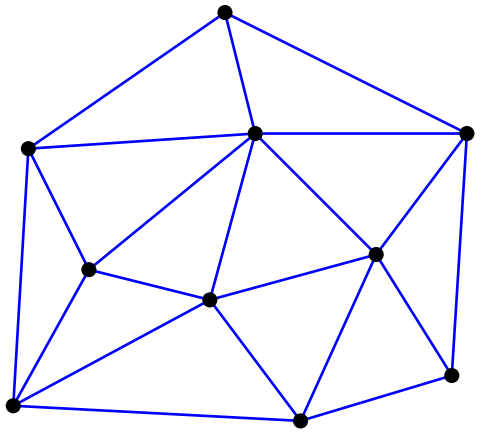
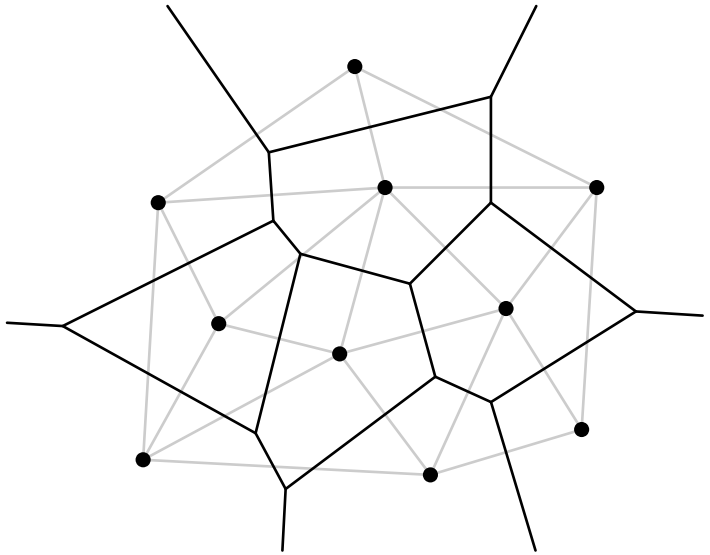
Learning Vector Quantization

Learning Vector Quantization

- Up to now: **fixed learning tasks**
  - The data consists of input/output pairs.
  - The objective is to produce desired output for given input.
  - This allows to describe training as error minimization.
- Now: **free learning tasks**
  - The data consists only of input values/vectors.
  - The objective is to produce similar output for similar input (clustering).
- **Learning Vector Quantization**
  - Find a suitable quantization (many-to-few mapping, often to a finite set) of the input space, e.g. a tesselation of a Euclidean space.
  - Training adapts the coordinates of so-called reference or codebook vectors, each of which defines a region in the input space.

Reminder: Delaunay Triangulations and Voronoi Diagrams

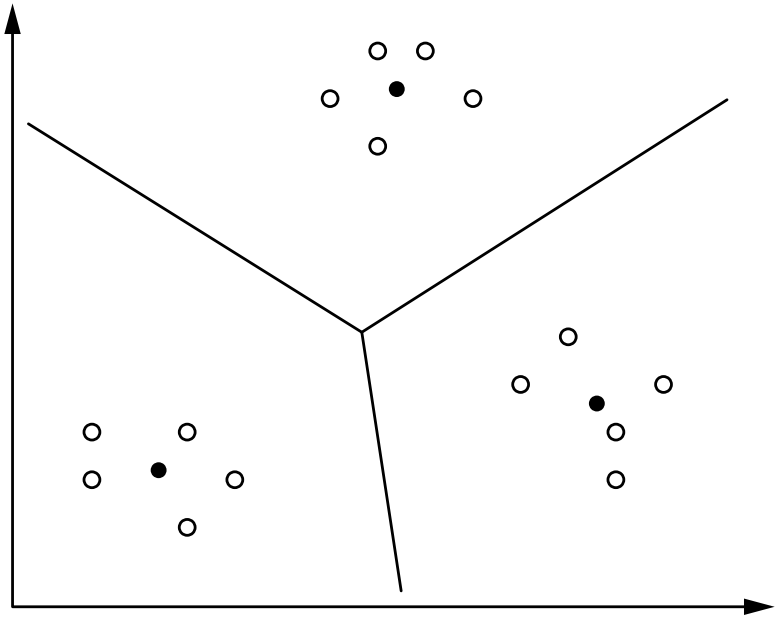




- Dots represent vectors that are used for quantizing the area.
- Left: **Delaunay Triangulation**
  (The circle through the corners of a triangle does not contain another point.)
- Right: **Voronoi Diagram / Tesselation**
  (Midperpendiculars of the Delaunay triangulation: boundaries of the regions of points that are closest to the enclosed cluster center (Voronoi cells)).

Learning Vector Quantization

Finding clusters in a given set of data points



- Data points are represented by empty circles (○).
- Cluster centers are represented by full circles (●).

Learning Vector Quantization Networks

A **learning vector quantization network (LVQ)** is a neural network with a graph $G = (U, C)$ that satisfies the following conditions

(i) $U_{\text{in}} \cap U_{\text{out}} = \emptyset$, $U_{\text{hidden}} = \emptyset$

(ii) $C = U_{\text{in}} \times U_{\text{out}}$

The network input function of each output neuron is a **distance function** of the input vector and the weight vector, that is,

$$\forall u \in U_{\text{out}}: \qquad f_{\text{net}}^{(u)}(\vec{w}_u, \vec{\text{in}}_u) = d(\vec{w}_u, \vec{\text{in}}_u),$$

where $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_0^+$ is a function satisfying $\forall \vec{x}, \vec{y}, \vec{z} \in \mathbb{R}^n$:

$$\begin{aligned}
&(i) && d(\vec{x}, \vec{y}) = 0 \quad \Leftrightarrow \quad \vec{x} = \vec{y}, && \\
&(ii) && d(\vec{x}, \vec{y}) = d(\vec{y}, \vec{x}) && \text{(symmetry)}, \\
&(iii) && d(\vec{x}, \vec{z}) \leq d(\vec{x}, \vec{y}) + d(\vec{y}, \vec{z}) && \text{(triangle inequality)}.
\end{aligned}$$

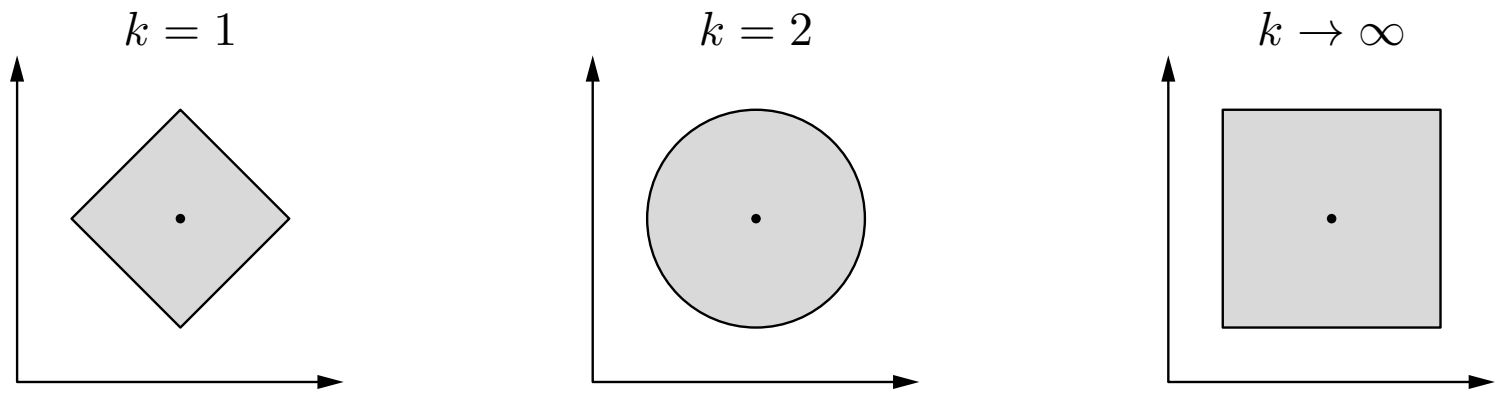
Reminder: Distance Functions

Illustration of distance functions: Minkowski family

$$d_k(\vec{x}, \vec{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^k \right)^{\frac{1}{k}}$$

Well-known special cases from this family are:

$k = 1:$	Manhattan or city block distance,
$k = 2:$	Euclidean distance,
$k \to \infty:$	maximum distance, that is, $d_\infty(\vec{x}, \vec{y}) = \max_{i=1}^{n} \lvert x_i - y_i \rvert$.



Learning Vector Quantization

The activation function of each output neuron is a so-called **radial function**, that is, a monotonically decreasing function

$$f : \mathbb{R}_0^+ \to [0, \infty] \quad \text{with} \quad f(0) = 1 \quad \text{and} \quad \lim_{x \to \infty} f(x) = 0.$$

Sometimes the range of values is restricted to the interval $[0, 1]$.
However, due to the special output function this restriction is irrelevant.

The output function of each output neuron is not a simple function of the activation of the neuron. Rather it takes into account the activations of all output neurons:

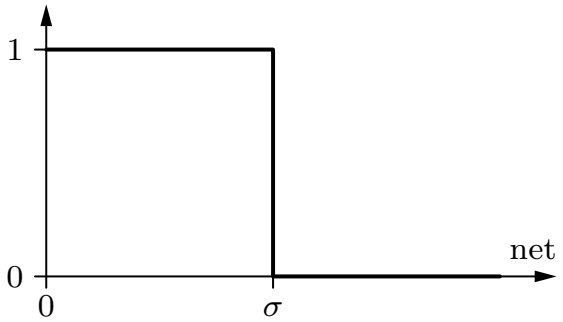
$$f_{\text{out}}^{(u)}(\text{act}_u) = \begin{cases} 1, & \text{if } \text{act}_u = \max\limits_{v \in U_{\text{out}}} \text{act}_v, \\ 0, & \text{otherwise.} \end{cases}$$

If more than one unit has the maximal activation, one is selected at random to have an output of 1, all others are set to output 0: **winner-takes-all principle**.

Radial Activation Functions

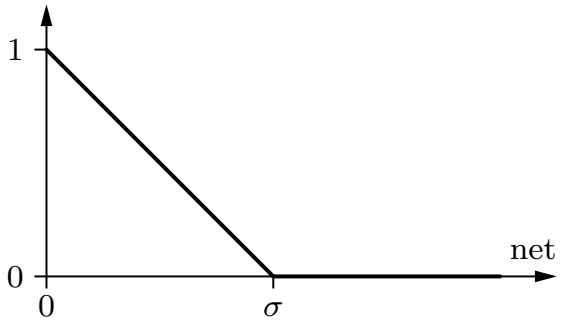
rectangle function:

$$f_{\text{act}}(\text{net}, \sigma) = \begin{cases} 0, & \text{if net} > \sigma, \\ 1, & \text{otherwise.} \end{cases}$$



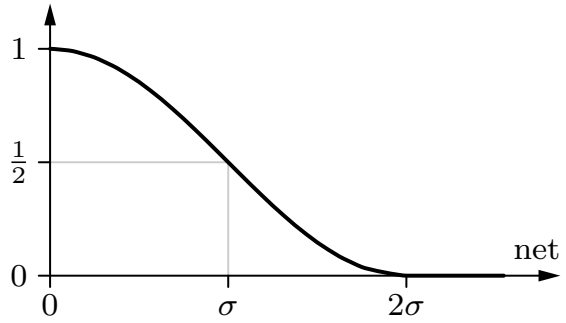
triangle function:

$$f_{\text{act}}(\text{net}, \sigma) = \begin{cases} 0, & \text{if net} > \sigma, \\ 1 - \frac{\text{net}}{\sigma}, & \text{otherwise.} \end{cases}$$



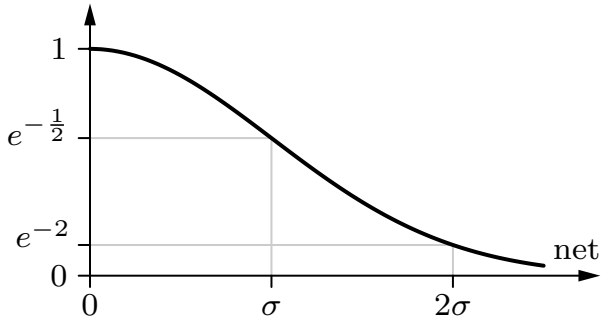
cosine until zero:

$$f_{\text{act}}(\text{net}, \sigma) = \begin{cases} 0, & \text{if net} > 2\sigma, \\ \frac{\cos\left(\frac{\pi}{2\sigma}\,\text{net}\right)+1}{2}, & \text{otherwise.} \end{cases}$$



Gaussian function:

$$f_{\text{act}}(\text{net}, \sigma) = e^{-\frac{\text{net}^2}{2\sigma^2}}$$



Learning Vector Quantization

Adaptation of reference vectors / codebook vectors

- For each training pattern find the closest reference vector.
- Adapt only this reference vector (winner neuron).
- For classified data the class may be taken into account:
  Each reference vector is assigned to a class.

Attraction rule (data point and reference vector have same class)

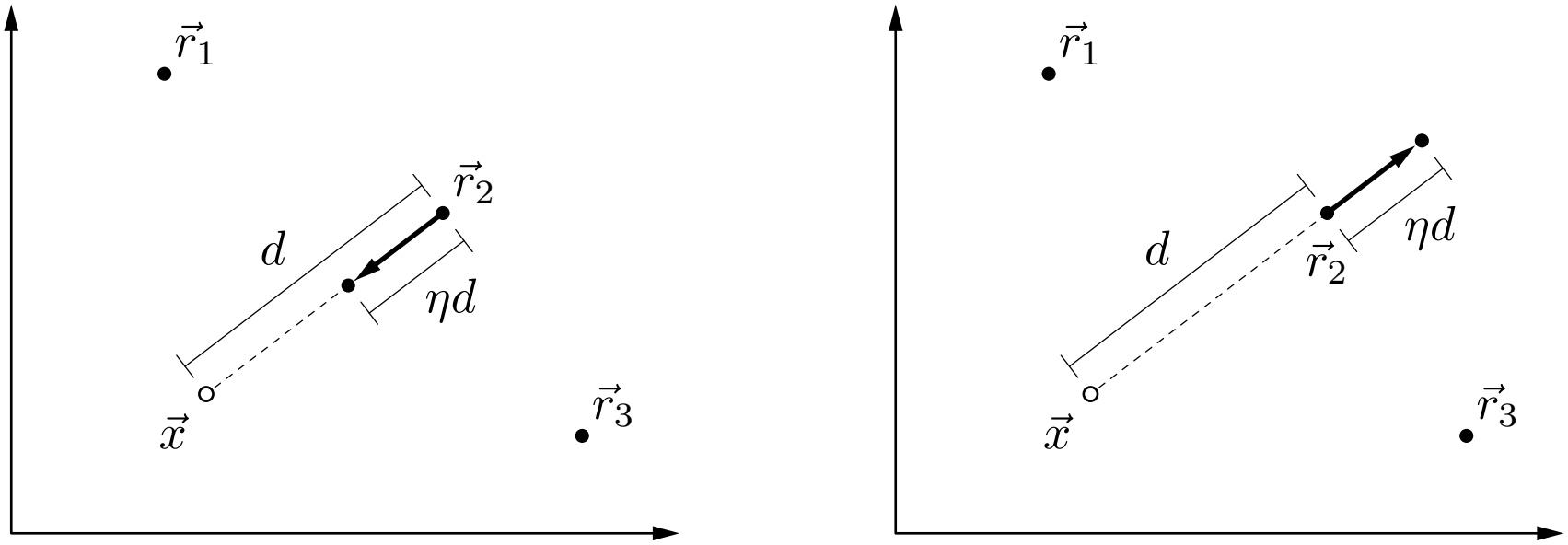
$$\vec{r}^{\,(\text{new})} = \vec{r}^{\,(\text{old})} + \eta(\vec{x} - \vec{r}^{\,(\text{old})}),$$

Repulsion rule (data point and reference vector have different class)

$$\vec{r}^{\,(\text{new})} = \vec{r}^{\,(\text{old})} - \eta(\vec{x} - \vec{r}^{\,(\text{old})}).$$

Learning Vector Quantization

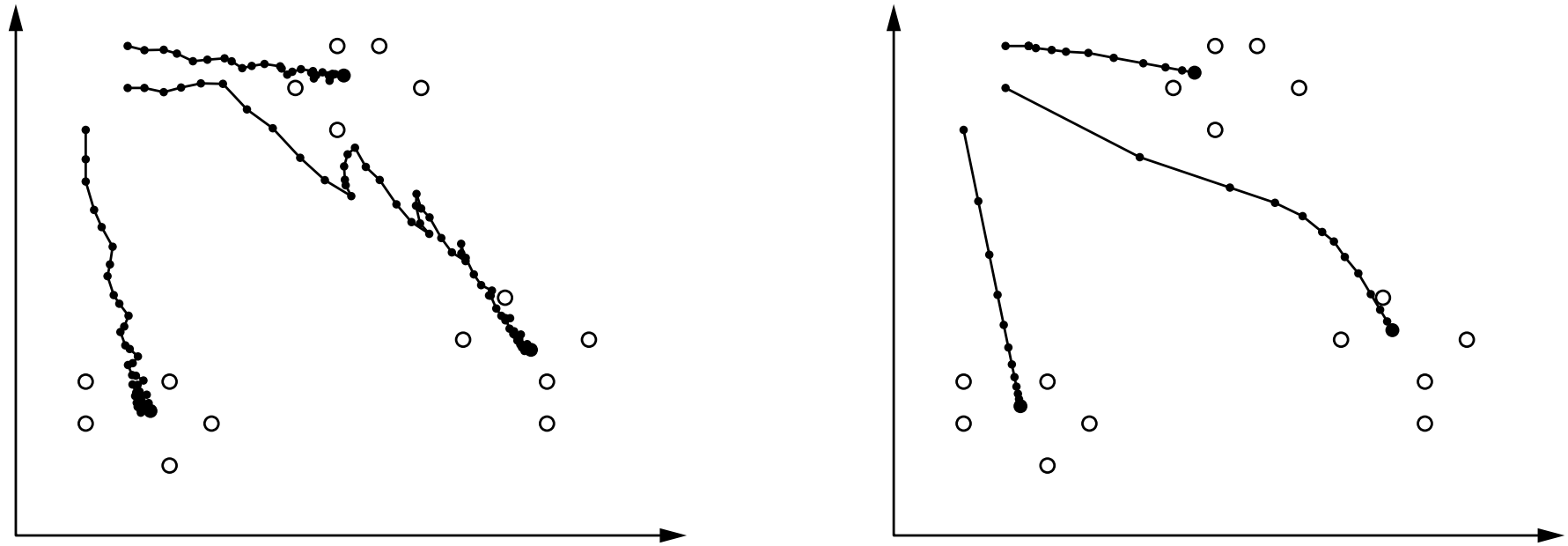
Adaptation of reference vectors / codebook vectors



- $\vec{x}$: data point, $\vec{r}_i$: reference vector
- $\eta = 0.4$ (learning rate)

Learning Vector Quantization: Example

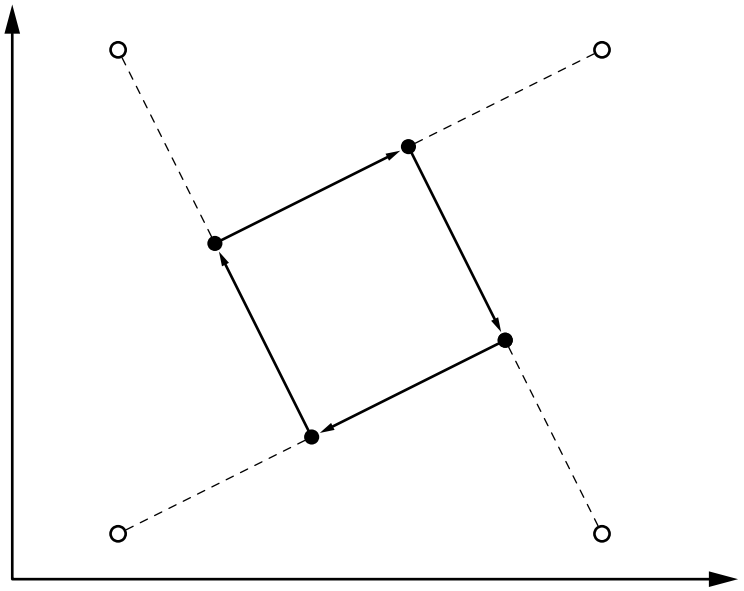
Adaptation of reference vectors / codebook vectors

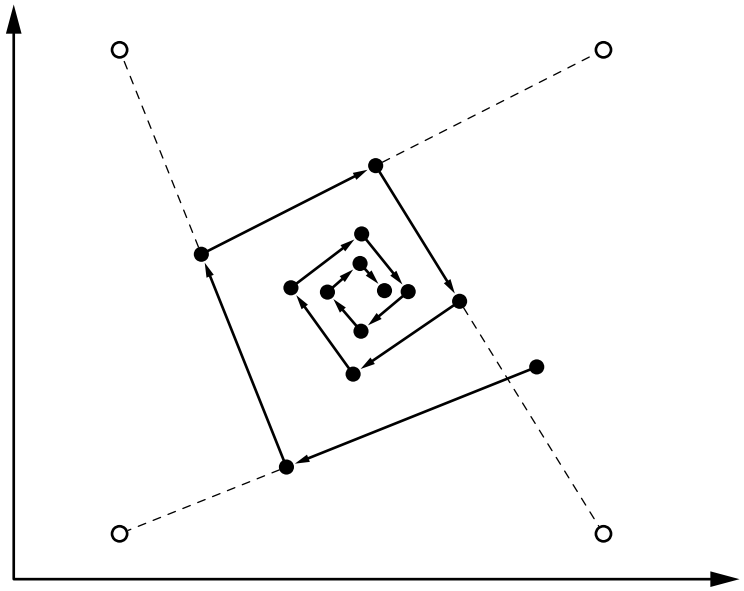


- Left: Online training with learning rate $\eta = 0.1$,
- Right: Batch training with learning rate $\eta = 0.05$.

Learning Vector Quantization: Learning Rate Decay

Problem: fixed learning rate can lead to oscillations





Solution: **time dependent learning rate**

$$\eta(t) = \eta_0 \alpha^t, \quad 0 < \alpha < 1, \qquad \text{or} \qquad \eta(t) = \eta_0 t^\kappa, \quad \kappa < 0.$$

Learning Vector Quantization: Classified Data

Improved update rule for classified data

- **Idea:** Update not only the one reference vector that is closest to the data point (the winner neuron), but **update the two closest reference vectors**.
- Let $\vec{x}$ be the currently processed data point and c its class.
  Let $\vec{r}_j$ and $\vec{r}_k$ be the two closest reference vectors and z_j and z_k their classes.
- Reference vectors are updated only if $z_j \neq z_k$ and either $c = z_j$ or $c = z_k$.
  (Without loss of generality we assume $c = z_j$.)
  The **update rules** for the two closest reference vectors are:

$$\begin{aligned} \vec{r}_j^{\,(\text{new})} &= \vec{r}_j^{\,(\text{old})} + \eta(\vec{x} - \vec{r}_j^{\,(\text{old})}) \qquad \text{and} \\ \vec{r}_k^{\,(\text{new})} &= \vec{r}_k^{\,(\text{old})} - \eta(\vec{x} - \vec{r}_k^{\,(\text{old})}), \end{aligned}$$

  while all other reference vectors remain unchanged.

Learning Vector Quantization: Window Rule

- It was observed in practical tests that standard learning vector quantization may drive the reference vectors further and further apart.
- To counteract this undesired behavior a **window rule** was introduced: update only if the data point $\vec{x}$ is close to the classification boundary.
- "Close to the boundary" is made formally precise by requiring

$$\min\left(\frac{d(\vec{x},\vec{r}_j)}{d(\vec{x},\vec{r}_k)}, \frac{d(\vec{x},\vec{r}_k)}{d(\vec{x},\vec{r}_j)}\right) > \theta, \qquad \text{where} \qquad \theta = \frac{1-\xi}{1+\xi}.$$

  ξ is a parameter that has to be specified by a user.
- Intuitively, ξ describes the "width" of the window around the classification boundary, in which the data point has to lie in order to lead to an update.
- Using it prevents divergence, because the update ceases for a data point once the classification boundary has been moved far enough away.

Soft Learning Vector Quantization

- **Idea:** Use soft assignments instead of winner-takes-all (approach descibed here: [Seo and Obermayer 2003]).
- **Assumption:** Given data was sampled from a mixture of normal distributions. Each reference vector describes one normal distribution.
- Closely related to clustering by estimating a **mixture of Gaussians**.
  - (Crisp or hard) learning vector quantization can be seen as an "online version" of c-means clustering.
  - Soft learning vector quantization can be seed as an "online version" of estimating a mixture of Gaussians (that is, of normal distributions). (In the following: brief review of the Expectation Maximization (EM) Algorithm for estimating a mixture of Gaussians.)
- Hardening soft learning vector quantization (by letting the "radii" of the Gaussians go to zero, see below) yields a version of (crisp or hard) learning vector quantization that works well without a window rule.

Expectation Maximization: Mixture of Gaussians

- **Assumption:** Data was generated by sampling a set of normal distributions. (The probability density is a mixture of Gaussian distributions.)
- **Formally:** We assume that the probability density can be described as

$$f_{\vec{X}}(\vec{x}; \mathbf{C}) = \sum_{y=1}^{c} f_{\vec{X},Y}(\vec{x}, y; \mathbf{C}) = \sum_{y=1}^{c} p_Y(y; \mathbf{C}) \cdot f_{\vec{X}|Y}(\vec{x}|y; \mathbf{C}).$$

$\mathbf{C}$	is the set of cluster parameters
$\vec{X}$	is a random vector that has the data space as its domain
Y	is a random variable that has the cluster indices as possible values (i.e., $\mathrm{dom}(\vec{X}) = \mathbb{R}^m$ and $\mathrm{dom}(Y) = \{1, \ldots, c\}$)
$p_Y(y; \mathbf{C})$	is the probability that a data point belongs to (is generated by) the y-th component of the mixture
$f_{\vec{X}\|Y}(\vec{x}\|y; \mathbf{C})$	is the conditional probability density function of a data point given the cluster (specified by the cluster index y)

Expectation Maximization

- **Basic idea:** Do a maximum likelihood estimation of the cluster parameters.
- **Problem:** The likelihood function,

$$L(\mathbf{X}; \mathbf{C}) = \prod_{j=1}^{n} f_{\vec{X}_j}(\vec{x}_j; \mathbf{C}) = \prod_{j=1}^{n} \sum_{y=1}^{c} p_Y(y; \mathbf{C}) \cdot f_{\vec{X}|Y}(\vec{x}_j|y; \mathbf{C}),$$

  is difficult to optimize, even if one takes the natural logarithm (cf. the maximum likelihood estimation of the parameters of a normal distribution), because

$$\ln L(\mathbf{X}; \mathbf{C}) = \sum_{j=1}^{n} \ln \sum_{y=1}^{c} p_Y(y; \mathbf{C}) \cdot f_{\vec{X}|Y}(\vec{x}_j|y; \mathbf{C})$$

  contains the natural logarithms of complex sums.
- **Approach:** Assume that there are "hidden" variables Y_j stating the clusters that generated the data points $\vec{x}_j$, so that the sums reduce to one term.
- **Problem:** Since the Y_j are hidden, we do not know their values.

Expectation Maximization

- **Formally:** Maximize the likelihood of the "completed" data set $(\mathbf{X}, \vec{y})$, where $\vec{y} = (y_1, \ldots, y_n)$ combines the values of the variables Y_j. That is,

$$L(\mathbf{X}, \vec{y}; \mathbf{C}) = \prod_{j=1}^{n} f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C}) = \prod_{j=1}^{n} p_{Y_j}(y_j; \mathbf{C}) \cdot f_{\vec{X}_j|Y_j}(\vec{x}_j|y_j; \mathbf{C}).$$

- **Problem:** Since the Y_j are hidden, the values y_j are unknown (and thus the factors $p_{Y_j}(y_j; \mathbf{C})$ cannot be computed).

- **Approach to find a solution nevertheless:**
  - See the Y_j as random variables (the values y_j are not fixed) and consider a probability distribution over the possible values.
  - As a consequence $L(\mathbf{X}, \vec{y}; \mathbf{C})$ becomes a random variable, even for a fixed data set $\mathbf{X}$ and fixed cluster parameters $\mathbf{C}$.
  - Try to **maximize the expected value** of $L(\mathbf{X}, \vec{y}; \mathbf{C})$ or $\ln L(\mathbf{X}, \vec{y}; \mathbf{C})$ (hence the name **expectation maximization**).

Expectation Maximization

- **Formally:** Find the cluster parameters as

$$\hat{\mathbf{C}} = \underset{\mathbf{C}}{\operatorname{argmax}}\, E([\ln] L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}),$$

  that is, maximize the expected likelihood

$$E(L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}) = \sum_{\vec{y} \in \{1,\ldots,c\}^n} p_{\vec{Y}|\mathcal{X}}(\vec{y}\,|\mathbf{X}; \mathbf{C}) \cdot \prod_{j=1}^{n} f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C})$$

  or, alternatively, maximize the expected log-likelihood

$$E(\ln L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}) = \sum_{\vec{y} \in \{1,\ldots,c\}^n} p_{\vec{Y}|\mathcal{X}}(\vec{y}\,|\mathbf{X}; \mathbf{C}) \cdot \sum_{j=1}^{n} \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C}).$$

- Unfortunately, these functionals are still difficult to optimize directly.
- **Solution:** Use the equation as an iterative scheme, fixing $\mathbf{C}$ in some terms (iteratively compute better approximations, similar to Heron's algorithm).

Excursion: Heron's Algorithm

- **Task:** Find the square root of a given number x, i.e., find $y = \sqrt{x}$.
- **Approach:** Rewrite the defining equation $y^2 = x$ as follows:

$$y^2 = x \quad \Leftrightarrow \quad 2y^2 = y^2 + x \quad \Leftrightarrow \quad y = \frac{1}{2y}(y^2 + x) \quad \Leftrightarrow \quad y = \frac{1}{2}\left(y + \frac{x}{y}\right).$$

- Use the resulting equation as an iteration formula, i.e., compute the sequence

$$y_{k+1} = \frac{1}{2}\left(y_k + \frac{x}{y_k}\right) \qquad \text{with} \qquad y_0 = 1.$$

- It can be shown that $0 \leq y_k - \sqrt{x} \leq y_{k-1} - y_n$ for $k \geq 2$.
  Therefore this iteration formula provides increasingly better approximations of the square root of x and thus is a safe and simple way to compute it.
  Ex.: $x = 2$: $y_0 = 1$, $y_1 = 1.5$, $y_2 \approx 1.41667$, $y_3 \approx 1.414216$, $y_4 \approx 1.414213$.
- Heron's algorithm converges very quickly and is often used in pocket calculators and microprocessors to implement the square root.

Expectation Maximization

- **Iterative scheme for expectation maximization:**

  Choose some initial set $\mathbf{C}_0$ of cluster parameters and then compute

$$\begin{aligned}
\mathbf{C}_{k+1} &= \operatorname*{argmax}_{\mathbf{C}} E(\ln L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}_k) \\
&= \operatorname*{argmax}_{\mathbf{C}} \sum_{\vec{y} \in \{1,\ldots,c\}^n} p_{\vec{Y}|\mathcal{X}}(\vec{y}\,|\mathbf{X}; \mathbf{C}_k) \sum_{j=1}^{n} \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C}) \\
&= \operatorname*{argmax}_{\mathbf{C}} \sum_{\vec{y} \in \{1,\ldots,c\}^n} \left(\prod_{l=1}^{n} p_{Y_l|\vec{X}_l}(y_l|\vec{x}_l; \mathbf{C}_k) \right) \sum_{j=1}^{n} \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C}) \\
&= \operatorname*{argmax}_{\mathbf{C}} \sum_{i=1}^{c} \sum_{j=1}^{n} p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k) \cdot \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, i; \mathbf{C}).
\end{aligned}$$

- It can be shown that each EM iteration increases the likelihood of the data and that the algorithm converges to a local maximum of the likelihood function (i.e., EM is a safe way to maximize the likelihood function).

Expectation Maximization

Justification of the last step on the previous slide:

$$\begin{aligned}
&\sum_{\vec{y}\in\{1,\ldots,c\}^n} \left(\prod_{l=1}^{n} p_{Y_l|\vec{X}_l}(y_l|\vec{x}_l;\mathbf{C}_k)\right) \sum_{j=1}^{n} \ln f_{\vec{X}_j,Y_j}(\vec{x}_j,y_j;\mathbf{C}) \\
&= \sum_{y_1=1}^{c} \cdots \sum_{y_n=1}^{c} \left(\prod_{l=1}^{n} p_{Y_l|\vec{X}_l}(y_l|\vec{x}_l;\mathbf{C}_k)\right) \sum_{j=1}^{n}\sum_{i=1}^{c} \delta_{i,y_j} \ln f_{\vec{X}_j,Y_j}(\vec{x}_j,i;\mathbf{C}) \\
&= \sum_{i=1}^{c}\sum_{j=1}^{n} \ln f_{\vec{X}_j,Y_j}(\vec{x}_j,i;\mathbf{C}) \sum_{y_1=1}^{c} \cdots \sum_{y_n=1}^{c} \delta_{i,y_j} \prod_{l=1}^{n} p_{Y_l|\vec{X}_l}(y_l|\vec{x}_l;\mathbf{C}_k) \\
&= \sum_{i=1}^{c}\sum_{j=1}^{n} p_{Y_j|\vec{X}_j}(i|\vec{x}_j;\mathbf{C}_k) \cdot \ln f_{\vec{X}_j,Y_j}(\vec{x}_j,i;\mathbf{C}) \\
&\quad \underbrace{\sum_{y_1=1}^{c} \cdots \sum_{y_{j-1}=1}^{c} \sum_{y_{j+1}=1}^{c} \cdots \sum_{y_n=1}^{c} \prod_{l=1,l\neq j}^{n} p_{Y_l|\vec{X}_l}(y_l|\vec{x}_l;\mathbf{C}_k)}_{= \prod_{l=1,l\neq j}^{n} \sum_{y_l=1}^{c} p_{Y_l|\vec{X}_l}(y_l|\vec{x}_l;\mathbf{C}_k) \;=\; \prod_{l=1,l\neq j}^{n} 1 \;=\; 1} .
\end{aligned}$$

Expectation Maximization

- The probabilities $p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k)$ are computed as

$$p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k) = \frac{f_{\vec{X}_j,Y_j}(\vec{x}_j, i; \mathbf{C}_k)}{f_{\vec{X}_j}(\vec{x}_j; \mathbf{C}_k)} = \frac{f_{\vec{X}_j|Y_j}(\vec{x}_j|i; \mathbf{C}_k) \cdot p_{Y_j}(i; \mathbf{C}_k)}{\sum_{l=1}^{c} f_{\vec{X}_j|Y_j}(\vec{x}_j|l; \mathbf{C}_k) \cdot p_{Y_j}(l; \mathbf{C}_k)},$$

  that is, as the relative probability densities of the different clusters (as specified by the cluster parameters) at the location of the data points $\vec{x}_j$.

- The $p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k)$ are the posterior probabilities of the clusters given the data point $\vec{x}_j$ and a set of cluster parameters $\mathbf{C}_k$.

- They can be seen as **case weights** of a "completed" data set:
  - Split each data point $\vec{x}_j$ into c data points $(\vec{x}_j, i)$, $i = 1, \ldots, c$.
  - Distribute the unit weight of the data point $\vec{x}_j$ according to the above probabilities, i.e., assign to $(\vec{x}_j, i)$ the weight $p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k)$, $i = 1, \ldots, c$.

Expectation Maximization: Cookbook Recipe

Core Iteration Formula

$$\mathbf{C}_{k+1} = \underset{\mathbf{C}}{\operatorname{argmax}} \sum_{i=1}^{c} \sum_{j=1}^{n} p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k) \cdot \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, i; \mathbf{C})$$

Expectation Step

- For all data points $\vec{x}_j$:
  Compute for each normal distribution the probability $p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k)$
  that the data point was generated from it
  (ratio of probability densities at the location of the data point).
  $\rightarrow$ "weight" of the data point for the estimation.

Maximization Step

- For all normal distributions:
  Estimate the parameters by standard maximum likelihood estimation
  using the probabilities ("weights") assigned to the data points
  w.r.t. the distribution in the expectation step.

Expectation Maximization: Mixture of Gaussians

Expectation Step: Use Bayes' rule to compute

$$p_{C|\vec{X}}(i|\vec{x};\mathbf{C}) = \frac{p_C(i;\mathbf{c}_i)\cdot f_{\vec{X}|C}(\vec{x}|i;\mathbf{c}_i)}{f_{\vec{X}}(\vec{x};\mathbf{C})} = \frac{p_C(i;\mathbf{c}_i)\cdot f_{\vec{X}|C}(\vec{x}|i;\mathbf{c}_i)}{\sum_{k=1}^{c} p_C(k;\mathbf{c}_k)\cdot f_{\vec{X}|C}(\vec{x}|k;\mathbf{c}_k)}.$$

$\rightarrow$ "weight" of the data point $\vec{x}$ for the estimation.

Maximization Step: Use maximum likelihood estimation to compute

$$\varrho_i^{(t+1)} = \frac{1}{n}\sum_{j=1}^{n} p_{C|\vec{X}_j}(i|\vec{x}_j;\mathbf{C}^{(t)}), \qquad \vec{\mu}_i^{(t+1)} = \frac{\sum_{j=1}^{n} p_{C|\vec{X}_j}(i|\vec{x}_j;\mathbf{C}^{(t)})\cdot\vec{x}_j}{\sum_{j=1}^{n} p_{C|\vec{X}_j}(i|\vec{x}_j;\mathbf{C}^{(t)})},$$

$$\text{and} \qquad \mathbf{\Sigma}_i^{(t+1)} = \frac{\sum_{j=1}^{n} p_{C|\vec{X}_j}(i|\vec{x}_j;\mathbf{C}^{(t)})\cdot\left(\vec{x}_j-\vec{\mu}_i^{(t+1)}\right)\left(\vec{x}_j-\vec{\mu}_i^{(t+1)}\right)^\top}{\sum_{j=1}^{n} p_{C|\vec{X}_j}(i|\vec{x}_j;\mathbf{C}^{(t)})}$$

Iterate until convergence (checked, e.g., by change of mean vector).

Expectation Maximization: Technical Problems

- If a fully general mixture of Gaussian distributions is used,
  the likelihood function is truly optimized if
  - all normal distributions except one are contracted to single data points and
  - the remaining normal distribution is the maximum likelihood estimate for the remaining data points.
- This undesired result is rare,
  because the algorithm gets stuck in a local optimum.
- Nevertheless it is recommended to take countermeasures,
  which consist mainly in reducing the degrees of freedom, like
  - Fix the determinants of the covariance matrices to equal values.
  - Use a diagonal instead of a general covariance matrix.
  - Use an isotropic variance instead of a covariance matrix.
  - Fix the prior probabilities of the clusters to equal values.

Soft Learning Vector Quantization

Idea: Use soft assignments instead of winner-takes-all (approach descibed here: [Seo and Obermayer 2003]).

Assumption: Given data was sampled from a mixture of normal distributions. Each reference vector describes one normal distribution.

Objective: Maximize the log-likelihood ratio of the data, that is, maximize

$$\ln L_{\text{ratio}} = \sum_{j=1}^{n} \ln \sum_{\vec{r} \in R(c_j)} \exp\left(-\frac{(\vec{x}_j - \vec{r})^\top (\vec{x}_j - \vec{r})}{2\sigma^2}\right) - \sum_{j=1}^{n} \ln \sum_{\vec{r} \in Q(c_j)} \exp\left(-\frac{(\vec{x}_j - \vec{r})^\top (\vec{x}_j - \vec{r})}{2\sigma^2}\right).$$

Here σ is a parameter specifying the "size" of each normal distribution.
$R(c)$ is the set of reference vectors assigned to class c and $Q(c)$ its complement.

Intuitively: at each data point the probability density for its class should be as large as possible while the density for all other classes should be as small as possible.

Soft Learning Vector Quantization

Update rule derived from a maximum log-likelihood approach:

$$\vec{r}_i^{\,(\text{new})} = \vec{r}_i^{\,(\text{old})} + \eta \cdot \begin{cases} u_{ij}^{\oplus} \cdot (\vec{x}_j - \vec{r}_i^{\,(\text{old})}), & \text{if } c_j = z_i, \\ -u_{ij}^{\ominus} \cdot (\vec{x}_j - \vec{r}_i^{\,(\text{old})}), & \text{if } c_j \neq z_i, \end{cases}$$

where z_i is the class associated with the reference vector $\vec{r}_i$ and

$$u_{ij}^{\oplus} = \frac{\exp\left(-\frac{1}{2\sigma^2}(\vec{x}_j - \vec{r}_i^{\,(\text{old})})^\top (\vec{x}_j - \vec{r}_i^{\,(\text{old})})\right)}{\sum_{\vec{r} \in R(c_j)} \exp\left(-\frac{1}{2\sigma^2}(\vec{x}_j - \vec{r}^{\,(\text{old})})^\top (\vec{x}_j - \vec{r}^{\,(\text{old})})\right)} \quad \text{and}$$

$$u_{ij}^{\ominus} = \frac{\exp\left(-\frac{1}{2\sigma^2}(\vec{x}_j - \vec{r}_i^{\,(\text{old})})^\top (\vec{x}_j - \vec{r}_i^{\,(\text{old})})\right)}{\sum_{\vec{r} \in Q(c_j)} \exp\left(-\frac{1}{2\sigma^2}(\vec{x}_j - \vec{r}^{\,(\text{old})})^\top (\vec{x}_j - \vec{r}^{\,(\text{old})})\right)}.$$

$R(c)$ is the set of reference vectors assigned to class c and $Q(c)$ its complement.

Hard Learning Vector Quantization

Idea: Derive a scheme with hard assignments from the soft version.

Approach: Let the size parameter σ of the Gaussian function go to zero.

The resulting update rule is in this case:

$$\vec{r}_i^{\,(\text{new})} = \vec{r}_i^{\,(\text{old})} + \eta \cdot \begin{cases} u_{ij}^{\oplus} \cdot (\vec{x}_j - \vec{r}_i^{\,(\text{old})}), & \text{if } c_j = z_i, \\ -u_{ij}^{\ominus} \cdot (\vec{x}_j - \vec{r}_i^{\,(\text{old})}), & \text{if } c_j \neq z_i, \end{cases}$$

where

$$u_{ij}^{\oplus} = \begin{cases} 1, & \text{if } \vec{r}_i = \underset{\vec{r} \in R(c_j)}{\operatorname{argmin}}\, d(\vec{x}_j, \vec{r}), \\ 0, & \text{otherwise,} \end{cases} \qquad u_{ij}^{\ominus} = \begin{cases} 1, & \text{if } \vec{r}_i = \underset{\vec{r} \in Q(c_j)}{\operatorname{argmin}}\, d(\vec{x}_j, \vec{r}), \\ 0, & \text{otherwise.} \end{cases}$$

$\vec{r}_i$ is closest vector of same class

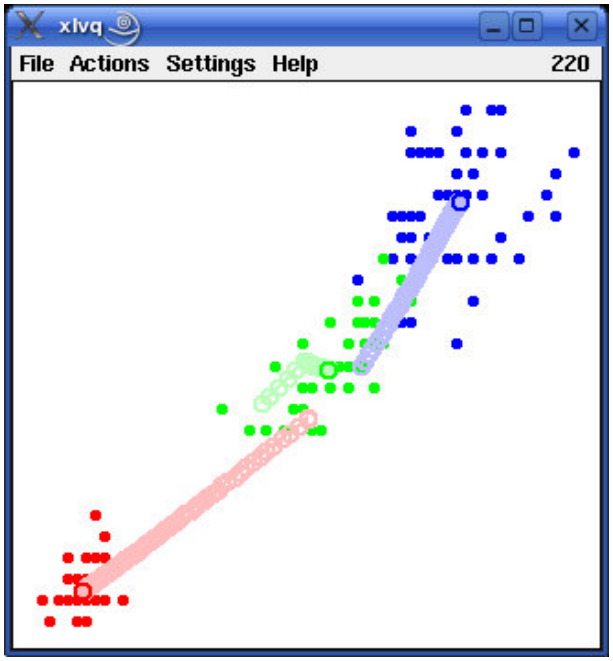
$\vec{r}_i$ is closest vector of different class

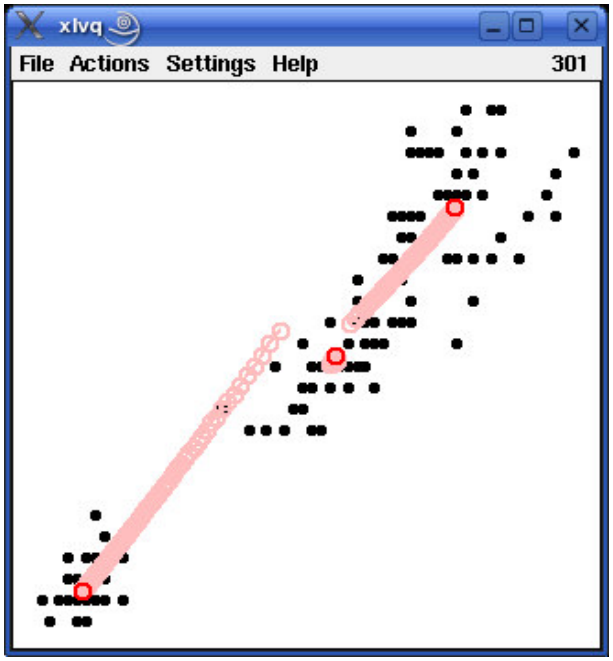
This update rule is stable without a *window rule* restricting the update.

Learning Vector Quantization: Extensions

- **Frequency Sensitive Competitive Learning**
  - The distance to a reference vector is modified according to the number of data points that are assigned to this reference vector.
- **Fuzzy Learning Vector Quantization**
  - Exploits the close relationship to fuzzy clustering.
  - Can be seen as an online version of fuzzy clustering.
  - Leads to faster clustering.
- **Size and Shape Parameters**
  - Associate each reference vector with a cluster radius. Update this radius depending on how close the data points are.
  - Associate each reference vector with a covariance matrix. Update this matrix depending on the distribution of the data points.

Demonstration Software: xlvq/wlvq





Demonstration of learning vector quantization:

- Visualization of the training process
- Arbitrary datasets, but training only in two dimensions
- `http://www.borgelt.net/lvqd.html`