clusters to have different shapes. (DBSCAN) Density-based spatial clustering of applications with noise is a data clustering algorithm proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu in 1996. It is a density-based clustering algorithm because it finds a number of clusters starting from the estimated density distribution of corresponding nodes. DBSCAN is one of the most common clustering algorithms and also most cited in scientific literature. OPTICS can be seen as a generalization of DBSCAN to multiple ranges, effectively replacing the \varepsilon parameter with a maximum search radius.

Advantages:

DBSCAN does not require one to specify the number of clusters in the data a priori, as opposed to k-means. DBSCAN can find arbitrarily shaped clusters. It can even find a cluster completely surrounded by (but not connected to) a different cluster. Due to the MinPts parameter, the so-called single-link effect (different clusters being connected by a thin line of points) is reduced. DBSCAN has a notion of noise. DBSCAN requires just two parameters and is mostly insensitive to the ordering of the points in the database. (However, points sitting on the edge of two different clusters might swap cluster membership if the ordering of the points is changed, and the cluster assignment is unique only up to isomorphism.) DBSCAN is designed for use with databases that can accelerate region queries, e.g. using an R* tree.

Disadvantages:

The quality of DBSCAN depends on the distance measure used in the function regionQuery(P,\varepsilon). The most common distance metric used is Euclidean distance. Especially for high-dimensional data, this metric can be rendered almost useless due to the so-called "Curse of dimensionality", making it difficult to find an appropriate value for \varepsilon. This effect, however, is also present in any other algorithm based on Euclidean distance. DBSCAN cannot cluster data sets well with large differences in densities, since the minPts-\varepsilon combination cannot then be chosen appropriately for all clusters.

A list of 10 of the more popular algorithms is as follows:

Affinity Propagation Agglomerative Clustering BIRCH DBSCAN K-Means Mini-Batch K-Means Mean Shift OPTICS Spectral Clustering Mixture of Gaussians

Cluster Evaluation

21. What are the aspects of cluster validation?

There are multiple methods to understand the goodness of a cluster. I presume you mean the same by validity of a cluster. They can be categorized into 3, External measures, Internal measures and relative measures. External measures are applicable when there is prior knowledge about the data. This situation is not so common. But there are measures like Matching based measures, Entropy based measures, Pairwise measures. Internal measures are derived from the data itself. You have Beta CV measures, Normalized Cut and Modularity indices. Relative measures are when you compare clusters by modifying the parameters of the algorithm. Simple measures like Similarity measures, Dissimilarity measures, Dissimilarity matrix are also a means to understand the goodness of the cluster. Distance functions you use in these cases are dependent on the type of data, whether numerical, assymetric binary variables, symmetric binary variables, vector, categorical, ratio, ordinal variables.

22. What is a confusion matrix?

1. **Internal cluster validation**, which uses the internal information of the clustering process to evaluate the goodness of a clustering structure without reference to external information. It can be also used for estimating the number of clusters and the appropriate clustering algorithm without any external data.
2. **External cluster validation**, which consists in comparing the results of a cluster analysis to an externally known result, such as externally provided class labels. It measures the extent to which cluster labels match externally supplied class labels. Since we know the “true” cluster number in advance, this approach is mainly used for selecting the right clustering algorithm for a specific data set.
3. **Relative cluster validation**, which evaluates the clustering structure by varying different parameter values for the same algorithm (e.g.,: varying the number of clusters k). It’s generally used for determining the optimal number of clusters.

Internal measures for cluster validation

In this section, we describe the most widely used clustering validation indices. Recall that the goal of partitioning clustering algorithms (Part @ref(partitioning-clustering)) is to split the data set into clusters of objects, such that:

- the objects in the same cluster are similar as much as possible,
- and the objects in different clusters are highly distinct

✔ That is, we want the average distance within cluster to be as small as possible; and the average distance between clusters to be as large as possible.

