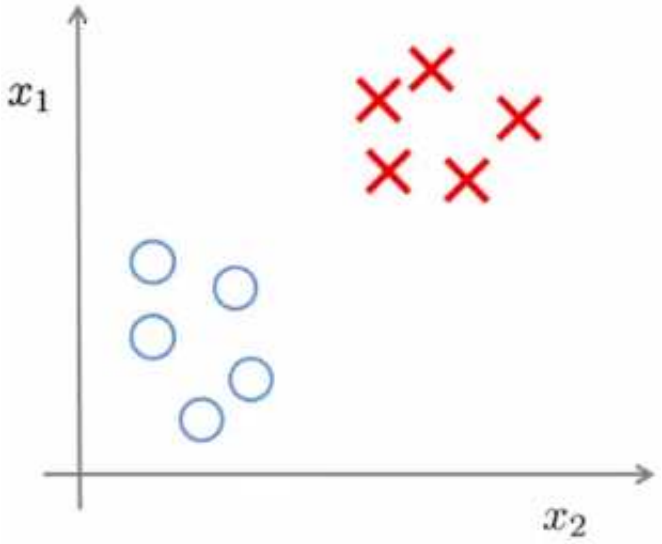
Unsupervised learning

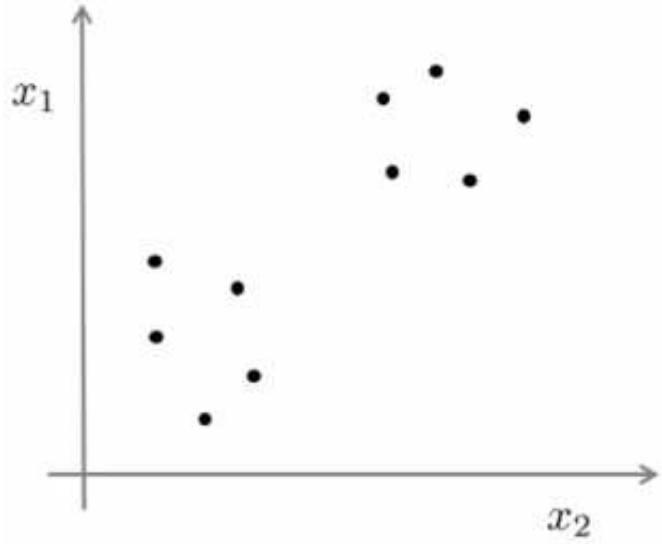
Supervised, classification:



Training set:

$\{\langle(x_1^1, x_2^1), c^1\rangle, \langle(x_1^2, x_2^2), c^2\rangle, ...\langle(x_1^N, x_2^N), c^N\rangle\}$

Unsupervised, **clustering**:



Training set:

$\{\langle(x_1^1, x_2^1)\rangle, \langle(x_1^2, x_2^2)\rangle, ...\langle(x_1^N, x_2^N)\rangle\}$

The k-means algorithm

The **k-means** algorithm offers a very simple, popular and effective clustering method. It is based on comparing distances and determining clusters represented by their geometric centers — **centroids**, minimizing a certain cost function.

The algorithm assumes that K — the number of clusters to be generated — is known. It repeats two steps: the labeling step and the centroids shift step.

The k-means algorithm:

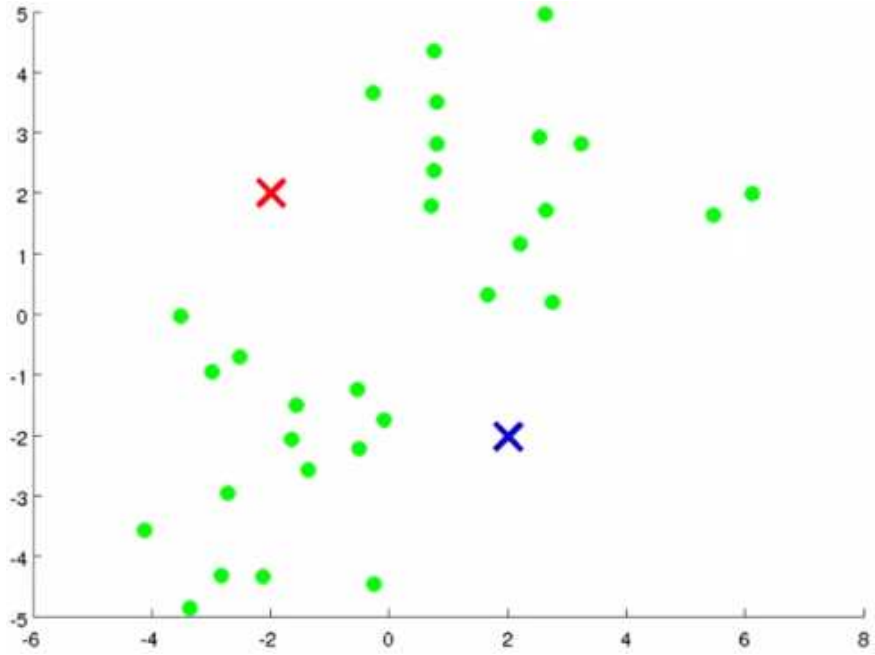
Step 0 (initialization): set the initial values of all K centroids

REPEAT {

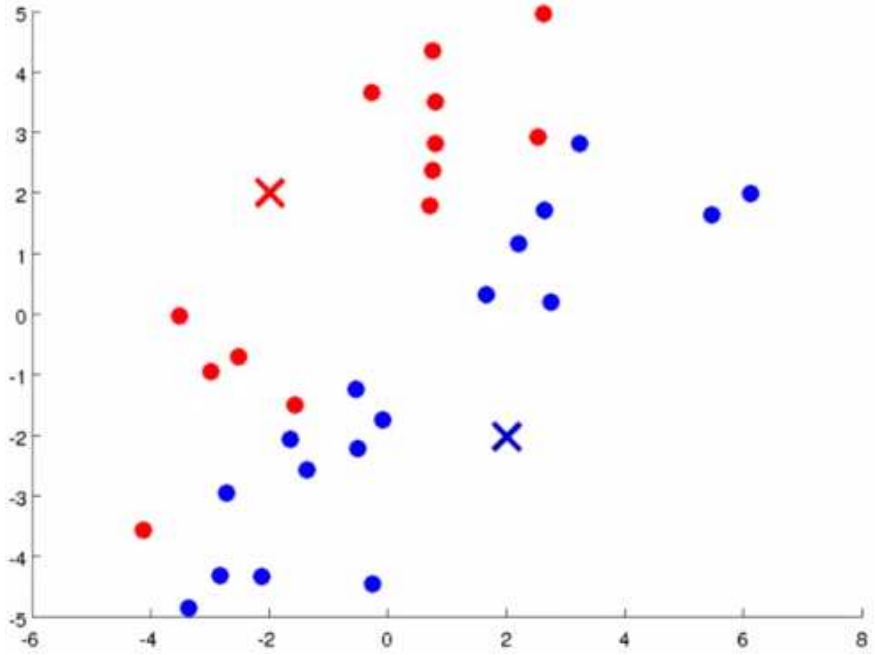
Step 1 (labeling): mark all samples with the label of the nearest centroid

Step 2 (centroids shift): move all centroids to the geometric centers of their clusters

