Unsupervised learning
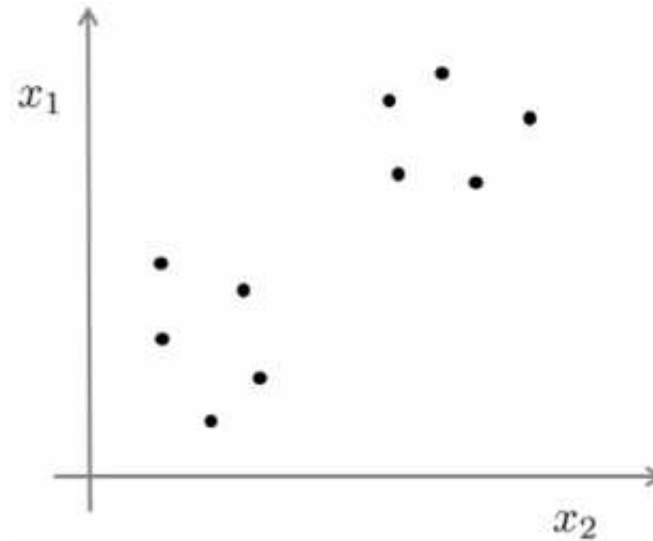
Supervised, classification:

Unsupervised, **clustering**:
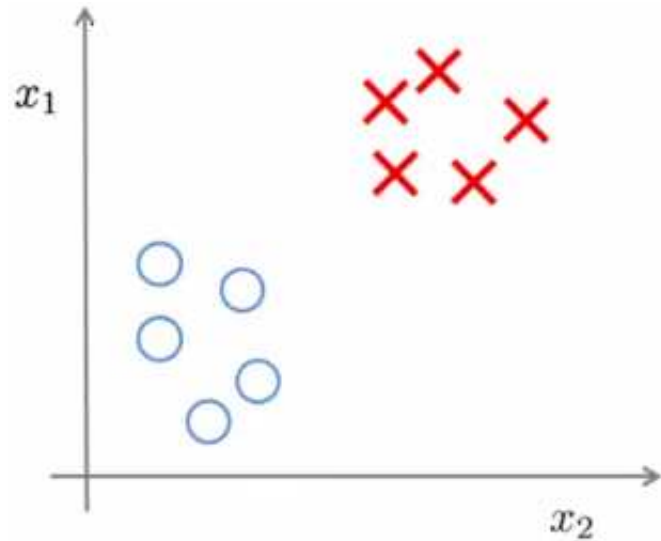
Training set:

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

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.
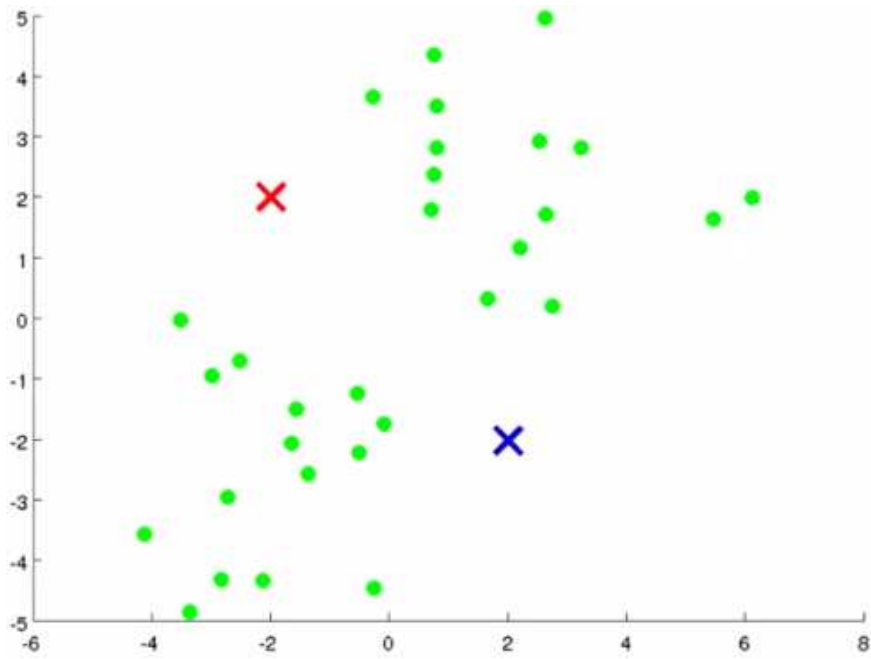
The k-means algorithm:

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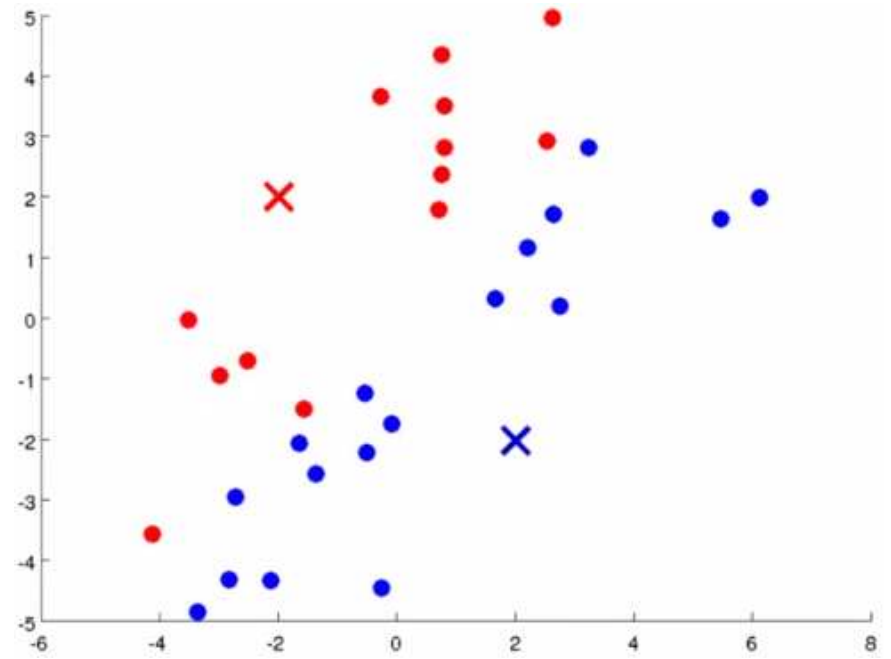
REPEAT {

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
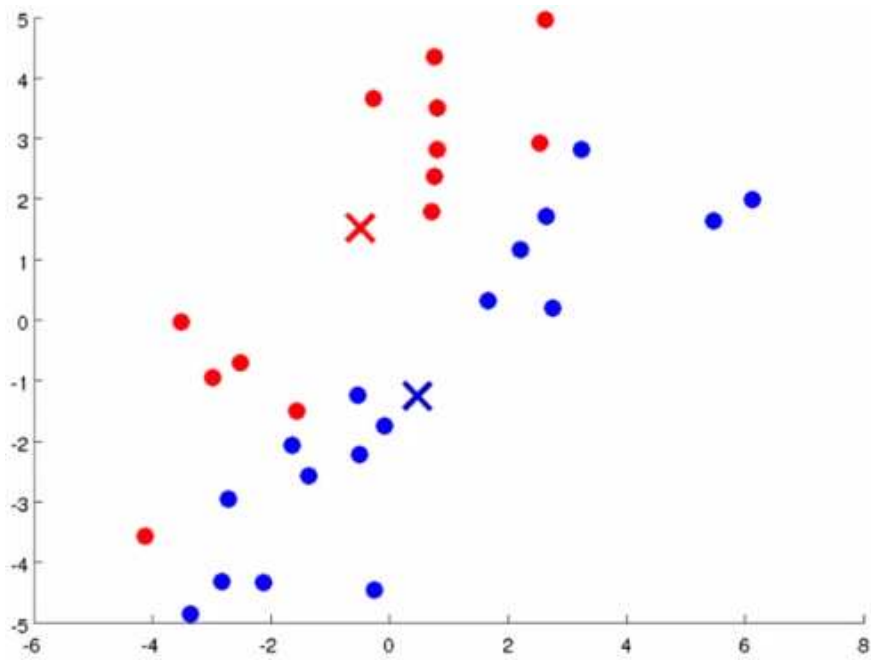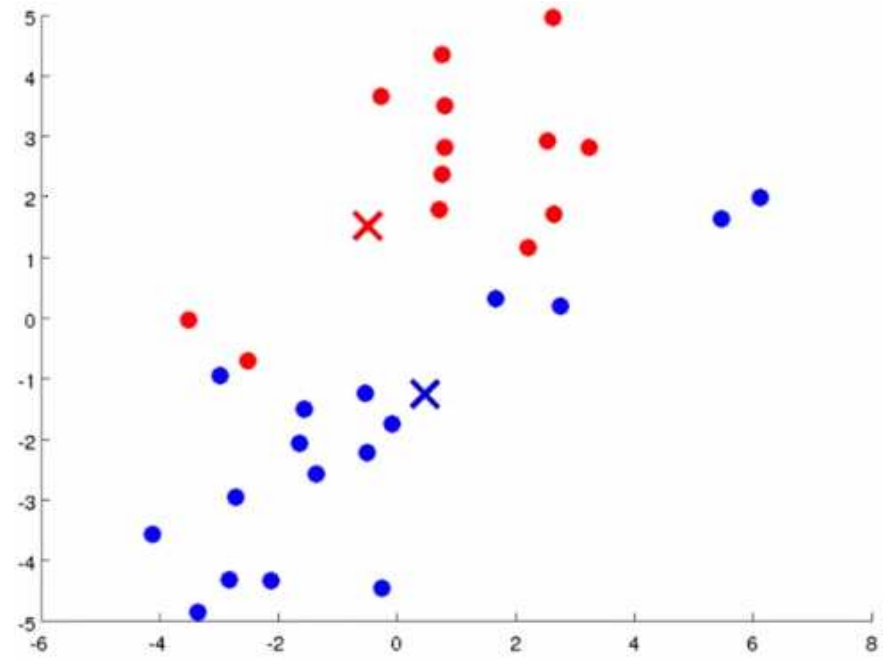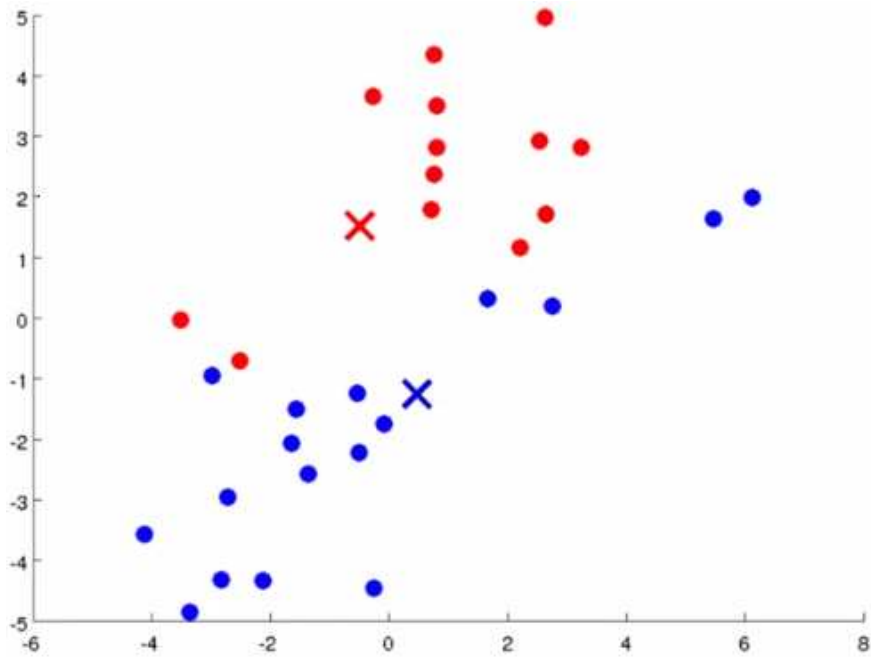
}

