

Grundlagen der Künstlichen Intelligenz

14 Maschinelles Lernen

Warum Lernen funktioniert, Entscheidungslisten
Gruppenlernen, AdaBoost, Entscheidungswälder

Volker Steinhage

Inhalt

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

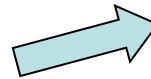
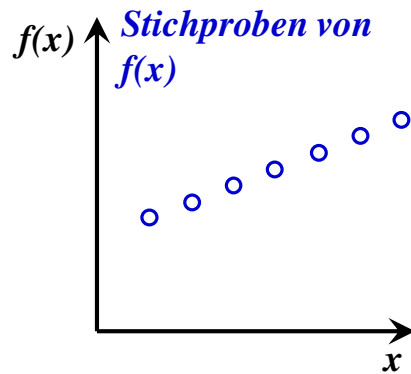
Wie können wir feststellen, ob das „Richtige“ gelernt wurde?

Beispiel der **induktiven Inferenz**:

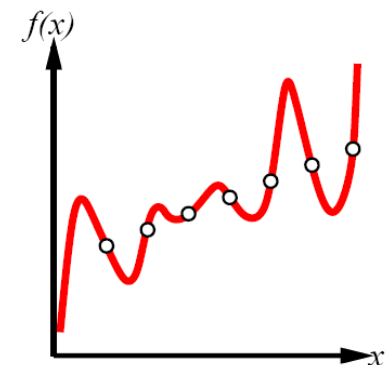
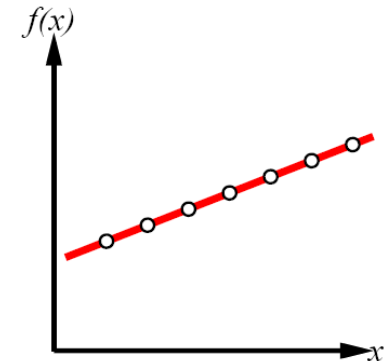
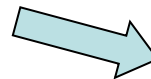
Geg.: Menge $(x_i, f(x_i))$ von Stichproben einer Funktion f

Ziel: Lernen einer **Hypothese** h für f

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



▪
▪
▪

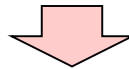


Wahrscheinlich annähernd richtiges Lernen (WARL)

Eine Hypothese h , die auf einer **hinreichend großen Trainingsmenge** konsistent ist, heißt *wahrscheinlich annähernd korrekt* (*probably approximately correct* – kurz: *PAC*), wenn

- h auf ungesehenen Daten mit hoher W'keit (\rightarrow „*probably*“)
- einen geringen Generalisierungsfehler zeigt (\rightarrow „*approximately correct*“).⁽¹⁾⁽²⁾

Jeder Lernalgorithmus, der wahrscheinlich annähernd korrekte Hypothesen generiert, wird als PAC-Lernalgorithmus (*PAC learning algorithm*) bezeichnet.



Neue Frage: Kann man abschätzen, wie viele Beispiele benötigt werden?

⁽¹⁾ Dabei gilt *Stationarität* als Grundannahme des sog. PAC-Learning: Trainingsmenge und ungesehene Daten werden mit der gleichen W'keitsverteilung aus der Population ausgewählt.

⁽²⁾ Als Umkehrung folgt: Jede ernsthaft falsche Hypothese wird mit hoher Wahrscheinlichkeit bereits nach einer kleinen Anzahl von Beispielen entdeckt, weil sie eine falsche Vorhersage macht.

Präzisierung des PAC-Lernens (1)

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

Gegeben:

\mathcal{T} Trainingsmenge

D Verteilung, nach der Elemente von \mathcal{T} 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 \mathcal{T} wird definiert als die *W'keit*, dass sich h von f für ein Beispiel x unterscheidet:

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

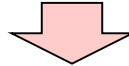
Präzisierung des PAC-Lernens (2)

Eine Hypothese h heißt *annähernd korrekt*, wenn

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

für eine kleine positive Konstante ϵ .

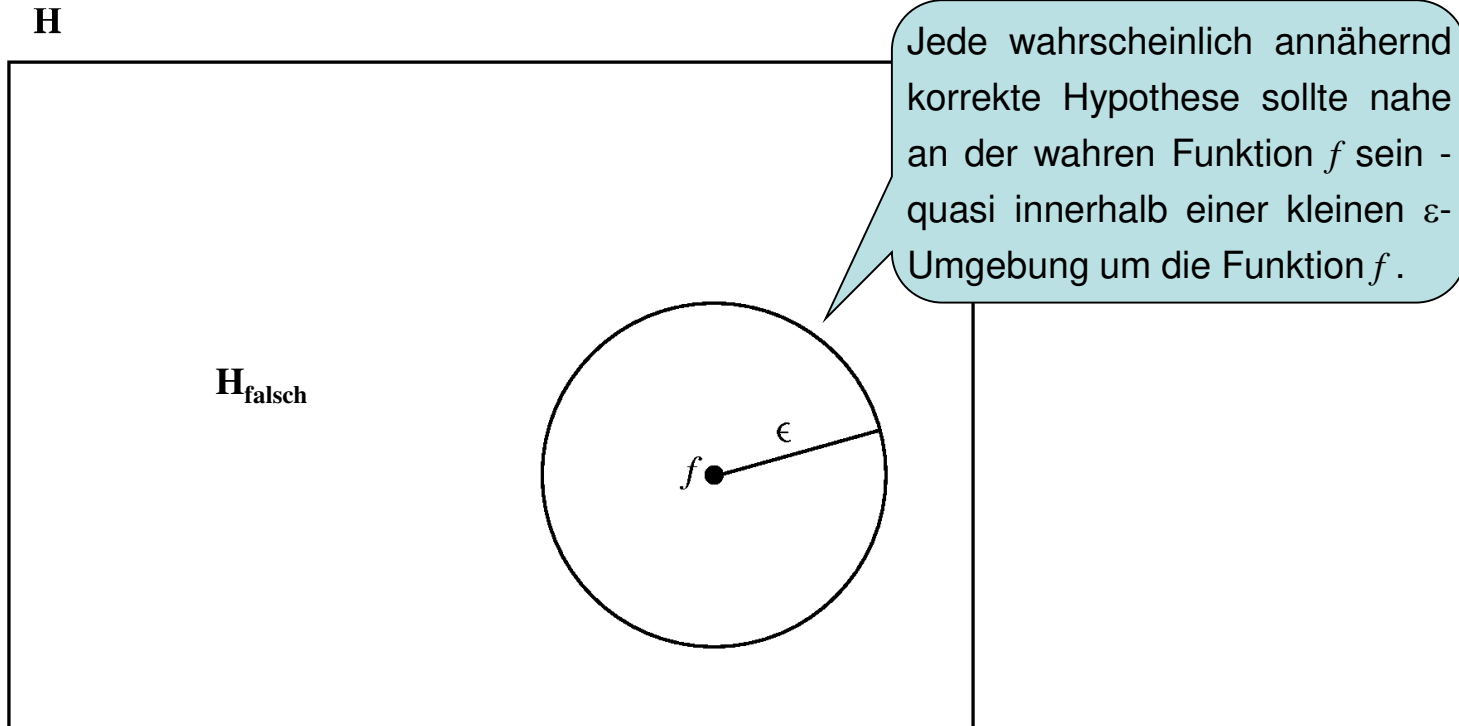
Zu zeigen: Nach dem Training mit N Beispielen ist jede konsistente Hypothese mit hoher W'keit annähernd korrekt.



