Intelligenza Artificiale (parte 2)

Colognese, Rossini

maggio 2018

Incertezza e Probabilità

Ci sono diversi metodi per manipolare l'incertezza: *logica di default o non-monotona* (assunzioni più o meno ragionevoli), *fuzzy logic* (sono leciti valori intermedi tra vero e falso; manipola il *degree of truth* e non l'incertezza), *probabilità* (date delle evidenze, si può calcolare un grado di attendibilità).

Probabilità

La probabilità ***Bayesiana (o soggettiva)*** si basa sul grado di conoscenza (non di verità) che varia con nuove evidenze.
Decision Theory = Utility Theory + Probability Theory, dove l'*utility* indica la preferenza per raggiungere lo scopo.

Maximum Expected Utility (MEU): scegliere l'azione che produce l'*utility* prevista più elevata, calcolata come media di tutti i possibili risultati dell'azione.

Sintassi delle Proposizioni

Basic proposition: variabili randomiche (RV).
Proposizioni: combinazioni booleane arbitrarie di RV (booleane o discrete).

Evento Atomico: assegnamento di tutte le variabili ({Cavity, Toothache}).

Joint Probability Distribution: tabella contenente l'insieme delle RV data la probabilità di ogni evento atomico su queste RV (somma degli eventi = 1).

$\mathbf{P}(Weather, Cavity)$ = a 4×2 matrix of values:

$Weather =$	*sunny*	*rain*	*cloudy*	*snow*
$Cavity = true$	0.144	0.02	0.016	0.02
$Cavity = false$	0.576	0.08	0.064	0.08

Probabilità Condizionale

È una probabilità a posteriori, cioè indica come cambia una probabilità dopo una certa evidenza (probabilità di avere una carie sapendo che ho mal di denti; **P(cavity | Toothache) = 0.6**).
Alcune evidenze possono essere irrilevanti: **P(cavity | Toothache, Sunny) = P(cavity | Toothache)**.

$$P(A|B) = \frac{P(A \wedge B)}{P(B)} \quad \text{if } P(B) \neq 0 \qquad P(A \wedge B) = P(A|B) * P(B) = P(B|A) * P(A)$$

Chain Rule (si ripete fino ad avere una variabile): $P(X_1, \ldots, X_n) = P(X_1, \ldots, X_{n-1}) * P(X_n|X_1, \ldots, X_{n-1})$

Normalization: con una *costante di normalizzazione* α (Cavity = distribuzione, quando c'è/non c'è la carie).
$P(Cavity|toothache) = \alpha P(Cavity, toothache) = \alpha[P(Cavity, toothache, catch) + P(Cavity, toothache, catch)]$
$= \alpha[\langle 0.108, 0.016\rangle + \langle 0.012, 0.064\rangle] = \alpha\langle 0.12, 0.08\rangle = \langle 0.6, 0.4\rangle$

	toothache		¬ *toothache*	
	catch	¬ *catch*	*catch*	¬ *catch*
cavity	.108	.012	.072	.008
¬ *cavity*	.016	.064	.144	.576

Toothache è la variabile di evidenza (E), *catch* è la variabile nascosta (H), *cavity* è la variabile *query* (Y).
$P(Y|E = e) = \alpha\, P(Y, E = e) = \alpha\, \Sigma_h P(Y, E = e, H = h)$
Eventi indipendenti *(per ridurre la tabella)*: $P(A|B) = P(A)$ *or* $P(B|A) = P(B)$ *or* $P(A, B) = P(A) * P(B)$

Bayes' Rule: *P(Causa (A) | Effetto(B))* $= \frac{P(B|A)*P(A)}{P(B)}$. Dato un effetto, calcola la probabilità di una possibile causa.