Step 0 (initialization)

Step 1 (labeling)

Step 2 (centroid shift)

Step 1 (labeling)

Step 1 (labeling)
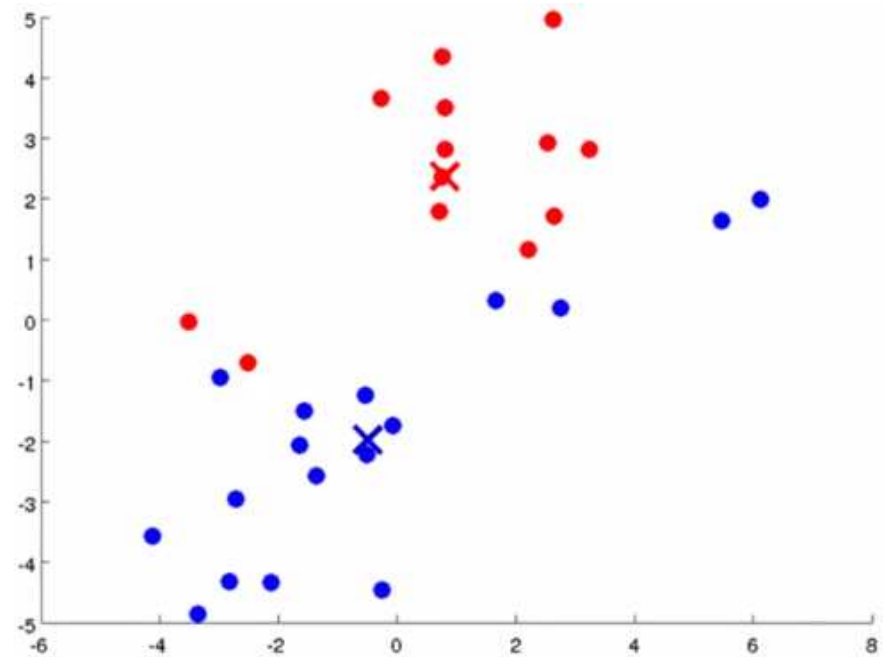
Step 2 (centroid shift)
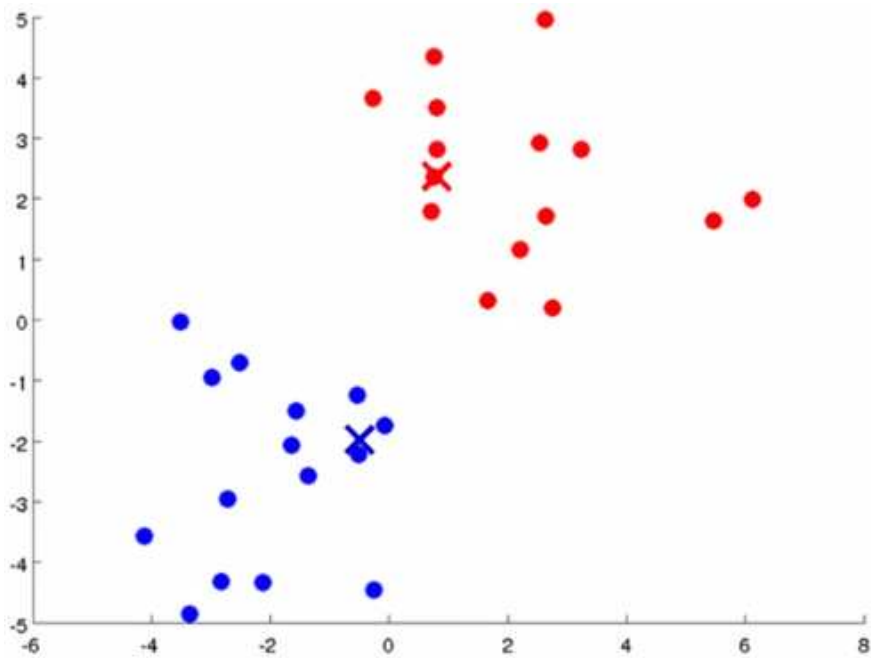
Step 1 (labeling)
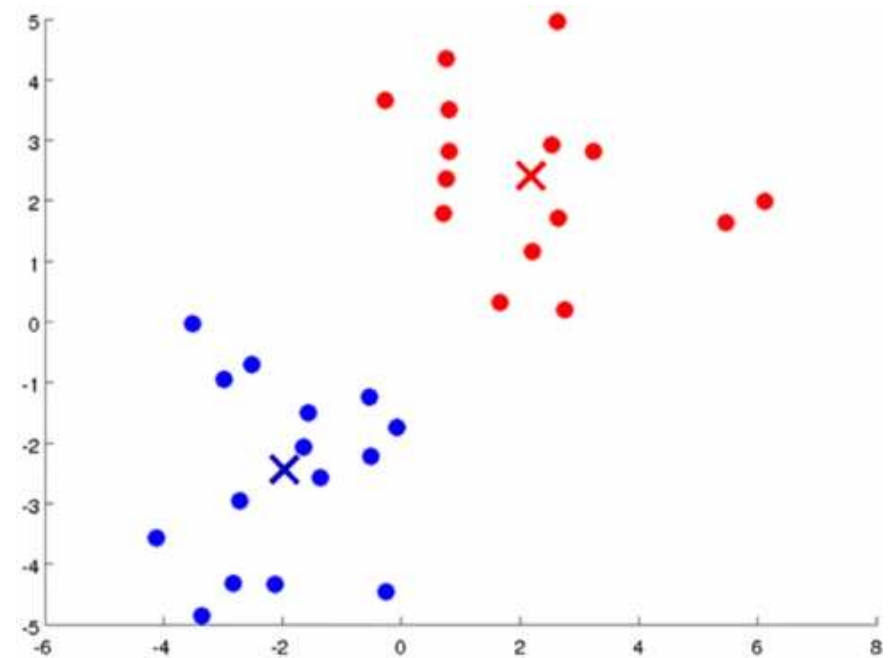
Step 2 (centroid shift)

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:

$$
\begin{array}{ll}
\text{Euclidean} & \sqrt{\Sigma_i(a_i - b_i)^2} \\
\text{Manhattan} & \Sigma_i |a_i - b_i| \\
\text{Max} & \max_i |a_i - b_i|
\end{array}
$$

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 a!number of metrics dedicated for them. Simple examples of such metrics are the Hamming and Levenshtein distance.

Hamming distance (only for strings of equal length) = the number of positions at which the corresponding symbols in both strings differ. It is equivalent to the minimum number of single character substitution required to transform one string into the other.

Levenshtein distance (for any strings) = the smallest number of single character insertions, deletions, or substitutions required to transform one string into the other.

