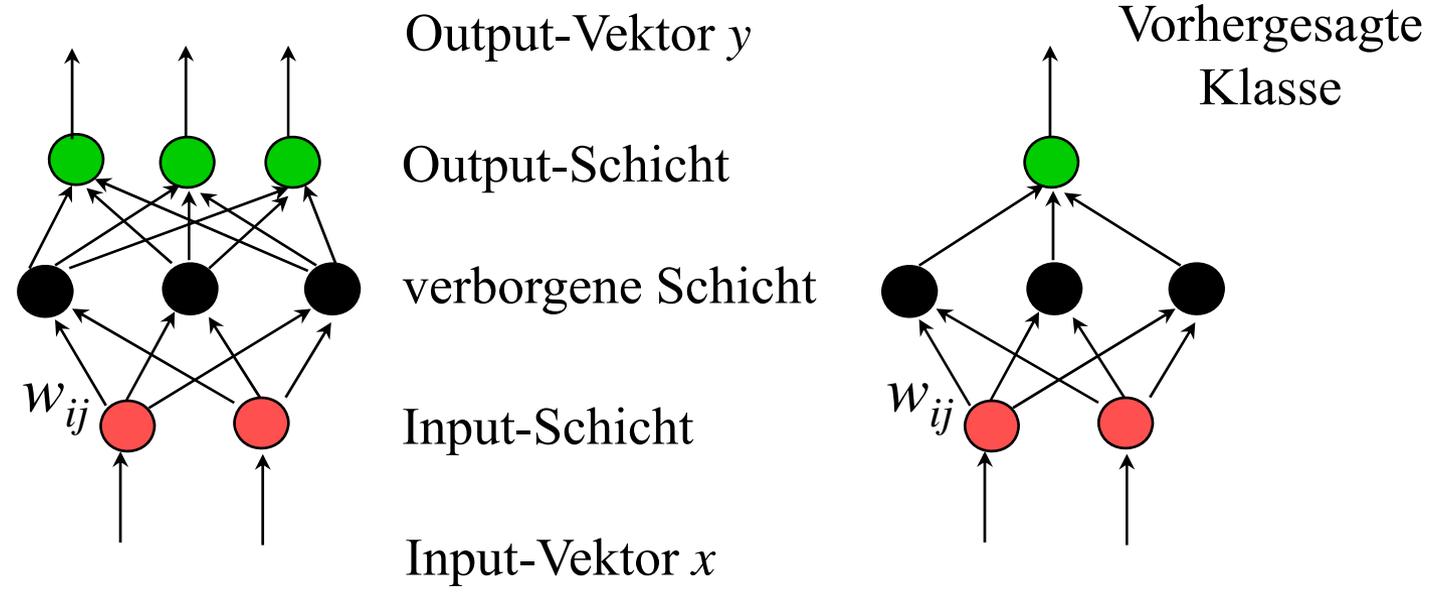


Grundlagen [Bigus 1996], [Bishop 1995]

- Paradigma für ein Maschinen- und Berechnungsmodell
- Funktionsweise ähnlich der von biologischen Gehirnen
 - Neuronales Netz*: Menge von Neuronen, über Kanten miteinander verbunden
 - Neuron*: entspricht biologischem Neuron: Aktivierung durch Input-Signale an den Synapsen
- Menschliches Gehirn:
 - 10^{11} Neuronen, jedes verbunden mit $\sim 10^4$ anderen
 - schnellste Aktionszeit 10^{-3} Sek. (3GHz Prozessor: $\sim 10^{-10}$)
 - komplexe Entscheidungen sehr schnell (visuelle Erkennung der eigenen Mutter: 10^{-1} Sek. ≈ 100 Schritte)
- Erzeugung eines Output-Signals, das zu anderen Neuronen weitergeleitet wird
- Organisation eines neuronalen Netzes:
 - *Input-Schicht*
 - *verborgene Schichten*
 - *Output-Schicht*
 - Knoten einer Schicht mit allen Knoten der vorhergehenden Schicht verbunden



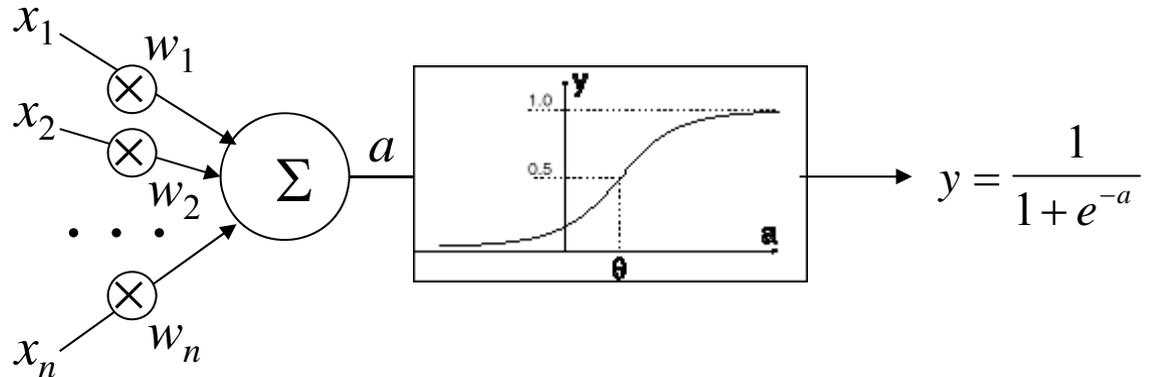
- Netzwerk: Knoten und (gerichtete) Kanten
- Kanten besitzen Gewichte
- Funktion eines neuronalen Netzes:



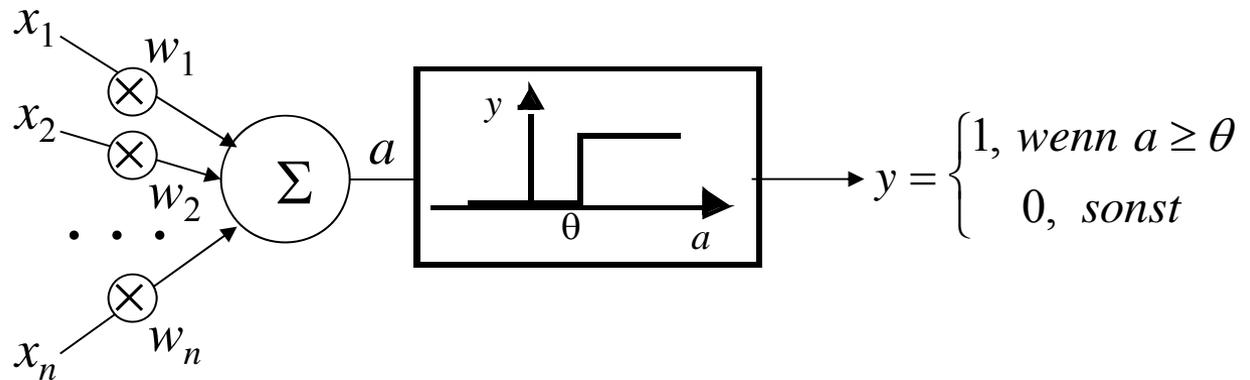
allgemeines Neuron (auch: Perzeptron)

a : Aktivierungswert

$$a = \sum_{i=1}^n w_i \cdot x_i$$



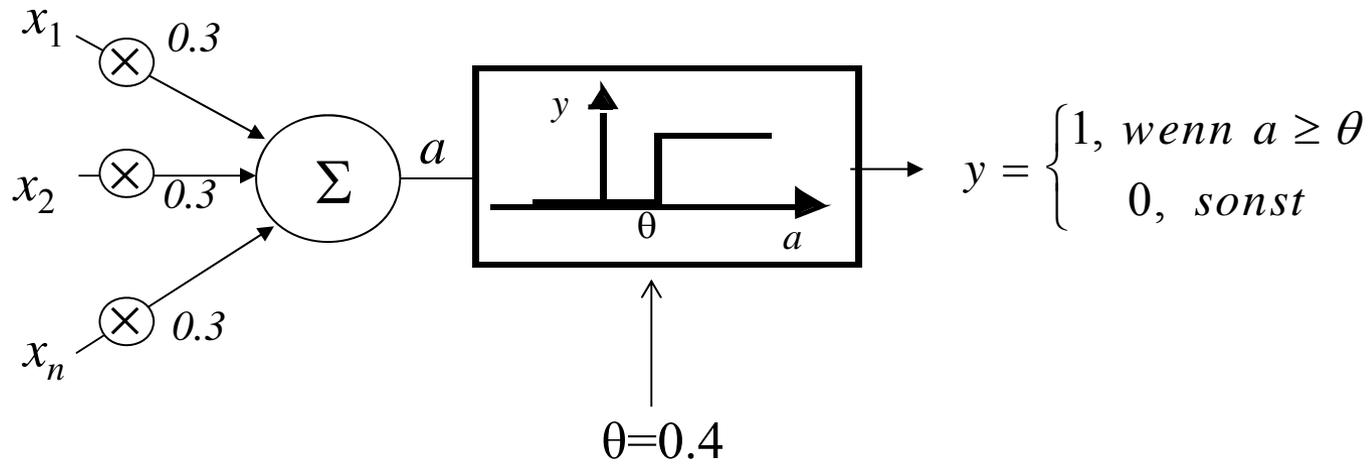
Threshold Logic Unit (TLU)



Beispiel: Modellierung einer booleschen Funktion

(zwei oder mehr Variablen = 1 \Rightarrow y=1)

x_1	x_2	x_3	y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0



\Rightarrow Lernen der Gewichte w_i

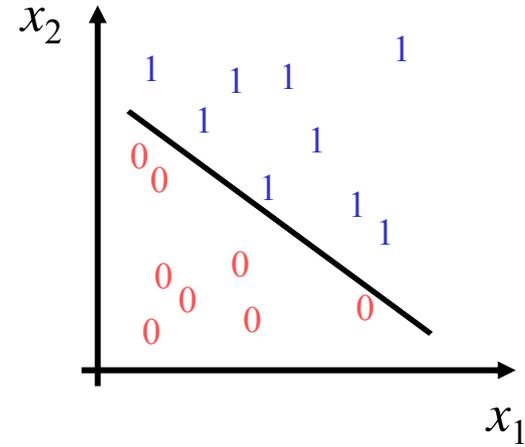
- Klassifikation mit Hilfe einer TLU

- repräsentiert eine (Hyper-)Ebene

$$\sum_{i=1}^n w_i \cdot x_i = \theta$$

- links von der Ebene: Klasse 0

- rechts von der Ebene: Klasse 1



- Trainieren einer TLU

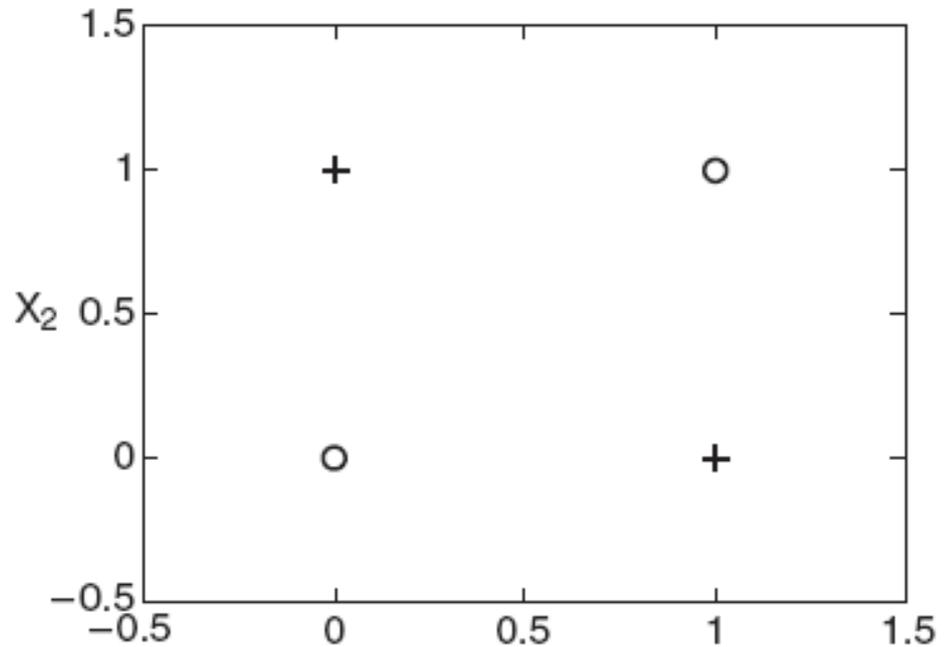
- Lernen der „richtigen“ Gewichte zur Unterscheidung der zwei Klassen: iterative Anpassung der Gewichte w_{ij}

- Rotation der durch w und θ gegebene Hyperebene um einen kleinen Betrag in Richtung v , wenn v noch nicht auf der richtigen Seite der Ebene liegt

XOR-Problem

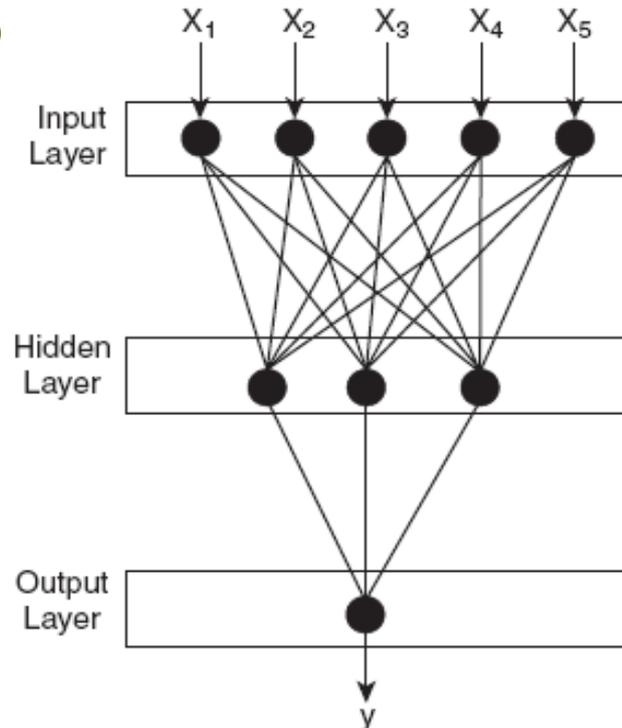
nicht linear separierbares Problem

X_1	X_2	y
0	0	-1
1	0	1
0	1	1
1	1	-1



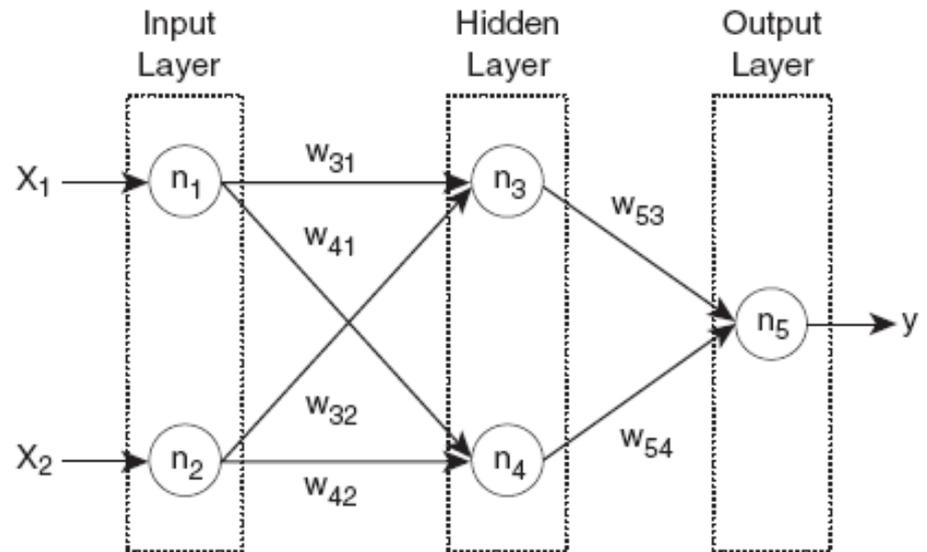
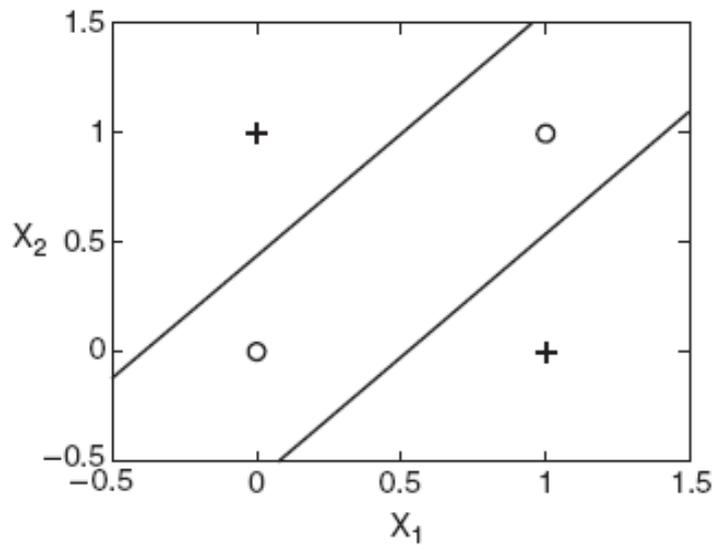
zwei Klassen, die nicht linear separierbar sind:

⇒ mehrere innere Knoten (hidden layer) und ein Output-Knoten (zwei-Klassen-Problem, sonst mehr)

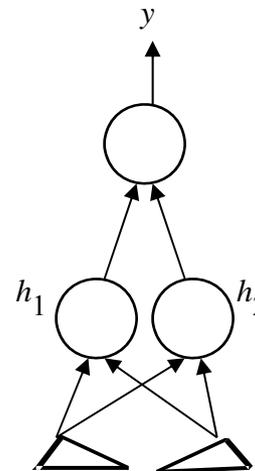
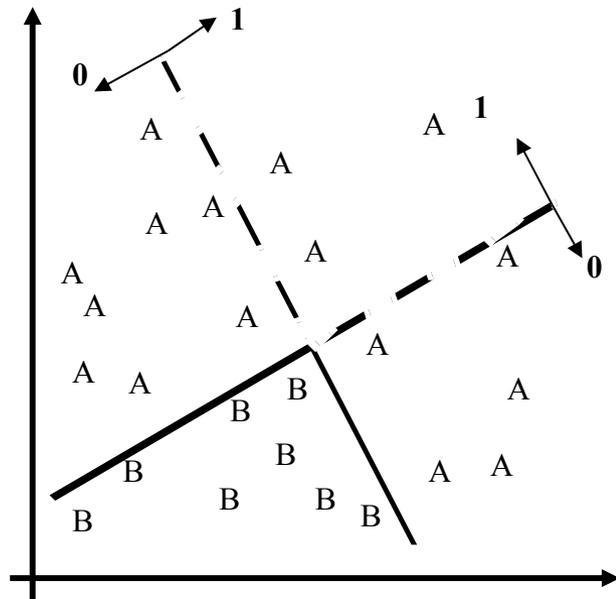


Varianten: feed-forward vs. recurrent (auch Verbindungen auf derselben Ebene oder zu vorherigen Ebenen möglich)

XOR



Beispiel



$h_1 = 0 \wedge h_2 = 0: y = 0$ (Klasse B)

andernfalls: $y = 1$ (Klasse A)

Bei Abweichung von vorhergesagter und tatsächlicher Klasse:

Anpassung der Gewichte mehrerer Knoten

Frage

In welchem Maße sind die verschiedenen Knoten an dem Fehler beteiligt?

Anpassung der Gewichte

- durch Gradientenverfahren, das den Gesamtfehler minimiert
- *Gesamtfehler*: Summe der (quadratischen) Abweichungen des tatsächlichen Outputs y vom gewünschten Output t für die Menge der Inputvektoren (Least Squares Optimierung)
- *Voraussetzung*: Output y *stetige* Funktion der Aktivierung a

für jedes Paar(v, t) // v = Input, t = gewünschter Output

„*forward pass*“:

Bestimme den tatsächlichen Output y für Eingabe v ;

„*backpropagation*“:

Bestimme den Fehler ($t - y$) der Output-Einheiten
und passe die Gewichte der Output-Einheiten in die
Richtung an, die den Fehler minimiert;

Solange der Input-Layer nicht erreicht ist:

Propagiere den Fehler auf die nächste Schicht und
passe auch dort die Gewichte der Einheiten in
fehlerminimierender Weise an;

Bestimmung von

- Anzahl der Input-Knoten
- Anzahl der inneren Schichten und jeweilige Anzahl der Knoten
- Anzahl der Output-Knoten

starker Einfluss auf die Klassifikationsgüte:

- zu wenige Knoten
⇒ niedrige Klassifikationsgüte
- zu viele Knoten
⇒ Overfitting

nach [SPSS Clementine 2000]

Statische Topologie

- Topologie wird a priori festgelegt
- eine verborgene Schicht reicht in vielen Anwendungen aus

Dynamische Topologie

- dynamisches Hinzufügen von Neuronen (und verborgenen Schichten) solange Klassifikationsgüte signifikant verbessert wird

Multiple Topologien

- Trainieren mehrerer dynamischer Netze parallel
 - z.B. je ein Netz mit 1, 2 und 3 verborgenen Schichten

Pruning

- Trainieren eines Netzes mit statischer Topologie
- nachträgliches Entfernen der unwichtigsten Neuronen solange Klassifikationsgüte verbessert wird

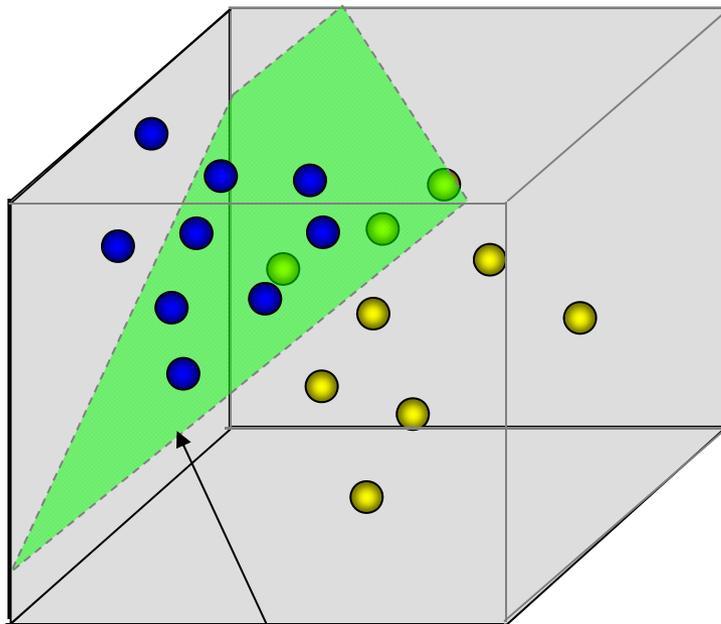
Schlussfolgerung

- statische Topologie: niedrige Klassifikationsgüte, aber relativ schnell.
- Pruning: beste Klassifikationsgüte, aber sehr hoher Laufzeitaufwand zum Training.

- + im Allgemeinen sehr hohe Klassifikationsgüte
beliebig komplexe Entscheidungsflächen (möglichst einfache Topologie
auswählen, um Overfitting zu vermeiden)
- + redundante Features relativ problemlos, Gewichte werden klein
- + Effizienz der Anwendung

- schlechte Verständlichkeit
(lernt nur Gewichte, aber keine Klassenbeschreibung)
- Ineffizienz des Lernens (sehr lange Trainingszeiten)
- keine Integration von Hintergrundwissen
- empfindlich gegen Fehler in den Trainingsdaten
- mögliche Konvergenz in lokales Minimum

Motivation: Lineare Separation



trennende Hyperebene

- Vektoren in R^d repräsentieren Objekte.
- Objekte gehören zu genau einer von je 2 Klassen

Klassifikation durch lineare Separation:

- Suche Hyperebene, die beide Klassen „maximal stabil“ voneinander trennt.
- Ordne unbekannte Elemente der Seite der Ebene zu, auf der sie sich befinden.

Probleme bei linearer Separation:

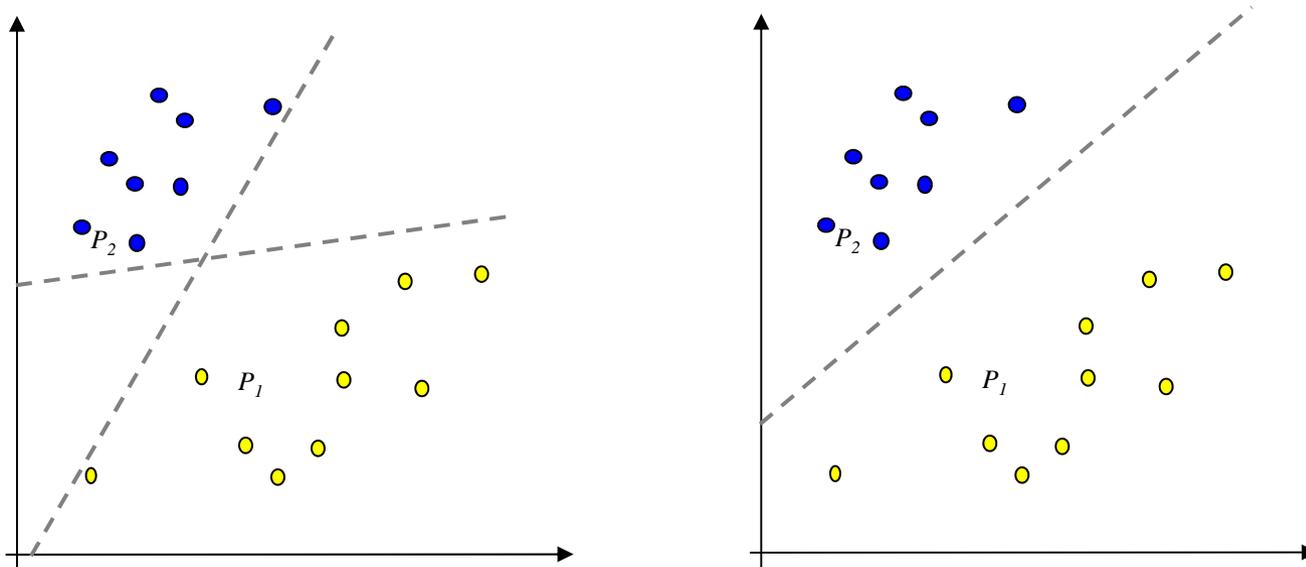
- Was ist die „maximal stabile“ Hyperebene und wie berechnet man sie effizient?
- Klassen nicht immer linear trennbar.
- Berechnung von Hyperebenen nach Auswahl sehr aufwendig.
- Einschränkung auf 2 Klassen.
- ...



Lösungen dieser Probleme

mit Support Vector Machines(SVMs) [Vapnik 1979 u. 1995].

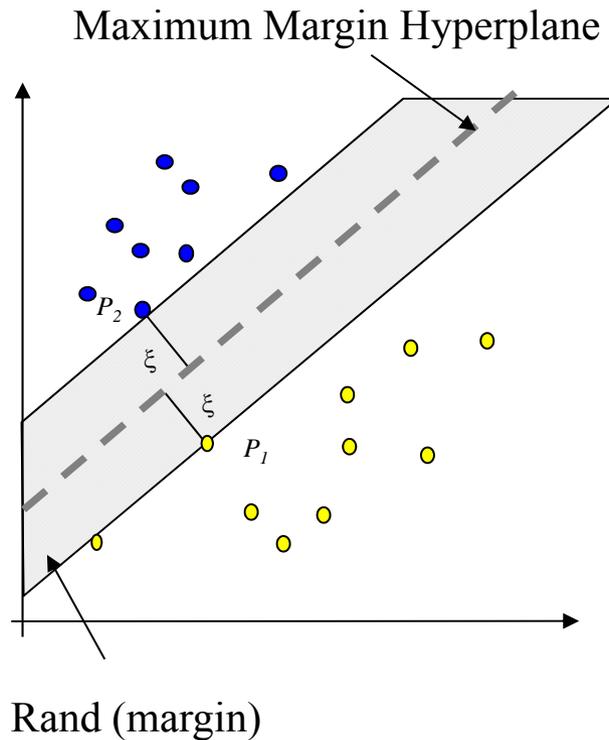
Problem: Hyperebene, die P_1 und P_2 trennt, ist nicht eindeutig.
 \Rightarrow Welche Hyperebene ist für die Separation die Beste ?



Kriterien:

- Stabilität beim Einfügen
- Abstand zu den Objekten beider Klassen

Lineare Separation mit der „Maximum Margin Hyperplane“



- Abstand zu Punkten aus beiden Mengen ist maximal, d.h. mind. ξ
- Wahrscheinlichkeit, dass beim Einfügen die trennende Hyperebene verschoben werden muss, ist minimal
- generalisiert am besten

⇒ Maximum Margin Hyperplane (MMH) ist „maximal stabil“

MMH ist nur von Punkten P_i abhängig, die Abstand ξ zur Ebene aufweisen.

⇒ P_i heißt *Support Vector*

Zusammenfassung der Schreibweisen der benötigten algebraischen Konstrukte für Featurespace FS :

Skalarprodukt zweier Vektoren: $\langle \vec{x}, \vec{y} \rangle, \vec{x}, \vec{y} \in FS$

z.B. $\langle \vec{x}, \vec{y} \rangle = \sum_{i=1}^d (x_i \cdot y_i)$ kanonisches Skalarprodukt

Beschreibung einer Hyperebene: $H(\vec{w}, b) = \left\{ \vec{x} \in FS \mid 0 = \langle \vec{w}, \vec{x} \rangle + b \right\}$

Abstand eines Vektors zur Ebene: $dist(\vec{x}, H(\vec{w}, b)) = \left| \frac{1}{\sqrt{\langle \vec{w}, \vec{w} \rangle}} \langle \vec{w}, \vec{x} \rangle + b \right|$

Berechnung der Maximum Margin Hyperplane

1. Bedingung: kein Klassifikationsfehler (Klasse 1: $y_i=1$, Klasse 2: $y_i=-1$)

$$\left. \begin{array}{l} (y_i = -1) \Rightarrow [\langle \vec{w}, \vec{x}_i \rangle + b] < 0 \\ (y_i = 1) \Rightarrow [\langle \vec{w}, \vec{x}_i \rangle + b] > 0 \end{array} \right\} \Leftrightarrow y_i [\langle \vec{w}, \vec{x}_i \rangle + b] > 0$$

2. Bedingung: Maximaler Rand (Margin)

maximiere: $\xi = \min_{x_i \in TR} \left| \frac{1}{\sqrt{\langle \vec{w}, \vec{w} \rangle}} (\langle \vec{w}, \vec{x}_i \rangle + b) \right|$ (Abstand von x_i zur Ebene $H(\vec{w}, b)$)

oder

maximiere: ξ , so dass $\left[y_i \left(\frac{1}{\sqrt{\langle \vec{w}, \vec{w} \rangle}} [\langle \vec{w}, \vec{x}_i \rangle + b] \right) \geq \xi \right]$ für $\forall i \in [1..n]$

maximiere ξ in $\left[y_i \left(\frac{1}{\sqrt{\langle \vec{w}, \vec{w} \rangle}} \left[\langle \vec{w}, \vec{x}_i \rangle + b \right] \right) \geq \xi \right]; \text{ für } \forall i \in [1..n]$

Setze $\frac{1}{\sqrt{\langle \vec{w}, \vec{w} \rangle}} = \xi : \max. \frac{1}{\sqrt{\langle \vec{w}, \vec{w} \rangle}}$, mit $(y_i \cdot (\xi \cdot (\langle \vec{w}, \vec{x}_i \rangle + b))) \geq \xi \quad \forall i \in [1..n]$

$\Rightarrow \max. \frac{1}{\sqrt{\langle \vec{w}, \vec{w} \rangle}}$, mit $(y_i (\langle \vec{w}, \vec{x}_i \rangle + b) \geq 1) \quad \forall i \in [1..n]$

Statt $\frac{1}{\sqrt{\langle \vec{w}, \vec{w} \rangle}}$ invertiere, quadriere und minimiere das Ergebnis:

Primäres OP: minimiere $J(\vec{w}, b) = \langle \vec{w}, \vec{w} \rangle$

unter Nebenbedingung für $\forall i \in [1..n]$ sei $(y_i (\langle \vec{w}, \vec{x}_i \rangle + b) \geq 1)$

Zur Berechnung wird das primäre Optimierungsproblem in ein duales OP überführt.

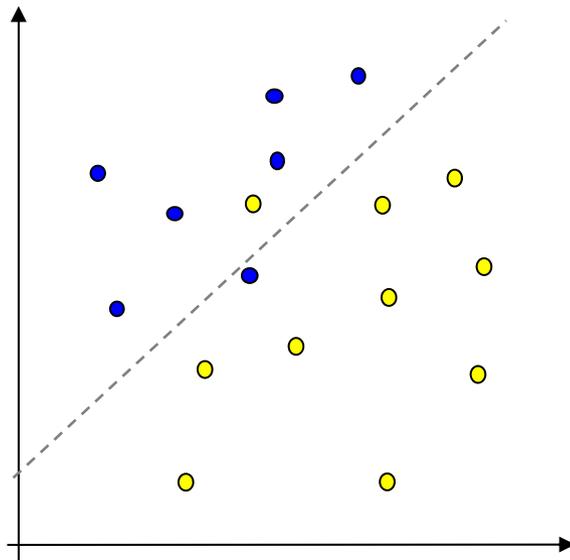
(Umformulierung in Form mit Lagrange Multiplikatoren nach Karush-Kuhn-Tucker)

$$\text{Duales OP: maximiere } L(\vec{\alpha}) = \left(\sum_{i=1}^n \alpha_i \right) - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot \langle \vec{x}_i, \vec{x}_j \rangle$$

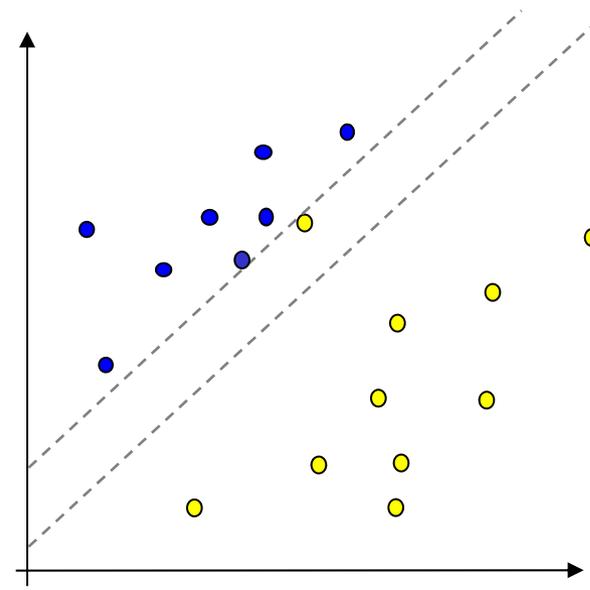
$$\text{unter Bedingung } \sum_{i=1}^n \alpha_i \cdot y_i = 0, 0 \leq \alpha_i \text{ und } \vec{\alpha} \in \mathbb{R}^n$$

- ⇒ Lösung des Problems mit Algorithmen aus der Optimierungstheorie
- ⇒ bis jetzt nur linear separierbarer Fall: Soft Margin Optimierung
- ⇒ Einführung von Kernelfunktionen zur Steigerung der Kapazität

Behandlung nicht linear trennbarer Daten: *Soft Margin Optimierung*



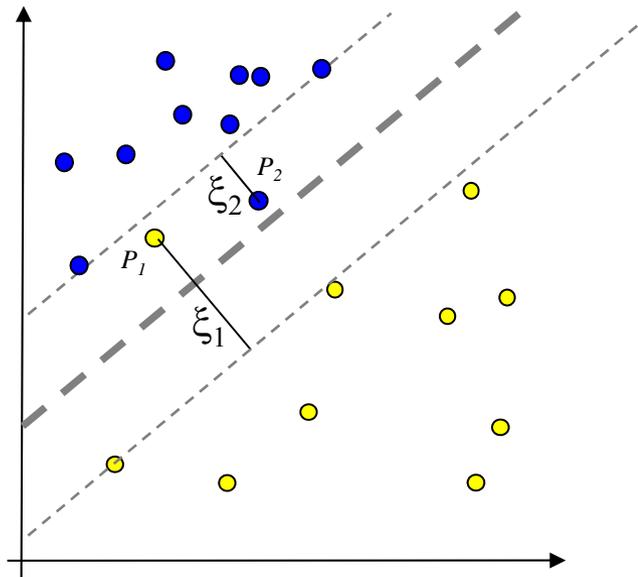
Daten nicht separierbar



vollständige Separation ist nicht optimal

⇒ Trade-Off zwischen Trainingsfehler und Breite des Randes

Betrachte beim Optimieren zusätzlich noch die Anzahl der Trainingsfehler.



- ξ_i ist der Abstand von P_i zum Rand (wird auch Slack-Variable genannt)
- C reguliert den Einfluss eines einzelnen Trainingsvektors

Primäres OP :

$$\text{minimiere } J(\vec{w}, b, \vec{\xi}) = \frac{1}{2} \langle \vec{w}, \vec{w} \rangle + C \cdot \sum_{i=1}^n \xi_i$$

$$\text{unter Nebenbedingung für } \forall i \in [1..n] \text{ sei } y_i (\langle \vec{w}, \vec{x}_i \rangle + b) \geq 1 - \xi_i \quad \text{und } \xi_i \geq 0$$

⇒ Primäres Optimierungsproblem unter weichen Grenzen (Soft Margin)

Das duale OP mit Lagrange Multiplikatoren verändert sich wie folgt:

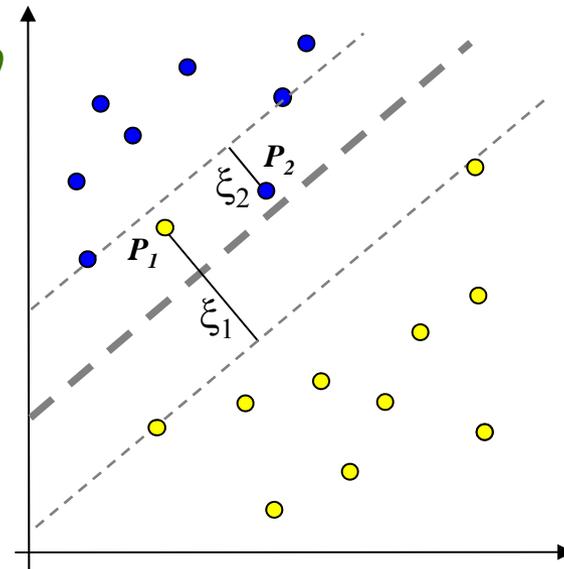
Duales OP: maximiere $L(\vec{\alpha}) = \left(\sum_{i=1}^n \alpha_i \right) - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot \langle \vec{x}_i, \vec{x}_j \rangle$

mit Bedingung $\sum_{i=1}^n \alpha_i \cdot y_i = 0$ und $0 \leq \alpha_i \leq C$

- $0 < \alpha_i < C \Leftrightarrow$ Support Vektor mit $\xi_i = 0$
- $\alpha_i = C \Leftrightarrow$ Support Vektor mit $\xi_i > 0$
- $\alpha_i = 0$ sonst

Entscheidungsregel:

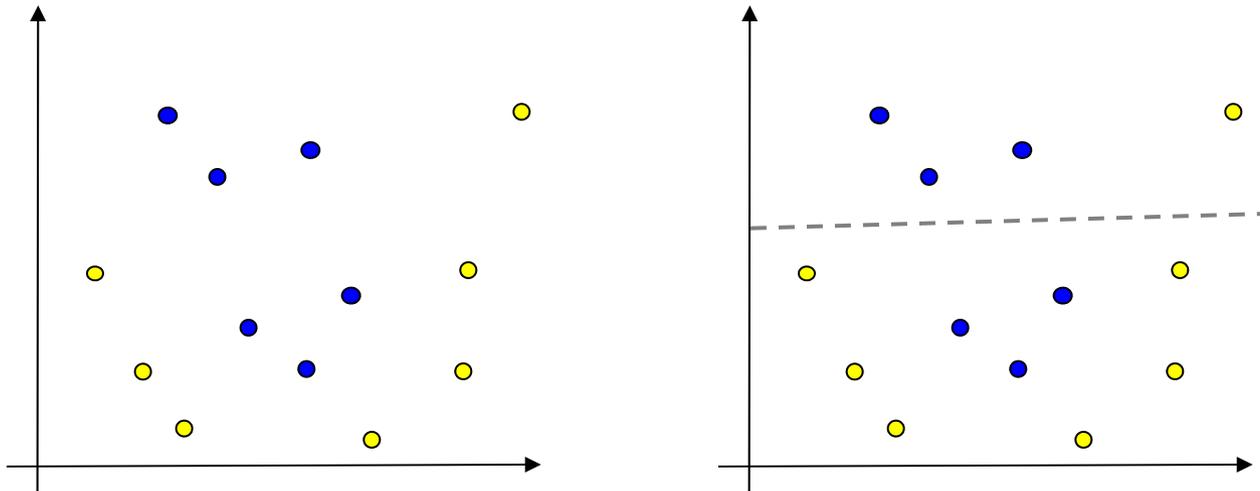
$$h(\vec{x}) = \text{sign} \left(\sum_{x_i \in SV} \alpha_i \cdot y_i \langle \vec{x}_i, \vec{x} \rangle + b \right)$$



Lernen bei nicht linear trennbaren Datenmengen

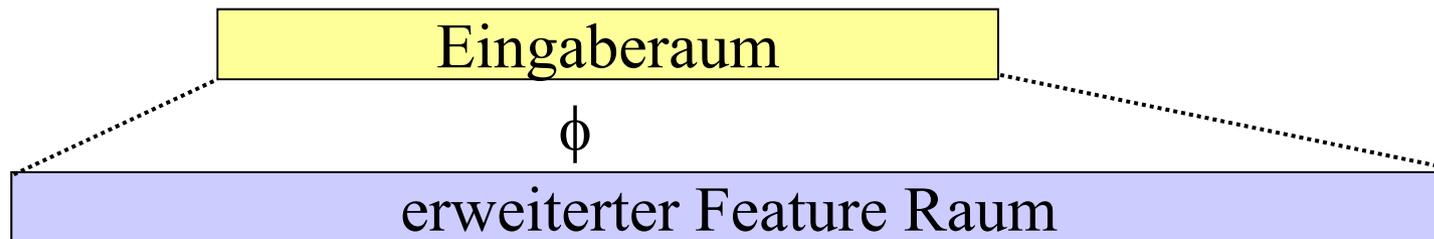
Problem: Bei realen Problemen ist häufig keine lineare Separation mit hoher Klassifikationsgenauigkeit mehr möglich.

Idee: Transformiere Daten in einen nicht linearen Feature-Raum und versuche sie im neuen Raum linear zu separieren.
(Erweiterung des Hypothesenraumes)



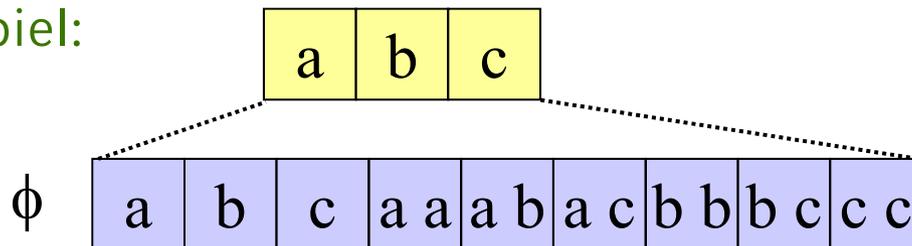
Beispiel: quadratische Transformation

Erweiterung der Hypothesenraumes



⇒ Versuche jetzt in erweitertem Feature Raum linear zu separieren

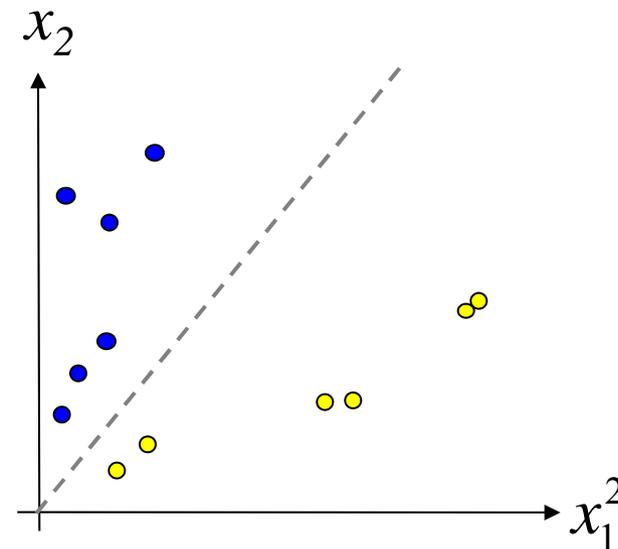
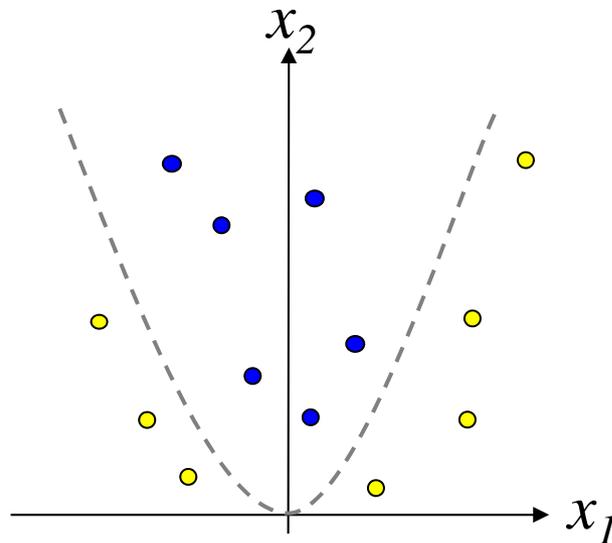
Beispiel:



hier: Eine Hyperebene im erweiterten Feature Raum ist ein Polynom 2. Grades im Eingaberaum.

Eingaberaum: $\vec{x} = (x_1, x_2)$ (2 Attribute)

im erweiterten Raum: $\phi(\vec{x}) = (x_1^2, x_2^2, \sqrt{2} \cdot x_1, \sqrt{2} \cdot x_2, \sqrt{2} \cdot x_1 \cdot x_2, 1)$
(6 Attribute)



Einführung eines Kernels \Leftrightarrow (implizite) Featuretransformation
mittels $\phi(\vec{x}): FS_{alt} \longrightarrow FS_{neu}$

Duales OP: maximiere

$$L(\vec{\alpha}) = \left(\sum_{i=1}^n \alpha_i \right) - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot \langle \phi(\vec{x}_i), \phi(\vec{x}_j) \rangle$$

mit Bedingung $\sum_{i=1}^n \alpha_i \cdot y_i = 0$ und $0 \leq \alpha_i \leq C$

Zusätzliche Featuretransformation wirkt sich nur auf das Skalarprodukt der Trainingsvektoren aus. \Rightarrow Kernel K ist eine Funktion mit:

$$K_{\phi}(\vec{x}_i, \vec{x}_j) = \langle \phi(\vec{x}_i), \phi(\vec{x}_j) \rangle$$

Wann ist eine Funktion $K(x,y)$ ein Kernel ?

Wenn die **Kernel-Matrix** (Gram Matrix) KM

$$KM(K) = \begin{pmatrix} K(\vec{x}_1, \vec{x}_1) & .. & K(\vec{x}_1, \vec{x}_n) \\ .. & .. & .. \\ K(\vec{x}_n, \vec{x}_1) & .. & K(\vec{x}_n, \vec{x}_n) \end{pmatrix}$$

positiv (semi-) definit ist, also keine negativen Eigenwerte besitzt, dann ist $K(x,y)$ ein Kernel (Mercer's Theorem)

Notwendige Bedingungen:

- $K_\phi(\vec{x}, \vec{y}) = \langle \phi(\vec{x}), \phi(\vec{y}) \rangle = \langle \phi(\vec{y}), \phi(\vec{x}) \rangle = K_\phi(\vec{y}, \vec{x})$ (Symmetrie)
- $K_\phi(\vec{x}, \vec{y})^2 \leq K_\phi(\vec{x}, \vec{x}) \cdot K_\phi(\vec{y}, \vec{y})$ (Cauchy-Schwarz)

Symmetrie und Cauchy-Schwarz sind keine hinreichenden Bedingungen!

einige Regeln zur Kombination vom Kernen:

$$K(\vec{x}, \vec{y}) = K_1(\vec{x}, \vec{y}) \cdot K_2(\vec{x}, \vec{y})$$

$$K(\vec{x}, \vec{y}) = K_1(\vec{x}, \vec{y}) + K_2(\vec{x}, \vec{y})$$

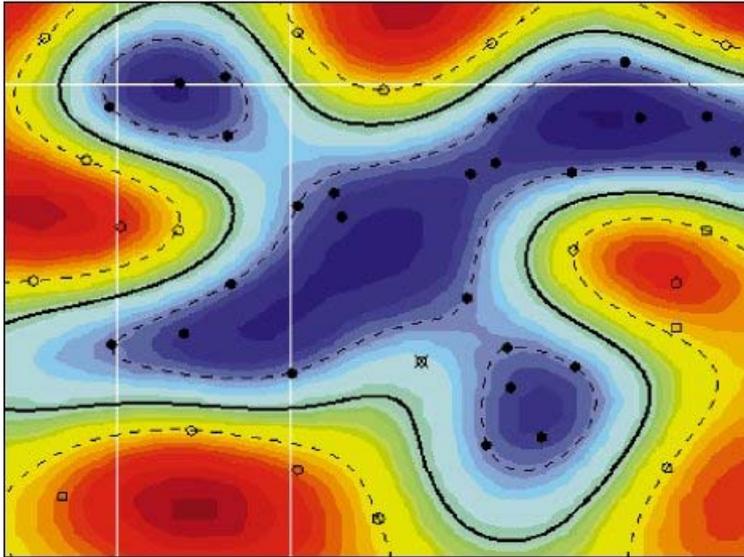
$$K(\vec{x}, \vec{y}) = a \cdot K_1(\vec{x}, \vec{y})$$

$$K(\vec{x}, \vec{y}) = \vec{x}^T \cdot \mathbf{B} \cdot \vec{y}$$

für K_1, K_2 Kernelfunktionen, a eine positive Konstante und \mathbf{B} eine symmetrische positiv semi-definite Matrix.

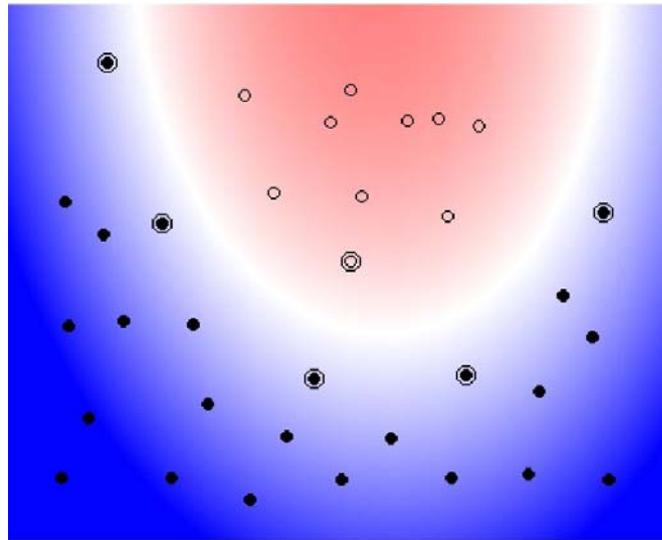
Beispiele für verwendete Kernel-Funktionen:

- linear: $K(\vec{x}, \vec{y}) = \langle \vec{x}, \vec{y} \rangle$
- polynomiell: $K(\vec{x}, \vec{y}) = \left(\langle \vec{x}, \vec{y} \rangle + c \right)^d$
- Radiale Basisfunktionen: $K(\vec{x}, \vec{y}) = \exp\left(-\gamma \cdot \left| \vec{x} - \vec{y} \right|^2 \right)$
- Gauss Kernel: $K(\vec{x}, \vec{y}) = \exp\left(-\frac{\left\| \vec{x} - \vec{y} \right\|^2}{2\sigma^2} \right)$
- Sigmoid: $K(\vec{x}, \vec{y}) = \tanh\left(\gamma \cdot \langle \vec{x}, \vec{y} \rangle + c \right)$



Radial Basis Kernel

Polynomieller Kernel (Grad 2)



zu lösen ist folgendes Problem:

Duales OP: maximiere

$$L(\vec{\alpha}) = \left(\sum_{i=1}^n \alpha_i \right) - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot K(\vec{x}_i, \vec{x}_j)$$

mit Bedingung $\sum_{i=1}^n \alpha_i \cdot y_i = 0$ und $0 \leq \alpha_i \leq C$

oder

$$\max \begin{bmatrix} 1 \\ \dots \\ 1 \end{bmatrix}^T \cdot \begin{bmatrix} \alpha_1 \\ \dots \\ \alpha_n \end{bmatrix} - \frac{1}{2} \begin{bmatrix} \alpha_1 \\ \dots \\ \alpha_n \end{bmatrix}^T \cdot \begin{bmatrix} y_1 y_1 K(\vec{x}_1, \vec{x}_1) & \dots & y_1 y_n K(\vec{x}_1, \vec{x}_n) \\ \dots & \dots & \dots \\ y_n y_1 K(\vec{x}_n, \vec{x}_1) & \dots & y_n y_n K(\vec{x}_n, \vec{x}_n) \end{bmatrix} \cdot \begin{bmatrix} \alpha_1 \\ \dots \\ \alpha_n \end{bmatrix}$$

mit Bedingung $\sum_{i=1}^n \alpha_i \cdot y_i = 0$ und $0 \leq \alpha_i \leq C$

zur Lösung:

- Standardalgorithmen aus der Optimierungstheorie für *konvexe quadratische Probleme*
- für große Trainingsmengen numerische Algorithmen notwendig
- es existieren einige Spezialalgorithmen für SVM-Training:
 - Chunking / Decomposition
 - Sequential Minimal Optimisation (SMO)

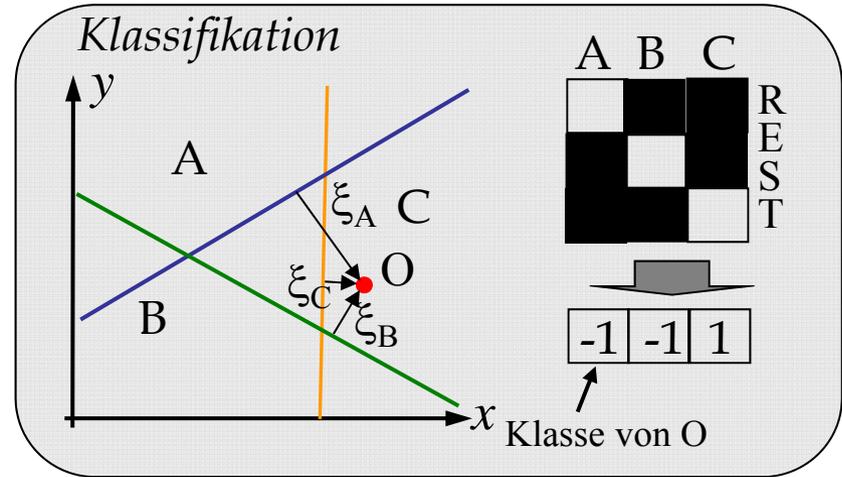
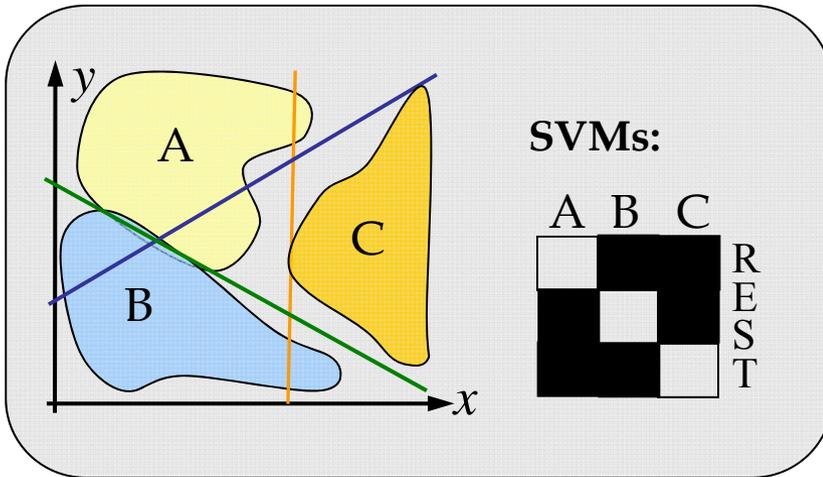
Bisher SVMs nur anwendbar auf 2 Klassen Probleme !!

Idee: Kombination mehrere 2-Klassen SVMs zu Klassifikatoren, die beliebig viele Klassen unterscheiden können.

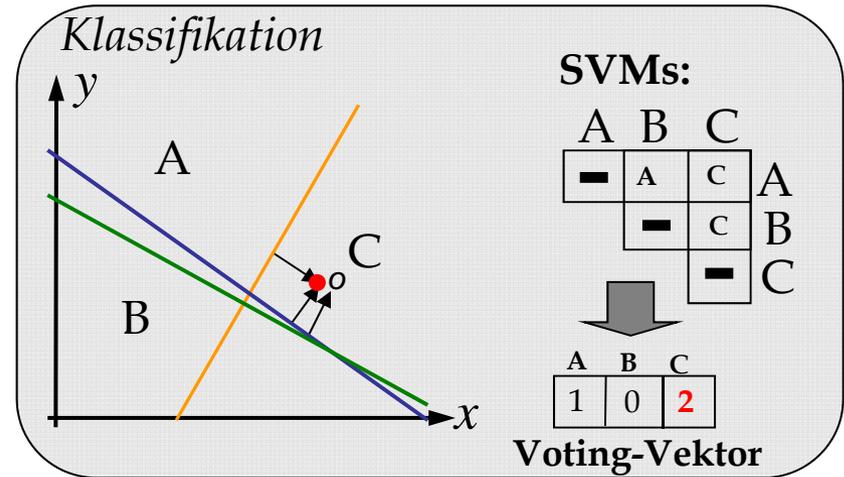
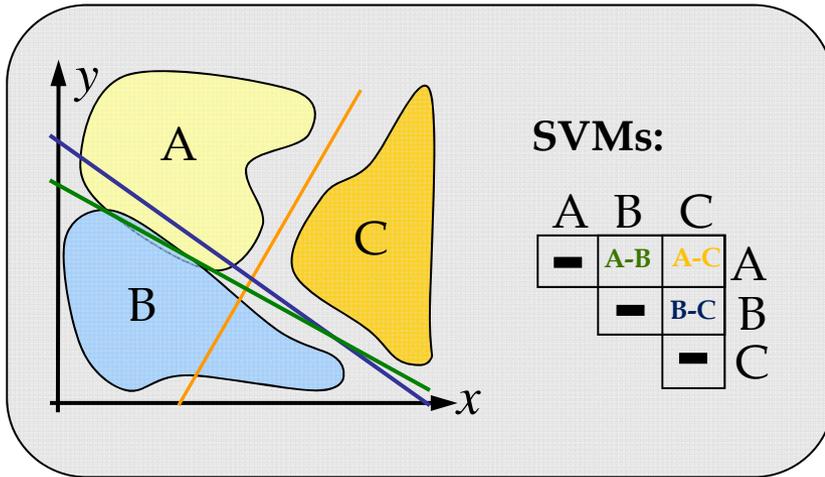
Multi-Class SVMs

2 klassische Ansätze:

- ⇒ Unterscheide jede Klasse von allen anderen
(*1-versus-Rest*)
- ⇒ Unterscheide je 2 Klassen
(*1-versus-1*)



- 1-versus-Rest Ansatz : 1 SVM für jede Klasse.
- SVM trennt jedes Klasse von Vereinigung aller anderen Klassen ab
- Klassifiziere O mit allen Basis-SVMs.
 ⇒ **Multiple Klassenzugehörigkeit möglich (Multi-Classification)**
 oder
Entscheidung für die Klasse, bei der Abstand ξ_i am größten ist.



- 1-versus-1 Ansatz : 1 SVM für jedes Paar von Klassen.
- Klassifikation von Objekt O:
 1. Klassifiziere O mit allen Basis-SVMs.
 2. Zähle "Stimmen" (Votes) für jede Klasse.
- Maximale Anzahl an Votes => Ergebnis.

Vergleich 1-versus-rest und 1-versus-1

Kriterium	1-versus-rest	1-versus-1
Aufwand Training	linear zur Anzahl der Klassen ($O(K)$)	Quadratisch zur anzahl der Klassen ($O(K ^2)$)
Aufwand Klassifikation	linear zur Anzahl der Klassen ($O(K)$)	Quadratisch zur Anzahl der Klassen ($O(K ^2)$) Verbesserung: „Decision Directed Acyclic Graphs“ Klassifikation in $O(K)$ [Platt,Christianini 1999]
Genauigkeit	tendenziell schlechter	tendenziell höher, (nutzt wissen über unterschiedliche Klassen besser aus)

Diskussion

- + erzeugt Klassifikatoren mit hoher Genauigkeit
- + verhältnismäßig schwache Tendenz zu Overfitting
(Begründung durch Generalisierungstheorie)
- + effiziente Klassifikation neuer Objekte
- + kompakte Modelle
- unter Umständen lange Trainingszeiten
- aufwendige Implementierung
- gefundene Modelle schwer zu deuten

Literatur:

- C. Cortes, V. Vapnik: Support-vector networks.
Machine Learning, 20:273-297, November 1995.
- C.J.C. Burges: A tutorial on support vector machines for pattern recognition.
Data Mining and Knowledge Discovery, 2(2):121-167,1998.
- T. Joachims: Text categorisation with Support Vector Machines.
in Proceedings of European Conference on Machine Learning (ECML), 1998.
- N. Cristianini, J. Shawe-Taylor: An Introduction to Support Vector Machines and other kernel-based learning methods.
Cambridge University Press 2000.

Was haben Sie gelernt?

- Unterschied Clustering – Klassifikation
- Modell/Hypothese/Annahmen/Bias
 - „bias-free learning is futile“
 - overfitting
- Bewertung von Klassifikatoren
 - Bewertungsmaße
 - Evaluationsmethoden (Kreuzvalidierung, Bootstrap, leave-one-out)
- einige Modelle zur Klassifikation und ihre Annahmen
 - Bayes Klassifikatoren (insb. Naïve Bayes, LDA)
 - kNN Klassifikation
 - Decision Trees
 - Neuronale Netze
 - Support Vector Machines und Kernel Machines
- lazy-learning vs. eager-learning