→ wie groß muss N sein?

Hypothesenraum

Zu zeigen also: Nach dem Training mit N Beispielen ist jede konsistente Hypothese mit hoher W'keit annähernd korrekt



Weg:

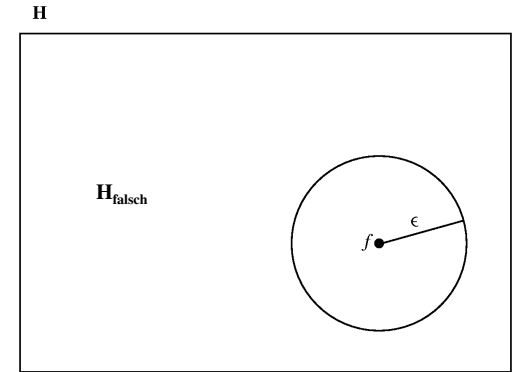
Wir berechnen die Wahrscheinlichkeit, mit der eine ernsthaft falsche Hypothese Hypothese $h_f \in H_{\text{falsch}}$ *konsistent* mit den ersten N Beispielen ist

Herleitung von N (1)

Annahme also: **nicht** PAC-Hypothese $h_f \in \mathbf{H}_{\text{falsch}}$

mit $\text{error}(h_f) = P(h_f(x) \neq f(x)) > \epsilon$

ist konsistent mit den ersten N Beispielen!



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

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

$$\leadsto P(\mathbf{H}_{\text{falsch}} \text{ enthält mit } N \text{ Beispielen konsistente } h_f) \leq |\mathbf{H}_{\text{falsch}}| \cdot (1 - \epsilon)^N$$

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

Diese Wahrscheinlichkeit sollte natürlich klein sein!

$$\leadsto P(\mathbf{H}_{\text{falsch}} \text{ enthält mit } N \text{ Beispielen konsistente } h_f) \leq |\mathbf{H}| \cdot (1 - \epsilon)^N$$

Herleitung von N (2)

Forderung ist nun:

Diese W'keit δ sollte natürlich klein sein!

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

$$\leadsto \ln |\mathbf{H}| + N \cdot \ln(1 - \varepsilon) \leq \ln \delta$$

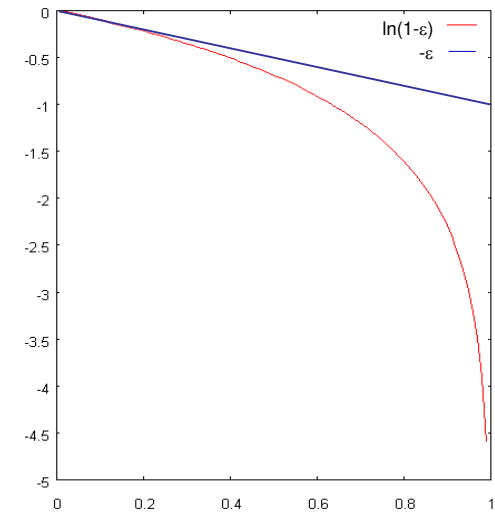
$$\leadsto N \geq (\ln \delta - \ln |\mathbf{H}|) / \ln(1 - \varepsilon)$$

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

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

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

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



Stichprobenkomplexität

Die Stichprobenkomplexität (Sample Complexity)

$$N \geq \frac{1}{\varepsilon} \left(\ln \frac{1}{\delta} + \ln |H| \right)$$

entspricht der Anzahl benötigter Beispiele für wahrscheinlich annähernd korrekte Hypothesen h als Funktion über ε und δ

Die Stichprobenkomplexität N ist damit ein Maß zur Abschätzung des Trainingsaufwandes :

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 (2)

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 $\ln|H| = 2^n$

Für eine *Booleschen Funktionen* mit n Variablen ist 2^n aber gerade die Anzahl aller möglichen Beispiele!

↪ Es gibt keinen Lernalgorithmus, der besser ist als eine Lookup-Tabelle und gleichzeitig konsistent mit allen bekannten Beispielen ist!

Stichprobenkomplexität (3)

Lösung 1: Forderung nach Lösungen, die konsistent und kompakt sind

Bspl.: **Kompaktheit bei Entscheidungsbäumen**

Der Algorithmus **DECISION-TREE-LEARNING** mit der Funktion **CHOOSE-ATTRIBUTE**, die das Attribut A auswählt, das den *Informationsgewinn* $\text{Gain}(A)$ maximiert, berücksichtigt bei der Hypothesengenerierung nicht nur die Konsistenz, sondern auch die Kompaktheit der Hypothesen.

Ist die Ermittlung *einfacher* konsistenter Hypothesen handhabbar, ist die Stichprobenkomplexität i.A. besser als bei Verfahren, die nur auf Konsistenz basieren.

Stichprobenkomplexität (4)

Lösung 2: Angemessene Einschränkung des Hypothesenraumes H auf Teilräume *syntaktisch einfacherer Formen*

Allg. geht damit eine Beschränkung der *Ausdruckskraft* einher, ähnlich wie bei Beschränkung auf Hornklauselmengen in der PL1-Inferenz.

Die eingeschränkte *Ausdruckskraft* kann aber hinreichend für die gegebenen Anwendungen sein!

