

AI

Kunstmatige Intelligentie (AI)

Hoofdstuk 18, 19.1 en 21.1/3 van Russell/Norvig = [RN]
Leren

voorjaar 2011 — College 11, 19 april 2011

www.liacs.nl/home/kosters/AI/

Er zijn vele soorten **leren**:

supervised leren zowel input als juiste bijbehorende output zijn beschikbaar voor het leren (bijvoorbeeld ID3)

reinforcement leren het juiste antwoord wordt niet verteld, maar er is beloning/straf voor betere/slechtere antwoorden (bijvoorbeeld Genetische algoritmen)

unsupervised leren niets bekend over de juiste antwoorden; probeer *patronen* te vinden (Data mining)

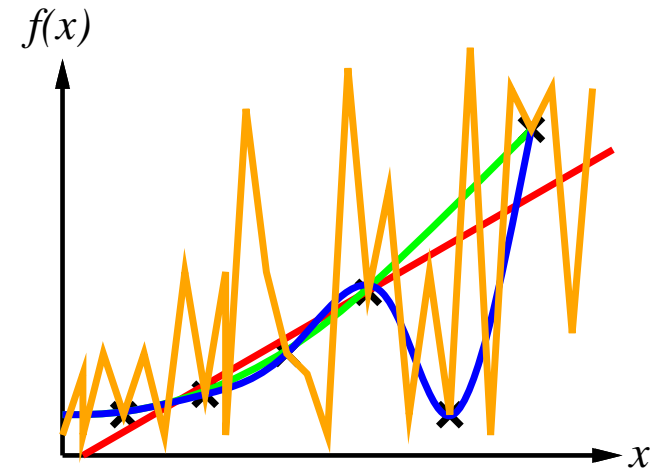
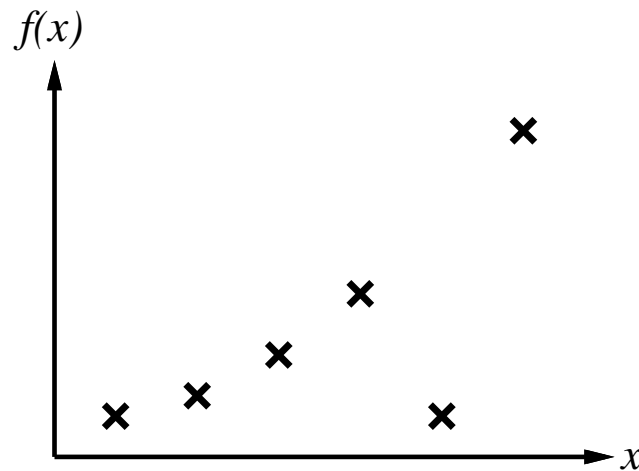
Sommigen onderscheiden ook **statistische leermethoden**, zoals Neurale netwerken en technieken voor Bayesiaanse netwerken.

Zuiver inductief leren (inductie) is het zo goed mogelijk leren van een doelfunctie f op grond van voorbeeldparen $(x, f(x))$. We proberen een **hypothese** h te vinden die zoveel mogelijk op f lijkt: $h \approx f$, gegeven een **trainingsset**.

Voorbeeld: Neurale netwerken.

Dit is een simplificatie van het echte leren: zo gebruiken we geen voorkennis, nemen een observeerbare deterministische omgeving aan, en hebben voorbeelden nodig.

Een goede hypothese **generaliseert**, dat wil zeggen: kan ook nieuwe voorbeelden correct afhandelen.



Een **bias** is een voorkeur voor de ene hypothese boven de andere. **Ockham's razor** (ook: Occam) geeft de voorkeur aan eenvoudiger hypothesen (bij vergelijkbare prestatie).



Beslissingsbomen (decision trees) proberen uit een aantal voorbeelden een ja/nee beslissingsalgoritme af te leiden. Het is een classificatie-algoritme. Zie ook het college Data mining!

Elke interne knoop van de boom correspondeert met een test van de waarde van een van de eigenschappen (attributen) van de voorbeelden, en de takken naar de kinderen corresponderen met de mogelijke waarden.

Voor de classificatie van een nieuw voorbeeld v wandel je, beginnend bij de wortel, door de boom; je kiest steeds het kind dat correspondeert met v 's waarde van het betreffende attribuut. Een blad bevat de Booleaanse waarde die je moet retourneren als je dat blad bereikt: de klasse.

