

Grundlagen der Künstlichen Intelligenz

14 Maschinelles Lernen

Warum Lernen funktioniert, Entscheidungslisten, Gruppenlernen

Volker Steinhage

Inhalt

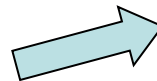
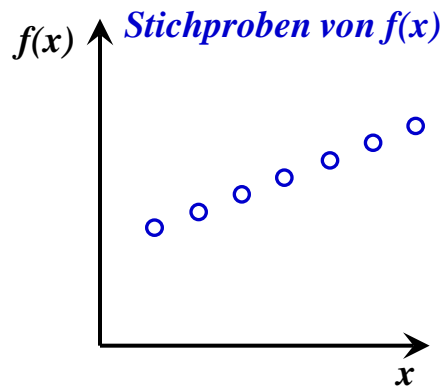
- Warum Lernen funktioniert: PAC-Learning
- Lernen von Entscheidungslisten
- Ensemble-Verfahren bzw. Gruppenlernen: Bagging, Boosting

Warum Lernen funktioniert (1)

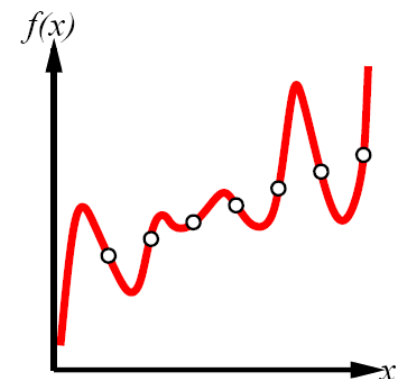
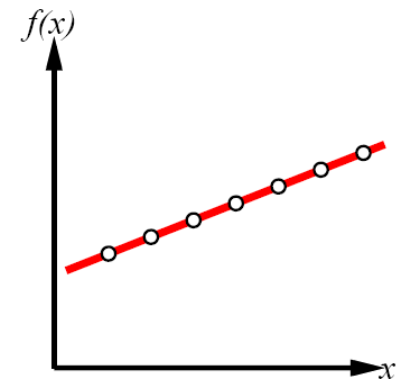
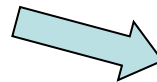
Wie können wir feststellen, ob wir das „*Richtige*“ gelernt haben?

Genauer am Beispiel der **induktiven Inferenz**, bei der für eine Menge $(x_i, f(x_i))$ von Stichproben einer Funktion f eine **Hypothese** h für f zu lernen ist:

Wie können wir entscheiden, dass h dicht an f liegt,
wenn f doch unbekannt ist?



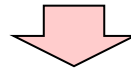
▪
▪
▪



Warum Lernen funktioniert (2)

Die Frage führt zum Begriff „*Probably Approximately Correct*“:

- *Jede ernsthaft falsche Hypothese wird fast sicher mit hoher Wahrscheinlichkeit bereits nach einer kleinen Anzahl von Beispielen „entdeckt“, weil sie eine falsche Vorhersage macht.*
- Bzw.: Jede Hypothese, die auf einer hinreichend großen Trainingsmenge konsistent ist, ist *wahrscheinlich approximativ korrekt*.*



Kernfrage dann: Kann man abschätzen, wie viele Beispiele benötigt werden?

* Dabei gilt *Stationarität* als Grundannahme des sog. PAC-Learning: Trainings- und Validierungsmenge werden mit der gleichen Verteilung aus der Population ausgewählt – ansonsten gäbe es keine Verbindung zwischen Trainings- und Validierungsmenge!

Warum Lernen funktioniert (3)

Frage also: Wie viele Beispiele braucht man für PAC-Learning?

Nötige Begriffe:

\mathbf{X} Trainingsmenge,

D Verteilung, nach der Elemente von \mathbf{X} gezogen werden,

H Hypothesenraum ($f \in H$),

N Anzahl der Beispiele in der Trainingsmenge.

Der Fehler von Hypothese h zur wahren Funktion f bei Verteilung D von \mathbf{X} wird definiert als die Wahrscheinlichkeit, dass sich h von f für ein Beispiel x unterscheidet:

$$\text{error}(h) = P(h(x) \neq f(x) | x \text{ aus } D \text{ gezogen}).$$

PAC Lernen (1)

- Eine Hypothese h heißt *approximativ korrekt*, wenn

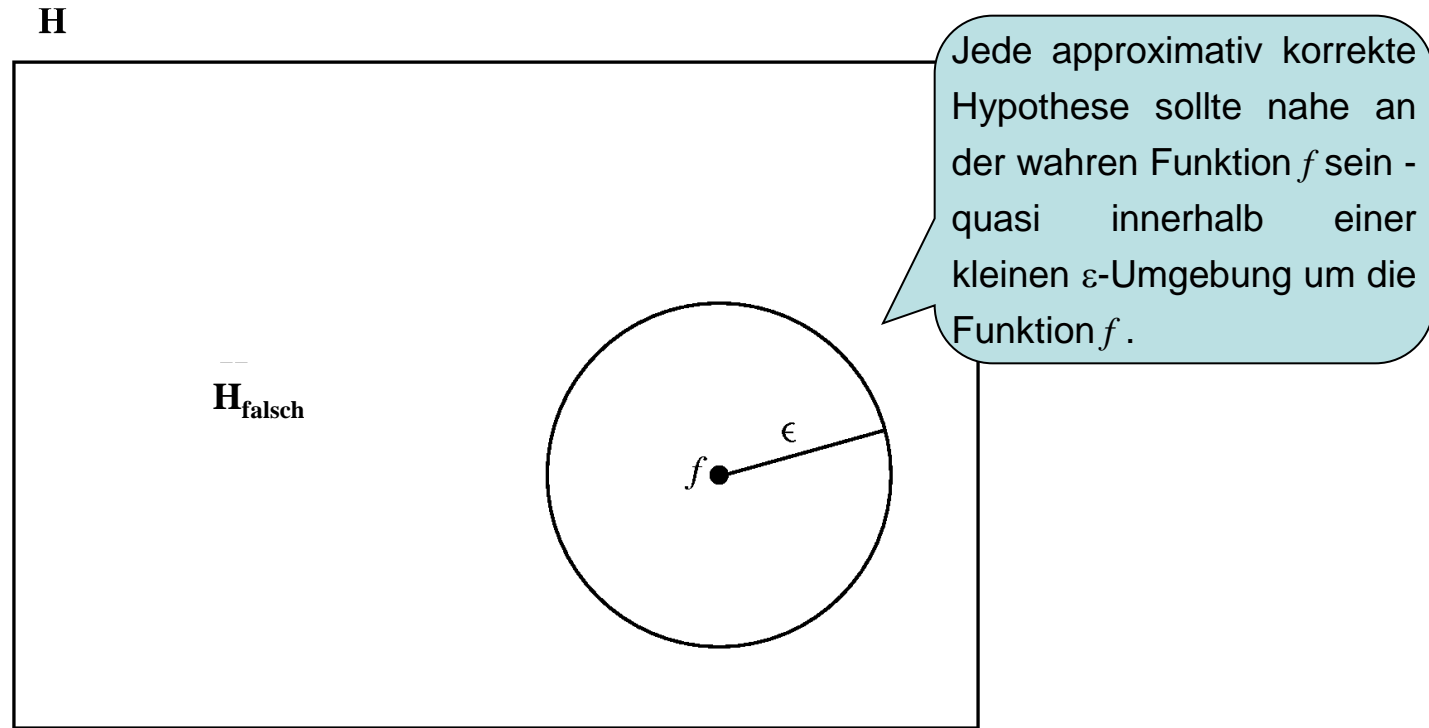
$$\text{error}(h) = P(h(x) \neq f(x) | x \text{ aus } D \text{ gezogen}) \leq \varepsilon$$

für eine kleine positive Konstante ε .