Hier als Bspl.: **Entscheidungslisten**

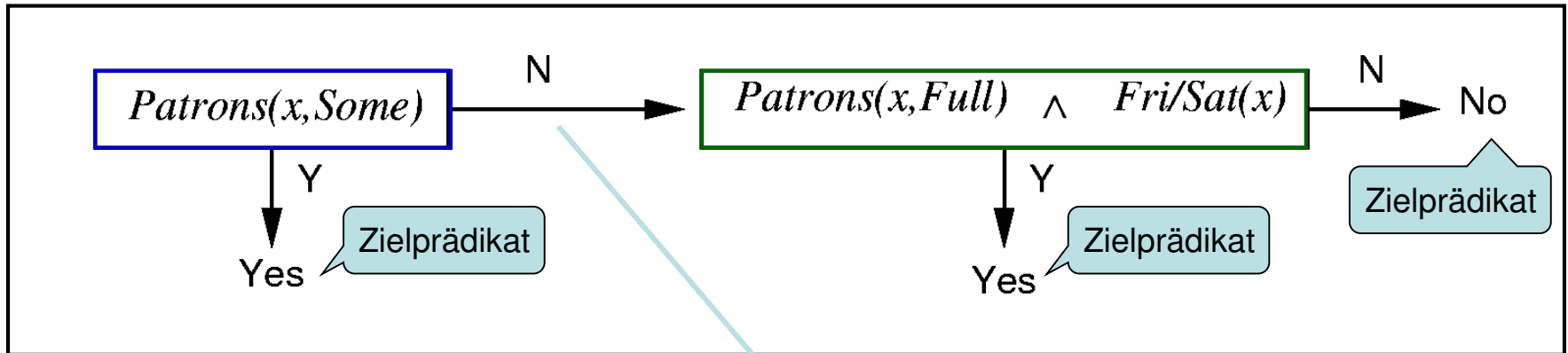
Lernen von Entscheidungslisten

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

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

Ausdruckskraft von Entscheidungslisten

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

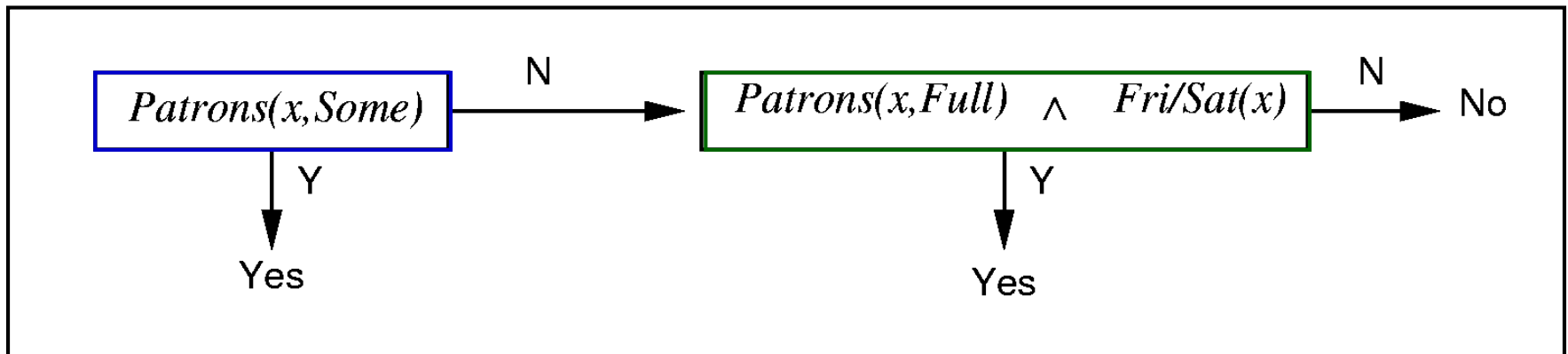
Bezeichnen

Konjunkte mit max. k Literalen

- **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:

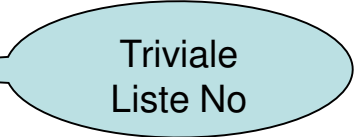
$$\mathbf{k\text{-}DT} \subseteq \mathbf{k\text{-}DL}$$



→ die Entscheidungsliste im Beispiel 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* 

$t \leftarrow$ 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* 

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 \rightarrow also den Informationsgewinn maximieren!

Stichprobenkomplexität von k-DL

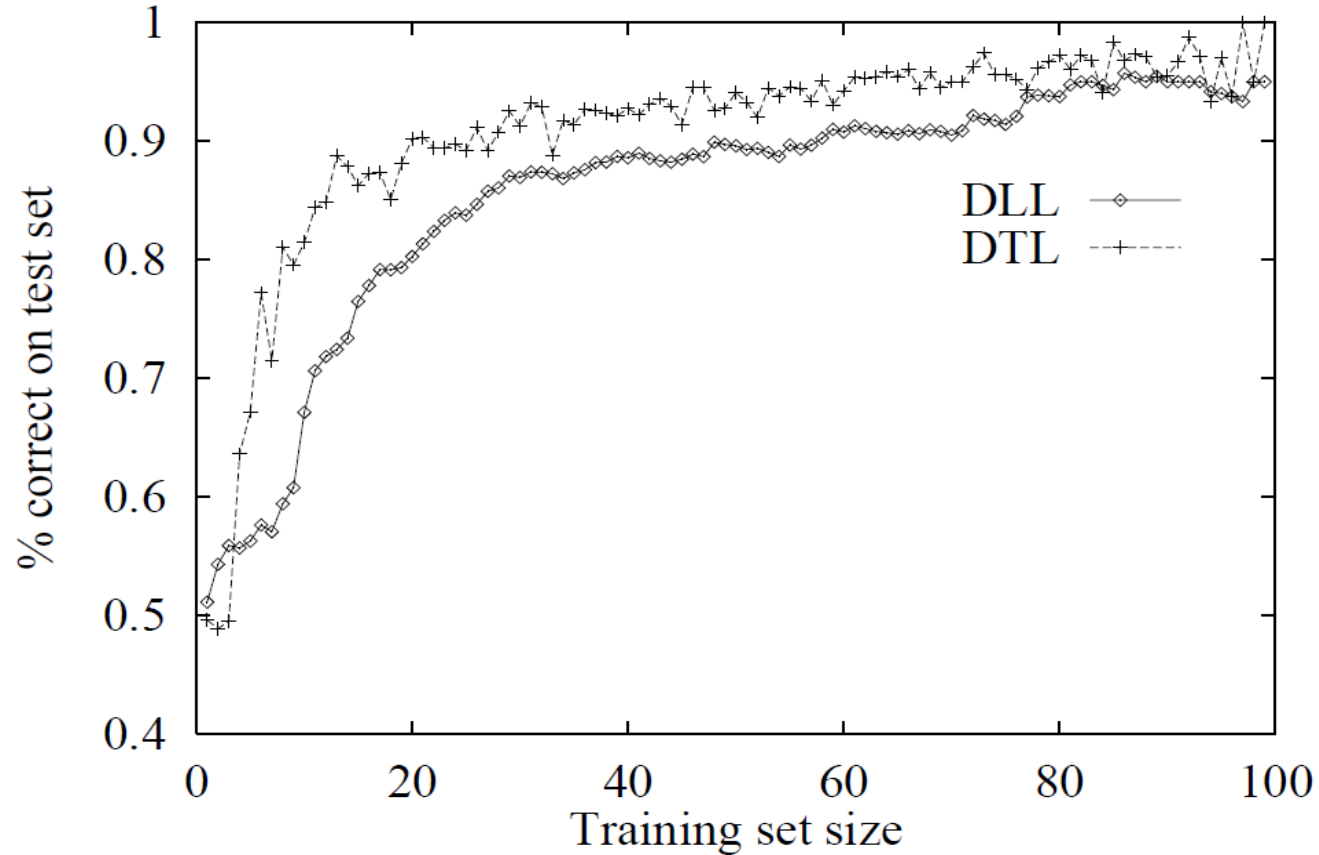
Zu zeigen: jede Funktion der *Sprache k-DL* kann wahrscheinlich annähernd korrekt nach einer hinreichenden Zahl von Trainingsbeispielen gelernt werden.

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. Also gilt für die *Sprache k-DL(n)* über n Attributen:

$$\begin{aligned} |k-DL(n)| &\leq 3^{|Conj(n,k)|} |Conj(n,k)|! && \text{(Yes, No, kein Test, Permutationen)} \\ |Conj(n,k)| &= \sum_{i=0}^k \binom{2n}{i} && \text{(Kombination ohne Wdh. von pos/neg Attribut)} \\ &= O(n^k) \\ |k-DL(n)| &= 2^{O(n^k \log_2(n^k))} && \text{"Nach einiger Arbeit"} \\ N &\geq \frac{1}{\varepsilon} \left(\ln \frac{1}{\delta} + O(n^k \log_2(n^k)) \right) \end{aligned}$$

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.