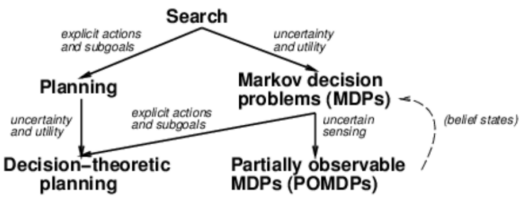
This is an example of a naive Bayes model:

$$\mathbf{P}(Cause, Effect_1, \ldots, Effect_n) = \mathbf{P}(Cause) \prod_i \mathbf{P}(Effect_i|Cause)$$

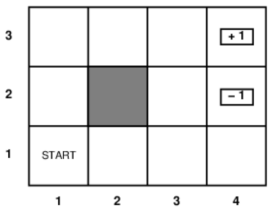
data la causa, gli effetti sono presi indipendenti tra di loro

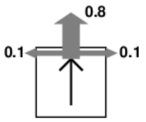
Decisioni Complesse

Le **decisioni** corrispondono a *sequenze di azioni*, prese in base al *reward* che si vuole raggiungere. Il valore di un'azione va oltre il beneficio immediato: *utility* a lungo termine (studente a lezione non solo per quella lezione ma per passare l'esame) e acquisire conoscenze. È dunque necessario un framework per prendere decisioni sequenziali e *far fronte all'incertezza*.

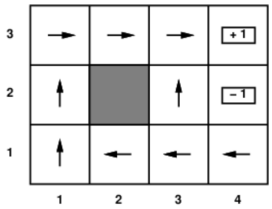


States $s \in S$, actions $a \in A$
Model $T(s, a, s') \equiv P(s'|s, a) =$ probabilità che a in s vada in s'
Reward function $R(s)$ (or $R(s, a)$, $R(s, a, s')$)
$\begin{cases} -0.04 & \text{(small penalty) for nonterminal states} \\ \pm 1 & \text{for terminal states} \end{cases}$





L'approccio non è ottimo perché è sconveniente valutare tutte le sequenze senza considerare il risultato reale ($\mathcal{O}(k^t n^t)$ con k azioni, t stages e n risultati). È complesso stimare l'*utility* di una sequenza rispetto a quella di un singolo stato.
Per trovare una sequenza ottimale si usa una ***policy** ottimale* $\pi(s)$ che massimizza la somma attesa dei rewards.



Decision Trees

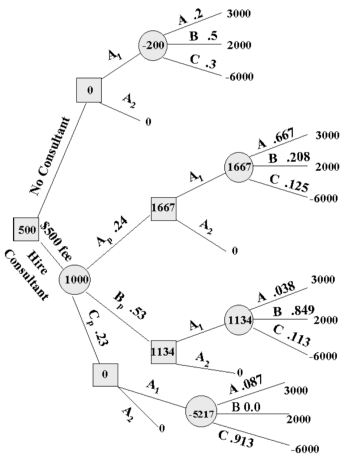
L'idea principale è un approccio backward dalle foglie, facendo uso della *Maximum Expected Utility (MEU)*.

Il valore di un nodo foglia C è: $EU(C)$.

Il valore di un *Chance Node C* (cerchi) è:
$EU(C) = \Sigma_{D \in Child(C)} PR(D) EU(D)$

Il valore di un *Decision Node D* (quadrati) è:
$EU(D) = max_{C \in Child(D)} EU(C)$

La *policy* massimizza l'utility di un decision node:
$\pi(D) = argmax_{C \in Child(D)} EU(C)$



Markov Decision Processes

I ***Markov Decision Problems*** sono una classe generica di problemi di ricerca non-deterministici (più generali e meno strutturati dei *decision trees*; possono avere cicli).
Ci sono quattro componenti (S, A, R, Pr):

- S è un insieme finito di stati ($|S| = n$);
- A è un insieme finito di azioni ($|A| = m$);
- una *funzione di transizione* Pr: $p(s', a, s) = Pr\{S_{t+1} = s' | S_t = s, A_t = a\}$;
- *real valued reward function* R: $r(s', a, s) = \mathbb{E}[R_{t+1} | S_{t+1} = s', A_t = a, S_t, s]$. ($\mathbb{E}$ = *expected value*).