Bij het bezoek aan een plaatselijk restaurant moeten we op grond van 10 attributen beslissen of we blijven wachten tot er plaats voor ons is (de “target” WW = WillWait). Er zijn 12 voorbeelden (Examples):

Ex	Attributes										Target WW
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	
X ₁	T	F	F	T	Some	\$\$\$	F	T	French	0–10	T
X ₂	T	F	F	T	Full	\$	F	F	Thai	30–60	F
X ₃	F	T	F	F	Some	\$	F	F	Burger	0–10	T
X ₄	T	F	T	T	Full	\$	F	F	Thai	10–30	T
X ₅	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X ₆	F	T	F	T	Some	\$\$	T	T	Italian	0–10	T
X ₇	F	T	F	F	None	\$	T	F	Burger	0–10	F
X ₈	F	F	F	T	Some	\$\$	T	T	Thai	0–10	T
X ₉	F	T	T	F	Full	\$	T	F	Burger	>60	F
X ₁₀	T	T	T	T	Full	\$\$\$	F	T	Italian	10–30	F
X ₁₁	F	F	F	F	None	\$	F	F	Thai	0–10	F
X ₁₂	T	T	T	T	Full	\$	F	F	Burger	30–60	T

We hebben 10 attributen:

Alt: is er dichtbij een alternatief?

Bar: is er een bar?

Fri: is het vandaag vrijdag/zaterdag of niet?

Hun: hebben we honger?

Pat: aantal klanten (“patrons”)

Price: prijs

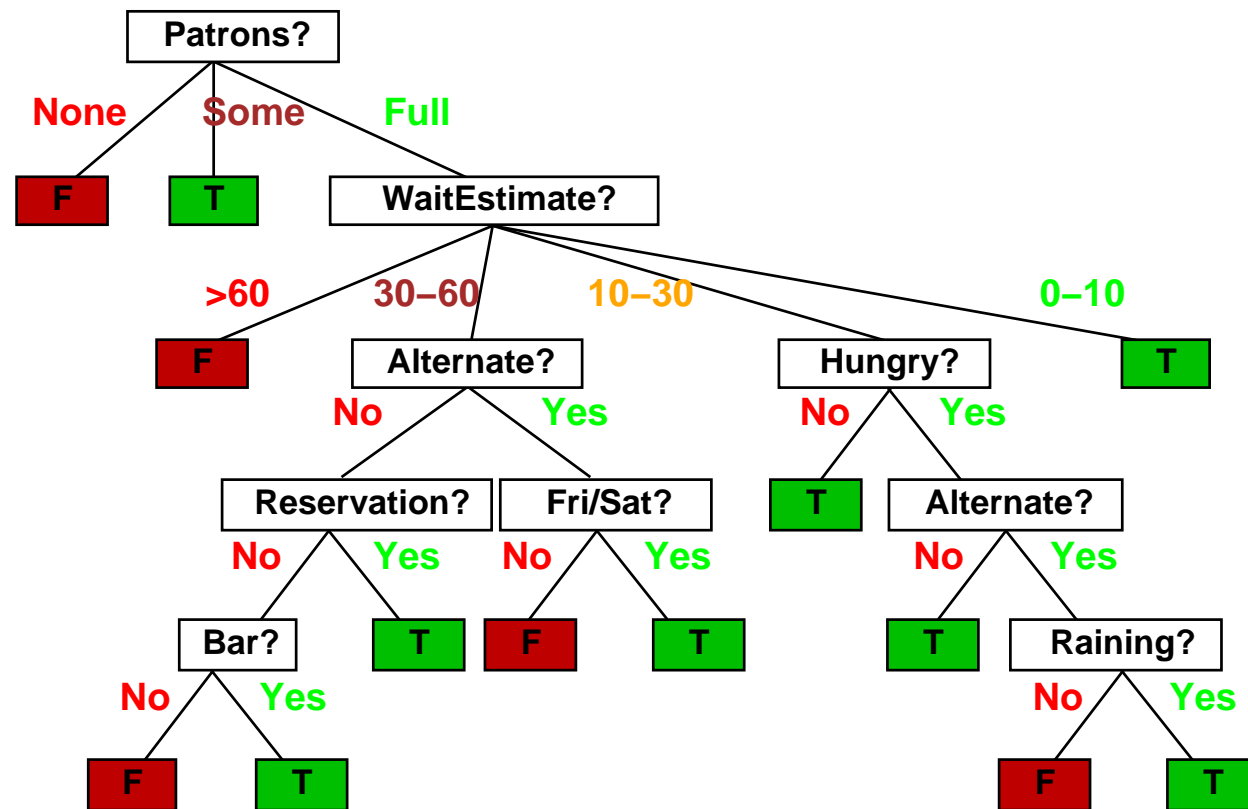
Rain: regent het nu?