Einschub: Attribute in Entscheidungsbäumen (1)

Folgende Kategorien werden bei Attributen unterschieden:

- **Nominale Attribute**: Mengen ohne Ordnung (z.B. Früchte)
- **Ordinale Attribute**: Diskrete Mengen mit einer Ordnung (z.B. natürliche oder ganze Zahlen, Schulnoten)
- **Kontinuierliche Attribute**: Teilmengen der rationalen oder reellen Zahlen (z. B. für Temperatur, Geschwindigkeit)

Einschub: Attribute in Entscheidungsbäumen (2)

- Für Attribute mit **geordnetem Wertebereich** (ordinale und kontinuierliche Attribute) sind auch **Tests über die Relationen „>“, „<“, „≥“ und „≤“** möglich bzw. z.T. auch nötig, da vollständige Aufzählungen aller möglicher Werte gar nicht möglich sind.
- Beim Restaurant-Beispiel wären bzgl. des ordinalen Attributs *Price* mit dem geordneten Wertebereich \$, \$\$, \$\$\$ auch Entscheidungsknoten möglich wie „*Price* ≤ \$\$“ oder „*Price* > \$“. Bei einem ordinalen Attribut mit dem Wertebereich der natürl. Zahlen sind Entscheidungsknoten wie „*n* < 100“ sogar nötig.

Prinzipiell sind dann für die Suche nach möglichst kompakten Entscheidungsbäumen aber auch verschiedene binäre Aufteilungen der Trainingsmenge bzgl. solcher Attribute zu untersuchen.

Einschub: Attribute in Entscheidungsbäumen (3)

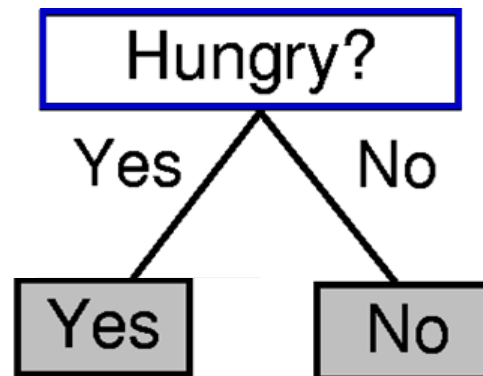
- Um bei der Suche nach möglichst kompakten Entscheidungsbäumen das Testen verschiedener binäre Aufteilungen der Trainingsmenge bzgl. ordinaler und kontinuierlicher Attribute mit großen oder unendlichen Wertebereichen einzugrenzen, kann man das sog. *Binning* umsetzen.
- Beim *Binning* wird der Wertebereich der Größe nach aufsteigend in disjunkte Intervalle – sogenannte *bins* (engl. für *Behälter*) – eingeteilt.
- Beim Restaurant-Beispiel wurde z.B. der kontinuierliche Wertebereich des Attributs *WaitEstimate* auf die vier Bins *0-10*, *10-30*, *30-60*, *>60* unterteilt, sodass ein ordinales Attribut mit 4 möglichen Wertebelegungen entstand.*
- Analog wurde der Wertebereich des Attributs *Prize* auf drei Bins \$, \$\$, \$\$\$ unterteilt.

* Formal nachhaltiger für die Tests wäre natürlich eine Unterteilung in disjunkte Intervalle wie z.B. $[0,10)$, $[10,30)$, $[30,60)$, $[60,\infty)$ gewesen.

Einschub: Entscheidungstümpfe

Ein Entscheidungstumpf (decision stump) ist ein Entscheidungsbaum der Tiefe 1 (one-level decision tree).

- Es gibt nur einen Testknoten (der Wurzelknoten), der direkt mit den Blattknoten verbunden ist
- Ein Entscheidungstumpf trifft Vorhersagen also alleine aufgrund eines Attributwertes



Einschub: Entscheidungstümpfe (2)

Entscheidungstümpfe unterscheiden sich nach der Attributkategorie.

- Für nominale Attribute kann ein Stumpf
 - einen Blattknoten für jeden Attributwert haben ODER
 - einen ersten Blattknoten für einen ausgewählten Attributwert und einen zweiten Blattknoten für die restlichen Attributwerte
- Für ordinale und kontinuierliche Attribute
 - wird häufig ein Schwellwert (Vergleichswert) gewählt, sodass der Stumpf zwei Blattknoten für Attributwerte unterhalb bzw. oberhalb des Schwellwertes zeigt (genauer: \geq und $<$ bzw. \leq und $>$) ODER
 - können auch $k > 1$ Schwellwerte gewählt werden, sodass der Stumpf $k+1$ Blattknoten für die ausgewählten Attributwertebereiche zeigt (s. Binning)

Mehr als 2 Blattknoten

2 Blattknoten

2 Blattknoten

Mehr als 2 Blattknoten

Gruppenlernen

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

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

Beispiel:

- Wir erzeugen $M = 5$ 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** (Bootstrap **A**ggregating) ausgenutzt:

- Erzeuge M unterschiedliche Trainingsmengen durch Bootstrap Sampling* aus der Originalmenge
- Lerne M verschiedene Klassifizierer auf diesen M Untermengen
- Für jede Anfrage: Lasse alle M Klassifizierer vorhersagen und nehme den Mehrheitsentscheid

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

* Bootstrap sampling: Beispiele ziehen mit Zurücklegen

Boosting

Idee:

- trainiere *sequentielle Folge* von Klassifizierern auf allen Trainingsdaten
- jeder Klassifizierer soll 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!

→ Im folg. Beispiel: Entscheidungstümpfe

Boosting: Gewichtung von Beispielen und Klassifikatoren

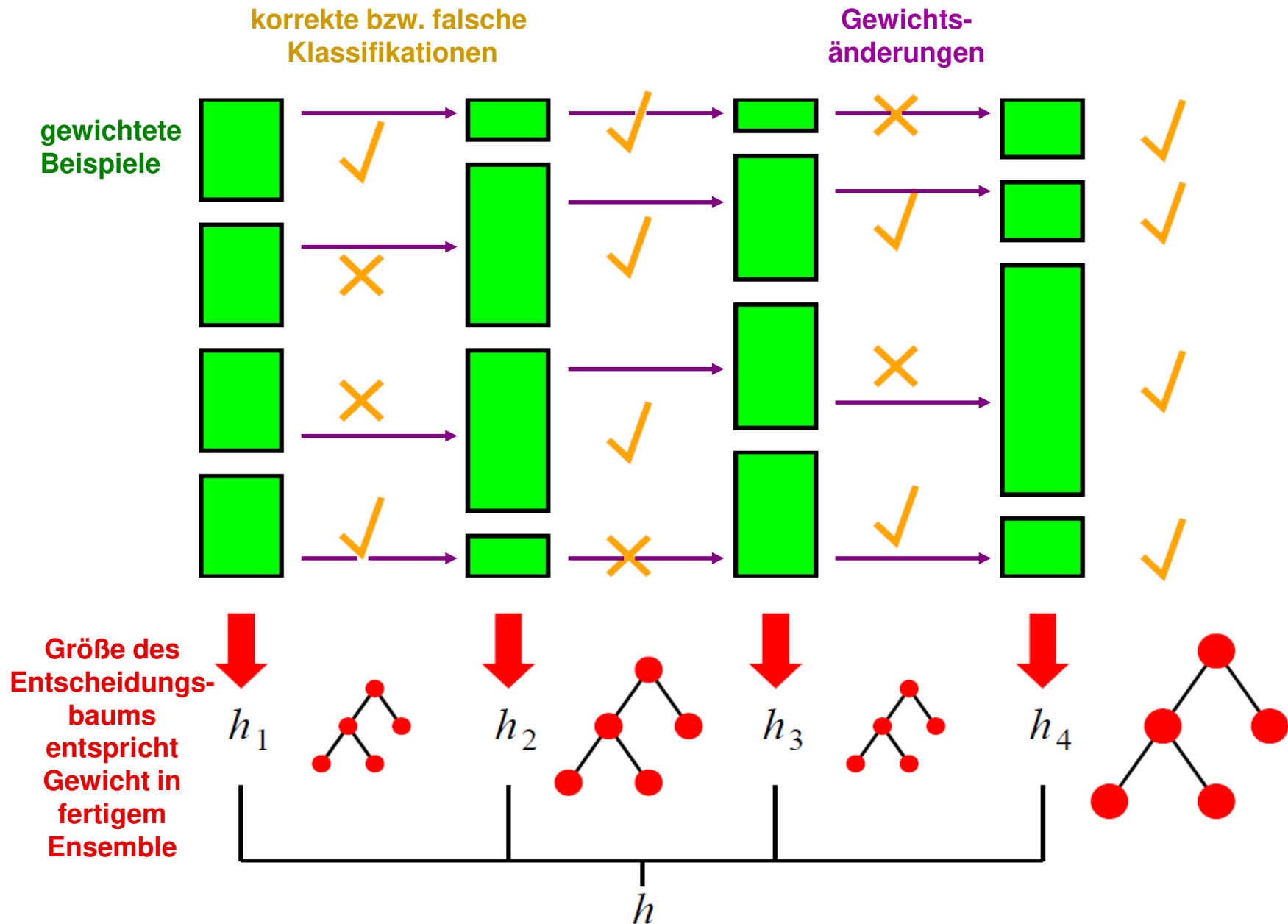
- 1) Boosting **gewichtet die Trainingsbeispiele**: Beispiele, die falsch *klassifiziert* wurden, erhalten ein *erhöhtes Gewicht* bzw. Beispiele, die richtig *klassifiziert* wurden, erhalten ein *erniedrigtes Gewicht*