Nei *Markov Decision Problems* le azioni/stati passati sono irrilevanti quando si prendono decisioni (solo stato corrente). Non dipende dalla storia e dal tempo (stazionarietà). È *completamente osservabile*: non si predice esattamente lo stato futuro ma conosciamo quello presente.

$$Pr\{R_{t+1}, S_{t+1} | S_t, A_t\} = Pr\{R_{t'+1}, S_{t'+1} | S'_t, A'_t\} \quad \forall t, t'$$

Ci sono principalmente tre tipi di *policies*:

- *Non stazionaria*: $\pi : S \times T \to A$. $\pi(s, t)$ azione allo stato s con t stati al termine;
- *Stazionaria*: $\pi : S \to A$. $\pi(s)$ azione per lo stato s (senza guardare il tempo);
- *Stocastica*: $\pi(a|s)$ probabilità di scegliere l'azione a nello stato s (non la vedremo).

Policy non stazionaria

Value function: $V : S \to \Re$. Associa un valore considerando i rewards accumulati nel tempo.
$v_\pi(s)$ denota il valore della policy π per lo stato s.
Il problema si ha quando ci sono infinite sequenze di stati (e di conseguenza un numero infinito di reward). Soluzioni:

- Si sceglie un limite finito (*horizon*) che può essere un numero fissato di passi;
- *stato assorbente*: garantisce che per ogni policy verrà raggiunto uno stato terminale;
- uso del *discounting* γ: uso una funzione per dare più importanza ai reward nuovi rispetto a quelli vecchi. Più γ è piccolo e più si ragiona sul presente. I reward recenti hanno un'*utility* più alta di quelli vecchi.

Value di uno stato s seguendo la policy π: si accumulano i reward (*discounted*); $v_\pi(s) = \mathbb{E}\{\Sigma_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\}$
Q-Value *(action value or quality function)* prendendo l'azione a nello stato s seguendo la policy π;

$$q_\pi(s, a) = \Sigma_{s'} p(s'|a, s)(r(s, a, s') + \gamma v_\pi(s')) \quad \text{dove } v_\pi(s) = q_\pi(s, \pi(s))$$

Bellman Equations

Il valore dello stato iniziale deve essere uguale al valore *discounted* dello stato successivo atteso, più il reward atteso lungo il percorso: $v_\pi(s) = \Sigma_{s'} p(s'|\pi(s), s)(r(s, \pi(s), s') + \gamma v_\pi(s'))$

Optimal policy: $\pi_*(s)$ è una policy ottimale sse $v_{\pi_*}(s) \geq v_\pi(s) \ \ \forall s, \pi$.
Optimal value:$v_*(s) = max_\pi v_\pi(s)$ expected utility partendo da s e agendo da lì in poi in modo ottimale.
Optimal action-value function: $q_*(a, s) = max_\pi q_\pi(s, a)$.

$v_*(s)$ è il valore ottimo, dunque la *consistency condition* può essere scritta come segue:

$$v_*(s) = max_{a \in \mathcal{A}(s)} q_*(s, a) = max_{a \in \mathcal{A}(s)} \Sigma_{s'} p(s'|a, s)(r(s, a, s') + \gamma v_*(s'))$$

In altre parole: prima l'azione era data dalla *policy*, ora l'azione è quella migliore.

Value Iteration

Si trasforma l'equazione di Bellman combinando *policy evaluation* (calcolando v_π di una data policy) e la *policy improvement* (cioè l'aggiornamento dei valori, rendendo π greedy rispetto a v_π). Il metodo risultante **Value Iteration** è una successiva approssimazione.
Salva il valore di ogni stato per produrre la nuova stima della value function ($k+1$ stage).

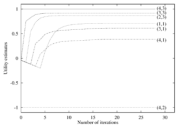
Initialize array v arbitrarily (e.g., $v(s) = 0$ for all $s \in \mathcal{S}^+$)

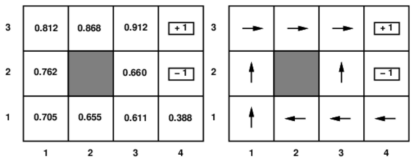
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $temp \leftarrow v(s)$
    $v(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma v(s')]$
    $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
until $\Delta < \theta$ (a small positive number) limite scelto

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma v(s')\right]$

Bellman backup: $v^{k+1}(s) = max_a \Sigma_{s'} P(s'|a,s)(r(a,s,s') + \gamma v^k(s'))$

Example of bellman back-up
$v(1,1) = -0.04$
$+ \gamma$ max{$0.8v(1,2) + 0.1v(2,1) + 0.1v(1,1)$, up
$0.9v(1,1) + 0.1v(1,2)$ left
$0.9v(1,1) + 0.1v(2,1)$ down
$0.8v(2,1) + 0.1v(1,2) + 0.1v(1,1)$} right





Policy is a greedy selection of best action for every state
considering the MPDs dynamics
See policy for state $(3,1)$, $\pi^*((3,1))$ = left but state with highest value is up.

Value Iteration garantisce la convergenza ad una funzione di valore ottimale. *Complessità* quadratica nel numero numero di stati e lineare nel numero di azioni.

Policy Iteration

Si sceglie una policy iniziale arbitraria poi ri ripete il seguente passaggio fino a quando π non viene modificata: dato π calcolare le utilities (*policy evaluation*) e aggiornare π con le nuove utilities (*policy improvement*).

Per calcolare le utilities data una policy π (**policy evaluation**): $v(s) = \Sigma_{s'} p(s'|s,\pi(s))(r(s,\pi(s),s') + \gamma v(s'))$
Si risolvono n equazioni lineari in n incognite (complessità $\mathcal{O}(n^3)$).

Policy improvement: dati i valori di ogni stato $v(s)$, se il valore di uno stato può essere migliorato, si adotta la nuova azione nella policy.

L'algoritmo itera fino a quando non ci sono più miglioramenti. La policy trovata sarà sicuramente ottimale.

1. Initialization
   $v(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$
2. Policy Evaluation
   Repeat
     $\Delta \leftarrow 0$
     For each $s \in \mathcal{S}$:
       $temp \leftarrow v(s)$
       $v(s) \leftarrow \sum_{s'} p(s'|s,\pi(s))\left[r(s,\pi(s),s') + \gamma v(s')\right]$
       $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
   until $\Delta < \theta$ (a small positive number)
3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
     $temp \leftarrow \pi(s)$
     $\pi(s) \leftarrow \arg\max_a \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma v(s')\right]$
     If $temp \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return v and π; else go to 2

La policy iteration converge spesso in poche iterazione ma è molto dispendioso. È necessario modificarla.

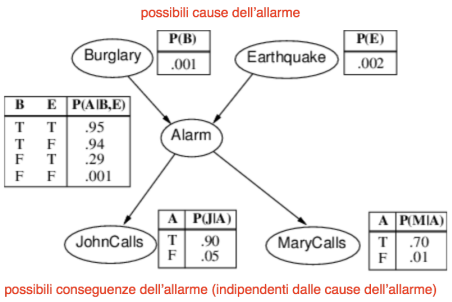
Modified Policy Iteration: si fanno alcuni passi di *value iteration* (ma con π fissato) partendo dalla value function prodotta l'ultima volta, per produrre un passo di valutazione con policy approssimata.
Converge molto più velocemente di *Value Iteration* e *Policy Iteration*.