The k-means algorithm — another example

An example from Wikipedia:

`https://upload.wikimedia.org/wikipedia/commons/e/ea/K-means_convergence.gif`
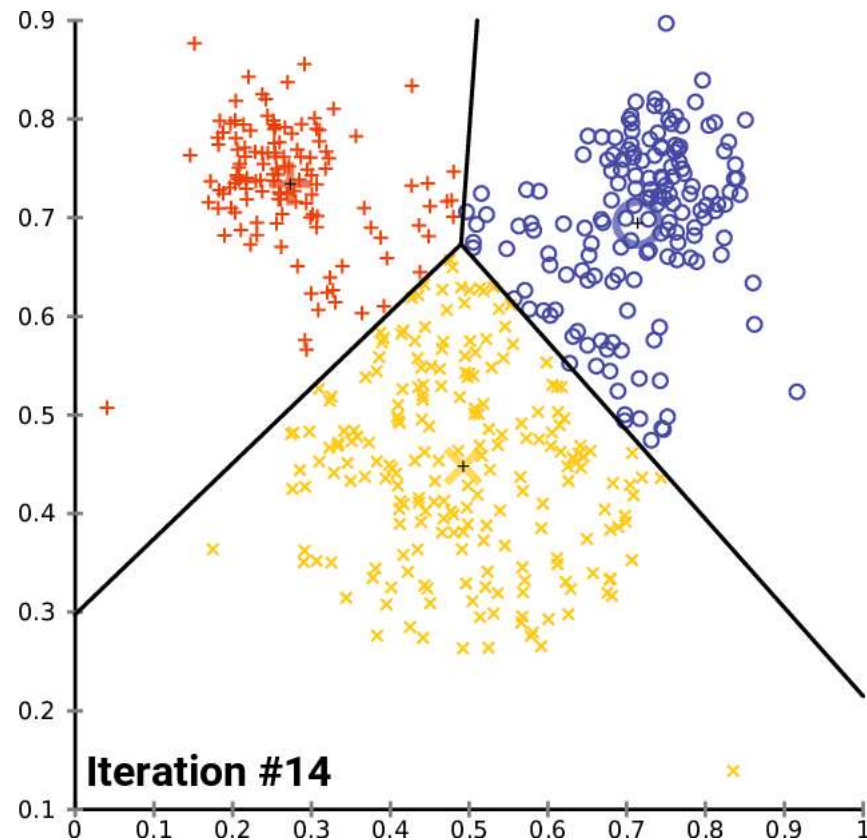
As we can see in the above example, the k-means algorithm does not always generate so intuitively correct results, as in the previous examples. There are a number of special cases that need to be considered, to obtain optimal results.

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:
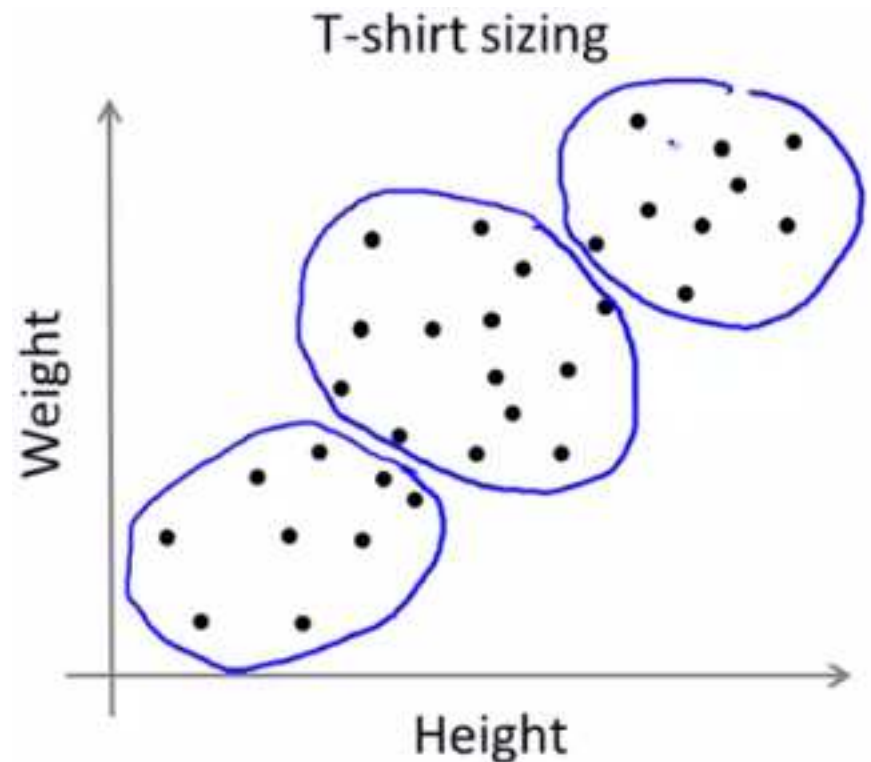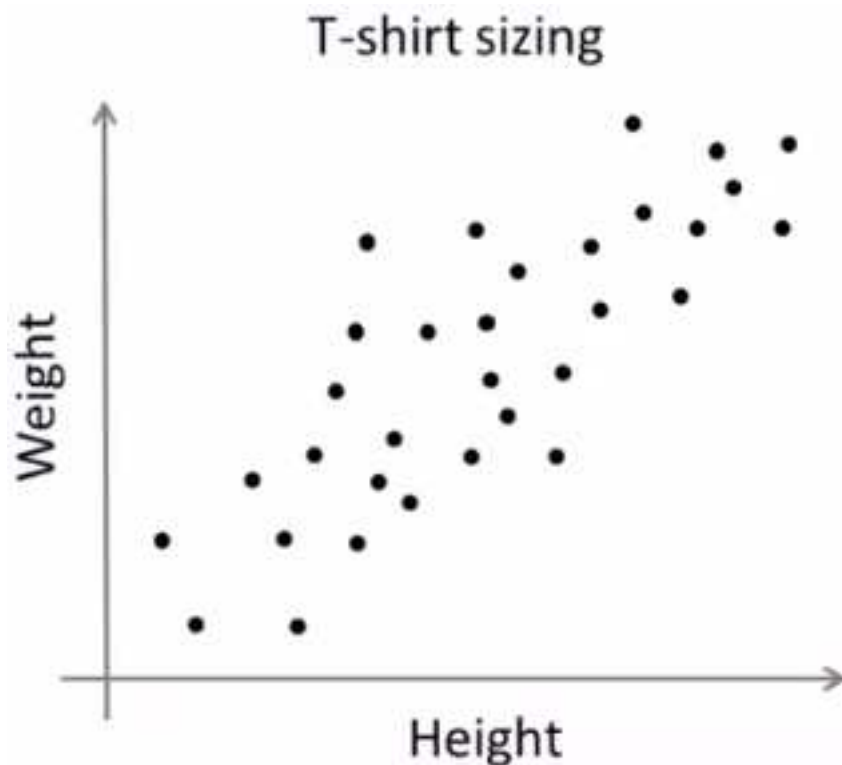
The k-means algorithm — initialization

In the simplest case, the initialization can be random, ie. arbitrary K training samples. However, it does not always give good results.
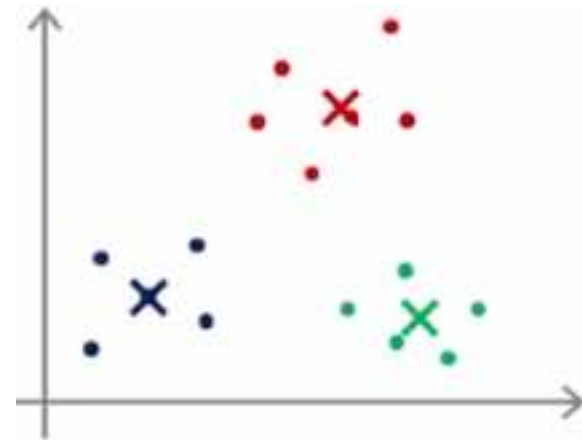
In the case like above left, we may get the desired solution (above right). However, unfortunate initialization can lead to any of the solutions below.

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.

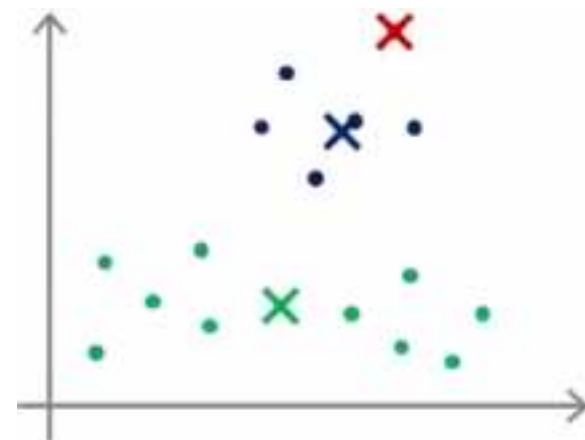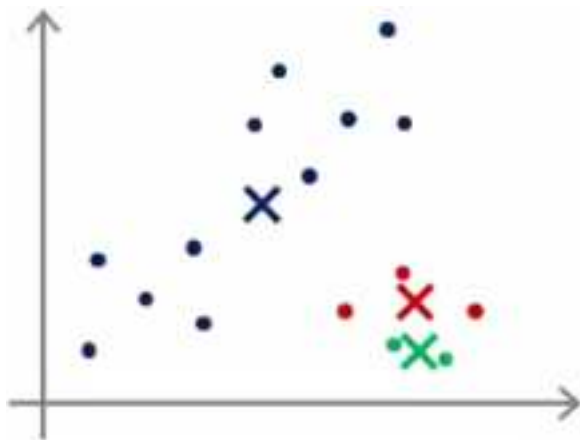In practice, this means multiple (100?, 1000 times?) repetitions of the k-means algorithm for randomly selected starting points, and selecting the solution globally minimizing the cost function.

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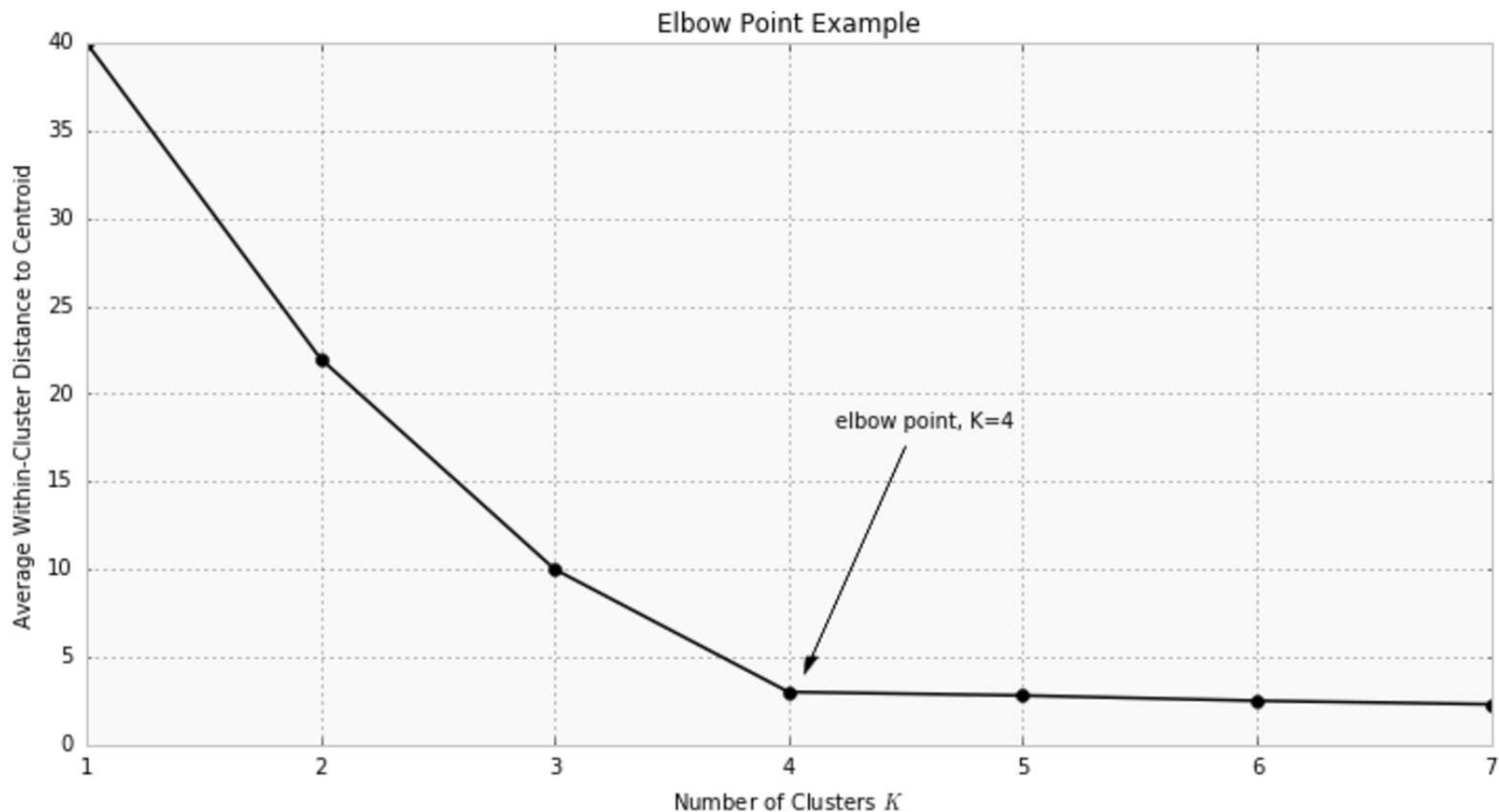
The elbow point method does not always work. Often the curve does not have the characteristic bend point, and simply asymptotically decreases as the number of clusters increases.

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.
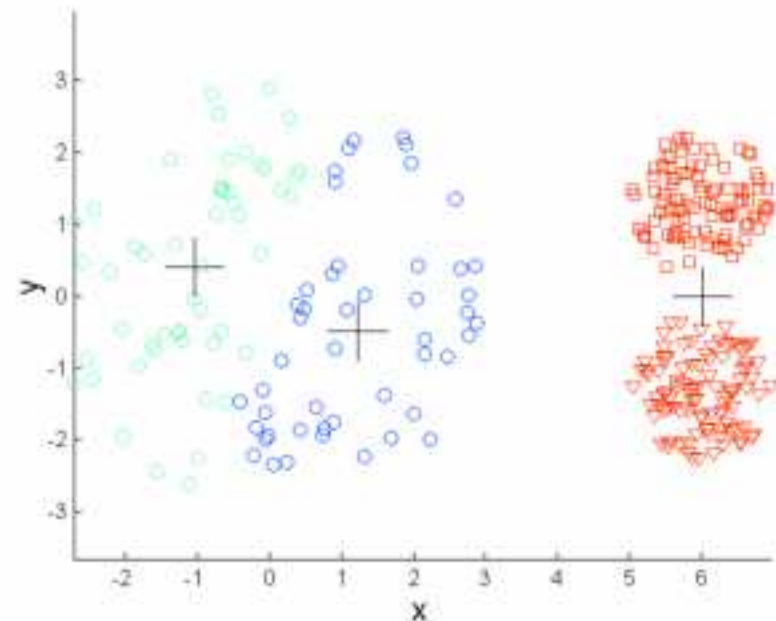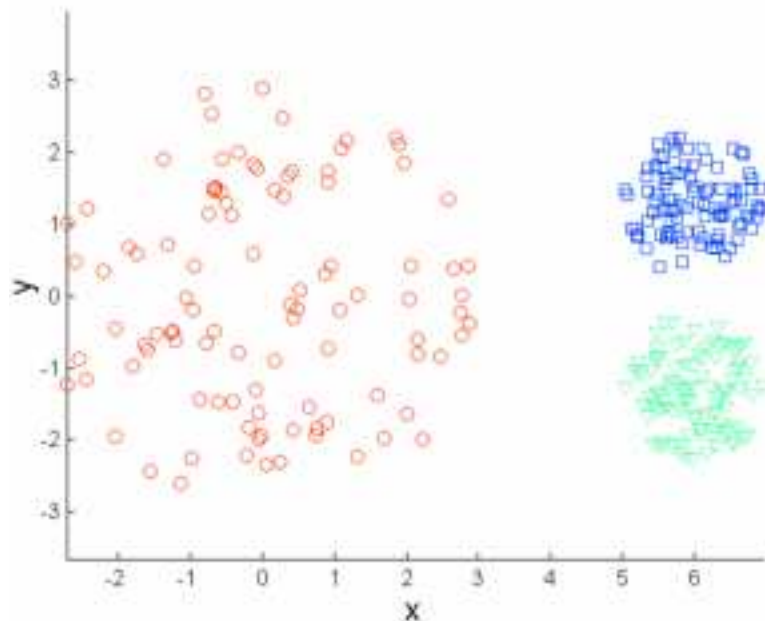
The k-means algorithm — special problems (2)
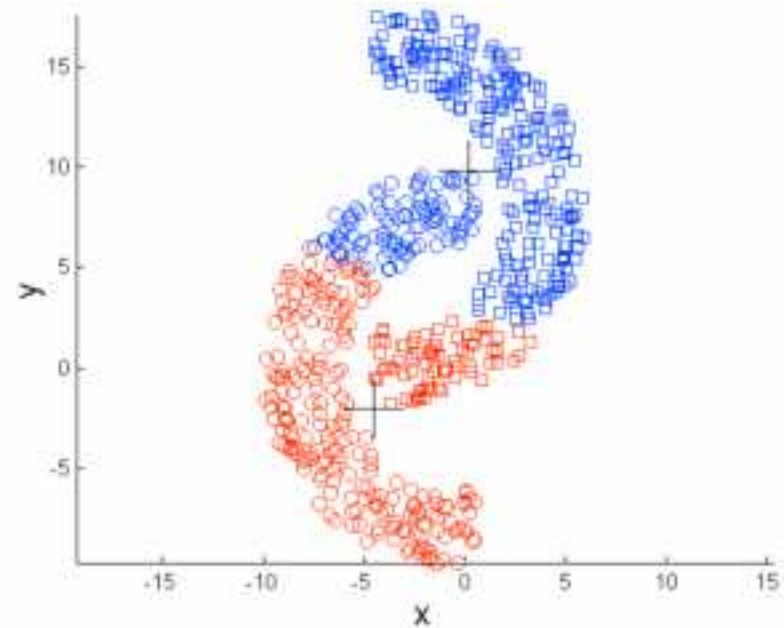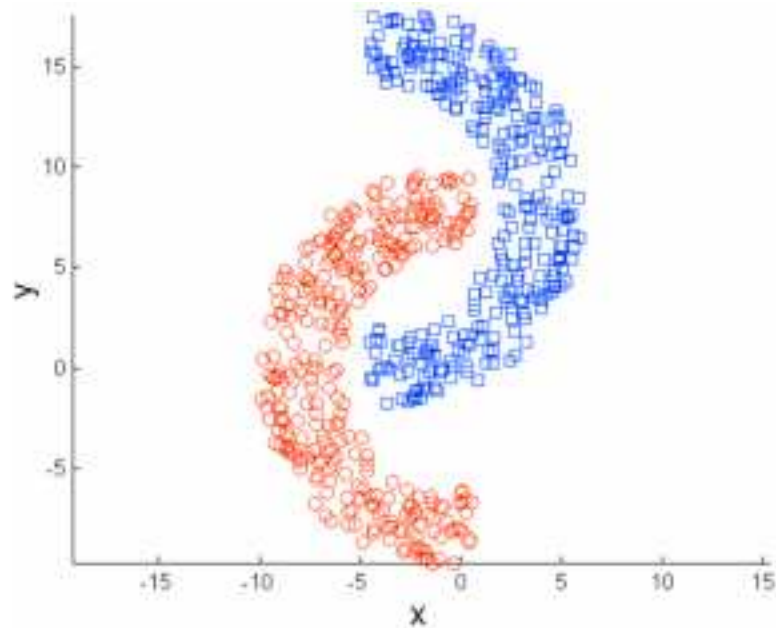
The problem with concave clusters
can be solved indirectly by
increasing the number of clusters.
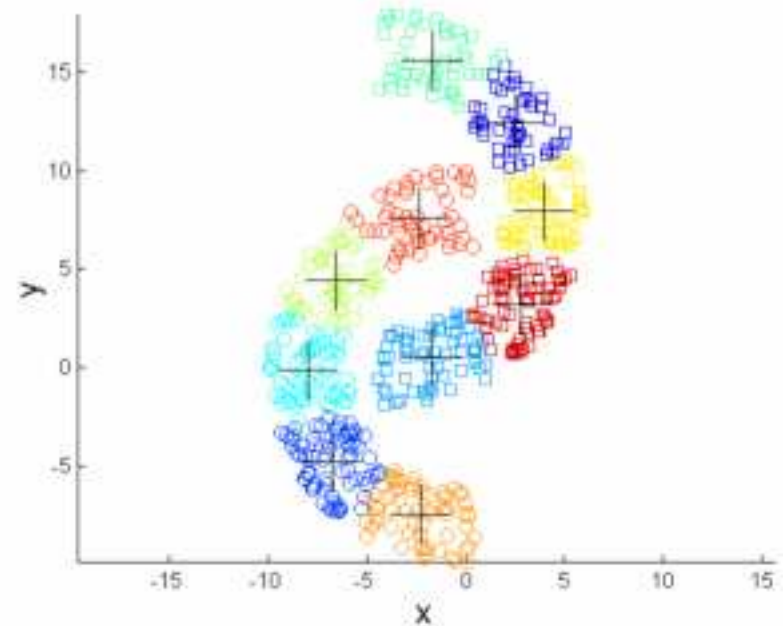
The k-means algorithm — summary