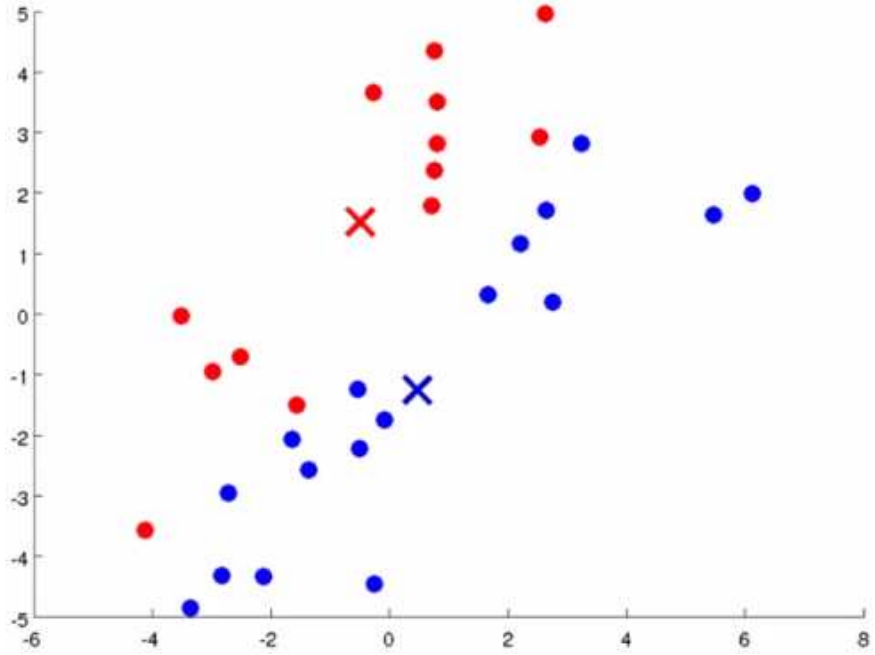
}



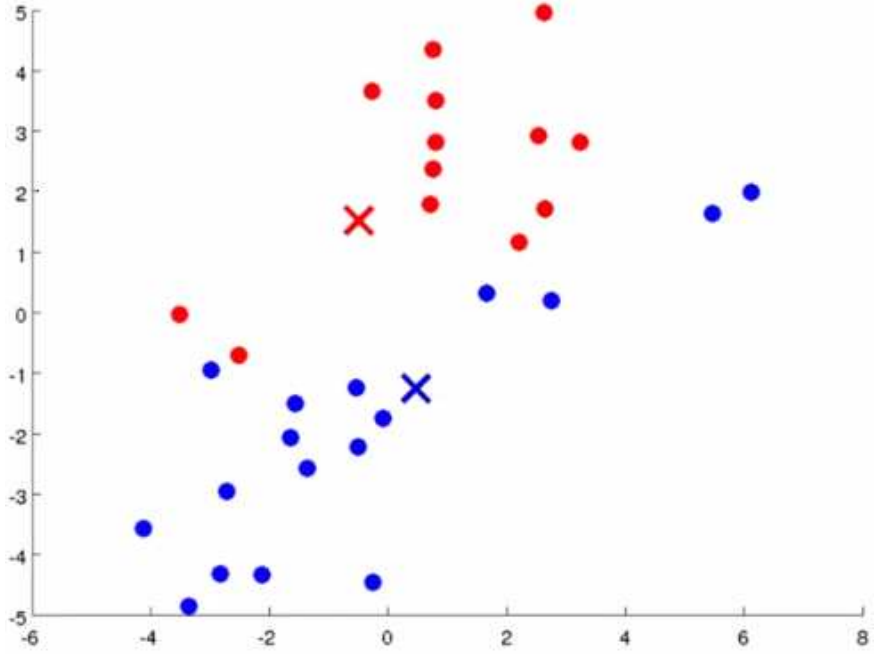
Step 0 (initialization)



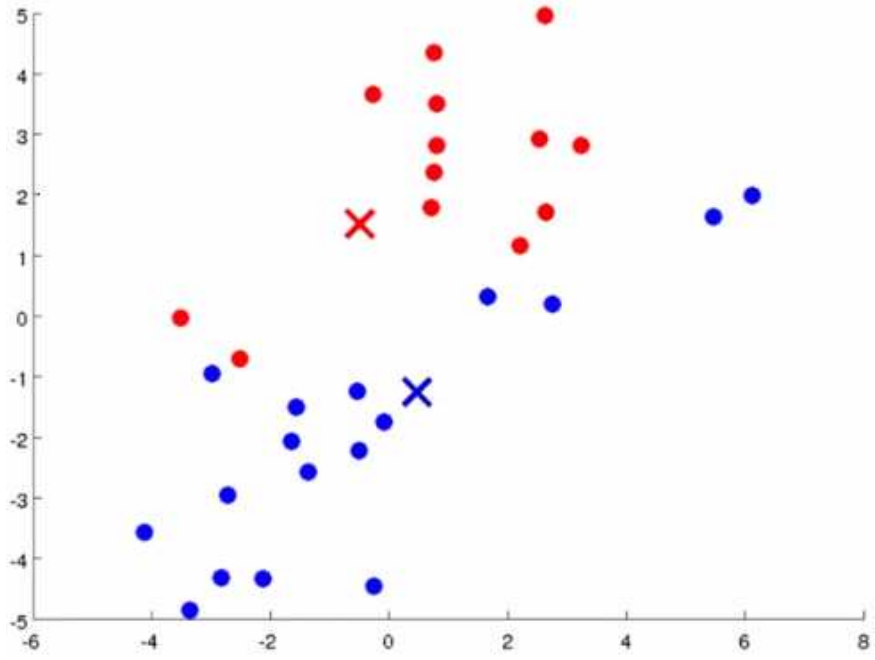
Step 1 (labeling)



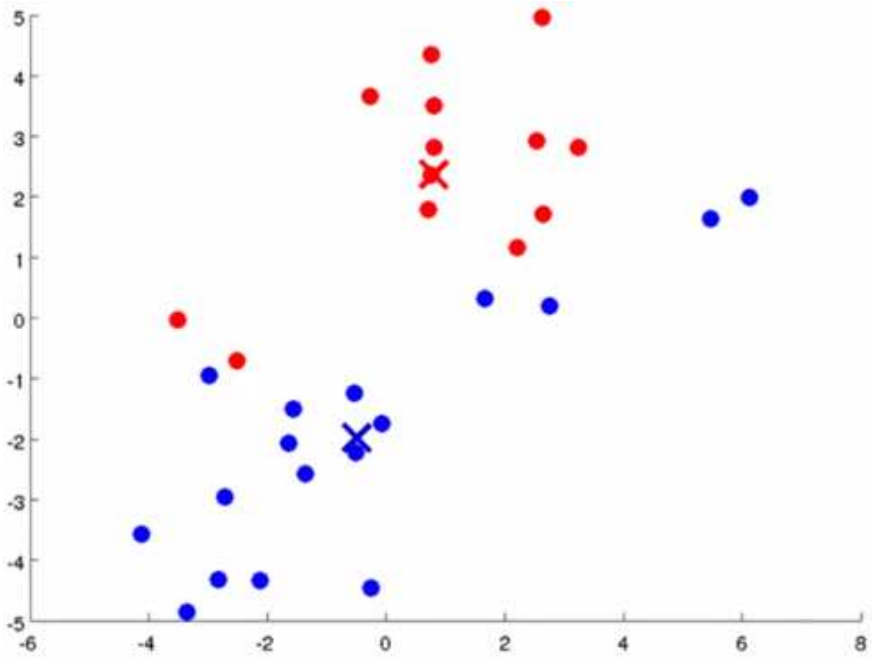
Step 2 (centroid shift)



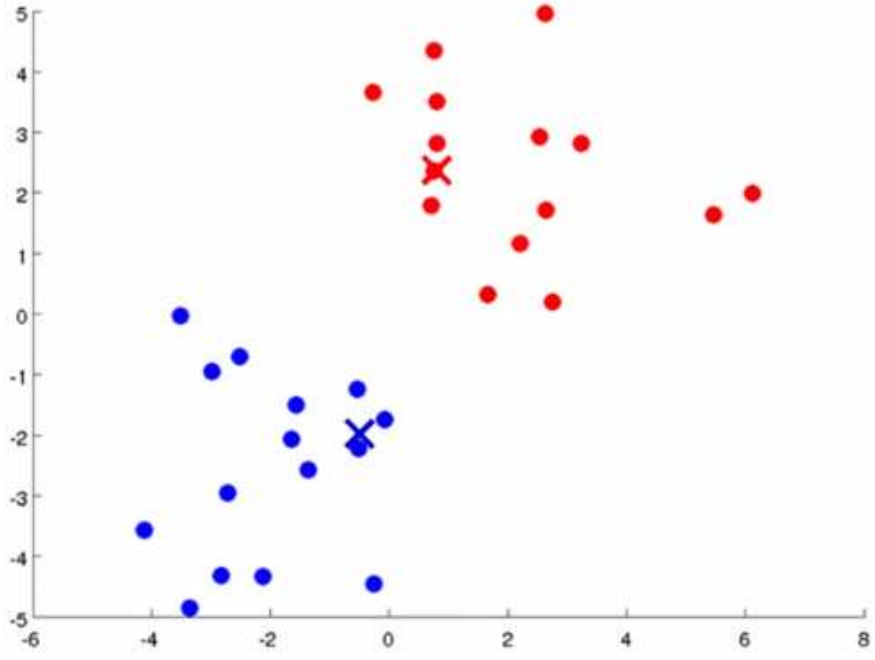
Step 1 (labeling)



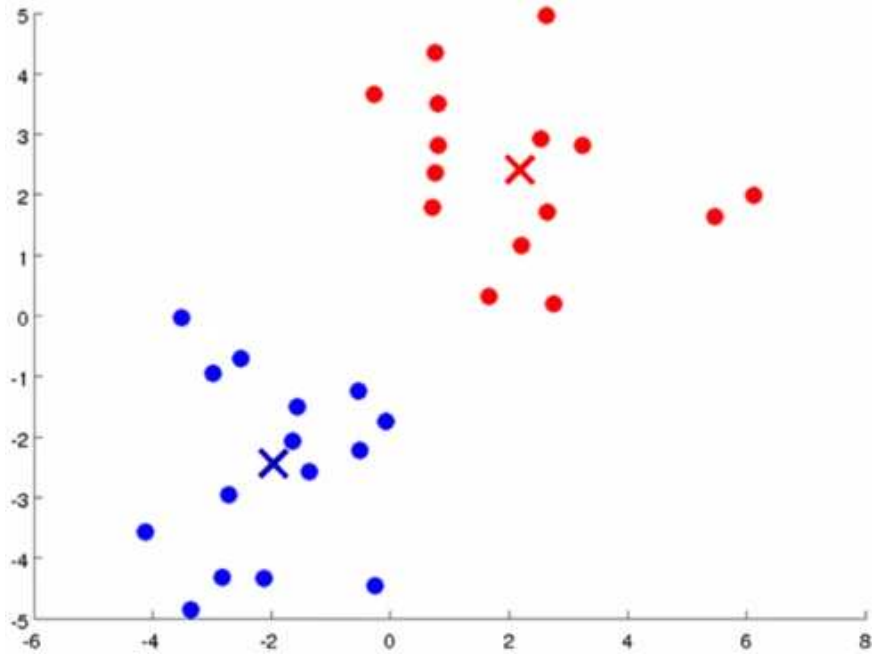
Step 1 (labeling)



Step 2 (centroid shift)



Step 1 (labeling)



Step 2 (centroid shift)

The k-means algorithm — the quality criterion

The k-means algorithm attempts to find the minimum of a certain cost function that is a measure of the quality of the generated set of clusters. This cost function is the weighted sum of the distances of all points to the cluster centroids.

We can observe that the first step of the algorithm (labeling) optimizes this cost function w.r.t. the mean distance, while keeping the centroids fixed.

The second step of the algorithm (centroid shift) optimizes the same function w.r.t. the position of the centroids, while preserving the current clusters.

The k-means algorithm — the distance measures

Various methods can be used to calculate the distance in the feature space:

Euclidean	$\sqrt{\Sigma_i (a_i - b_i)^2}$
Manhattan	$\Sigma_i \left\lvert a_i - b_i \right\rvert$
Max	$\max_i \left\lvert a_i - b_i \right\rvert$

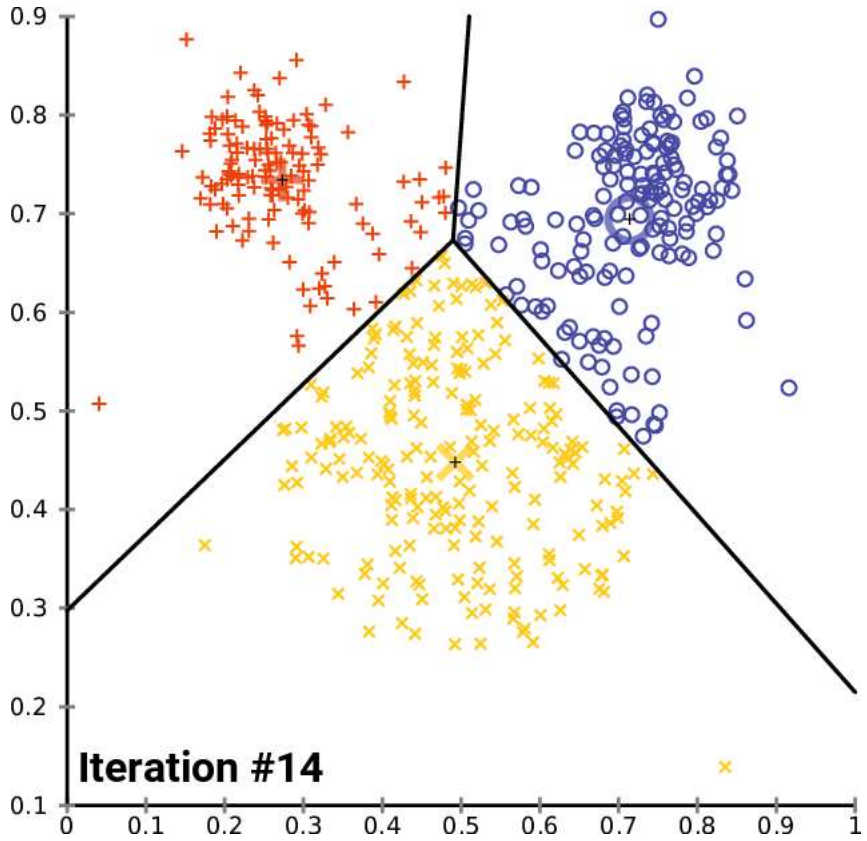
In general: due to the possible scale discrepancy, just as it is done in other methods based on distance calculation, individual coordinates should be scaled to calculate the distance in the space of features. The scaling factor can be the variance of the given attribute value on the training set.

Non-numeric data pose a special problem. In some cases, such as text strings, there are metrics such as the Hamming or Levenshtein distance.

The k-means algorithm — another example

An example from Wikipedia:

`https://upload.wikimedia.org/wikipedia/commons/e/ea/K-means_convergence.gif`



As we can see in the above example, the k-means algorithm does not always generate so intuitively correct results, as in the previous examples. There are a number of special cases that need to be considered, to obtain optimal results.

K-means special case — centroid with empty set

What to do when a centroid with an empty set is created during the operation of the algorithm?

Method 1: eliminate that centroid and continue, effectively (K-1) clusters.

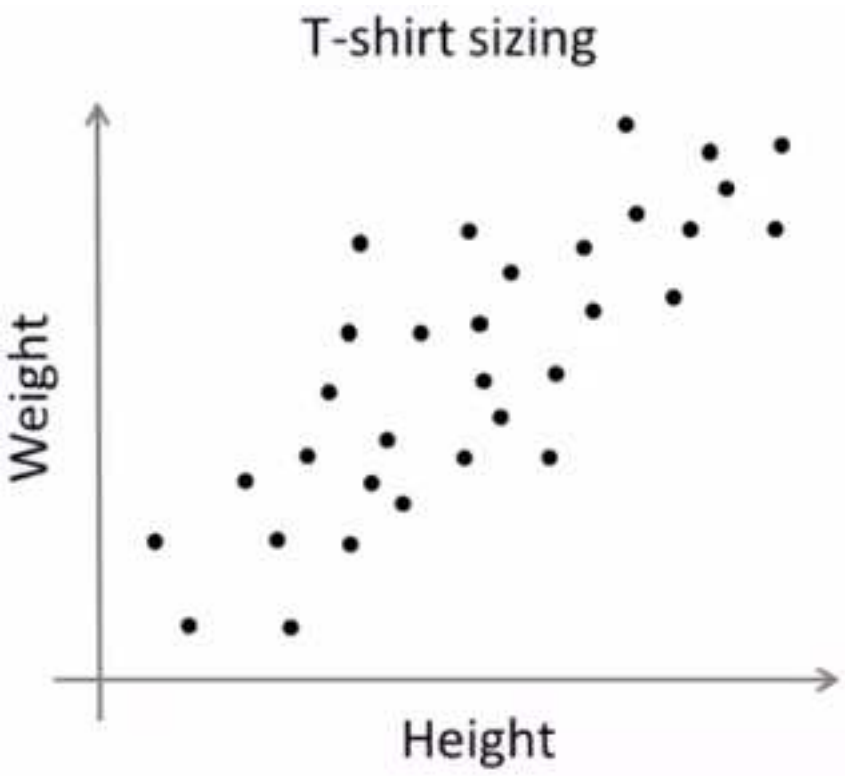
However, it is possible that the number of clusters is imposed, and we need to preserve it. Then:

Method 2: re-initialize the location of this centroid, and continue.