Res: hebben we een reservering gemaakt?

Type: soort restaurant

Est: door de eigenaar geschatte wachttijd

Een mogelijke beslissingsboom is:



Het probleem is nu om bij gegeven voorbeelden (de trainingsset), positieve zowel als negatieve, een beslissingsboom te vinden die zo goed mogelijk bij de voorbeelden past en (Ockham) zo eenvoudig mogelijk is.

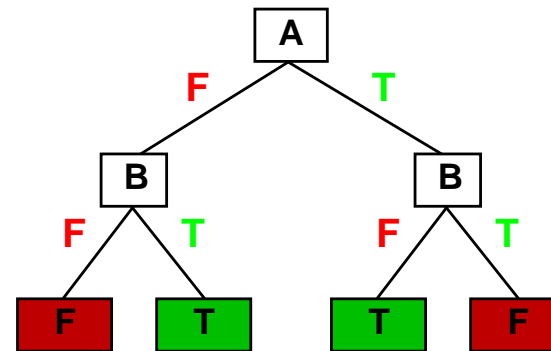
Er kunnen voorbeelden zijn die elkaar tegenspreken! En de boom hoeft ook niet alle attributen te gebruiken . . .

J. Ross Quinlan bedacht hiervoor in 1986 de methode **ID3** (“Iterative Dichotomiser 3”), in 1993 verbeterd tot **C4.5** en **J48** — zoals het in het gratis data mining pakket **Weka** heet.



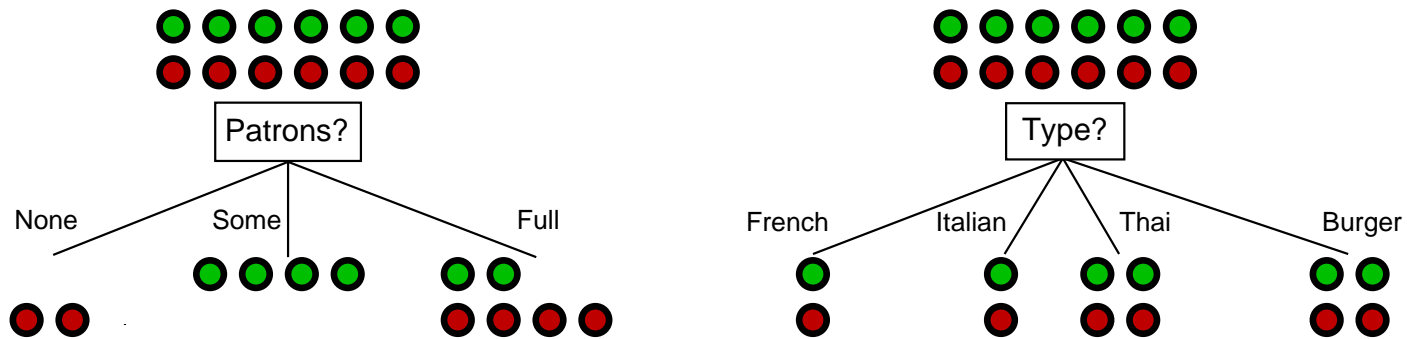
Nog een eenvoudig voorbeeld, een beslissingsboom voor het XOR-probleem:

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



Je kunt dus voor ieder voorbeeld een pad in de boom maken, maar dat ziet er in het algemeen niet uit en generaliseert niet goed (en het gaat ook niet als voorbeelden elkaar tegenspreken).

Het idee van ID3 is als volgt. We moeten eerst het wortel-attribuut vinden, het attribuut waar we als eerste op splitsen: de splijtende vraag die je als eerste stelt.



Het lijkt er op dat het attribuut Pat(rons; aantal klanten) beter schift dan Type. (Dat maken we straks preciezer.) Dus kiezen we dat, en zo verder.

Het recursieve **ID3-algoritme** werkt als volgt.

Stel we zitten in een knoop, en hebben de verzameling V van alle voorbeelden die in die knoop terecht komen. Dan zijn er vier gevallen:

1. Geen voorbeelden meer ($V = \emptyset$)? Geef defaultwaarde, bijvoorbeeld “majority-value” van ouderknoop.
2. Alle voorbeelden dezelfde classificatie? Geef die.
3. Geen attributen meer? “Majority-value” van V .
4. Bepaal “beste” attribuut (**hoe?** — zie straks), splits daarop, en ga recursief verder met de kinderen.