Internal validation measures reflect often the **compactness**, the **connectedness** and the **separation** of the cluster partitions.

1. **Compactness** or cluster cohesion: Measures how close are the objects within the same cluster. A lower **within-cluster variation** is an indicator of a good compactness (i.e., a good clustering). The different indices for evaluating the compactness of clusters are base on distance measures such as the cluster-wise within average/median distances between observations.
2. **Separation**: Measures how well-separated a cluster is from other clusters. The indices used as separation measures include:
   - distances between cluster centers
   - the pairwise minimum distances between objects in different clusters
3. **Connectivity**: corresponds to what extent items are placed in the same cluster as their nearest neighbors in the data space. The connectivity has a value between 0 and infinity and should be minimized.

Generally most of the indices used for internal clustering validation combine compactness and separation measures as follow:

$$Index = \frac{(\alpha \times Separation)}{(\beta \times Compactness)}$$

Where α and β are weights.

✔ In this section, we'll describe the two commonly used indices for assessing the goodness of clustering: the **silhouette width** and the **Dunn index**. These internal measure can be used also to determine the optimal number of clusters in the data.

Silhouette coefficient

The silhouette analysis measures how well an observation is clustered and it estimates the **average distance between clusters**. The silhouette plot displays a measure of how close each point in one cluster is to points in the neighboring clusters.

For each observation i, the silhouette width s_i is calculated as follows:

1. For each observation i, calculate the average dissimilarity a_i between i and all other points of the cluster to which i belongs.
2. For all other clusters C, to which i does not belong, calculate the average dissimilarity $d(i, C)$ of i to all observations of C. The smallest of these $d(i, C)$ is defined as $b_i = \min_C d(i, C)$. The value of b_i can be seen as the dissimilarity between i and its "neighbor" cluster, i.e., the nearest one to which it does not belong.
3. Finally the silhouette width of the observation i is defined by the formula: $S_i = (b_i - a_i)/max(a_i, b_i)$.

Silhouette width can be interpreted as follow:

- Observations with a large S_i (almost 1) are very well clustered.
- A small S_i (around 0) means that the observation lies between two clusters.
- Observations with a negative S_i are probably placed in the wrong cluster.

Dunn index

The **Dunn index** is another internal clustering validation measure which can be computed as follow:

1. For each cluster, compute the distance between each of the objects in the cluster and the objects in the other clusters
2. Use the minimum of this pairwise distance as the inter-cluster separation (*min.separation*)
3. For each cluster, compute the distance between the objects in the same cluster.
4. Use the maximal intra-cluster distance (i.e maximum diameter) as the intra-cluster compactness
5. Calculate the *Dunn index* (D) as follow:

The simplified format the *eclust()* function is as follow:

```
eclust(x, FUNcluster = "kmeans", hc_metric = "euclidean", ...)
```

- **x:** numeric vector, data matrix or data frame
- **FUNcluster:** a clustering function including "kmeans", "pam", "clara", "fanny", "hclust", "agnes" and "diana". Abbreviation is allowed.
- **hc_metric:** character string specifying the metric to be used for calculating dissimilarities between observations. Allowed values are those accepted by the function *dist()* [including "euclidean", "manhattan", "maximum", "canberra", "binary", "minkowski"] and correlation based distance measures ["pearson", "spearman" or "kendall"]. Used only when FUNcluster is a hierarchical clustering function such as one of "hclust", "agnes" or "diana".
- **...:** other arguments to be passed to FUNcluster.

The function **eclust()** returns an object of class **eclust** containing the result of the standard function used (e.g., kmeans, pam, hclust, agnes, diana, etc.).

It includes also:

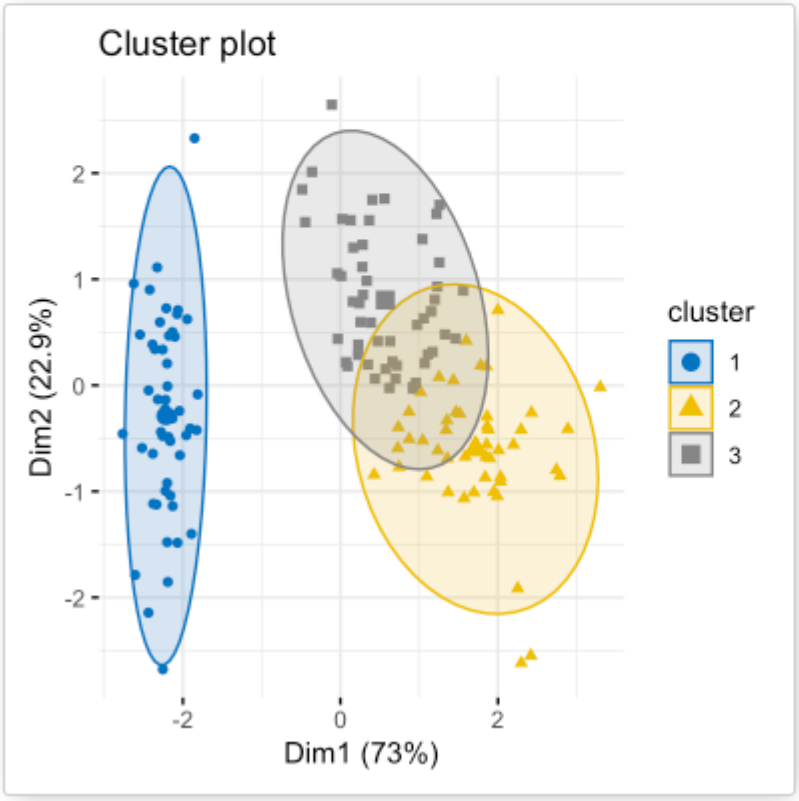
- **cluster:** the cluster assignment of observations after cutting the tree
- **nbclust:** the number of clusters
- **silinfo:** the silhouette information of observations
- **size:** the size of clusters
- **data:** a matrix containing the original or the standardized data (if stand = TRUE)
- **gap_stat:** containing gap statistics

To compute a partitioning clustering, such as k-means clustering with k = 3, type this:

a partitioning clustering, such as k-means clustering with k = 3, type this:

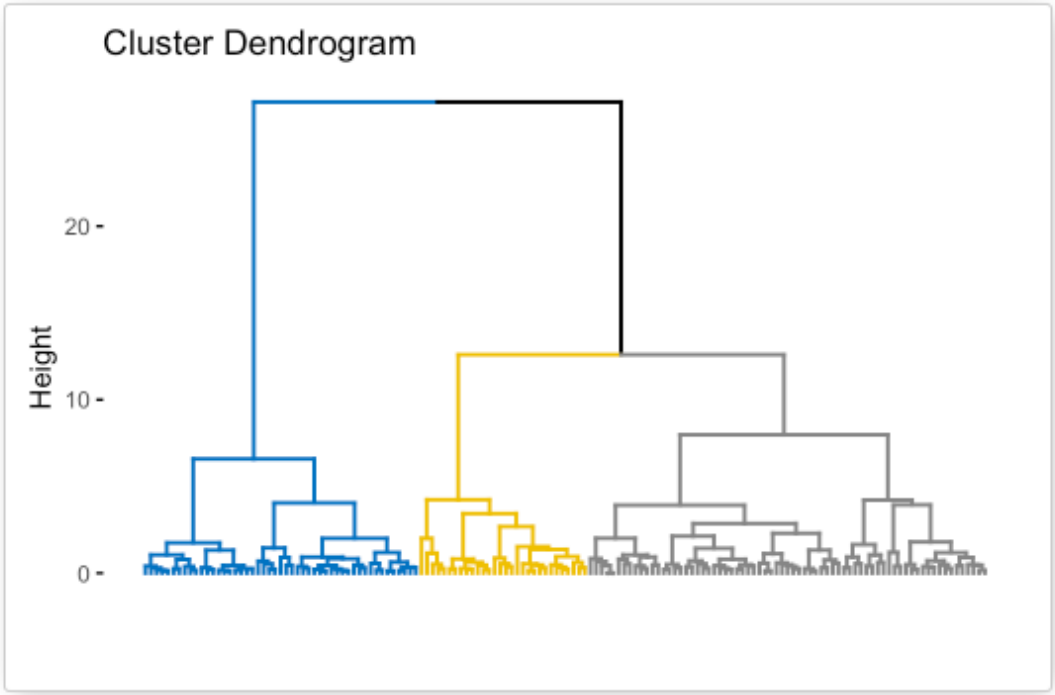
```
# K-means clustering
km.res <- eclust(df, "kmeans", k = 3, nstart = 25, graph = FALSE)
# Visualize k-means clusters
fviz_cluster(km.res, geom = "point", ellipse.type = "norm",
             palette = "jco", ggtheme = theme_minimal())
```

HOME LEARN TOPICS



```
# Hierchical clustering
hc.res <- eclust(df, "hclust", k = 3, hc_metric = "euclidean",
                 hc_method = "ward.D2", graph = FALSE)

# Visualize dendrograms
fviz_dend(hc.res, show_labels = FALSE,
          palette = "jco", as.ggplot = TRUE)
```



Clustering validation

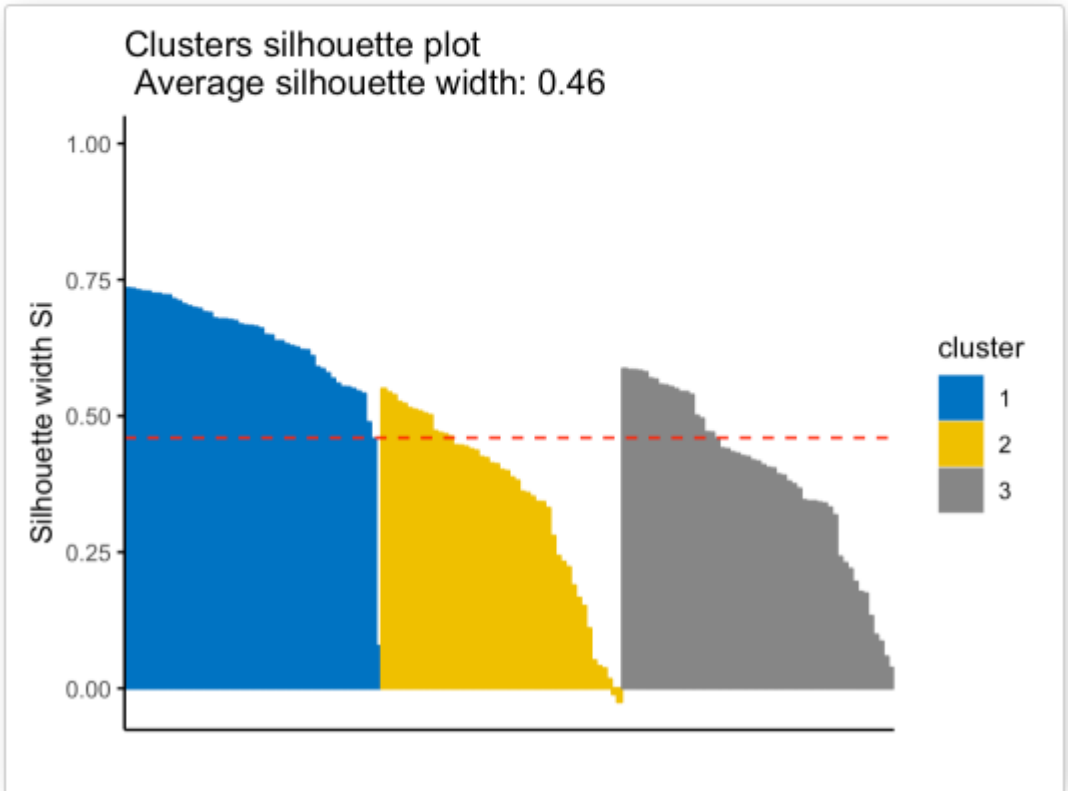
Silhouette plot

Recall that the silhouette coefficient (S_i) measures how similar an object i is to the the other objects in its own cluster versus those in the neighbor cluster. S_i values range from 1 to - 1:

- A value of S_i close to 1 indicates that the object is well clustered. In the other words, the object i is similar to the other objects in its group.
- A value of S_i close to -1 indicates that the object is poorly clustered, and that assignment to some other cluste would probably improve the overall results.

It's possible to draw silhouette coefficients of observations using the function *fviz_silhouette*() [*factoextra* package] which will also print a summary of the silhouette analysis output. To avoid this, you can use the option *print.summary = FALSE*.

```
##   cluster size ave.sil.width
## 1       1   50          0.64
## 2       2   47          0.35
## 3       3   53          0.39
```

- d: a distance object between cases as generated by the **dist()** function

- **clustering:** vector containing the cluster number of each observation
- **alt.clustering:** vector such as for clustering, indicating an alternative clustering

The function *cluster.stats()* returns a list containing many components useful for analyzing the intrinsic characteristics of a clustering:

- **cluster.number:** number of clusters
- **cluster.size:** vector containing the number of points in each cluster
- **average.distance**, **median.distance:** vector containing the cluster-wise within average/median distances
- **average.between:** average distance between clusters. We want it to be as large as possible
- **average.within:** average distance within clusters. We want it to be as small as possible
- **clus.avg.silwidths:** vector of cluster average silhouette widths. Recall that, the **silhouette width** is also an estimate of the average distance between clusters. Its value is comprised between 1 and -1 with a value of 1

- **within.cluster.ss**: a generalization of the within clusters sum of squares (k-means objective function), which is obtained if d is a Euclidean distance matrix.
- **dunn, dunn2**: Dunn index
- **corrected.rand, vi**: Two indexes to assess the similarity of two clustering: the corrected Rand index and Meila's VI

All the above elements can be used to evaluate the internal quality of clustering.

In the following sections, we'll compute the clustering quality statistics for k-means. Look at the **within.cluster.ss** (within clusters sum of squares), the **average.within** (average distance within clusters) and **clus.avg.silwidths** (vector of cluster average silhouette widths).

Confusion matrix:

n= number of points

m_i=points in *cluster i*

c_j=points in *class j*

n_{ij}= points in cluster i coming from cluster j

$p_{ij} = \frac{n_{ij}}{m_i}$= probability of element from cluster i to be assigned to class j

	Class 1	Class 2	Class 3	
Cluster 1	n_{11}	n_{12}	n_{13}	m_1
Cluster 2	n_{21}	n_{22}	n_{23}	m_2
Cluster 3	n_{31}	n_{32}	n_{33}	m_3
	c_1	c_2	c_3	n

	Class 1	Class 2	Class 3	
Cluster 1	p_{11}	p_{12}	p_{13}	m_1
Cluster 2	p_{21}	p_{22}	p_{23}	m_2
Cluster 3	p_{31}	p_{32}	p_{33}	m_3
	c_1	c_2	c_3	n

23. What is Jaccard's coefficient?

In [ ]:

24. What is Rand Index?

In [ ]:

25. What is the entropy of a cluster?

In [ ]:

26. Discuss the purity of a cluster.

In [ ]:

27. What are cohesion and compression?

In [ ]:

28. What are the steps for AWS deployment?

In [ ]:

29. What difficulties did you face while deploying to AWS?

In [ ]: