Übersicht

Zeitplan:

- 1 Doppelstunde für 9.1, 9.2
- 1 Doppelstunde für 9.3 und ein Teil von 9.4, 9.5.

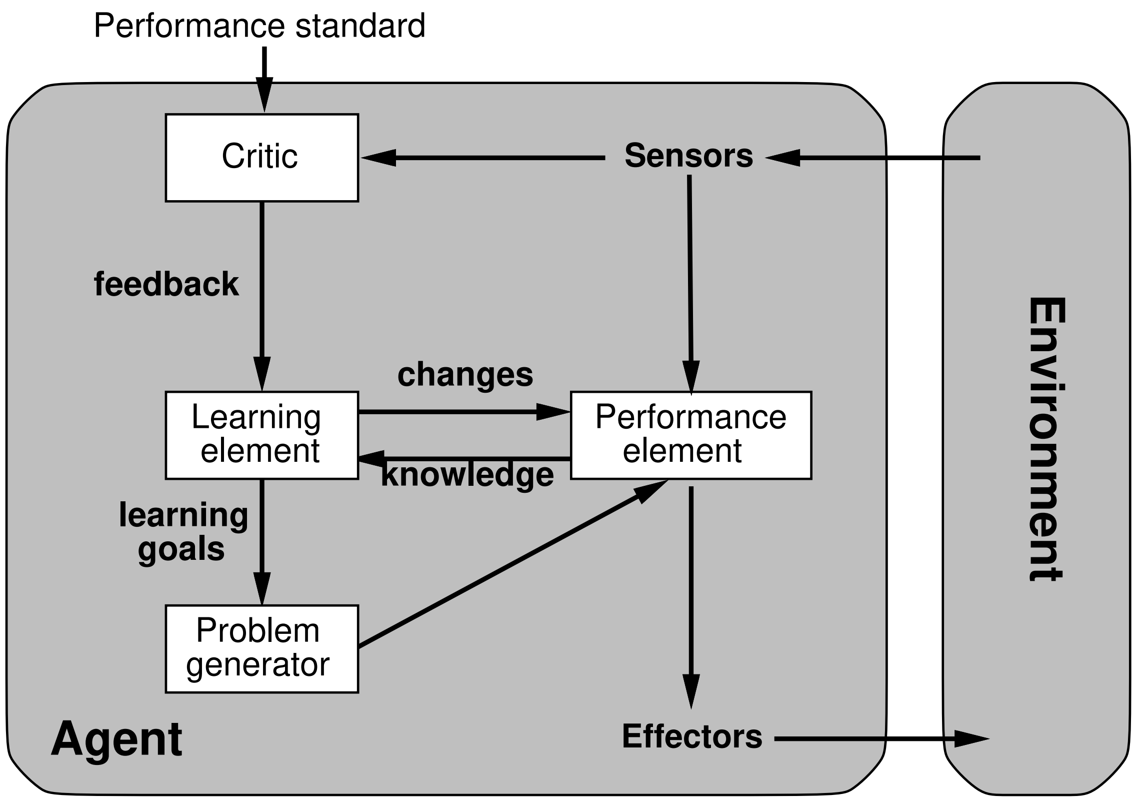
Lernen im allgemeinen

Bisher: Die Intelligenz eines Agenten steckt in seinem Programm, sie ist fest verdrahtet.

Jetzt: Der Agent soll autonomer **werden**, durch eigene **Beobachtungen** (Erfahrungen) soll er seine Umgebung verstehen lernen.

Wichtig: Falls der Bereich in dem er arbeitet nur unvollständig beschrieben werden kann.

8.1 Grundlagen



Performance-Element:

Kritik:

Lern-Element:

Problem Generator:

Beispiel: Taxi fahren.

Performance-Element: Beobachtet und wählt Aktionen aus. War bisher der gesamte Agent.

Kritik: Beobachtet das Resultat der Aktionen und bewertet sie. Bewertet anhand eines Standards **von aussen**!! (Warum?: sonst wählt er seinen eigenen Standard so niedrig, dass er ihn imer erfüllt!)

Lern-Element: Modifiziert das Performance-Element indem es die Kritik berücksichtigt (und den inneren Aufbau des Perf.-Elem.). Außerdem stellt es neue Ziele auf (um die Effekte von Aktionen besser zu verstehen).

Problem Generator: Er schlägt vor Aktionen auszuführen, um die Goals des Learning-Elementes zu befriedigen. Dies müssen nicht immer die "besten" Aktionen sein (i. S. d. Perf.-Elem.): aber sie sollen **informativ** sein und neues Wissen über die Welt liefern.

Wovan hängt Lernen (also der Entwurf des Lern-Elementes) eigentlich ab?

1. Welche Komponenten des Performance-Elementes sollen verbessert werden?
2. Wie sind diese Komponenten repräsentiert?
   ($\leadsto$ 9.2: Entscheidungsbäume, $\leadsto$ 9.3: PL1-formalisierte Bereiche)
3. Welches feedback gibt es?
   - **Überwachtes L.:** Das Ausführen einer inkorrekten Aktion erzeugt als feedback die "richtige" Lösung (z.B. wie stark aufs Bremspedal treten). *Fahrlehrer*.
   - **Reinforcement L.:** Nur das Resultat wird wahrgenommen. Kritik sagt, ob es gut oder schlecht, aber nicht was richtig gewesen wäre.
   - **Nicht-Überwachtes L.:** Gar keine Hinweise über korrekte Aktionen.
4. Welche a-priori-Informationen gibt es?
   (Oft gibt es nützliches Hintergrundwissen.)

Komponenten des Performance Elementes

1. Abbildung von Bedingungen des aktuellen Zustandes in die Menge der Aktionen.
2. Ableiten relevanter Eigenschaften der Welt aus den Wahrnehmungen.
3. Informationen darüber, wie sich die Welt entwickelt.
4. Informationen über die Folgen von Aktionen.
5. Bewertungsfunktion über die Zustände der Welt.
6. Bewertungsfunktion über die Güte einzelner Aktionen in einem Zustand.
7. Beschreibung von Zustandsklassen, die die Nutzenfunktion des Agenten maximieren.

Alle diese Komponenten sind, mathematisch gesehen, **Funktionen**.

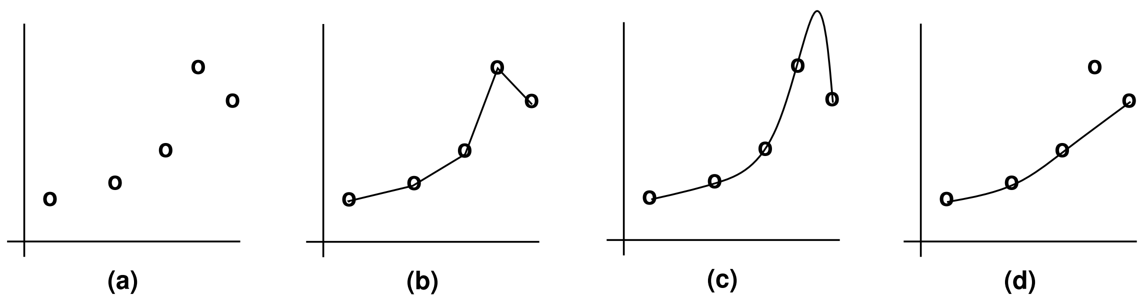
Lernen heißt diese Funktionen zu repräsentieren.

Induktives Lernen

Gegeben Paare (x, y). Bestimme f mit $f(x) = y$.

Beispiel 8.1 (Zahlenfolgen fortsetzen)
Wie lautet die nächste Zahl:

- *3, 5, 7, ?*
- *3, 5, 17, 257, ?*



Einfacher Reflex-Lern Agent:

global *examples* $\leftarrow \{\}$

---

function REFLEX-PERFORMANCE-ELEMENT(*percept*) **returns** an action

  if (*percept, a*) **in** *examples* **then return** *a*
  else
    h $\leftarrow$ INDUCE(*examples*)
    return *h*(*percept*)

---

procedure REFLEX-LEARNING-ELEMENT(*percept, action*)
  inputs: *percept*, feedback percept
      action, feedback action

  examples $\leftarrow$ *examples* $\cup$ $\{($ *percept*, *action*$)\}$

Beispiel 8.2 (Verifizieren und falsifizieren)
Man betrachte eine Menge von Karten. Jede Karte enthält auf einer Seite einen Buchstaben und auf der anderen Seite eine Zahl. Nachdem man sich einige Karten angeschaut hat kommt man zu der Hypothese

Ist auf einer Seite einer Karte ein Vokal, so ist auf der anderen Seite eine gerade Zahl.

Es liegen nun folgende Karten auf dem Tisch:

A T 4 7.

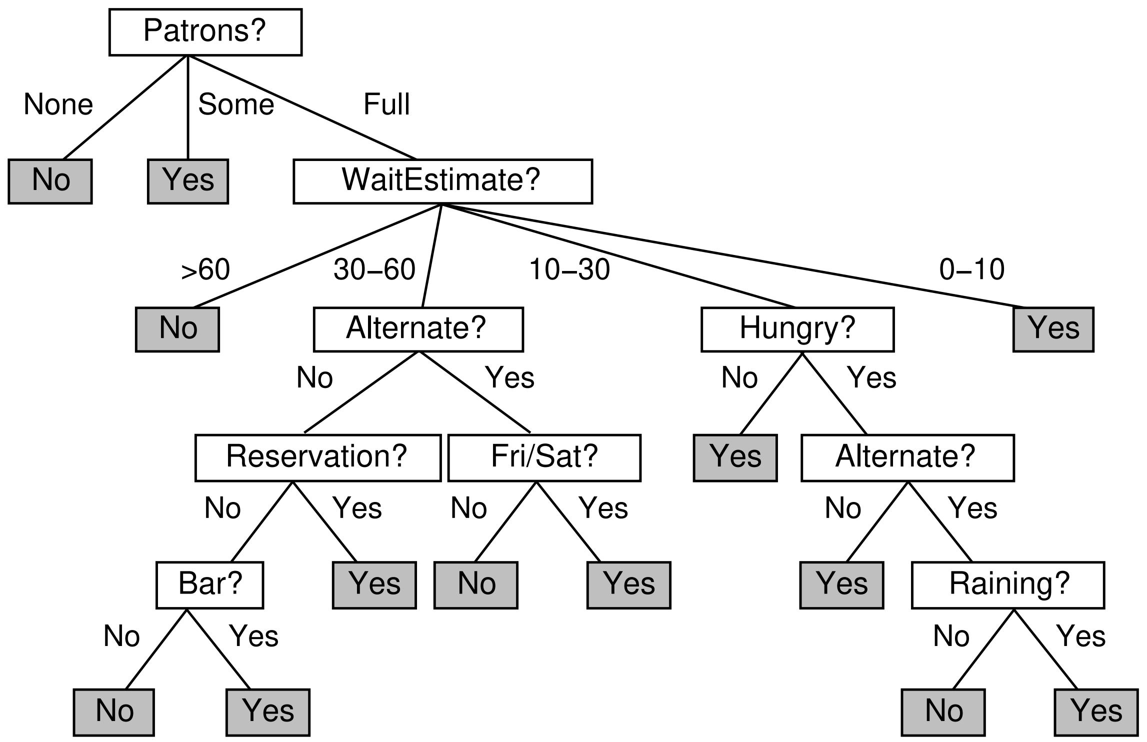
Sie dürfen höchstens 2 Karten herumdrehen und sollen die Hypothese prüfen. Welche Karte drehen Sie herum?

Es müssen die **A** und die **7** herumgedreht werden. Die **A** weil damit die Hypothese direkt überprüft werden kann, die **7** weil das die einzige Möglichkeit ist, die Hypothese auch zu falsifizieren. Dies wird oft nicht gemacht: man sucht nur nach Bestätigung, aber man versucht nicht die Hypothese zu widerlegen.

8.2 Entscheidungsbäume

Entscheidungsbäume sind Repräsentanten von booleschen Funktionen.

Beispiel: Man geht essen und erreicht ein Restaurant. Soll man auf einen freien Tisch warten oder weitergehen?



Entscheidungsbaum = Konjunktion von Implikationen[a]

$$\forall r\ (Patrons(r, Full) \wedge Wait\_estimate(r, 10-30) \wedge \neg Hungry(r)) \longrightarrow Will\_Wait(r)$$

Dies ist zwar in PL! geschrieben, aber Entscheidungsb'aume betreffen immer nur einzelne Objekte (das r oben), dessen Eigenschaften abgefragt werden. Also geht es um Aussagenlogik.

Kann jede boolesche Funktion als Entscheidungsbaum geschrieben werden?

Ja: Zeile der Wahrheitstabelle entspricht Pfad im Baum.

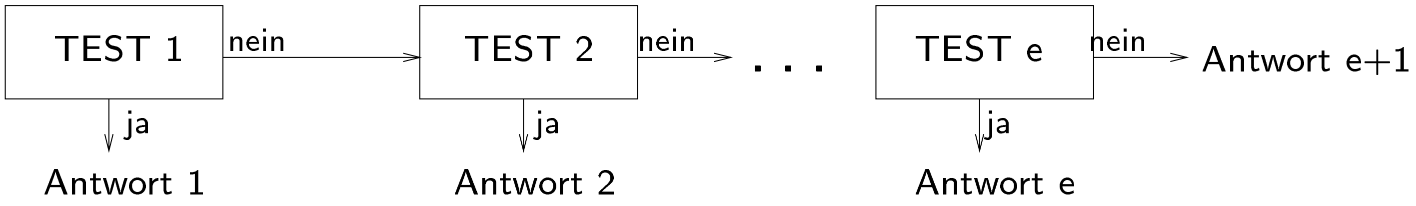
[a] Implikation = Jeder Pfad zu einem Blatt.

Wichtig: E-Bäume können sehr viel kleiner sein! Aber es gibt auch bool. Funktionen, die nur durch exponentiell große Bäume dargestellt werden können:

$$\text{Paritätsfunktion:}par(x_1, \ldots, x_n) := \begin{cases} 1, & \text{falls } \sum_{i=1}^{n} x_i \text{ ist gerade} \\ 0, & \text{sonst} \end{cases}$$

Eine Variante der E-Bäume sind

Beispiel 8.3 (Entscheidungslisten)
Alle Attribute seien booleschwertig. Eine Entscheidungsliste hat die Form



wobei $Antwort_i \in \{Ja, NEIN\}$ *und* $Test_i$ *eine Konjunktion von (eventuell negierten) Attributen ist. (*$\leadsto$ *Aufgabe zum Vergleich E-Listen und E-Bäume).*

Sei $k - DL(n)$ *die Menge der booleschen Funktionen in* n*-Attributen, die durch E-Listen mit höchstens* k *Abfragen in jedem Test dargestellt werden können.*

Offensichtlich gilt:

- $n - DL(n) =$ Menge aller booleschen Funktionen,
- $card(n - DL(n)) = 2^{2^n}$.

Dass gilt $card(n - DL(n)) = 2^{2^n}$, sieht man anhand der Wahrheitstafel: für eine spezielle Funktion muss man 2^n Zeilen festlegen um sie zu definieren. In jeder Zeile gibt es 2 Möglichkeiten (0 oder 1).

Oft mache ich noch: Gibt es einen Unterschied zwischen $2^{(2^n)}$ und $(2^2)^n$? Offensichtlich ist doch

$$2^{(2^2)} = 2^4 = 16 = 4^2$$

Wie lernt man denn nun E-Bäume?

Example	Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	*Yes*
X_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	*No*
X_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	*Yes*
X_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*10–30*	*Yes*
X_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	*No*
X_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	*Yes*
X_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	*No*
X_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	*Yes*
X_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	*No*
X_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	*No*
X_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	*No*
X_{12}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	*Yes*

Die Menge der Beispiele aus denen gelernt werden soll heißt **Trainingsmenge**. Beispiele können positiv (Attribut trifft zu) oder negativ (trifft nicht zu) sein.

Triviale Lösung des Lernens: Die Pfade im Baum sind genau die Beispiele. **Nachteil:** Neue Fälle können nicht behandelt werden.

Idee: Nimm den einfachsten Baum (bzw. den allgemeinsten) der mit den Beispielen verträglich ist.

Ockham's Messer: Entia non sunt multiplicanda praeter necessitatem.

Beispiel 8.4 (Raten Sie!)

Ein Computerprogramm kodiert Zahlentripel gemäß einer bestimmten Vorschrift. Sie sollen diese Vorschrift herausfinden.

Sie geben dem Programm Tripel (x_1, x_2, x_3) *Ihrer Wahl vor (*$X_i \in \mathbb{N}$*) und bekommen als Antwort "Ja" oder "Nein".*

Zur Vereinfachung gibt Ihnen das Programm schon zu Anfang bekannt, daß folgende Tripel in der Menge enthalten sind:

$$(4, 6, 8),\ (6, 8, 12),\ (20, 22, 40)$$

Stellen Sie weitere Anfragen (etwa 10) und versuchen Sie die Eigenschaft herauszufinden.

Das Computerprogramm ist natürlich der Vortragende. Die Lösung

$$\{(x, y, z) : \ x < y < z\}$$

wurde vorher schon an die Tafel (hinter einer anderen Tafel versteckt) angeschrieben.

Das Problem ist, daß man nur Tripel der Form

$$(x, x + 2, 2x)$$

ausprobiert, also nur positive Beispiele versucht. Man ist zu speziell und versucht nicht, die einmal gefasste Hypothese kritisch zu hinterfragen, also sie zu falsifizieren. Man fragt nach einer speziellen Vorschrift, nicht nach der allgemeinsten.

Idee des Lern-Algorithmus

Ziel: Möglichst kleiner Baum. Teste zunächst die *wichtigsten* Attribute (um eine möglichst schnelle Klassifikation zu bekommen). *Dies formalisieren wir später, unter Rückgriff auf etwas Informationstheorie..*

Fahre dann rekursiv so fort: d.h. es gibt immer weniger Beispiele und Attribute.

Wir mache folgende Fallunterscheidungen:

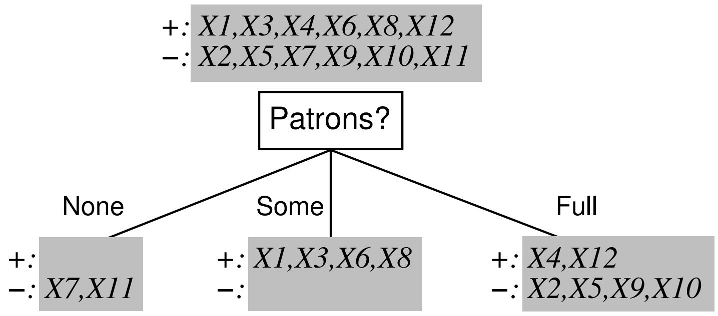
1. Fall: *Es gibt positive und negative Beispiele.*
   Wähle das beste Attribut.

2. Fall: *Es gibt nur positive oder nur negative Beispiele.*
   Fertig: Eindeutige Antwort ist möglich.

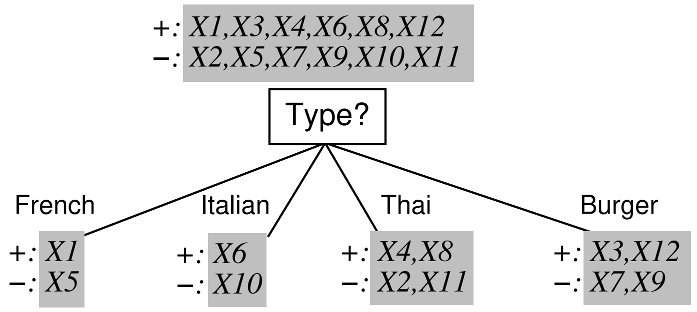
3. Fall: *Es gibt keine Beispiele mehr.*
   Dann muss man einen Default-Wert nehmen, etwa die Mehrheit der Beispiele des Elter-Knotens.

4. Fall: *Es gibt sowohl positive als auch negative Beispiele, aber es gibt keine Attribute mehr.*
   Dann reichen die zugrundeliegenden Attribute eben nicht aus, um zu einer Entscheidung zu kommen. Es ist einfach nicht genügend Information gegeben.

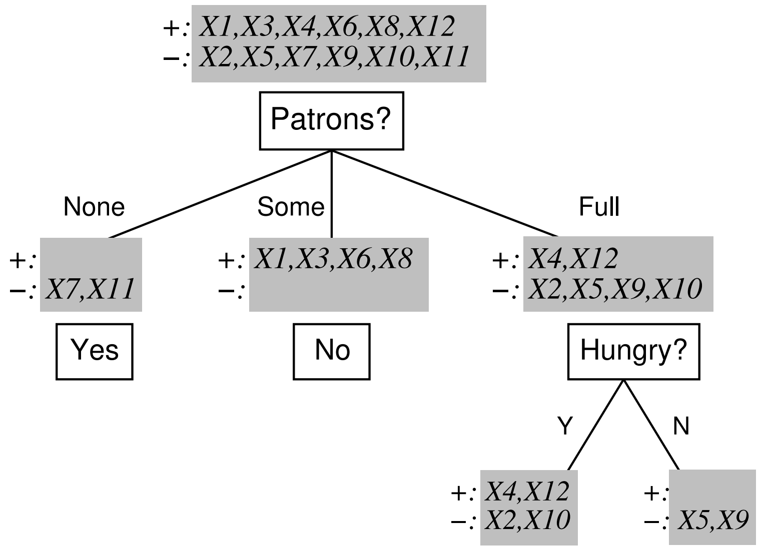
(a)


+: X1,X3,X4,X6,X8,X12
−: X2,X5,X7,X9,X10,X11
Patrons?
None
Some
Full
+:
−: X7,X11
+: X1,X3,X6,X8
−:
+: X4,X12
−: X2,X5,X9,X10

(b)

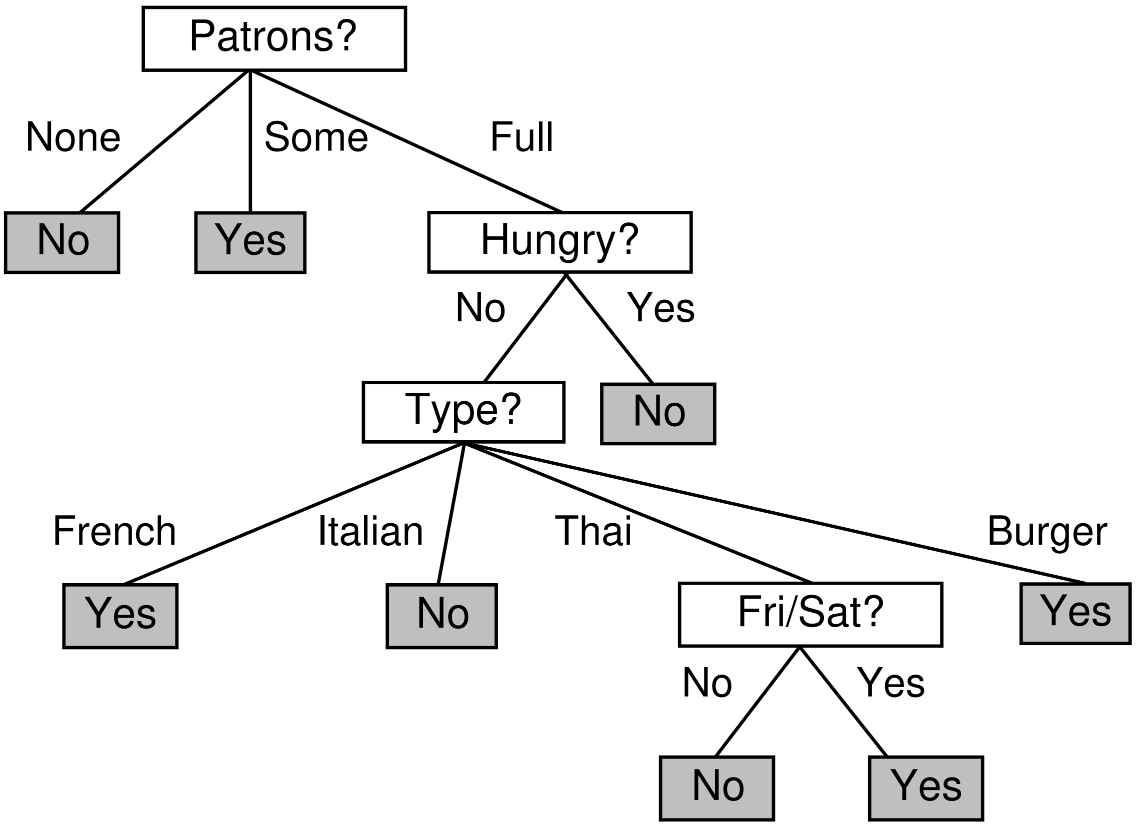

+: X1,X3,X4,X6,X8,X12
−: X2,X5,X7,X9,X10,X11
Type?
French
Italian
Thai
Burger
+: X1
−: X5
+: X6
−: X10
+: X4,X8
−: X2,X11
+: X3,X12
−: X7,X9

(c)


+: X1,X3,X4,X6,X8,X12
−: X2,X5,X7,X9,X10,X11
Patrons?
None
Some
Full
+:
−: X7,X11
+: X1,X3,X6,X8
−:
+: X4,X12
−: X2,X5,X9,X10
Yes
No
Hungry?
Y
N
+: X4,X12
−: X2,X10
+:
−: X5,X9

function DECISION-TREE-LEARNING(*examples, attributes, default*) **returns** a decision tree
  inputs: *examples*, set of examples
    attributes, set of attributes
    default, default value for the goal predicate

  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** MAJORITY-VALUE(*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 ← DECISION-TREE-LEARNING($examples_i$, *attributes* − *best*,
        MAJORITY-VALUE(*examples*))
      add a branch to *tree* with label v_i and subtree *subtree*
    end
    return *tree*

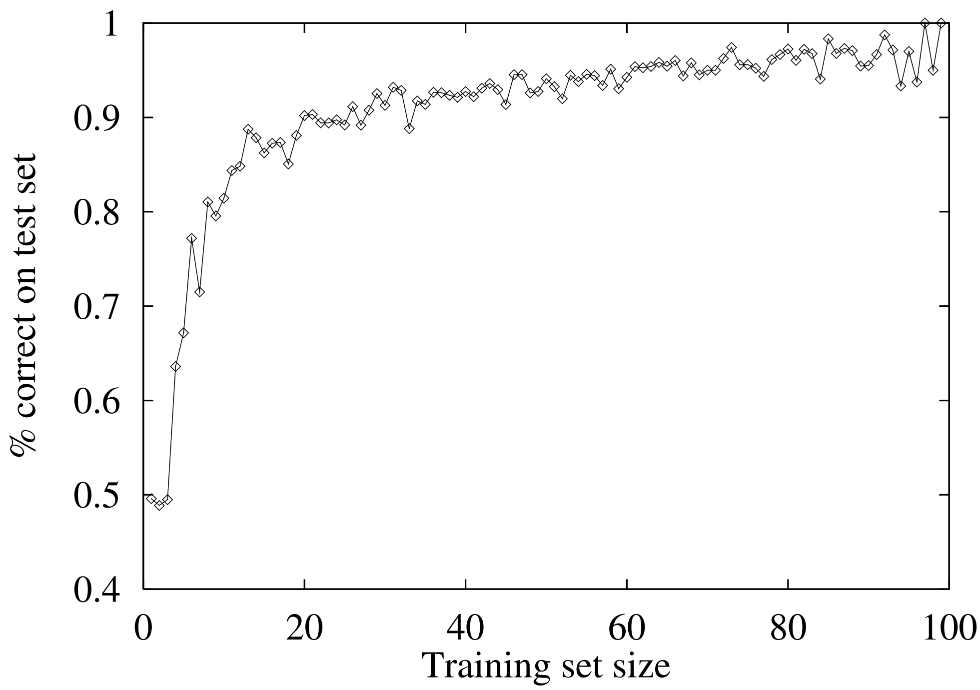


Der Algorithmus entwirft einen möglichst einfachen Baum, der mit den gegebenen Beispielen konsistent ist.

Frage: Wie gut ist der generierte Baum? Wie weit ist er von dem “tatsächlichen” Baum entfernt? Kann man a-priori Abschätzungen geben? ($\rightsquigarrow$ Kapitel 9.4).

Empirisches Vorgehen

1. Wähle eine Menge M_{Bsp} von Beispielen aus.
2. Teile diese in 2 Mengen auf: $M_{Bsp} = M_{Trai} \cup M_{Test}$.
3. Wende Lernalgorithmus auf M_{Trai} an und erhalte eine Hypothese H.
4. Bestimme den Anteil der Elemente aus M_{Test} die korrekt klassifiziert werden.
5. Wiederhole 1.-4. für viele $M_{Trai} \cup M_{Test}$ mit zufällig erzeugten M_{Trai}.



Informationstheorie

Wie wählt man das “beste” Attribut aus? Das beste Attribut ist dasjenige, das die größtmögliche Information liefert.

Beispiel: Fairer Münzwurf.

Definition 8.5 (1 Bit, Information)
1 Bit ist die Information die man braucht, um bei einem fairen Münzwurf das richtige Ergebnis anzusagen.

Allgemeiner: Angenommen, ein Experiment kann n-Ausgänge $v_1, \ldots, v_n$ haben. Jeder Ausgang v_i ergibt sich mit einer Wahrscheinlichkeit $P(v_i)$. Die Information die im Ergebnis (dem ausführen des Experimentes) steckt, ist definiert als

$$\mathbf{I}(P(v_1), \ldots, P(v_n)) := \sum_{i=1}^{n} -P(v_i) log_2 P(v_i)$$

Angenommen eine Münze ist präpariert. Zu 90 % kommt Kopf. Dann ist

$$\mathbf{I}(0.1, 0.9) = \ldots \approx 0.47$$

FRAGE: Für jedes Attribut A: Wenn man diese Attribut auf der momentanen Trainigsmenge auswertet, wieviel Information gewinnt man dadurch?

Das "beste" Attribut ist das mit der größtmöglichen Information!

Angenommen A hat ν verschiedene Ausgänge (Patron: 3, Hungry: 2, Type: 4) und teilt M_{Trai} in $E_1, \ldots, E_\nu$ auf. Jedes E_i enthalte p_i positive und n_i negative Beispiele. In jedem Zweig brauchen wir also noch

$$\mathbf{I}(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i})$$

Information um jedes Beispiel eindeutig zu klassifizieren.

Ein Beispiel ist in E_i mit der Wahrscheinlichkeit $\frac{p_i+n_i}{p+n}$ (wir nehmen an, daß M_{Trai} genau p positive und n negative Beispiele enthält).

Die Information um jedes Beispiel aus $M_{Trai} = \bigcup_i^\nu E_i$ zu klassifizieren ist

$$\text{Fehlende\_Inf}(A) = \sum_{i=1}^{\nu} \frac{p_i + n_i}{p + n} \mathbf{I}(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i})$$

Also gewinnen wir durch den Test des Attributes A folgende Information

$$\text{Gewinn}(A) = \mathbf{I}(\frac{p}{p + n}, \frac{n}{p + n}) - \text{Fehlende\_Inf}(A)$$

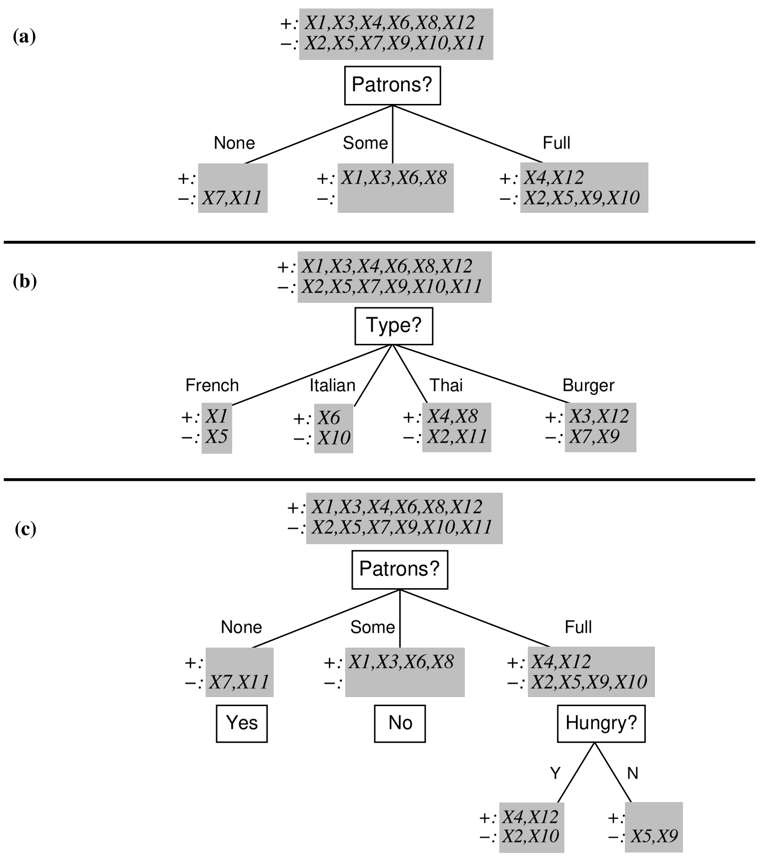
Choose_Attribute wählt das A aus mit maximalem Gewinn(A).

Also gewinnen wir durch den Test des Attributes A folgende Information

$$\text{Gewinn}(A) = \mathbf{I}\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - \text{Fehlende\_Inf}(A)$$

Choose_Attribute wählt das A aus mit maximalem Gewinn(A).

Aus Abb. 18.6 ergibt sich Gewinn(Patron)≈ 0.54. Berechnen Sie Gewinn(Type), Gewinn(Hungry) (Hungry als erstes Attribut), Gewinn(Hungry) (wenn vorher Patron ausgewählt wurde).



8.3 PL1 formalisierte Bereiche

Ziel: Allgemeinerer Rahmen.

Idee: Lernen ist Suchen im Hypothesen-Raum. ($\leadsto$ Planen)

Zielprädikat: $Q(x)$, einstellig (bisher: $Will\_Wait$)

Wir suchen eine Definition von $Q(x)$, d.h. eine Formel $C(x)$ mit

$$\forall x \; Q(x) \leftrightarrow C(x)$$

Genauer: Jedes Beispiel X_i stellt eine Menge von Bedingungen dar, unter denen $Q(X_i)$ gilt oder nicht. Wir suchen eine Erklärung dafür: eine Formel $C(x)$, die die in den Beispielen auftretenden Prädikate benutzt. Z.B.:

$$
\begin{array}{llcll}
\forall r & Will\_Wait(r) & \leftrightarrow & & Patrons(r, Some) \\
& & & \vee & (Patrons(r, Full) \wedge \neg Hungry(r) \wedge Type(r, French)) \\
& & & \vee & (Patrons(r, Full) \wedge \neg Hungry(r) \wedge Type(r, Thai) \wedge Fri\_Sat(r)) \\
& & & \vee & (Patrons(r, Full) \wedge \neg Hungry(r) \wedge Type(r, Burger))
\end{array}
$$

Definition 8.6 (Hypothese, Kandidatenfunktion)
Eine Formel $C_i(x)$ mit $\forall x C(x) \leftrightarrow C_i(x)$ heißt Kandidatenfunktion. Die gesamte Formel heißt Hypothese:

$$H_i : \quad \forall x C(x) \leftrightarrow C_i(x)$$

Definition 8.7 (Hypothesenraum $\mathcal{H}$)
Der Raum aller Hypothesen $\mathcal{H}$ eines Lern-Alg. ist die Menge all jener Hypothesen, die der Algorithmus bilden kann.

Unter der Ausprägung einer Hypothese H bezüglich des Zielprädikates Q verstehen wir die Menge der Beispiele, Die H erfüllen (sie bestätigen H).

Hypothesen mit unterschiedlicher Ausprägung sind zusammen inkonsistent.

Hypothesen mit gleicher Ausprägung sind logisch äquivalent.

Allg. Situation:

- Wir haben eine Menge von Beispielen $\{X_1, \dots, X_n\}$. Jedes Beispiel X beschreiben wir durch einen Satz und die Angabe $Q(x)$ (trifft zu) oder $\neg Q(x)$ (trifft nicht zu).

Example	Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	*Yes*
X_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	*No*
X_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	*Yes*
X_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*10–30*	*Yes*
X_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	*No*
X_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	*Yes*
X_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	*No*
X_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	*Yes*
X_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	*No*
X_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	*No*
X_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	*No*
X_{12}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	*Yes*

X_1 ist beschrieben durch

$$Alt(X_1) \wedge \neg Bar(X_1) \wedge \neg Fri\_Sat(X_1) \wedge Hungry(X_1) \wedge \dots$$

und $Will\_Wait(X_1)$. $\mathcal{H}$ ist die Menge der Entscheidungsbäume.

- Die Trainingsmenge ist die Menge all dieser Sätze.

- **Wir suchen eine Hypothese, die konsistent mit der Trainingsmenge ist.**

Unter welchen Bedingungen ist eine Hypothese inkonsistent mit einem Beispiel X?

Falsch negativ: Hypothese sagt nein ($\neg Q(X)$) aber in Wirklichkeit gilt '$Q(X)$.

Falsch positiv: Hypothese sagt ja ($Q(X)$) aber in Wirklichkeit gilt '$\neg Q(X)$.

Induktives Lernen in logik-basierten Bereichen heißt mit jedem Beispiel die Menge der möglichen Hypothesen weiter einschränken.

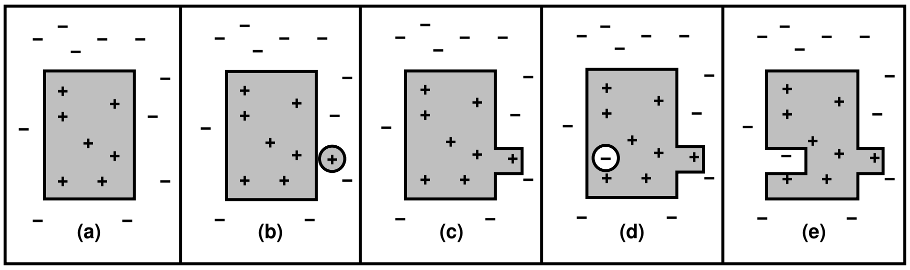
Für PL1 ist $\mathcal{H}$ im allgemeinen unendlich, daher ist automatisches Beweisen viel zu allgemein.

1. **Ansatz:** Wir behalten eine einzige Hypothese und modifizieren sie, falls die Beispiele mit ihr inkonsistent sind.

2. **Ansatz:** Wir behalten den ganzen Teilraum der noch mit den Beispielen konsistent ist (*Version-Space*. Dieser wird durch 2 Mengen effizient repräsentiert (so wie man mit [1,2] das entsprechende Intervall aller reellen Zahlen repräsentiert).

1. Ansatz: Current-Best-Hypothesis-Suche

Starte mit einfacher Hypothese H. Falls ein neues Beispiel damit konsistent: ok.
Falls es “falsch negativ”: Verallgemeinere H.
Falls es “falsch positiv”: Spezialisiere H.



Das ergibt einen Algorithmus:

function CURRENT-BEST-LEARNING(*examples*) **returns** a hypothesis

$H \leftarrow$ any hypothesis consistent with the first example in *examples*
for each remaining example in *examples* **do**
  if *e* is false positive for *H* **then**
    $H \leftarrow$ **choose** a specialization of *H* consistent with *examples*
  else if *e* is false negative for *H* **then**
    $H \leftarrow$ **choose** a generalization of *H* consistent with *examples*
  if no consistent specialization/generalization can be found **then fail**
end
return *H*

Wie verallgemeinert, bzw. spezialisiert man?

$$\begin{aligned} H_1 &: \quad \forall x \ (Q(x) \leftrightarrow C_1(x)) \\ H_2 &: \quad \forall x \ (Q(x) \leftrightarrow C_2(x)) \end{aligned}$$

H_1 verallgemeinert H_2, falls $C_2(x) \rightarrow C_1(x)$, H_1 spezialisiert H_2, falls $C_1(x) \rightarrow C_2(x)$.

Verallgemeinern bedeutet: in einer Konjunktion lässt man $\wedge$-Glieder weg. Spezialisieren bedeutet: in einer Disjunktion nimmt man $\vee$-Glieder hinzu.

2. Ansatz: Version-Space

```
function VERSION-SPACE-LEARNING(examples) returns a version space
   local variables: V, the version space: the set of all hypotheses

   V ← the set of all hypotheses
   for each example e in examples do
      if V is not empty then V ← VERSION-SPACE-UPDATE(V, e)
   end
   return V
-------------------------------------------------------------------
function VERSION-SPACE-UPDATE(V, e) returns an updated version space
   V ← {h ∈ V : h is consistent with e}
```

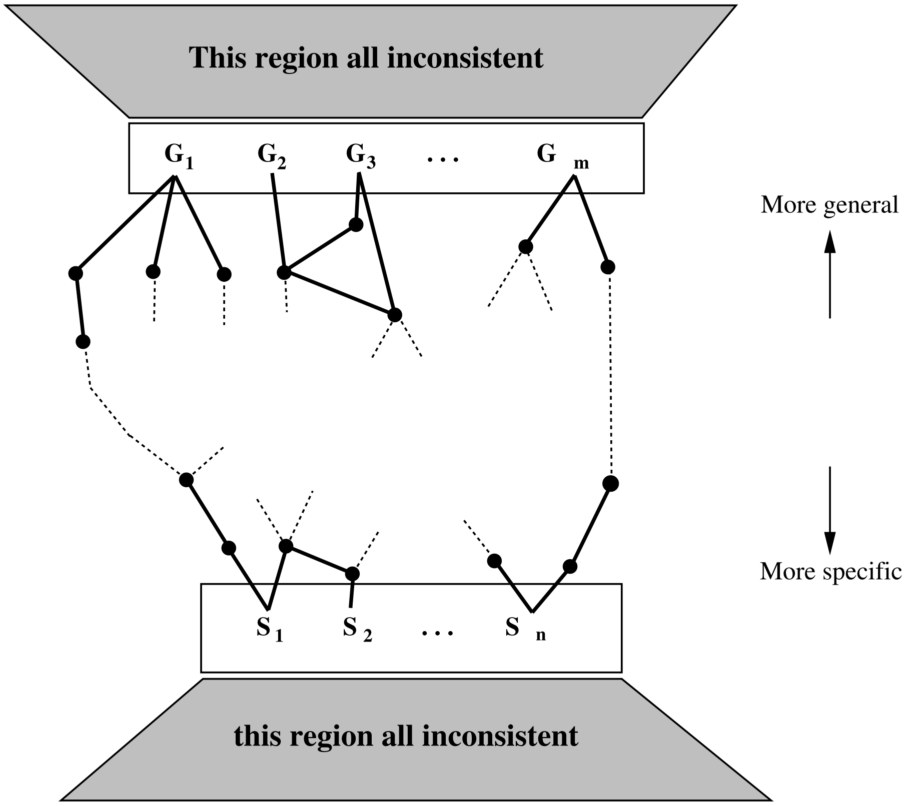
Problem: $\mathcal{H}$ ist eine große Disjunktion $H_1 \vee \ldots \vee H_n$. Wie soll man das repräsentieren?

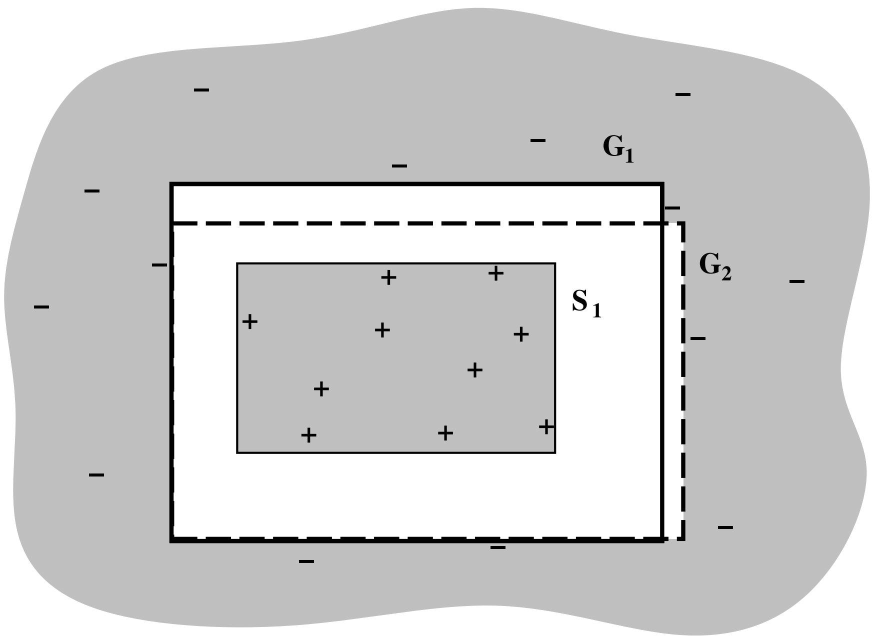
Wie repräsentiert man die Menge aller reellen Zahlen zwischen 0 und 1? Als Intervall [0,1].

Hier: Es gibt eine partielle Ordnung auf $\mathcal{H}$ (Verallg./Spezial.). Die Grenzen sind gegeben durch

- G-Menge: ist konsistent mit allen bisherigen Beispielen und es gibt keine konsistente Hypothese die allgemeiner ist,
- S-Menge: ist konsistent mit allen bisherigen Beispielen und es gibt keine konsistente Hypothese die spezifischer ist.

Werden neue Beispiele betrachtet, müssen die G- und S-Mengen in VERSION-SPACE-UPDATE geeignet modifiziert werden.





Was kann passieren?

1. Es bleibt nur eine Hypothese übrig: Hurra!
2. Der Raum kollabiert: $G = \emptyset$ oder $S = \emptyset$.
3. Keine Beispiele mehr, aber mehrere Hypothesen. D.h. wir enden mit einer grossen Disjunktion.

8.4 Warum funktioniert das alles?

Wie nah ist eine aus dem Lern-Algorithmus berechnete Hypothese h von der richtigen Funktion f entfernt?

$\leadsto$ Computational Learning Theory: **PAC**-Lernen – **P**robably **A**pproximately **C**orrect.

Idee: Ist eine Hypothese konsistent mit einer großen Trainingsmenge, so kann sie nicht völlig falsch sein.

Wie verhalten sich Trainings und Test-Menge zueinander? Wir machen folgende Annahme:

Trainings- und Testmenge sind gemäß derselben Wahrscheinlichkeitsverteilung aus der Menge X aller Beispiele entnommen.

Definition 8.8 (error(h))
Sei $h \in \mathcal{H}$ eine Hypothese und f die richtige (zu lernende) Funktion. Interessant ist die Menge

$$Diff(f,h) := \{x : h(x) \neq f(x)\}.$$

Wir bezeichnen mit $error(h)$ die Wahrscheinlichkeit, daß ein beliebiges Beispiel aus X in der Menge $Diff(f,h)$ liegt.

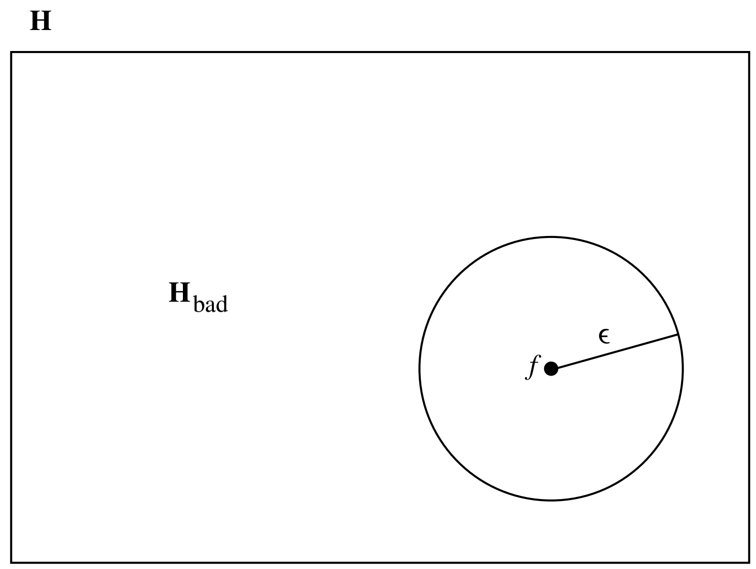
Für $\epsilon > 0$ heißt die Hypothese h ϵ-approximativ korrekt, falls $error(h) \leq \epsilon$.

Sei $\epsilon > 0$ gegeben. Wieviele Beispiele muß die Trainingsmenge enthalten, damit die durch einen Lern.-Alg. erzeugte Hypothese ϵ-approximativ korrekt ist?

Definition 8.9 (Beispielkomplexität)
Gegeben $\delta > 0$ und $\epsilon > 0$. Unter der Beispielkomplexität verstehen wir die Zahl m von Beispielen die ein beliebiger Lern-Algorithmus braucht, damit die von ihm erzeugte Hypothese h mit mindestens Wahrscheinlichkeit $1 - \delta$ jedenfalls ϵ-approximativ korrekt ist.

Theorem 8.10
Die Beispielkomplexität m hängt in folgender Weise von ϵ, δ und dem Hypothesenraum $\mathcal{H}$ ab:

$$m \geq \frac{1}{\epsilon}(ln\frac{1}{\delta} + ln|\mathcal{H}|)$$



Was bedeutet das letzte Ergebnis? Komplexität hängt von $ln(|\mathcal{H}|)$ ab. Für boolesche Funktionen in n-Attributen ist

$$ln(|\mathcal{H}|) = 2^n$$

also braucht man exponentiell viele Beispiele selbst um nur mit einer Wahrscheinlichkeit ϵ-approximativ korrekt zu sein! Also:

- Verschiedene Mengen von Beispielen ergeben verschiedene Aussagen und damit Werte von ϵ. Es kommt also nicht nur darauf an, *wieviele* sondern auch *welche* Beispiele man auswählt.

  Wir relativieren darum unsere Frage und wollen nur noch fordern, daß die Hypothese mit einer Wahrscheinlichkeit von $1-\delta$, wobei $\delta > 0$,ϵ-approximativ korrekt ist.

- Wir möchten auch von speziellen Lern-Algorithmen abstrahieren und eine Aussage über alle möglichen Lern-Algorithmen machen. Wir gehen daher davon aus, daß jeder Lern-Alg. jedenfalls eine Hypothese berechnet, die mit den bisherigen Beispielen konsistent ist.

Beweis des Theorems: Sei $h_b \in \mathcal{H}$ mit $error(h_b) > \epsilon$.

- Wie groß ist die Wahrscheinlichkeit P_b, daß h_b konsistent mit m gewählten Beispielen ist? Für ein Beispiel ist es gerade $\leq (1-\epsilon)$ wegen der Definition von $error$. Also:

$$P_b \leq (1-\epsilon)^m$$

- Wie groß ist die Wahrscheinlichkeit P', daß es überhaupt eine Hypothese h_b mit $error(h_b) > \epsilon$ gibt, die mit m Beispielen konsistent ist?

$$\begin{aligned} P' &\leq \sum_{Hypothesen} (1-\epsilon)^m \\ &\leq |\{h : error(h) > \epsilon\}|(1-\epsilon)^m \\ &\leq |\mathcal{H}|(1-\epsilon)^m \end{aligned}$$

- Wir wollen, daß $P' \leq \delta$:

$$|\mathcal{H}|(1-\epsilon)^m \leq \delta$$

Einges umformen ergibt:

$$\begin{array}{lcl} (1-\epsilon)^m & \leq & \frac{\delta}{|\mathcal{H}|} \\ m ln(1-\epsilon) & \leq & ln(\delta) - ln(|\mathcal{H}|) \\ m & \geq & -\frac{1}{ln(1-\epsilon)}(ln(\frac{1}{\delta}) + ln(|\mathcal{H}|)) \\ m & \geq & frac1\epsilon(ln(\frac{1}{\delta}) + ln(|\mathcal{H}|)) \end{array}$$

Die letzte Zeile gilt wegen $ln(1-\epsilon) < -\epsilon$.

Lernen ist nie besser als in einer Tabelle nachzuschlagen!

1. Ausweg: Wir fordern, daß nicht eine beliebige konsistente Hypothese generiert wird, sondern möglichst spezielle. (Das ist aber dann von der Komplexität noch schlechter.)

2. Ausweg: Wir geben es auf, beliebige boolesche Funktionen zu lernen und beschränken uns auf geeignete Teilklassen.

So können wir etwa **Entscheidungslisten** betrachten.

Theorem 8.11 (E-Listen werden in polynomialer Zeit gelernt)
Jede Funktion aus $k - DL(n)$ (E-Listen mit höchstens k-Tests) haben eine PAC-Komplexität von

$$m = \frac{1}{\epsilon}(ln(\frac{1}{\delta}) + O(n^k log_2(n^k)))$$

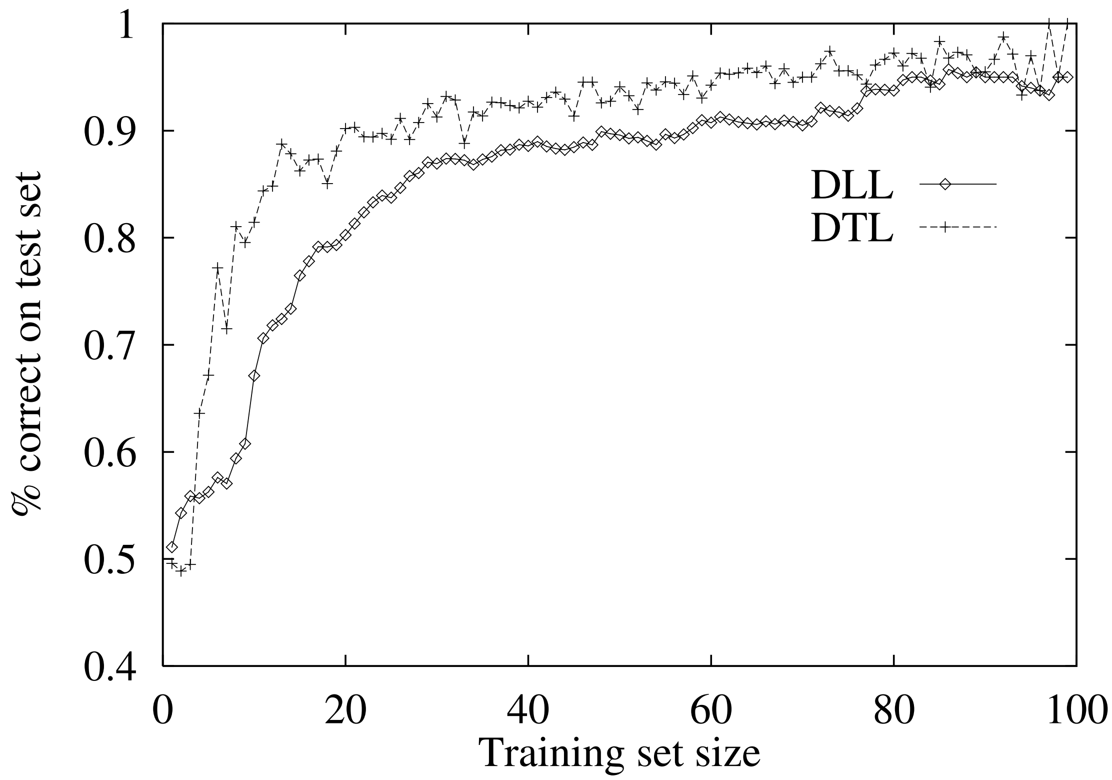
Mit anderen Worten: Jeder Algorithmus der für eine Menge von Beispielen eine konsistente Entscheidungsliste zurückgibt, kann in einen PAC-Lern-Algorithmus verwandelt werden, der eine $k - DL(n)$ Funktion nach höchstens m-Beispielen lernt.

Folgende Abschätzungen sind wichtig:

- $|Conj(n,k)| = \sum_{i=0}^{k} \binom{2n}{i} = \mathsf{O}(n^k)$,
- $|k-DL(n)| \leq 3^{|Conj(n,k)|}|Conj(n,k)\mathbf{!}$,
- $|k-DL(n)| \leq 2^{\mathsf{O}(n^k log_2(n^k))}$.

function DECISION-LIST-LEARNING(*examples*) **returns** a decision list, *No* or failure

  if *examples* is empty **then return** the value *No*
  t ← 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($examples - examples_t$)



8.5 Rauschen und Overfitting

Rauschen:

- Beispiele sind inkonsistent (einmal $Q(x)$, dann $\neg Q(x)$).
- Keine Attribute mehr um Beispiele zu klassifizieren.
- Macht Sinn, wenn die Umgebung nichtdeterministisch

Overfitting: Dual zum Rauschen: überbestimmt.

- Verbleibende Beispiele können mittels Attributen klassifiziert werden, die eine *Ordnung* ergeben, welche nicht vorhanden ist (irrelevante Attribute). **Beispiel:** Mehrere farbige Würfel werden geworfen. Jeder Wurf wird durch (Tag, Monat, Uhrzeit, Farbe) beschrieben. Solnge es keine Inkonsistenz gibt, ist jeder Wurf einzig und durch eine völlig überbestimmte Hypothese beschrieben.

Andere Beispiele:

- Pyramiden,
- Astrologie,
- "Mein magisches Fahrrad".