Einführung in die Methoden der Künstlichen Intelligenz

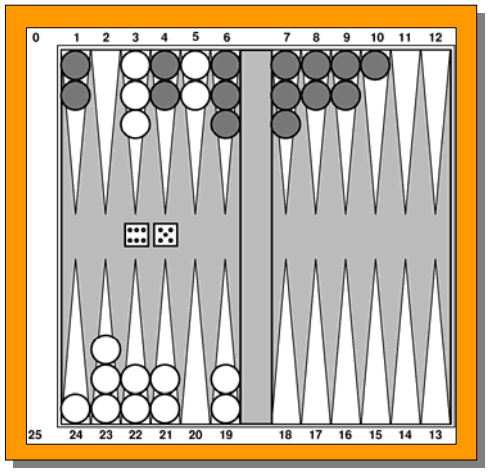
--- Vorlesung vom 15.5.2007 ---
Sommersemester 2007

Prof. Dr. Ingo J. Timm, Andreas D. Lattner
Professur für Wirtschaftsinformatik und Simulation (IS)

Spiele mit Zufallselementen

Spiele mit Zufall

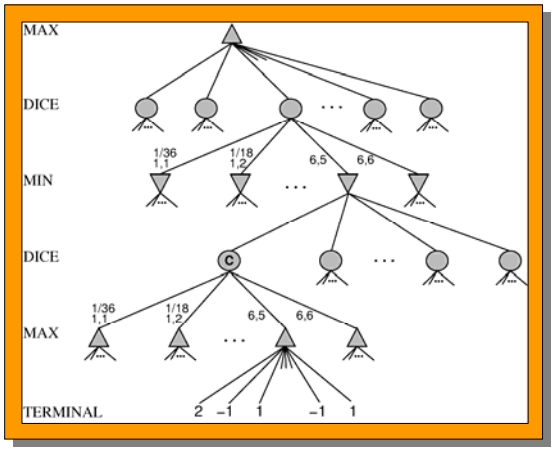
- Unvorhersehbare Ereignisse bringen neue Situationen
- Wissen und Glück (Würfel)
- Bsp.:
  Weiß hat eine 6 und eine 5 gewürfelt und hat dadurch vier Möglichkeiten: (5-10,5-11), (5-11,19-24), (5-10,10-16), (5-11,11-16)
- Problem:
  Weiß kennt nicht die Augen, die Schwarz würfeln wird, also auch nicht, was Schwarz machen wird.
- Konstruktion eines kompletten Suchbaumes ist nicht möglich
- --> *Chance nodes* zusätzlich.



Spiele mit Zufallselementen

Chance Nodes (CN)

- CN als Kreise dargestellt
  - Wir können nicht ausrechnen, welches der beste Zug ist
  - Aber: wir können ein Mittel ausrechnen, ein Mittel über alle möglichen Augen, die vorkommen können.
- Endzustände
  - wie in deterministischen Spielen
- Chance *C:*
  - Sei d_i ein Wurf und $P(d_i)$ die Wahrscheinlichkeit dazu. Für jeden Wurf wird die Utility für den besten Weg für MIN errechnet, aufsummiert und gewichtet



Spiele mit Zufallselementen

Expectiminimax-Wert

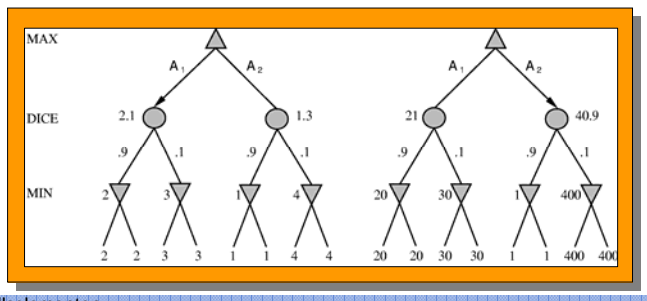
- Expectiminimax-Wert
  - Minimax Wert für Spiele mit Chance Nodes
- Aber:
  - Wert keine "richtigen" Minimax-Werte
  - Nur: wahrscheinlicher Wert
- Wahrscheinlichkeit
  - über die Würfelaugen
- Generalisierung
  - des Minimax-Wertes zu Expectiminimax-Wert

$$\text{EXPECTIMINIMAX}(n) = \begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in Successors(n)} \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in Successors(n)} \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a MIN node} \\ \sum_{s \in Successors(n)} P(s) \cdot \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a chance node} \end{cases}$$

Spiele mit Zufallselementen

Positionsevaluierung

- Abschneiden der Suche und Evaluationsfunktion anwenden ist offensichtlich
- Bei Minimax: ordnungserhaltende Transformation der Blätter macht keinen Unterschied *(1,2,3,4)* vs. *(1,20,30,400)*,
  → Freiheit bei der Wahl der Funktion
- Bei Zufallsverhalten verliert man diese Freiheit: mit *(1,2,3,4)* ist A_1 die beste Wahl, mit *(1,20,30,400)* ist A_2 besser
  → Programm arbeitet anders!
- Vermeidung: Evaluierungsfunktion muss eine *positive lineare* Transformation der Gewinnwahrscheinlichkeit sein
- Wichtig bei Situationen, wo Unsicherheit im Spiel ist

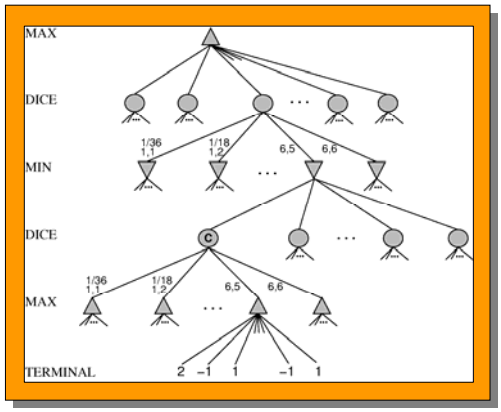


Spiele mit Zufallselementen

JOHANN WOLFGANG GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Komplexität von Expectiminimax

- Minimax $O(b^m)$
- Expectiminimax $O(b^m n^m)$
  - n ist die Anzahl der möglichen Würfelkombinationen
  - viel Extrakosten (z.B. für Backgammon $n = 21$, $b \sim 20$), manchmal aber auch $b=4000$ (Pasch))
- Alpha-Beta Pruning
  - Obere Grenze für C?
  - Möglich, wenn Grenzen für Utilityfunktion gegeben sind



