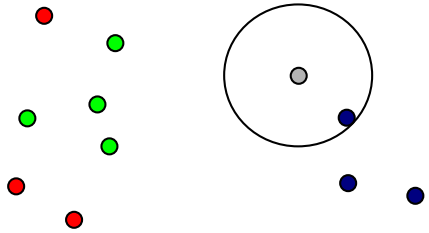
5.4 Nächste-Nachbarn-Klassifikatoren



- Schrauben
- Nägel
- Klammern

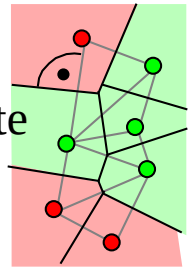
} Trainingsdaten

- Neues Objekt

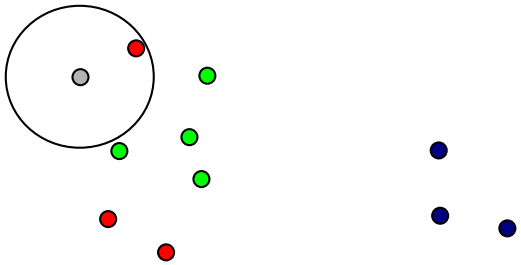
Instanzbasiertes Lernen (*instance based learning*)

- Einfacher Nächster-Nachbar-Klassifikator: Zuordnung zu der Klasse des nächsten Nachbarpunkts
- Im Beispiel: Nächster Nachbar ist eine Schraube
- Regionen der Klassenzuordnung können als *Voronoi-Diagramme* dargestellt werden:

Mittel-senkrechte



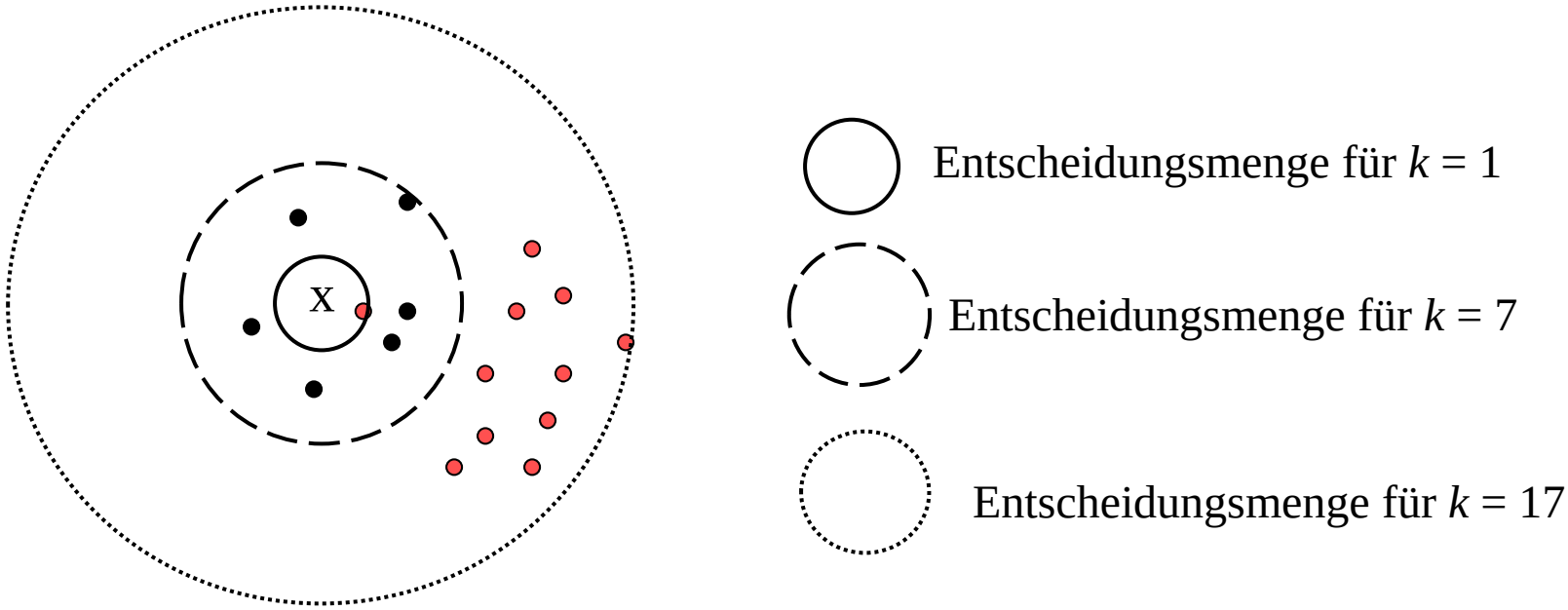
Nächste-Nachbarn-Klassifikatoren



- Problem: Punkt links oben wahrscheinlich nur Ausreißer
- Besser: Betrachte mehr als nur einen Nachbarn
  $\Rightarrow$ *k*-Nächste-Nachbarn-Klassifikator
- *Entscheidungsmenge*
  die Menge der zur Klassifikation betrachteten *k*-nächsten Nachbarn
- *Entscheidungsregel*
  Wie bestimmt man aus den Klassen der Entscheidungsmenge die Klasse des zu klassifizierenden Objekts?
  - Interpretiere Häufigkeit einer Klasse in der Entscheidungsmenge als Wahrscheinlichkeit der Klassenzugehörigkeit
  - Maximum-Likelihood-Prinzip: Mehrheitsentscheidung
  - Ggf. Gewichtung

Wahl des Parameters k

- „zu kleines“ *k*: hohe Sensitivität gegenüber Ausreißern
- „zu großes“ *k*: viele Objekte aus anderen Clustern (Klassen) in der Entscheidungsmenge.
- mittleres *k*: höchste Klassifikationsgüte, oft $1 << k < 10$



x: zu klassifizieren

Entscheidungsregel

- Standardregel

  $\Rightarrow$ wähle die Mehrheitsklasse der Entscheidungsmenge

- Gewichtete Entscheidungsregel

  gewichte die Klassen der Entscheidungsmenge

  - nach Distanz, meist invers quadriert: $weight(dist) = 1/dist^2$
  - nach Verteilung der Klassen (oft sehr ungleich!)

    Problem: Klasse mit zu wenig Instanzen ($< k/2$) in der Trainingsmenge bekommt keine Chance, ausgewählt zu werden, selbst bei optimaler Distanzfunktion

    - Klasse A: 95 %, Klasse B 5 %
    - Entscheidungsmenge = {A, A, A, A, B, B, B}
    - Standardregel $\Rightarrow$ A, gewichtete Regel $\Rightarrow$ B

Klassifikation von Sternen

Analyse astronomischer Daten

Entfernen von Rauschen

Bildsegmentierung

Feature-Extraktion

Automatische *Klassifikation* des Sterntyps

Manuelle Analyse der interessanten Sterntypen

Klassifikation des Sterntyps mit Nächste-Nachbarn Klassifikator basierend auf dem Hipparcos-Katalog

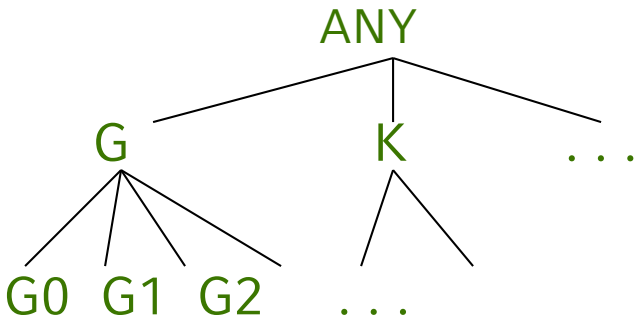
Klassifikation von Sternen

Hipparcos-Katalog [ESA 1998]

- enthält ca. 118 000 Sterne
- mit 78 Attributen (Helligkeit, Entfernung, Farbe,. . .)
- Klassenattribut: Spektraltyp (Attribut H76)

z.B.

H76: G0
H76: G7.2
H76: KIII/IV



- Werte des Spektraltyps sind vage

➡ Hierarchie von Klassen: benutze die erste Ebene der Klassenhierarchie

Klassifikation von Sternen

Verteilung der Klassen

Klasse	#Instanzen	Anteil Instanzen	
K	32 036	27.0	häufige Klassen
F	25 607	21.7	
G	22 701	19.3	
A	18 704	15.8	
B	10 421	8.8	
M	4 862	4.1	
O	265	0.22	seltene Klassen
C	165	0.14	
R	89	0.07	
W	75	0.06	
N	63	0.05	
S	25	0.02	
D	27	0.02	

Klassifikation von Sternen

Experimentelle Untersuchung [Poschenrieder 1998]

- Distanzfunktion
  - mit 6 Attributen (Farbe, Helligkeit und Entfernung)
  - mit 5 Attributen (ohne Entfernung)

  ⇒ beste Klassifikationsgenauigkeit mit 6 Attributen
- Anzahl *k* der Nachbarn

  ⇒ beste Klassifikationsgenauigkeit für $k = 15$
- Entscheidungsregel
  - Gewichtung nach Distanz
  - Gewichtung nach Klassenverteilung

  ⇒ beste Klassifikationsgenauigkeit bei Gewichtung nach Distanz aber nicht nach Klassenverteilung

Klassifikation von Sternen

Klasse	Falsch klassifiziert	Korrekt klassifiziert	Klassifikations-genauigkeit
K	408	2338	85.1%
F	350	2110	85.8%
G	784	1405	64.2%
A	312	975	75.8%
B	308	241	43.9%
M	88	349	79.9%
C	4	5	55.6%
R	5	0	0%
W	4	0	0%
O	9	0	0%
N	4	1	20%
D	3	0	0%
S	1	0	0%
Total	2461	7529	75.3%

hohe Klassifikationsgenauigkeit für die häufigen Klassen, schlechte Genauigkeit für die seltenen Klassen

die meisten seltenen Klassen besitzen weniger als $k / 2 = 8$ Instanzen!

Nächste-Nachbarn-Klassifikatoren

Diskussion

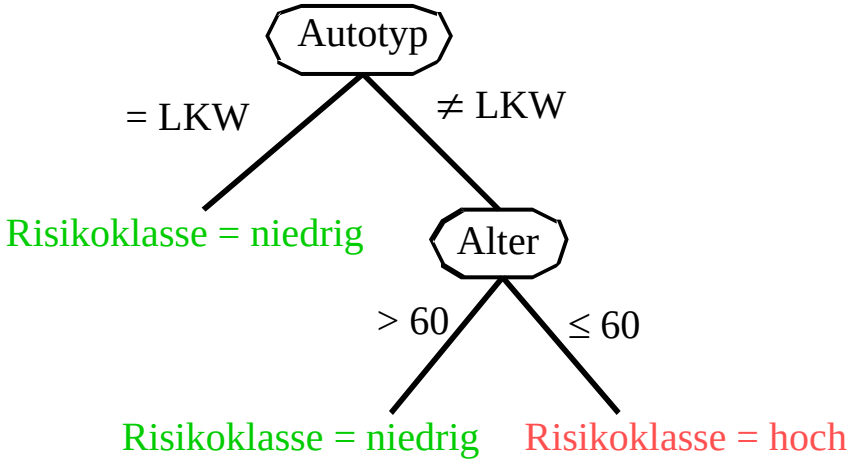
+ Anwendbarkeit: erfordert nur die Trainingsdaten
+ hohe Klassifikationsgenauigkeit in vielen Anwendungen
+ inkrementell: Klassifikator kann sehr einfach an neue Trainingsobjekte adaptiert werden
+ keine Trainingsphase erforderlich („lazy learner“)

- Ineffizienz: Auswertung des “Modells” erfordert *k*-nächste-Nachbarn Anfrage an die Datenbank
- liefert kein explizites Wissen über die Klassen

5.5 Entscheidungsbaum-Klassifikatoren

Motivation

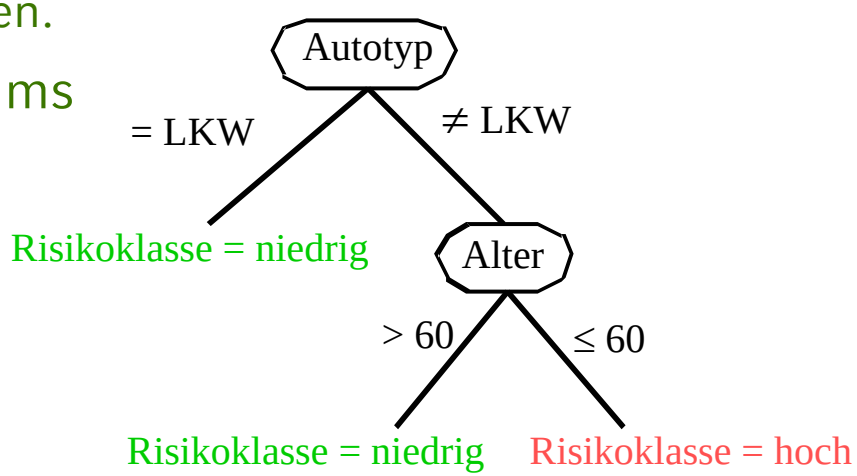
ID	Alter	Autotyp	Risiko
1	23	Familie	hoch
2	17	Sport	hoch
3	43	Sport	hoch
4	68	Familie	niedrig
5	32	LKW	niedrig



- finden explizites Wissen
- Entscheidungsbäume liefern ein für den Benutzer verständliches Modell (Hierarchie von Regeln)

Grundbegriffe

- Ein *Entscheidungsbaum* ist ein Baum mit folgenden Eigenschaften:
  - ein innerer Knoten repräsentiert ein Attribut,
  - eine Kante repräsentiert einen Test auf dem Attribut des Vaterknotens,
  - ein Blatt repräsentiert eine der Klassen.
- Konstruktion eines Entscheidungsbaums
  - anhand der Trainingsmenge
  - Top-Down



- Anwendung eines Entscheidungsbaums

  Durchlauf des Entscheidungsbaum von der Wurzel zu einem der Blätter

  $\Rightarrow$ eindeutiger Pfad

  Zuordnung des Objekts zur Klasse des erreichten Blatts

Konstruktion eines Entscheidungsbaums

Basis-Algorithmus

- Anfangs gehören alle Trainingsdatensätze zur Wurzel.
- Das nächste Attribut wird ausgewählt (Splitstrategie).
- Die Trainingsdatensätze werden unter Nutzung des Splitattributs partitioniert.
- Das Verfahren wird rekursiv für die Partitionen fortgesetzt.

  $\Rightarrow$ lokal optimierender Algorithmus

Abbruchbedingungen

- keine weiteren Splitattribute
- alle Trainingsdatensätze eines Knotens gehören zur selben Klasse

Entscheidungsbaum-Klassifikatoren

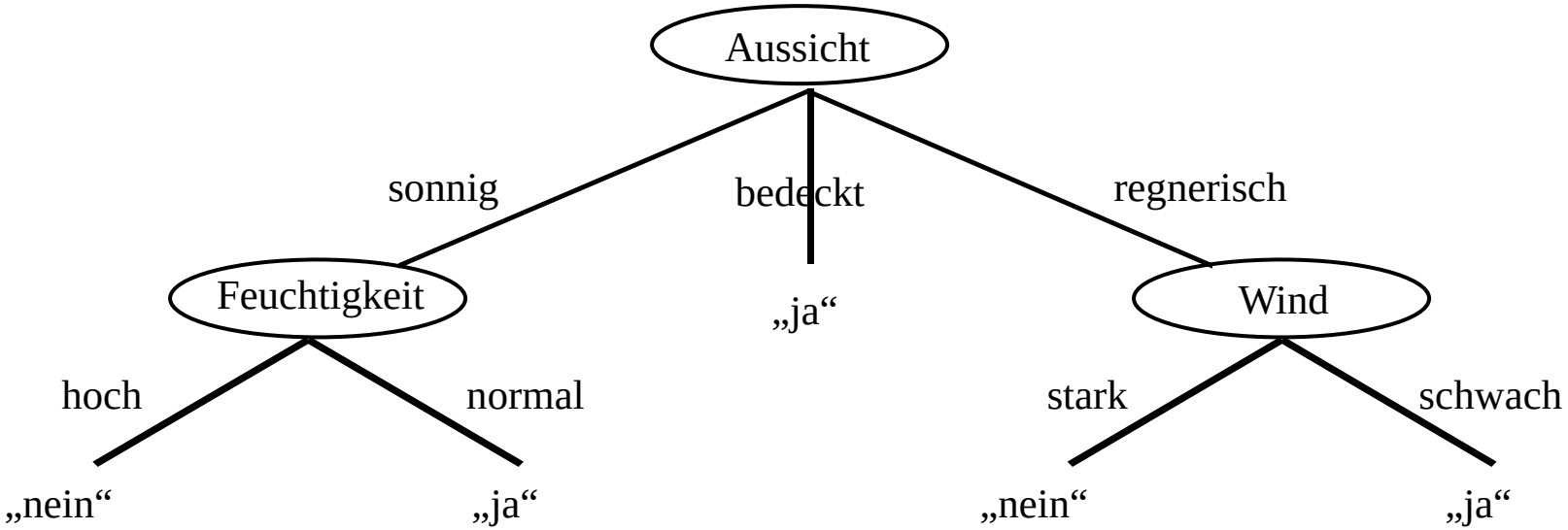
Beispiel

Tag	Aussicht	Temperatur	Feuchtigkeit	Wind	Tennispielen
1	sonnig	heiß	hoch	schwach	nein
2	sonnig	heiß	hoch	stark	nein
3	bedeckt	heiß	hoch	schwach	ja
4	regnerisch	mild	hoch	schwach	ja
5	regnerisch	kühl	normal	schwach	ja
6	regnerisch	kühl	normal	stark	nein
7	bedeckt	kühl	normal	stark	ja
8	sonnig	mild	hoch	schwach	nein
9	sonnig	kühl	normal	schwach	ja
10	regnerisch	mild	normal	schwach	ja
11	sonnig	mild	normal	stark	ja
12	bedeckt	mild	hoch	stark	ja
13	bedeckt	heiß	normal	schwach	ja
14	regnerisch	mild	hoch	stark	nein

Ist heute ein Tag zum Tennisspielen?

Entscheidungsbaum-Klassifikatoren

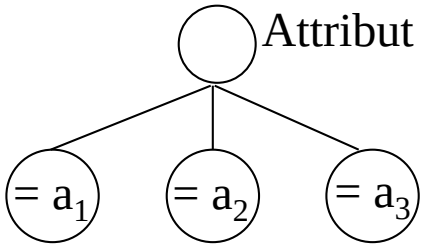
Beispiel

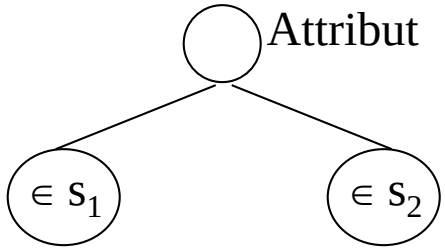


Splitstrategien

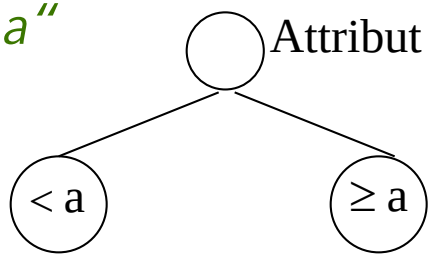
Typen von Splits

- Kategorische Attribute
  Splitbedingungen der Form „*Attribut = a*“ or „*Attribut ∈ set*“
  viele mögliche Teilmengen





- Numerische Attribute
  Splitbedingungen der Form „*Attribut < a*“
  viele mögliche Splitpunkte



Numerische Splitgrenzen

Wo sollen diskrete Attribute gesplittet werden?

=> An den Stellen, die die Qualität maximieren.

Idee: Ordnen der numerischen Attributwerte

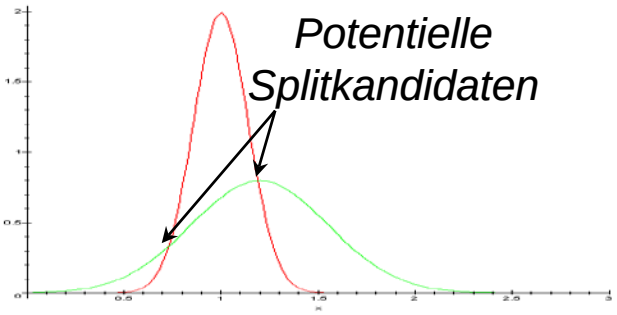
Wert	0.9	0.8	0.65	0.5	0.45	0.3	0.15	0.01
Klasse	A	A	B	B	B	A	A	A

Potentielle Splitkandidaten

Teste die Kombination, die den höchsten Information Gain erzielen.

Schnellere Methode:

- Bilde Gauß-Kurve über alle Klassen
- Wähle Schnittpunkte der Gauß-Kurven als Kandidaten.



Splitstrategien

Qualitätsmaße für Splits

Gegeben

- eine Menge T von Trainingsobjekten
- eine disjunkte, vollständige Partitionierung $T_1, T_2, \ldots, T_m$ von T
- p_i die relative Häufigkeit der Klasse c_i in T

Gesucht

- ein Maß der *Unreinheit* einer Menge S von Trainingsobjekten in Bezug auf die Klassenzugehörigkeit
- ein Split von T in $T_1, T_2, \ldots, T_m$, der dieses Maß der Unreinheit *minimiert*

  $\Rightarrow$ Informationsgewinn, Gini-Index

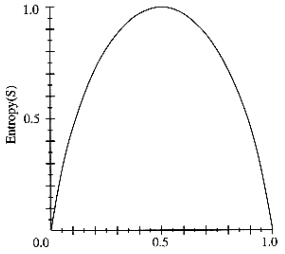
Splitstrategien

Informationsgewinn

Entropie: minimale Anzahl von Bits zum Codieren der Nachricht, mit der man die Klasse eines zufälligen Trainingsobjekts mitteilen möchte.

Die *Entropie* für eine Menge T von Trainingsobjekten ist definiert als

$$entropie(T) = -\sum_{i=1}^{k} p_i \cdot \log p_i$$



$entropie(T) = 0$, falls $p_i = 1$ für ein i

$entropie(T) = 1$ für $k = 2$ Klassen mit $p_i = 1/2$

Das Attribut A habe die Partitionierung $T_1, T_2, \ldots, T_m$ erzeugt.

Der *Informationsgewinn* des Attributs A in Bezug auf T ist definiert als

$$Informationsgewinn(T, A) = entropie(T) - \sum_{i=1}^{m} \frac{|T_i|}{|T|} \cdot entropie(T_i)$$

Gini-Index

Gini-Index für eine Menge T von Trainingsobjekten

$$gini(T) = 1 - \sum_{j=1}^{k} p_j^2$$

- kleiner Gini-Index $\Leftrightarrow$ geringe Unreinheit,
- großer Gini-Index $\Leftrightarrow$ hohe Unreinheit

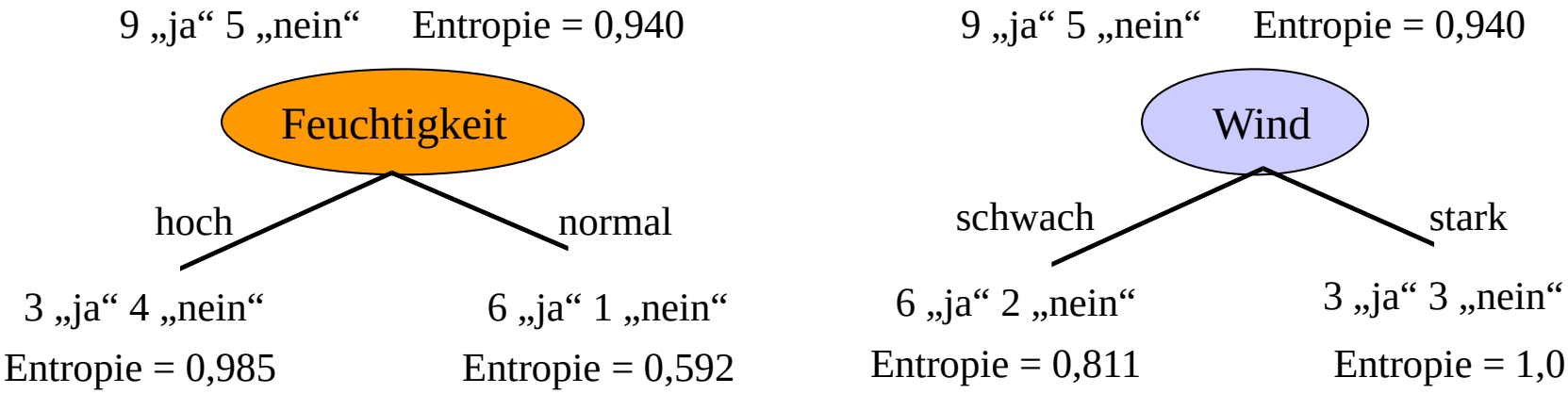
Das Attribut A habe die Partitionierung $T_1, T_2, \ldots, T_m$ erzeugt.

Gini-Index des Attributs A in Bezug auf T ist definiert als

$$gini_A(T) = \sum_{i=1}^{m} \frac{|T_i|}{|T|} \cdot gini(T_i)$$

Splitstrategien

Beispiel

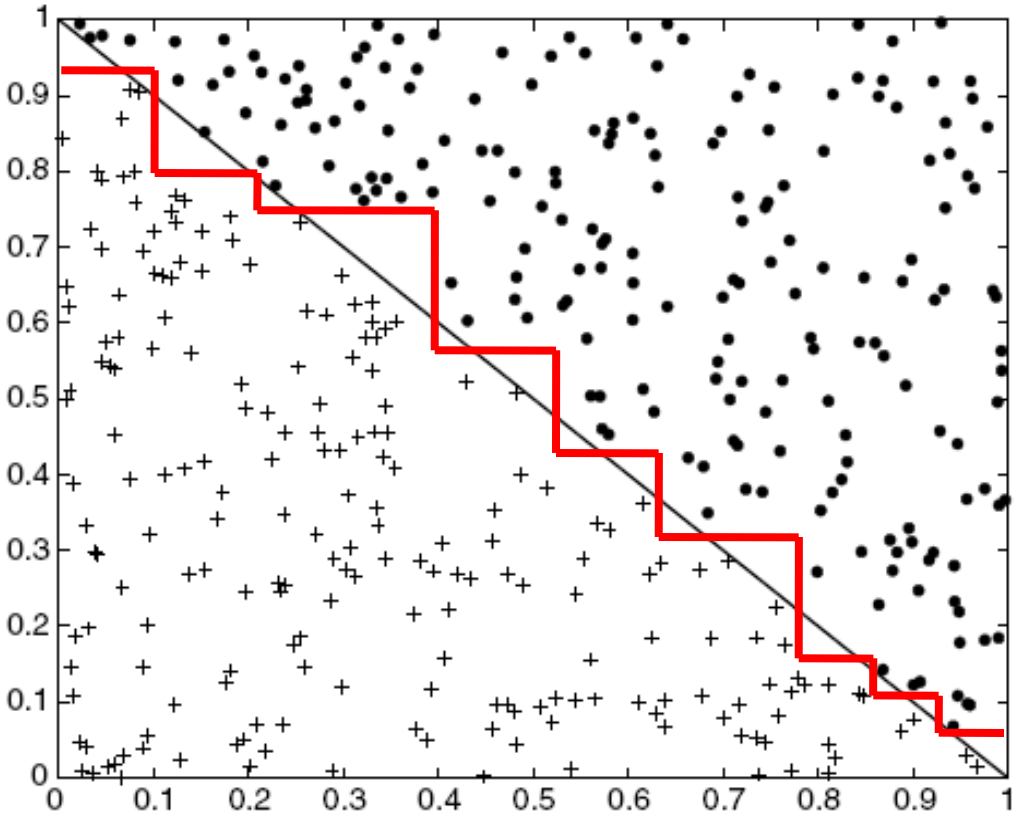


$$Informationsgewinn(T, Feuchtigkeit) = 0{,}94 - \frac{7}{14} \cdot 0{,}985 - \frac{7}{14} \cdot 0{,}592 = 0{,}151$$

$$Informationsgewinn(T, Wind) = 0{,}94 - \frac{8}{14} \cdot 0{,}811 - \frac{6}{14} \cdot 1{,}0 = 0{,}048$$

$\Rightarrow$ Feuchtigkeit liefert den höheren Informationsgewinn

Bias

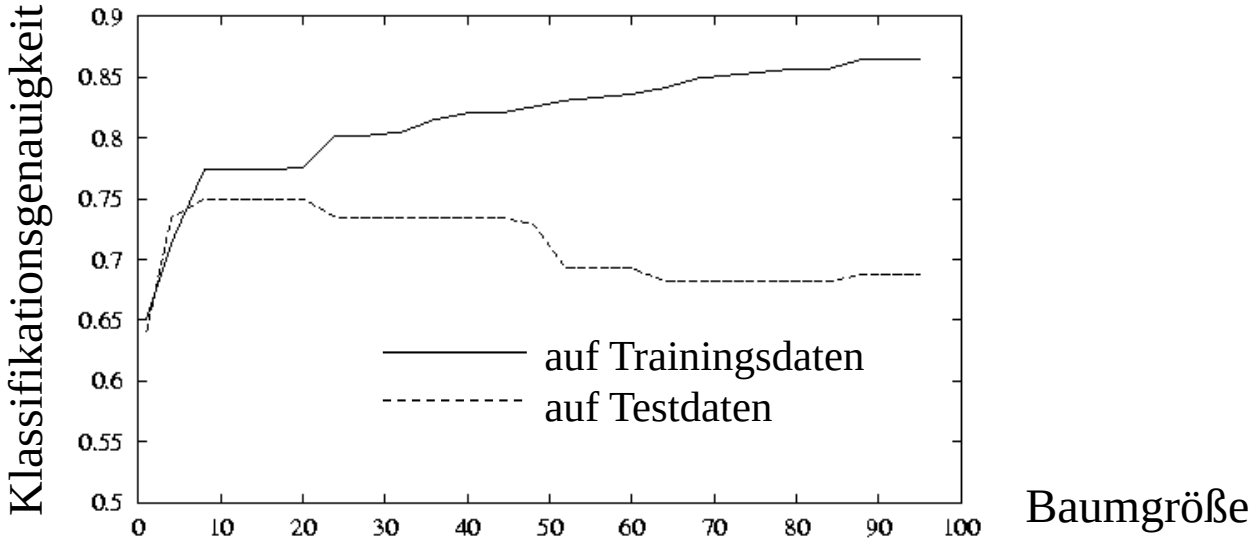


Overfitting

Einführung

Overfitting bei der Konstruktion eines Entscheidungsbaums, wenn es zwei Entscheidungsbäume E und E' gibt mit

- E hat auf der Trainingsmenge eine kleinere Fehlerrate als E',
- E' hat auf der Grundgesamtheit der Daten eine kleinere Fehlerrate als E.



Overfitting

Trainingsdaten

Name	Body Temp	Gives Birth	Four-legged	Hibernates	Mammal
porcupine	warm-blooded	yes	yes	yes	yes
cat	warm-blooded	yes	yes	no	yes
bat	warm-blooded	yes	no	yes	no*
whale	warm-blooded	yes	no	no	no*
salamander	cold-blooded	no	yes	yes	no
komodo dragon	cold-blooded	no	yes	no	no
python	cold-blooded	no	no	yes	no
salmon	cold-blooded	no	no	no	no
eagle	warm-blooded	no	no	no	no
guppy	cold-blooded	yes	no	no	no

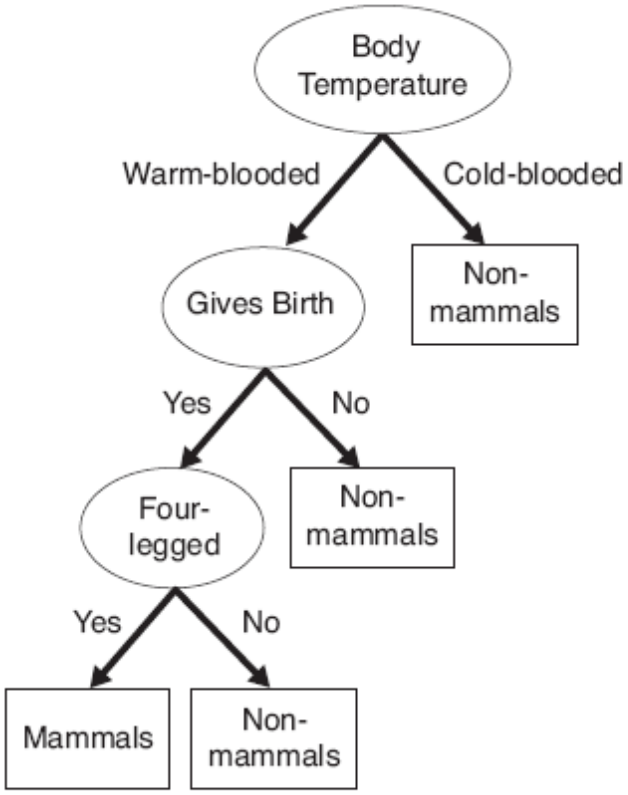
*Fehlerhafte Daten

Overfitting

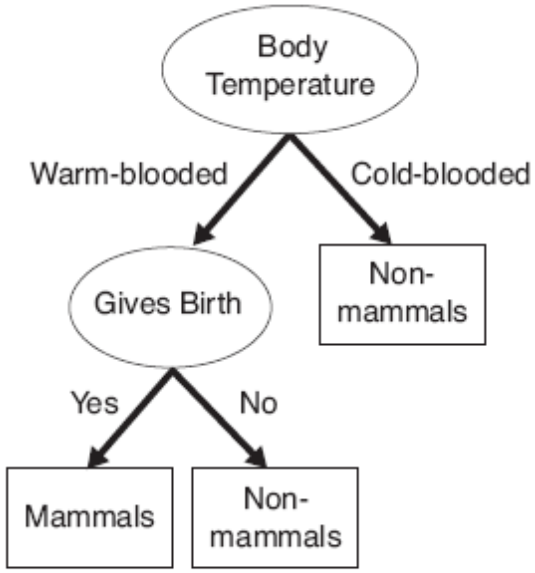
Testdaten

Name	Body Temp	Gives Birth	Four-legged	Hibernates	Mammal
human	warm-blooded	yes	no	no	yes
pigeon	warm-blooded	no	no	no	no
elephant	warm-blooded	yes	yes	no	yes
leopard shark	cold-blooded	yes	no	no	no
turtle	cold-blooded	no	yes	no	no
penguin	warm-blooded	no	no	no	no
eel	cold-blooded	no	no	no	no
dolphin	warm-blooded	yes	no	no	yes
spiny anteater	warm-blooded	no	yes	yes	yes
gila monster	cold-blooded	no	yes	yes	no

Overfitting



(a) Model M1



(b) Model M2

Trainingserror: 0%
Testerror: 30%

Trainingserror: 20%
Testerror: 10%

M1:

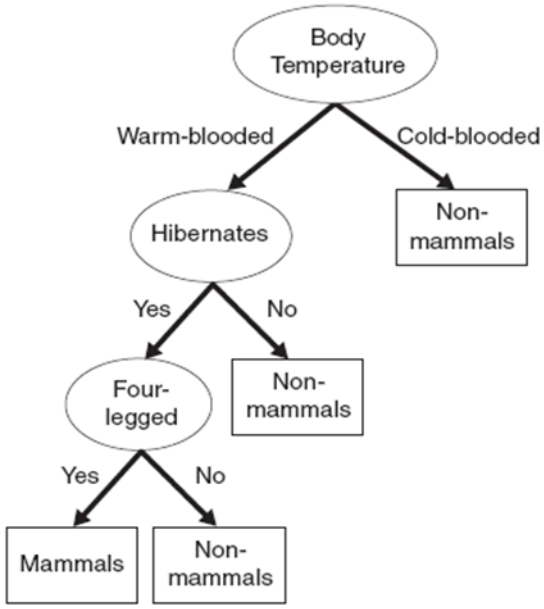
- Fehlklassifikation von human und dolphin aufgrund von fehlerhaften Trainingsdaten
- spiny anteater: ungewöhnlicher Fall

M2:

ungewöhnlicher Fall kann nicht vermieden werden

Overfitting

Name	Body Temp	Gives Birth	Four-legged	Hibernates	Mammal
salamander	cold-blooded	no	yes	yes	no
poorwill	warm-blooded	no	no	yes	no
platypus	warm-blooded	no	yes	yes	yes
eagle	warm-blooded	no	no	no	no
guppy	cold-blooded	yes	no	no	no



Problem:

Mangel an repräsentativen Daten

$\Rightarrow$ Fehlerrate auf Testset: 30%

Overfitting

Ansätze zum Vermeiden von Overfitting

- Entfernen von fehlerhaften Trainingsdaten

  insbesondere widersprüchliche Trainingsdaten
- Wahl einer geeigneten Größe der Trainingsmenge

  nicht zu klein, nicht zu groß
- Wahl einer geeigneten Größe des minimum support

  minimum support:

  Anzahl der Datensätze, die mindestens zu einem Blattknoten des Baums gehören müssen

  minimum support >> 1

Overfitting

Ansätze zum Vermeiden von Overfitting

- Wahl einer geeigneten Größe der minimum confidence

  minimum confidence: Anteil, den die Mehrheitsklasse eines Blattknotens mindestens besitzen muß.

  minimum confidence << 100%

  Blätter können auch fehlerhafte Datensätze oder Rauschen „absorbieren“

- nachträgliches Pruning des Entscheidungsbaums

  Abschneiden der überspezialisierten Äste

  kleinere Bäume sind weniger anfällig für Overfitting (Occam’s Razor)

Pruning von Entscheidungsbäumen

Fehlerreduktions-Pruning [Mitchell 1997]

- Aufteilung der klassifizierten Daten in Trainingsmenge und Testmenge
- Konstruktion eines Entscheidungsbaums *E* für die Trainingsmenge
- Pruning von *E* mit Hilfe der Testmenge *T*
  - bestimme denjenigen Teilbaum von *E*, dessen Abschneiden den Klassifikationsfehler auf *T* am stärksten reduziert
  - entferne diesen Teilbaum
  - fertig, falls kein solcher Teilbaum mehr existiert

nur anwendbar, wenn genügend viele klassifizierte Daten

Entscheidungsbaum-Klassifikatoren

Diskussion

+ Interpretation des gefundenen Baumes relativ einfach
+ Implizite Gewichtung der Attribute
+ Leistungsfähiger Klassifikator, häufig in der Praxis verwendet
+ Effiziente Auswertung des gefundenen Modells

- Finden eines optimalen Entscheidungsbaums ist exponentiell
- Heuristische Methoden können nur lokales Optimum finden
- Anfällig für Overfitting (besondere Methoden zur Vermeidung von Overfitting für Entscheidungsbäume)

5.6 Neuronale Netze

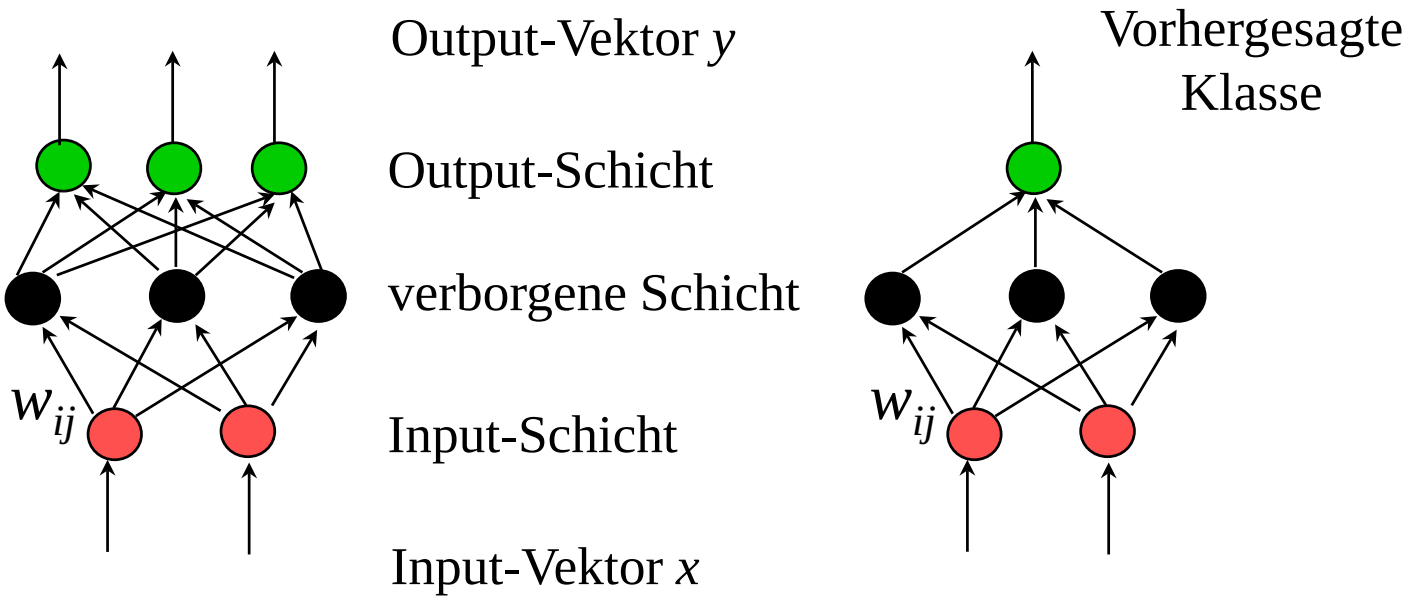
Grundlagen [Bigus 1996], [Bishop 1995]

- 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 ~10^4 anderen, schnellste Aktionszeit 10^{-3} Sek. (3GHz Prozessor: ~10^{-10}) – aber 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

Grundlagen

Kanten besitzen *Gewichte*

Funktion eines neuronalen Netzes

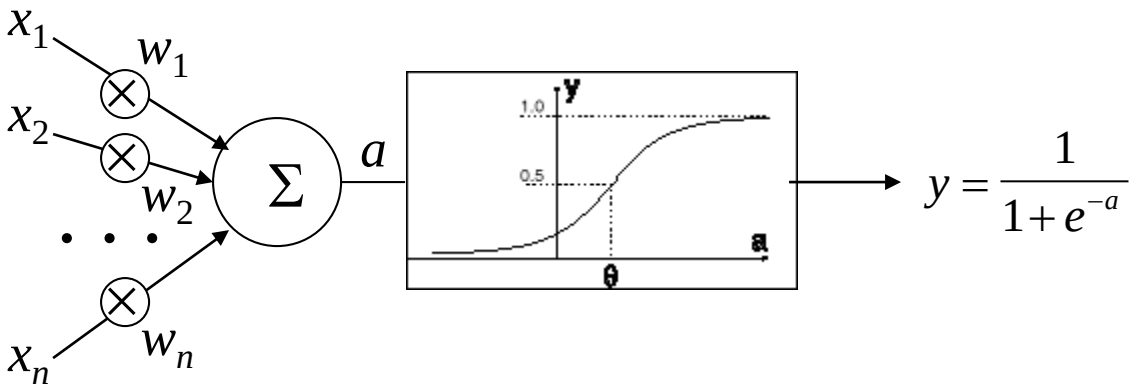


Neuronen

allgemeines Neuron (auch: Perzeptron)

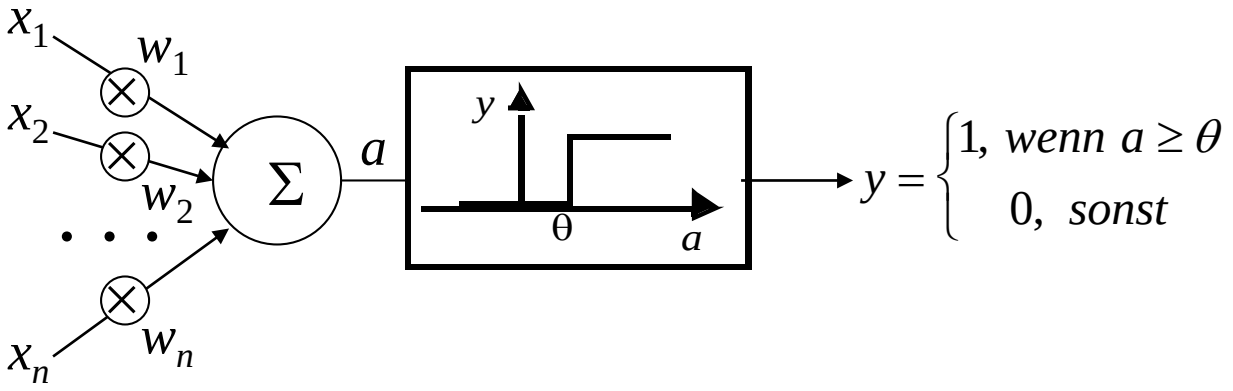
a: *Aktivierungswert*

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



Threshold Logic Unit

(TLU)

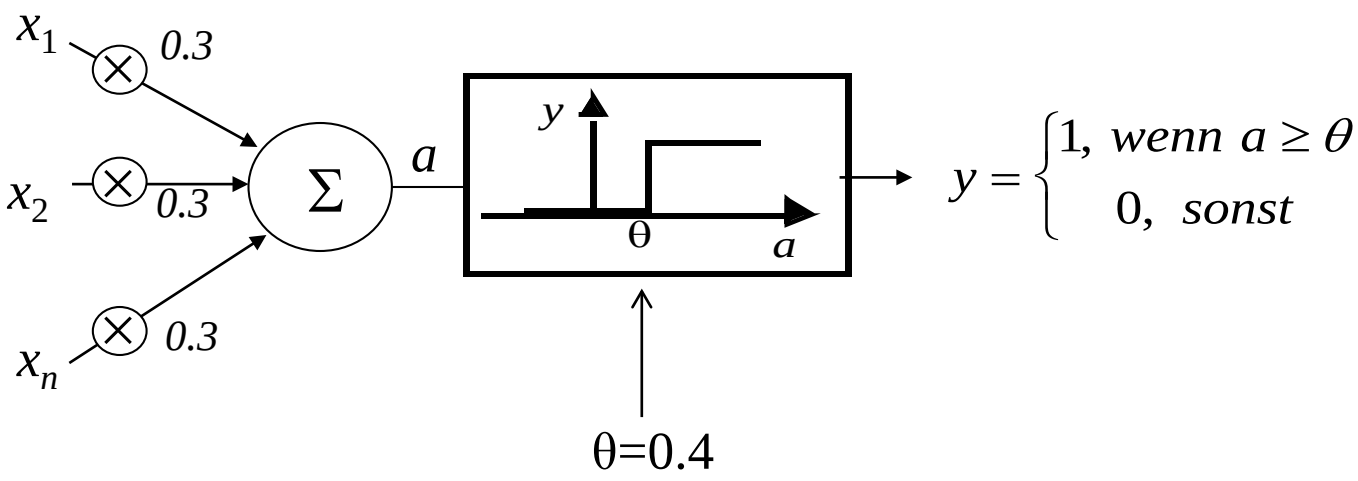


Neuronen

Beispiel: Modellierung einer booleschen Funktion

(zwei oder mehr Variablen = 1 ⇒ y=1)

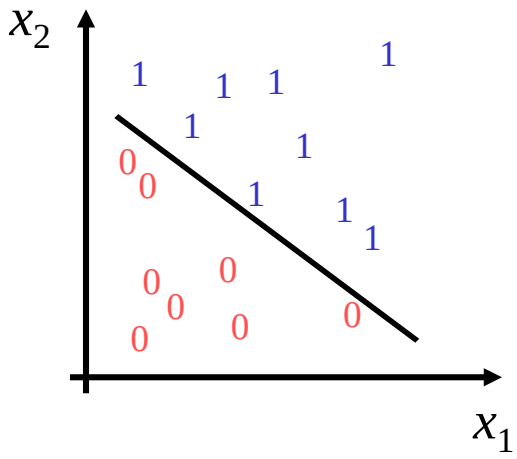
x_1	x_2	x_3	y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0



⇒ Lernen der Gewichte w_i

Neuronen

- 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

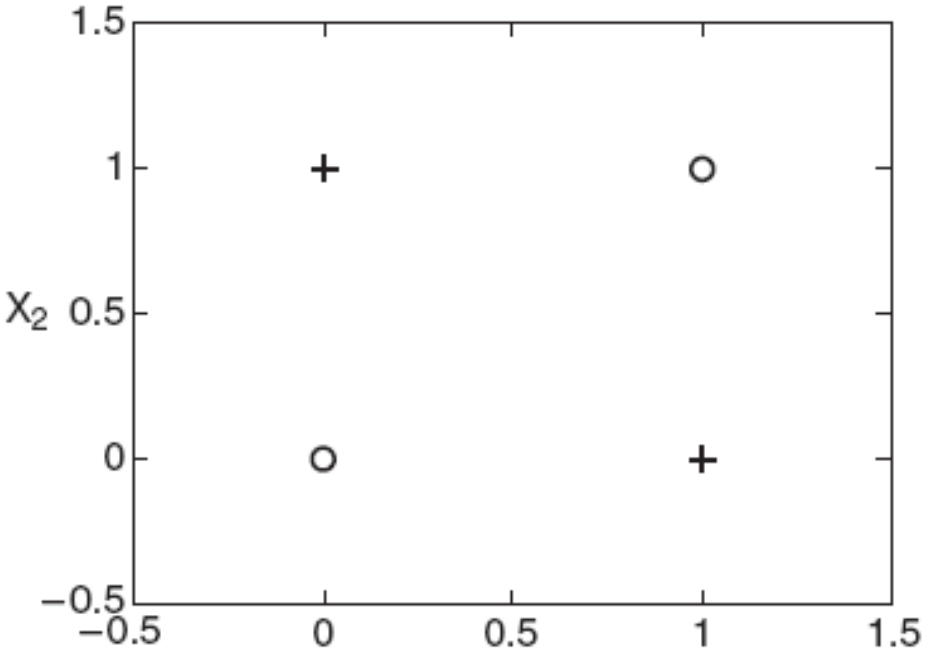


- Trainieren einer TLU
  - Lernen der „richtigen“ Gewichte zur Unterscheidung der zwei Klassen: iterative Anpassung der Gewichte w_{ij}
  - Rotation der durch w und θ gegebene Hyperebene um einen kleinen Betrag in Richtung v, wenn v noch nicht auf der richtigen Seite der Ebene liegt

XOR-Problem

nicht linear separierbares Problem

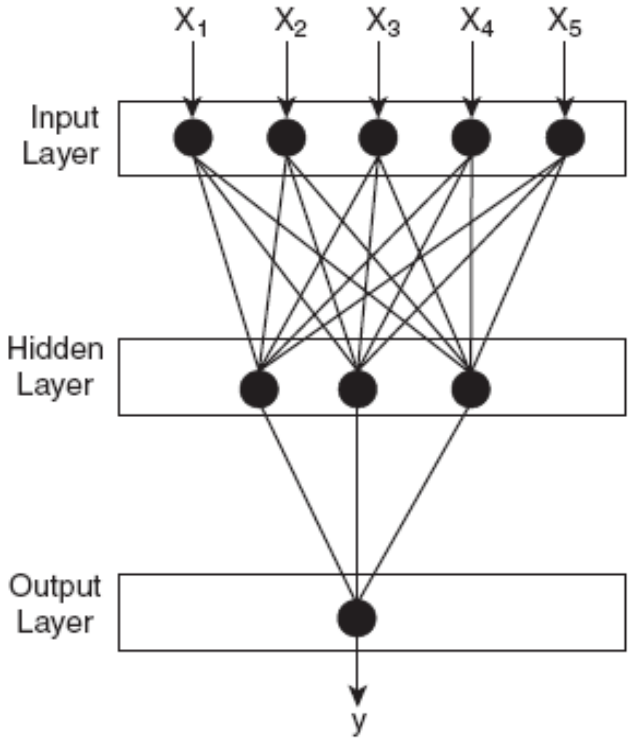
X_1	X_2	y
0	0	−1
1	0	1
0	1	1
1	1	−1



Kombination mehrerer Neuronen

zwei Klassen, die nicht linear separierbar sind:

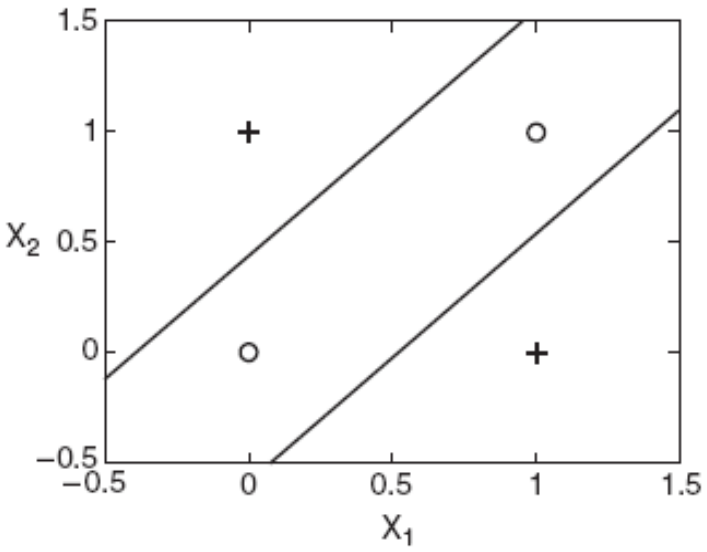
⇒ mehrere innere Knoten (hidden layer) und ein Output-Knoten (zwei-Klassen-Problem, sonst mehr)

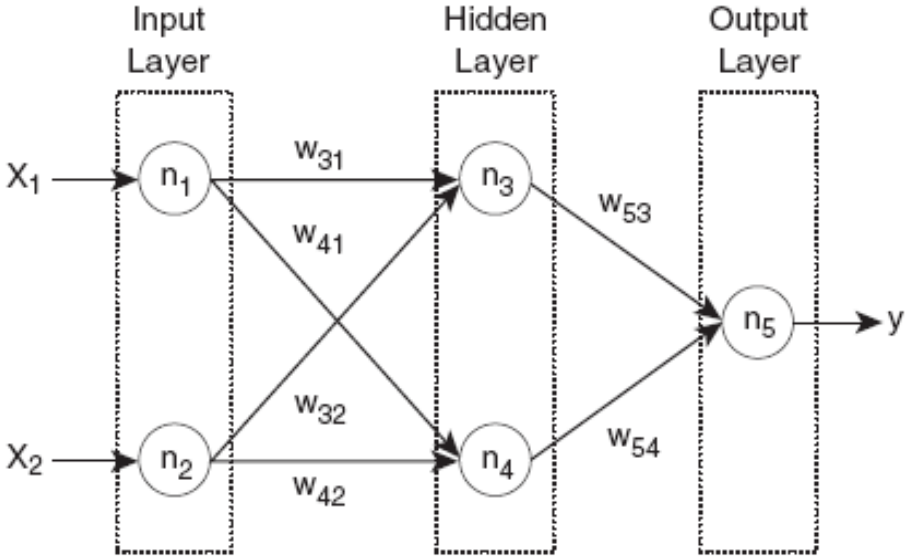


Varianten: feed-forward vs. recurrent (auch Verbindungen auf derselben Ebene oder zu vorherigen Ebenen möglich)

Kombination mehrerer Neuronen

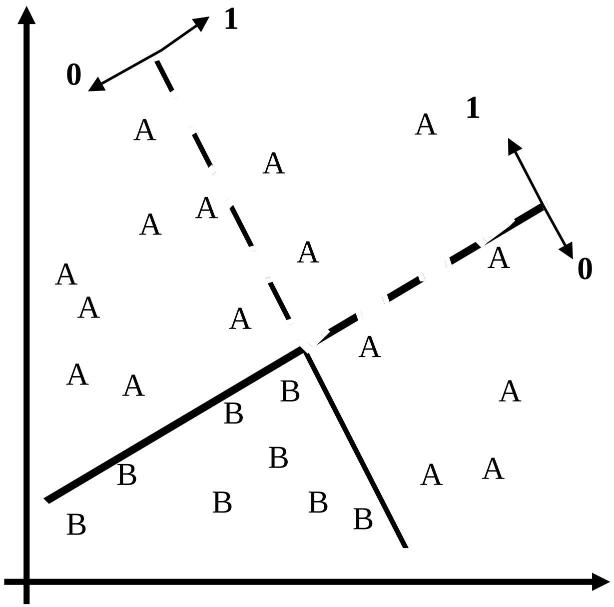
XOR

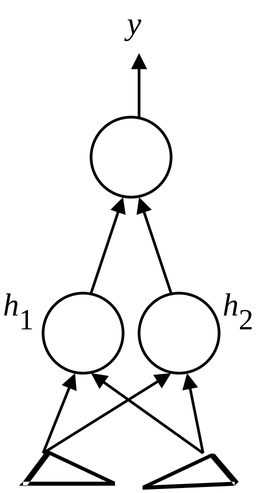




Kombination mehrerer Neuronen

Beispiel





$h_1 = 0 \land h_2 = 0{:}\ y = 0\ (Klasse\ \ B)$

$andernfalls{:}\ y = 1\ (Klasse\ \ A)$

Lernalgorithmus für komplexe Neuronale Netze

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*

Algorithmus Backpropagation

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;

Design der Netztopologie

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

  $\Rightarrow$ niedrige Klassifikationsgüte
- zu viele Knoten

  $\Rightarrow$ Overfitting

Bestimmung der Netztopologie

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

Bestimmung der Netztopologie

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.

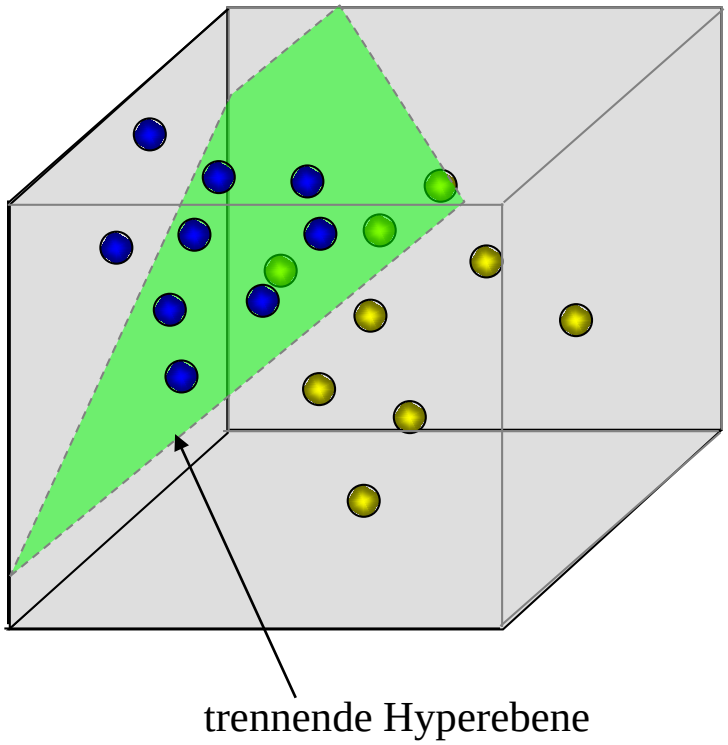
Diskussion

+ 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

5.7 Support Vector Machines

Motivation: Lineare Separation



- 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.

Support Vector Machines

Probleme bei linearer Separation:

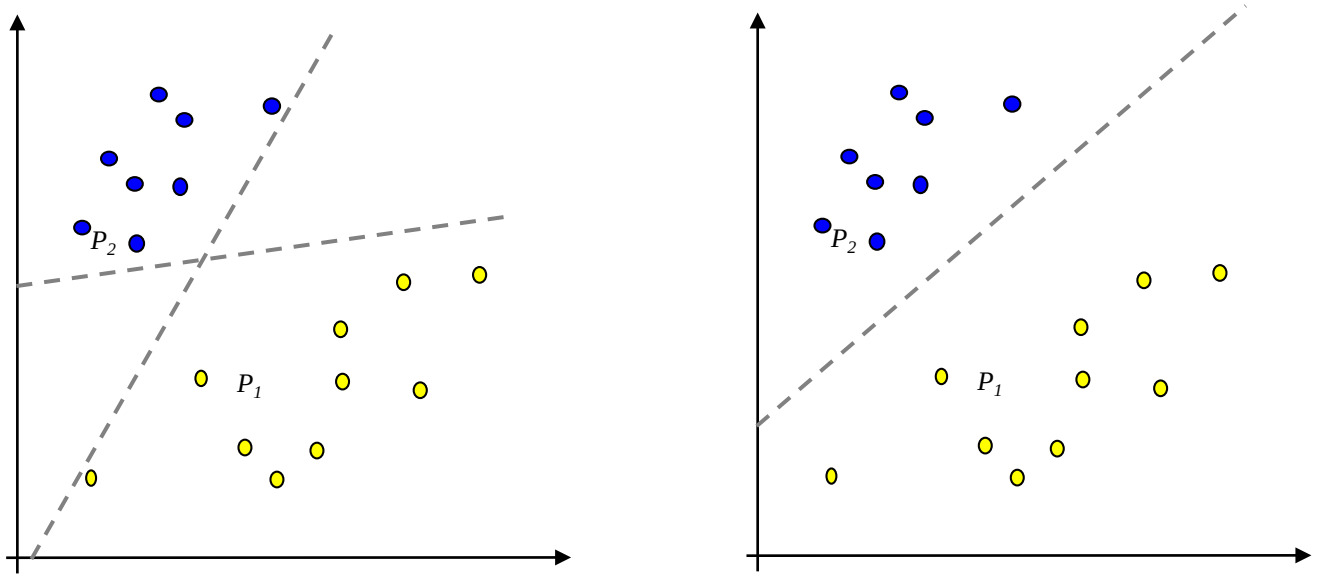
- Was ist die „maximal stabile“ Hyperebene und wie berechnet man sie effizient?
- Klassen nicht immer linear trennbar.
- Berechnung von Hyperebenen nach Auswahl sehr aufwendig.
- Einschränkung auf 2 Klassen.
- ...

⇨ *Lösungen dieser Probleme mit Support Vector Machines(SVMs) [Vapnik 1979 u. 1995].*

Maximum Margin Hyperplane

Problem: Hyperebene, die P_1 und P_2 trennt, ist nicht eindeutig.

$\Rightarrow$ *Welche Hyperebene ist für die Separation die Beste ?*

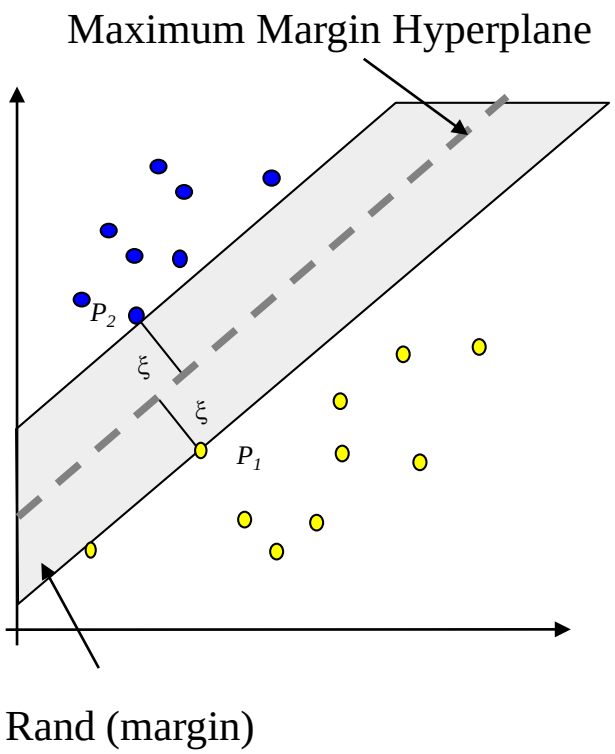


Kriterien:

- Stabilität beim Einfügen
- Abstand zu den Objekten beider Klassen

Maximum Margin Hyperplane

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*

Maximum Margin Hyperplane

Zusammenfassung der Schreibweisen der benötigten algebraischen Konstrukte für Featurespace *FS*:

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

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

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

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

Maximum Margin Hyperplane

Berechnung der Maximum Margin Hyperplane

1. Bedingung: kein Klassifikationsfehler (Klasse1: y_i=1,Klasse 2:y_i=-1)

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

2. Bedingung: Maximaler Rand (Margin)

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

oder

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

Maximum Margin Hyperplane

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

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

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 $\left(y_i \left(\langle \vec{w}, \vec{x}_i \rangle + b \right) \geq 1 \right)$

Maximum Margin Hyperplane

Zur Berechnung wird das primäre Optimierungsproblem in ein duales OP überführt.
(Umformulierung in Form mit Langrange Multiplikatoren nach Karush-Kuhn-Tucker)

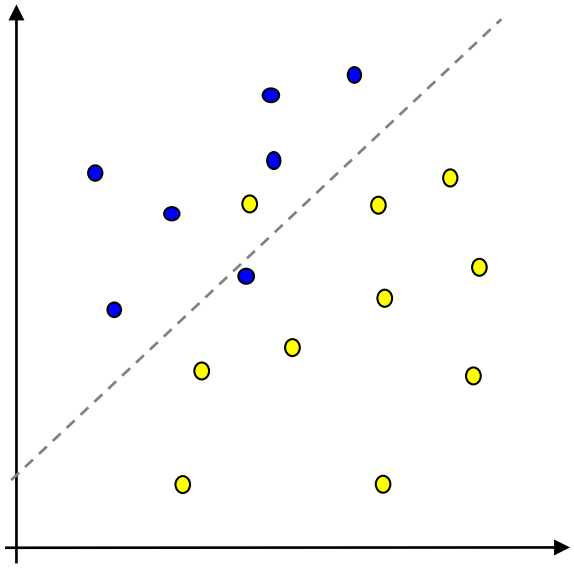
Duales OP: maximiere $$L(\vec{\alpha}) = \left(\sum_{i=1}^{n} \alpha_i \right) - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot \left\langle \vec{x}_i, \vec{x}_j \right\rangle$$

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

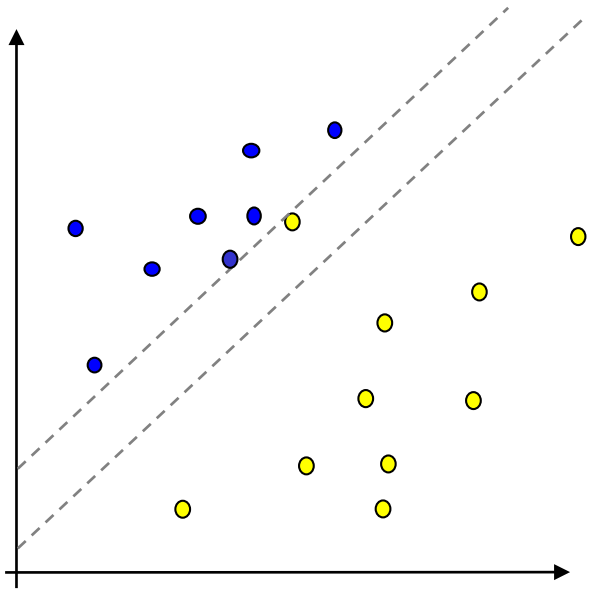
$\Rightarrow$ Lösung des Problems mit Algorithmen aus der Optimierungstheorie
$\Rightarrow$ bis jetzt nur linear separierbarer Fall: Soft Margin Optimierung
$\Rightarrow$ Einführung von Kernelfunktionen zur Steigerung der Kapazität

Soft Margin

Behandlung nicht linear trennbarer Daten: *Soft Margin Optimierung*



Daten nicht separierbar

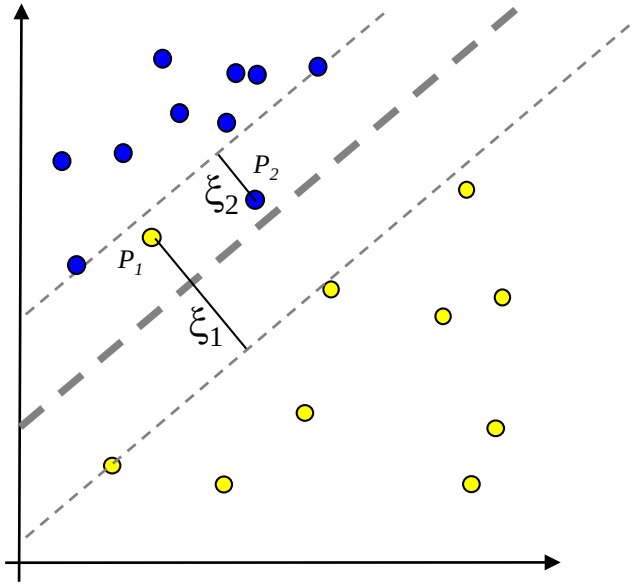


vollständige Separation ist nicht optimal

$\Rightarrow$ Trade-Off zwischen Trainingsfehler und Breite des Randes

Soft Margin

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 :

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

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

$\Rightarrow$ Primäres Optimierungsproblem unter weichen Grenzen (Soft Margin)

Soft Margin

Das duale OP mit Lagrange Multiplikatoren verändert sich wie folgt:

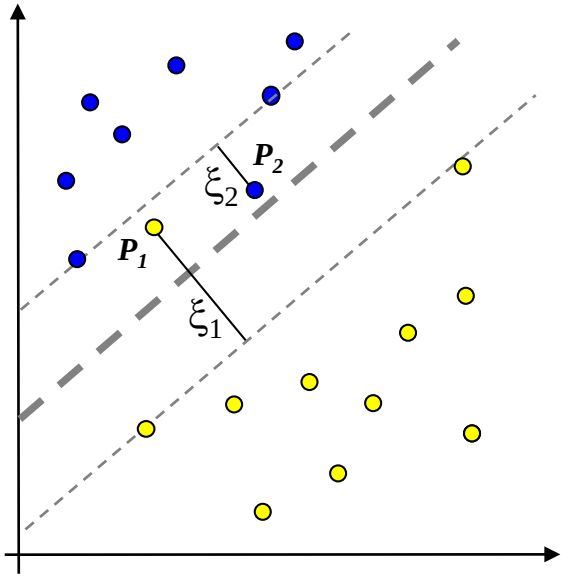
Duales OP: maximiere $L(\vec{\alpha}) = \left(\sum_{i=1}^{n} \alpha_i\right) - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot \left\langle \vec{x}_i, \vec{x}_j \right\rangle$

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

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

Entscheidungsregel:

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



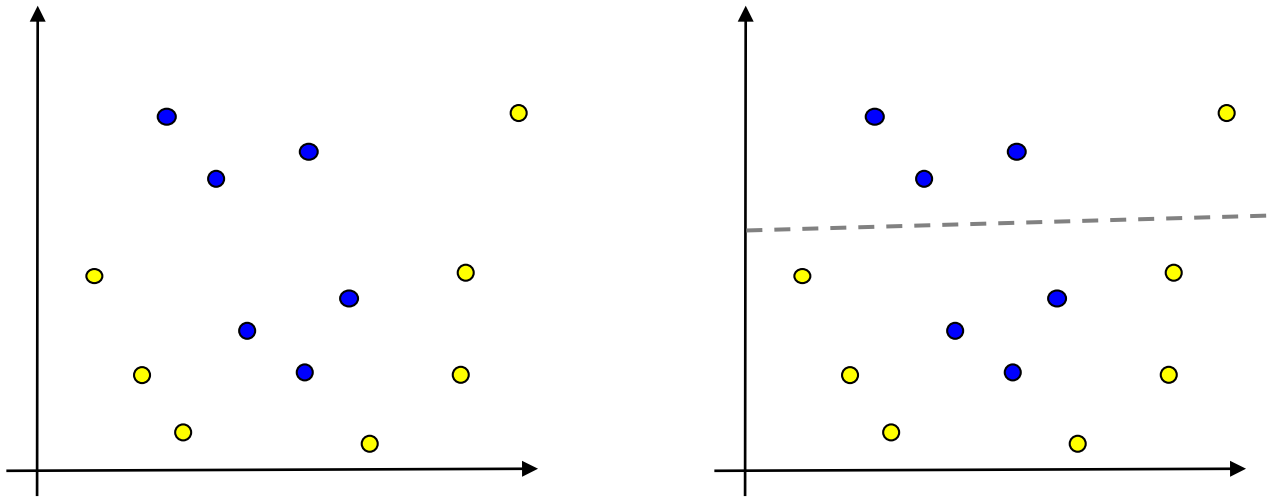
Kernel Machines

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.

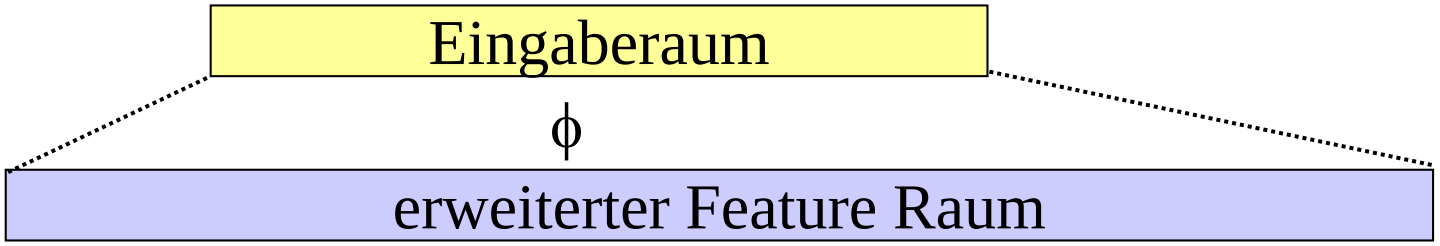
(Erweiterung des Hypothesenraumes)



Beispiel: quadratische Transformation

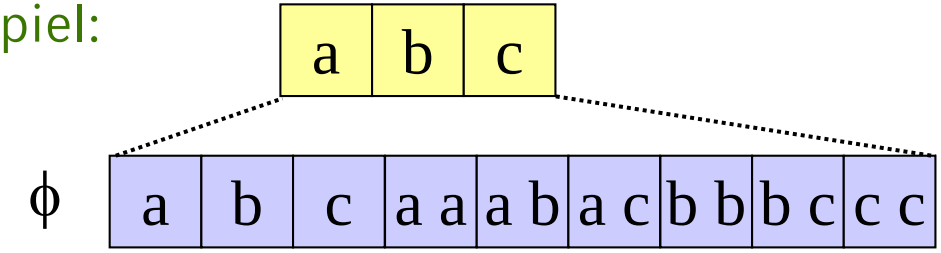
Kernel Machines

Erweiterung der Hypothesenraumes



$\Rightarrow$ Versuche jetzt in erweitertem Feature Raum linear zu separieren

Beispiel:

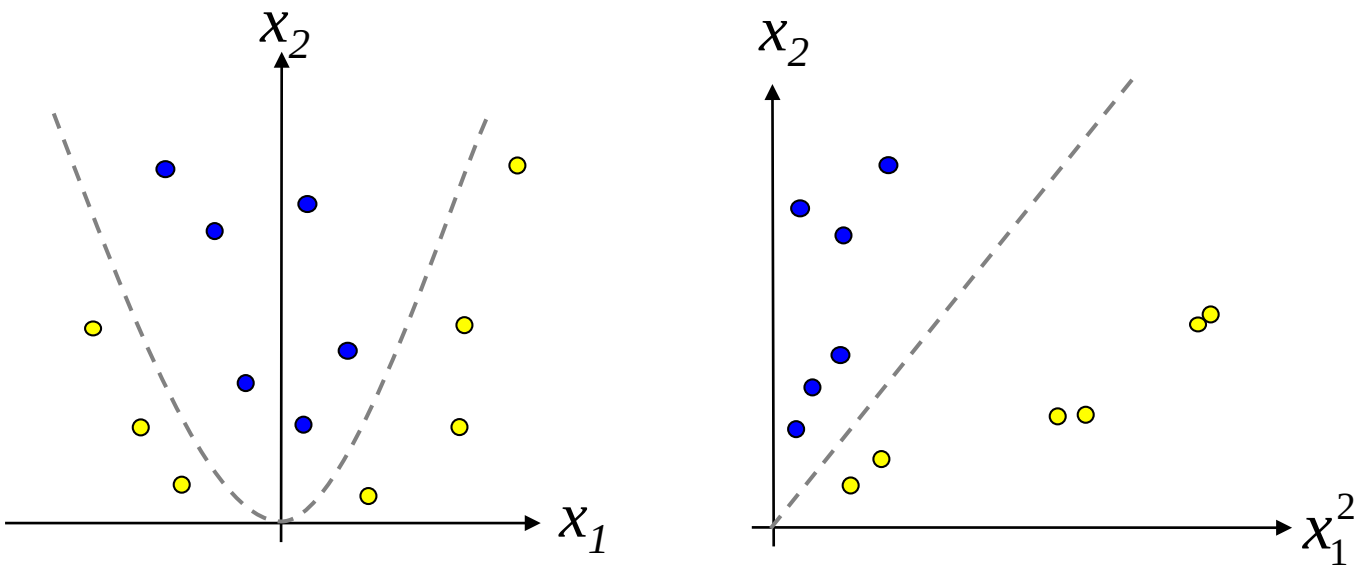


hier: Eine Hyperebene im erweiterten Feature Raum ist ein Polynom 2. Grades im Eingaberaum.

Kernel Machines

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)



Kernel Machines

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

Duales OP: maximiere

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

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. ⇒ Kernel *K* ist eine Funktion mit:

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

Kernel Machines

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:

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

Symmetrie und Cauchy-Schwarz sind keine hinreichenden Bedingungen!

Kernel Machines

einige Regeln zur Kombination vom Kerneln:

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

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

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

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

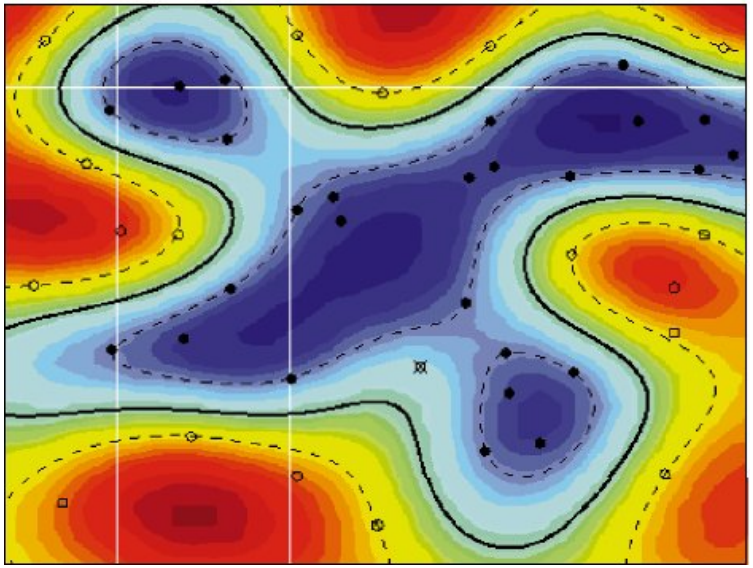
für K_1, K_2 Kernelfunktionen, a eine positive Konstante und B eine symmetrische positiv semi-definite Matrix.

Kernel Machines

Beispiele für verwendete Kernel-Funktionen:

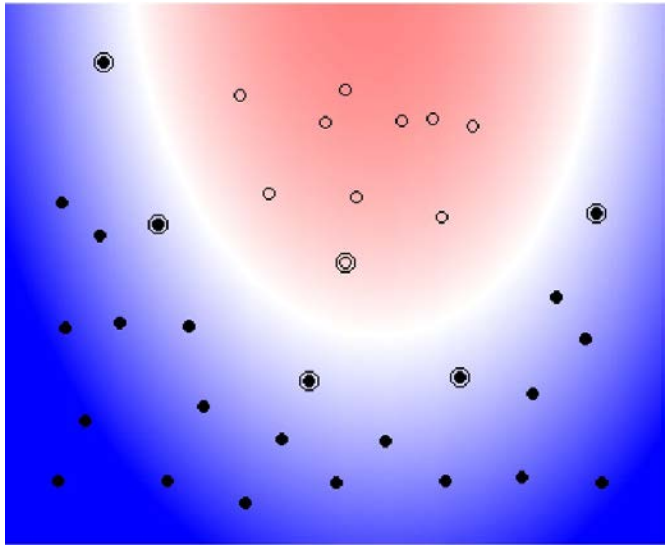
- linear: $K(\vec{x}, \vec{y}) = \langle \vec{x}, \vec{y} \rangle$
- polynomiell: $K(\vec{x}, \vec{y}) = \left(\langle \vec{x}, \vec{y} \rangle + c \right)^d$
- Radiale Basisfunktionen: $K(\vec{x}, \vec{y}) = \exp\left(-\gamma \cdot \left| \vec{x} - \vec{y} \right|^2 \right)$
- Gauss Kernel: $K(\vec{x},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

Kernel Machines



Radial Basis Kernel

Polynomieller Kernel (Grad 2)



Training einer SVM

zu lösen ist folgendes Problem:

Duales OP: maximiere

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

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

oder

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

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

Training einer SVM

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*)

Multi-Class SVMs

Bisher SVMs nur anwendbar auf 2 Klassen Probleme !!

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

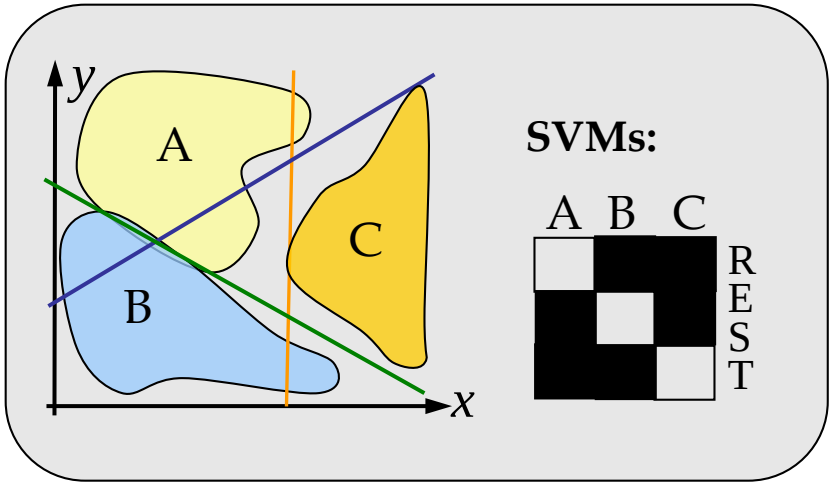
Multi-Class SVMs

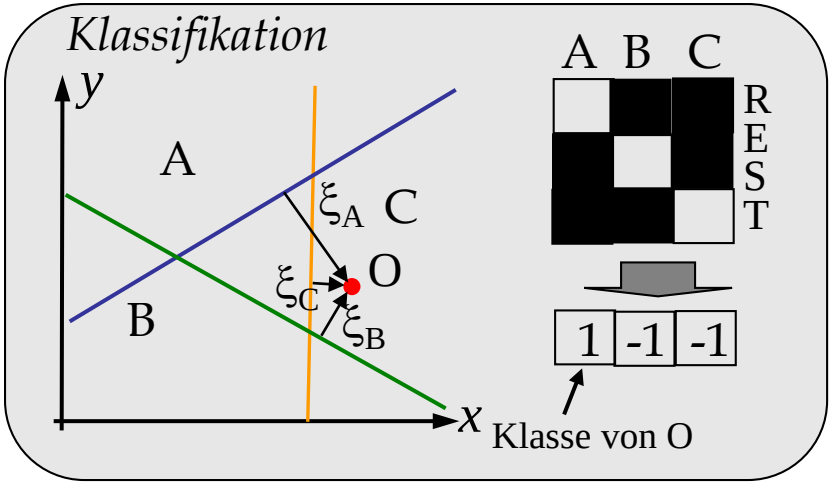
2 klassische Ansätze:

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

⇒ Unterscheide je 2 Klassen *(1-versus-1)*

1-versus-Rest Ansatz





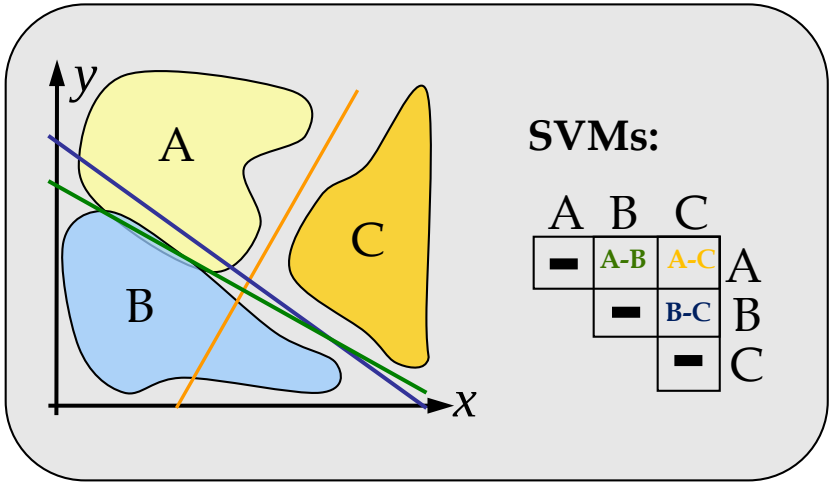
- 1-versus-Rest Ansatz : 1 SVM für jede Klasse.
- SVM trennt jedes Klasse von Vereinigung aller anderen Klassen ab
- Klassifiziere O mit allen Basis-SVMs.

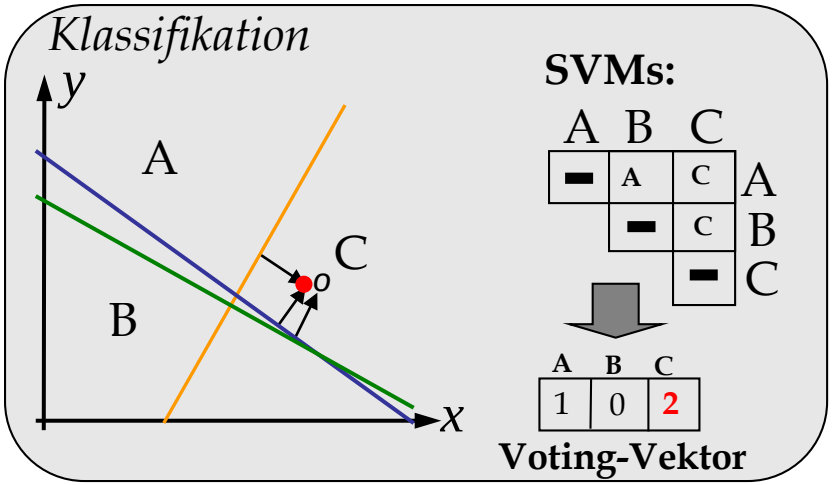
  $\Rightarrow$ **Multiple Klassenzugehörigkeit möglich (Multi-Classification)**

  oder

  Entscheidung für die Klasse, bei der Abstand ξ_i am größten ist.

1-versus-1 Ansatz





- 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)

Support Vector Machines

Diskussion

+ erzeugt Klassifikatoren mit hoher Genauigkeit
+ verhältnismäßig schwache Tendenz zu Overfitting
  (Begründung durch Generalisierungtheorie)
+ effiziente Klassifikation neuer Objekte
+ kompakte Modelle

- unter Umständen lange Trainingszeiten
- aufwendige Implementierung
- gefundene Modelle schwer zu deuten

Support Vector Machines

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 Shawne-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