- Jeder Lernalgorithmus, der Hypothesen erzeugt, die wahrscheinlich approximativ korrekt sind, wird als *PAC-Lernalgorithmus* bezeichnet.
- Zu zeigen: Nach dem Training mit N Beispielen ist jede konsistente Hypothese mit hoher Wahrscheinlichkeit approximativ korrekt.

PAC Lernen (2)

Zu zeigen also: Nach dem Training mit N Beispielen ist jede konsistente Hypothese mit hoher Wahrscheinlichkeit approximativ korrekt.



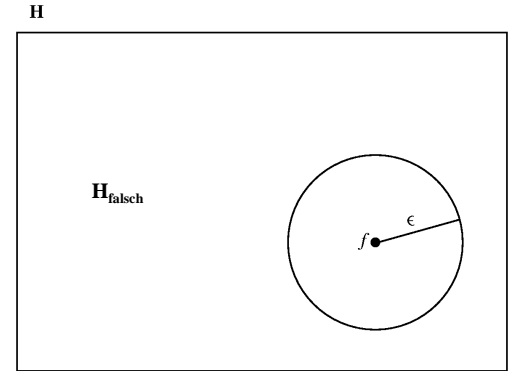
Herleitung: wie hoch ist die Wahrscheinlichkeit, dass eine *nicht* approximativ korrekte Hypothese $h_b \in H_{\text{falsch}}$ *konsistent* mit den ersten N Beispielen ist?

Stichprobenkomplexität (1)

Annahme also: **nicht** appr. korrekte Hypothese $h_f \in \mathbf{H}_{\text{falsch}}$

$$\text{also: } \text{error}(h_f) = P(h_f(x) \neq f(x)) > \varepsilon$$

ist konsistent mit den ersten N Beispielen!



$$\leadsto P(h_f \text{ konsistent mit 1 Beispiel}) \leq (1 - \varepsilon)$$

$$\leadsto P(h_f \text{ konsistent mit } N \text{ Beispielen}) \leq (1 - \varepsilon)^N$$

$$\leadsto P(\mathbf{H}_{\text{falsch}} \text{ enthält mit } N \text{ Bsplen. konsistente } h_f) \leq |\mathbf{H}_{\text{falsch}}| (1 - \varepsilon)^N$$

Mit Abschätzung $|\mathbf{H}_{\text{falsch}}| \leq |\mathbf{H}|$ folgt:

$$P(\mathbf{H}_{\text{falsch}} \text{ enthält konsistente } h_f) \leq |\mathbf{H}| (1 - \varepsilon)^N$$

Diese Wahrscheinlichkeit sollte natürlich klein sein!

Stichprobenkomplexität (2)

Forderung ist also:

Diese W'keit δ sollte natürlich klein sein!

$$|H|(1 - \varepsilon)^N \leq \delta \quad \text{für kleines } \delta$$

$$\leadsto \ln |H| + N \ln(1 - \varepsilon) \leq \ln \delta$$

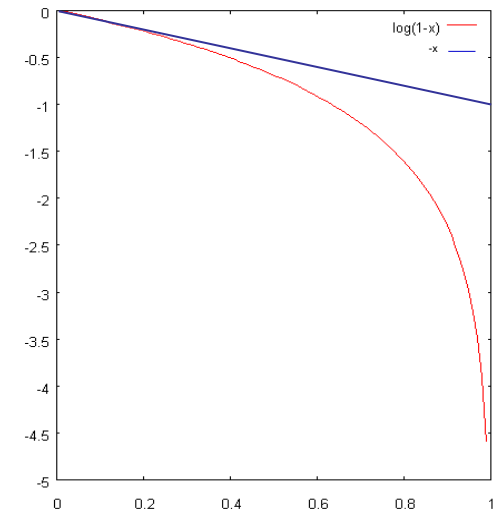
$$\leadsto N \geq (\ln \delta - \ln |H|) / \ln(1 - \varepsilon)$$

$$\leadsto N \geq (\ln \delta - \ln |H|) / (-\varepsilon) \quad \text{wegen } \ln(1 - \varepsilon) \leq -\varepsilon \text{ für kleine } \varepsilon$$

$$\leadsto N \geq (-\ln \delta + \ln |H|) / \varepsilon$$

$$\leadsto N \geq (\ln(1/\delta) + \ln |H|) / \varepsilon \quad \text{wegen } -\ln \delta = \ln(1/\delta)$$

$$\leadsto N \geq 1/\varepsilon (\ln(1/\delta) + \ln |H|)$$



Stichprobenkomplexität (3)

Also haben wir

- Stichprobenkomplexität (Sample Complexity) N mit

$$N \geq 1/\varepsilon (\ln (1/\delta) + \ln |H|).$$

Die Stichprobenkomplexität (Sample Complexity) N

- entspricht der Anzahl benötigter Beispiele für approximativ korrekte Hypothese h als Funktion über ε und δ ;
- realisiert ein Maß, um den Trainingsaufwand abzuschätzen:

Ist eine Hypothese mit $N \geq 1/\varepsilon (\ln(1/\delta) + \ln|H|)$ Beispielen konsistent, dann hat sie mit einer W'keit von mindestens $(1 - \delta)$ einen Fehler von höchstens ε . D.h. sie ist mit einer W'keit von mindestens $(1 - \delta)$ approximativ korrekt.

→ Problem: Die Größe des Hypothesenraums H .

Stichprobenkomplexität (4)

Beispiel: Boolesche Funktionen

- Für die *Booleschen Funktionen* wächst der Hypothesenraum H doppelt exponentiell über den n Attributen: $|H| = 2^{2^n}$
- ↪ die Stichprobenkomplexität wächst also exponentiell zu $\ln|H| = 2^n$
- 2^n ist aber gerade die Anzahl möglicher Beispiele
- ↪ gibt es keinen Lernalgorithmus, der besser ist als eine Lookup-Tabelle und gleichzeitig konsistent mit allen bekannten Beispielen ist.
- **Dilemma:**
 - Schränkt man den Raum der von einem Lernalgorithmus zu berücksichtigenden Hypothesen nicht ein, dann kann kein Algorithmus richtig lernen.
 - Schränkt man ihn ein, dann ist f eventuell nicht mehr in H enthalten.

Stichprobenkomplexität (5)

Lösung 1: Forderung nach Lösungen, die konsistent und einfach sind (z.B. **Kompaktheit bei Entscheidungsbäumen**).

Man kann zeigen: ist die Ermittlung *einfacher* konsistenter Hypothesen handhabbar, sind die Stichprobenkomplexitätsergebnisse i.A. besser als bei Analysen, die nur auf Konsistenz basieren.

Allg. geht damit eine Beschränkung der *Ausdruckskraft* einher, ähnlich wie bei Beschränkung auf Hornklauselmengen in PL1. Die eingeschränkte Ausdruckskraft kann aber hinreichend die für bestimmte Anwendungen sein!

Lösung 2: Angemessene Einschränkung des Hypothesenraumes H auf Teilräume *syntaktisch einfacherer Formen* \leadsto als Bspl. **Entscheidungslisten**.

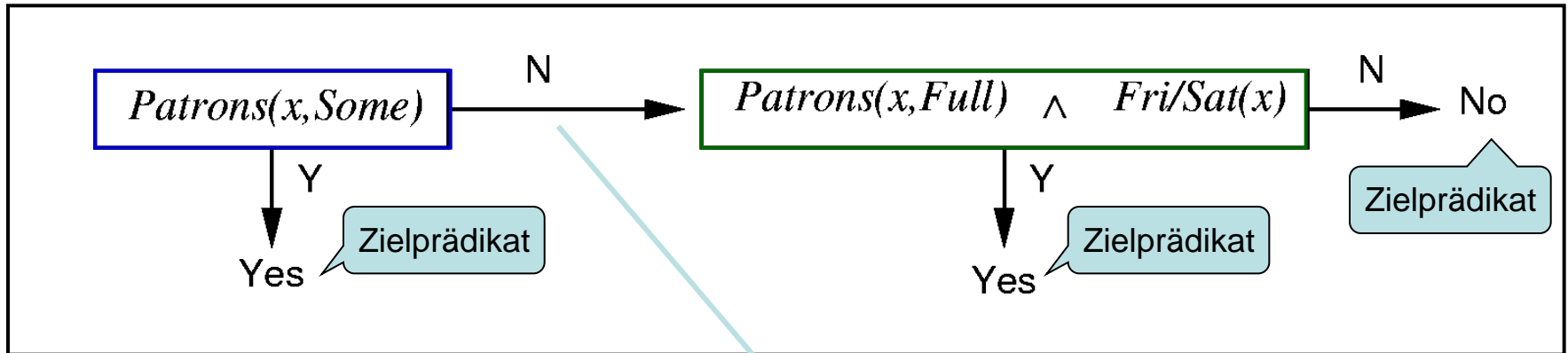
Lernen von **Entscheidungslisten**

Entscheidungsliste = **Folge von Tests** & jeder Test ist ein **Konjunkt von Literalen**.

Im Vergleich zu Entscheidungsbäumen:

- die Gesamtstruktur ist einfacher,
- der einzelne Test ist i.A. komplexer.

Bspl.:



Dies entspricht der Hypothese

$$H: \forall x \text{ WillWait}(x) \Leftrightarrow \text{Patrons}(x, \text{Some}) \vee [\text{Patrons}(x, \text{Full}) \wedge \text{Fri/Sat}(x)].$$

Ausdruckskraft von Entscheidungslisten

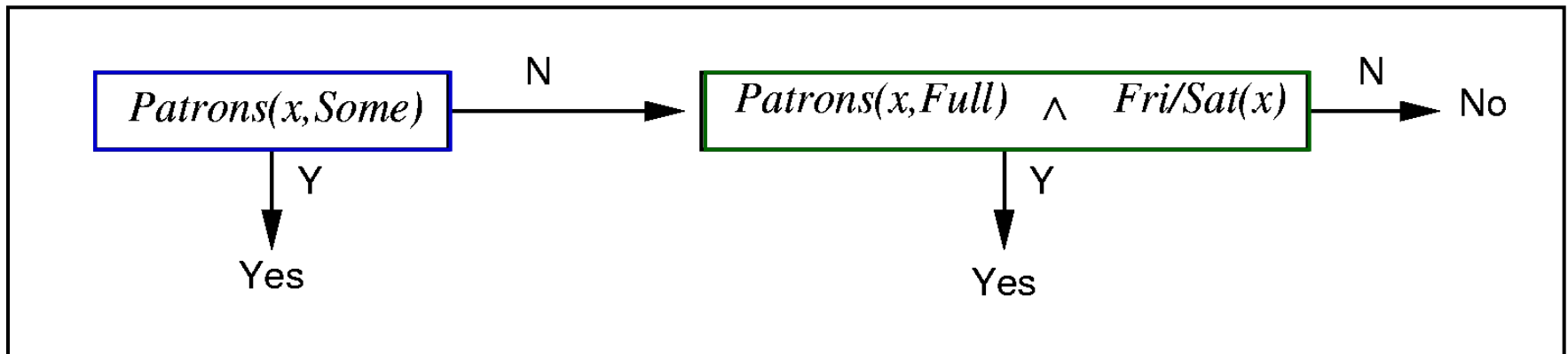
Sind Tests von beliebiger Länge zugelassen, so können beliebige Boolesche Funktionen dargestellt werden.

Bezeichnen

- **k-DL** die Sprache der Entscheidungslisten mit Tests der Länge $\leq k$
- **k-DT** die Sprache der Entscheidungsbäume mit Tiefe $\leq k$,

so gilt, dass : **k-DT** eine Untermenge von **k-DL** ist:

$$\mathbf{k-DT} \subseteq \mathbf{k-DL} .$$



Die obige Entscheidungsliste ist in der Sprache **2-DL**.

Algorithmus DECISION-LIST-LEARNING

function DECISION-LIST-LEARNING(*examples*) **returns** a decision list, or *failure*

if *examples* is empty **then return** the trivial decision list *No*

Triviale
Liste No

$t \leftarrow$ a test that matches a nonempty subset $examples_t$ of *examples*

such that the members of $examples_t$ are all positive or all negative

if there is no such t **then return** failure

if the examples in $examples_t$ are positive **then** $o \leftarrow$ *Yes*

else $o \leftarrow$ *No*

Antwort No

return a decision list with initial test t and outcome o

and remaining elements given by DECISION-LIST-LEARNING($examples - examples_t$)

Der Algorithmus spezifiziert keine Methode zur Auswahl der Tests!

Analog zu den Überlegungen bei Entscheidungsbäumen sind zu präferieren:
einfache Tests, die für einen großen Anteil der Beispiele zu möglichst eindeutigen Zuordnungen führen.

Lernbarkeit von k-DL

Zu zeigen: jede Funktion der *Sprache k-DL* kann approximativ korrekt nach einer hinreichenden Zahl von Trainingsbeispielen gelernt werden.

~ Sample Complexity?

Bezeichne *Conj(n,k)* die Sprache der Tests über Konjunkten mit max. *k* Literalen über *n* Attributen. Da in einer Entscheidungsliste jedem Test das Ergebnis Yes oder No zugeordnet werden kann oder der Test gar nicht vorkommt, gibt es höchstens $3^{|Conj(n,k)|}$ Mengen von Komponententests:

$$|k - DL(n)| \leq 3^{|Conj(n,k)|} |Conj(n,k)|! \quad (\text{Yes, No, keinTest, Permutationen})$$