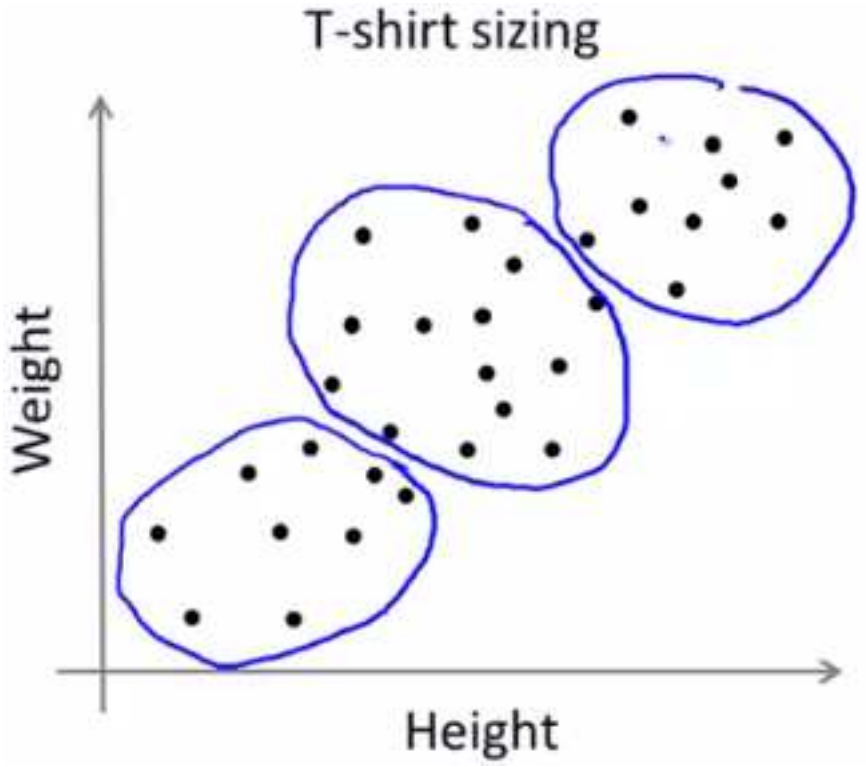
K-means special case — no cluster separation

Not always a set of samples breaks down naturally into clearly separated clusters. We may still want to group the data.

For example, the T-shirt manufacturer performed an anthropometric study to design well-fitting shirts in several sizes (eg: S, M, L):

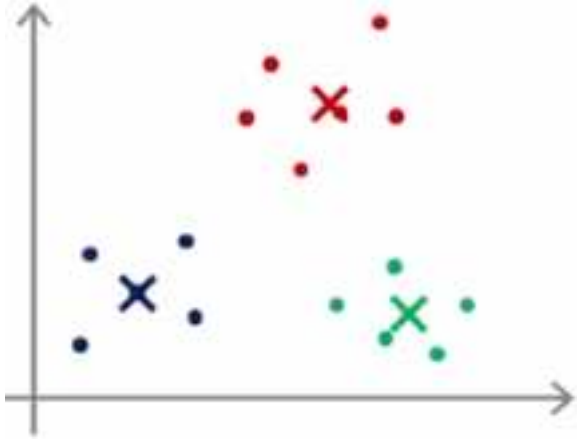


The algorithm still works correctly, finding the specified number of clusters based on distances:

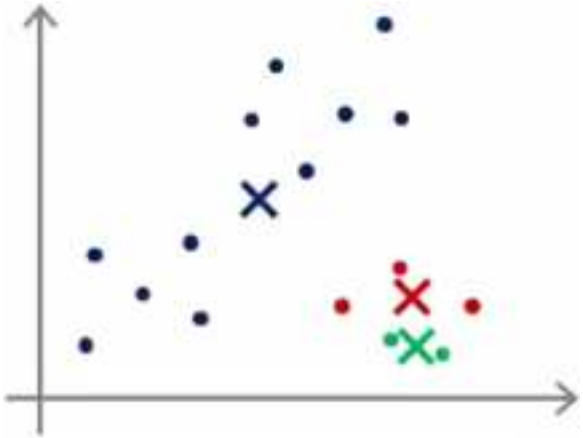


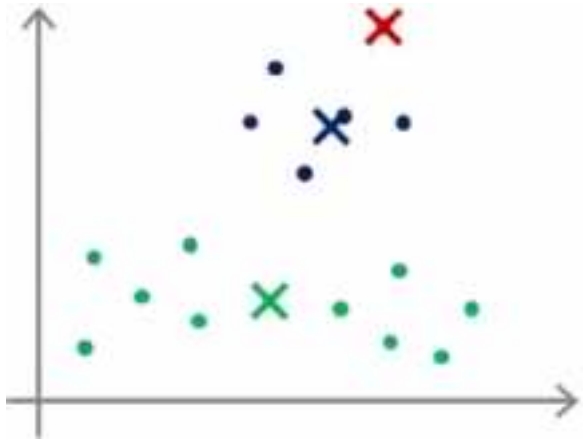
The k-means algorithm — initialization

In the simplest case, the initialization can be random, ie. arbitrary K training samples. However, it does not always give good results.



In the case like above left, we may get the desired solution (above right). However, unfortunate initialization can lead to any of the solutions below.





The k-means algorithm — initialization (cont.)

How can we avoid the effects of an unfortunate initialization that can lead to generating suboptimal clusters that reach local minima of the cost function?

As with the simulated annealing method, we can drop the generated centroids, and choose them again randomly. However, to compare the measure of quality (cost function, ie. the weighted sum of the distances of all points to their cluster centroids), the algorithm should be run to the end in both cases.

In practice, this means multiple (100?, 1000 times?) repetitions of the k-means algorithm for randomly selected starting points, and selecting the solution globally minimizing the cost function.

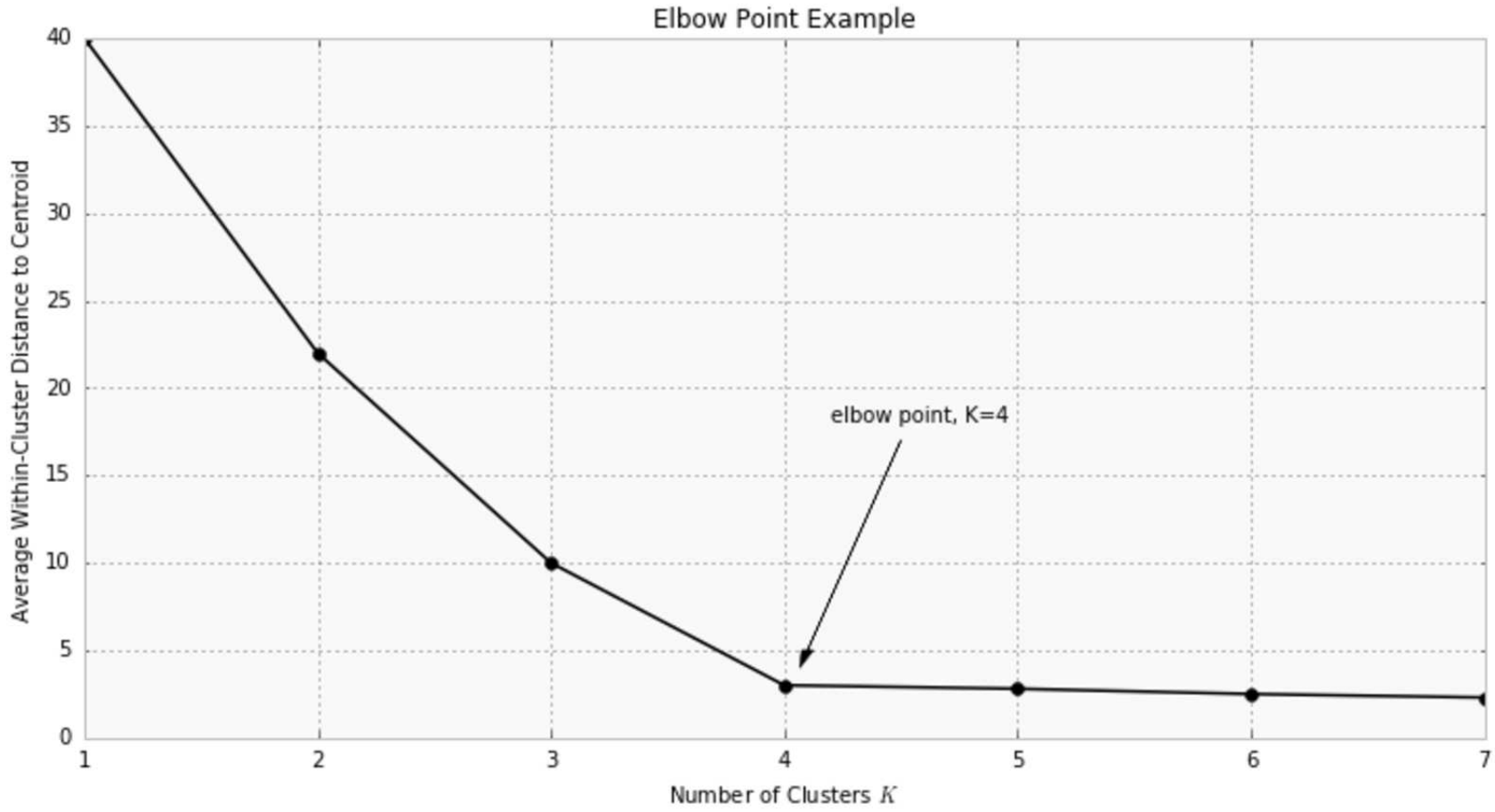
There are more "scientific" approaches to k-means initialization, such as the `k-means++` initialization algorithm, which greatly improves the outcome of subsequently applying k-means. `K-means++` doess k passes on the dataset, so it does not scale well for large datasets. Its improved version k-means$\|$ gives similar results and is much better scalable.