Het ruwe idee van **Shannon's informatietheorie** is: hoe meer we verrast worden door een mededeling, hoe meer “inhoud” die mededeling had. De **entropie** kun je zien als de verwachte verrassing.

Als we een eerlijke munt hebben (kans op kop is 0.5) geeft een uitslag van een muntworp 1 bit “informatie”.

Als de munt 0.99 kans op kop heeft (en dus 0.01 op munt) is dat (gemiddeld) 0.08 bits “informatie”.



Stel dat een random-variabele X de volgende kansverdeling heeft: waarde A heeft kans $1/2$, B kans $1/4$ en C en D beide kans $1/8$. Hoeveel ja/nee vragen heb je gemiddeld nodig om achter de waarde van X te komen? Dat zijn er (als je eerst naar A vraagt, dan (eventueel) naar B en dan (eventueel) naar C): $1/2 \cdot 1 + 1/4 \cdot 2 + 1/4 \cdot 3 = 1.75$.

Voor de **entropie** $H(X)$ van de kansverdeling X geldt (per definitie; \log_2 is logaritme met grondtal 2, $\log_2 2 = 1$):

$$\begin{aligned} H(X) &= - \sum_p p \log_2 p \\ &= -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{4} \log_2 \frac{1}{4} - \frac{1}{8} \log_2 \frac{1}{8} - \frac{1}{8} \log_2 \frac{1}{8} = 1.75 \end{aligned}$$

En als je “optimaal” wilt coderen? Nu: A als 1, B als 01, C als 001 en D als 000, met gemiddelde lengte weer 1.75.

Voor ID3 wordt dit: stel we hebben p positieve en n negatieve voorbeelden in een knoop. “Vooraf” is de entropie:

$$-\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n},$$

het gemiddeld aantal bits nodig om een voorbeeld te classificeren. (Voor het restaurant, met $p = n = 6$: 1 bit.)

Stel we splitsen op een zeker attribuut a met ν mogelijke waarden, waarbij p_i positieve en n_i negatieve gevallen waarde i hebben ($1 \leq i \leq \nu$). Dan is de entropie “achteraf”:

$$\sum_{i=1}^{\nu} \frac{p_i + n_i}{p+n} \cdot \left\{ -\frac{p_i}{p_i + n_i} \log_2 \frac{p_i}{p_i + n_i} - \frac{n_i}{p_i + n_i} \log_2 \frac{n_i}{p_i + n_i} \right\},$$

het verwachte aantal bits nog nodig voor classificatie — als we op a hebben gesplitst.

We kiezen nu het attribuut dat de maximale entropiewinst boekt. In ons restaurant-voorbeeld: vooraf

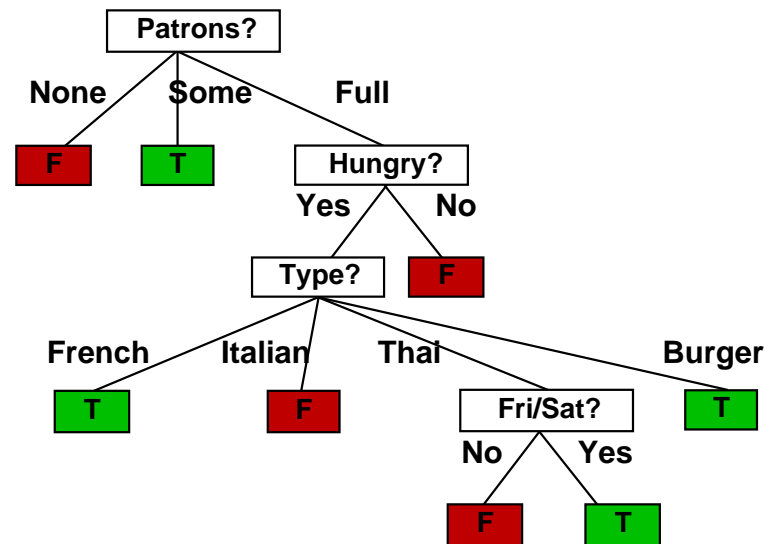
$$-\frac{6}{12} \log_2 \frac{6}{12} - \frac{6}{12} \log_2 \frac{6}{12} = 1;$$

en achteraf, bij splitsen op Pat:

$$\frac{2}{12} \cdot 0 + \frac{4}{12} \cdot 0 + \frac{6}{12} \cdot \left\{ -\frac{2}{6} \log_2 \frac{2}{6} - \frac{4}{6} \log_2 \frac{4}{6} \right\} = 0.459$$

(de zogeheten “**gain**” is $1 - 0.459 = 0.541$; interpreteer $0 \log_2 0$ als 0) en bij Type: nog steeds 1, met “gain” $1 - 1 = 0$. Kies dus het attribuut met de hoogste “gain”: Pat.

De uiteindelijke ID3-boom is:



Let op: ID3 is een gretig algoritme, en levert niet altijd de beste boom op!

In het algemeen gebruik je weer een trainingsset, testset en validatieset. En “cross-validatie” ...

Er zijn allerlei problemen, waaronder noise (positief en negatief voorbeeld met dezelfde attribuut-waarden), irrelevante attributen (de dag als voorspeller voor een dobbelsteenworp), continue attribuutwaarden (temperatuur), missende waarden, enzovoorts. In C4.5, de opvolger van ID3, wordt hier rekening mee gehouden. En het maakt de boom kleiner: “pruning”.

Een voorbeeld. Duidelijk is dat meerwaardige attributen vaak onterecht gekozen worden, bijvoorbeeld de datum of het identificatienummer bij ons restaurant-voorbeeld. Daarom wordt de entropiewinst vaak gedeeld door

$$- \sum_{i=1}^{\nu} \frac{p_i + n_i}{p + n} \log_2 \frac{p_i + n_i}{p + n} .$$

Bekijk de volgende dataset met trainings-voorbeelden:

Dag	Weer	Temp	Vochtigheid	Wind	Tennis?
D1	zon	warm	hoog	zwak	N
D2	zon	warm	hoog	sterk	N
D3	bewolkt	warm	hoog	zwak	J
D4	regen	gem	hoog	zwak	J
D5	regen	koud	normaal	zwak	J
D6	regen	koud	normaal	sterk	N
D7	bewolkt	koud	normaal	sterk	J
D8	zon	gem	hoog	zwak	N
D9	zon	koud	normaal	zwak	J
D10	regen	gem	normaal	zwak	J
D11	zon	gem	normaal	sterk	J
D12	bewolkt	gem	hoog	sterk	J
D13	bewolkt	warm	normaal	zwak	J
D14	regen	gem	hoog	sterk	N

We berekenen de “gain” van de vier mogelijke attributen:

Weer	Temp	Vochtigheid	Wind
0.246	0.029	0.151	0.048

Dus kiezen we voor Weer in de wortelknoop. Nu zijn er 5 dagen met Weer = zon, 2 positief en 3 negatief. De “gain” van Temp is hier $\left\{-\frac{3}{5} \log_2 \frac{3}{5} - \frac{2}{5} \log_2 \frac{2}{5}\right\} - \left\{\frac{2}{5} \cdot 0 + \frac{2}{5} \cdot 1 + \frac{1}{5} \cdot 0\right\} = 0.970 - 0.4 = 0.570$, maar die van Vochtigheid is het beste: 0.970 (voor Wind: 0.019), dus die kiezen we daar.

Uiteindelijk vinden we de beslissingsboom die hoort bij:
 (Weer = zon \wedge Vochtigheid = normaal) \vee
 (Weer = bewolkt) \vee (Weer = regen \wedge Wind = zwak)

Elke Booleaanse functie kun je als een beslissingsboom schrijven. De majority functie (zie Neurale netwerken) vergt een erg grote boom, evenals de “parity functie” (1 precies als een even aantal inputs 1 is). Een beslissingsboom is soms goed, soms niet.

Er zijn 2^{2^n} verschillende waarheidstafels met n Booleans, en dus evenveel mogelijke beslissingsbomen. Voor $n = 6$ zijn dat er 18.446.744.073.709.551.616. Dit is het totaal aantal mogelijke hypothesen.

Een beslissingsboom met maar één knoop heet een **decision stump**.

Elk algoritme dat hypothesen oplevert die waarschijnlijk bij benadering goed zijn heet een **probably approximately correct (PAC) learning** algoritme.

Hoeveel voorbeelden heb je nodig? Stel we hebben een *distributie* D waaruit we de voorbeelden trekken, en we proberen de (“ware”) functie f te benaderen met een hypothese h . Dan definiëren we $\text{error}(h)$ als de kans dat $h(x)$ en $f(x)$ verschillen als we x volgens D trekken.

Een hypothese heet *benaderend correct* als $\text{error}(h) \leq \varepsilon$ voor een kleine $\varepsilon > 0$.

De kans dat een echt foute hypothese h_{bad} (met $\text{error}(h_{\text{bad}}) > \varepsilon$; deze hypothesen vormen samen H_{bad}) consistent is met de eerste N (onafhankelijke) voorbeelden is hooguit $(1 - \varepsilon)^N$, want de kans dat bijvoorbeeld het eerste voorbeeld fout is, is $> \varepsilon$.

De kans dat er minstens één consistente hypothese in H_{bad} zit is dus hooguit $|H_{\text{bad}}|(1 - \varepsilon)^N \leq |H|(1 - \varepsilon)^N$, met $|H|$ het totaal aantal hypothesen.

Stel dat we die kans $\leq \delta$ willen hebben, dus $|H|(1 - \varepsilon)^N \leq \delta$. Dit kunnen we (dankzij $e^{-\varepsilon} \geq 1 - \varepsilon$) garanderen als $|H|e^{-N\varepsilon} \leq \delta$, oftewel $N \geq \frac{1}{\varepsilon}(\log \frac{1}{\delta} + \log |H|)$.

Voor onze beslissingsbomen groeit “dus” het aantal voorbeelden dat je moet zien als 2^n — wat toevallig net het totaal aantal mogelijke voorbeelden is.

Het leren van hypothesen kun je ook als volgt doen. Voor ons restaurant-voorbeeld komen de voorbeelden een voor een binnen als logische zinnen: $\text{Alt}(X_1) \wedge \neg\text{Bar}(X_1) \wedge \dots$. We zoeken een hypothese die de classificatie $\text{WillWait}(X)$ of $\neg\text{WillWait}(X)$ goed “voorspelt”.

Onze ID3-boom stelt de volgende hypothese voor:

$$\begin{aligned} \forall x \text{ WillWait}(x) \Leftrightarrow & \text{Pat}(x, \text{Some}) \\ & \vee (\text{Pat}(x, \text{Full}) \wedge \text{Hun}(x) \wedge \text{Type}(x, \text{French})) \\ & \vee (\text{Pat}(x, \text{Full}) \wedge \text{Hun}(x) \wedge \text{Type}(x, \text{Thai}) \\ & \quad \wedge \text{Fri}(x)) \\ & \vee (\text{Pat}(x, \text{Full}) \wedge \text{Hun}(x) \wedge \text{Type}(x, \text{Burger})) \end{aligned}$$

Na het zien van X_1 , positief, met $\text{Alt}(X_1)$ true, zou onze hypothese kunnen zijn

$$H_1 = \forall x \text{ WillWait}(x) \Leftrightarrow \text{Alt}(x)$$

Na X_2 , negatief, met $\text{Alt}(X_2)$ true (dus een **false positive**) gaan we specialiseren naar (bijvoorbeeld):

$$H_2 = \forall x \text{ WillWait}(x) \Leftrightarrow \text{Alt}(x) \wedge \text{Pat}(x, \text{Some})$$

Na de **false negative** X_3 generaliseren naar (bijvoorbeeld):

$$H_3 = \forall x \text{ WillWait}(x) \Leftrightarrow \text{Pat}(x, \text{Some})$$

Denk aan het kloppend houden van X_1 en X_2 . Na X_4 :

$$H_4 = \forall x \text{ WillWait}(x) \Leftrightarrow \text{Pat}(x, \text{Some}) \vee (\text{Pat}(x, \text{Full}) \wedge \text{Fri}(x))$$

Je kunt dus één hypothese onderhouden, die je aanpast bij nieuwe voorbeelden: de **current best hypothesis**.

Het algoritme is:

- bij “false positive”: specialiseren (voeg condities toe)
- bij “false negative”: generaliseren (laat condities weg)

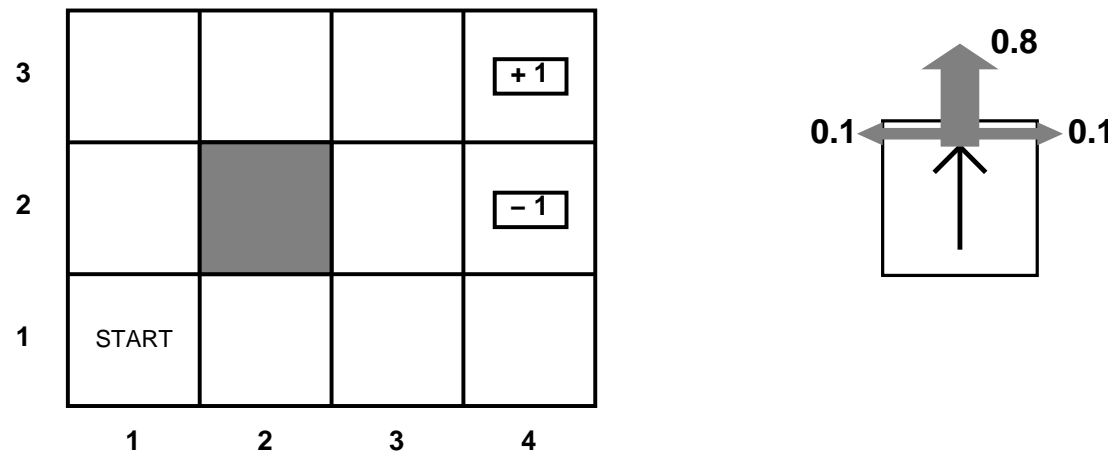
Problemen: herhaald erg veel controleren, lastige heuristieken, veel backtracking noodzakelijk.