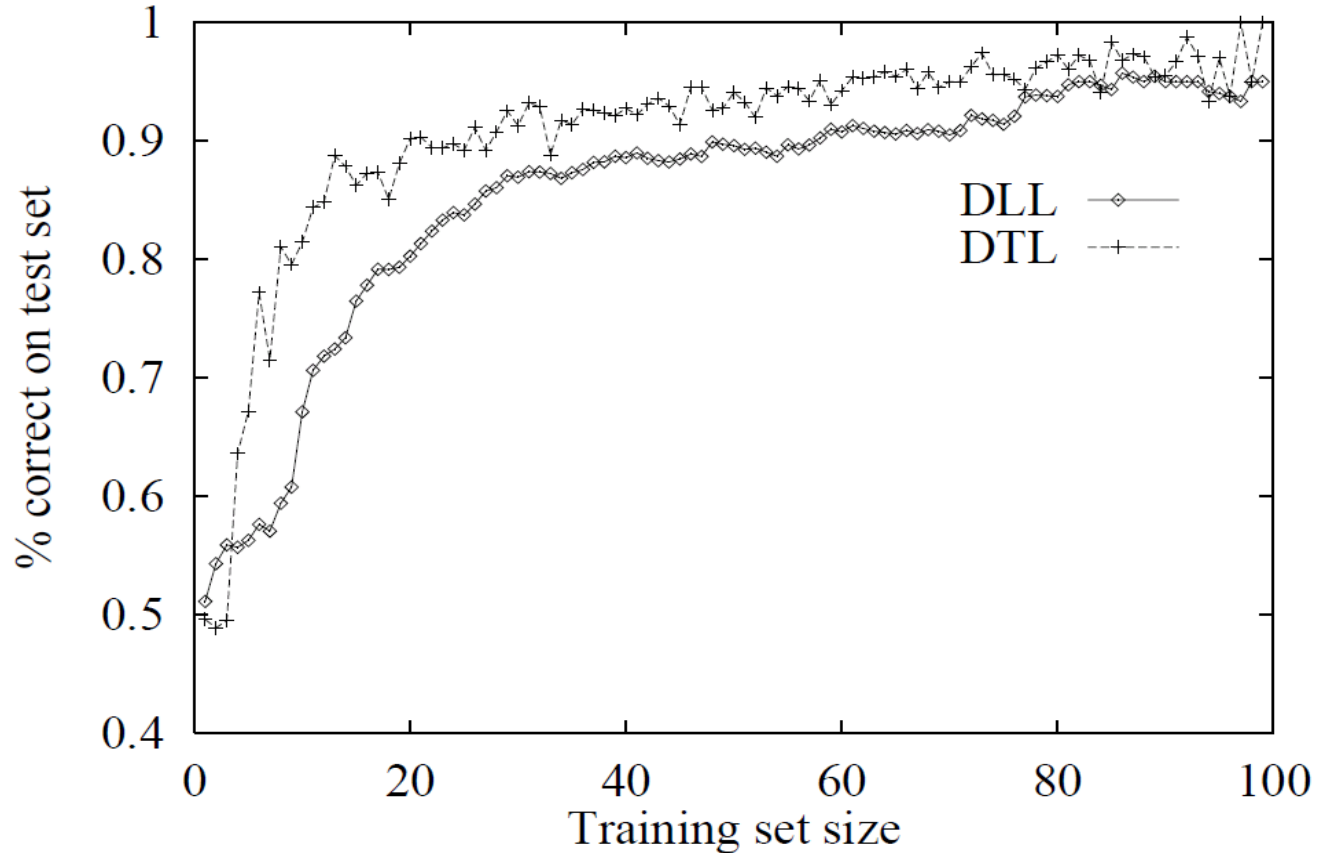
$$\begin{aligned} |Conj(n,k)| &= \sum_{i=0}^k \binom{2n}{i} && (\text{Kombination ohne Wdh. von pos/neg Attribut}) \\ &= O(n^k) \end{aligned}$$

$$|k - DL(n)| = 2^{O(n^k \log_2(n^k))} \quad \text{"Nach einiger Arbeit"}$$

$$N \geq \frac{1}{\varepsilon} \left(\ln \frac{1}{\delta} + O(n^k \log_2(n^k)) \right)$$

Die für das PAC-Lernen von k-DL-Funktionen benötigte Zahl von Beispielen ist damit polynomiell in *n* und damit für hinreichend kleine *k* handhabbar.

Performanz von k-DL



Lernkurven von DECISION-LIST-LEARNING und DECISION-TREE-LEARNING auf den Restaurantdaten.

Gruppenlernen

Bisher: Lernverfahren, die eine Hypothese h aus H auswählen, um eine Funktion f zu lernen.

Ensemble-Verfahren (für Klassifikation): Verwende eine *Menge von Hypothesen* $\{h_1, \dots, h_n\}$ und kombiniere ihre Vorhersagen.

Beispiel:

Annahme, wir haben $M = 5$ Hypothesen für eine Klassifikationsaufgabe und verbinden die Vorhersagen durch Mehrheitsentscheid.

Für eine Fehlklassifikation müssen dann mindestens drei Hypothesen falsch liegen!

Bagging

Diese Idee wird beim sog. **Bagging** (**B**ootstrap **A**ggregating) ausgenutzt:

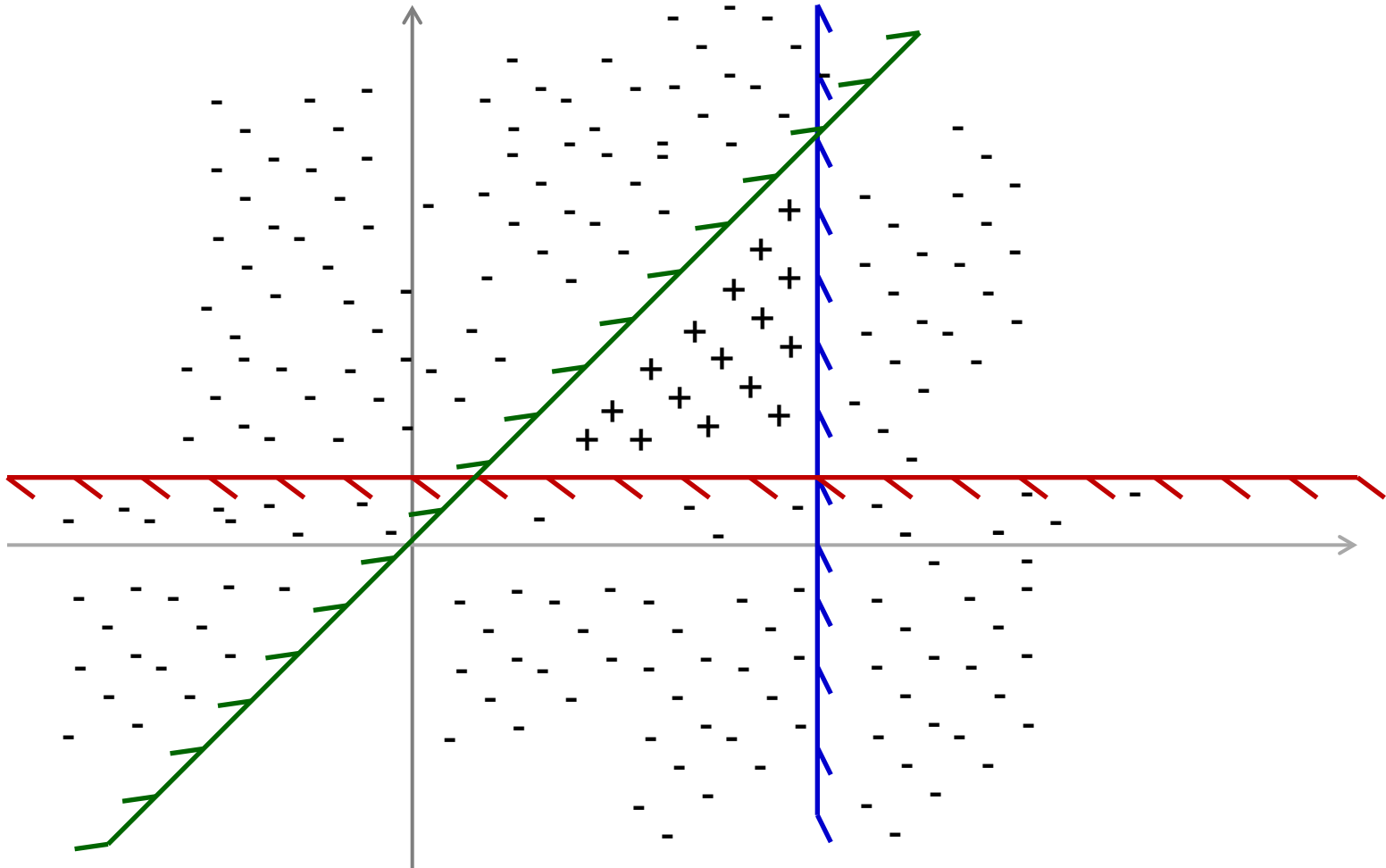
- Erzeuge M unterschiedliche Trainingsmengen durch Bootstrap Sampling aus der Originalmenge. (Bootstrap sampling: Beispiele ziehen mit Zurücklegen.)
- Lerne M verschiedene Klassifizierer auf diesen Untermengen.
- Für jede Anfrage: Lasse alle Klassifizierer vorhersagen und nehme den Mehrheitsentscheid (oder Mittelwert).