1.D.Arthur, S.Vassilvitskii: "K-means++: the advantages of careful seeding", 2007
2.B.Bahmani, B.Moseley, A.Vattani, R.Kumar, S.Vassilvitskii: "Scalable K-means++"

The k-means algorithm — choosing the number of clusters

The number of clusters K required by the algorithm is not always known in advance, and may sometimes need to be determined experimentally.

The **elbow point** method:



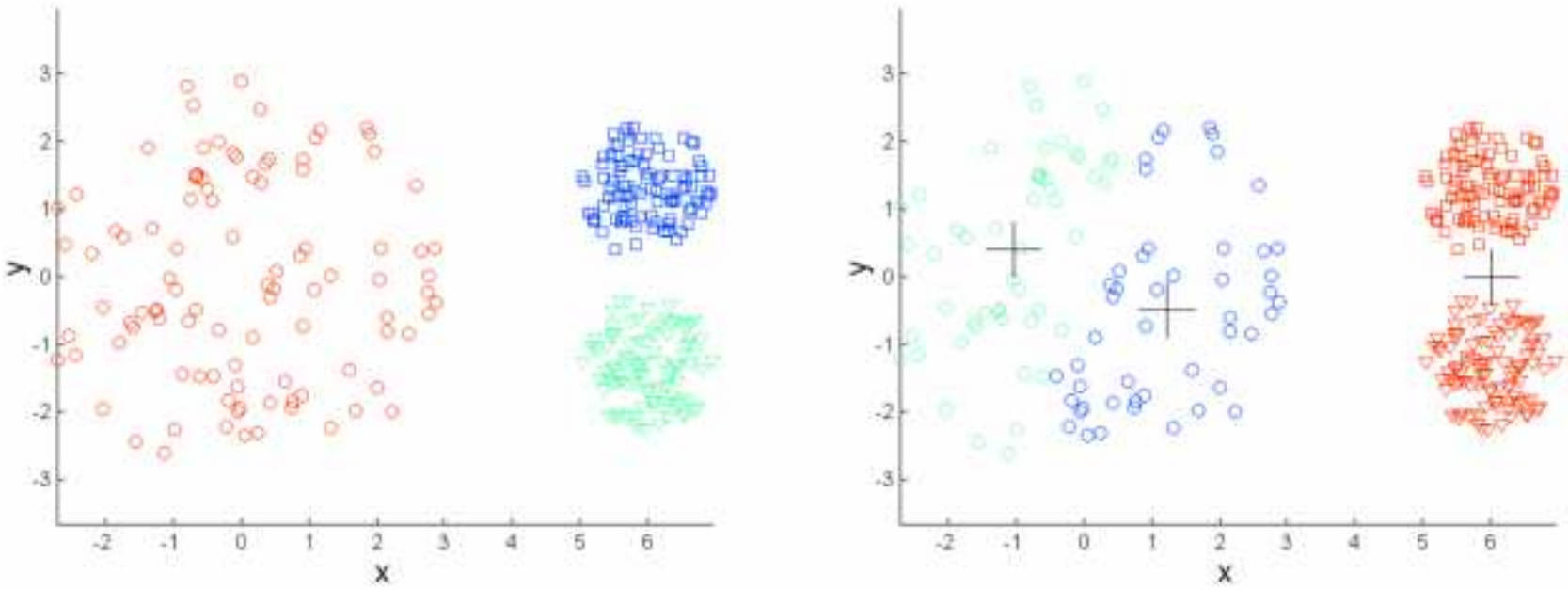
The elbow point method does not always work. Often the curve does not have the characteristic bend point, and simply asymptotically decreases as the number of clusters increases.

Unfortunately, in this case, we may not try to optimize the quality criterion, ie. the weighted sum of the distances of all points to their centroids. This is because the sum reaches zero for the number of clusters equal to the number samples $K = N$.

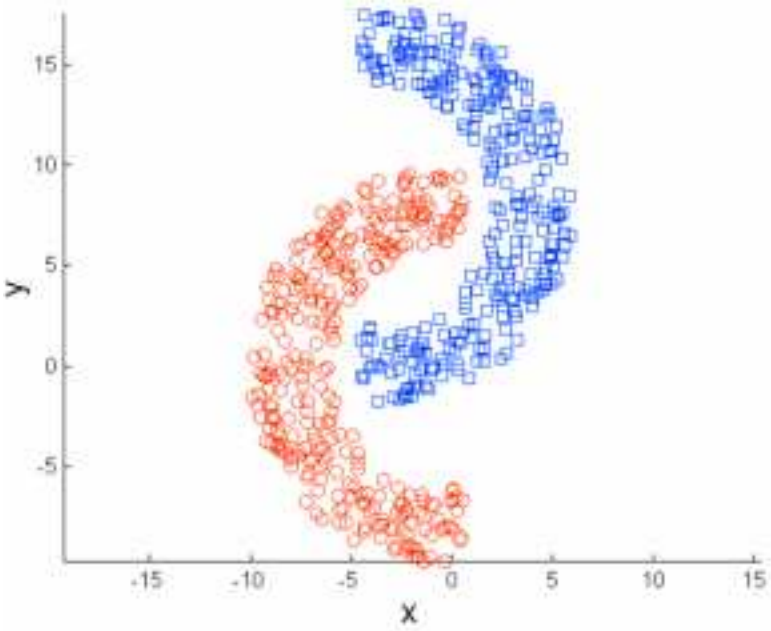
In this case, we can refer to the nature of the problem from which samples come. It is necessary to make a subjective assessment of the number of clusters which will be appropriate for this problem instance.

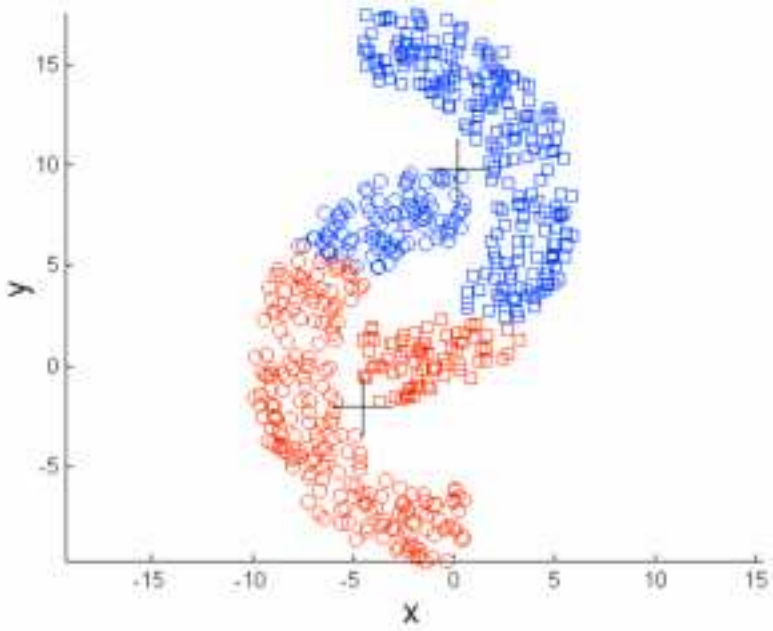
The k-means algorithm — special problems

The k-means algorithm works well in many practical cases. However, there are cases which it definitely cannot handle. Such cases are clusters of differing sizes, as well as clusters with different density of the samples in the training set.

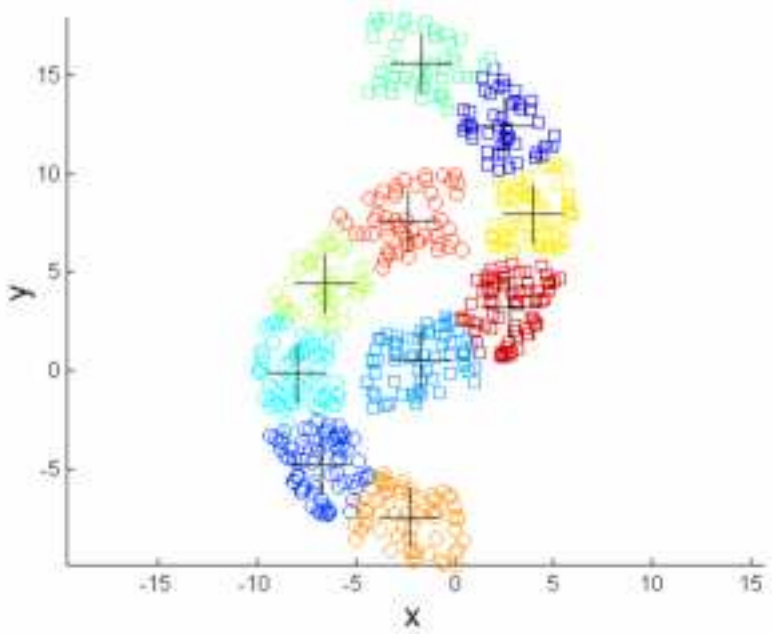


The k-means algorithm — special problems (2)





The problem with concave clusters can be solved indirectly by increasing the number of clusters.