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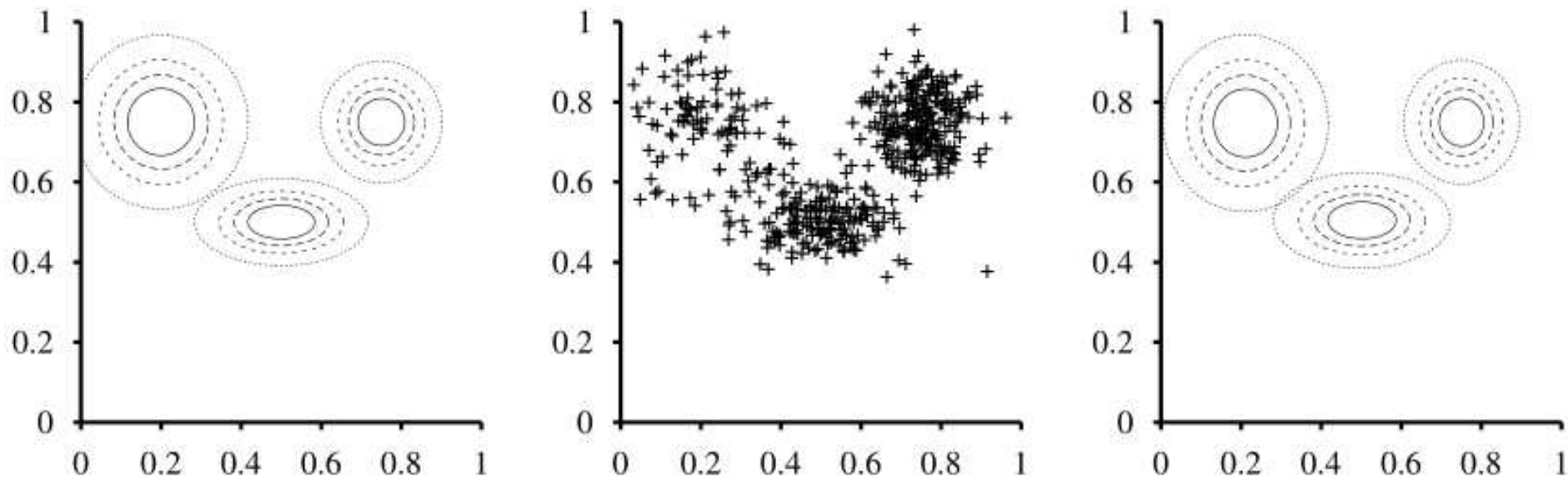
The figure on the left shows a mixture of three simulated normal distributions.
The middle figure shows the set of points generated for this distribution.
The figure on the right shows a mixture of distributions learned by the EM algorithm.

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 $\mathbf{x}$ 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:

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

}

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.

DBSCAN — density-based approach to clustering

DBSCAN is an algorithm developed to overcome the difficulty of k-means in clustering oddly-shaped clusters:

- classifies all points as: **core points** of some cluster, **border points** of a cluster, and **outliers**, which do not belong to any cluster,

- does not require the number of clusters to be specified in advance,

- can find arbitrarily-shaped clusters; it can even find a cluster completely surrounded by (but not connected to) a different cluster,

- requires two parameters: *minPts* specifies a minimum number of neighbor points to declare a point as a core point, and *eps* specifies the radius of a point's neighborhood.
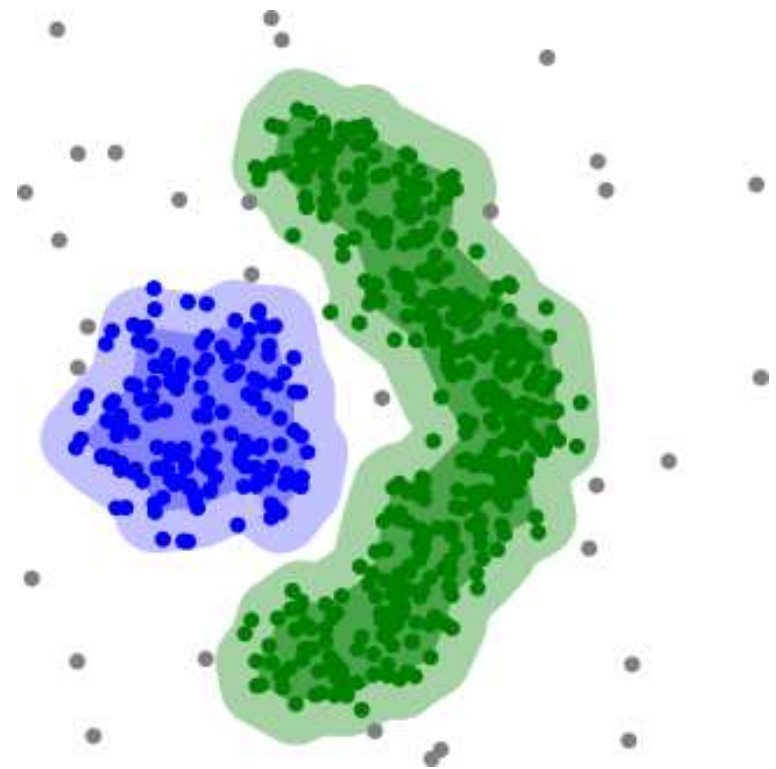
DBSCAN algorithm

1. Identify Core Points:

For each point in the dataset count the number of points in its *eps*-radius neighborhood; if it exceeds the *minPts*, then mark the point as a **core point**.

2. Form Clusters:

Find the connected components of core points on the *eps*-distance neighborhood graph. Each connected component is a separate cluster.

All non-core points within the *eps*-distance from a cluster point also belong to this cluster, but are marked as **border points**.

Points in such clusters are **density-connected**.

3. Label Noise Points:

After processing all points, any point that does not belong to a cluster is marked as **noise points**.
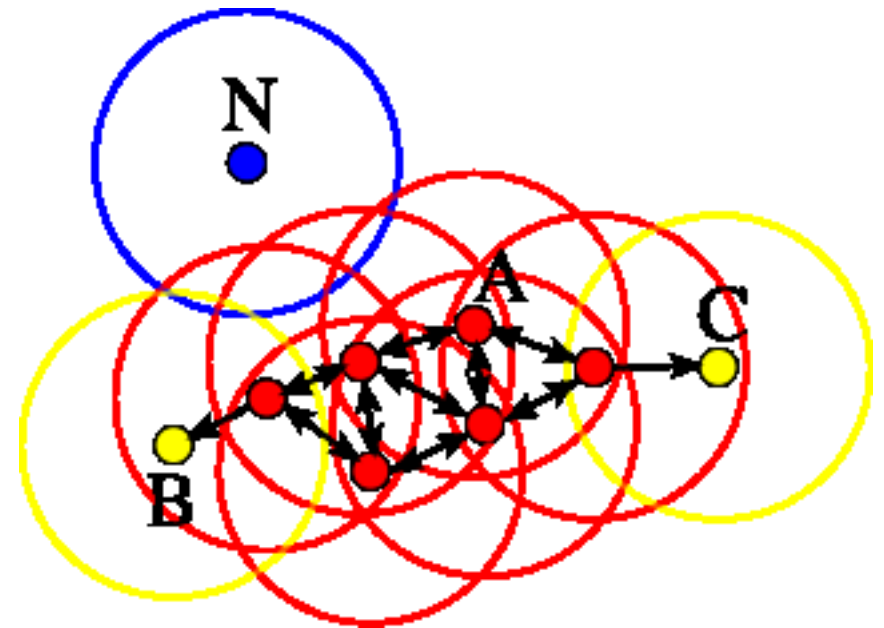
Hierarchical clustering

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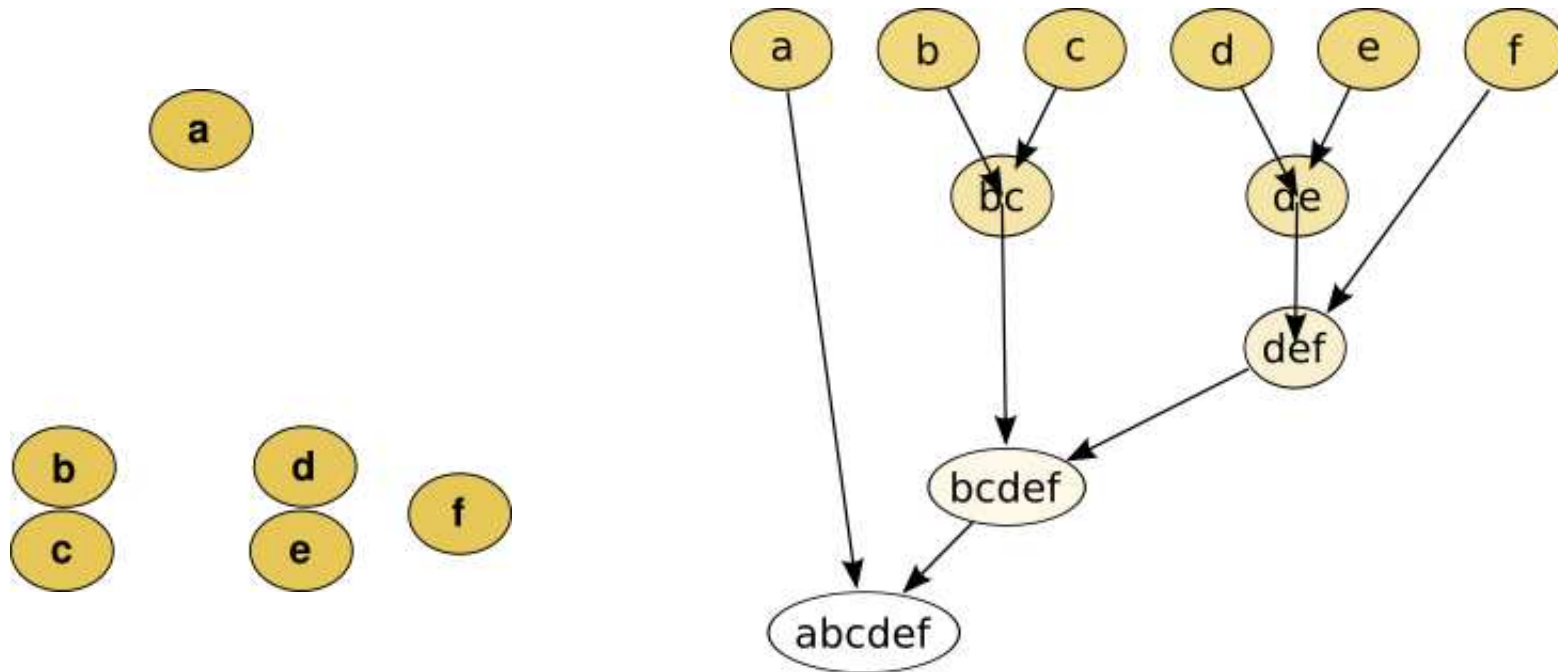
The resulting tree can be cut at a some height to produce the desired number of clusters.

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

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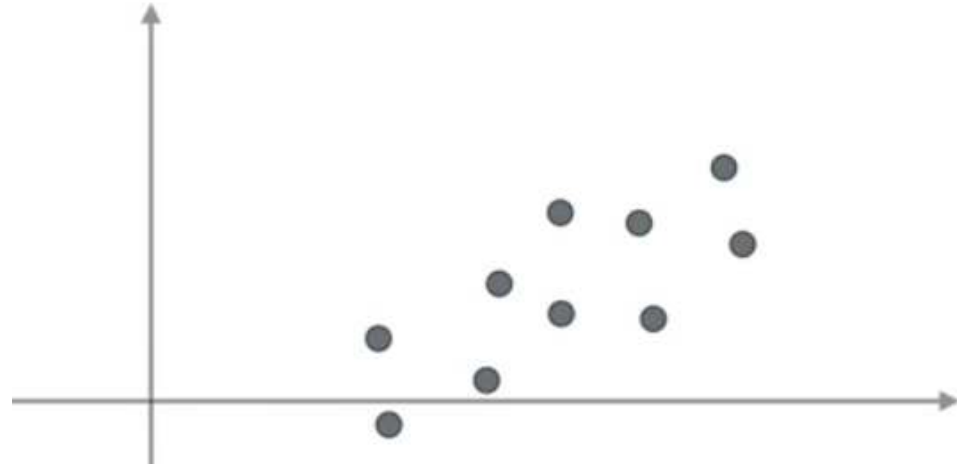
Move its geometric center to the origin of the coordinate system:

Compute the covariance matrix of the data set:

$$\Sigma = \left(\begin{array}{cc} var(x) & cov(x,y) \\ cov(x,y) & var(y) \end{array} \right) = \left(\begin{array}{cc} 9 & 4 \\ 4 & 3 \end{array} \right)$$

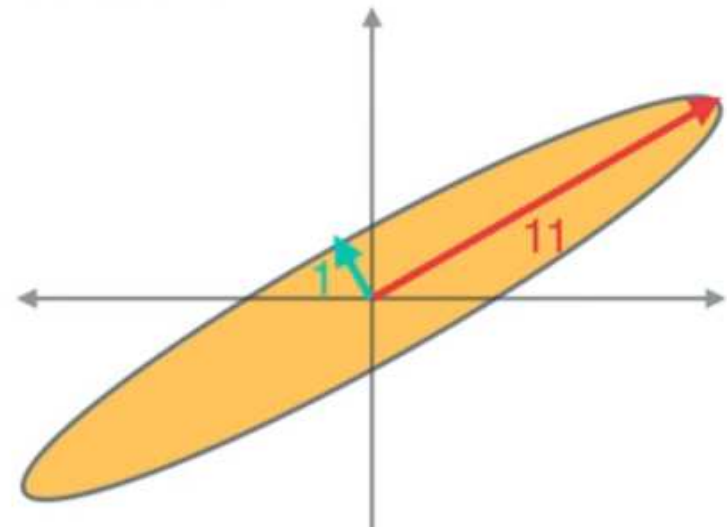This covariance matrix generates some linear transformation:

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

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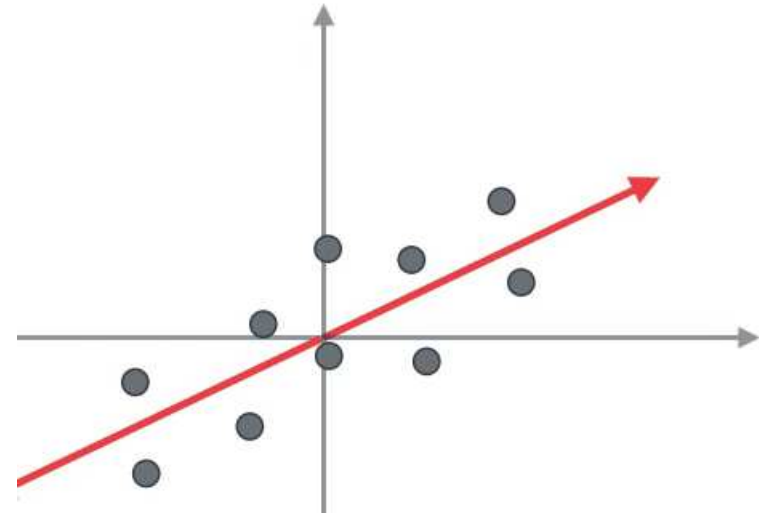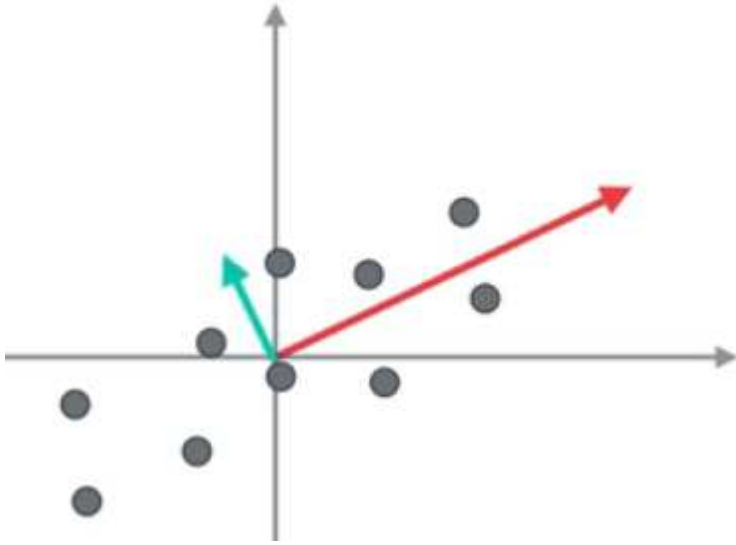
A dimension reduction is obtained by the representation of points by projecting them into a space with lower dimensions, i.e. in this case on the selected coordinate axis. It is therefore an approximate representation.
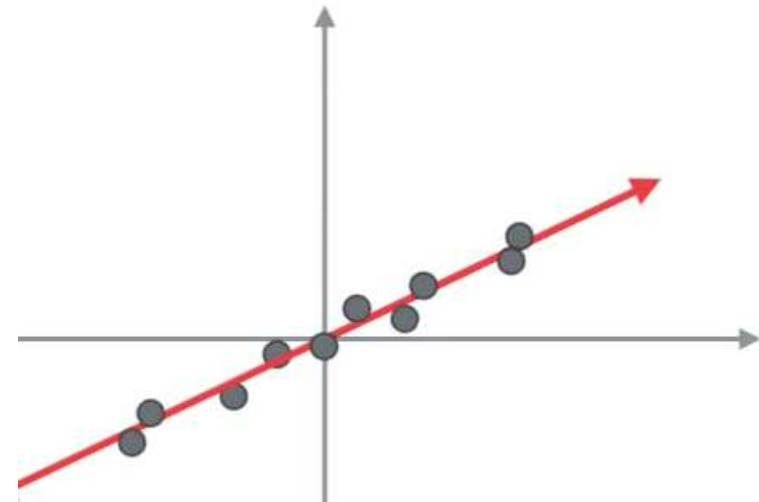
The PCA Algorithm

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.

Market basket analysis

There are a number of data mining/machine learning methods for purchases. Consider the two types of recommendations displayed by sales portals such as Amazon:

- Frequently purchased together: ...
- Customers who bought this product also bought ...

The first group of methods is referred to as **Market Basket Analysis**. MBA methods focus on **association rule mining** characterizing typical purchases, i.e., finding relationships between different items purchased in different transactions. Such relationships are represented, for example, in the form of IF-THEN rules. One of the most well-known algorithms of this group is Apriori.

A slightly different approach to mining purchase transaction history data is represented by **Recommender Systems**. They aim to determine the preferences of individual consumers. The best-known algorithms belong to the collaborative filtering group, and will be considered subsequently.

Market basket analysis — model

We consider the purchase transaction history as a set of transactions $T = \{t_1, t_2, ..., t_n\}$, where a single transaction is a subset of the $t \subseteq I$ of the set of all available items $I = \{i_i, i_2, ..., i_m\}$.

Example:

Transaction 1	🍎 🍺 🍚 🍗
Transaction 2	🍎 🍺 🍚
Transaction 3	🍎 🍺
Transaction 4	🍎 🍐
Transaction 5	🍼 🍺 🍚 🍗
Transaction 6	🍼 🍺 🍚
Transaction 7	🍼 🍺
Transaction 8	🍼 🍐

Market basket analysis — association rules

We want to discover relationships between elements of purchase transactions in the form of **association rules** of the form, assuming that A and B are disjoint ($A \cap B = \emptyset$) sets of purchase elements ($A, B \subseteq I$):

$$A \Rightarrow B$$

There are probably many such relationships throughout the history of buying, but some may be stronger than others. We introduce the following measures of the credibility of the $A \Rightarrow B$ rule based on a set of transactions T.

The **Support** of the rule $A \Rightarrow B$ in the set of transactions T we call the frequency of transactions that contain all the elements from A and B:

$$\text{Support}(A \Rightarrow B) = P(A \cup B)$$
$$\text{Support}(X) = P(X)$$

The **Confidence** of the rule $A \Rightarrow B$ in a set of transactions T we call the frequency of such transactions containing B, which also contain A:

$$\text{Confidence}(A \Rightarrow B) = P(B|A)$$

In other words, the confidence of a rule tells how often it has been true in the set of transactions.

In addition, we define the **Lift** as a fraction:

$$\text{Lift}(A \Rightarrow B) = \frac{P(A \cup B)}{P(A) \cdot P(B)}$$

The Lift indicates the overall meaning of the rule in the following sense:

$$\text{Lift}(A \Rightarrow B) \quad > 1 \quad \text{means that } A, B \text{ are positively correlated}$$

$$\text{Lift}(A \Rightarrow B) \quad < 1 \quad \text{means that } A, B \text{ are negatively correlated}$$

$$\text{Lift}(A \Rightarrow B) \quad = 1 \quad \text{means that } A, B \text{ are independent}$$

Market basket analysis — frequent sets

The search for association rules requires defining a minimum threshold of support and a minimum confidence threshold. The determination of all rules association rules can be implemented as follows:

- Find all sets of **frequent** purchased elements.

  A set of elements is frequent if it exceeds a given support threshold *minsupport*.

- Within each frequent set, determine the appropriate association rules.

  The association rule is must exceed the minimum confidence threshold *minconf* to be selected.

Finding frequent sets requires checking all subsets of the of the set of elements I of which there are $2^{|I|} - 1$ (excluding the empty subset). A full search of these subsets would be too inefficient.

The Apriori algorithm — generating frequent sets

The *apriori* property: all subsets of a frequent set are also frequent sets.
And conversely: a superset of a set that is not frequent also cannot be frequent.

This property can be used for the efficient generation of frequent sets according to the idea: find all frequent one-element sets, then with them generate all two-element sets and filter out from them only the frequent sets, then generate three-element frequent sets in the same way, and so on.

Algorithm:

1. Create a collection of one-element frequent sets F_1.
2. For $k=2..|I|$ while $F_{k-1} \neq \emptyset$ do:

   (a) based on F_{k-1} create a collection C_k such k-element frequent set candidates c_k, which are unions of two $(k-1)$-element frequent sets f^i_{k-1} and f^j_{k-1} with $(k-2)$ common elements
   (b) each $(k-1)$-element subset c_k must be a frequent set in F_{k-1}
   (c) create a collection of k-element frequent sets F_k filtering out infrequent sets from C_k.

Condition (b) results in early filtering of non-frequent sets based on the *apriori* property.

The Apriori algorithm — generating association rules

The algorithm for generating rules:

1. For each frequent set X

   (a) For each proper and nonempty subset A of set X:
   let $B = X \setminus A$
   $A \Rightarrow B$ be an association rule if Confidence$(A \Rightarrow B) \geq$ *minconf*

The Apriori algorithm — example

Consider a database of purchase transactions:

ID	Elements
T1	I1 I3 I4
T2	I2 I3 I5
T3	I1 I2 I3 I5
T4	I2 I5

To execute the Apriori algorithm we will assume the parameters $minsupport=0.5$; $minconf=0.75$.

First iteration:

ID	Elements
T1	I1 I3 I4
T2	I2 I3 I5
T3	I1 I2 I3 I5
T4	I2 I5

$\Rightarrow$ C1:

set	Support
{I1}	2/4=0.5
{I2}	3/4=0.75
{I3}	3/4=0.75
{I4}	1/4=0.25
{I5}	3/4=0.75

$\Rightarrow$ F1:

set	Support
{I1}	0.5
{I2}	0.75
{I3}	0.75
{I5}	0.75

Second iteration:

ID	Elements
T1	I1 I3 I4
T2	I2 I3 I5
T3	I1 I2 I3 I5
T4	I2 I5

$\Rightarrow$ C2:

set	Support
{I1,I2}	1/4=0.25
{I1,I3}	2/4=0.5
{I1,I5}	1/4=0.25
{I2,I3}	2/4=0.5
{I2,I5}	3/4=0.75
{I3,I5}	2/4=0.5

$\Rightarrow$ F2:

set	Support
{I1,I3}	2/4=0.5
{I2,I3}	2/4=0.5
{I2,I5}	3/4=0.75
{I3,I5}	2/4=0.5

Third iteration:

ID	Elements
T1	I1 I3 I4
T2	I2 I3 I5
T3	I1 I2 I3 I5
T4	I2 I5

$\Rightarrow$ C3:

set	Support
{I1,I2,I3}	1/4=0.25
{I1,I3,I5}	1/4=0.25
{I2,I3,I5}	2/4=0.5

$\Rightarrow$ F3:

set	Support
{I2,I3,I5}	2/4=0.5

It should be noted that the sets of {I1,I2,I3} and {I1,I3,I5} were rejected by the Apriori algorithm even before the calculation of their support because they contain subsets that do not belong to F_2 (step 2b of the algorithm).

The algorithm stopped on the fourth iteration because the set C_4 is empty. The frequent sets with F_2 and F_3 pass to rule generation because the one-element sets from F_1 do not generate any rules.

Rule generation:

ID	Elements
T1	I1 I3 I4
T2	I2 I3 I5
T3	I1 I2 I3 I5
T4	I2 I5

$\Rightarrow$

ID	zbiór	reguła			Confidence
R1	{I1,I3}	{I1}	$\Rightarrow$	{I3}	2/2=1.0
R2	{I1,I3}	{I3}	$\Rightarrow$	{I1}	2/3=0.66
R3	{I2,I3}	{I2}	$\Rightarrow$	{I3}	2/3=0.66
R4	{I2,I3}	{I3}	$\Rightarrow$	{I2}	2/3=0.66
R5	{I2,I5}	{I2}	$\Rightarrow$	{I5}	3/3=1.0
R6	{I2,I5}	{I5}	$\Rightarrow$	{I2}	3/3=1.0
R7	{I3,I5}	{I3}	$\Rightarrow$	{I5}	2/3=0.66
R8	{I3,I5}	{I5}	$\Rightarrow$	{I3}	2/3=0.66
R9	{I2,I3,I5}	{I2,I3}	$\Rightarrow$	{I5}	2/2=1.0
R10	{I2,I3,I5}	{I2,I5}	$\Rightarrow$	{I3}	2/3=0.66
R11	{I2,I3,I5}	{I3,I5}	$\Rightarrow$	{I2}	2/2=1.0
R12	{I2,I3,I5}	{I2}	$\Rightarrow$	{I3,I5}	2/3=0.66
R13	{I2,I3,I5}	{I3}	$\Rightarrow$	{I2,I5}	2/3=0.66
R14	{I2,I3,I5}	{I5}	$\Rightarrow$	{I2,I3}	2/3=0.66

Finally, for the given parameters, the Apriori algorithm generated five association rules.

Recommender systems

We will now look at another approach related to analyzing data purchasing, concerning recommendation systems. Wanting to recommend a certain commodity to a certain consumer, we want to predict his possible evaluation of that commodity. For this procedure of determining the consumer's possible evaluation is often used the term **filtering**, which in statistics is used in reference to the approximation of current (rather than future or past).

To determing the expected rating of item A by consumer X we can use:

used-based collaborative filtering
   if consumer X is similar to Y, and Y bought A; recommend A to consumer X

item-based collaborative filtering
   if consumer X bought A, and A is similar to B; recommend B to consumer X

There are also hybrid approaches possible, combining the above two in different ways.

Recommender systems learn to determine specific values based on memorized data, and therefore formally belong to supervised learning methods. We discuss them in the group of algorithms devoted to unsupervised learning because of the similarity of applications.

Collaborative and content-similarity based filtering

In principle, the task of determining the unknown parameter A of a sample X when the parameter this parameter is known in the learning series, is typically a classification task (when the the values of the parameter A are discrete in nature) or regression (when these values are selected from a continuous numerical range). Thus, previously learned algorithms can be apply previously learned algorithms. This approach to the issue of of recommendations is referred to as *model based*.

However, another approach is possible, which is to select from the the dataset we have of samples on consumers , "similar" in some sense to consumer X, and basing the recommendation on the parameters of only those consumers. (Or, in the case of filtering based on similarity of content, on selecting only goods , "similar" to the A in question.) This approach is referred to as *memory-based* and will be briefly introduced here.

One advantage of the latter approach is that the result obtained is easily presented in context and justified.

Collaborative filtering — calculating consumer similarity

Assuming that the available data contains ratings of multiple goods determined by consumers, we can calculate the "similarity" of consumers X and Y on the Based on their ratings $r_{x,1}, r_{x,1}, ..., r_{y,1}, r_{y,2}...$ of a certain group of goods I_{xy} rated by both consumers, using one of the following similarity models.

Pearson similarity model:

$$\text{simil}(x,y) = \frac{\sum\limits_{i \in I_{xy}} (r_{x,i} - \bar{r_x})(r_{y,i} - \bar{r_y})}{\sqrt{\sum\limits_{i \in I_{xy}} (r_{x,i} - \bar{r_x})^2} \sqrt{\sum\limits_{i \in I_{xy}} (r_{y,i} - \bar{r_y})^2}}$$

Similarity calculated as the cosine of the angle between the attribute vectors:

$$\text{simil}(x,y) = \cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{||\vec{x}|| \times ||\vec{y}||} = \frac{\sum\limits_{i \in I_{xy}} r_{x,i} r_{y,i}}{\sqrt{\sum\limits_{i \in I_{xy}} r_{x,i}^2} \sqrt{\sum\limits_{i \in I_{xy}} r_{y,i}^2}}$$

Collaborative filtering — determining consumer ratings

Having the similarities of consumers calculated with respect to the rated goods, we can select a certain set U of N consumers most similar to a given consumer u, and determine their rating of the good i as:

$$r_{u,i} \;=\; \frac{1}{N} \sum_{u' \in U} r_{u',i} \qquad \text{or:}$$

$$r_{u,i} \;=\; \frac{\sum\limits_{u' \in U} \mathsf{simil}(u, u') r_{u',i}}{\sum\limits_{u' \in U} |\mathsf{simil}(u, u')|} \qquad \text{or:}$$

$$r_{u,i} \;=\; \bar{r}_u + \frac{\sum\limits_{u' \in U} \mathsf{simil}(u, u')(r_{u',i} - \bar{r}_{u'})}{\sum\limits_{u' \in U} |\mathsf{simil}(u, u')|}$$

The first version is a simple average of ratings of "similar" consumers, the second is an average of ratings weighted by the mutual similarity of consumers, and the third additionally isolates from the absolute value of the ratings of individual consumers, calculating only the differences from their average ratings, and applying the relative difference in the rating of an item i to the average rating of the consumer u.

Item-based filtering

In item similarity-based filtering, we use the the same data of ratings of multiple goods by multiple consumers, but this time we calculate the similarity matrix of the goods, using the same similarity formulas.

Next, we determine the set of N goods most similar to the item whose rating for a consumer u we want to determine, and we determine the rating using one of the averaging methods similarly to how we averaged consumer ratings.

Rating matrix factorization

A completely different approach to collaborative filtering is the idea of factorization of the original rating matrix $R \in \mathcal{R}^{\text{users} \times \text{items}}$ into two matrices, of which the first one H has rows corresponding to all consumers, and the second W has columns corresponding to all goods. The columns of the first and rows of the second of the matrix are associated with newly created **latent factors** (*hidden variables*) in such a way that the vector product of the matrix H and W as faithfully as possible approximates the evaluation matrix, and then completes the evaluations, which are missing and which need to be determined:

$$\tilde{R} = H \times W$$

where:
$\tilde{R} \in \mathcal{R}^{\text{users} \times \text{items}}$ is a matrix of item rating predictions,
$H \in \mathcal{R}^{\text{users} \times \text{latent factors}}$ is a matrix of latent variables of users,
$W \in \mathcal{R}^{\text{latent factors} \times \text{items}}$ is a matrix of latent variable of items (transposed).

This idea of **matrix factorization** is to aims to identify a number of state variables (latent variables) allowing evaluations to be expressed using matrices of lower dimensionality. This is somewhat equivalent to the principal component analysis method (*Pricipal Component Analysis*, PCA).

Methods for factorization of evaluation matrix

Latent variables are usually determined by machine learning methods, such as such as neural networks. A number of methods have been proposed factorization of rating matrices. The first version of rating matrix factorization, called **Funk MF** was developed in response to a Netflix contest. In this version, the prediction $\bar{r}_{ui}$ of a user u rating of an item i is calculated according to the formula:

$$\bar{r}_{ui} = \sum_{f=0}^{n \text{ factors}} H[u, f]W[f, i]$$

The expressive power of the latent variable space is related to its dimension. For a single latent variable, this representation boils down to the choice of the the most frequently recommended item, regardless of the consumer. Increasing number of variables allows the introduction of personalization, and the quality of the recommendations resulting from the value of ratings increases, and above a certain number of variables begins to tend to overfit. To avoid overfitting, regularization is used, which involves adding a regularization component to the evaluation function.

The Funk MF method minimizes the evaluation function:

$$\text{argmin}_{H,W}||R - \tilde{R}||_F + \alpha||H|| + \beta||W||$$

Where $||.||_F$ is the Frobenius norm which is some generalization of the Euclidean norm.

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

4. Pedro Domingos: Data Mining, Machine learning, a collection of video lectures available by Youtube, Paul G. Allen School of Computer Science & Engineering, University of Washington, 2016

5. Wikipedia: Collaborative filtering
`https://en.wikipedia.org/wiki/Collaborative_filtering`

6. Wikipedia: Matrix factorization (recommender systems)
`https://en.wikipedia.org/wiki/Matrix_factorization_(recommender_systems)`