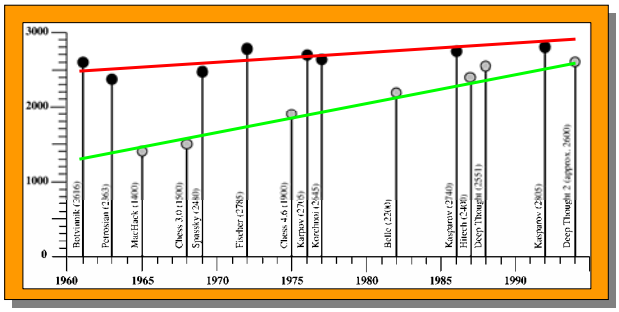
Spiele mit Zufallselementen

JOHANN WOLFGANG GOETHE UNIVERSITÄT FRANKFURT AM MAIN

State-of-the-Art-Programme

State-of-the-art-Programme

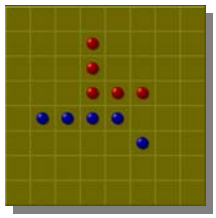
- Zwei Ziele bei der Entwicklung von Spielprogrammen:
  - Wahl der Aktionen in komplexen Domänen mit unsicherem Ergebnis
  - Entwicklung von high-performance Systemen für spezielle Spiele
- Hier: Ausführungen für das leztere (Schach)
  - Konzentration auf Schach extrem ausgeprägt
  - Speed-Schach (5 und 25min) Computer gewinnt gegen Kasparov



State-of-the-Art

State-of-the-art Programme

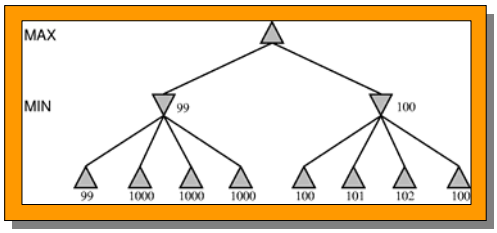
- Schach
  - Deep Blue: Im Schnitt 126 Millionen Knoten in der Sekunde; bis Tiefe 14
  - 2006: Deep Fritz besiegt Kramnik 4:2; http://www.rag.de/microsite_chess_com
- Othello/Reversi
  - Suchraum kleiner als Schach
  - 5 bis 15 legale Züge
  - Computer wesentlich besser als Menschen
  - 1997 Logistello (Buro, 2002) 6:1 gegen WM
- Backgammon
  - Unsicherheit durch Würfel, dadurch Suche teuer
  - TD-Gammon (Gerry Tesauro) auf Basis von KNN & RL
  - 1 Mio Trainingspiele gegen sich selbst
  - Unter den besten drei der Welt
- Go
  - *b* erreicht 360 bei 19x19 Brett, reguläre Suche ist nicht möglich
  - Systeme basieren auf dem wissensbasierten Ansatz
  - Noch kein Programm auf Expertenlevel



State-of-the-Art

Diskussion

- Optimale Entscheidungen bei Spielen meist nicht effizient
- Deswegen: Algorithmen machen Annahmen und Approximationen
  - Standardansatz, basierend auf Minimax, Evaluationsfunktionen und Alpha-Beta-Pruning ist ein Ansatz dafür
  - Minimax ist optimale Methode für den nächsten Schritt, wenn der Suchbaum gegeben ist und die Evaluierungen der Blätter exakt korrekt sind
  - Realität: nur Schätzungen, in Abb. scheint Minimax keine gute Wahl zu sein
  - Algorithmus entscheidet sich für den rechten Zweig, wobei es wahrscheinlicher ist, dass der linke Zweig in der Realität besser wäre
  - Minimaxannahme: alle rechten Knoten sind besser als 99 links



Zusammenfassung

Zusammenfassung

- Spiele spielen für KI ist wie Formel 1 im Motorsport. Hier die wichtigsten Ideen:
  - Ein Spiel kann als Initialzustand, Operatoren, Zieltest (zum Terminieren), und einer Evaluationfunktion definiert werden.
  - In Spielen mit zwei Spielern mit perfekter Information kann der **Minimax**-Algorithmus den nächsten besten Zug bestimmen (Annahme: der Gegner spielt perfekt). Der gesamte Suchbaum wir dabei aufgebaut.
  - Der **Alpha-Beta**-Algorithmus kalkuliert genauso wie Minimax, ist aber effizienter, weil Zweige aus dem Suchbaum abgeschnitten werden.
  - Normalerweise ist es nicht möglich, den gesamten Suchbaum aufzubauen (auch nicht mit Alpha-Beta), also muss man irgendwann die Suche abbrechen und die **Evaluationsfunktion** anwenden.
  - Spiele mit Zufall können mit Hilfe einer Minimax-Erweiterung gespielt werden. Hier werden die sog. Chance-Knoten evaluiert (mittlerer Nutzen aller Kinder, gewichtet über die Wahrscheinlichkeit).

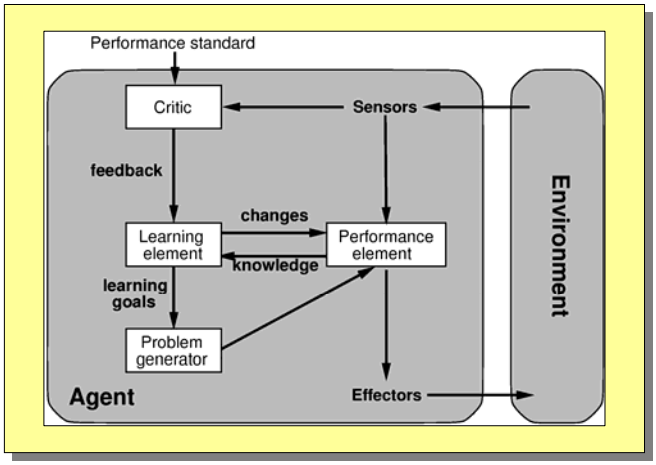
Lernen aus Beispielen

Agenten können ihr Verhalten durch Erfahrung verbessern

Übersicht

1. Generelles Modell vom lernenden Agenten
2. Induktionslernen
3. Entscheidungsbaumlernen
4. Lernen von generellen logischen Beschreibungen
5. Zusammenfassung

Generelles Modell vom lernenden Agenten

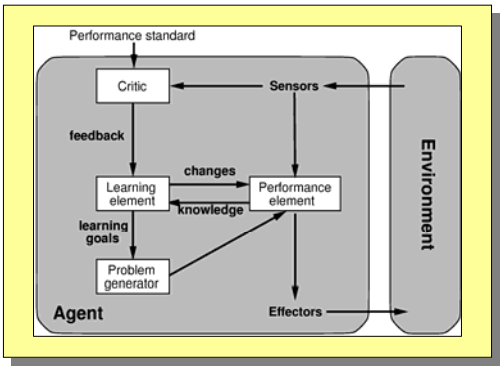


Generelles Modell

JOHANN WOLFGANG GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Vier konzeptuelle Komponenten

1. **Lernelement**
   - Verantwortlich für Verbesserungen
   - Hat Wissen über das Performance-Element und nimmt zusätzlich ein Feedback auf, wie sich der Agent in der Umgebung verhält und bestimmt dann, wie das Performance-Element verändert werden soll, damit sich die Leistung verbessert.
   - Design hängt vom Performance-Element ab.

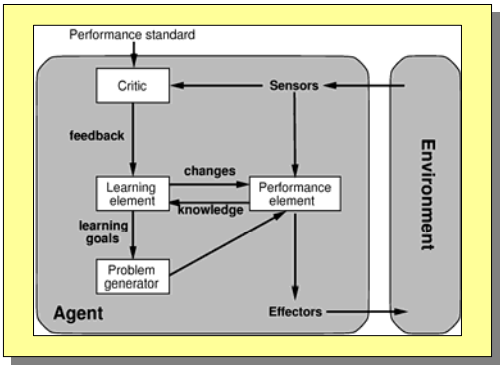


Generelles Modell

JOHANN WOLFGANG GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Vier konzeptuelle Komponenten

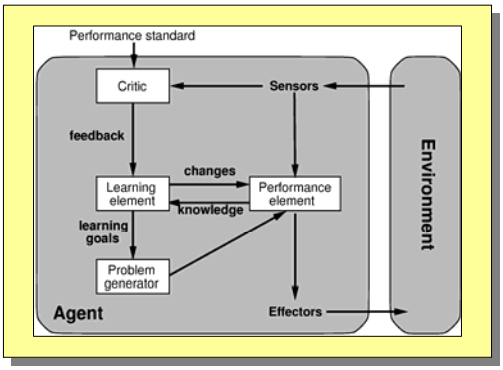
2. **Performance-Element**
   - Verantwortlich für externe Aktionen.
   - Nimmt Wahrnehmungen auf und entscheidet, welche Aktionen durchgeführt werden.
3. **Kritik**
   - Teilt dem Agenten mit, wie gut/schlecht er ist (Feedback).
   - Fester Performance-Standard (Wissen, was gut und was nicht gut ist).
   - Performance-Standard außerhalb des Agenten



Generelles Modell

Vier konzeptuelle Komponenten

4. **Problemgenerator**
   - Verantwortlich für neue Aktionen, die zu neuen informativen Erfahrungen führen.
   - Bsp.: kurzfristig suboptimale statt optimale Aktionen, um langfristig davon zu profitieren.



Generelles Modell

Beispiel

- **Taxifahren**
  - Performance-Element: Aktionen des Taxis (Bremsen, Beschleunigen, Hupen, …)
  - Lernelement: Formuliert Ziele, z.B. bessere Regel zwischen Bremsen und Beschleunigen, Geographische Verhältnisse lernen (Baustellen, …)
  - Kritik: Beobachtungen der Welt, die dem Lernelement gegeben werden (z.B. Spurwechsel am Arc de Triomphe)
  - Problem-Generator: versuche beim nächsten Mal die A1/A24, statt die A27/A2, um nach Berlin zu fahren.
- **Lernelemente**
  - Es existiert eine Vielzahl von Lernelementen. Um sie zu verstehen, schauen wir uns an, wie der Kontext, in dem sie arbeiten ihr Design beeinflusst. Das Design eines Lernelements beinhaltet drei wichtige Punkte:
    1. Welche Komponenten des Performance-Elementes sollen gelernt werden?
    2. Repräsentation?
    3. Feedback?

Generelles Modell

Design von Lernelementen

1. Komponenten des Performance-Elementes
   - Viele Wege, um ein Performance-Element eines Agenten zu bauen. Die folgenden Informationen könnten enthalten sein:
     - Direktes Mapping von Bedingungen zu Aktionen
     - Informationen über das Ergebnis der Aktionen
     - Utility-Informationen zur Bewertung von Weltzuständen
     - ...
   - Jede dieser Komponenten kann gelernt werden, solange ein Feedback vorhanden ist.

Generelles Modell

Design von Lernelementen

2. Repräsentation der Komponenten
   - Verschiedene Repräsentationsformen sind für diese Komponenten möglich.
     - Deterministische Beschreibungen wie linear gewichtete Utility-Funktionen bei Spielen
     - Aussagenlogik/FOL bei logischen Agenten
     - Wahrscheinlichkeitstheoretische Beschreibungen wie Belief-Netze
   - Für alle Formen gibt es Lernalgorithmen, sie unterscheiden sich in ihrer Form für jede Repräsentation, die globale Idee bleibt aber immer die gleiche

Generelles Modell

Design von Lernelementen

3. Verfügbares Feedback
   - Für manche Komponenten (z. B. Vorhersage eines Ergebnisses für eine Aktion) ist ein Feedback nur in Form eines korrekten Ergebnisses vorhanden.
     - Bsp.: Bremsen mit 30km/h → Stop in 10m.
     - Die Wahrnehmung nach dieser Aktion kann allerdings 15m sein.
   - Jede Situation, in der wir sowohl die Inputs als auch die Outputs kennen, wird *überwachtes* Lernen genannt.
   - Bekommt der Agent eine Form von Bewertung für seine Aktion (z. B. Strafzettel für zu starkes Bremsen und deshalb ein Unfall), aber keine korrekte Aktion (z. B. dosiertes Bremsen), wird dieses *reinforcement learning* genannt. Der Strafzettel wäre in diesem Fall das „Reinforcement".
   - Lernen ohne Idee über den korrekten Output wird *unüberwachtes* Lernen genannt.

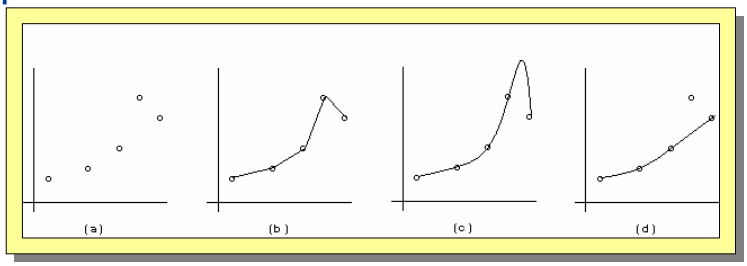
Generelles Modell

Induktionslernen

- Beim *überwachten* Lernen ist die Eingabe das korrekte Ergebnis für eine Menge von Beispieleingaben (Funktion der Inputwerte).
- Formal: Ein *Beispiel* ist ein Paar $(x,f(x))$, bei dem x der Input und $f(x)$ der Output der Funktion nach Anwenden auf x ist.
- Reine induktive Inferenz (oder Induktion) heißt: Gegeben ist eine Menge von Beispielen von *f*. Liefere eine Funktion *h,* die *f* approximiert.
- Die Funktion *h* nennt man auch *Hypothese*.

Induktionslernen

Beispiel



- a) *x,y* Wertepaare im Raum *y = f(x)*, Aufgabe: finde *h(x)*, die den Punkten angepasst ist.
- b) stückweise lineare Funktion *h*.
- c) etwas kompliziertere Funktion *h*. Beide stimmen mit den Punkten überein, unterscheiden sich aber in anderen *y*-Werten zu einem gegebenen *x*.
- d) ignoriert einen Punkt, passt zu den anderen aber mit einer simplen Funktion
- *f* ist unbekannt und ohne zusätzliches Wissen, können wir nicht entscheiden, ob b), c) oder d) die beste Lösung ist.
- Jede Bevorzugung einer Lösung über eine andere nennt man ***bias.***

Induktionslernen

Lernender Reflex-Agent

```
global examples ← {}

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

   if (percept, a) in examples then return a
   else
        h ← INDUCE(examples)
        return h(percept)

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

   examples ← examples ∪ {(percept,action)}
```

- Einfacher Reflex-Agent, der von einem Lehrer beobachtet worden ist. Die globale Variable *examples* wird aktualisiert. Sie enthält eine Liste mit perceptions und actions.
- Variante: Inkrementelles Lernen; jedes Mal, wenn ein neues Beispiel zum Datensatz hinzugefügt wird, wird auch gelernt

Induktionslernen

Lernen von logischen Aussagen

- Zwei Ansätze zum Lernen von logischen Aussagen:
  - *Entscheidungsbäume*: strikte Repräsentation von logischen Aussagen, speziell für Lernen entwickelt
  - *Versionsraum*: genereller als Entscheidungsbäume, aber häufig nicht effizient genug
- Nächste Vorlesung
  - Neuronale Netze: Repräsentation von nicht-linearen numerischen Funktionen

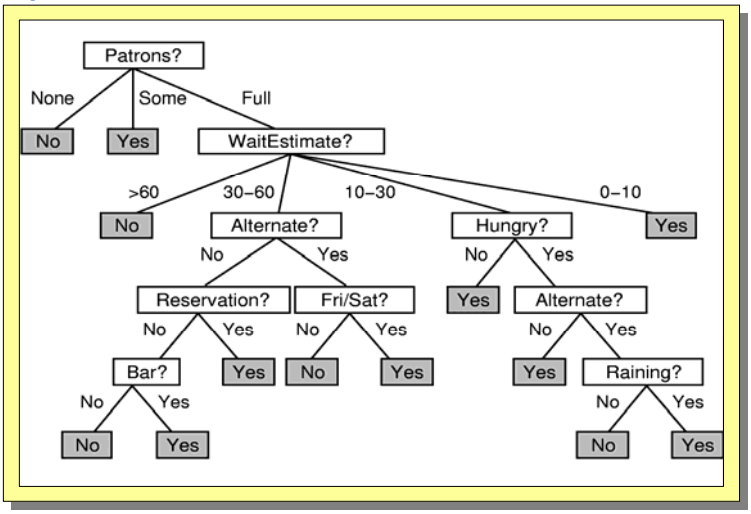
Induktionslernen

Entscheidungsbaumlernen

- Einfach und sehr erfolgreich
- Bekannte Vertreter: ID3, C4.5 (Quinlan)
- Entscheidungsbäume als Performance-Element
  - Input: Satz von Eigenschaften
  - Output: ja/nein-Entscheidung (Boolsche Funktionen). Andere Funktionen mit mehr Outputs können auch repräsentiert werden.
  - Knoten: Test des Wertes von einer der Eigenschaften
  - Kanten sind mit den möglichen Werten dieser Knoten benannt. Jeder Endknoten (Blatt) liefert den Boolschen Wert. (Bsp.: Warten im Restaurant)
- Zielprädikat *WillWait*, wir wollen die Definition dafür lernen
- Wir benötigen Attribute, die uns helfen, die Entscheidung zu fällen:
  - Alternativen vorhanden, Bar vorhanden, Anzahl der Gäste, Wochenende, Preis, Wetter, Reservierung vorhanden, Typ des Restaurants, ...

Entscheidungsbaumlernen

Beispiel



$$\forall r\ Patrons(r, Full) \wedge WaitEstimate(r, 10-30) \wedge Hungry(r, N) \Rightarrow WillWait(r)$$

Entscheidungsbaumlernen

Entscheidungsbauminduktion…

- …aus Beispielen
  - Ein *Beispiel* besteht aus Werten von *Attributen* und einem *Ziel*.
  - Der Wert des Zieles ist die *Klassifikation* des Beispiels.
  - Wenn das Ziel TRUE ist, *positives* Bsp., sonst *negatives* Bsp.
  - Kompletter Satz mit Beispielen ist das *Trainings-Set.*

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*

Entscheidungsbaumlernen

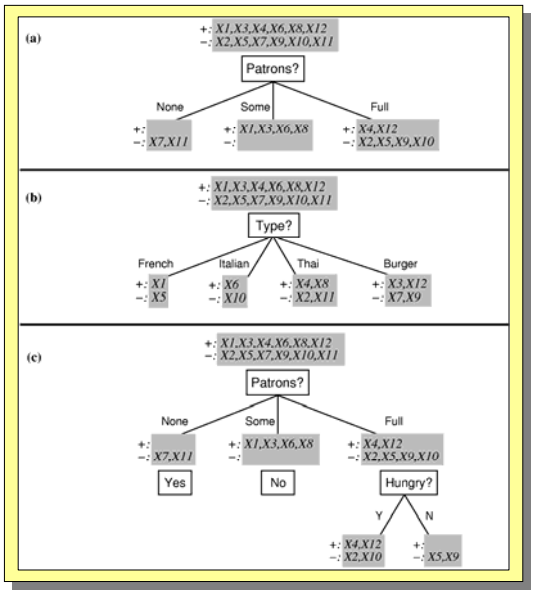
Entscheidungsbauminduktion aus Beispielen

- Ein trivialer Baum erinnert sich an seine Beobachtungen, er ist nicht in der Lage, *Pattern* herauszuziehen und diese dann für Vorhersagen zu nutzen.
- Wir suchen nach einem Muster, das eine große Anzahl von Beispielen abdeckt und das in einer möglichst kurzen Form.
- Diese Strategie des generellen Prinzips von Induktionslernen wird auch *Ockham's razor* genannt.
- *Die wahrscheinlichste Hypothese ist die einfachste, die mit allen Beobachtungen konsistent ist.*
- Idee hinter Decision-Tree-Learning: Teste das wichtigste Attribut zuerst.
- Wichtig meint hier: das Attribut, das den größten Unterschied zur Klassifikation eines Beispiels beiträgt.
- Ziel: möglichst wenige Tests, korrekte Klassifikation
- „Divide and conquer" (vs. „sequential covering")

Entscheidungsbaumlernen

Beispiel

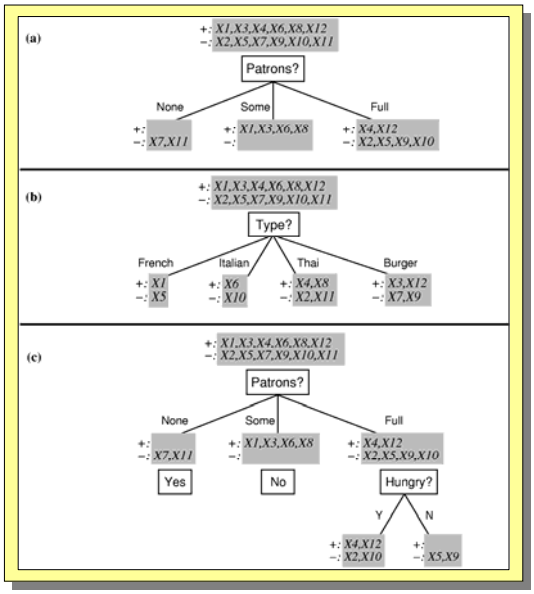
- 12 Trainingsbeispiele
- Positiv/negativ Klassifikation
- Entscheidung, welches Attribut zuerst
- (a) zeigt, das *Patrons* wichtig ist, weil für *None* und *Some* eindeutig sind
- (b) zeigt, das Type kein gutes Attribut ist, weil vier Optionen mit jeweils pos./neg. Werten
- Wir checken jedes Attribut auf diesem Weg und entscheiden uns für das beste
- (c) jetzt wird gesplittet. Jede Option ist ein neuer Baum mit weniger Beispielen und einem Attribut weniger



Entscheidungsbaumlernen

Beispiel

- 4 Fälle sind für dieses rekursive Problem zu beachten:
- 1) wenn pos. und neg. Bsp. vorhanden, dann wähle bestes Attribut zum Splitten
- 2) wenn alle übrig gebliebenen Bsp. positiv (oder negativ), dann fertig
- 3) wenn keine Beispiele mehr vorhanden sind → Bsp. wurde nicht beobachtet, return default, z.B. Majority
- 4) wenn keine Attribute mehr, aber pos. und neg. Bsp. → Problem. Gleiche Daten haben unterschiedliche Klassen. Nicht korrekte Daten → NOISE, Rückgabe: z. B. Majority



Entscheidungsbaumlernen

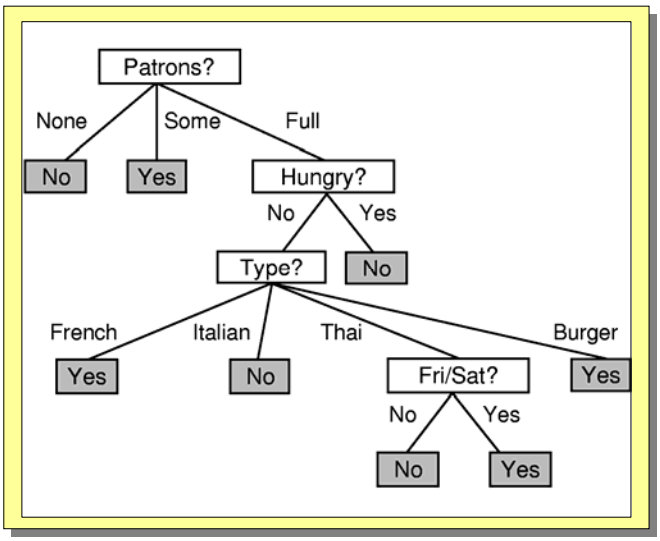
Entscheidungsbaumalgorithmus

```
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
```

Entscheidungsbaumlernen

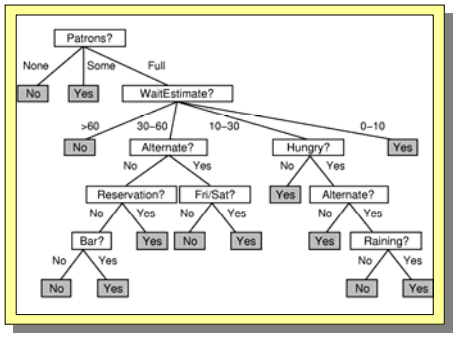
Ergebnis aus Restaurantbeispiel

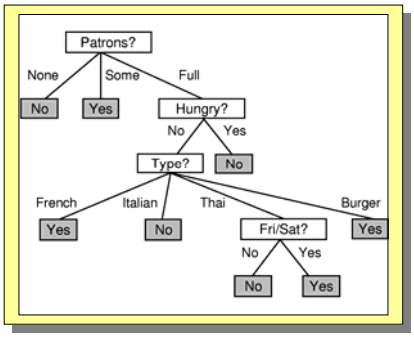


- Entscheidungs-baum aus Beispielen

Entscheidungsbaumlernen

Vergleich

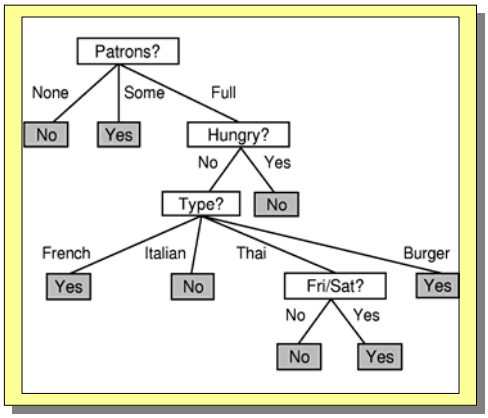




- Baum nach Algorithmus, *Raining* und *Reservations* sind nicht notwendig, um zu einer Entscheidung zu kommen, die Klassifikationen werden auch ohne sie richtig erstellt.
- Außerdem: Der Algorithmus hat eine Regularität entdeckt: Thai on Weekends

Entscheidungsbaumlernen

Problemfall



- Problem: kein Fall, wo das Restaurant voll ist, und das Warten zwischen 0 und 10 min ist.
- Wenn *Hungry* false ist, würde der Baum nicht warten, aber der Gast würde.
- Die Frage ist: Wie inkorrekt ist der Baum? Wenn er alle Beispiele konsistent abdeckt, heißt das nicht, dass er auch für *andere* Beispiele korrekt ist.

Entscheidungsbaumlernen

Entropie

- Diskrete Quelle
  - $X = x_1,...,x_n$ ein endlicher Zeichenvorrat mit *n* Elementarzeichen
- Auftrittswahrscheinlichkeit
  - $p(x_1) = 1/n$
- Informationsgehalt eines Zeichens x_k ist
  - $I(x_k) = log_2(1/p(x_k))$ *bit*
  - *d.h., wenn 16 Zeichen, dann* $log_2(16) = 4$
  - *4 Bits für die Identifikation eines Zeichens*

Da $p(x_1)$ max 1 ist $I(x_k)$ positiv. Je unwahrscheinlicher das Zeichen, desto größer $I(x_k)$.

- Generell:
  - Gegeben: Wahrscheinlichkeitsverteilung $P(x_1,x_2,...,x_n)$
  - Entropie (mittlerer Informationsgehalt einer Quelle) von *P*
    - $I(P) = -(x_1 * log_2(x_1) + x_2 * log_2(x_2) + ... + x_n * log_2(x_n))$

Entscheidungsbaumlernen

JOHANN WOLFGANG GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Informationsgehalt

- Beispiele:
  - *P = (0.5, 0.5)*
  - *I(P) = 1*
  - *P = (0.67, 0.33)*
  - *I(P) = 0.92*
  - *P = (1, 0)*
  - *I(P) = 0*
- Wichtig:
  - Je gleichverteilter die Wahrscheinlichkeitsverteilung, desto größer ist der Informationswert

JOHANN WOLFGANG GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Informationsgehalt

- *T* sind Beispiele, die in disjunkte Klassen $C_1, C_2, ..., C_n$ aufgeteilt sind (Basis: die Werte des Zielattributes).
- Die Information, die benötigt wird, um die Klasse eines Elementes von *T* zu identifizieren ist *Info(T) = I(P)*, wobei *P* die Wahrscheinlichkeitsverteilung der Klassen $C_1, C_2, ..., C_n$ ist.

$$P = \left(|C_1|/T, |C_2|/T, ..., |C_n|/T \right)$$

- Golfbeispiel:

$$Info(T) = I\left(9/14, 5/14\right) = 0.94$$

Outlook	Temperature	Humidity	Windy	Play
sunny	85	85	FALSE	don't play
sunny	80	90	TRUE	don't play
overcast	83	78	FALSE	play
rain	70	96	FALSE	play
rain	68	80	FALSE	play
rain	65	70	TRUE	don't play
overcast	64	65	TRUE	play
sunny	72	95	FALSE	don't play
sunny	69	70	FALSE	play
rain	75	80	FALSE	play
sunny	75	70	TRUE	play
overcast	72	90	TRUE	play
overcast	81	75	FALSE	play
rain	71	80	TRUE	don't play

UNIVERSITÄT FRANKFURT AM MAIN

Informationsgehalt

- Wenn man T in Partitionen von T_1, T_2, ..., T_n aufteilt (Basis: die Werte eines Nicht-Zielattributes X), dann ist die Information, die benötigt wird, um ein Element von T einer Klasse zuzuordnen das gewichtete Mittel der Information, die benötigt wird, um die Klasse eines Elementes von T_i zu identifizieren, d.h., das gewichtete Mittel von $Info(T_i)$.

$$Info(X,T) = \sum_{i=1}^{n} \frac{|T_i|}{|T|} Info(T_i)$$

- Golfbeispiel:

$$Info(Outlook,T) = \frac{5}{14} I(\frac{2}{5},\frac{3}{5}) + \frac{4}{14} I(\frac{4}{4},0) + \frac{5}{14} I(\frac{3}{5},\frac{2}{5}) = 0.694$$

Entscheidungsbaumlernen

UNIVERSITÄT FRANKFURT AM MAIN

Beispiel

- Golfbeispiel:

$$Info(Outlook,T) = \frac{5}{14} I(\frac{2}{5},\frac{3}{5}) + \frac{4}{14} I(\frac{4}{4},0) + \frac{5}{14} I(\frac{3}{5},\frac{2}{5}) = 0.694$$

Outlook	Temperature	Humidity	Windy	Play
sunny	85	85	FALSCH	don't play
sunny	80	90	WAHR	don't play
overcast	83	78	FALSCH	play
rain	70	96	FALSCH	play
rain	68	80	FALSCH	play
rain	65	70	WAHR	don't play
overcast	64	65	WAHR	play
sunny	72	95	FALSCH	don't play
sunny	69	70	FALSCH	play
rain	75	80	FALSCH	play
sunny	75	70	WAHR	play
overcast	72	90	WAHR	play
overcast	81	75	FALSCH	play
rain	71	80	WAHR	don't play

Entscheidungsbaumlernen

Beispiel

- Golfbeispiel:

$$Info(Outlook,T) = \frac{5}{14} I(\frac{2}{5},\frac{3}{5}) + \frac{4}{14} I(\frac{4}{4},0) + \frac{5}{14} I(\frac{3}{5},\frac{2}{5}) = 0.694$$

Outlook	Temperature	Humidity	Windy	Play
sunny	85	85	FALSCH	don't play
sunny	80	90	WAHR	don't play
overcast	83	78	FALSCH	play
rain	70	96	FALSCH	play
rain	68	80	FALSCH	play
rain	65	70	WAHR	don't play
overcast	64	65	WAHR	play
sunny	72	95	FALSCH	don't play
sunny	69	70	FALSCH	play
rain	75	80	FALSCH	play
sunny	75	70	WAHR	play
overcast	72	90	WAHR	play
overcast	81	75	FALSCH	play
rain	71	80	WAHR	don't play

Entscheidungsbaumlernen

Beispiel

- Golfbeispiel:

$$Info(Outlook,T) = \frac{5}{14} I(\frac{2}{5},\frac{3}{5}) + \frac{4}{14} I(\frac{4}{4},0) + \frac{5}{14} I(\frac{3}{5},\frac{2}{5}) = 0.694$$

Outlook	Temperature	Humidity	Windy	Play
sunny	85	85	FALSCH	don't play
sunny	80	90	WAHR	don't play
overcast	83	78	FALSCH	play
rain	70	96	FALSCH	play
rain	68	80	FALSCH	play
rain	65	70	WAHR	don't play
overcast	64	65	WAHR	play
sunny	72	95	FALSCH	don't play
sunny	69	70	FALSCH	play
rain	75	80	FALSCH	play
sunny	75	70	WAHR	play
overcast	72	90	WAHR	play
overcast	81	75	FALSCH	play
rain	71	80	WAHR	don't play

Entscheidungsbaumlernen

Information Gain

- *Gain(X,T):*
  - Die Differenz zwischen der Information, die benötigt wird, um ein Element von *T* zu identifizieren und der Information, die benötigt wird, um ein Element von *T* zu identifizieren, **nachdem** der Wert des Attributes *X* bekannt ist.
  - Das ist der ***Gewinn*** an Informationsgehalt durch das Attribut *X*.

$$Gain(X,T) = Info(T) - Info(X,T)$$

- Golfbeispiel:

$$Gain(Outlook,T) = Info(T) - Info(Outlook,T) = 0.94 - 0.694 = 0.246$$

- Aber:
  - *Info(Windy,T)* = 0.892 und *Gain(Windy,T)* = 0.048
  - → *Outlook bietet mehr Zunahme an Informationsgehalt !*

Entscheidungsbaumlernen

Ausdrucksmächtigkeit von Entscheidungsbäumen

- Entscheidungsbäume können als Konjunktionen von individuellen Implikationen angesehen werden. Sie sind aber limitiert für ein Objekt.
- Sie sind in der Klasse “Aussagenlogische Sprachen” voll ausdrucksfähig, d. h. jede Boolsche Funktion kann als Entscheidungsbaum dargestellt werden.
- z. B. jede Reihe in einer Wahrheitstabelle ist ein Pfad im Baum.
- Komplexität: exponentiell
- Parity-Function und Majority-Function sind Beispiele dafür, dass Entscheidungsbäume exponentiell groß sein können.
- Mit anderen Worten: Entscheidungsbäume sind gut für manche Funktionen und nicht gut für andere.
- Es gibt keine Repräsentation, die gut für alle Funktionen ist!

Entscheidungsbaumlernen

UNIVERSITÄT FRANKFURT AM MAIN

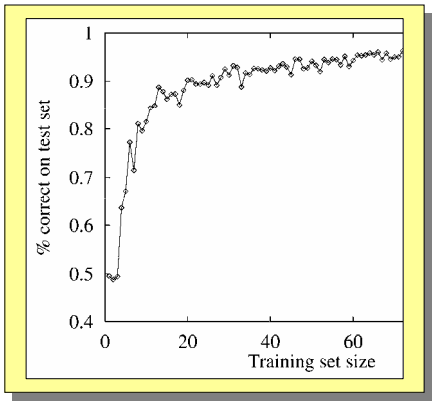
Bemerkungen

- Bewertung der Performanz bei Lernalgorithmen
  - Große Mengen Daten
  - Trainings- und Testset (disjunkt)
  - Lernalgorithmus auf Trainingsset anwenden (Generieren von *h*).
  - Messen der korrekt erzeugten Beispiele im Testset (*h* angewendet).
  - Wiederholen der letzten vier Schritte für unterschiedliche Größen von Trainings- und Testset (Zufallszahlen verwenden).
- Das Ergebnis ist die mittlere Vorhersagequalität in Form einer Funktion der Größe des Trainingssets. Wenn man dieses in einer Graphik darstellt, erhält man die *Lernkurve*.

UNIVERSITÄT FRANKFURT AM MAIN

Lernkurve

- *Lernkurve* für unser Restaurantbeispiel
- Happy Graph: Qualität der Vorhersage nimmt zu, je größer das Trainingsset ist



Entscheidungsbaumlernen

Lernen von generellen logischen Beschreibungen

- Lernen von generellen logischen Repräsentationen
- Genereller Rahmen für Lernalgorithmen mit dem Hintergrund, dass induktives Lernen als ein Prozess der Suche einer guten Hypothese im Raum verstanden werden kann (Hypothesenraum).
- Hypothesen
  - Start mit Zielprädikat Q (*WillWait*). Q ist unär. Wir suchen einen äquivalenten logischen Ausdruck, den wir für die Klassifikation verwenden können.
  - Jede Hypothese, die wir finden ergibt einen solchen Ausdruck. Wir nennen das *Kandidatendefinition* des Zielprädikates.
  - Wenn C_i eine solche Def. ist, ist jede H_i ein Ausdruck der Form $\forall x\ Q(x) \Leftrightarrow C_i(x)$

Lernen von generellen logischen Beschreibungen

- Der Entscheidungsbaum von vorhin würde dann so aussehen:

$$\forall r\ WillWait(r) \Leftrightarrow Patrons(r,Some) \vee$$
$$Patrons(r,Full) \wedge \neg Hungry(r) \wedge Type(r,French) \vee$$
$$Patrons(r,Full) \wedge \neg Hungry(r) \wedge Type(r,Thai) \wedge Fri/Sat(r) \vee$$
$$Patrons(r,Full) \wedge \neg Hungry(r) \wedge Type(r,Burger)$$

- Der Hypothesenraum H_r besteht aus allen Hypothesen, die der Lernalgorithmus finden kann.

  H = $\{H_1,\ldots,H_n\}$, Am Anfang nimmt der Lernalgorithmus an, dass eine der Hypothesen wahr ist:
- Jede Hypothese behauptet, dass ein bestimmter Satz von Beispielen Beispiele für das Zielprädikat sind. Dieses nennt man *Erweiterung* des Prädikates.
- Zwei Hypothesen mit unterschiedlichen Erweiterungen sind deshalb logisch inkonsistent.

Beispiele

- Logisch gesehen ist ein Beispiel ein Objekt, zu dem ein Zielprädikat passt oder auch nicht.
- X_i ist das *i*-te Beispiel, $D_i(X_i)$ ist eine logische Beschreibung dafür.
- Die Klassifikation ist gegeben durch eine Aussage $Q(X_i)$ wenn das Beispiel positiv ist und $\neg Q(X_i)$, wenn es negativ ist.

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

- *Bsp.:* $Alternate(X_1) \wedge \neg Bar(X_1) \wedge \neg Fri/Sat(X_1) \wedge$ Hungry(X_1) ... und die Klassifikation $WillWait(X_1)$

Beispiele

- Das komplette Trainings-Set ist eine Konjunktion aller dieser Aussagen.
- Eine Hypothese passt zu diesen Beispielen dann und nur dann, wenn sie logisch konsistent zum Trainings-Set ist.
- Konsistenz
  - Was heißt Inkonsistenz?
  - Ein Beispiel kann **falsch negativ** sein, d. h. die Hypothese sagt, es wäre negativ, es ist aber positiv (Neues Bsp. #13)

    $Patrons(X_{13},Full) \wedge Wait(X_{13},0\text{-}10) \wedge \neg Hungry(X_{13}) \wedge ,\ldots, \wedge WillWait(X_{13})$

    wäre falsch negativ für die Hypothese H_r
  - Aus H_r und der Beispielbeschreibung kann man sowohl $WillWait(X_{13})$ als auch $\neg WillWait(X_{13})$ deduzieren.
  - Es gibt dementsprechend auch **falsch positive** Beispiele.
- Wenn ein Beispiel falsch positiv oder falsch negativ für eine Hypothese ist, sind sie logisch inkonsistent zueinander.

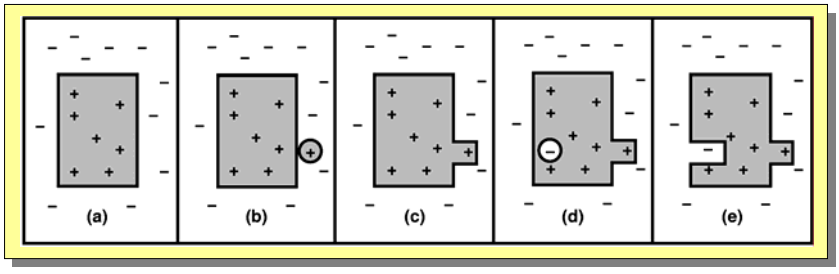
Beispiele

- Annahme: Beispiel ist ok → dann kann die Hypothese verworfen werden.
  - Logisch gesehen ist das genau das, was die Resolutionsregel macht, wo die Disjunktionen von Hypothesen zu Klauseln korrespondiert und das Beispiel zu Literalen korrespondiert
- Ein einfaches logisches Schlussfolgerungssystem ist in der Lage, von Beispielen zu lernen (prinzipiell), weil es Hypothesen verwerfen kann.
  - I_1 ist ein Beispiel und der Hypothesenraum ist $H_1 \vee H_2 \vee H_3 \vee H_4$. Wenn I_1 mit H_2 und H_3 inkonsistent ist, kann das System den neuen Hypothesenraum $H_1 \vee H_4$ deduzieren.
- Man kann deshalb sagen, dass induktives Lernen logisch gesehen ein Prozess ist, der graduell Hypothesen eliminiert, die inkonsistent mit den Beispielen sind.
- Weil der Hypothesenraum am Anfang SEHR groß sein kann, sollte man keinen Resolutionsbeweiser benutzen.
- Im folgenden betrachten wir zwei Ansätze, die weniger aufwendig sind.

Lernen logischer Beschreibungen

Current-best-hypothesis-Suche
(Stuart Mill, 1843)

- **Die Idee: einzelne Hypothese lange behalten, sie nachjustieren, wenn neue Beispiele kommen, so dass Konsistenz gewahrt wird.**
- **Annahme, H_r ist ok, bis ein falsch negatives Beispiel kommt.**



a) Konsistente Hypothese
b) Falsch negativ
c) Generalisierung
d) Falsch positiv
e) Spezialisierung

Lernen logischer Beschreibungen

Current-best-hypothesis-Suche Bemerkungen

- Bei Generalisierung und Spezialisierung auf Konsistenz checken.
- Algorithmus und Varianten davon sind in vielen Machine-Learning-Systemen integriert. Bei großen Mengen von Daten gibt es Probleme:
  - alle Beispiele auf Konsistenz prüfen ist teuer
  - Heuristiken sind schwer zu finden und Backtracking dauert lange.

Least-commitment-Suche

- Current Best Hypothesis:
  - wir *müssen* eine Hypothese auswählen (nach bestem Gewissen), auch wenn wir noch nicht alle erforderlichen Daten haben. Alternative? → Alle Hypothesen behalten, die konsistent mit den Daten bis dato sind.
- Annahme: Hypothesenraum ist $H_1 \vee H_2 \vee H_3 \ldots \vee H_n$.
- Inkonsistente Hypothesen werden eliminiert. Übrig gebliebene Hypothesen enthalten Lösung.
  - Diese Hypothesen nennt man *Versionsraum* und der Lernalgorithmus dazu Version-Space-Learning-Algorithmus (auch *Candidate-Elimination*-Algorithmus).
- Algorithmus findet einen Subset von Hypothesen, die mit den Beispielen konsistent sind.

Least-commitment-Suche

- Inkrementelles Verfahren, kein Rückschritt, betrachtet keine „alten“ Beispiele
- Alle konsistenten Hypothesen werden behalten
- Problem: Hypothesenraum ist riesig. Wie können wir diese Disjunktionen überhaupt zu Papier bringen?
  - **Analogie hilft:**
  - Reelle Zahlen zwischen 1 und 2
  - [1,2] Intervalle, die nur die Grenzen definieren helfen bei diesem Problem. Es funktioniert, weil wir eine Reihenfolge im Intervall haben.
  - Wir haben aber auch eine Reihenfolge im Hypothesenraum: Generalisierung/Spezialisierung (partial ordering). Jede Grenze ist eine Menge von Hypothesen → **boundary set**.
  - **G-set** ist das generellste und **S-set** das speziellste boundary set. Alles dazwischen ist garantiert konsistent.

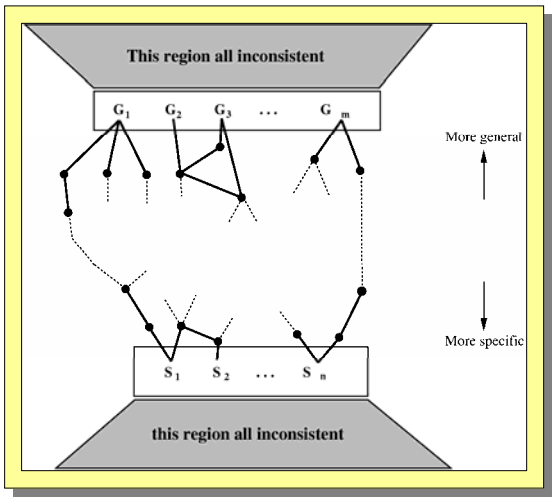
Lernen logischer Beschreibungen

Least-commitment-Suche

- Der aktuelle Versionsraum ist ein Satz von Hypothesen, der mit allen Beispielen konsistent ist. Er wird durch den S-set und den G-set repräsentiert.
- Der S-set beeinhaltet nur konsistente Hypothesen und es gibt keine konsistenten Hypothesen, die noch spezieller sind.
- Der G-set beeinhaltet nur konsistente Hypothesen und es gibt keine konsistenten Hypothesen, die noch genereller sind.
- G-set wird am Anfang auf TRUE, S-set auf FALSE gesetzt.
  - 1) Jede konsistente Hypothese (außerhalb der „boundary sets“) ist spezifischer als eine der generellen und genereller als eine der spezifischen.
  - 2) Jede Hypothese, die spezifischer als eine aus dem G-set und genereller als eine aus dem S-set ist, ist konsistent.

Lernen logischer Beschreibungen

Least-commitment-Suche



G-set wird am Anfang auf TRUE (Hypothese, die alles enthält), S-set auf FALSE (Hypothese, dessen Erweiterung leer ist) gesetzt.

Eigenschaften:

1) Jede konsistente Hypothese ist spezifischer als eine der generellen und genereller als eine der spezifischen.

2) Jede Hypothese, die spezifischer als eine aus dem G-set und genereller als eine aus dem S-set ist, ist konsistent.

Lernen logischer Beschreibungen

Least-commitment

- Was passiert bei einem Update?
  - Wenn falsch positiv oder falsch negativ:
    - Falsch positiv für S_j, d. h. S_i ist zu generell, keine konsistenten Spezialisierungen für S_i → verwerfen.
    - Falsch negativ für S_i d. h. S_i ist zu speziell → ersetzen durch alle seine Generalisierungen.
    - Falsch positiv für G_j, d. h. G_i ist zu generell, → ersetzen durch alle seine Spezialisierungen.
    - Falsch negativ für G_i, d. h. G_i ist zu speziell, keine konsistenten Generalisierungen für G_i → verwerfen.
  - Dieses wird für jede Instanz neu durchgeführt, bis eine von drei Situationen eintritt:
    - Genau ein Konzept ist über im Versionsraum → return als unique Hypothese.
    - Versionsraum kollabiert, S oder G sind leer → keine konsistenten Beispiele.
    - Keine Beispiele mehr, aber mehr als eine Hypothese im Versionsraum.

Lernen logischer Beschreibungen

Diskussion

- Zwei Probleme mit Versionsraum:
  - Er wird kollabieren, wenn Rauschen in den Daten
  - Wenn nicht limitierte Disjunktionen im Hypothesenraum zugelassen sind, beinhaltet der S-set immer eine einzige sehr spezifische Hypothese → die Disjunktion der Beschreibung der positiven Beispiele, die bis dato gesehen wurden. Der G-set enthält demnach die Negation der Disjunktionen der Beschreibungen der negativen Beispiele.
- Rauschen ist noch nicht befriedigend gelöst.
- *Meta-Dendral* und *Lex* sind zwei Bespiele für Systeme, die den Versionsraum einsetzen.
- Versionsraum-Methoden sind eher unpraktikabel für reale Anwendungen (Rauschen), geben aber einen guten Einblick in die logische Struktur des Hypothesenraumes.

Lernen logischer Beschreibungen

Zusammenfassung

- Lernen kann als Lernen einer Funktion angesehen werden. Wir haben uns auf Induktion konzentriert (Lernen von Input/Output-Paaren):
  - Lernen in intelligenten Agenten ist wichtig, um mit unbekannten Umgebungen fertig zu werden.
  - Es wird zwischen Performance-Element (verantw. für die Selektion von Aktionen) und Lernelement (verantw. für die Modifikation des Performance-Elements) unterschieden.
  - Die Schwierigkeit beim Lernen ist von der Repräsentation abhängig. Funktionen können als logische Aussagen, Polynome, Belief-Netze, Neuronale Netze u. a. repräsentiert werden.
  - Entscheidungsbäume sind effizient, um deterministische Boolsche Funktionen zu lernen.
  - Ockham's razor bedeutet, die einfachste Hypothese zu nehmen, die die beobachteten Beispiele abdeckt.
  - Die Performanz von Lernalgorithmen kann mittels Lernkurven gemessen werden
  - Zwei generelle Ansätze für das Lernen logischer Theorie: Der Current-best-Ansatz wartet und justiert eine Hypothese, der Version-Space-Ansatz wartet die Repräsentation aller konsistenten Hypothesen. Beide sind anfällig für Rauschen.