The k-means algorithm — summary

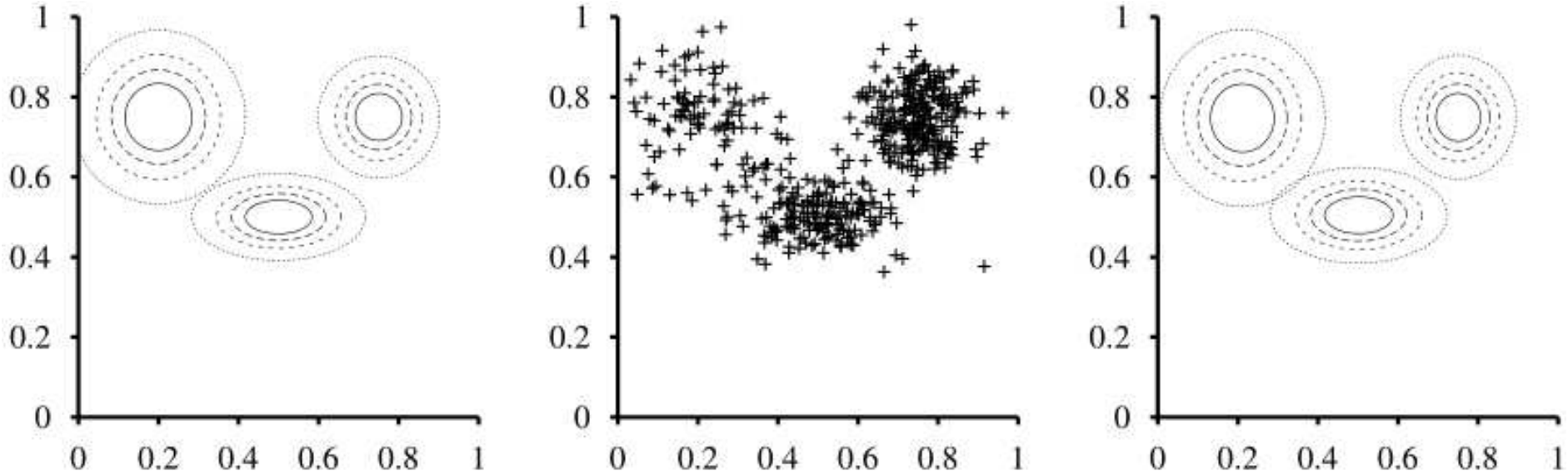
K-means is a simple and effective clustering algorithm. Its computational complexity is $O(tKN)$ where K, N are respectively the number of clusters and samples, while t is the number of iterations of the algorithm. Usually $K, t \ll N$.

However, it has some important problems that make it difficult, or they prevent its use:

- requires the number K of clusters to be known,
- is sensitive to the initialization of centroids; can converge to nonlocal minima,
- works with numerical data (calculation of means and distances); has problems with categorical data,
- has problems with clusters with non-convex shapes,
- has problems with clusters of varying sizes,
- has problems with clusters of different densities.

The Expectation Maximization (EM) algorithm

An approach similar to the k-means algorithm can be made on the probabilistic grounds. Assuming that the training set points belong to K clusters with some random probability distribution, it is natural to assume that these clusters result from normal probability distributions, so-called **mixture of normal** or **Gaussian** distributions. The algorithm **EM** (**Expectation Maximization**) learns the parameters of such a mixture of distributions.



The figure on the left shows a mixture of three simulated normal distributions.
The middle figure shows the set of points generated for this distribution.
The figure on the right shows a mixture of distributions learned by the EM algorithm.

The Expectation Maximization (EM) algorithm (cont.)

Assuming that the variable C denotes the mixture component in the range $1, ..., K$, the probability distribution of the mixture is given by the formula:

$$P(\mathbf{x}) = \sum_{i=1}^{K} P(C = i) P(\mathbf{x}|C = i)$$

where **x** is the vector of the sample attributes.

The parameters of the distribution are: $w_i = P(C = i)$ (weight of the component i), μ_i (the mean of the component i), and Σ_i (the covariance of the component i).

The idea of the algorithm is that we initially assume certain parameter values of the above distribution. In each cycle of the algorithm, for each point, the probability that it belongs to subsequent components is calculated. Then, the parameters of all components are recalculated based on all points, with weights as membership probabilities of a given point to a given component. These two steps are repeated until the algorithm converges, as with the k-means method.

EM — Expectation Maximization algorithm (cont.)

EM algorithm:

Initialization: set the initial values of all component parameters

REPEAT {

Step E: Calculate the probabilities $p_{ij} = P(C = i|\mathbf{x}_j)$ that the sample $\mathbf{x}_j$ belongs to component i. Under the Bayesian rule, we have: $p_{ij} = \alpha P(\mathbf{x}_j|C = i)P(C = i)$. We define $n_i = \Sigma_j \, p_{ij}$, which is the effective number of points currently assigned to component i.

Step M: Calculate new means, covariances, and component weights using the following formulas:

$$\begin{aligned} \mu_i &\leftarrow \sum_j p_{ij}\mathbf{x}_j/n_i \\ \Sigma_i &\leftarrow \sum_j p_{ij}(\mathbf{x}_j - \mu_i)(\mathbf{x}_j - \mu_i)^\top/n_i \\ w_i &\leftarrow n_i/N \end{aligned}$$

}

EM — Expectation Maximization algorithm (cont.)

The EM algorithm is not free from some problems. It is possible that one of the components will be reduced to a single point with zero variance and probability equal to 1. Another problem is the overlap (complete) of two components, which then share the same set of points.

Such phenomena lead to the convergence of the algorithm to a local maximum. This is a serious problem, especially in multidimensional spaces. The solution may be to reinitialize the component with new parameters, similar to the k-means algorithm.

Relationship between the k-means and EM methods

These algorithms are in some ways similar, they take two steps alternately: (1) generate clusters, and (2) transfer samples between clusters.

One significant difference is that in the k-means algorithm, points are categorically assigned to clusters, while EM assigns all points the probability of belonging to all distributions.

Another difference is the Gaussian model underlying the EM algorithm. The k-means algorithm, unlike the EM, is able to generate result distributions that are in no way similar to Gauss distributions. On the other hand, many natural phenomena follow the Gaussian model, so the EM algorithm works correctly for them.

Hierarchical clustering

Cluster analysis can be performed by building a hierarchy of clusters in one of two basic ways:

- bottom-up way, by starting from each sample representing a separate cluster, then merging them pairwise, to build larger clusters;
  this approach is referred to as **agglomerative clustering**

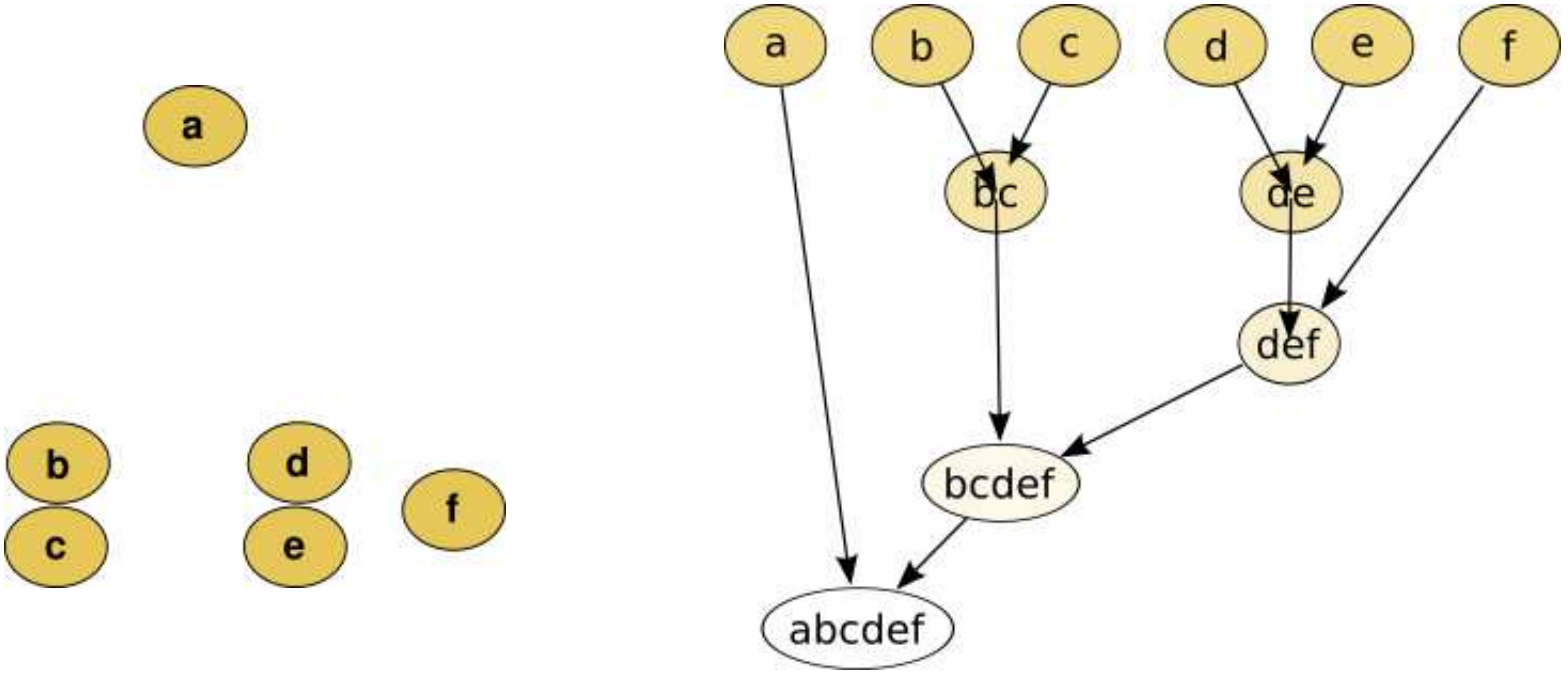
- top-down way, by starting from one cluster representing all samples, then splitting larger clusters into smaller ones;
  this is termed **divisive clustering**

The decisions of which clusters should be merged, or which cluster to split, and where exactly, is typically based on measuring distances between samples and clusters.

So in some way this is similar to the k-means approach, but is also different, in the necessity to measure the distance between clusters.

Agglomerative clustering — example

Agglomerative clustering is the more common approach. An example:



The resulting tree can be cut at a some height to produce the desired number of clusters.

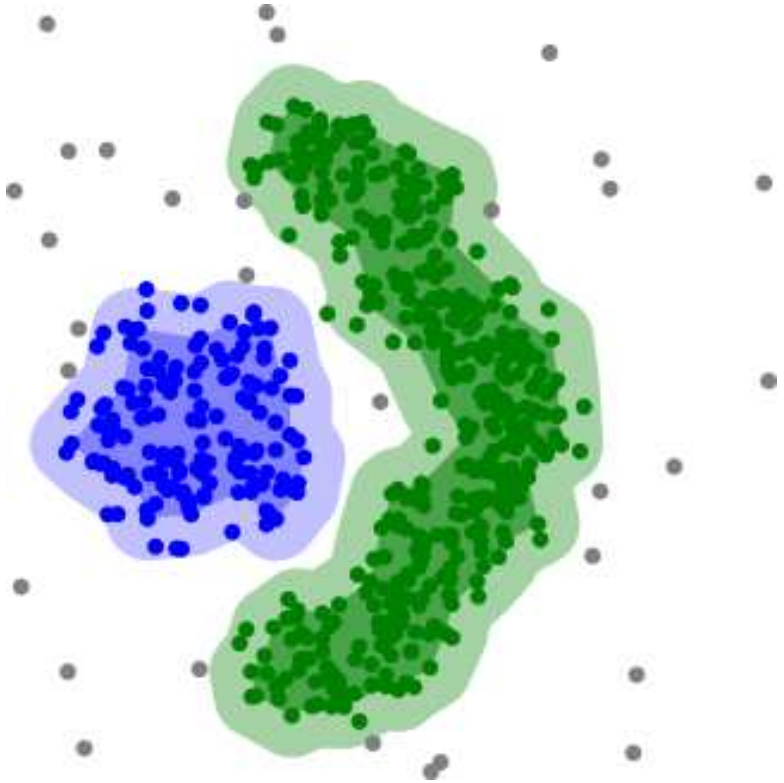
Cluster distance metrices

There are quite a few methods for measuring the inter-cluster distances:

- MIN — tends to produce long "loose" clusters
- MAX — tends to produce more compact clusters
- group average — considers the average distance between each point in one cluster to every point in the other one
- distance between centroids —
- Ward's method — similar as group average but sums up the squares of distances, minimizes the total within-cluster variance

Other approaches to clustering

There are a number of other approaches to clustering, such as the DBSCAN algorithm based on sample density. It is able to solve some problems that neither k-means nor EM can, like this:



`https://en.wikipedia.org/wiki/DBSCAN`

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu: "A density-based algorithm for discovering clusters in large spatial databases with noise", in: Evangelos, Simoudis, Jiawei Han, Usama M. Fayyad (eds): Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press. pp. 226-231, 1996

Dimension reduction

There are a number of **dimension reduction** methods that can transform a data representation into another space with a smaller dimension. One of the reasons motivating this transformation is the **curse of dimensionality**, which is one of the main problems of machine learning.

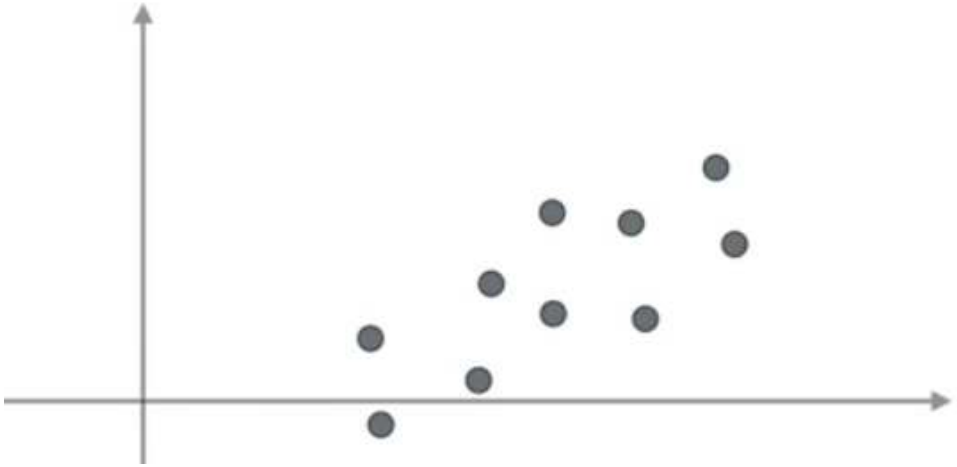
Figuratively speaking, many machine learning algorithms that efficiently process data in low-dimensional spaces, cease to function satisfactorily when the space dimension is large.

From another point of view, the data is typically represented by a number of parameters, some of which may not have a significant (or any) impact on the ability to classify or cluster these data. Such redundant parameters not only do not help in the automatic detection of patterns existing in the data, but significantly hamper the learning process, because they introduce false apparent correlations which mislead the algorithms.

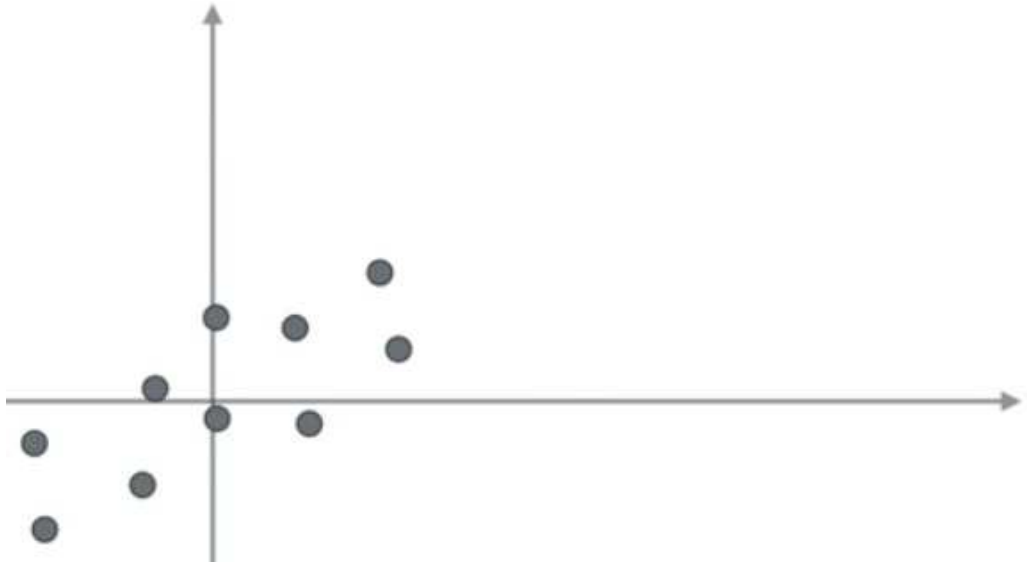
Therefore, it is profitable to perform an analysis and reduction of dimensions before proceeding to a machine learning experiment. One effective method of this is the **Principal Component Analysis** (PCA).