Bayesian Network

Una notazione grafica per le asserzioni di indipendenza condizionale e dunque per la specifica compatta della *full joint distribution*; sintassi:

- un insieme di nodi, uno per variabile;
- un grafo diretto aciclico (il collegamento indica "influenza direttamente");
- una distribuzione condizionale per ogni nodo, dati i suoi padri $P(X_i|Parents(X_i))$

Nel caso più semplice, una distribuzione condizionale è rappresentata come una *Conditional Probability Table (CPT)*, dando una distribuzione su X_i per ogni combinazione di valori del padre.



Usando questa network e dividendo dunque le tabelle dei vari nodi, abbiamo una CPT per ogni nodo X_i con 2^k righe (k = parents(X_i). Se ogni nodo non ha più di k padri, la rete completa richiede $\mathcal{O}(n * 2^k)$, invece di $\mathcal{O}(2^n)$.

Global Semantics: definisce la *full joint distribution* come il prodotto delle distribuzioni locali condizionali.

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i|parents(X_i))$$

e.g., $P(j \wedge m \wedge a \wedge \neg b \wedge \neg e)$

$$\begin{aligned} &= P(j|a)P(m|a)P(a|\neg b, \neg e)P(\neg b)P(\neg e) \\ &= 0.9 \times 0.7 \times 0.001 \times 0.999 \times 0.998 \\ &\approx 0.00063 \end{aligned}$$

Local Semantics: ogni nodo è condizionalmente indipendente dai suoi non-discententi, dati i suoi padri.
Teorema: *Local Semantics* $\iff$ *Global Semantics*

Markov Blanket: ogni nodo è condizionalmente dipendente dai nodi presenti nella sua markov blanket.

Markov blanket = *parents + children + children's parents*

Costruzione di una Bayesian Network

È necessario un metodo tale che una serie di asserzioni localmente verificabili di indipendenza condizionale garantisca la semantica globale richiesta.

- si scegli un ordine di variabili $X_1, \ldots, X_n$;
- per i da 1 a n, si aggiunge X_i alla rete e si selezionano i padri tali che $P(X_i|Parents(X_i)) = P(X_i|X_1, \ldots, X_{i-1})$

$$\begin{aligned} \mathbf{P}(X_1, \ldots, X_n) &= \prod_{i=1}^{n} \mathbf{P}(X_i|X_1, \ldots, X_{i-1}) \quad \text{(chain rule)} \\ &= \prod_{i=1}^{n} \mathbf{P}(X_i|Parents(X_i)) \quad \text{(by construction)} \end{aligned}$$

La scelta dei padri garantisce la semantica globale.

$P(J|M) = P(J)$? No
$P(A|J, M) = P(A|J)$? $P(A|J, M) = P(A)$? No
$P(B|A, J, M) = P(B|A)$? Yes
$P(B|A, J, M) = P(B)$? No
$P(E|B, A, J, M) = P(E|A)$? No
$P(E|B, A, J, M) = P(E|A, B)$? Yes

Nell'esempio abbiamo scelto prima gli effetti e poi le cause, rendendo complessa la decisione per le indipendenze condizionali. Il grafico inoltre diventa meno compatto e più confuso (aumentando le righe delle rispettive tabelle).

Compact Conditional Distributions

Le *CPT* crescono esponenzialmente nel numero dei padri. La soluzione sono delle *distribuzioni canoniche compatte*, che rendono lineare il n° di parametri nel n° di padri.
Noisy-Or distributions: è una rappresentazione delle cause indipendenti tra loro. Vengono considerate solo alcune cause, dunque si può aggiungere un *leak node* per raggruppare tutte le altre possibili.

raffreddore influenza

Cold	*Flu*	*Malaria*	*P(Fever)*	*P(¬Fever)*
F	F	F	0.0	1.0 (in grassetto: non c'è febbre e solo una malattia (mi interessano questi casi))
F	F	T	0.9	**0.1**
F	T	F	0.8	**0.2** (Da questi ottengo gli altri casi moltiplicando)
F	T	T	0.98	$0.02 = 0.2 \times 0.1$
T	F	F	0.4	**0.6**
T	F	T	0.94	$0.06 = 0.6 \times 0.1$
T	T	F	0.88	$0.12 = 0.6 \times 0.2$
T	T	T	0.988	$0.012 = 0.6 \times 0.2 \times 0.1$

Simple Query: calcola a posteriori probabilità marginali (togliendo le variabili che non interessano) $P(X_i|E = e)$
Ci sono due algoritmi per le simple query: *Enumeration Algorithm* e *Variable Elimination*.