Jede Runde lernt einen neuen Klassifizierer auf den jeweils neu gewichteten Trainingsbeispielen

- 2) Boosting **gewichtet die Klassifikatoren**: **Klassifikatoren** mit *niedrigerem 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

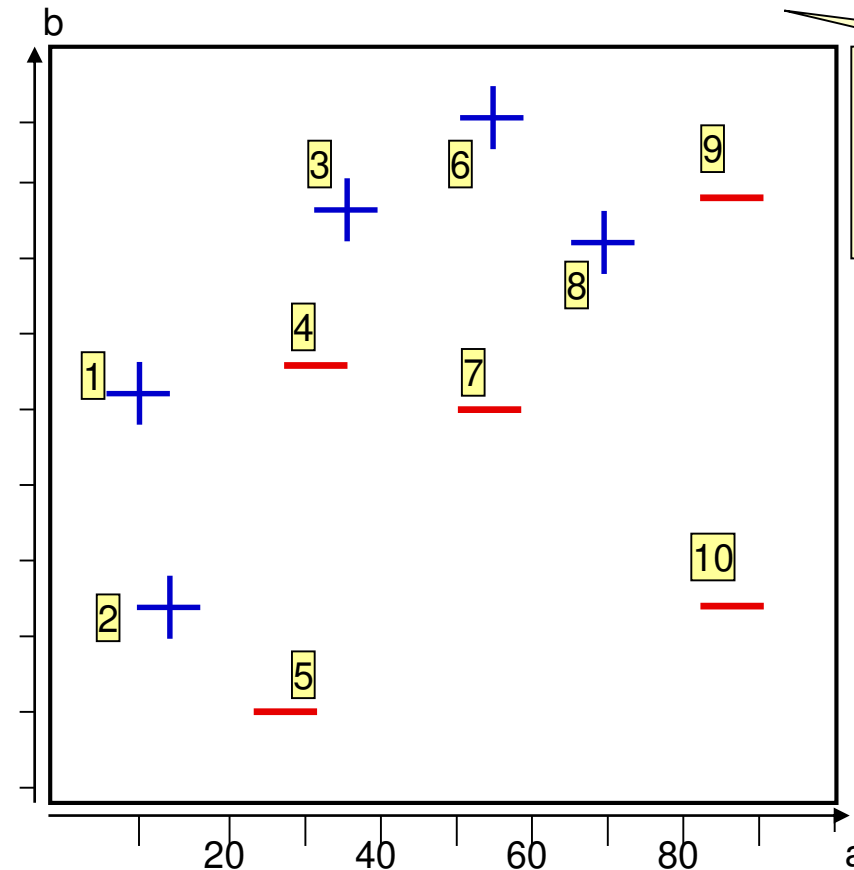
Die spezielle Variante **AdaBoost** (Adaptive Boosting):

- M Iterationen für M Hypothesen durch M einfache Lerner
- N Trainingsdatensätze

```
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 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 $w[j] = 1/N = 1/10 = 0.1$
- Jedes Beispiel x_j mit zwei Attributen a_j, b_j , also $x_j = (a_j, b_j)$, und Zuordnung $y_j \in \{+, -\}$
- Wir verwenden lineare achsenparallele Einzelklassifikatoren (z.B. Entscheidungstümpfe)

Entscheidungsstümpfe

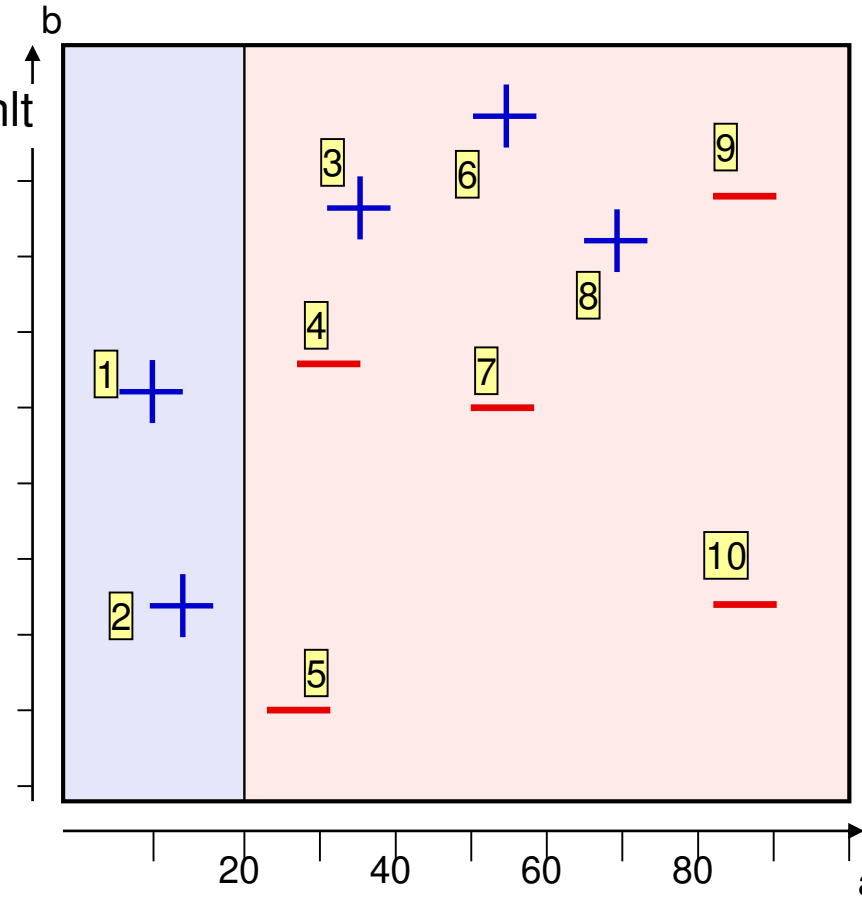
Ein Entscheidungstumpf wird wie folgt bestimmt:

- 1) Zunächst wird das Attribut bzw. die **Dimension d** randomisiert gewählt
- 2) Dann werden k (k vorgegeben) **Schwellwerte s_i** ($1 \leq i \leq k$) randomisiert für die Klassifikation gewählt
- 3) Über den Informationgewinn (Gain) wird **s_{best}** als bester der k Schwellwerte gewählt

Im Bspl.: $d = a$, $s_{best} = 20$ mit

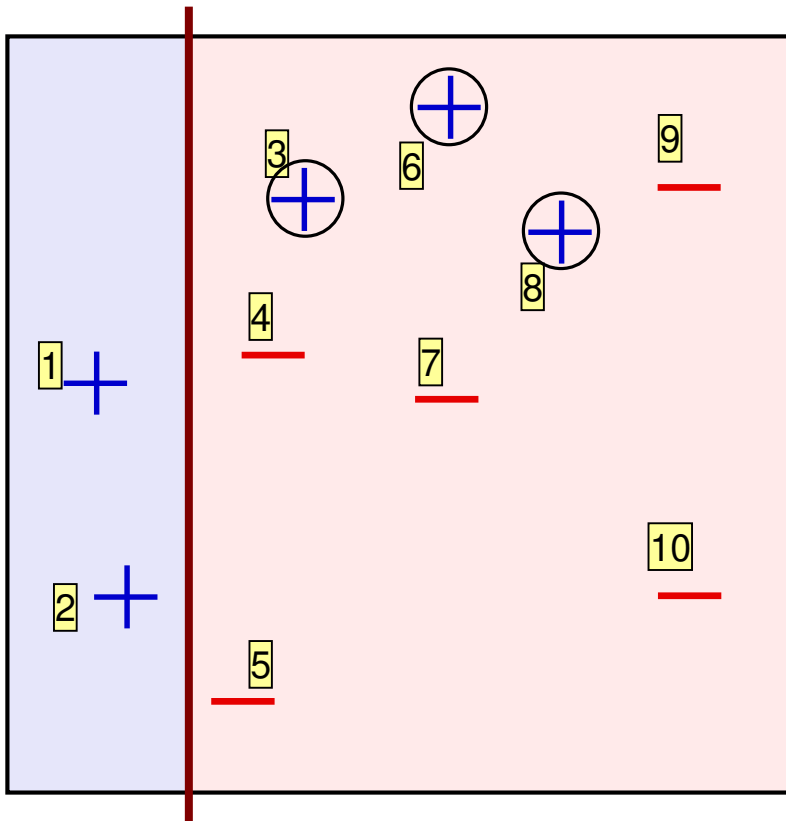
$$\text{Gain}(s_i) = 1 - [0.2 \cdot H(1,0) + 0.8 \cdot H(3/8, 5/8)]$$

Dieser Entscheidungstumpf klassifiziert also über die Parameterwahl $(a, 20)$.

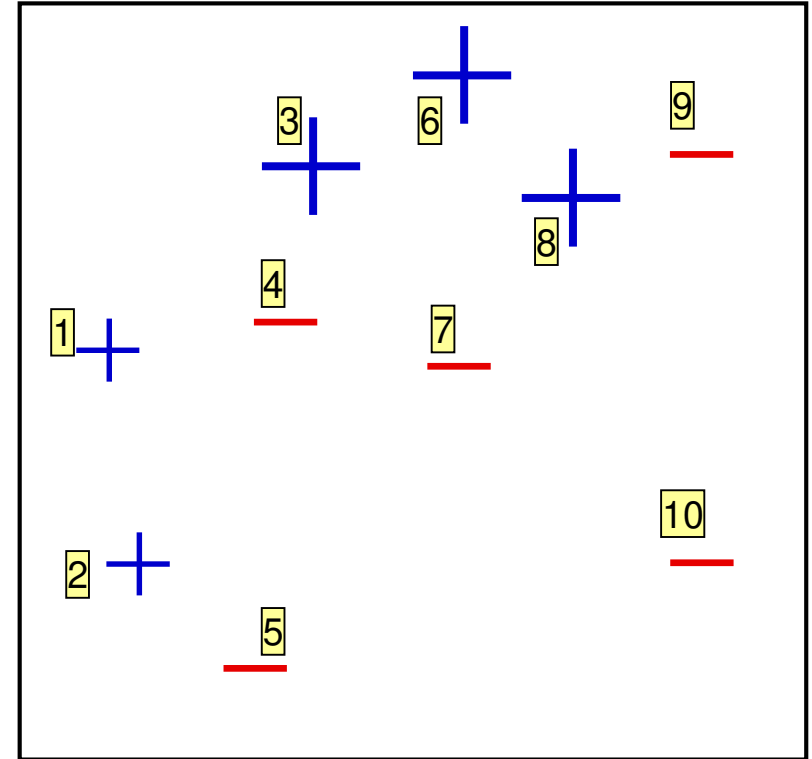


AdaBoost Beispiel (2)

Hypothese 1



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



Neue Beispielgewichte

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

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

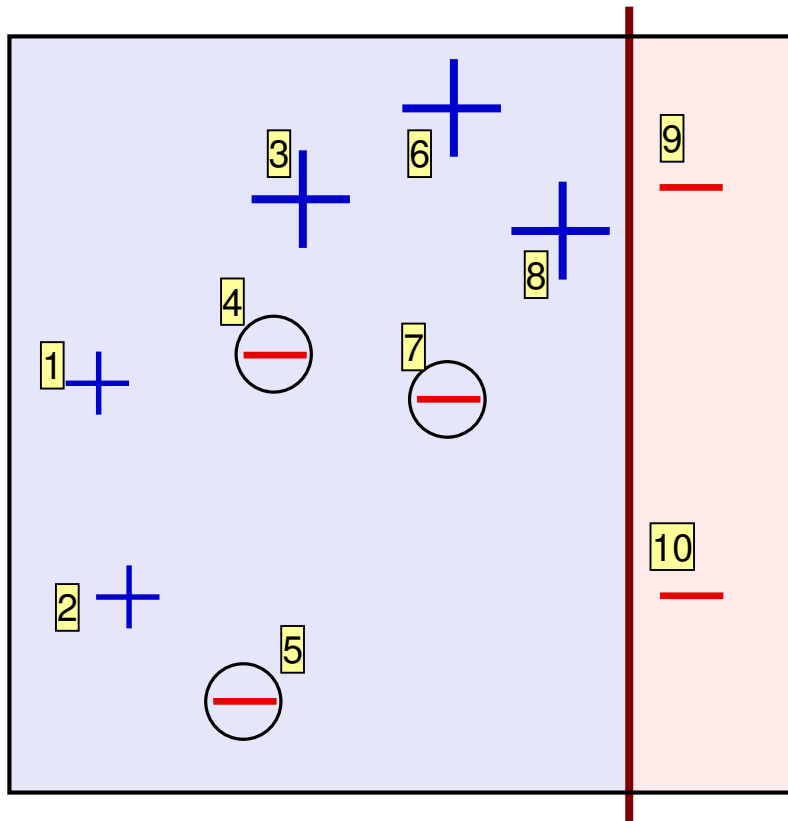
$\leadsto w[j] = 0.1 \cdot 0.429 = 0.0429$ für $j = 1, 2, 4, 5, 7, 9, 10$

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

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

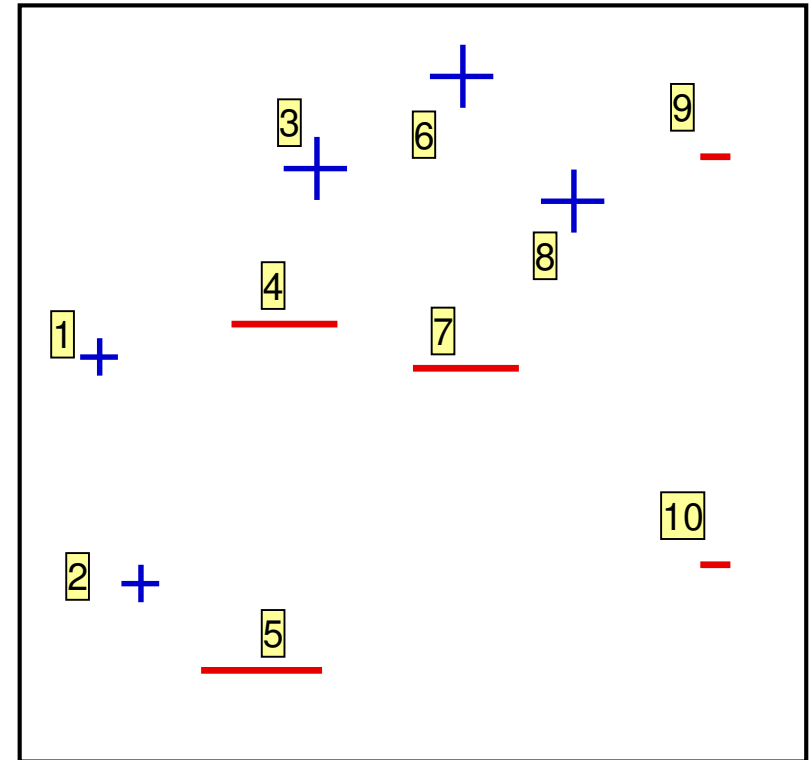
AdaBoost Beispiel (3)

Hypothese 2



Hypothese $h[2]$ mit

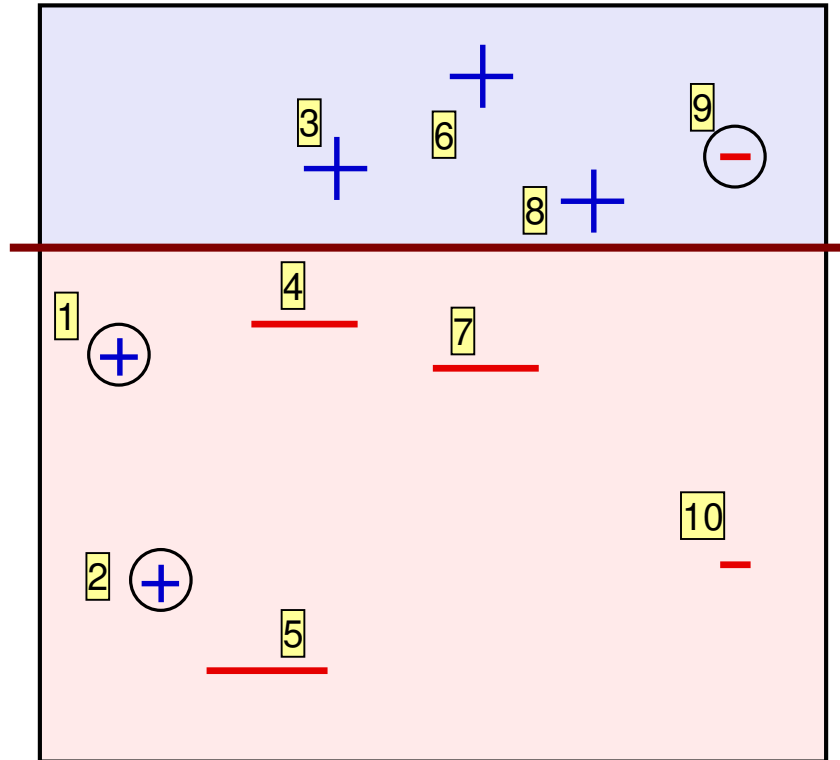
- Parameterwahl $(a, 80)$
- Gewicht $z[2] = 1.30$



Neue Beispielgewichte

Beispiel (4)

Hypothese 3



Hypothese $h[3]$ mit

- Parameterwahl $(b, 70)$
- Gewicht $z[3] = 1.85$

Beispiel (5)

Gewichtete Gesamthypothese

h_{final}

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

Lernkurven

Restaurantbeispiel mit Entscheidungstümpfen sowie Boosting über Entscheidungstümpfen:

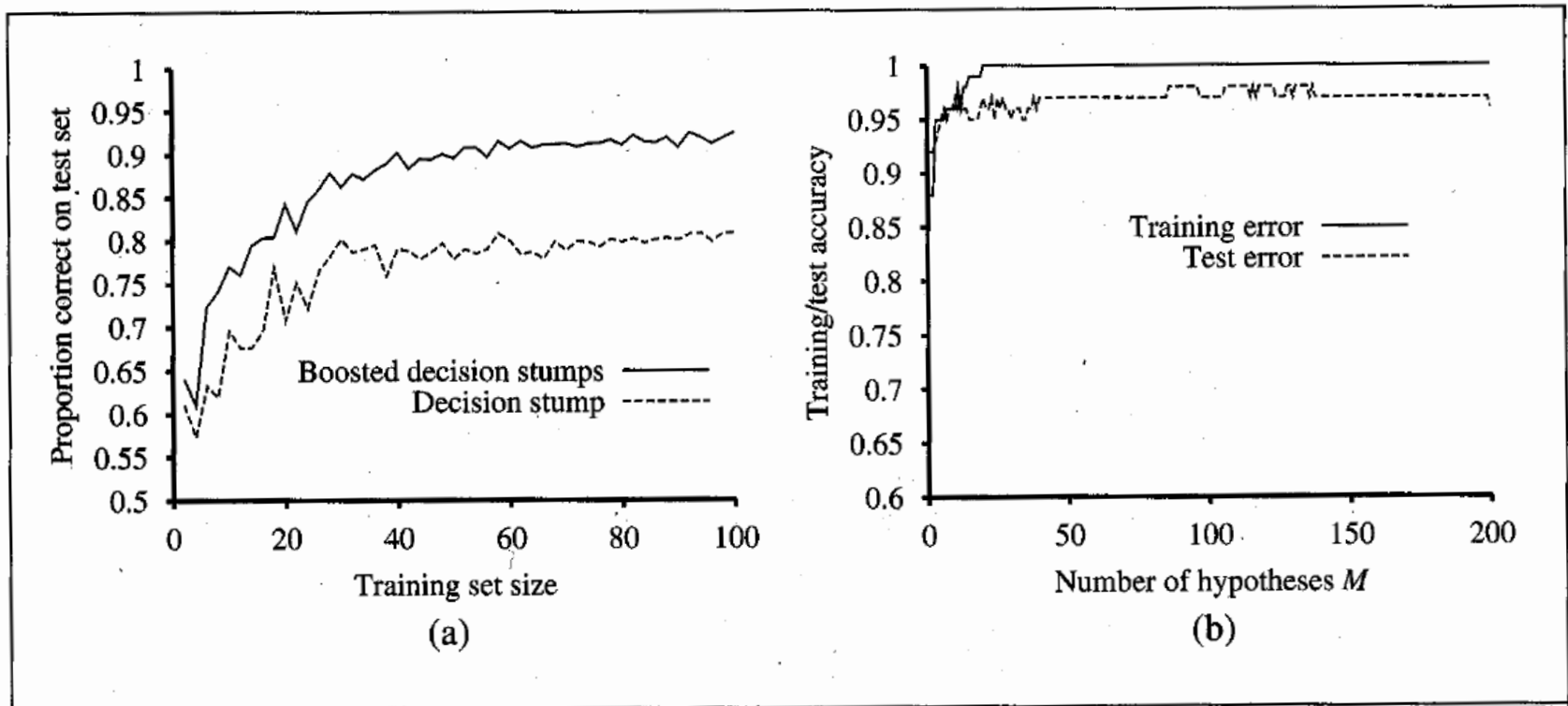


Figure 18.11 (a) Graph showing the performance of boosted decision stumps with $M = 5$ versus decision stumps on the restaurant data. (b) The proportion correct on the training set and the test set as a function of M , the number of hypotheses in the ensemble. Notice that the test set accuracy improves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

Randomisierte Entscheidungswälder

Randomisierte Entscheidungswälder (Random Forests) haben sich als Ensemble-Verfahren bes. im Computersehen mit vielen Varianten (s.u.) etabliert

Entscheidungswälder zeigen folg. Stärken:

1. Konzeptuelle Einfachheit
2. Keine “black magic”
3. Keine komplexen Optimierungen nötig
4. Viele Varianten für unterschiedliche Aufgaben:
Regression, halbüberwachtes Lernen, On-line-Lernen, etc.

-
1. Gall, Yao, Razavi, van Gool, Lempitsky: Hough Forests for Object Detection, Tracking, and Action Recognition. IEEE T-PAMI (11): pp. 2188-2202, 2011.
 2. Nowozin, Rother, Bagon, Sharp, Yao, Kohli. Decision tree fields. IEEE ICCV, pp. 1668-1675, 2011.
 3. Schuster, Wohlhart, Leistner, Saffari, Roth, Bischof. Alternating Decision Forests. IEEE CVPR, 2013.

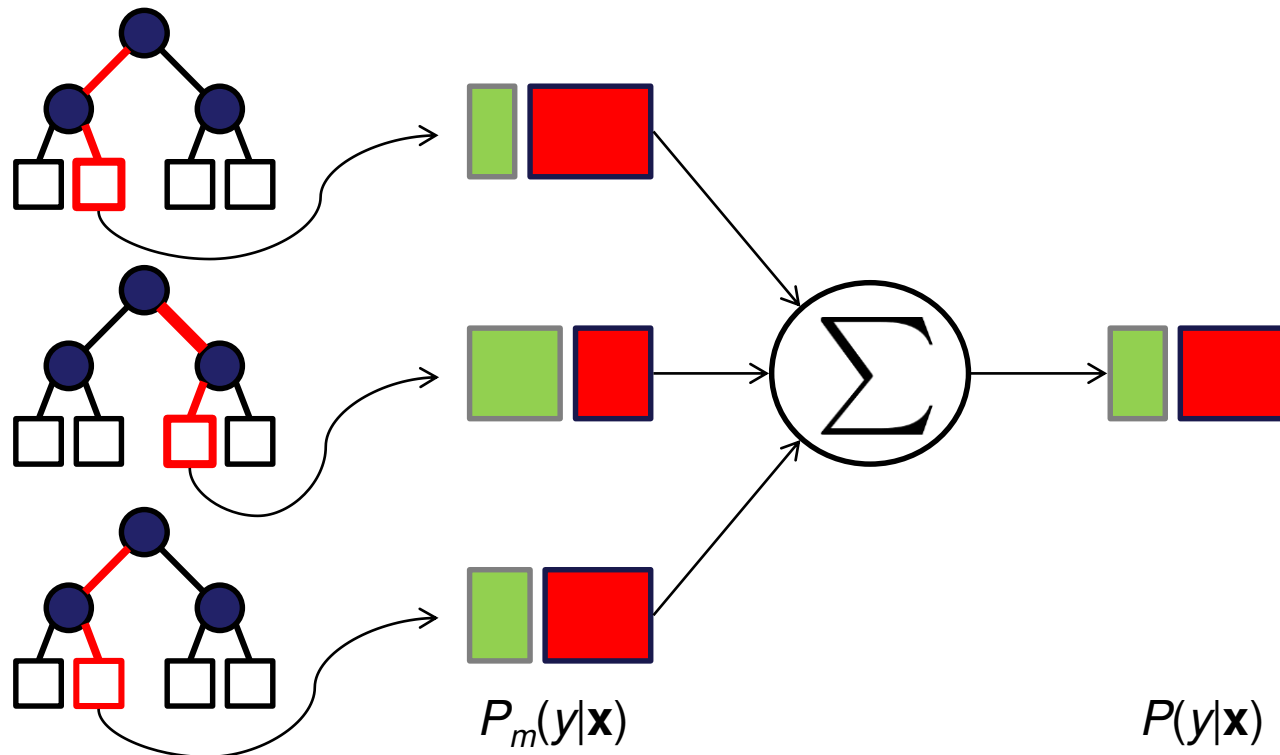
Randomisierte Entscheidungswälder

Randomisierte Entscheidungswälder werden wie folgt genutzt:

- **Bagging** (Bootstrap Aggregating): Es werden M verschiedene Bäume auf M unterschiedliche Trainingsmengen durch Bootstrap Sampling aus einer Originalmenge mit $N > M$ Beispielen gelernt.
- **Random Subspace Method**: Bei K Attributen (Merkmale oder Dimensionen) der Beispiele werden an jedem Knoten im Baum $k \ll K$ Attribute zufällig gewählt, die zur Betrachtung des Entscheidungsknotens herangezogen werden.
- **Random Split Method**: Zusätzlich können Entscheidungsschwellwerte für Attribute (s. Folie 23) zufällig gewählt werden anstatt diese nach dem Prinzip der Maximierung des Informationsgewinns zu bestimmen.

Klassifikation durch rand. Entscheidungswälder

- **Entscheidungswald** = Ensemble von M **Entscheidungsbaumen**
- Jeder Entscheidungsbaum erzeugt eine Prädiktion $P_m(y|\mathbf{x})$, $m \in 1, \dots, M$
- Die finale Prädiktion des **Entscheidungswaldes** ist
 - Mehrheitsentscheid ODER
 - $P(y|\mathbf{x}) = M^{-1} \sum_m P_m(y|\mathbf{x})$



Zusammenfassung

- *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 durch Gewichtung der Trainingsbeispiele.
 - *Randomisierte Entscheidungswälder* nutzen nach dem Bagging-Prinzip mehrere verschiedene, unkorrelierte Entscheidungsbäume zur Klassifikation.