Gliederung

1. KI im Allgemeinen und in dieser Vorlesung
2. Heuristische Suche
3. Logik und Inf
4. Wissensrepräs
5. Handlungspl
6. **Lernen**
7. Sprachverarbeitung
8. Umgebungswahrnehmung

1. **Überblick**
2. **Lernen von Entscheidungsbäumen**
3. **Induktives Logisches Programmieren**
4. **Reinforcement-Lernen**

Was ist Lernen?

Zusatzquellen: M. Riedmiller, Vorlesung KI, U. Dortmund
K. Morik, Vorlesung Masch. Lernen, U. Dortmund

„… changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently next time.“

Herbert Simon

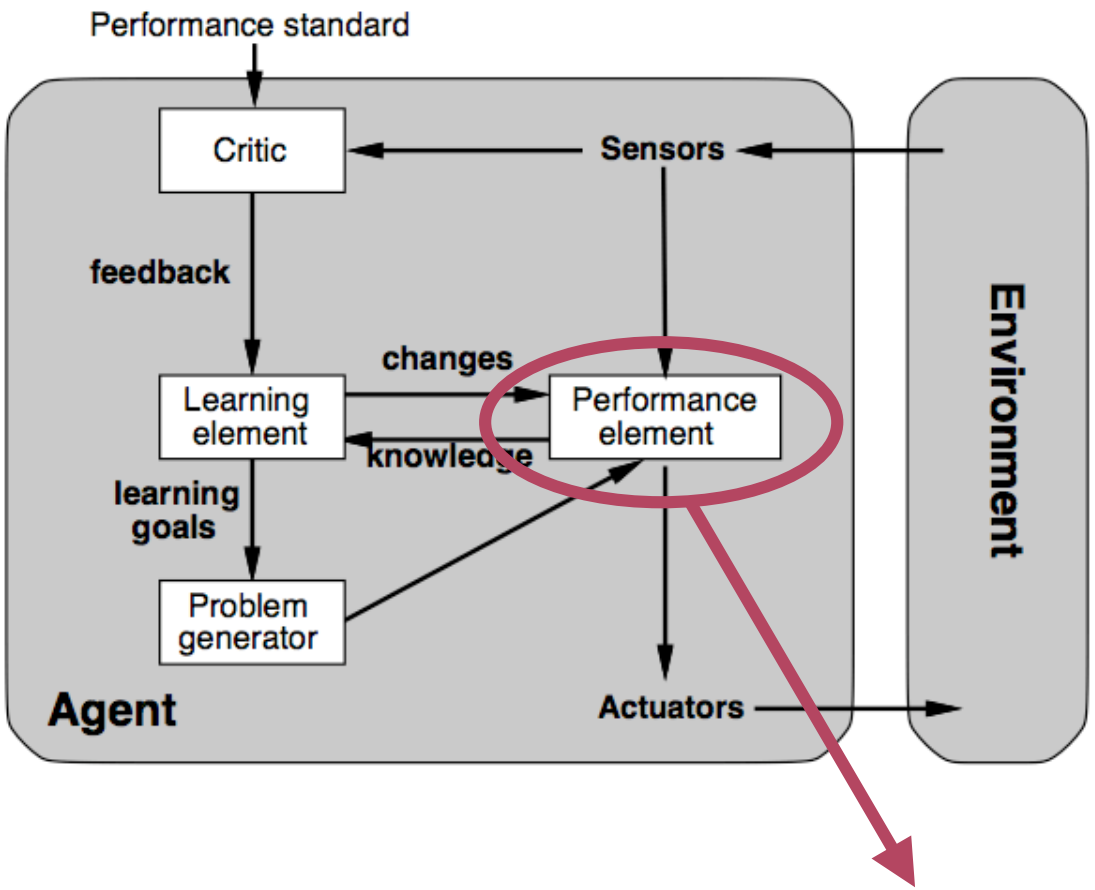
… zwar sind darin viele Punkte unklar oder speziell:

- wer macht die „changes“, darf das ein Programmierer sein?
- was sind „changes“? wieso „in“ the system? was zählt zu „the system“?
- was ist „the same task“? was ist „the population“?
- was heißt „more efficiently“?
- warum eigentlich erst „next time“, darf man nicht fürs „first time“ lernen?

… doch eine wirklich bessere Definition gibt es auch nicht!

↪ Arbeiten wir mit dem Alltagsverständnis des Begriffs!

Programmschema: Lernender Agent (s. Folie 26)



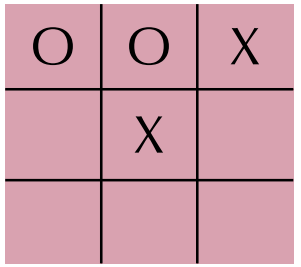
Eines der vorgenannten Agentenprogramme

- Was ist das Performance element? (A*, POP, TBox, …)
- Welche Komponente darin wird verändert? ($h(n)$, Operatormenge, Regel, …)
- Wie ist sie repräsentiert? (Funktion, ADL-Operator, Logik-Formel, …)
- Was ist das Feedback, wie messen wir Verbesserung? (Suchzeit, Zielerreichung, Repräsentationsgröße, …)

Varianten von Lernen

- **Überwachtes Lernen**: Gegeben Paare ⟨***In***, *Erwartet_Out*⟩, leite Funktion *f* ab, sodass für *neue* Eingaben gilt *f*(***In'***)=*Erwartet_Out'*
  Beispiel: Lerne Vorhersage für Wechselkurs $~€, gegeben Wirtschaftsdaten und Wechselkurse aus der Vergangenheit
- **Unüberwachtes Lernen**: Gegeben Merkmalvektoren ⟨***In***⟩, leite Funktion *f* ab, die Regularitäten/Einteilungen beschreibt
  Beispiel: Finde potenziell „interessante" Muster in riesengroßen Datenbeständen (↪ **Data Mining**), z.B. in Kundendaten von PayBack
- **Reinforcement-Lernen**: Gegeben Tripel ⟨***In***,***Out***,*Reward*⟩, modifiziere das Programm, sodass in Zukunft *Reward* durch *f*(***In***) optimiert wird (Jargon: *Reward* ↪ **Reinforcement**-Signal)
  Beispiel: Lerne die Koordination der Motorsignale einer sechsbeinigen Laufmaschine für eine „glatte", effiziente Schreitbewegung in Richtung eines Zielpunkts (Reward z.B.: 1/ (Integral der Rollwinkel + Entfernung zum Ziel))

Induktives Lernen

Beispiel Aus Menge von **Lernbeispielen** ⟨  , (0,0) ⟩

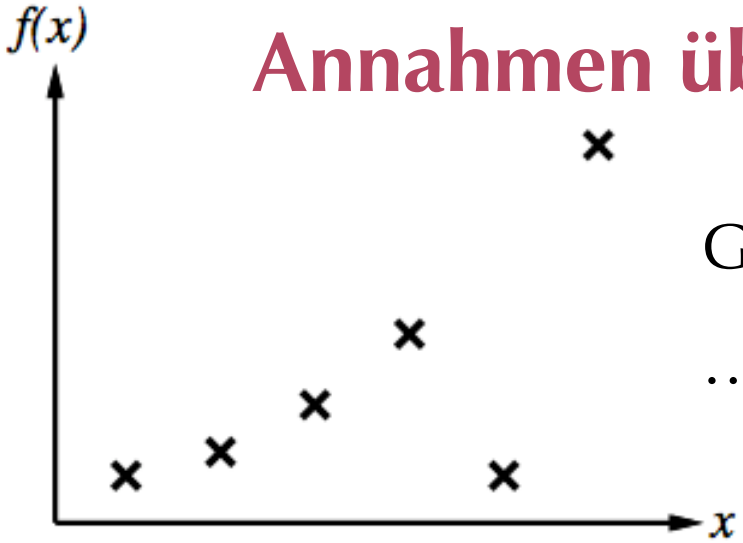
leite Funktion h (Hypothese) ab, die Zielfunktion f approximiert!
↪ **Induktives** statt deduktivem Schließen

Diskrete Fkt.: **Klassifikation**; kontinuierliche Fkt.: **Regression**

Vereinfachende Annahmen

- Genau diese Funktion ist zu lernen
- Lernbeispiele sind vorgegeben, fehlerfrei
- Kein Vorwissen ist zu berücksichtigen („*tabula rasa*“)
- Umgebung ist deterministisch und beobachtbar

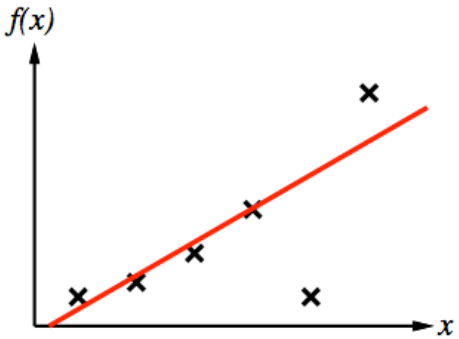
Annahmen über die Zielfunktion



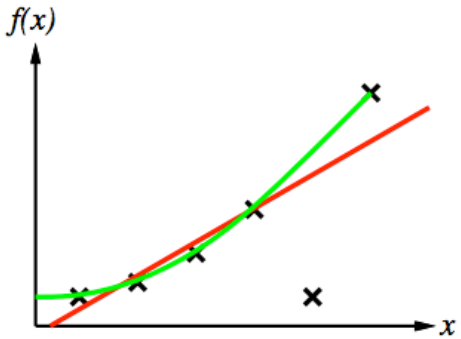
Gegeben Lernbeispiele …

… ist die Hypothese …

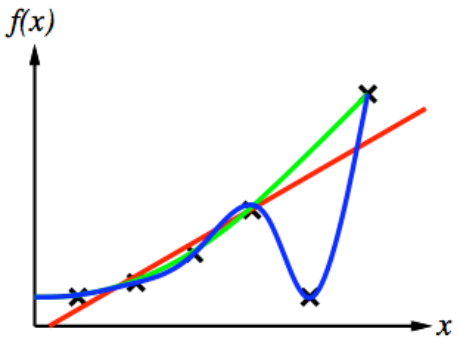
… linear?



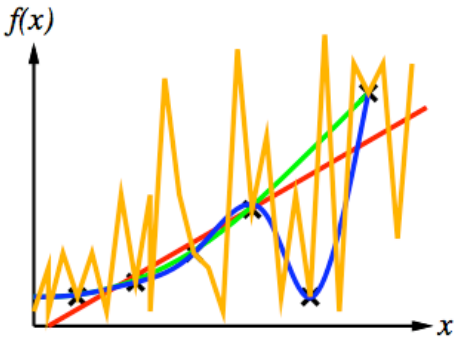
… quadratisch?



… kubisch?



… ??? ?



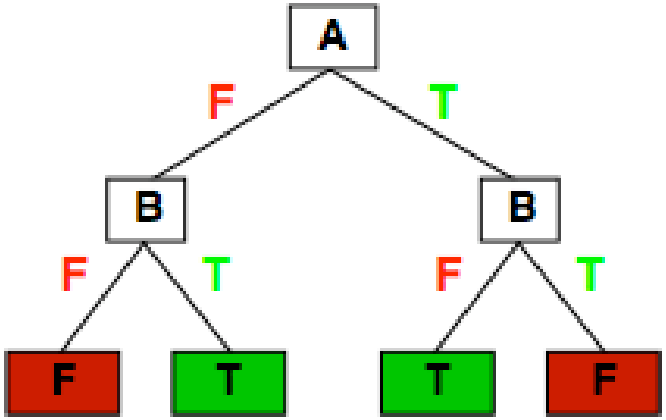
Bekanntlich definieren n Werte ein Polynom n-ten Grades, doch führt das automatisch zur besten Zielfunktion?

Entscheidungsbäume

… repräsentieren diskrete endliche Funktionen über Attributen.

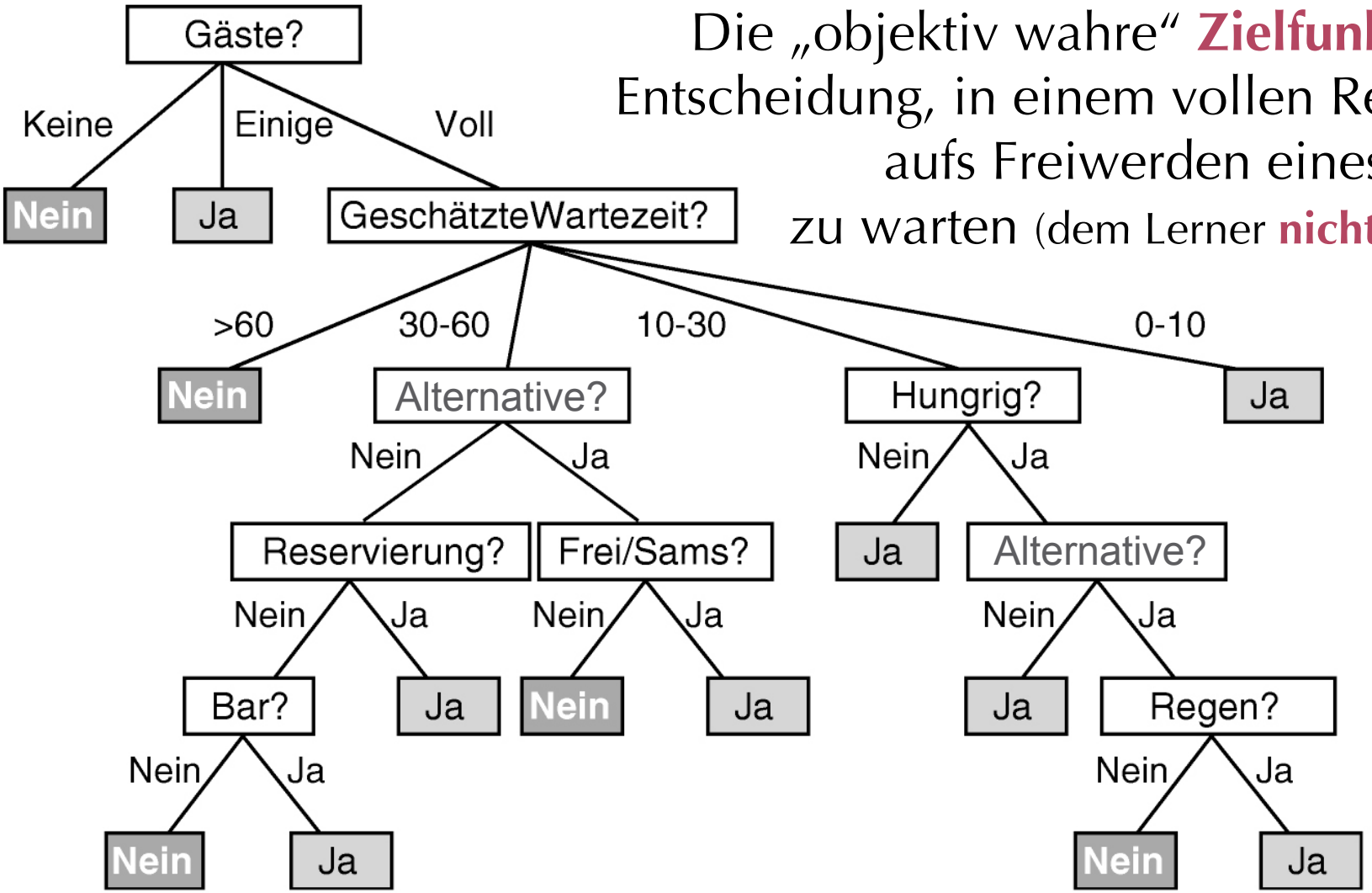
Für Boolesche Funktionen: Syntaktische Variante von Wahrheitstabellen

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 Polynom zur Beschreibung von numerischer Funktion)

Beispiel: Warten als Entscheidungsbaum



Die „objektiv wahre“ **Zielfunktion** zur Entscheidung, in einem vollen Restaurant aufs Freiwerden eines Tisches zu warten (dem Lerner **nicht** bekannt!)

Die Lernbeispiele

Bei-spiel	Attribute										Ziel
	Alt	Bar	Frei	Hung	Gäste	Preis	Regen	Reser.	Typ	Wart	Werden-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*

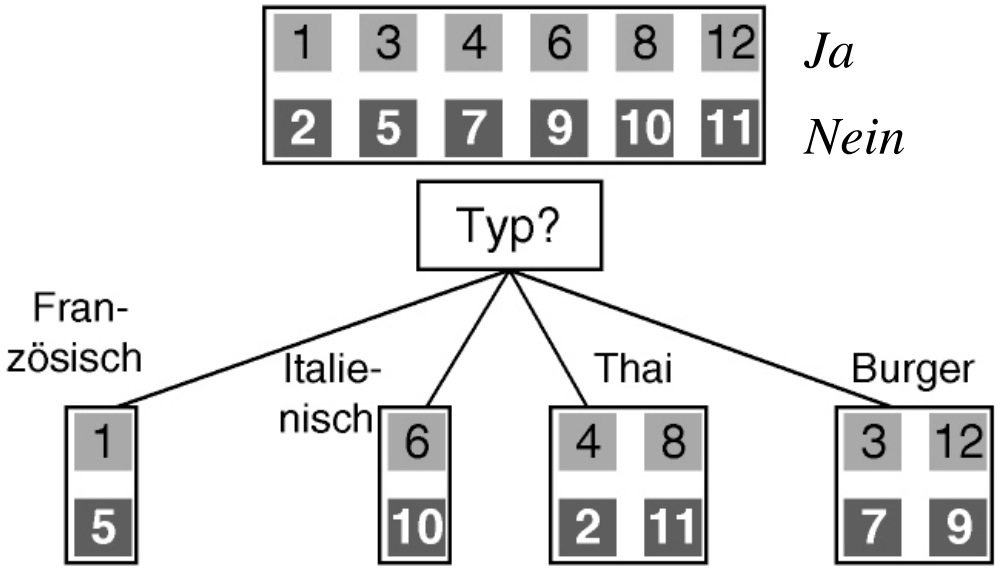
die „wahre“ Zielfunktion hängt nicht von allen bekannten Attributen ab!

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

Beispiel

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



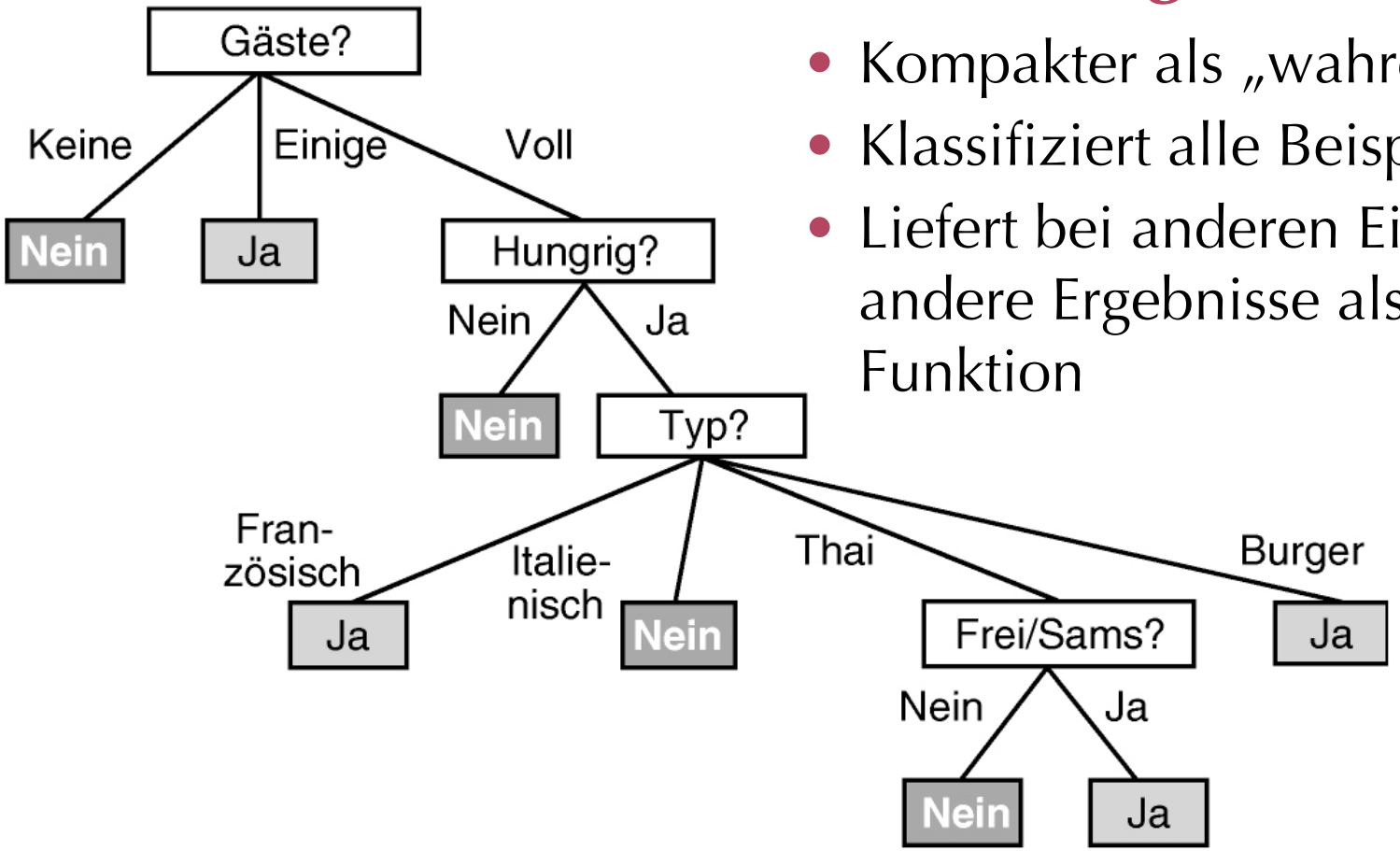
Decision Tree Learning (DTL)

```
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))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

wie im Buch:
MAJORITY-VALUE
MOST-COMMON-VALUE

MAJORITY-VALUE nur sinnvoll bei binärer Entscheidung!

Gelernter Entscheidungsbaum

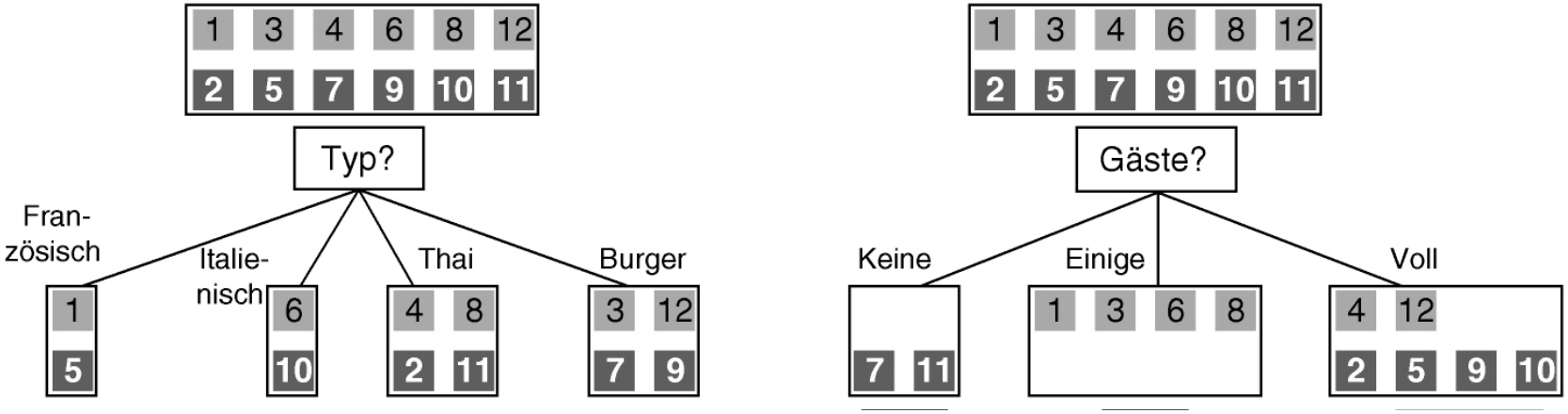


- Kompakter als „wahre“ Funktion
- Klassifiziert alle Beispiele korrekt
- Liefert bei anderen Eingaben z.T. andere Ergebnisse als „wahre“ Funktion

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

Information

Grundidee: Wähle das Attribut, das die danach erforderliche **Information** minimiert!



Für Typ? noch 12, für Gäste? nur noch 6 Lernbeispiele „offen“

Informationsmaß (Shannon/Weaver, 1949):

1 **bit** ist das Maß an Information, das dem Ergebnis eines binären Zufallsexperiments entspricht (W‘vert. ⟨0,5, 0,5⟩, Lott. [0,5, X; 0,5, Y])

Informationsgehalt

Gegeben eine W'verteilung $\mathbf{P}(V)=\langle P(v_1),\ldots, P(v_n)\rangle$.

Informationsgehalt (auch: **Entropie**) $I(\mathbf{P}(V)) := \sum_{i=1}^{n} -P(v_i)\log_2 P(v_i)$

Beispiel: Information aus einem Münzwurf:

fair: $I(\langle 0.5, 0.5\rangle) = -0.5\log_2(1/2) - 0.5\log_2(1/2) = 1$ [bit]

unfair: $I(\langle 0.25, 0.75\rangle) = -0.25\log_2(1/4) - 0.75\log_2(0.75)$

$\approx 0.5 + 0.31 \approx 0.81$ [bit]

Für Entscheidungsbaumlernen (binär):

p Beispiele „Ja", n Beispiele „Nein", wir brauchen also

$I(\langle p/(p+n), n/(p+n)\rangle) = -p/(p+n)\log_2(p/(p+n)) - n/(p+n)\log_2(n/(p+n))$ bit!

Restaurantbeispiel: $p=n=6$, also $I(\langle 0.5, 0.5\rangle)=1$ bit

Jedes Attribut ergibt maximal 1 bit Information!

Maximal informative Attributauswahl

Attribut A mit v Werten zerlegt die Lernbeispiele in $E_1, \ldots, E_v$, jedes E_i hat p_i positive und n_i negative Lernbeispiele

Beliebig gewähltes Lernbeispiel trägt i-ten Wert von A mit W'keit $(p_i+n_i)/(p+n)$

Erwarteter Rest von erforderlicher Information nach A :

$$Remainder(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} I\left(\left\langle \frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i} \right\rangle\right)$$

Informationsgewinn durch Auswahl von A:

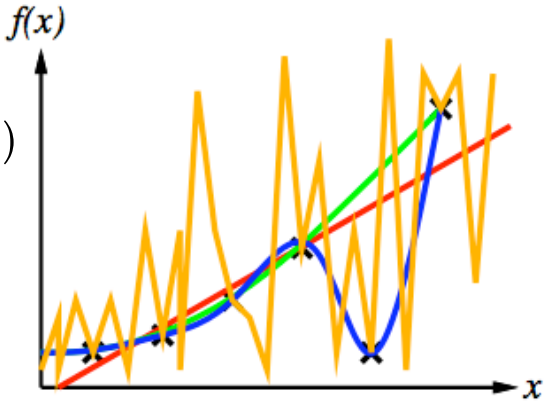
$$Gain(A) = I(\langle p/(p+n), n/(p+n)\rangle) - Remainder(A)$$

$$Gain(\text{Gäste?}) = 1 - [2/12 I(0,1) + 4/12 I(1,0) + 6/12 I(2/6, 4/6)] \approx 0.541$$

$$Gain(\text{Typ?}) = 1 - [2 \text{ x } 2/12\ I(1/2,1/2) + 2 \text{ x } 4/12\ I(2/4,2/4)\ ] = 0$$

Probleme und Weiterführendes

- Wieviele Lernbeispiele nimmt man?
- Welche Lernbeispiele nimmt man?
- Welche Attribute gibt es? (bei versteckten nie 100% Erfolg!)
- Welche/wieviele Messfehler sind in den Daten?
- Wie vermeidet man *overfitting*, also unnötig spezielle Lernfunktionen?
- Welche Einschränkungen der Lernfunktion macht man?
  (***Ockham's razor***: wähle *einfachste* konsistente Hypothese!
  Aber was ist die *einfachste*?)
  - linear: einfach, aber hohe Fehlersumme
  - n-te Ordnung: komplizierter, aber passt (*overfitting*?)
  - nichtlinear: kompliziert, aber passt immer (*overf.*?)

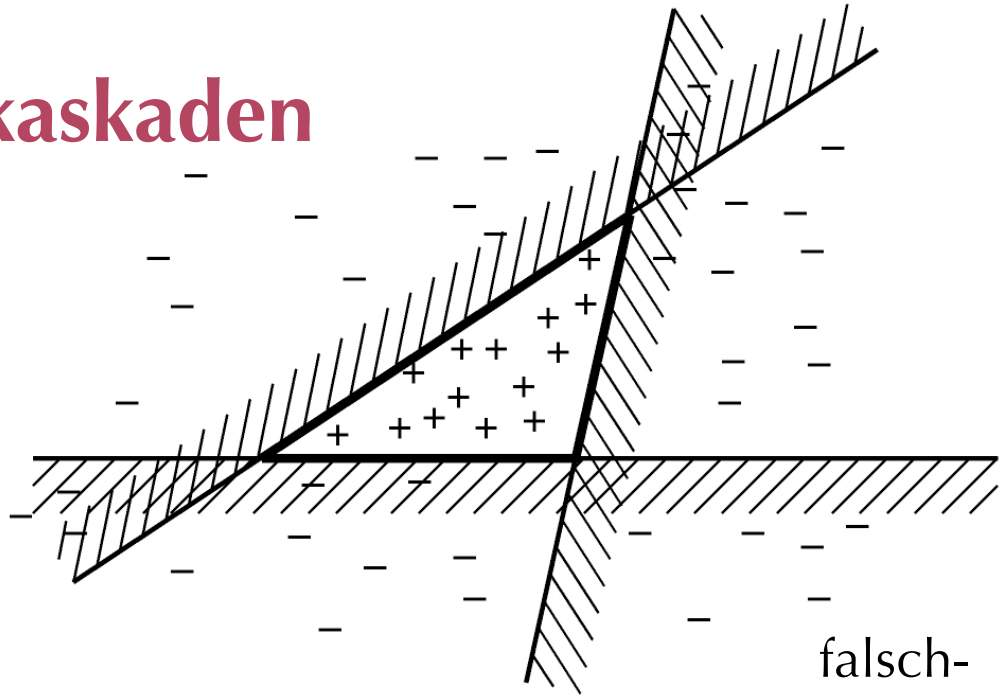


Mehr dazu Russell/Norvig Kap. 18!

Lernkaskaden

Grundidee:

- Lerne viele schnelle (lineare) Klassifikatoren, die alle (bei w'keitsbasierten Lernern: $P \approx 1$) positiven Lernbeispiele enthalten, und möglicherweise viele (aber $P << 1$) negative („falsch-positive").
- „Kaskadiere" diese: Ein Testbeispiel wird positiv klassifiziert, wenn die *gewichtete Mehrheit* der Einzelklassifikatoren es akzeptieren



falsch-pos.

Gäste ≠ Keine	7, 11
Wart ≠ >60	5, 9
Typ≠Thai ∧ Wart≠30-60	2
Typ≠Ital ∧ Preis≠€€€	10

AdaBoost (*Adaptive Boosting*)

AdaBoost ist ein **Meta-Lern-algorithmus**

function ADABOOST($examples$, L, M) **returns** a weighted-majority hypothesis
  inputs: $examples$, set of N labelled examples $(x_1, y_1), \ldots, (x_N, y_N)$
    L, a learning algorithm
    M, the number of hypotheses in the ensemble
  local variables: **w**, a vector of N example weights, initially $1/N$
    h, a vector of M hypotheses
    z, a vector of M hypothesis weights

  for $m = 1$ **to** M **do**
    $\mathbf{h}[m] \leftarrow L(examples, \mathbf{w})$
    $error \leftarrow 0$
    for $j = 1$ **to** N **do**
      if $\mathbf{h}[m](x_j) \neq y_j$ **then** $error \leftarrow error + \mathbf{w}[j]$
    for $j = 1$ **to** N **do**
      if $\mathbf{h}[m](x_j) = y_j$ **then** $\mathbf{w}[j] \leftarrow \mathbf{w}[j] \cdot error/(1 - error)$
    $\mathbf{w} \leftarrow$ NORMALIZE(**w**)
    $\mathbf{z}[m] \leftarrow \log(1 - error)/error$
  return WEIGHTED-MAJORITY(**h**, **z**)

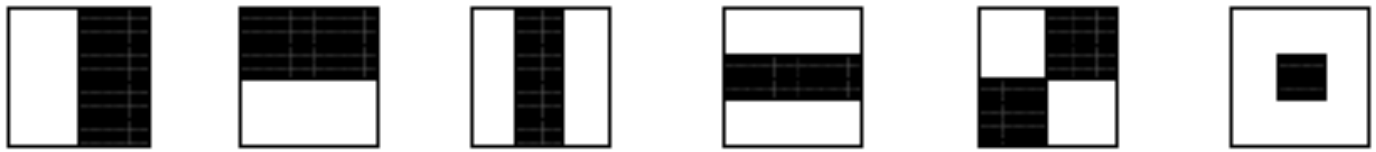
[Nüchter et al, IAS-2004]

Beispiel: Stühle in Laserdaten erkennen

Lernbeispiele: positiv

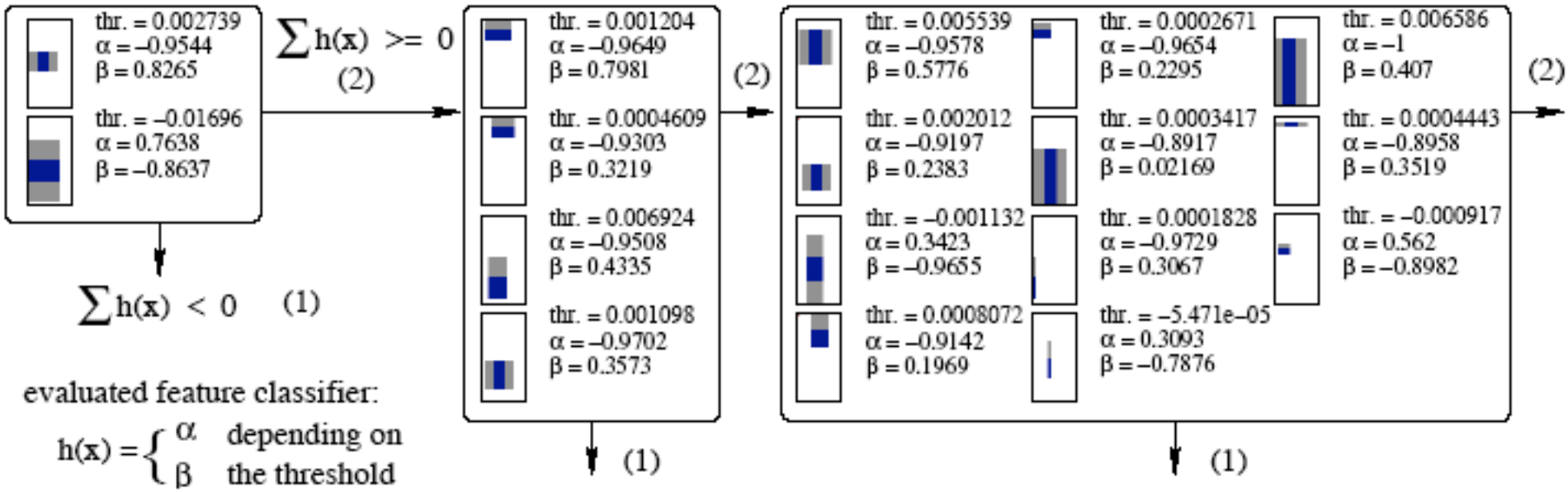
Lernbeispiel: negativ

Die Merkmale: **Haar-Filter** in selektierten Bildregionen



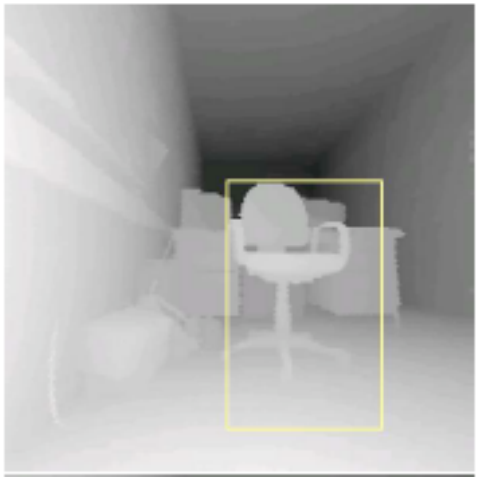
Beispiel: Stühle, Kaskade von Ensembles

… die ersten drei Stufen (jede ein ADABOOST-Klassifikator)



Ausgabe auf jeder Stufe: $\text{sign}\left(\sum_{m=1}^{M} h_m(x)\right)$ wobei $h(x)=\alpha$, wenn $x \geq thr$, $h(x)=\beta$ sonst

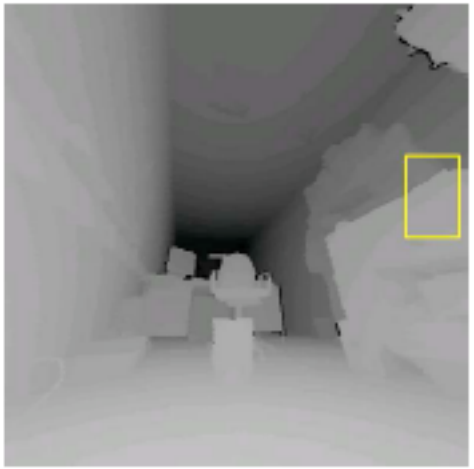
Beispiel Stühle: Ein paar Ergebnisse



Korrekt in beiden Sensormodalitäten

Falsch negativ in Reflexionsdaten

Falsch positiv und falsch negativ in Tiefendaten



number of stages	hit rate		false alarms	
	reflect. img.	depth img.	reflect. img.	depth img.
15	0.9	0.866	0.067	0.067
30	0.867	0.767	0.067	0.033
(15 + 15) applied to 6 img.	0.967		0.0	