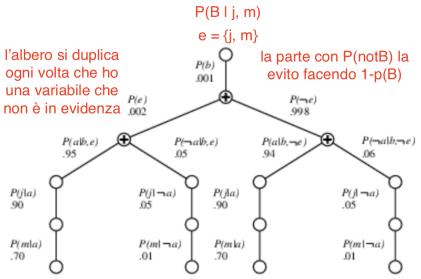
Inference by Enumeration: un metodo utile per sommare delle variabili del joint senza darne la rappresentazione esplicita. Riscrive la *full joint* usando il prodotto delle righe della CPT. I nodi dipenderanno solo dai padri.

$$P(B|j,m) = \alpha P(B)\ \Sigma_e P(e)\ \Sigma_a P(a|B,e)P(j|a)P(m|a)$$

Enumeration Algorithm

```
function Enumeration-Ask(X, e, bn) returns a distribution over X
  inputs: X, the query variable
          e, observed values for variables E
          bn, a Bayesian network with variables {X} ∪ E ∪ Y
                                                  (variabili nascoste)
  Q(X) ← a distribution over X, initially empty
  for each value x_i of X do   (cioè metto xi=T/F e lo metto tra le evidenze)
      extend e with value x_i for X
      Q(x_i) ← Enumerate-All(Vars[bn], e)
  return Normalize(Q(X))
```

```
function Enumerate-All(vars, e) returns a real number
  if Empty?(vars) then return 1.0
  Y ← First(vars)
  if Y has value y in e   (se Y non ha valore nell'evidenza)
    then return P(y | Pa(Y)) × Enumerate-All(Rest(vars), e)
    else return Σ_y P(y | Pa(Y)) × Enumerate-All(Rest(vars), e_y)
  (marginalizzo)  where e_y is e extended with Y = y  (aggiungo ey all'evidenza)
```



L'enumerazione è inefficiente perchè ho calcoli ripetuti.

Variable Elimination

Risolve le sommatorie da destra a sinistra, tenendo i risultati parziali (*fattori*) per evitare calcoli ripetuti.

```
function Elimination-Ask(X, e, bn) returns a distribution over X
  inputs: X, the query variable
          e, evidence specified as an event
          bn, a belief network specifying joint distribution
P(X_1, ..., X_n)

  factors ← []; vars ← Reverse(Vars[bn])
  for each var in vars do
      factors ← [Make-Factor(var, e)|factors]
      if var is a hidden variable then factors ← Sum-Out(var, factors)
  return Normalize(Pointwise-Product(factors))
```

$$\begin{aligned}
&\mathbf{P}(B|j,m)\\
&= \alpha \underbrace{\mathbf{P}(B)}_{B} \Sigma_e \underbrace{P(e)}_{E} \Sigma_a \underbrace{\mathbf{P}(a|B,e)}_{A} \underbrace{P(j|a)}_{J} \underbrace{P(m|a)}_{M}\\
&= \alpha \mathbf{P}(B) \Sigma_e P(e) \Sigma_a \mathbf{P}(a|B,e) P(j|a) f_M(a)\\
&= \alpha \mathbf{P}(B) \Sigma_e P(e) \Sigma_a \mathbf{P}(a|B,e) f_J(a) f_M(a)\\
&= \alpha \mathbf{P}(B) \Sigma_e P(e) \Sigma_a f_A(a,b,e) f_J(a) f_M(a)\\
&= \alpha \mathbf{P}(B) \Sigma_e P(e) f_{\bar{A}JM}(b,e) \text{ (sum out } A\text{)}\\
&= \alpha \mathbf{P}(B) f_{\bar{E}\bar{A}JM}(b) \text{ (sum out } E\text{)}\\
&= \alpha f_B(b) \times f_{\bar{E}\bar{A}JM}(b) \quad \text{(avrò due valori: uno per b=T e uno per b=F)}
\end{aligned}$$