Bij **least commitment search** onderhoudt je een partieel geordende verzameling van toegestane hypothesen (de **version space**), met twee grensverzamelingen bestaande uit de meest algemene (de G -verzameling; “general”) en de meest specifieke (de S -verzameling) hypothesen.

Kom je een false positive tegen voor een $S_i \in S$, dan gooi je S_i er uit. En bij een false negative vervang je S_i door redelijke generalisaties. Analoog bij G .

Je komt wel voor ingewikkelde problemen te staan.

Een voorbeeld van een **Markov Decision Process (MDP)**:

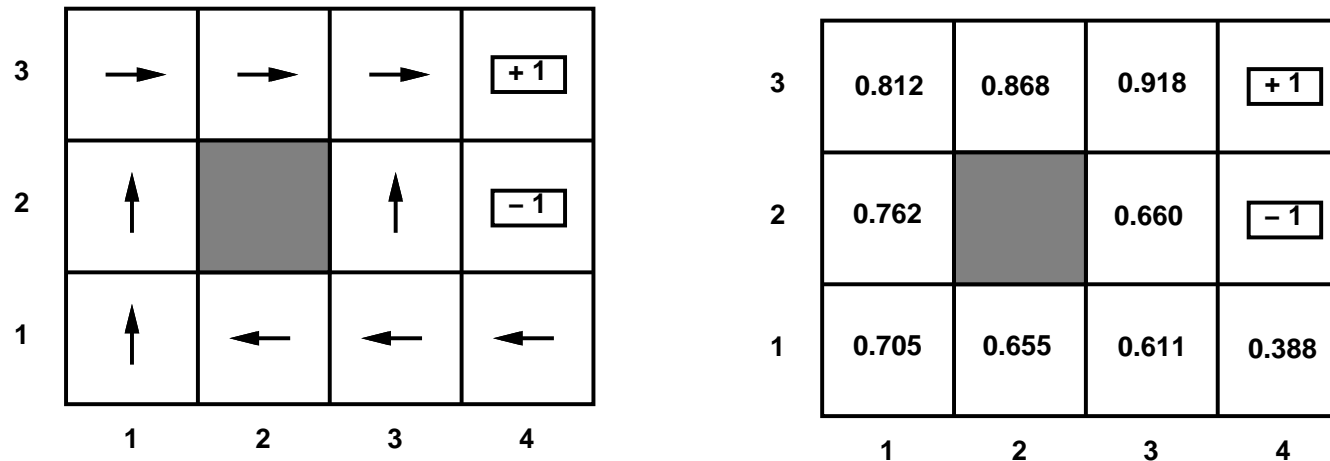


Het **transitiemodel** $T(s, a, s')$ (rechts) geeft de kans dat actie a vanuit toestand s resulteert in toestand s' . De beloning (**reward**) voor een niet-doel toestand s is $R(s) = -0.04$.

Een **policy** π vertelt in elke toestand s wat te doen: $\pi(s)$. De **utility**(-functie) U is de “som van de opgedane rewards”.

Een **optimale policy** π^* geeft de hoogste verwachte utility.

Een optimale policy met bijbehorende utilities:



De verwachte utility hangt dus af van de policy:

$$U^\pi(s) = E \left\{ \sum_{t=0}^{\infty} \gamma^t R(s_t) \mid s = s_0, \pi \right\}$$

Hier is $0 < \gamma \leq 1$ de **discount factor**; $\pi^*(s)$ is de actie a met maximale $\sum_{s'} T(s, a, s') U^{\pi^*}(s')$.

De (optimale) utilities $U(s) = U^{\pi^*}(s)$ voldoen aan de **Bellman vergelijking** (1957):

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s')$$

In bijvoorbeeld vakje (3, 3) geldt (als $\gamma = 1$):

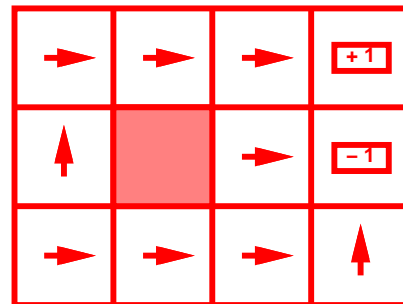
$$0.918 = -0.04 + 1 \cdot (0.1 \cdot 0.918 + 0.8 \cdot 1 + 0.1 \cdot 0.660)$$

Je kunt de vergelijking(en) benaderend oplossen met:

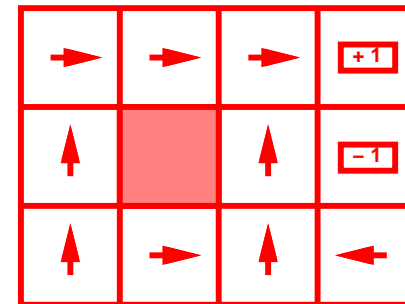
$$U_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U_i(s')$$

Dit heet **value iteration**; ook bestaat **policy iteration**.

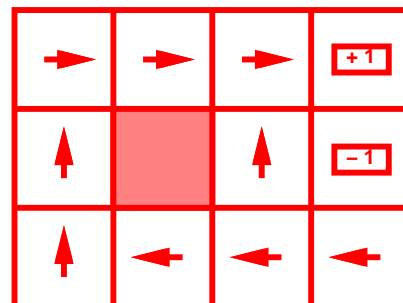
Afhankelijk van de waarde $r = R(s)$ van de reward in niet-doel toestanden heb je verschillende optimale policy's:



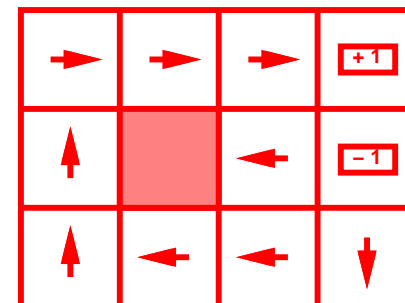
r in $[-\infty ; -1.6284]$



r in $[-0.4278 ; -0.0850]$



r in $[-0.0480 ; -0.0274]$



r in $[-0.0218 ; 0.0000]$

Nu proberen we, bij geven policy π , de utility $U^\pi(s)$ te leren, terwijl je $T(s, a, s')$ niet kent. Voorbeeldtechnieken:

Bij **direct utility estimation** bekijk je grote aantallen **trials**, en schat op grond daarvan:

$(1, 1)_{-0.04} \rightarrow (1, 2)_{-0.04} \rightarrow (1, 3)_{-0.04} \rightarrow (1, 2)_{-0.04}$
 $\rightarrow (1, 3)_{-0.04} \rightarrow (2, 3)_{-0.04} \rightarrow (3, 3)_{-0.04} \rightarrow (4, 3)_{+1}$
 levert schatting $0.84 = (0.80 + 0.88)/2$ voor $U^\pi((1, 3))$ op.

Bij **Temporal Difference (TD)** leren gebruik je de leerregel

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha (R(s) + \gamma U^\pi(s') - U^\pi(s))$$

als je $s \rightarrow s'$ waarneemt (α is de leersnelheid).

Als je namelijk denkt dat $U^\pi((1, 3)) = 0.84$ en $U^\pi((2, 3)) = 0.92$, en je ziet nu $(1, 3) \rightarrow (2, 3)$, verwacht je $U^\pi((1, 3)) = -0.04 + 1 \cdot U^\pi((2, 3))!$

Dit keer proberen we een (optimale) policy te leren. Er zijn weer vele mogelijkheden:

Temporal difference leren kan — met dezelfde vergelijking als tevoren — ingezet worden.

Bij **Q-leren** probeer je $Q(a, s)$, de utility als je a doet in toestand s , te leren; $U(s) = \max_a Q(a, s)$. We kunnen

$$Q(a, s) = R(s) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(a', s')$$

leren via

$$Q(a, s) \leftarrow Q(a, s) + \alpha (R(s) + \gamma \max_{a'} Q(a', s') - Q(a, s))$$

als $s \xrightarrow{a} s'$. Dit is ook weer TD leren ...

Het huiswerk voor de volgende keer (dinsdag 26 april 2011): lees **Hoofdstuk 13 en 14**, p. 484–499 en 510–529 van [RN], over Onzekerheid en Bayesiaanse netwerken door.

Werk aan de vierde opgave: **Genetische algoritmen**, en kijk nog eens naar de oude tentamens.