Unsupervised Learning and Data Clustering

Sanatan Mishra Follow
May 19, 2017 · 15 min read

A task involving machine learning may not be linear, but it has a number of well known steps:

- Problem definition.
- Preparation of Data.
- Learn an underlying model.
- Improve the underlying model by quantitative and qualitative evaluations.
- Present the model.

One good way to come to terms with a new problem is to work through identifying and defining the problem in the best possible way and learn a model that captures meaningful information from the data. While problems in Pattern Recognition and Machine Learning can be of various types, they can be broadly classified into three categories:

- Supervised Learning:
  The system is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs.
- Unsupervised Learning:
  No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).
- Reinforcement Learning:
  A system interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle or playing a game against an opponent). The system is provided feedback in terms of rewards and punishments as it navigates its problem space.

Between supervised and unsupervised learning is semi-supervised learning, where the teacher gives an incomplete training signal: a training set with some (often many) of the target outputs missing. We will focus on unsupervised learning and data clustering in this blog post.

Unsupervised Learning

In some pattern recognition problems, the training data consists of a set of input vectors x without any corresponding target values. The goal in such unsupervised learning problems may be to discover groups of similar examples within the data, where it is called *clustering*, or to determine how the data is distributed in the space, known as *density estimation*. To put forward in simpler terms, for a n-sampled space x1 to xn, true class labels are not provided for each sample, hence known as *learning without teacher*.

Issues with Unsupervised Learning:

- Unsupervised Learning is harder as compared to Supervised Learning tasks..
- How do we know if results are meaningful since no answer labels are available?
- Let the expert look at the results (external evaluation)
- Define an objective function on clustering (internal evaluation)

Why Unsupervised Learning is needed despite of these issues?

- Annotating large datasets is very costly and hence we can label only a few examples manually. Example: Speech Recognition
- There may be cases where we don't know how many/what classes is the data divided into. Example: Data Mining
- We may want to use clustering to gain some insight into the structure of the data before designing a classifier.

Unsupervised Learning can be further classified into two categories:

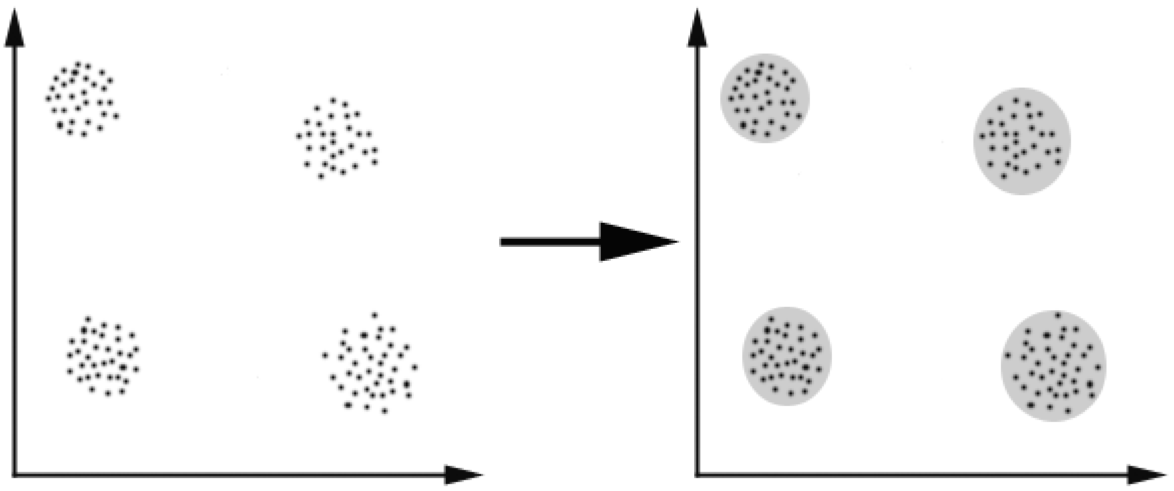
- *Parametric Unsupervised Learning*
  In this case, we assume a parametric distribution of data. It assumes that sample data comes from a population that follows a probability distribution based on a fixed set of parameters. Theoretically, in a normal family of distributions, all members have the same shape and are *parameterized* by mean and standard deviation. That means if you know the mean and standard deviation, and that the distribution is normal, you know the probability of any future observation. Parametric

Unsupervised Learning involves construction of Gaussian Mixture Models and using Expectation-Maximization algorithm to predict the class of the sample in question. This case is much harder than the standard supervised learning because there are no answer labels available and hence there is no correct measure of accuracy available to check the result.

- *Non-parametric Unsupervised Learning*
  In non-parameterized version of unsupervised learning, the data is grouped into clusters, where each cluster(hopefully) says something about categories and classes present in the data. This method is commonly used to model and analyze data with small sample sizes. Unlike parametric models, nonparametric models do not require the modeler to make any assumptions about the distribution of the population, and so are sometimes referred to as a distribution-free method.

What is 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.



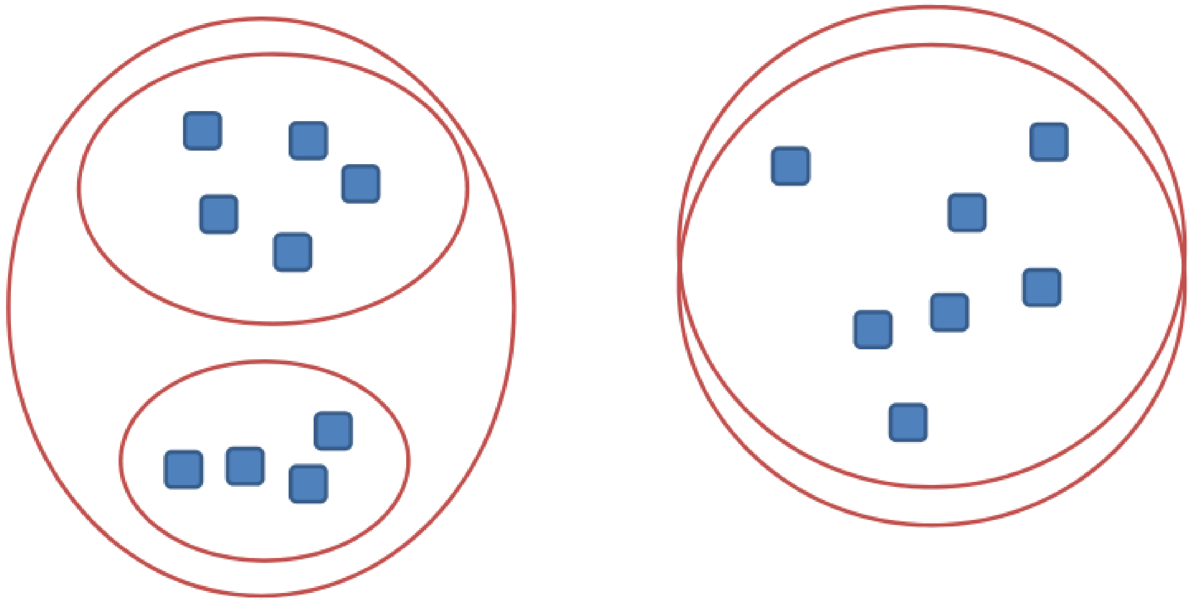
Distance-based clustering.

Given a set of points, with a notion of distance between points, grouping the points into some number of *clusters*, such that

- internal (within the cluster) distances should be small i.e members of clusters are close/similar to each other.
- external (intra-cluster) distances should be large i.e. members of different clusters are dissimilar.

The Goals of Clustering

The goal of clustering is to determine the internal 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 who should supply this criterion, in such a way that the result of the clustering will suit their needs.



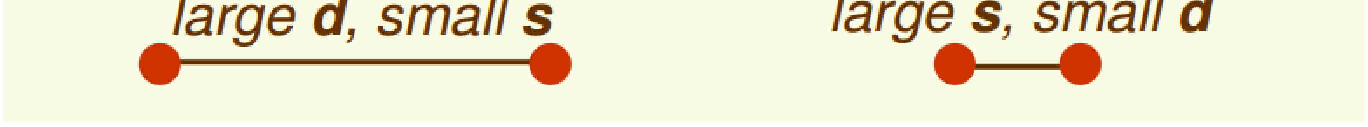
In the above image, how do we know what is the best clustering solution?

To find a particular clustering solution , we need to define the similarity measures for the clusters.

Proximity Measures

For clustering, we need to define a proximity measure for two data points. Proximity here means how similar/dissimilar the samples are with respect to each other.

- Similarity measure S(xi,xk): large if xi,xk are similar
- Dissimilarity(or distance) measure D(xi,xk): small if xi,xk are similar



There are various similarity measures which can be used.

- Vectors: Cosine Distance

$$s(\mathbf{x}, \mathbf{x}') = \frac{\mathbf{x}^t \mathbf{x}'}{\|\mathbf{x}\| \; \|\mathbf{x}'\|}$$

- Sets: Jaccard Distance

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

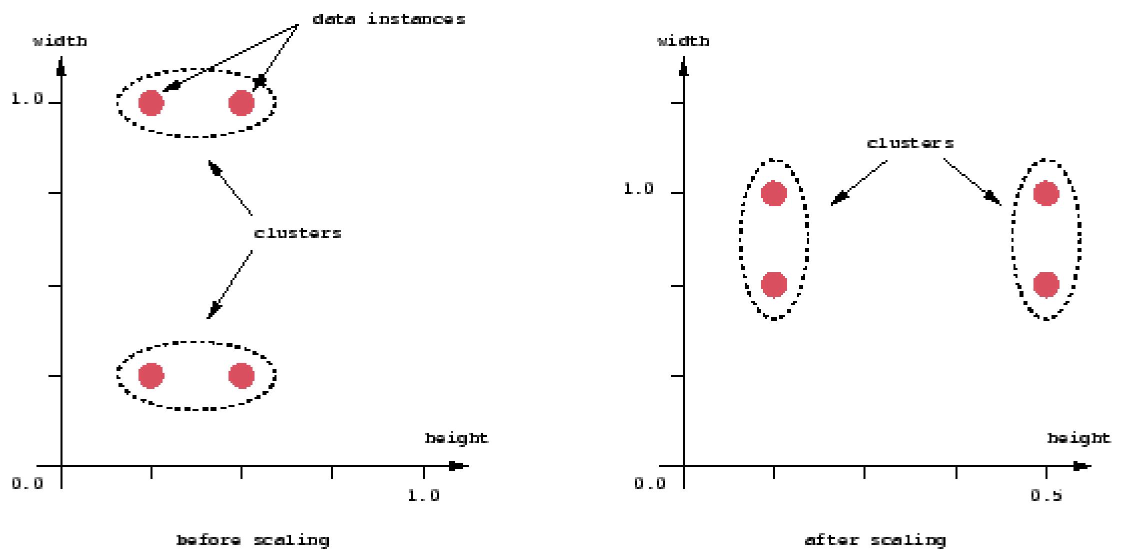
(If A and B are both empty, we define $J(A,B) = 1$.)

$$0 \leq J(A,B) \leq 1.$$

- Points: Euclidean Distance
  q=2

$$d(\mathbf{x}, \mathbf{x}') = \left(\sum_{k=1}^{d} |x_k - x'_k|^q \right)^{1/q},$$

A "good" proximity measure is VERY application dependent. The clusters should be invariant under the transformations "natural" to the problem. Also, while clustering it is not advised to normalize data that are drawn from multiple distributions.

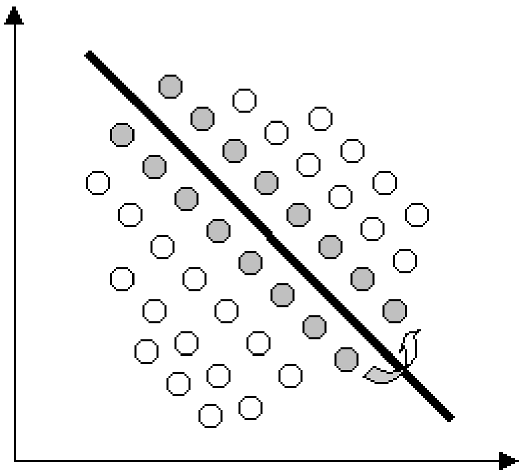


Clustering Algorithms

Clustering algorithms may be classified as listed below:

- Exclusive Clustering
- Overlapping Clustering
- Hierarchical Clustering
- Probabilistic Clustering

In the first case data are grouped in an exclusive way, so that if a certain data point belongs to a definite cluster then it could not be included in another cluster. A simple example of that is shown in the figure below, where the separation of points is achieved by a straight line on a bi-dimensional plane.



On the contrary, the second type, the overlapping clustering, uses fuzzy sets to cluster data, so that each point may belong to two or more clusters with different degrees of membership. In this case, data will be associated to an appropriate membership value.

A hierarchical clustering algorithm is based on the union between the two nearest clusters. The beginning condition is realized by setting every data point as a cluster. After a few iterations it reaches the final clusters wanted.

Finally, the last kind of clustering uses a completely probabilistic approach.

In this blog we will talk about four of the most used clustering algorithms:

- K-means
- Fuzzy K-means
- Hierarchical clustering

- Mixture of Gaussians

Each of these algorithms belongs to one of the clustering types listed above. While, K-means is an *exclusive clustering* algorithm, Fuzzy K-means is an *overlapping clustering* algorithm, Hierarchical clustering is obvious and lastly Mixture of Gaussians is a *probabilistic clustering* algorithm. We will discuss about each clustering method in the following paragraphs.

K-Means Clustering

K-means is one of the simplest unsupervised learning algorithms that solves 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 centres, one for each cluster. These centroids should be placed in a smart way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending, the first step is completed and an early groupage is done. At this point we need to re-calculate k new centroids as barycenters of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new centroid. A loop has been generated. As a result of this loop we may notice that the k centroids change their location step by step until no more changes are done. In other words centroids do not move any more.

Finally, this algorithm aims at minimizing an *objective function*, in this case a squared error function. The objective function

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - c_j \right\|^2$$

where

$$\left\| x_i^{(j)} - c_j \right\|^2$$

is a chosen distance measure between a data point xi and the cluster centre cj, is an indicator of the distance of the *n* data points from their respective cluster centres.

The algorithm is composed of the following steps:

- *Let X = {x1,x2,x3,……..,xn} be the set of data points and V = {v1,v2,…….,vc} be the set of centers.*

- *Randomly select 'c' cluster centers.*
- *Calculate the distance between each data point and cluster centers.*
- *Assign the data point to the cluster center whose distance from the cluster center is minimum of all the cluster centers.*
- *Recalculate the new cluster center using:*

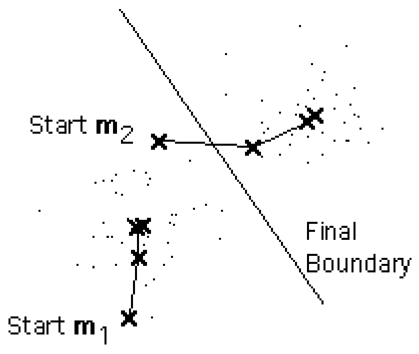
$$v_i = (1/c_i) \sum_{j=1}^{c_i} x_i$$

where, 'ci' represents the number of data points in ith cluster.

- *Recalculate the distance between each data point and new obtained cluster centers.*
- *If no data point was reassigned then stop, otherwise repeat from step 3).*

Although it can be proved that the procedure will always terminate, the k-means algorithm does not necessarily find the most optimal configuration, corresponding to the global objective function minimum. The algorithm is also significantly sensitive to the initial randomly selected cluster centres. The k-means algorithm can be run multiple times to reduce this effect.

K-means is a simple algorithm that has been adapted to many problem domains. As we are going to see, it is a good candidate for extension to work with fuzzy feature vectors.

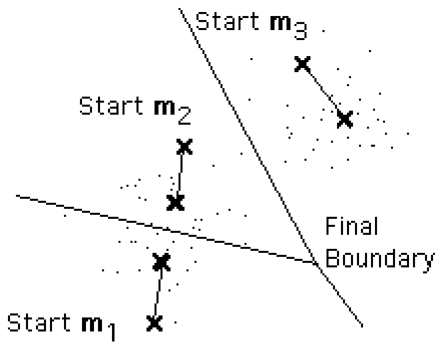


The k-means procedure can be viewed as a greedy algorithm for partitioning the n samples into k clusters so as to minimize the sum of the squared distances to the cluster centers. It does have some weaknesses:

- The way to initialize the means was not specified. One popular way to start is to randomly choose k of the samples.

- It can happen that the set of samples closest to **m**i is empty, so that **m**i cannot be updated. This is a problem which needs to be handled during the implementation, but is generally ignored.
- The results depend on the value of k and there is no optimal way to describe a best "k".

This last problem is particularly troublesome, since we often have no way of knowing how many clusters exist. In the example shown above, the same algorithm applied to the same data produces the following 3-means clustering. Is it better or worse than the 2-means clustering?



Unfortunately there is no general theoretical solution to find the optimal number of clusters for any given data set. A simple approach is to compare the results of multiple runs with different k classes and choose the best one according to a given criterion, but we need to be careful because increasing k results in smaller error function values by definition, but also increases the risk of overfitting.

Fuzzy K-Means Clustering

In fuzzy clustering, each point has a probability of belonging to each cluster, rather than completely belonging to just one cluster as it is the case in the traditional k-means. Fuzzy k-means specifically tries to deal with the problem where points are somewhat in between centers or otherwise ambiguous by replacing distance with probability, which of course could be some function of distance, such as having probability relative to the inverse of the distance. Fuzzy k-means uses a weighted centroid based on those probabilities. Processes of initialization, iteration, and termination are the same as the ones used in k-means. The resulting clusters are best analyzed as probabilistic distributions rather than a hard assignment of labels. One should realize that k-means is a special case of fuzzy k-means when the probability function used is simply 1 if the data point is closest to a centroid and 0 otherwise.

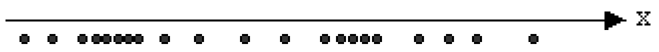
The fuzzy k-means algorithm is the following:

- **Assume** a fixed number of clusters *K*.
- **Initialization:** Randomly initialize the k-means *μk* associated with the clusters and compute the probability that each data point *Xi* is a member of a given cluster *K*, *P(PointXiHasLabelK|Xi,K)*.
- **Iteration:** Recalculate the centroid of the cluster as the weighted centroid given the probabilities of membership of all data points *Xi* :

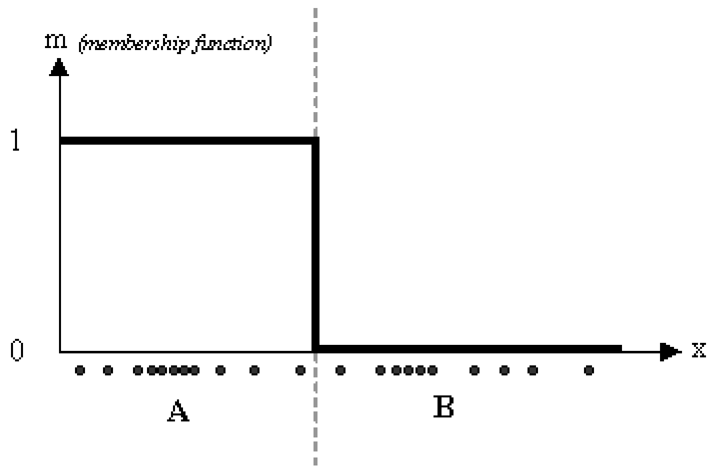
$$\mu_k(n+1) = \frac{\sum_{x_i \in k} x_i \times P(\mu_k|x_i)^b}{\sum_{x_i \in k} P(\mu_k|x_i)^b}$$

- **Termination**: Iterate until convergence or until a user-specified number of iterations has been reached (the iteration may be trapped at some local maxima or minima)

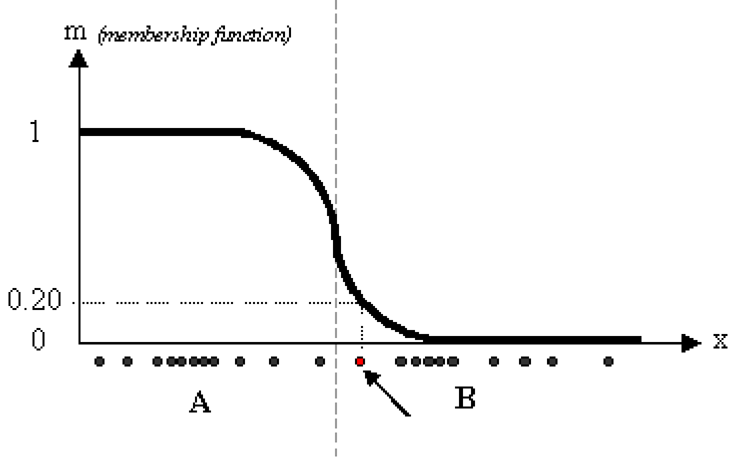
For a better understanding, we may consider this simple mono-dimensional example. Given a certain data set, suppose to represent it as distributed on an axis. The figure below shows this:



Looking at the picture, we may identify two clusters in proximity of the two data concentrations. We will refer to them using 'A' and 'B'. In the first approach shown in this tutorial — the k-means algorithm — we associated each data point to a specific centroid; therefore, this membership function looked like this:



In the Fuzzy k-means approach, instead, the same given data point does not belong exclusively to a well defined cluster, but it can be placed in a middle way. In this case, the membership function follows a smoother line to indicate that every data point may belong to several clusters with different extent of membership.

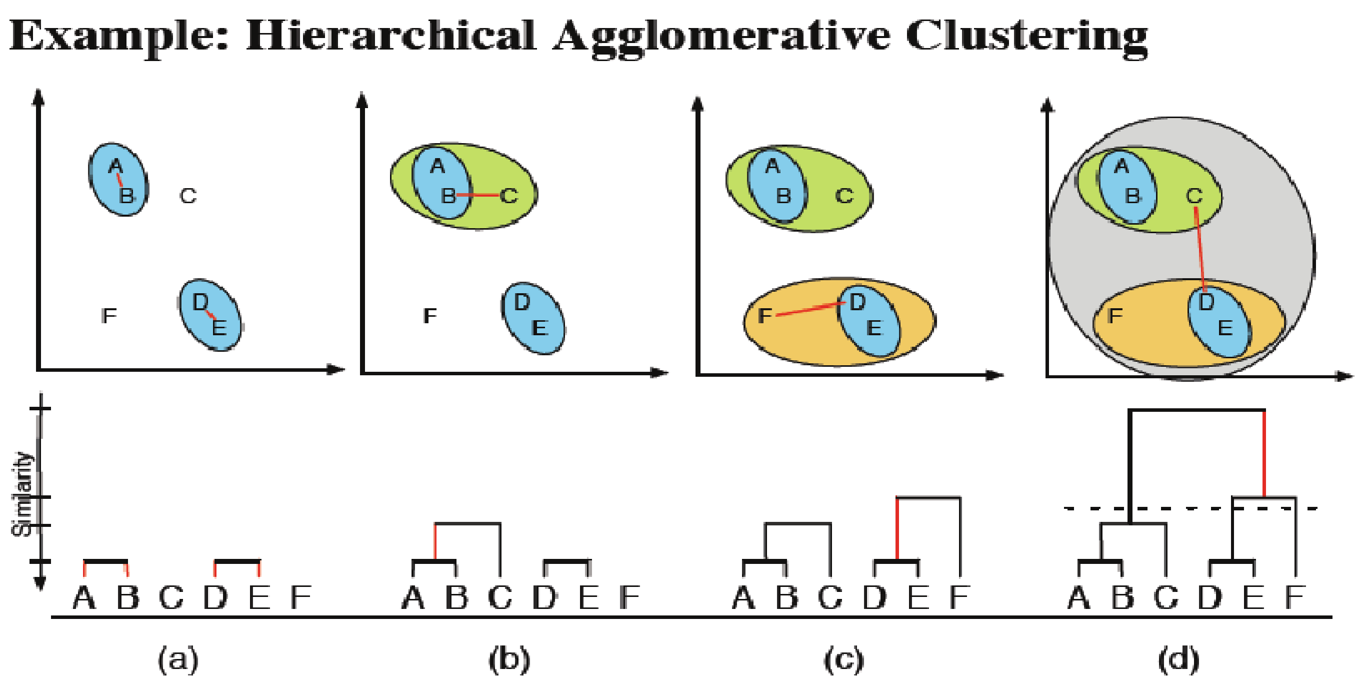


In the figure above, the data point shown as a red marked spot belongs more to the B cluster rather than the A cluster. The value 0.2 of 'm' indicates the degree of membership to A for such data point.

Hierarchical Clustering Algorithms

Given a set of N items to be clustered, and an N*N distance (or similarity) matrix, the basic process of hierarchical clustering is this:

- Start by assigning each item to a cluster, so that if you have N items, you now have N clusters, each containing just one item. Let the distances (similarities) between the clusters the same as the distances (similarities) between the items they contain.
- Find the closest (most similar) pair of clusters and merge them into a single cluster, so that now you have one cluster less.
- Compute distances (similarities) between the new cluster and each of the old clusters.
- Repeat steps 2 and 3 until all items are clustered into a single cluster of size N.



Clustering as a Mixture of Gaussians

There's another way to deal with clustering problems: a *model-based* approach, which consists in using certain models for clusters and attempting to optimize the fit between the data and the model.

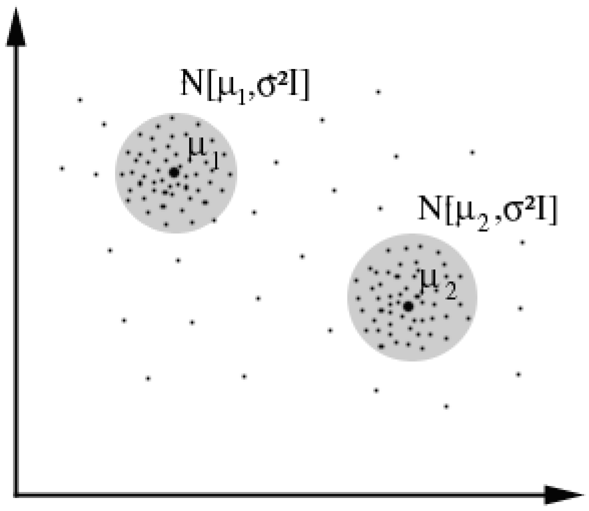
In practice, each cluster can be mathematically represented by a parametric distribution, like a Gaussian. The entire data set is therefore modelled by a *mixture* of these distributions.
A mixture model with high likelihood tends to have the following traits:

- component distributions have high "peaks" (data in one cluster are tight);
- the mixture model "covers" the data well (dominant patterns in the data are captured by component distributions).

Main advantages of model-based clustering:

- well-studied statistical inference techniques available;
- flexibility in choosing the component distribution;
- obtain a density estimation for each cluster;
- a "soft" classification is available.

Mixture of Gaussians
The most widely used clustering method of this kind is based on learning a *mixture of Gaussians*:



A mixture model is a mixture of k component distributions that collectively make a mixture distribution *f*(*x*):

$$f(x) = \sum_{k=1}^{K} \alpha_k f_k(x)$$

The *αk* represents the contribution of the *kth* component in constructing *f(x)*. In practice, parametric distribution (e.g. gaussians), are often used since a lot work has been done to understand their behaviour. If you substitute each *fk*(*x*) for a gaussian you get what is known as a gaussian mixture models (GMM).

The EM Algorithm

Expectation-Maximization assumes that your data is composed of multiple multivariate normal distributions (note that this is a *very* strong assumption, in particular when you fix the number of clusters!). Alternatively put, EM is an algorithm for maximizing a likelihood function when some of the variables in your model are unobserved (i.e. when you have latent variables).
You might fairly ask, if we're just trying to maximize a function, why don't we just use the existing machinery for maximizing a function. Well, if you try to maximize this by taking derivatives and setting them to zero, you find that in many cases the first-order conditions don't have a solution. There's a chicken-and-egg problem in that to solve for your model parameters you need to know the distribution of your unobserved data; but the distribution of your unobserved data is a function of your model parameters.

Expectation-Maximization tries to get around this by iteratively guessing a distribution for the unobserved data, then estimating the model parameters by maximizing something that is a lower bound on the actual likelihood function, and repeating until convergence:

The Expectation-Maximization algorithm

- Start with guess for values of your model parameters
- **E-step**: For each datapoint that has missing values, use your mo for the distribution of the missing data given your current guess of the model parameters and given the observed data (note that you are solving for a distribution for each missing value, not for the expected value). Now that we have a distribution for each missing value, we can calculate the *expectation* of the likelihood function with respect to the unobserved variables. If our guess for the model parameter was

correct, this expected likelihood will be the actual likelihood of our observed data; if the parameters were not correct, it will just be a lower bound.

- **M-step**: Now that we've got an expected likelihood function with no unobserved variables in it, maximize the function as you would in the fully observed case, to get a new estimate of your model parameters.
- Repeat until convergence.

Problems associated with clustering

There are a number of problems with clustering. Among them:

- dealing with large number of dimensions and large number of data items can be problematic because of time complexity;
- the effectiveness of the method depends on the definition of "distance" (for distance-based clustering). If an *obvious* distance measure doesn't exist we must "define" it, which is not always easy, especially in multidimensional spaces;
- the result of the clustering algorithm (that in many cases can be arbitrary itself) can be interpreted in different ways.

Possible Applications

Clustering algorithms can be applied in many fields, for instance:

- *Marketing*: finding groups of customers with similar behavior given a large database of customer data containing their properties and past buying records;
- *Biology*: classification of plants and animals given their features;
- *Insurance*: identifying groups of motor insurance policy holders with a high average claim cost; identifying frauds;
- *Earthquake studies*: clustering observed earthquake epicenters to identify dangerous zones;
- *World Wide Web*: document classification; clustering weblog data to discover groups of similar access patterns.

Machine Learning Clustering Data Science Unsupervised Learning Towards Data Science