Und wenn Klassen nicht linear separierbar sind?

Support-Vektor-Maschinen (SVM) …

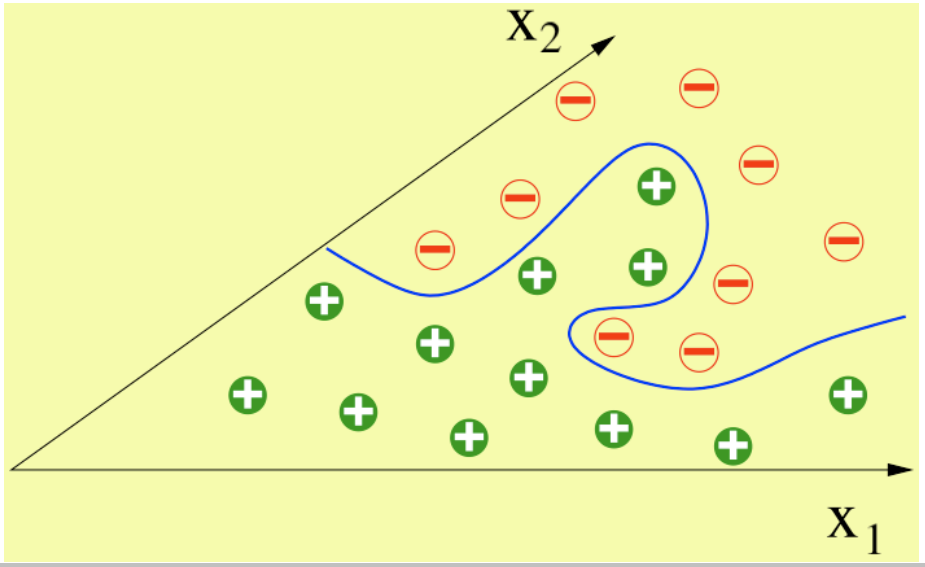
… transformieren die Eingaben (Lernbeispiele, „Vektoren“) in einen Parameterraum, in dem sie linear separierbar *sind*! (das ist nicht trivial …)

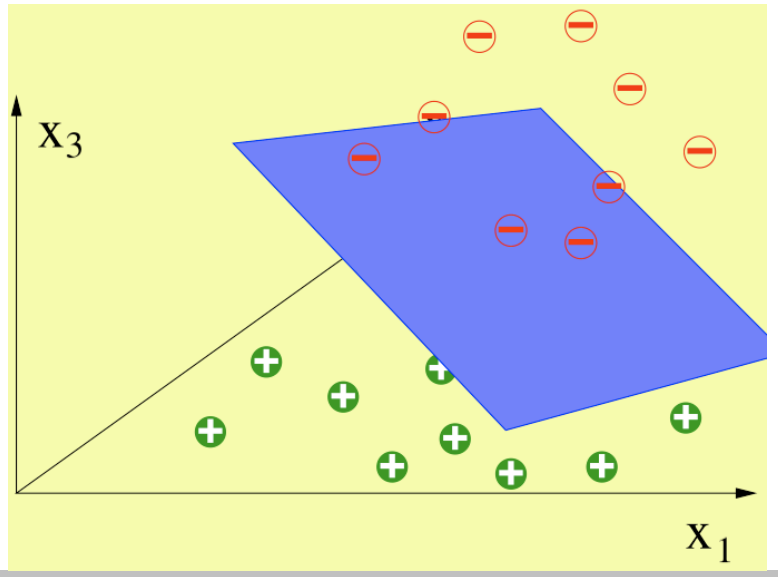
… verwenden dafür 2 Ideen:
(1) die Transformation („Kernel Trick“) und
(2) Finden einer optimal separierenden Hyperebene im transformierten Parameterraum

Beide Ideen nachfolgend nur angedeutet!

SVM: Der Kernel Trick (Prinzip)

- Der **Kernel** ist eine Funktion, die den originalen n-dimensionalen Merkmalsraum in einen anderen, höherdimensionalen transformiert (in Abhängigkeit der Klassenzuordnung der Trainingsbeispiele), in dem die Trainingsbeispiele linear separierbar sind.
- Das geht immer für widerspruchsfreie Daten, aber guter Kernel ist i.a. nicht leicht zu konstruieren

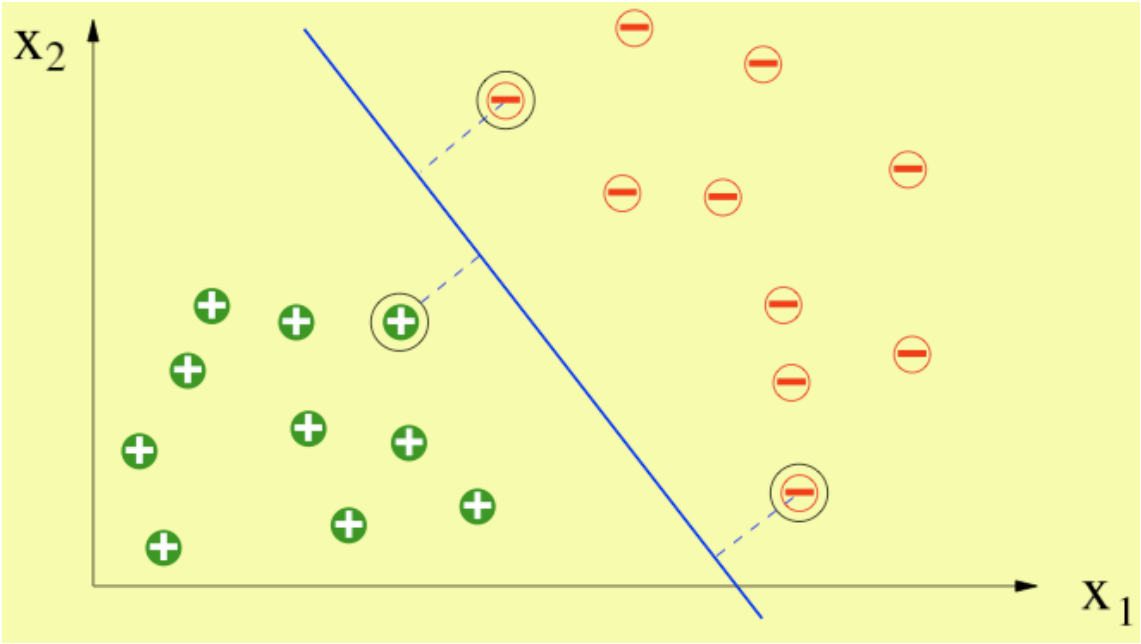




SVM: Bestimme Support-Vektoren

Support-Vektoren sind diejenigen Eingabebeispiele (Vektoren), die im transformierten Parameterraum den minimalen Abstand zur optimal separierenden Hyperebene haben.

Kernel & Support-Vektoren hängen voneinander ab.



Bestimmung von Kernel & Support-Vektoren ist nicht trivial, daher hier (und bei Ertel) nicht vertieft!

Nearest-Neighbor-Verfahren

Methode

- Auswendiglernen mit lokaler Generalisierung bei Anwendung
- Definiere **Ähnlichkeitsmaß** auf Lernbeispielen
- **Beispiel**:
  - Arzt merkt sich Krankheitsbilder und Diagnosen
  - Bei neuem Patienten: Erinnert sich an „ähnliche“ Fälle
  - Stelle gleiche Diagnose für ähnliche Fälle
- **Problem**:
  - Definition des Ähnlichkeitsmaßes
  - Wie ähnlich ist ähnlich genug?: Wann ist der ähnlichste Fall so verschieden, dass alte Diagnose nicht passt?
- Verschiebe Generalisierung von der Lernphase in die Anwendungsphase! (***lazy learning*** vs. ***eager learning***)

Ähnlichkeit als Abstand im Merkmalsraum

Voraussetzung

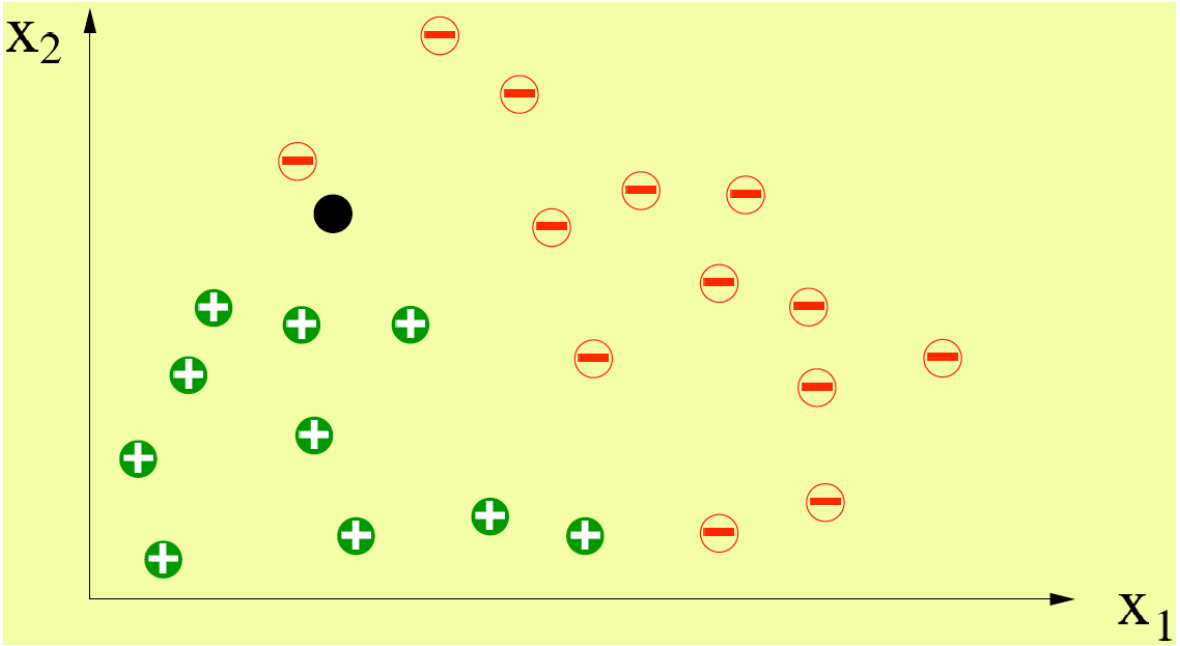
Auf Merkmalsraum $\langle x_1,\ldots,x_n\rangle$ ist Maß definiert

Zwei Datensätze sind umso ähnlicher, je kleiner ihr Abstand im Merkmalsraum ist

Beispiel

Falls Ähnlichkeitsmaß **Euklidischer Abstand**:

● ist ein ⊖, da ähnlichster (= euklidisch nächster) Nachbar auch ⊖ ist.



NEARESTNEIGHBOR

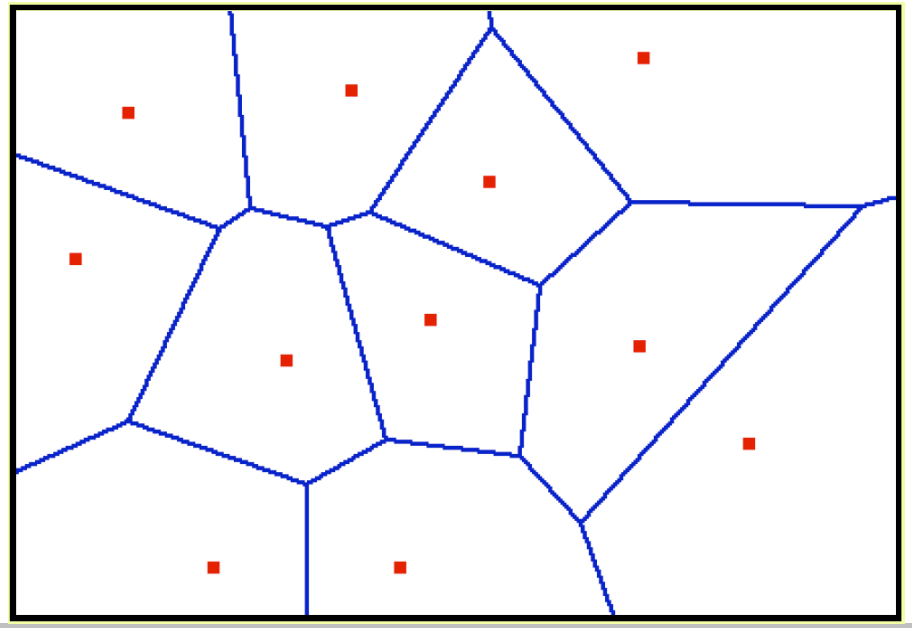
function NEARESTNEIGHBOR ($M+$,$M-$, s) **returns** + or –
inputs: $M+$, $M-$: positive bzw negative Trainingsbeispiele
s: zu klassifizierender Datensatz

$t \leftarrow \text{argmin}_{x \in M+ \cup M-}\{d(s,x)\}$
if $t \in M+$ **then return** +
else return –

- Funktioniert entsprechend auf mehreren diskreten Klassen
- Speicher: $O(|M|)$
- Zeit: $O(|M|) \cdot$ Laufzeit von d (cleverer: $O(\log |M|) \cdot \ldots$)

Separierung durch Nearest-Neighbor

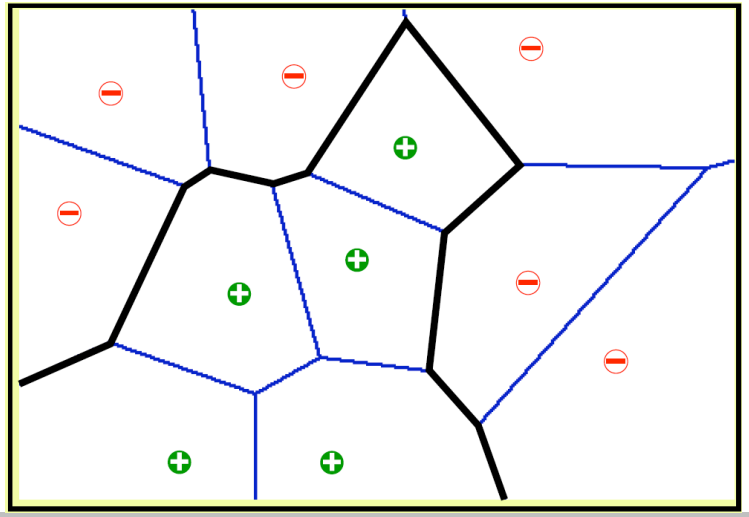
Voronoi-Linien einer Punktmenge im $\mathcal{R}^2$: Linien gleichen Abstands zu den beiden nächstgelegenen Nachbarpunkten.
Verallgemeinerbar auf $\mathcal{R}^n$



Nearest-Neighbor separiert Punktmengen im $\mathcal{R}^n$ entlang Voronoi-Linien.

Beispiel

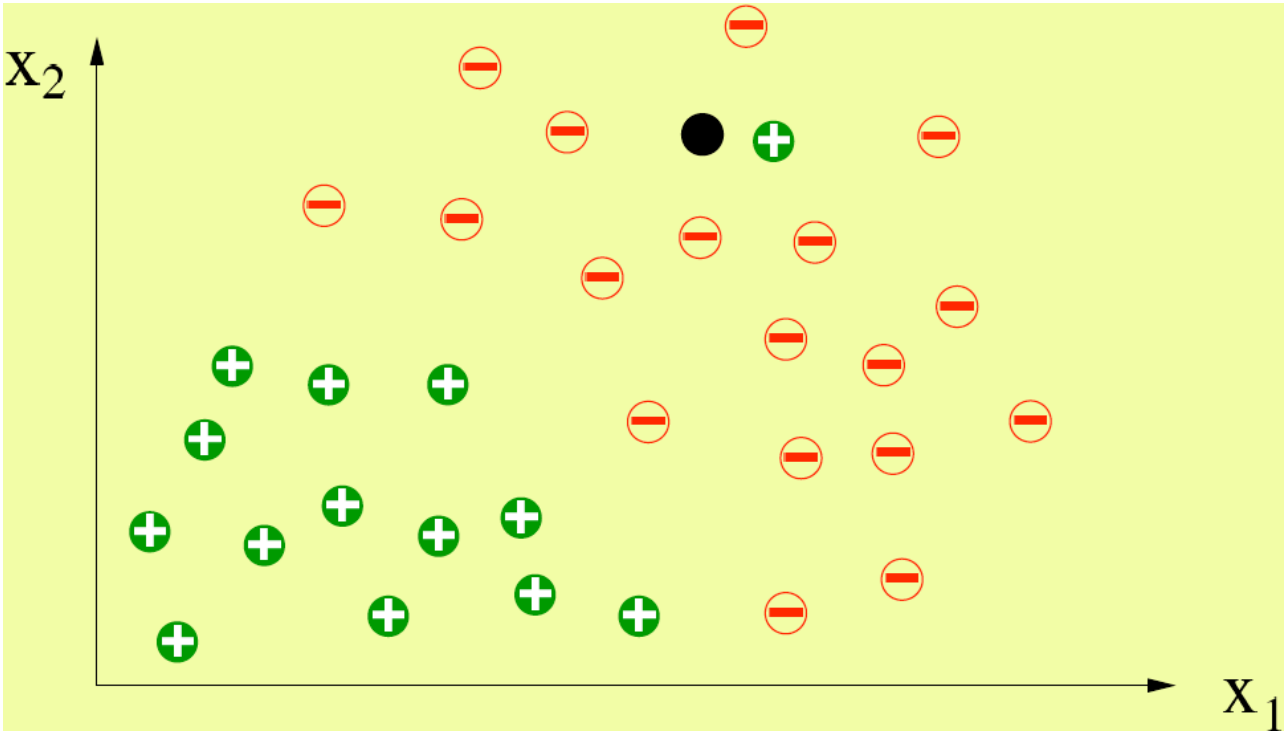
im $\mathcal{R}^2$ für die eingezeichnete Punkt-Klassifikation



Überanpassung (*overfitting*)

… ist potenziell ein Problem für alle Lernverfahren:

- Wenn Messfehler/„Ausreißer“ in der Trainingsmenge!
- Wüsste man die Messfehler, würde man sie löschen!



Beispiel

Punkt ● wird als ✚ klassifiziert. Aber ist das (intuitiv) richtig?

Robustheit durch größere Nachbarschaft

function k-NEARESTNEIGHBOR ($M+$,$M-$, s) **returns** + or –
inputs: $M+$, $M-$: positive bzw negative Trainingsbeispiele
s: zu klassifizierender Datensatz

$V \leftarrow$ die k nächsten Nachbarn von s in $M+ \cup M-$
if $|M+ \cap V| > |M- \cap V|$ **then** **return** +
else if $|M+ \cap V| < |M- \cap V|$ **then** **return** –
else **return** Random(+,–)

- Klassifiziert nach „Mehrheitsmeinung“ der k Nachbarn (sinnvoll bis $k \approx 10$)
- Entsprechend auf mehreren diskreten Klassen
- Selber Aufwand wie „einfacher“ Nearest-Neighbor:
  - Speicher: $O(|M|)$
  - Zeit: $O(\log |M|) \cdot$ Laufzeit von d

Gewichteter k-Nearest-Neighbor

Statt einfacher Mehrheitsmeinung unter den k Nachbarn gewichte ihre „Stimmen“ durch Abstand zum zu klassifizierenden Datensatz, also mit dem Gewicht:

$$w_i = \frac{1}{d(\mathbf{s}, \mathbf{x_i})^2}$$

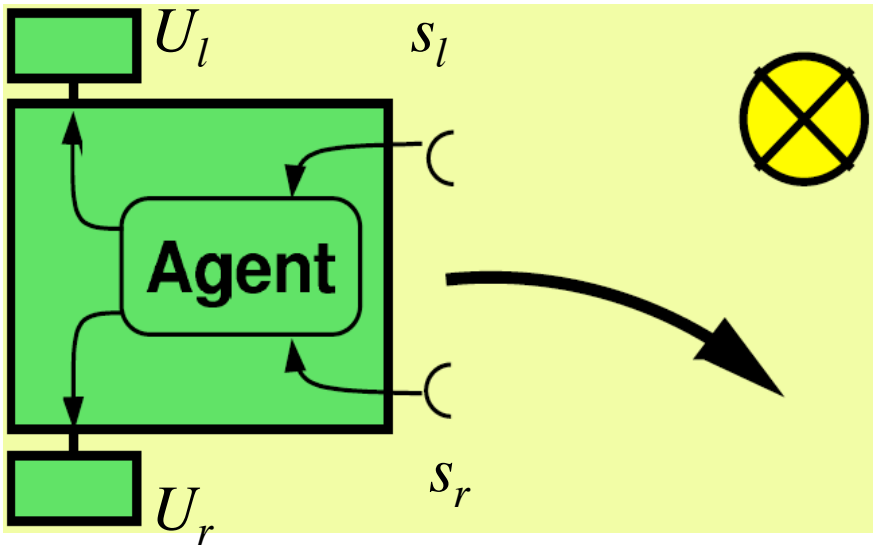
Hier quadratischer Abstand – kann man auch anders machen.

Von Klassifikation zu Regression 1/3

Folie 233: Klassifikation mit kontinuierlichen „Klassen“ heißt **Regression**

Beispiel

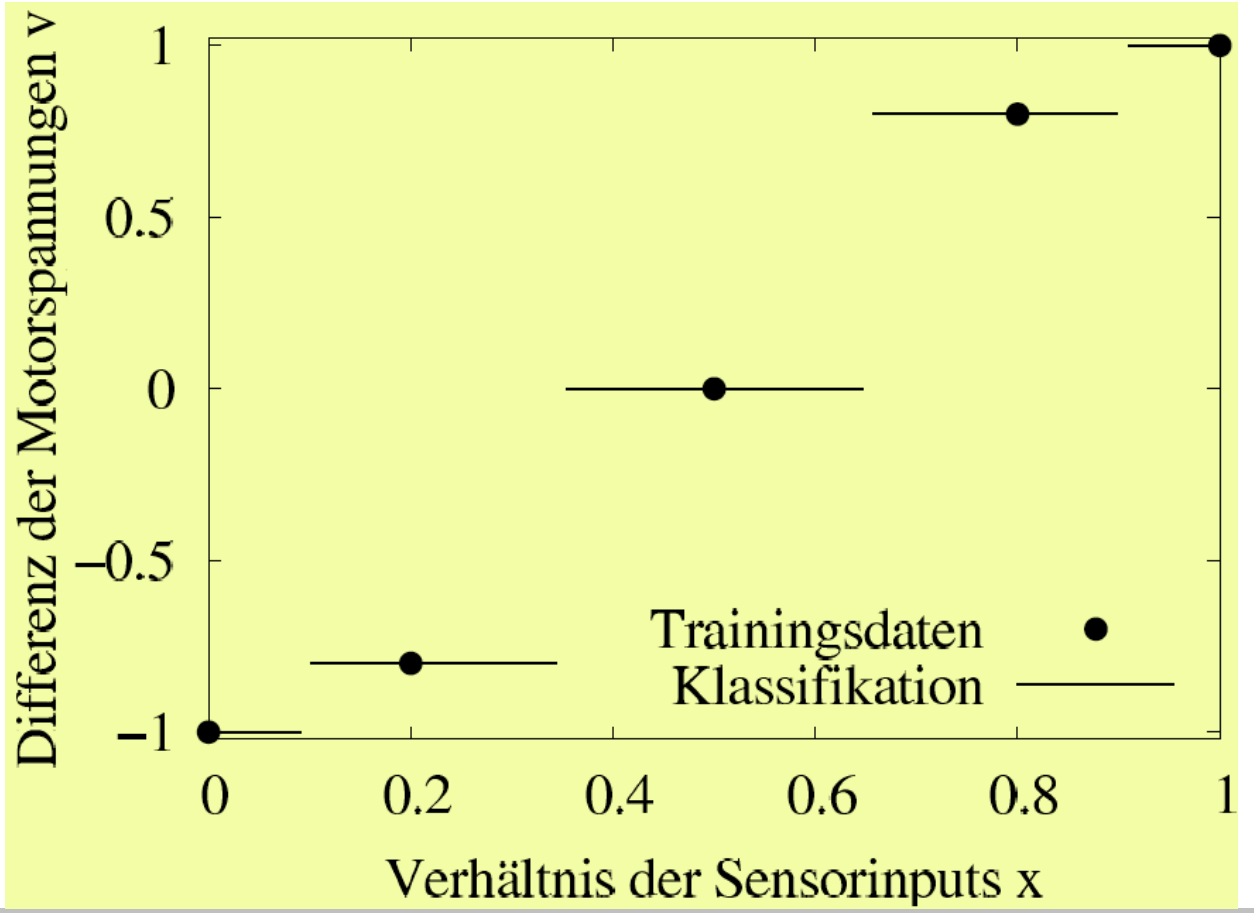
Roboter soll lernen, einer Lichtquelle auszuweichen



- Richtung zum Licht miss durch Verhältnis der Helligkeit an zwei Sensoren s_l, s_r
- Motorspannung U_l, U_r setze je proportional zu s_l, s_r
- Fahrtrichtung des Roboters ergibt sich daraus als Differenz der beiden Motorspannungen $U_r – U_l$
  - Differenz negativ: Rechtskurve; positiv: Linkskurve

Von Klassifikation zu Regression 2/3

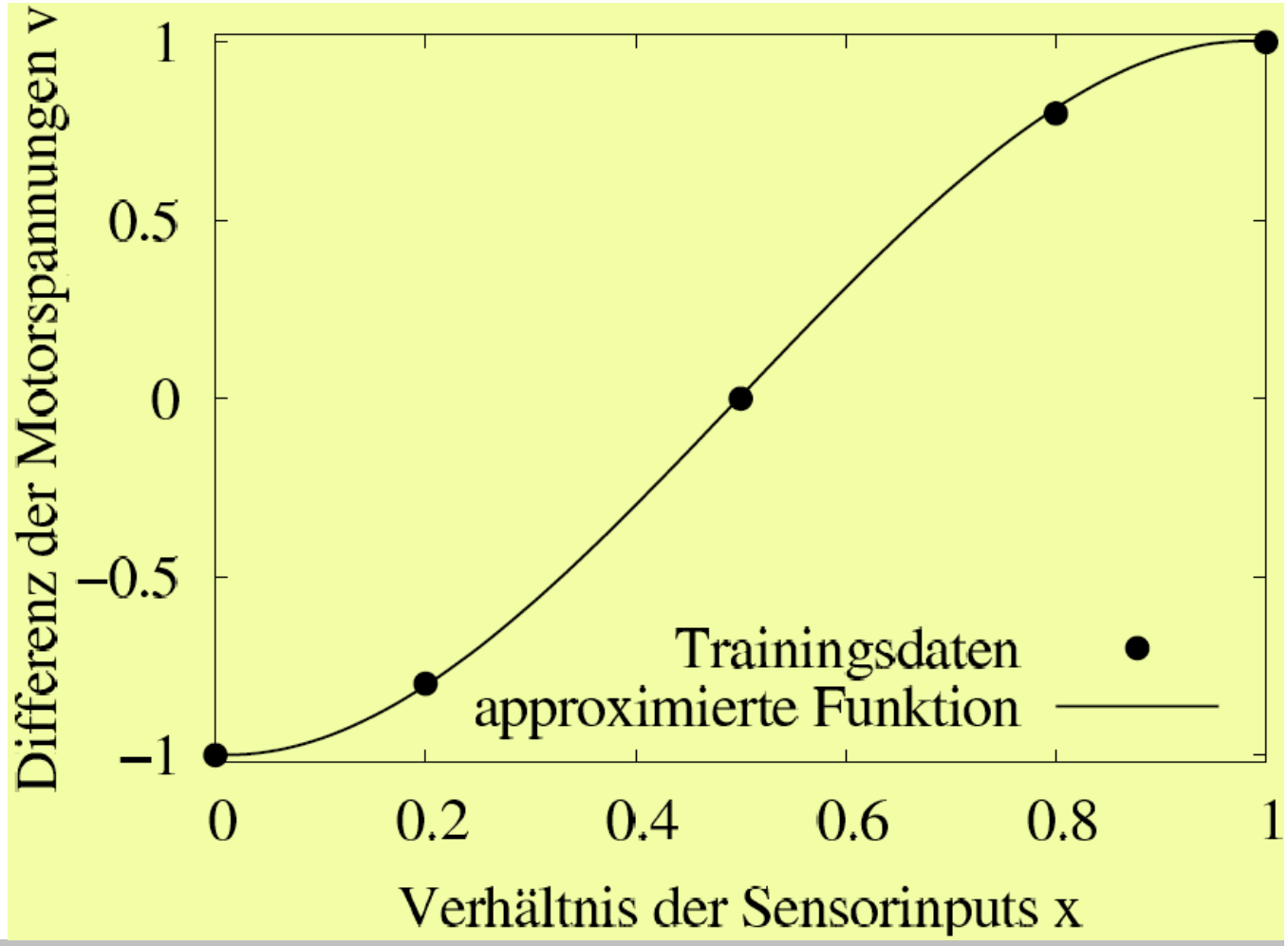
Liegen Trainingsdaten für die Verhältnisse 0, 0.2, 0.5, 0.8, 1 vor, lernt Nearest-Neighbor die folgende Abbildung:



Von Klassifikation zu Regression 3/3

Abhilfe: Regression durch Mittelwert der k nächsten Nachbarn (möglicherweise gewichtet):

$$\bar{f}(\mathbf{s}) = \frac{1}{k}\sum_{i=1}^{k} f(\mathbf{x}_i)$$



Ausblick: Fallbasiertes Schließen 1/2

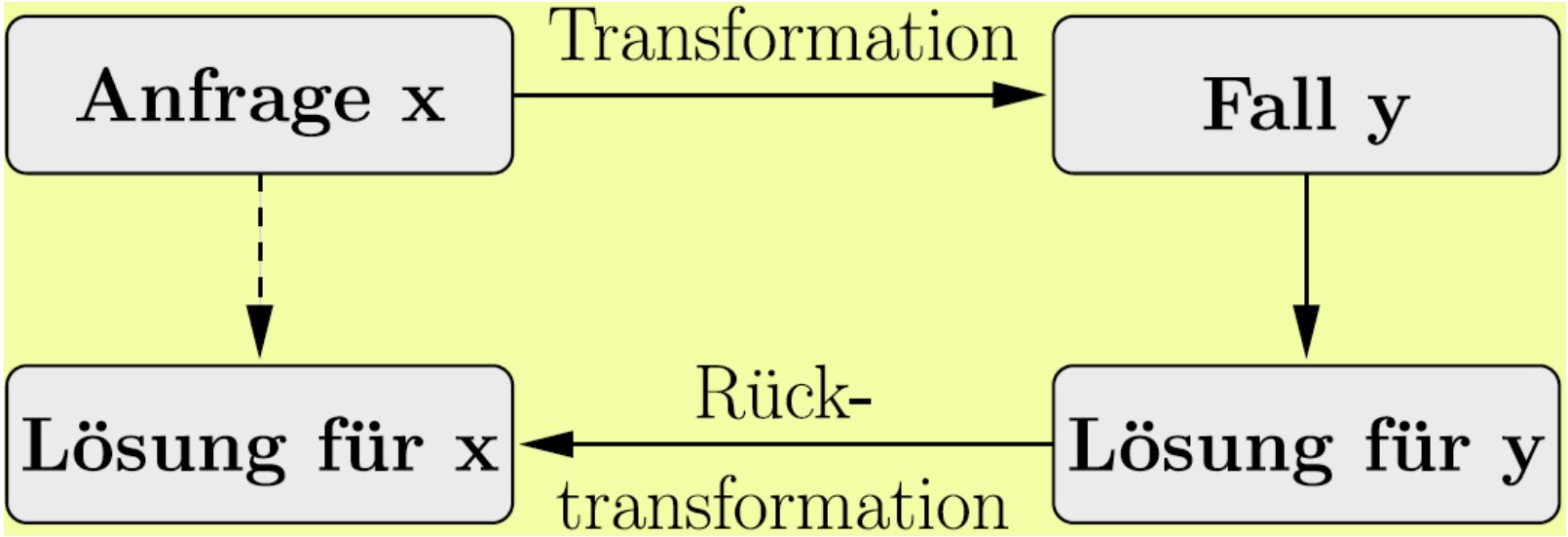
case-based reasoning, CBR

- Übertragung von Nearest-Neighbor auf Problemlösung textuell/symbolisch gegebener Probleme
- Einsätze z.B. Diagnosen, Telefon-Hotlines

Beispiel:

Merkmal	Anfrage	Fall aus Fallbasis
Defektes Teil:	Rücklicht	Vorderlicht
Fahrrad Modell:	Marin Pine Mountain	VSF T400
Baujahr:	1993	2001
Stromquelle:	Batterie	Dynamo
Zustand der Birnen:	ok	ok
Lichtkabelzustand:	?	ok
	Lösung	
Diagnose:	?	Massekontakt vorne fehlt
Reparatur:	?	Stelle Massekontakt vorne her

Fallbasiertes Schließen 2/2



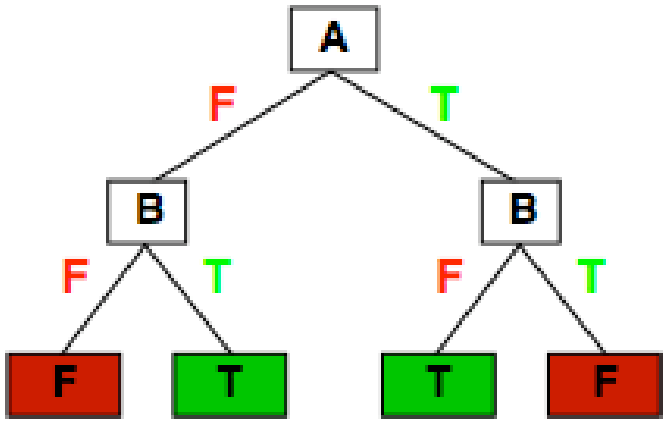
Probleme

- Ähnlichkeitsmaß auf Fällen (wie bei Nearest-Neighbor, nur intuitiv noch schwieriger)
- Definition der Rücktransformation

Entscheidungsbäume

… repräsentieren diskrete endliche Fkt.en über Attributen.
Für Boolesche Funktionen: Syntaktische Variante von Wahrheitstabellen

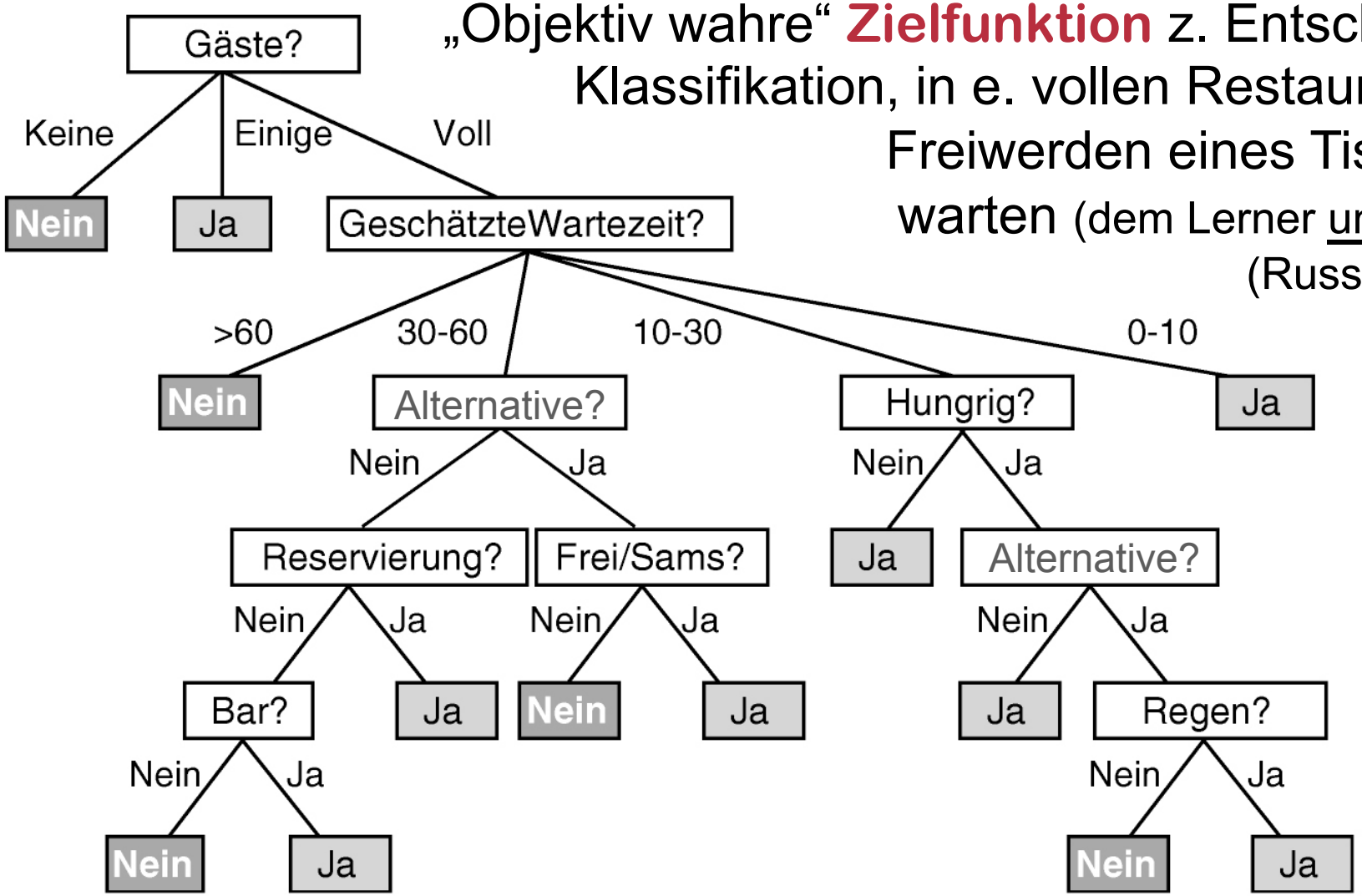
A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



- Für jede konsistente Menge von deterministischen Lernbeispielen gibt es mindestens einen Entscheidungsbaum
- Dem Lernsystem wird eine kleine Menge (verglichen mit allen Möglichkeiten) von Lernbeispielen gegeben
- Gesucht ist ein möglichst „kompakter“ Entscheidungsbaum (Analogie zu „niedrigem“ Polynom für numerische Funktion)

Beispiel: Warten als Entscheidungsbaum

„Objektiv wahre“ **Zielfunktion** z. Entscheidung/ Klassifikation, in e. vollen Restaurant aufs Freiwerden eines Tisches zu warten (dem Lerner unbekannt!) (Russell/Norvig)



Die Lernbeispiele

Bei-spiel	Attribute										Ziel
	Alt	Bar	Frei	Hung	Gäste	Preis	Regen	Reser.	Typ	Wart	Warten?
X_1	*Ja*	*Nein*	*Nein*	*Ja*	*Einige*	€€€	*Nein*	*Ja*	*Franz.*	0-10	*Ja*
X_2	*Ja*	*Nein*	*Nein*	*Ja*	*Voll*	€	*Nein*	*Nein*	*Thai*	30-60	*Nein*
X_3	*Nein*	*Ja*	*Nein*	*Nein*	*Einige*	€	*Nein*	*Nein*	*Burger*	0-10	*Ja*
X_4	*Ja*	*Nein*	*Ja*	*Ja*	*Voll*	€	*Ja*	*Nein*	*Thai*	10-30	*Ja*
X_5	*Ja*	*Nein*	*Ja*	*Nein*	*Voll*	€€€	*Nein*	*Ja*	*Franz.*	>60	*Nein*
X_6	*Nein*	*Ja*	*Nein*	*Ja*	*Einige*	€€	*Ja*	*Ja*	*Ital.*	0-10	*Ja*
X_7	*Nein*	*Ja*	*Nein*	*Nein*	*Keine*	€	*Ja*	*Nein*	*Burger*	0-10	*Nein*
X_8	*Nein*	*Nein*	*Nein*	*Ja*	*Einige*	€€	*Ja*	*Ja*	*Thai*	0-10	*Ja*
X_9	*Nein*	*Ja*	*Ja*	*Nein*	*Voll*	€	*Ja*	*Nein*	*Burger*	>60	*Nein*
X_{10}	*Ja*	*Ja*	*Ja*	*Ja*	*Voll*	€€€	*Nein*	*Ja*	*Ital.*	10-30	*Nein*
X_{11}	*Nein*	*Nein*	*Nein*	*Nein*	*Keine*	€	*Nein*	*Nein*	*Thai*	0-10	*Nein*
X_{12}	*Ja*	*Ja*	*Ja*	*Ja*	*Voll*	€	*Nein*	*Nein*	*Burger*	30-60	*Ja*

Die „wahre“ Zielfunktion hängt nicht notwendig von allen bekannten Attributen ab!

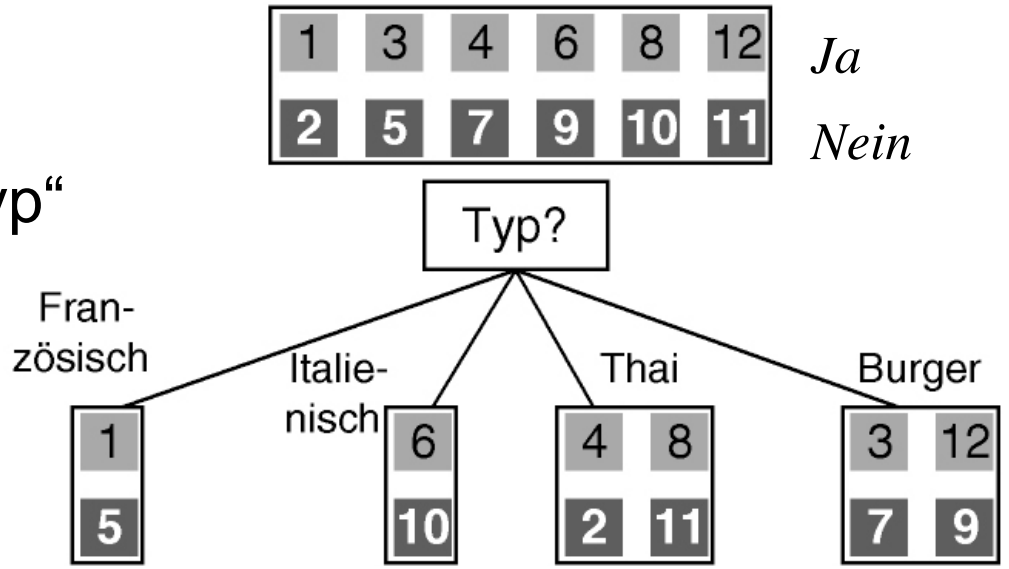
Lernen von Entscheidungsbäumen

- Wähle Attribut a, das die Lernbeispiele „gut“ separiert
- Für die n Werte von a generiere n rekursive Aufrufe des Lerners jeweils mit den Lernbeispielen, wo a Wert v_i hat
- Bau die Ergebnisse zu einem Baum in Wurzel a mit den n Ästen zusammen

(Gegen-)Beispiel

Separierung nach Attribut „Typ“

(schlechte Separierung, da die Unter-Lernprobleme keine Separierung in „Ja“ und „Nein“ erreichen)



Decision Tree Learning (DTL)

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return MODE(examples)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value v_i of best do
            examples_i ← {elements of examples with best = v_i}
            subtree ← DTL(examples_i, attributes − best, MODE(examples_i))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

wie im R/N:

MAJORITY-VALUE
MOST-COMMON-VALUE

MAJORITY-VALUE (abs. Mehrheit) bei binärer Entscheidung!

Eigenschaften von DTL

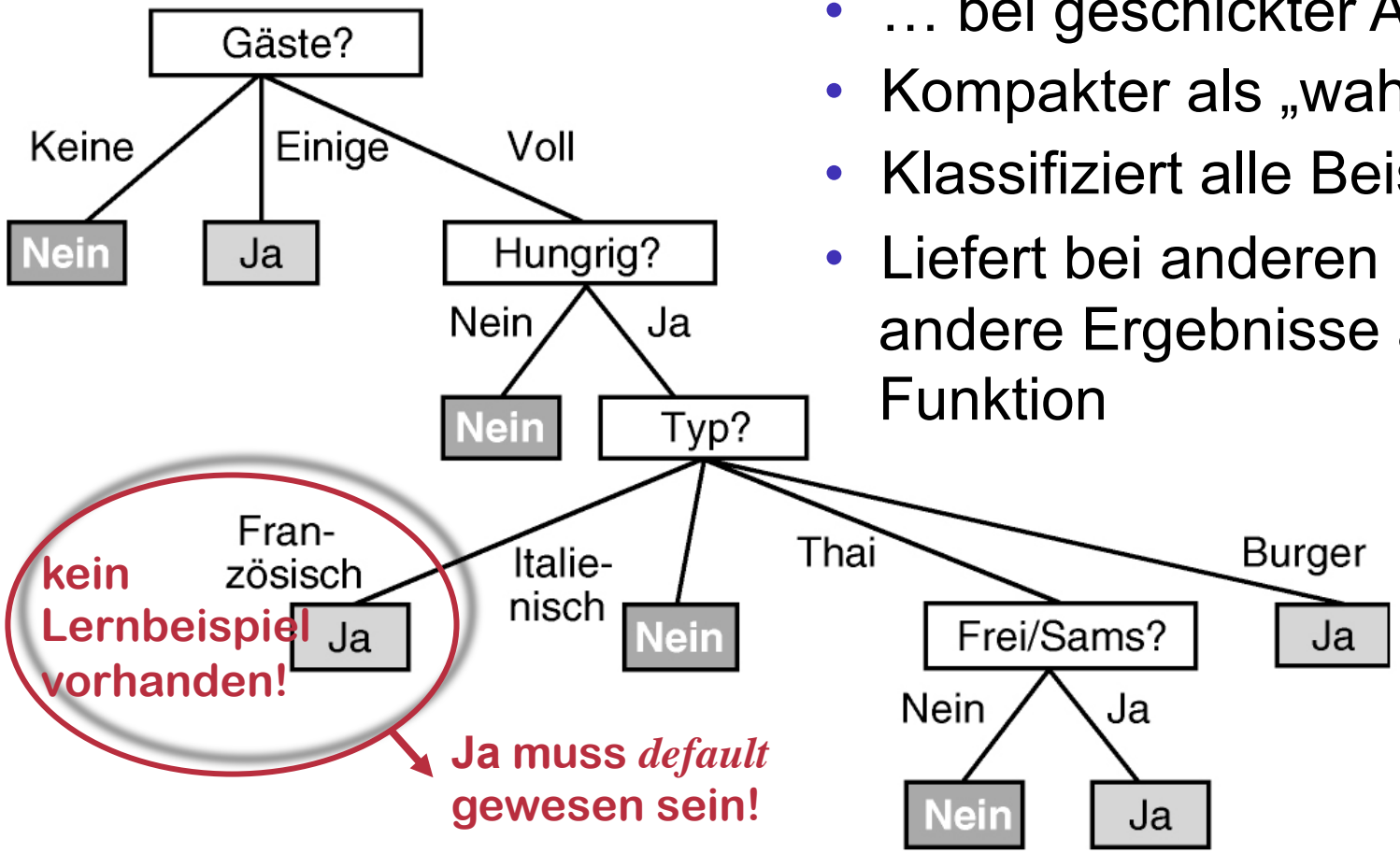
Für

- E Trainingsbeispiele,
- A Attribute,
- W Maximalzahl unterschiedlicher Werte eines Attributs

ergibt sich:

- Speicher: $O(W^A)$
  (praktisch deutlich weniger, da bei guter Attributauswahl nur ganz wenige Bereiche des E-Baums tief sind!)
- Zeit: $O(E \cdot A \cdot W)$
- Gelernter E-Baum nicht notwendig „korrekt“ bzw. „optimal“ (bei Lernproblemen problematische Begriffe!)

Gelernter Entscheidungsbaum



- … bei geschickter Attributwahl
- Kompakter als „wahre“ Funktion
- Klassifiziert alle Beispiele korrekt
- Liefert bei anderen Eingaben z.T. andere Ergebnisse als „wahre“ Funktion

Bleibt die **Frage**:
Wie wählt man Attribute geschickt, damit Baum kompakt wird?