Irrelevant Variables

Data $P(B|j,m) = \alpha P(B)\ \Sigma_e P(e)\ \Sigma_a P(a|B,e)P(j|a)P(m|a)$, la variabile m è irrilevante (somma $= 1$).
Teorema 1: Y è irrilevante a meno che $Y \in$ *Anchestors*$(\{X\} \cup E)$.
Moral Graph *di una Bayes Net*: si collegano i padri e si tolgono le frecce (non è più un grafo diretto).
A è ***m-separato*** da B a causa di C, sse sono separati da C nel *moral graph*.
Teorema 2: Y è irrilevante se è E lo m-separa da X. $P(J|A = true)$, B ed E sono irrilevanti.

Approximate Inference by Stocastic Simulation

Prior Sampling: si prende un ordine topologico e si fanno i vari *sample* (sequenze) tenendo conto della probabilità. Per ogni sequenza si calcola la probabilità stimata $\hat{P}(F,F,F,T,F) = \frac{X}{N}$, con $X =$ n° di occorrenze del *sample*.
Rejection Sampling: rigetto i *sample* che non sono in accordo con l'evidenza. Per ogni *sample* valido computo $N[J = F] = +1$ oppure $N[J = T] = +1$, aggiungendo ad $\langle 0, 0\rangle$. Infine si normalizza trovando il $P(J|b)$ cercato.

Reinforcement Learning

Idea di base: imparare a mappare le situazioni alle azioni, per massimizzare la sequenza di reward.
Si fanno tentativi/test ed errori mentre si interagisce con l'ambiente. Interesse per i reward a lungo termine.
Una MDP che non conosce le dinamiche non sa quali sono gli stati buoni e dove portano le azioni.
Ci sono due approcci, uno fa uso di modelli, l'altro di campioni:

- ***Model-Based***: cerca di imparare un modello evitando stati/azioni non buoni, riducendo i passi di esecuzione e facendo un uso efficiente di dati.
- ***Model-Free***: impara *Q-function* e *policy* direttamente; è più semplice perché non ha bisogno di un modello.

Esempio: si vuole calcolare la media attesa dell'età di una classe ($\mathbb{E}[A] = \Sigma_a P(a)\ a$), con i due approcci.

- ***Model-Based***: si stima $\hat{P}(a) = \frac{num(a)}{N}$, dove $num(a)$ è il n° di studenti con età a. Abbiamo $\mathbb{E}[A] \approx \Sigma_a P(a) * a$
- ***Model-Free***: non c'è stima. Abbiamo $\mathbb{E}[A] \approx \frac{1}{N}\Sigma_i a_i$ dove a_i è l'età della persona i presa da un campione.

Model-Based

Si stima $P(x)$ dai sample acquisiti $x_i \sim P(x)$. Si stima $\hat{P}(x) = count(x)/k$. Si stima $\hat{T}(s,a,s')$ dai sample acquisiti $(s_0, a_0, s_1, a_1, s_2 \dots)$. Si stima il modello $\hat{T}(s,a,s') = \frac{count(s_{t+1}=s', a_t=a, s_t=s)}{count(s_t=s, a_t=a)}$.

