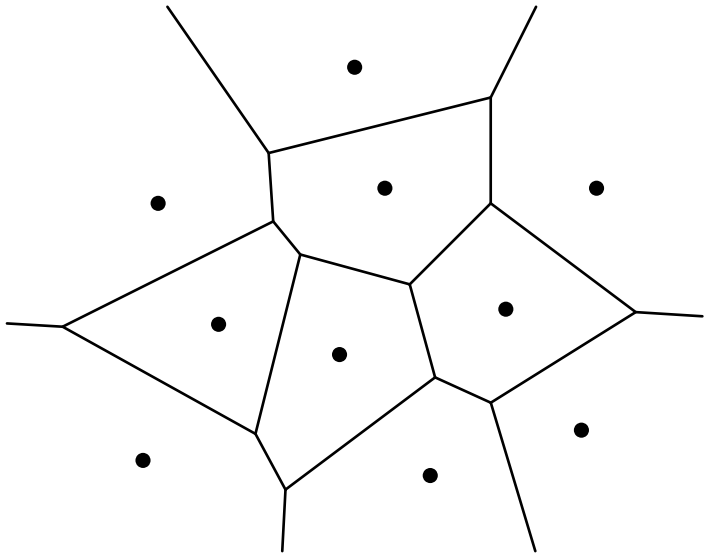
Lernende Vektorquantisierung

(engl. Learning Vector Quantization)

Motivation

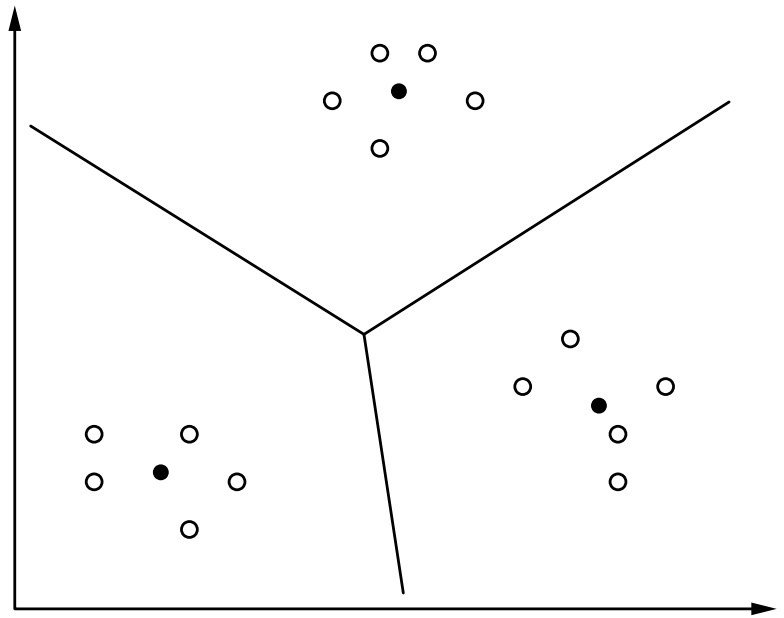
- Bisher: festes Lernen, jetzt freies Lernen, d.h. es existieren keine festgelegten Klassenlabels oder Zielwerte mehr für jedes Lernbeispiel
- Grundidee: ähnliche Eingaben führen zu ähnlichen Ausgaben
- Ähnlichkeit zum Clustering: benachbarte (ähnliche) Datenpunkte im Eingaberaum liegen auch im Ausgaberaum benachbart

Voronoidiagramm einer Vektorquantisierung



- Punkte repräsentieren Vektoren, die zur Quantisierung der Fläche genutzt werden.
- Linien sind die Grenzen der Regionen, deren Punkte am nächsten zu dem dargestellten Vektor liegen.

Finden von Clustern in einer gegebenen Menge von Punkten



- Datenpunkte werden durch leere Kreise dargestellt (○).
- Clusterzentren werden durch gefüllte Kreise dargestellt (●).

Lernende Vektorquantisierung, Netzwerk

Ein **Lernendes Vektorquantisierungsnetzwerk (LVQ)** ist ein neuronales Netz mit einem Graphen $G = (U, C)$ das die folgenden Bedingungen erfüllt:

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

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

Die Netzeingabefunktion jedes Ausgabeneurons ist eine **Abstandsfunktion** zwischen Eingabe- und Gewichtsvektor, d.h.

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

wobei $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_0^+$ eine Funktion ist, die $\forall \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{R}^n$:

$$\begin{array}{rll} (i) & d(\boldsymbol{x}, \boldsymbol{y}) = 0 \quad \Leftrightarrow \quad \boldsymbol{x} = \boldsymbol{y}, & \\ (ii) & d(\boldsymbol{x}, \boldsymbol{y}) = d(\boldsymbol{y}, \boldsymbol{x}) & \text{(Symmetrie)}, \\ (iii) & d(\boldsymbol{x}, \boldsymbol{z}) \leq d(\boldsymbol{x}, \boldsymbol{y}) + d(\boldsymbol{y}, \boldsymbol{z}) & \text{(Dreiecksungleichung)} \end{array}$$

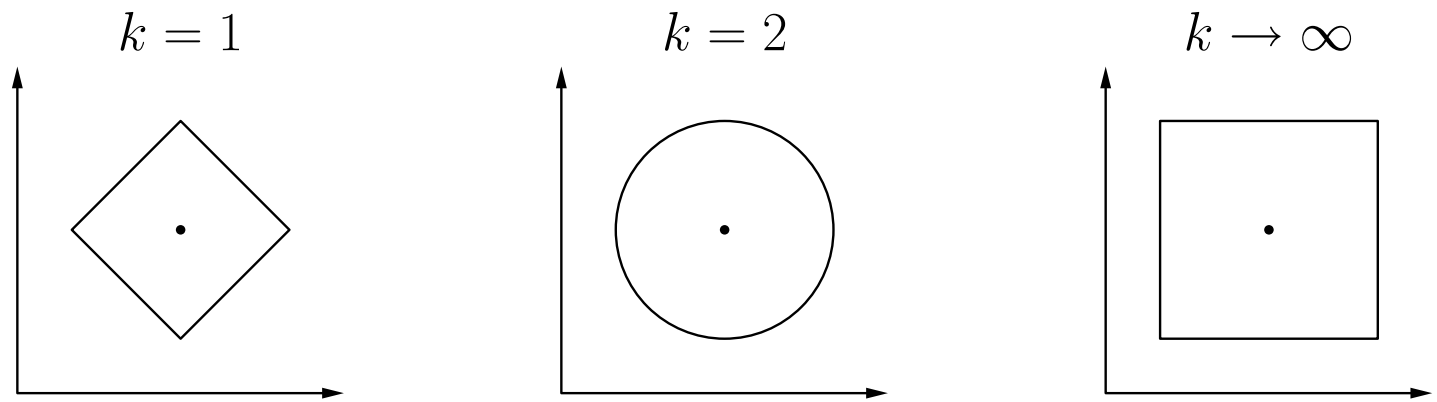
erfüllt.

Veranschaulichung von Abstandsfunktionen

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

Bekannte Spezialfälle:

$k = 1:$	Manhattan- oder City-Block-Abstand,
$k = 2:$	Euklidischer Abstand,
$k \to \infty:$	Maximum-Abstand, d.h. $d_\infty(\boldsymbol{x}, \boldsymbol{y}) = \max_{i=1}^{n} \lvert x_i - y_i \rvert$.



(alle Punkte auf dem Kreis bzw. den Vierecken haben denselben Abstand zum Mittelpunkt, entsprechend der jeweiligen Abstandsfunktion)

Die Aktivierungsfunktion jedes Ausgabeneurons ist eine sogenannte **radiale Funktion**, d.h. eine monoton fallende Funktion

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

Manchmal wird der Wertebereich auf das Intervall $[0, 1]$ beschränkt.
Durch die spezielle Ausgabefunktion ist das allerdings unerheblich.

Die Ausgabefunktion jedes Ausgabeneurons ist keine einfache Funktion der Aktivierung des Neurons. Sie zieht stattdessen alle Aktivierungen aller Ausgabeneuronen in Betracht:

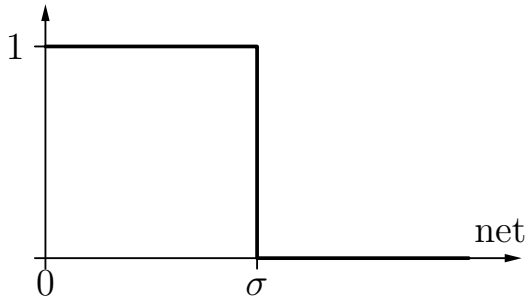
$$f_{\text{out}}^{(u)}(\text{act}_u) = \begin{cases} 1, & \text{falls } \text{act}_u = \max\limits_{v \in U_{\text{out}}} \text{act}_v, \\ 0, & \text{sonst.} \end{cases}$$

Sollte mehr als ein Neuron die maximale Aktivierung haben, wird ein zufällig gewähltes Neuron auf die Ausgabe 1 gesetzt, alle anderen auf Ausgabe 0: **Winner-Takes-All-Prinzip**.

Radiale Aktivierungfunktionen

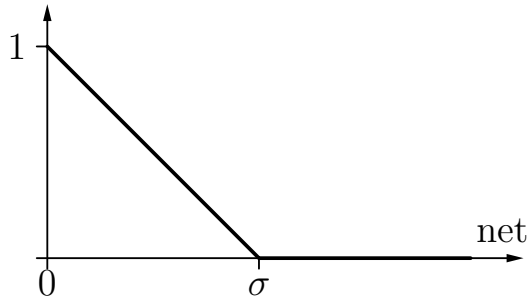
Rechteckfunktion:

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



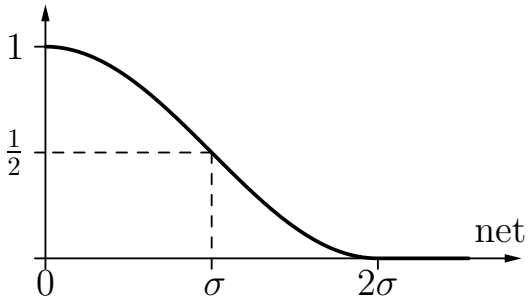
Dreiecksfunktion:

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



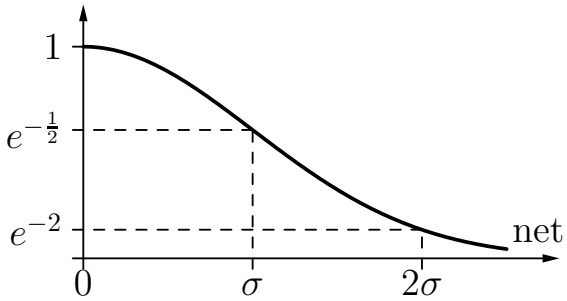
Kosinus bis Null:

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



Gauß-Funktion:

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



Anpassung der Referenzvektoren (Codebuch-Vektoren)

- Bestimme zu jedem Trainingsbeispiel den nächsten Referenzvektor.
- Passe nur diesen Referenzvektor an (Gewinnerneuron).
- Für Klassifikationsprobleme kann die Klasse genutzt werden:
  Jeder Referenzvektor wird einer Klasse zugeordnet.

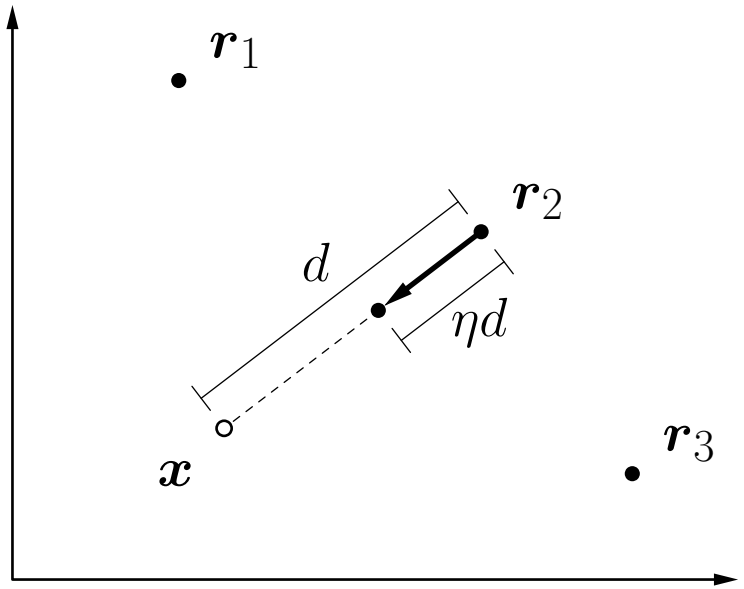
Anziehungsregel (Datenpunkt und Referenzvektor haben dieselbe Klasse)

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

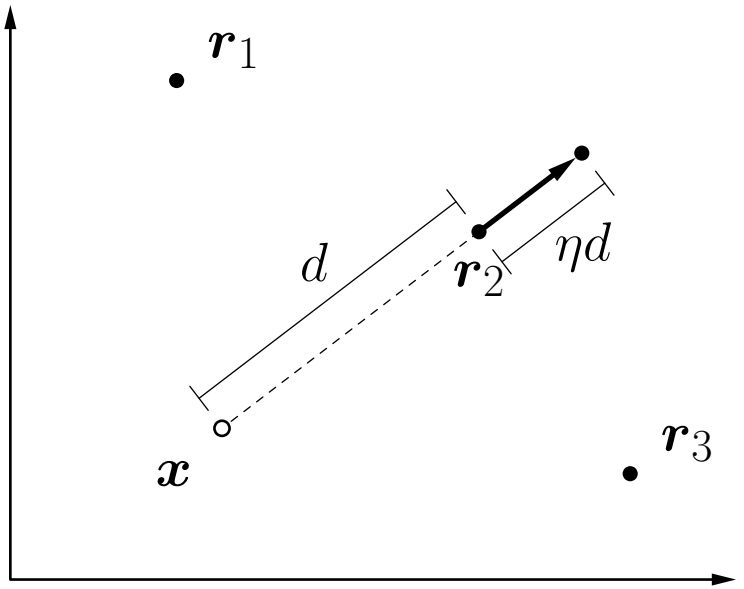
Abstoßungsregel (Datenpunkt und Referenzvektor haben verschiedene Klassen)

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

Anpassung der Referenzvektoren



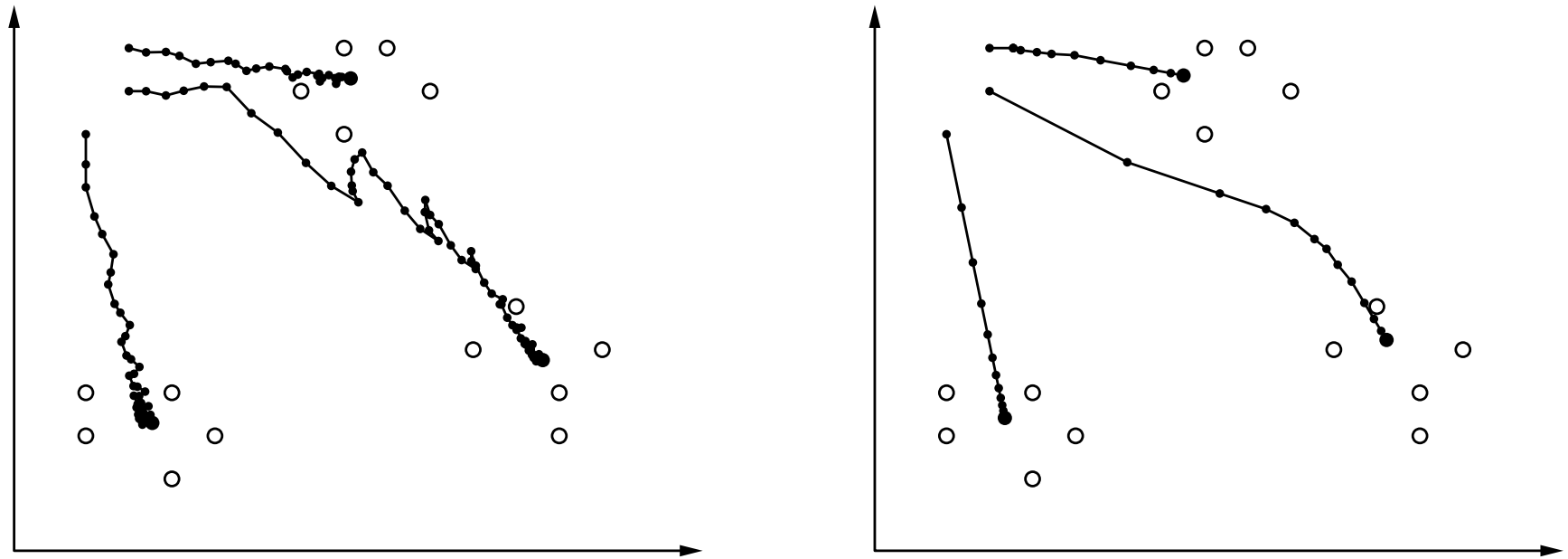
Anziehungsregel



Abstoßungsregel

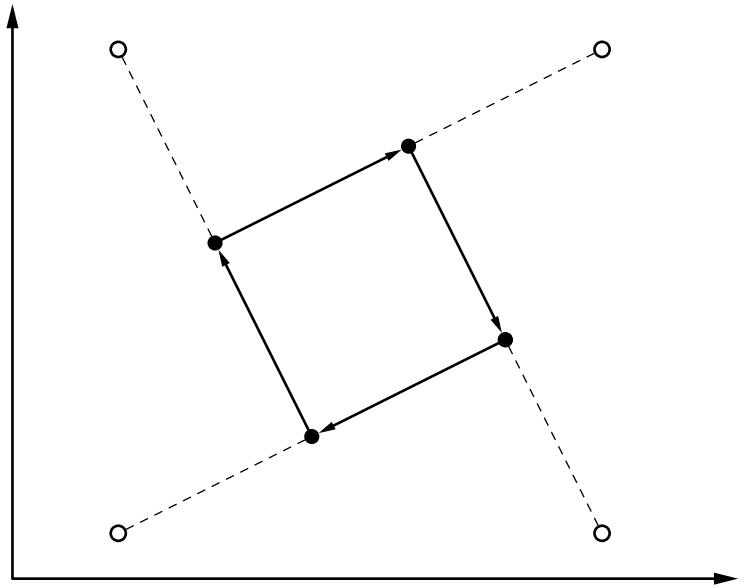
- $\boldsymbol{x}$: Datenpunkt, $\boldsymbol{r}_i$: Referenzvektor
- $\eta = 0.4$ (Lernrate)

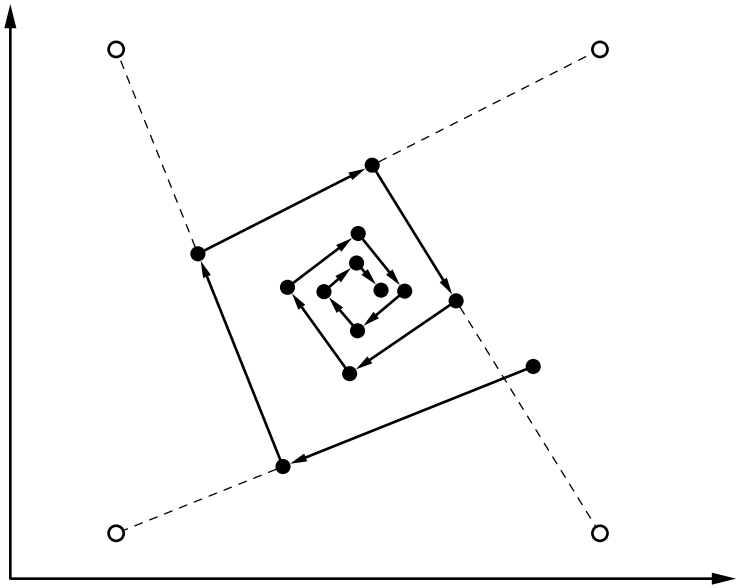
Anpassung der Referenzvektoren



- Links: Online-Training mit Lernrate $\eta = 0.1$,
- Rechts: Batch-Training mit Lernrate $\eta = 0.05$.

Problem: feste Lernrate kann zu Oszillationen führen





Lösung: **zeitabhängige Lernrate**

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

Verbesserte Anpassungsregel für klassifizierte Daten

- **Idee:** Passe nicht nur den Referenzvektor an, der am nächsten zum Datenpunkt liegt (das Gewinnerneuron), sondern passe **die zwei nächstliegenden Referenzvektoren**.

- Sei $\boldsymbol{x}$ der momentan bearbeitete Datenpunkt und c seine Klasse.
  Seien $\boldsymbol{r}_j$ und $\boldsymbol{r}_k$ die zwei nächstliegenden Referenzvektoren und z_j sowie z_k ihre Klassen.

- Referenzvektoren werden nur angepasst, wenn $z_j \neq z_k$ und entweder $c = z_j$ oder $c = z_k$. (o.B.d.A. nehmen wir an: $c = z_j$.)

  Die **Anpassungsregeln** für die zwei nächstgelegenen Referenzvektoren sind:

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

  wobei alle anderen Referenzvektoren unverändert bleiben.

Lernende Vektorquantisierung: "Window Rule"

- In praktischen Experimenten wurde beobachtet, dass LVQ in der Standardausführung die Referenzvektoren immer weiter voneinander wegtreibt.
- Um diesem Verhalten entgegenzuwirken, wurde die **window rule** eingeführt: passe nur dann an, wenn der Datenpunkt $\boldsymbol{x}$ in der Nähe der Klassifikationsgrenze liegt.
- "In der Nähe der Grenze" wird formalisiert durch folgende Bedingung:

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

  ξ ist ein Parameter, der vom Benutzer eingestellt werden muss.
- Intuitiv beschreibt ξ die "Größe" des Fensters um die Klassifikationsgrenze, in dem der Datenpunkt liegen muss, um zu einer Anpassung zu führen.
- Damit wird die Divergenz vermieden, da die Anpassung eines Referenzvektors nicht mehr durchgeführt wird, wenn die Klassifikationsgrenze weit genug weg ist.

Soft LVQ

Idee: Benutze weiche Zuordnungen anstelle von *winner-takes-all*.

Annahme: Die Daten wurden aus einer Mischung von Normalverteilungen gezogen. Jeder Referenzvektor beschreibt eine Normalverteilung.

Ziel: Maximiere das Log-Wahrscheinlichkeitsverhältnis der Daten, also

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

Hierbei ist σ ein Parameter, der die "Größe" jeder Normalverteilung angibt.
$R(c)$ ist die Menge der Referenzvektoren der Klasse c und $Q(c)$ deren Komplement.

Intuitiv: für jeden Datenpunkt sollte die Wahrscheinlichkeitsdichte für seine Klasse so groß wie möglich sein, während die Dichte für alle anderen Klassen so klein wie möglich sein sollte.

Anpassungsregel abgeleitet aus Maximum-Log-Likelihood-Ansatz:

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

wobei z_i die dem Referenzvektor $\boldsymbol{r}_i$ zugehörige Klasse ist und

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

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

$R(c)$ ist die Menge der Referenzvektoren, die zu Klasse c gehören und $Q(c)$ ist deren Komplement.

Hard LVQ

Idee: Leite ein Schema mit scharfen Zuordnungen aus der unscharfen Version ab.

Ansatz: Lasse den Größenparameter σ der Gaußfunktion gegen Null streben.

Die sich ergebende Anpassungsregel ist somit:

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

wobei

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

$\boldsymbol{r}_i$ ist der nächstgelegene Vektor derselben Klasse

$\boldsymbol{r}_i$ ist der nächstgelegene Vektor einer anderen Klasse

Diese Anpassungsregel ist stabil, ohne dass eine *window rule* die Anpassung beschränken müsste.

Lernende Vektorquantisierung: Erweiterungen

- **Frequency Sensitive Competitive Learning**
  - Der Abstand zu einem Referenzvektor wird modifiziert, indem berücksichtigt wird, wieviele Datenpunkte diesem Referenzvektor zugewiesen sind.
- **Fuzzy LVQ**
  - Nutzt die enge Verwandschaft zum Fuzzy-Clustering aus.
  - Kann als Online-Version des Fuzzy-Clustering angesehen werden.
  - Führt zu schnellerem Clustering.
- **Größen- und Formparameter**
  - Weise jedem Referenzvektor einen Clusterradius zu.
    Passe diesen Radius in Abhängigkeit von der Nähe der Datenpunkte an.
  - Weise jedem Referenzvektor eine Kovarianzmatrix zu.
    Passe diese Matrix abhängig von der Verteilung der Datenpunkte an.