Grundidee: Wenn die Klassifizierer unabhängige Fehler machen, erhöht das Ensemble die Performanz.

Schematisches Beispiel

Einfache Klassifizierer: lineare Entscheidungsgrenzen.

Bagging, das akzeptiert, wenn alle drei Klassifizierer akzeptieren.



Boosting

Idee:



- trainiere *sequentiell* Klassifizierer auf allen Trainingsdaten
- jeder nächste Klassifizierer sollte sich auf die Beispiele konzentrieren, die seine Vorgänger falsch klassifiziert haben.
- für Anfragen: kombiniere Klassifizierer durch gewichteten Mehrheitsentscheid.

Die einzelnen Klassifizierer können dabei schwach sein; z.B. sogenannte **Entscheidungsstümpfe** (decision stumps).

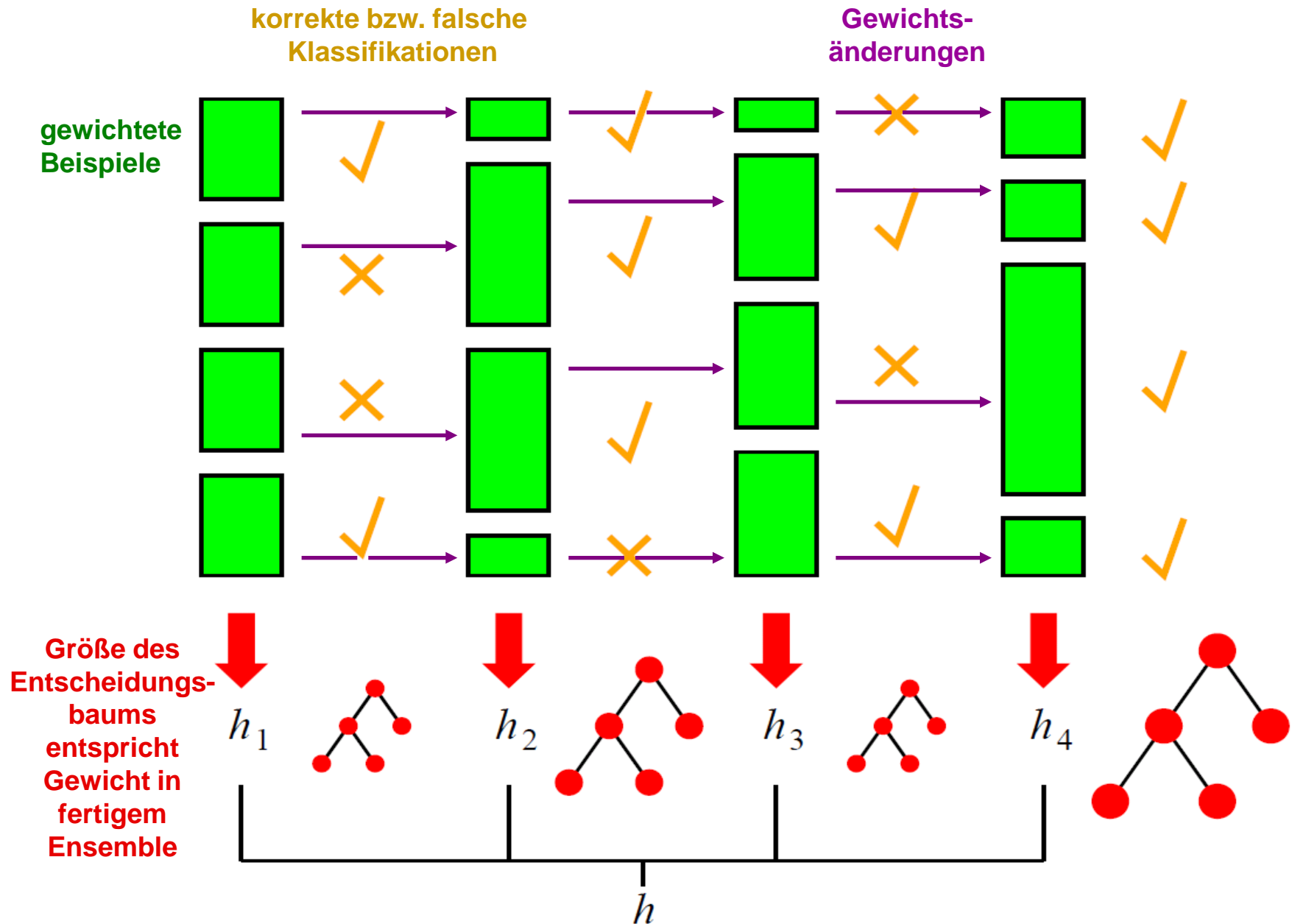
Ein Entscheidungstumpf ist im Prinzip ein Entscheidungsbaum, der nur aus einem Test besteht. Ein solcher Test vergleicht den Wert eines einzelnen Attributes mit einem Schwellwert und begründen damit die Klassifikation.

Die Wahl des Attributes und des Schwellwertes (innerhalb des jeweils gültigen Wertebereiches) können randomisiert erfolgen (s. Folie 26).

Boosting: Vorgehensweise

- Boosting **gewichtet die Trainingsbeispiele**: Beispiele, die falsch *klassifiziert* wurden, erhalten ein *erhöhtes Gewicht*. 
- Jede Runde lernt einen neuen Klassifizierer auf den jeweils neu gewichteten Daten.
- **Klassifizierer** mit *niedrigerem gewichteten Trainingsfehler* **erhalten** *höheres* **Gewicht**. 
- Abbruch entweder nach fester Anzahl von Runden oder in Abhängigkeit von der Performanz auf den Testdaten.

Boosting: Schematisches Beispiel



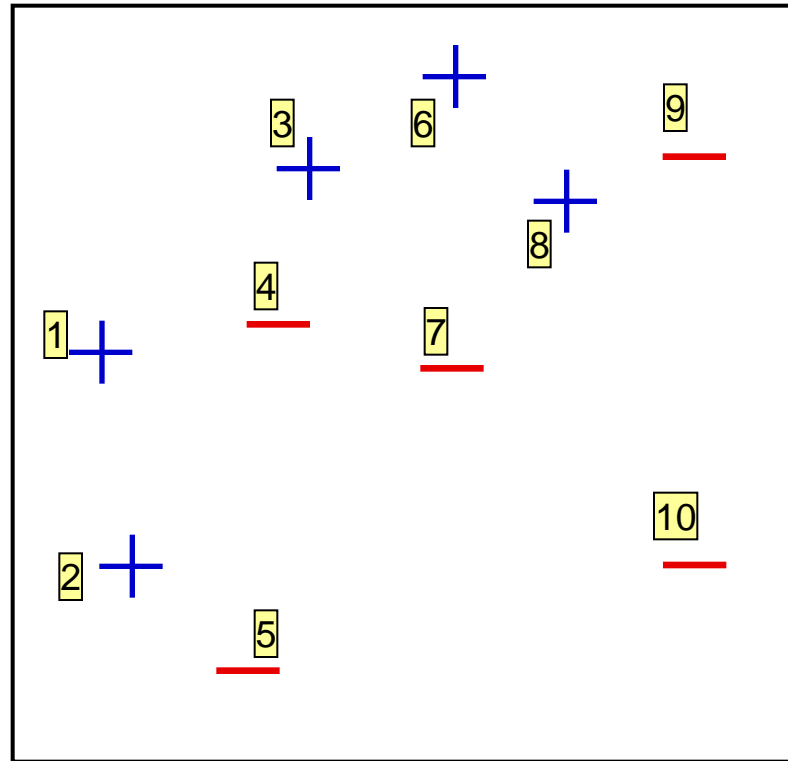
AdaBoost und dessen Pseudo-Code

Die spezielle Variante **AdaBoost** (Adaptive Boosting):

```
function AdaBoost(examples, L, M) returns a weighted-majority hypothesis
inputs: examples, set of  $N$  labeled examples  $(x_1, y_1), \dots (x_N, y_N)$ 
           $L$ , a learning algorithm
           $M$ , the number of hypotheses in the ensemble
local variables:  $w$ , a vector of  $N$  example weights, every  $w[j]$  is initially  $1/N$ 
                    $h$ , a vector of  $M$  hypotheses
                    $z$ , a vector of  $M$  hypothesis weights
for  $m = 1$  to  $M$  do                                // for every hypothesis
     $h[m] \leftarrow L(\text{examples}, w)$                 // learn hypothesis
    error  $\leftarrow 0$ 
    for  $j = 1$  to  $N$  do                                // for every example
        if  $h[m](x_j) \neq y_j$  then error  $\leftarrow$  error +  $w[j]$            // count errors
    for  $j = 1$  to  $N$  do                                // for every example
        if  $h[m](x_j) = y_j$  then  $w[j] \leftarrow w[j] \cdot \text{error} / (1 - \text{error})$  // adjust example weight
     $w \leftarrow \text{Normalize}(w)$ 
     $z[m] \leftarrow \log((1 - \text{error}) / \text{error})$  // compute hypothesis weight
return Weighted-Majority( $h, z$ )
```


AdaBoost Beispiel (1)

Ein grobes Beispiel für AdaBoost, in dem 3 schwache Lerner kombiniert werden:



$N = 10$:
Bsp. (x_j, y_j) mit
Gewichten $w[j]$
für $j = 1, \dots, 10$

$M=3$:
Hypothesen $h[1], h[2], h[3]$
mit Gewichten $z[1], z[2], z[3]$

- 10 Trainingsbeispiele (x_j, y_j) , initial alle gleich gewichtet mit $w[j] = 1/N = 1/10 = 0.1$.
- Wir verwenden lineare achsenparallele Einzelklassifikatoren (z.B. Entscheidungstümpfe).

Entscheidungsstümpfe

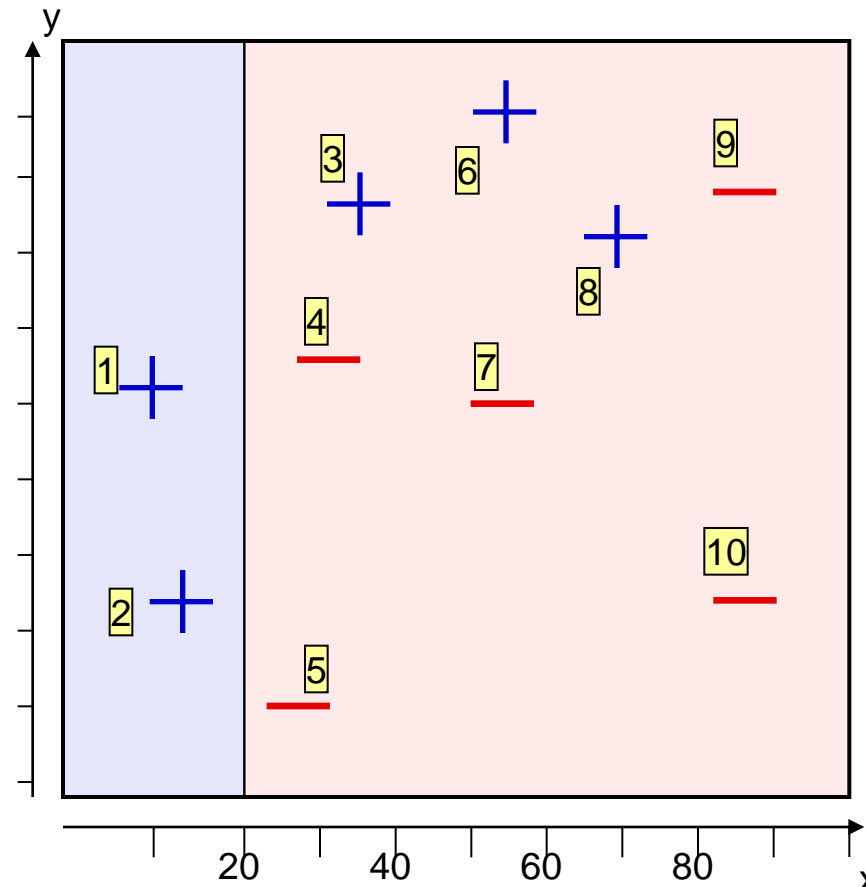
Ein Entscheidungstumpf ist ein Entscheidungsbaum, der nur einen Test zeigt. Dieser ist bestimmbar, indem zunächst die **Dimension d** randomisiert gewählt wird. Dann werden k (k vorgegeben) **Schwellwerte s_i** ($1 \leq i \leq k$) randomisiert für die Klassifikation gewählt.

Über die Entropie bzw. den Gain lässt sich der beste der k Schwellwerte ermitteln.

Im Bspl.: $d = x$, $s_i = 20$.

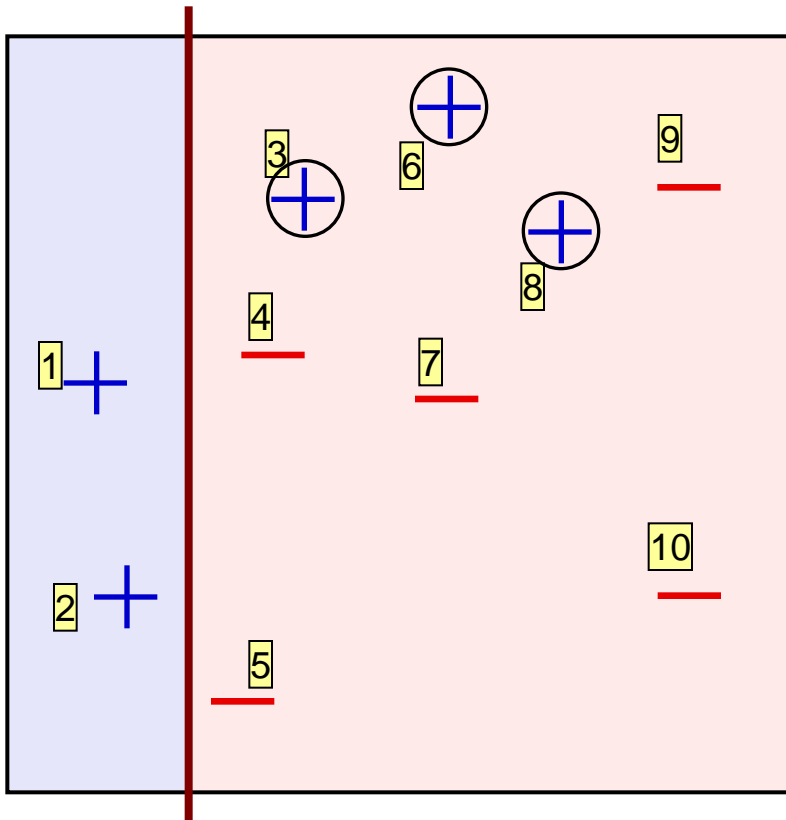
$$\text{Gain}(s_i) = 1 - [0.2 \cdot I(1,0) + 0.8 \cdot I(3/8, 5/8)]$$

Dieser Entscheidungstumpf klassifiziert also über die Parameter $(x, 20)$.

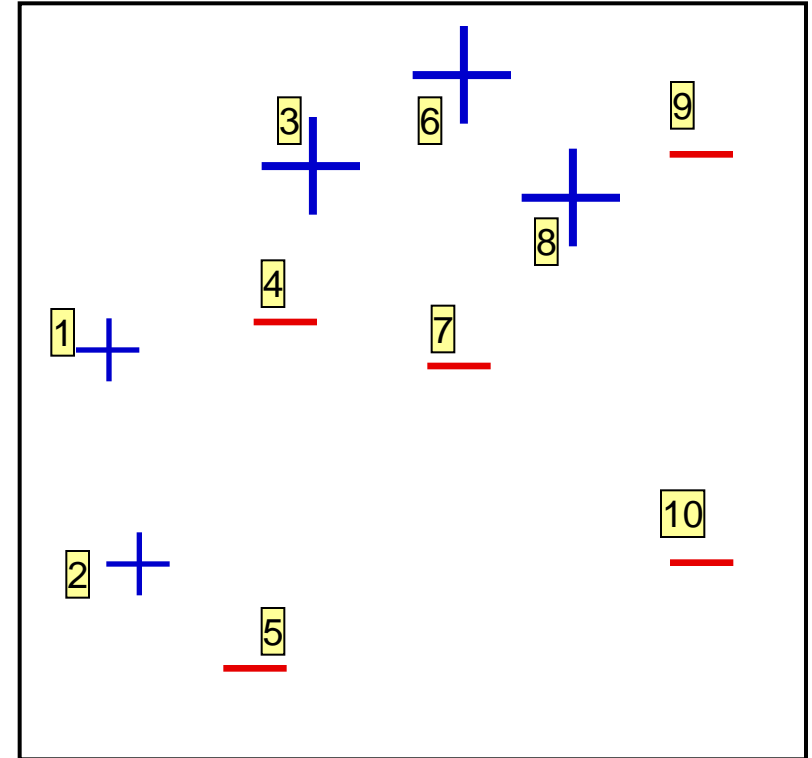


AdaBoost Beispiel (2)

Hypothese 1



Hypothese $h[1]$ mit Gewicht $z[1] \approx 0.85$



Neue Beispielgewichte

10 Bsp. $\leadsto w[j] = 1/10 = 0.1$ für alle Bsp. $j = 1, \dots, 10$

3 Fehler $\leadsto \text{error} = 3 \cdot 0.1 \leadsto \text{error} / (1 - \text{error}) = 0.3 / 0.7 = 0.429$

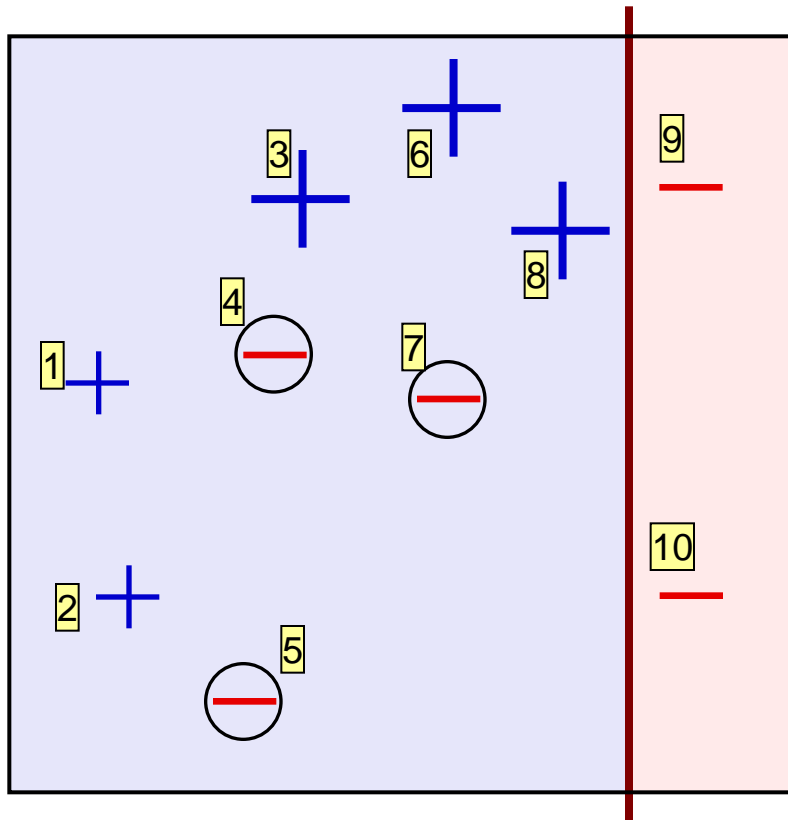
$\leadsto w[j] = 0.1 \cdot 0.429 = 0.0429$ für $j = 1, 2, 4, 5, 7, 9, 10$

$\leadsto z[1] = \log \left(\frac{1 - \text{error}}{\text{error}} \right) \approx 0.85$

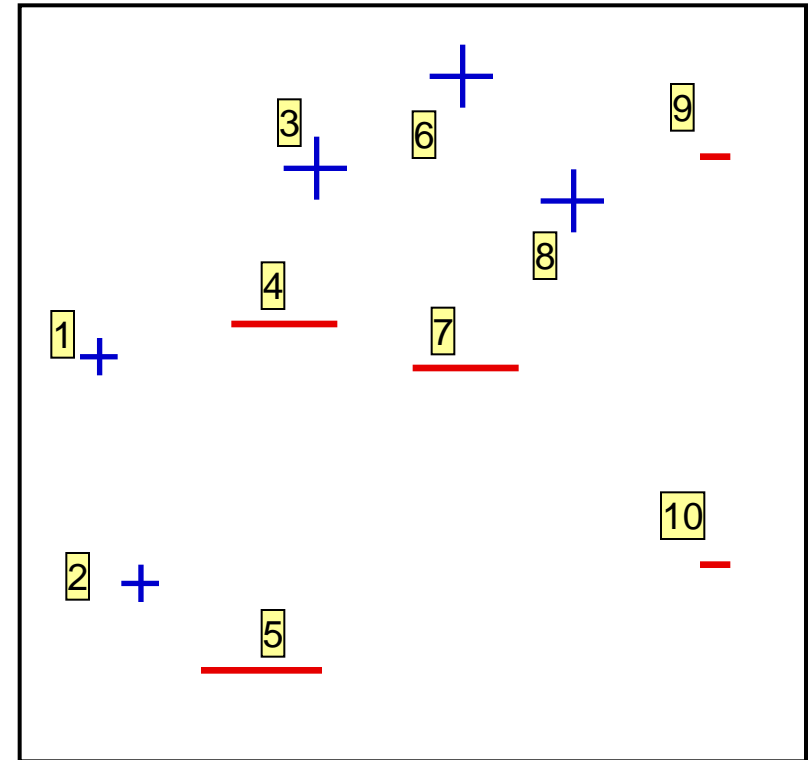
Im Allg. werden Gewichte $w[j]$ für *richtig* klassifizierte Bsp. *verringert*. In der Grafik werden Gewichte für *falsch* klassifizierte Bsp. *vergrößert*.

AdaBoost Beispiel (3)

Hypothese 2



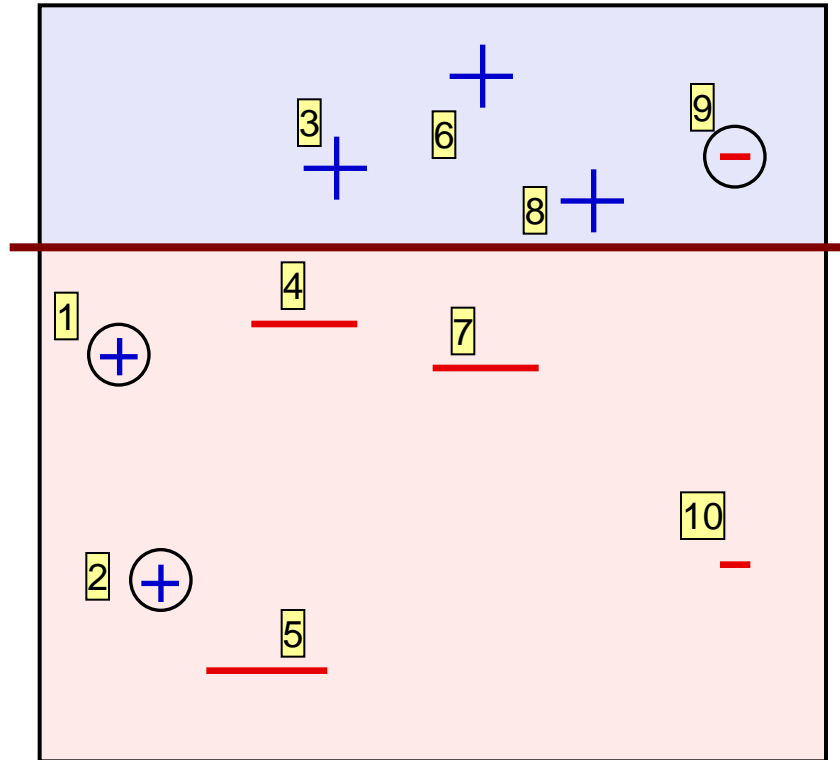
Hypothese $h[2]$ mit Gewicht $z[2] = 1.30$



Neue Beispielgewichte

Beispiel (4)

Hypothese 3



Hypothese $h[3]$ mit Gewicht $z[3] = 1.85$

Beispiel (5)

Gewichtete Gesamthypothese

h_{final}

$$= \text{sign} \left(0.84 \begin{array}{|c|} \hline \text{blue} \\ \hline \end{array} + 1.30 \begin{array}{|c|} \hline \text{blue} \\ \hline \end{array} + 1.85 \begin{array}{|c|} \hline \text{blue} \\ \hline \end{array} \right)$$

The diagram illustrates the weighted sum of three hypotheses. Each hypothesis is represented by a square divided into two regions: blue (left) and red (right). The weights for each hypothesis are 0.84, 1.30, and 1.85. The final decision is the sign of the weighted sum.

Hypothesis	Weight	Blue Region	Red Region
1	0.84	Left	Right
2	1.30	Left	Right
3	1.85	Top	Bottom

Lernkurven

Restaurantbeispiel mit decision stumps als Klassifikatoren.

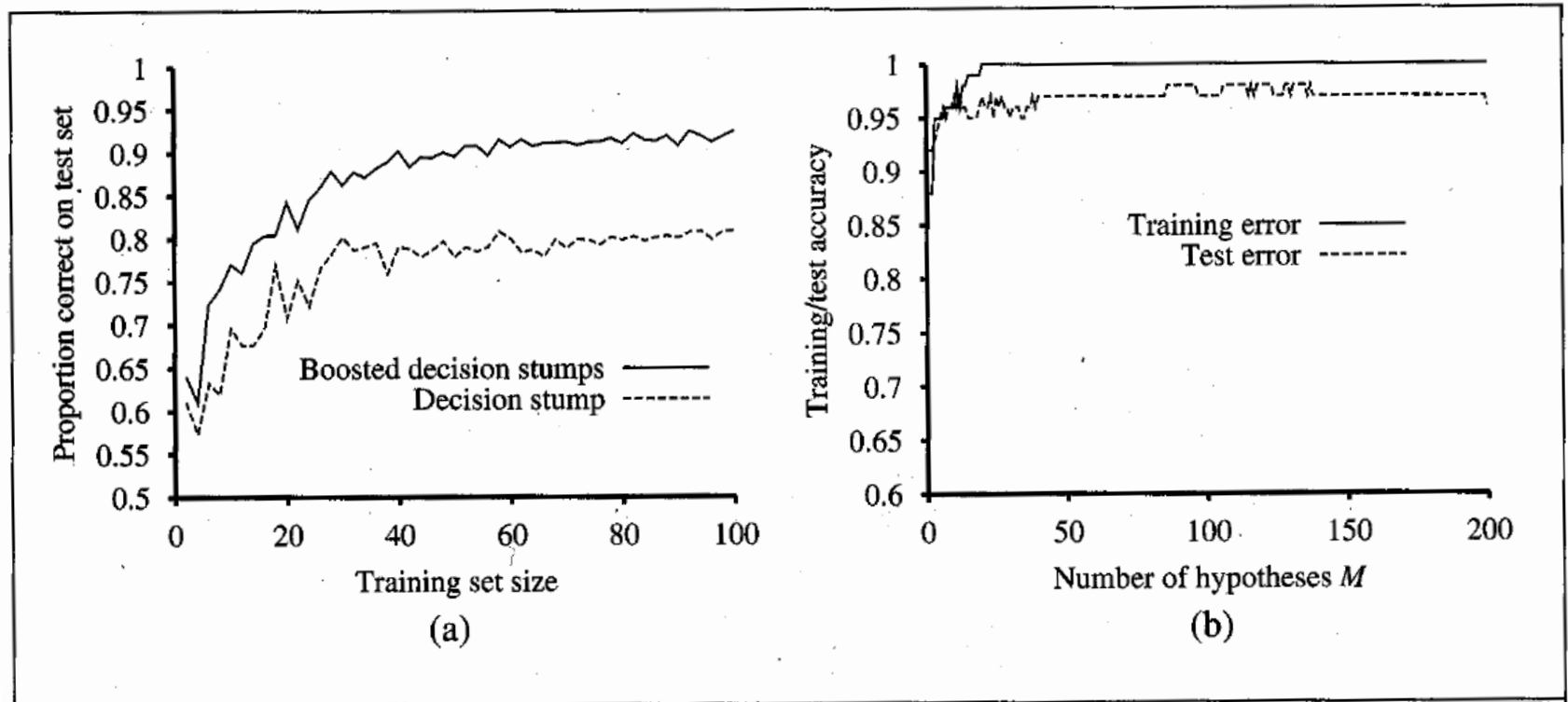


Figure 18.11 (a) Graph showing the performance of boosted decision stumps with $M = 5$ versus decision stumps on the restaurant data. (b) The proportion correct on the training set and the test set as a function of M , the number of hypotheses in the ensemble. Notice that the test set accuracy improves slightly even after the training accuracy reaches 1, i.e., after the ensemble fits the data exactly.

Zusammenfassung

Induktives Lernen als das Lernen der Repräsentation einer Funktion anhand von Beispielen ihres Eingabe/Ausgabe-Verhaltens.

- *PAC-Lernen* beschäftigt sich mit der Komplexität des Lernens.
- *Entscheidungslisten* als „einfach“ zu lernende Funktionen.
- *Ensemble-Verfahren* setzen einfache (schwächere) Lernverfahren ein, indem eine robuste Gesamthypothese durch Abstimmung über die Hypothesen von schwachen Lernern erfolgt.
 - *Bagging* lernt auf unterschiedlichen Trainingsmengen.
 - *Boosting* steuert das Lernen der Hypothesen durch Gewichtung der Trainingsbeispiele.