The Principal Component Analysis (PCA) — an example

Consider some set of points:



Move its geometric center to the origin of the coordinate system:



Compute the covariance matrix of the data set:

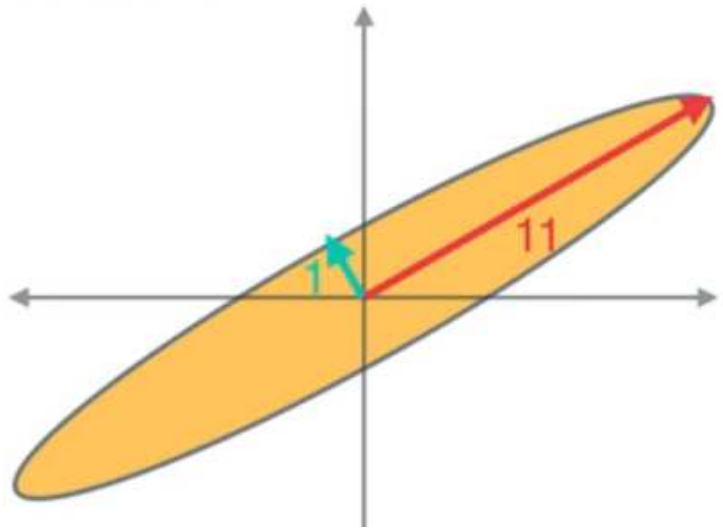
$$\Sigma = \begin{pmatrix} var(x) & cov(x,y) \\ cov(x,y) & var(y) \end{pmatrix} = \begin{pmatrix} 9 & 4 \\ 4 & 3 \end{pmatrix}$$

This covariance matrix generates some linear transformation:

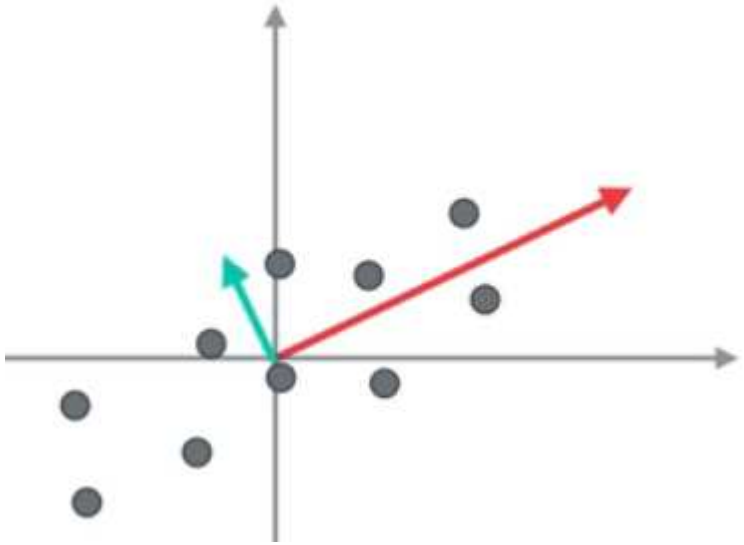
$$(x, y) \longrightarrow (9x + 4y, 4x + 3y)$$

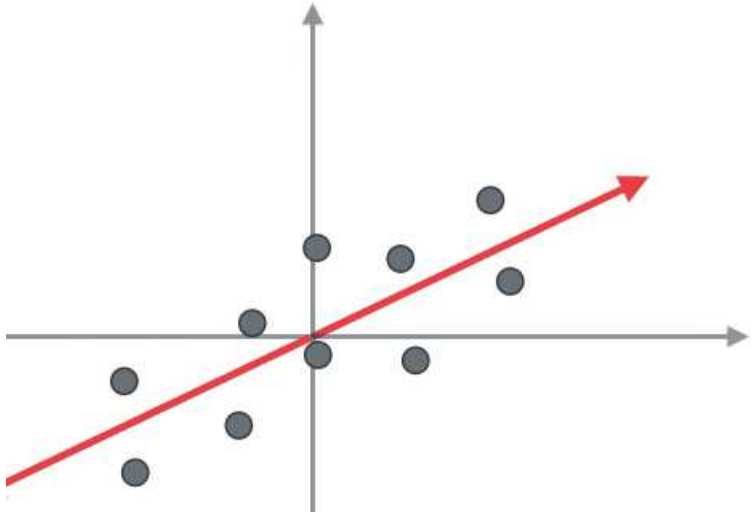
This transformation transfers the original points to a coordinate system whose axes are eigenvectors of the covariance matrix, and the eigenvalues represent the linear extension:

Eigenvectors	$\begin{bmatrix} 2 \\ 1 \end{bmatrix}$	$\begin{bmatrix} -1 \\ 2 \end{bmatrix}$
Eigenvalues	11	1

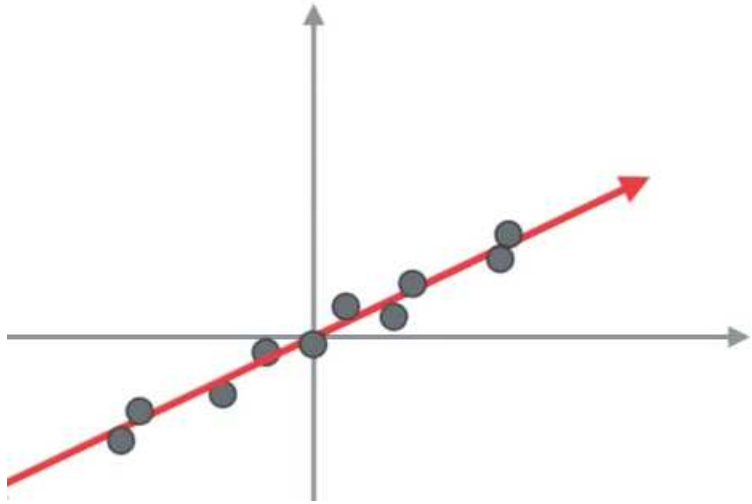


The original points are exactly represented in the new coordinate system whose axes are called the **principal components**. However, for the purposes of dimension reduction, we want to keep only one coordinate. It must be the main component with the greatest eigenvalue.





A dimension reduction is obtained by the representation of points by projecting them into a space with lower dimensions, i.e. in this case on the selected coordinate axis. It is therefore an approximate representation.



The PCA Algorithm

The PCA algorithm finds a M -dimensional approximation of the data set $\{\mathbf{x}^n : n = 1, ..., N\}$ with the original dimension $\dim(\mathbf{x}^n) = D$ $(M < D)$:

1. Calculate the vector of the means $\mathbf{m}$ of a set of samples with size $D \times 1$ and covariance matrix $\mathbf{S}$ size $D \times D$:

$$\mathbf{m} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}^n, \quad \mathbf{S} = \frac{1}{N-1} \sum_{n=1}^{N} (\mathbf{x}^n - \mathbf{m})(\mathbf{x}^n - \mathbf{m})^T.$$

2. Determine the eigenvectors $\mathbf{e}^1, ..., \mathbf{e}^D$ of the covariance matrix $\mathbf{S}$ sorted by decreasing eigenvalues of the eigenvectors. Create the matrix $\mathbf{E} = [\mathbf{e}^1, ...\mathbf{e}^M]$.

3. A low-dimensional representation of $\mathbf{y}^n$, and an approximate reconstruction $\mathbf{x}'^n$ of each of the $\mathbf{x}^n$ samples are given by:

$$\mathbf{y}^n = \mathbf{E}^T(\mathbf{x}^n - \mathbf{m}), \quad \mathbf{x}^n \approx \mathbf{x}'^n = \mathbf{m} + \mathbf{E}\mathbf{y}^n.$$

4. The total square error of the approximation for the training set is:

$$\sum_{n=1}^{N} (\mathbf{x}^n - \mathbf{x}'^n) = (N-1) \sum_{j=M+1}^{D} \lambda_j$$

where $\lambda_{M+1}, ..., \lambda_N$ are the omitted eigenvalues.

Note: the PCA algorithm is sensitive to the magnitudes of parameter values. If one parameter has much greater values than another, then the PCA algorithm will invariably select that first parameter as the first primary component, and the other one as the second. For this reason, the parameters should be rescaled uniformly before applying the PCA.

Useful resources

In this presentation materials from the following works were used:

1. Andrew Ng: Unsupervised learning, Coursera video lecture

2. Stuart J. Russell, Peter Norvig: Artificial Intelligence A Modern Approach (Third Edition), Prentice-Hall, 2010

3. Kevin P. Murphy: Machine Learning A Probabilistic Perspective, MIT Press, 2012