Algorithm 1 Model Based approach to RL

Require: $[A, S], S_0$
Ensure: $\hat{T}, \hat{R}, \hat{\pi}$
  Initialize $\hat{T}, \hat{P}, \hat{\pi}$
  repeat
    Execute π for a learning episode
    Acquire a sequence of tuples $\langle (s, a, s', r) \rangle$
    Update $\hat{T}$ and $\hat{R}$ according to tuples $\langle (s, a, s', r) \rangle$
    Given current dynamics compute a policy (e.g., VI or PI)
  until termination condition is met

La durata del *learning episode* è data da quando uno stato terminale viene raggiunto oppure vengono fatti un certo numero di time-step.

Esegue sempre l'azione migliore, dato il modello corrente; non c'è esplorazione.

Model-Free

Si vuole calcolare un'aspettativa pesata di $P(x)$: $\mathbb{E}[f(x)] = \Sigma_x P(x)\ f(x)$.
Questo approccio stima l'aspettativa direttamente dai sample: $x_i \sim P(x), \mathbb{E}[f(x)] \approx \frac{1}{N}\Sigma_i f(x_i)$.

Si valuta la *Value Function* dall'esperienza.: si esegue π per alcuni *learning episodes*, si calcola il *discounted* reward ogni volta che si visita uno stato e infine si calcola la media sui sample.

Sample-Based Policy Evaluation:data la formula di Bellman $V_\pi^{k+1}(s) = \Sigma_{s'} T(s, \pi(s), s')(R(s, \pi(s), s') + \gamma V_\pi^k(s'))$
Questa viene applicata ad N sample (rimuovendo T): $sample_i = R(s, \pi(s), s'_i) + \gamma V_\pi^k(s'_i)$
Si fa calcola poi la media su tutti i sample: $V_\pi^{k+1}(s) = \frac{1}{N}\ \Sigma_i\ sample_i$

Temporal Difference Learning: si impara da ogni esperienze, non dopo un episodio. Si aggiorna $V(s)$ dopo ogni azione, dato (s, a, s', r) ottenuto.
Update $V_\pi(s)$: $V_\pi(s) \leftarrow (1-\alpha)V_\pi(s) + \alpha(R(s, \pi(s), s') + \gamma V_\pi(s')))$ decrementando α ad ogni passo.

Q-Learning

Si vuole calcolare la policy basata su $V(s)$ senza usare direttamente V.
Q-Learning: sample based Q-Value iteration.

$Q_{k+1}(s,a) = \Sigma_{s'}\ T(s,a,s')(R(s,a,s') + \gamma max_{a'}\ Q_k(s',a'))$

Proprietà di Q-Learning: converge alla policy ottima se si esplora abbastanza e se si rende il *learning rate* piccolo abbastanza (ma non decresce così velocemente).
Off Policy Learning: imparare la policy ottimale senza seguirla (la scelta delle azioni non impatta sulla convergenza).

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    $S \leftarrow S'$;
  until S is terminal

ϵ-greedy: sceglie la miglior azione la maggior parte delle volte, ma alcune volte (con probabilità ϵ) scegli un'azione in modo randomico tra tutte le azioni (con uguale probabilità).

SARSA

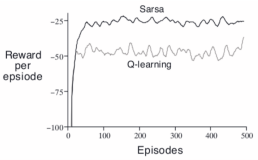
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

SARSA: deriva dalla tupla (S, A, R, S', A')

È caratterizzata dal fatto che si calcola l'azione successiva basandosi sulla policy (on-policy).

Q-Learning impara la policy ottimale ma occasionalmente fallisce a causa della selezione dell'azione *ϵ-greedy*.

SARSA, essendo on-policy, ha una migliore on-line performance.



The Exploration vs Exploiting Dilemma

- *Exploration*: serve per migliorare la value function e massimizzare il reward; però esplorare parti irrilevanti causa una perdita di tempo.
- *Exploiting*: usare la conoscenza per non perdere tempo esplorando parti irrilevanti.

Ci sono due metodi principali:

- *ϵ-greedy*: scegliere in modo greedy la maggior parte delle volte (probabilità $1 - \epsilon$) e in maniera randomica con probabilità ϵ.
- *Soft-max (or Boltzmann)*: