Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Benchmark di sistemi per la programmazione Bayesiana: tempo, accuratezza, concisione

Relatore:
Chiar.mo Prof.
UGO DAL LAGO

Presentata da:
VALENTINA FERRAIOLI

Sessione II
Anno Accademico 2018/2019

Questa è la DEDICA*:*
ognuno può scrivere quello che vuole,
anche nulla …

Introduzione

Elenco delle figure

Elenco delle tabelle

Indice

Capitolo 1

Probabilità Bayesiana

1.1 Probabilità

La probabilità è una misura che permette di quantificare la possibilità che un evento si verifichi oppure no. Ha lo scopo di formulare e studiare modelli matematici che permettono di descrivere il comportamento di un evento, nonostante non si sia in grado di predirlo con totale certezza. Secondo la definizione classica la probabilità di un evento casuale è data dal rapporto tra il numero di casi favorevoli al presentarsi di tale evento ed il numero totale di casi possibili, supponendo che tutti i possibili casi siano equiprobabili. In termini matematici:

$$P(E) = \frac{n_E}{n} \tag{1.1}$$

dove n_E corrisponde al numero di volte in cui potrebbe verificarsi E ed n è il numero di esperimenti fatti. Tuttavia è possibile riconoscere due diverse interpretazioni della probabilità:

- interpretazione frequentista: la probabilità viene definita in modo oggettivo come il rapporto tra il numero di volte in cui si verifica l'evento di interesse ed il numero totale di prove effettuate, per un numero di prove che tende all'infinito. In termini matematici:

$$P(E) = \lim_{x \to \infty} \frac{n_e}{n} \tag{1.2}$$

- interpretazione bayesiana: la probabilità viene definita in modo soggettivo, infatti alle ipotesi viene associata una probabilità che indica il grado di fiducia che un individuo ha nel verificarsi di un certo evento, il quale dipende dalle informazioni che ha riguardo l'evento.

Non si può definire se un approccio sia migliore dell'altro, ma si può notare che entrambe le interpretazioni soddisfano gli assiomi della probabilità:
Assumendo che Ω sia l'insieme di casi possibili, che $|\Omega| = n$ sia la sua cardinalità e che E_i, con i=1,2..,n siano gli eventi di Ω, allora:

$$0 \leq P(E_i) \leq 1 \tag{1.3}$$

$$P(\Omega) = 1 \tag{1.4}$$

$$P(\cup_{n\to\infty}^{\infty} E_i) = \Sigma_{n\to\infty}^{\infty} P(E_i) \tag{1.5}$$

La differenza sostanziale dei due approcci si deve cercare nel concetto che viene assegnato alla probabilità, che inciderà sull'utilizzo e l'importanza del teorema di Bayes, anch'esso comune nelle due interpretazioni.

In questa sezione verrà analizzata in maniera più approfondita la probabilità e l'inferenza Bayesiana. Per cominciare verranno richiamate alcune definizioni che sono alla base di questa, quali il teorema di Bayes, probabilità condizionata, distribuzione a priori e a posteriori.

1.2 Richiamo definizioni

1.2.1 Distribuzione

La funzione di distribuzione descrive la probabilità che un evento assuma un particolare valore.

1.2.2 Modello probabilistico:

Fornisce una descrizione degli eventi che si possono verificare e della probabilità con il quale ci si aspetta si verifichino.

1.2.3 Probabilità condizionata

In teoria della probabilità la probabilità condizionata di un evento A rispetto a un evento B è la probabilità che si verifichi A, sapendo che B si è verificato. Viene indicata come:

$$P(A|B) = \frac{P(A,B)}{P(A)} \tag{1.6}$$

dove $P(B) \neq 0$ e P(A,B) indica la probabilità congiunta.

La probabilità condizionata esprime una "correzione" delle aspettative per A, dettata dall'osservazione di B.

1.2.4 Probabilità congiunta

La probabilità congiunta indica la probabilità che due eventi A e B si verifichino nello stesso momento. Si indica come:

$$P(A,B) = P(A|B)P(A) \tag{1.7}$$

Nel caso in cui due eventi siano indipendenti, ovvero conoscere lo stato dell'evento A non ci fornisce alcuna informazione in più sullo stato di B, la probabilità congiunta equivale a:

$$P(A,B) = P(A)P(B) \tag{1.8}$$

1.2.5 Marginalizzazione

Conoscendo la distribuzione congiunta P(A,B) è possibile ricavare la distribuzione di P(X) usando la marginalizzazione

$$P(A) = \Sigma_B P(A,B)) \tag{1.9}$$

1.2.6 Teorema di Bayes

Il teorema di Bayes mette in relazione la probabilità condizionata con quella congiunta.

Enunciato: La probabilità a posteriori è uguale alla probabilità a priori per likelihood ratio.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}) \tag{1.10}$$

dove:

- probabilità a priori: corrisponde a P(A), ovvero alla probabilità che l'evento A si verifichi prima di prendere in considerazione le osservazioni.
- probabilità a posteriori: corrisponde a $P(A|B)$, ovvero alla probabilità che l'evento si verifichi avendo considerato le osservazioni. Può essere considerata un aggiustamento della probabilità a priori.
- funzione di verosomiglianza (likelihood): corrisponde a $P(B|A)$,ovvero alla probabilità che si verifichi un evento B, avendo osservato A .

1.3 Probabilistic Reasoning

Probabilistic reasoning combina la nostra conoscenza rispetto ad una situazione con le regole della probabilità. Determina fattori critici che non sono stati analizzati,ai fini di prendere una buona decisione in situazioni di incertezza.
Per fare ciò si ha bisogno di definire un modello, di applicare le evidenze possedute sul modello e di trarre delle conclusioni. Utilizzando il probabilistic reasoning è possibile eseguire tre diversi tipi di ragionamento:

- predire eventi futuri: attraverso le osservazioni possedute in questo momento si cerca di predire la probabilità che un evento accada.
- inferire le cause agli effetti: abbiamo un osservazione sul presente e vogliamo definire la probabilità che alcuni eventi si siano verificati.
- imparare da eventi passati per predire in modo migliore eventi futuri: vengono prese tutte le informazioni riguardo a situazioni precedenti più quella corrente, verrà fatta inferenza che migliorerà i risultati dell'evento predetto.

Inferenza

Il processo di inferenza permette partendo da un modello di rispondere a domande basate sull'osservazione di alcuni eventi. E' un processo che viene fatto quotidianamente dalla mente umana in situazioni di incertezza, per esempio:
"Attraversando la strada tutti abbiamo fatto ciò: Sei di fretta, quindi invece di arrivare alle prossime strisce pedonali guardi a destra e a sinistra e vedi una macchina ad una distanza sufficiente perchè tu riesca ad attraversare la strada senza essere investito. Quindi tu cammini (e immagino) attraversi la strada illeso.[Las12]

In questo esempio non si sa con certezza che la macchina vista non stesse andando troppo veloce da investirci eppure conosciamo un modello, composto anche da esperienze passate, e nel vedere la macchina ad una certa distanza e con una certa velocità , riusciamo a decidere se attraversa la strada senza rischiare di essere investiti. In queste situazioni si sta compiendo un processo di inferenza.

1.3.1 Bayesian reasoning

Come per la definizione di probabilità anche il probabilistic reasoning si divide in Bayesian e Frequentist reasoning.
Nella prospettiva bayesiana, abbiamo detto che la probabilità descrive l'incertezza con il quale è possibile prevedere l'esito del fenomeno in questione e che questa incertezza può derivare dall'intrinseca variabilità del processo, ma anche dalla nostra imperfetta conoscenza di esso.
Ora l'idea base è quella di fare inferenza (bayesiana), ossia sviluppare delle previsioni (probabilità a posteriori), su quantità non note in base all'evidenza disponibile e a un'ipotesi iniziale(probabilità a priori). "E' possibile fare tutto ciò utilizzando il teorema di Bayes che viene interpretato in questo modo:

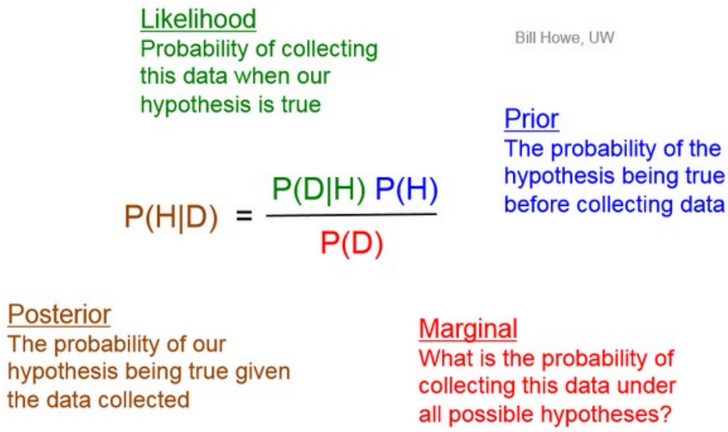


Figura 1.1: Bayes theorem.
Fonte:

1.3.2 Analisi Bayesiana

L'analisi Bayesiana consiste quindi nel:

- creare un modello di probabilità che rappresenti i dati
- identificare le evidenze
- eseguire l'inferenza

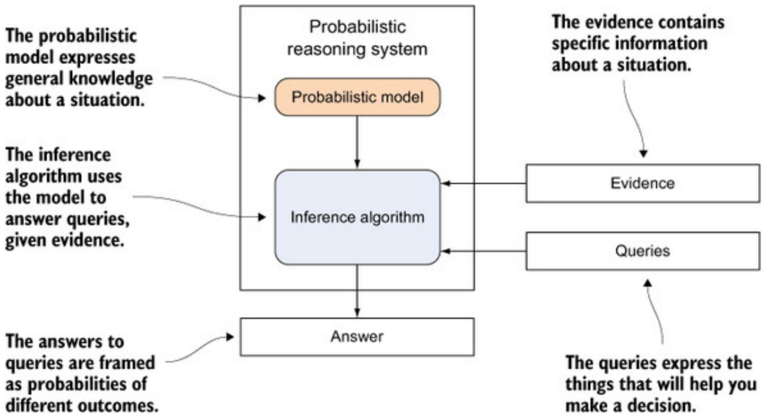


Figura 1.2: Bayes theorem.
Fonte:"https://dpzbhybb2pdcj.cloudfront.net/pfeffer/Figures/01fig03alt.jpg"

Esempio 1:

Un giorno Tracey si sveglia è realizza che l'erba del suo giardino è bagnata. Avrà piovuto durante la notte oppure si è dimenticata di spegnere l'irrigatore? Dopodichè si rende conto che anche l'erba del suo vicino è bagnata. Qual è la probabilità che Tracey abbia lasciato l'irrigatore acceso durante la notte?

Modello:

$R \in 0, 1 \rightarrow R = 1$ ha piovuto, 0 altrimenti
$S \in 0, 1 \rightarrow S = 1$ Tracey ha lasciato l'irrigatore acceso, 0 altrimenti
$J \in 0, 1 \rightarrow J = 1$ l'erba di Jack *è* bagnata, 0 altrimenti
$T \in 0, 1 \rightarrow T = 1$ l'erba di Tracey *è* bagnata, 0 altrimenti

Il modello per il problema di Tracey corrisponde alla probabilita congiunta di P(T,J,R,S). Per procedere con la risoluzione *e'* necessario, inoltre, definire il tipo di dipendenza che c'è tra i diversi eventi: $P(T|S, R)$ e $P(J|R)$.

$$P(T, J, R, S) = P(J|S, R)P(J|R)P(R)P(S)$$

Queste relazioni possono essere rappresentate anche in modo grafico attraverso una Bayes Network, che verrà spiegata approfonditamente successivamente.

Dal problema evince che T=1 e R=1. Supponiamo che le probabilità a priori siano: P(R)=0.2 P(S)=0.1 Mentre le altre probabilità corrispondono a:

$$p(J = 1|R = 1) = 1, p(J = 1|R = 0) = 0.2$$

$$p(T = 1|R = 1, S = 0) = 1, p(T = 1|R = 1, S = 1) = 1$$

$$p(T = 1|R = 0, S = 1) = 0.9, p(T = 1|R = 0, S = 0) = 0$$

.

A questo punto si può iniziare ad eseguire l'inferenza andando ad applicare il teorema di Bayes. L'obiettivo è calcolare P(S—T,J). Iniziamo con il calcolare P(S—T) e vediamo come cambiano le due probabilità con l'aggiunta di un'evidenza.

$$
\begin{aligned}
P(S|T) = \frac{P(S,T)}{P(T)} &= \frac{\Sigma_{J,R}\, P(T,J,R,S)}{\Sigma_{J,R\;,S}\, P(T,J,R,S)} \\
&= \frac{\Sigma_{J,R}\, P(J|R)P(T|R,S)P(R)P(S)}{\Sigma_{J,R\;,S}\, P(J|R)P(T|R,S)P(S)} \\
&= \frac{\Sigma_R P(T|R,S)P(R)P(S)}{\Sigma_{R,S}\, P(T|R,S)P(S)P(R)} = 0,3340
\end{aligned}
$$

Si può notare che la probabilità a posteriori è aumentata rispetto a quella a priori, tenendo conto dell'evidenza che T = 1.

Ora andiamo a calcolare $P(S|T,J)$ e vediamo come si comporta la probabilità a posteriori in questo caso.

$$
\begin{aligned}
P(S|T,J) = \frac{(S,T,J)}{P(T,J)} &= \frac{\Sigma_R P(T,J,R,S)}{\Sigma_{R,S}\, p(T,J,R,S)} \\
&= \frac{\Sigma_R P(J|R)P(T|R,S)P(R)P(S)}{\Sigma_R P(J|R)P(T|R,S)P(R)P(S)} = 0.160
\end{aligned}
$$

La probabilità che Tracey abbia lasciato l'irrigatore acceso, data l'evidenza che anche l'erba di Jack è bagnata è inferiore di quella che tiene conto solo dell'erba di Tracey. Questo perchè l'osservazione che l'erba di Jack è bagnata aumenta la probabilità che sia vero che abbia piovuto.

1.4 Belief Network

Definizione:
Una belief Network è una distribuzione della forma:

$$P(x_1, ..., x_n) = \Pi_{i=1}^{D}(x_i|Pa(x_1)) \tag{1.11}$$

dove $Pa(x_i)$ equivale alle variabili parenti di x_i.

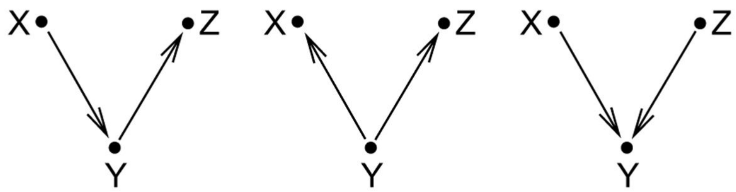
Come è stato accennato precedentemente è possibile rappresentare un modello probabilistico attraverso una Belief Network(Bayesian Belief Network) che non è altro che un DAG (Direct Acyclic Graph) che è in grado di rappresentare le dipendenze condizionali di un modello.

All'interno della Belief network ogni nodo rappresenta una variabile ed è associato ad una funzione di probabilità,che prende in input un insieme di possibili valori che potrebbe assumere il nodo padre e restituisce la distribuzione di probabilità del nodo. Gli archi, invece, rappresentano le dipendenze condizionali delle variabili. L'assenza di un cammino tra due nodi implica che le due variabili sono condizionalmente indipendenti. Rappresentare la dipendenza tuttavia non è così immediato, infatti bisognerebbe tenere conto di alcuni pattern, quali:

- $A \rightarrow B \rightarrow C$: viene chiamato "chain ". A primo impatto si potrebbe pensare che il valore di A influenzi C attraverso B, ma non è così. Infatti, è osservabile che condizionando la varibile B =1 il valore di C non varierà indipendentemente dal valore che assume B. Quindi le variabili A e C sono indipendenti.

- $A \leftarrow B \rightarrow C$: viene chiamato "fork". Anche in questo caso abbiamo che A e C sono indipendenti, nonostante sembrerebbero essere correlati attraverso B.

- $A \rightarrow B \leftarrow C$: viene chiamato "collider". A e C sono indipendenti in linea generale, ma condizionando B diventano dipendenti. Più precisamente si ha che A e B sono inversamente proporzionali, quindi un valore alto di A spiega il perchè di un valore basso di C.



chain $X \perp Z \mid Y$ fork $X \perp Z \mid Y$ collider $X \not\perp Z \mid Y$

Figura 1.3: Fonte:
https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-8-S6-S5/figures/3

le Bayesian Network si prestano bene a fare inferenza, sfruttano la probabilità condizionale, del quale dispongono già, data la definizione di belief network. A partire da questa viene fatta la valutazione della variabile desiderata. Per fare inferenza, si possono utilizzare anche altre tecniche, come ad esempio MCMC, che si presta bene ad inferire su reti di grandi dimensioni.

Esempio 2:

Si andrà a rappresentare una belief network per l'esempio 1.

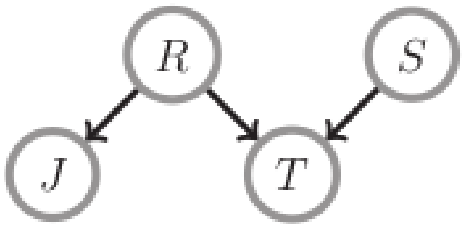


Figura 1.4

Ogni nodo del grafo rappresenta una delle variabili della probabilità congiunta, e le variabili che alimentano altre, come ad esempio R e S , rappresentano le variabili che si trovano alla destra in una probabilità condizionata. E' possibile vedere che vengono anche rispettati i pattern spigati sopra; infatti R e S sono indipendenti e non presentano cammini tra essi, mentre J che è dipendente da R viene raggiunto solo da R e T che dipende da R e S viene raggiunto da entrambi.

Capitolo 2

Probabilistic Programming

Finora si è arrivati alla conclusione che grazie alla probabilità e all'inferenza è possibile ragionare in condizioni di incertezza. Negli ultimi anni c'è stato un grande sviluppo in campi come Artificial Intelligence e Data Science, nei quali si ha bisogno di fare questo tipo di ragionamento , ma soprattutto si ha la necessità di lavorare su modelli di dimensione sempre maggiore. Per semplificare la modellizzazione e rendere l'inferenza automatica in questo tipo di problemi è stata introdotta la programmazione probabilistica. Per eseguire la programmazione probabilistica è necessario utilizzare dei PPL (Probabilistic programming language), che permettono di esprimere i modelli come programmi e non come costrutti matematici e possiedono una suite di algoritmi di inferenza che potranno essere applicati ai programmi.
Tutti questi programmi nascono come estensioni di linguaggi tradizionali, ad esempio, il linguaggio logico problog, basato su prolog e anglican, linguaggio funzionale che estende Clojoure.

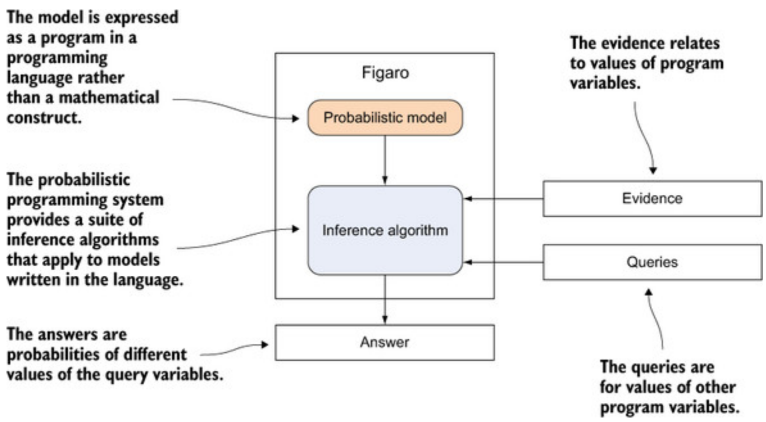


Figura 2.1: Bayes theorem.
Fonte:https://dpzbhybb2pdcj.cloudfront.net/pfeffer/Figures/01fig07alt.jpg

Lo scopo di questa sezione sarà quella di approfondire alcuni linguaggi probabilistici; per questo verranno analizzati i paradigmi dal quale nascono e il modo che impiegano per affrontare la programmazione probabilistica.

2.1 Il paradigma imperativo

La programmazione imperativa è un paradigma di programmazione in cui il programma consiste in una sequenza di istruzioni (statement) che permettono di arrivare al risultato desiderato.
Gli statement sono il costrutto principale di questo tipo di programmazione perchè permettono di cambiare lo stato di un programma, che potrebbe influire sul flusso di operazioni da eseguire. Questo cambiamento di stato prende il nome di side-effect.
Un tipo particolare di programmazione imperativa è quella procedurale che prevede la divisione del codice in "blocchi" che possono essere riutilizzati in punti diversi del programma e una strutturazione del programma più naturale.

Esempio:

```
int factorial(int n){
    int result = 1;
```

```
    for( ; i < n; (n --){
        result*=n;
    }
    return result;
}
int main(){

    result=factorial(10)

}
```

2.2 Il paradigma orientato agli oggetti

La programmazione ad oggetti è un paradigma di programmazione basato sul concetto di "oggetto". Per poter spiegare questo tipo di paradigma abbiamo bisogno di definire:

- **oggetto:** può essere visto come una capsula che contiene dei dati(attributi) e delle operazioni con cui manipolarli(metodi) e che fornisce un' interfaccia attraverso il quale è possibile accedere all'oggetto.
- **classe:** contiene la dichiarazione degli attributi e dei metodi di un oggetto.

. Un oggetto è, quindi, una particolare istanza della classe. La programmazione ad oggetti fornisce tre meccanismi particolari:

- **incapsulamento:** consiste nella separazione dell'interfaccia di una classe dalla sua implementazione. Questo fa si che un utente possa definire un oggetto ed utilizzarlo senza però conoscerne l'implementazione.
- **ereditarietà:** permette di definire una classe(sottoclasse) a partire da un'altra(superclasse). La sottoclasse eredita attributi e metodi della superclasse, ma può anche definirne di e ridefinire alcuni metodi della superclasse.

- **polimorfismo:** permette di utilizzare lo stesso codice con istanze di classi diverse. E' utile quando la versione del metodo da utilizzare dipende dal valore assunto da una variabile a runtime.

  Esempio:

```
class Math{
    public:
        float piGreco= 3,14
        int n=10
        int factorial()
}
int Math::factorial(){
    int result = 1;
    for( ; i < this.n; (this.n --){
        result*=this.n;
    }
    return result;
}
```

2.3 Il paradigma funzionale

La programmazione funzionale è un paradigma di programmazione dichiarativo, ovvero, viene chiesto cosa bisogna calcolare, descrivendo la logica di una computazione senza definire come deve essere calcolato, ovvero viene omesso l'intero flusso di controllo. In questo tipo di programmazione gli statement vengono sostituiti con delle espressioni e la computazione consiste nella valutazione di queste attraverso la riscrittura. La riscrittura è alla base del paradigma funzionale e consiste nel semplificare ripetutamente un'espressione sino a raggiungere una forma non ulteriormente riducibile. Inoltre, ha la caratteristica di apportare modifiche soltanto in locale, non avendo quindi bisogno della

nozione di memoria, che invece è necessaria nei linguaggi convenzionali.
Oltre questo la programmazione funzionale dispone di altri elementi caratterizzanti, quali:

- funzioni di ordine superiore: sono funzioni che possono ricevere altre funzioni come parametro o che possono restituire altre funzioni.

- funzioni pure: non hanno side-effects, questo evita le dipendenza di una funzione da altre, consentendo di modificare una funzione senza influire su altre.

- ricorsione: è il costrutto più utilizzato per eseguire dei cicli all'interno della programmazione funzionale.

- trasparenza referenziale: nei linguaggi funzionali non sono presenti assegnamenti, questo permette di essere certi che quando una funzione viene valutata in un certo ambiente verrà ritornato sempre lo stesso valore.

Esempio

Linguaggio imperativo vs funzionale:
Funzionale:

```
fact::Integer -> Integer
fact 0 = 1
fact n | n > 0 = n * fact (n - 1)
```

E' possibile vedere confrontando il programma che calcola il fattoriale nei due paradigmi come in quello funzionale viene data solamente la definizione matematica di fattoriale, mentre in quello imperativo vengono esplicitati tutti i passi da eseguire per poter arrivare al calcolo del fattoriale. Da notare, che anche il programma imperativo potrebbe essere riscritto utilizzando la ricorsione, ma si vuole sottolineare come nel paradigma imperativo si ha la possibilità di scegliere mentre in quello logico si è costretti ad utilizzarla.

2.4 Il paradigma logico

Il paradigma logico, come quello funzionale, fa parte della famiglia dei linguaggi dichiarativi ed è basato sulla logica del prim'ordine.
La nozione alla base di questo paradigma corrisponde ad un' equazione:

$$Algoritmo = logica + controllo$$

infatti, algoritmi di questo tipo prevedono una netta separazione tra la logica, che definisce "cosa" debba essere fatto ed è implementata dal programmatore, e il controllo, che definisce "come" arrivare alla soluzione. La computazione, di cui se ne occupa il controllo, è vista come deduzione logica, tant' è che viene utilizzata una regola di deduzione per individuare, all'interno dello spazio delle possibili soluzioni, quella che rappresenta la "logica" specificata.
i programmi logici sono composti da:

- **regole**: corrispondono a delle implicazioni e sono della forma:

  $$H : -A_1, ..., A_n.$$

  La parte a sinistra di :- è detta "testa", mentre quella a destra è detta "corpo ". Questa forma si legge: "testa è vera se corpo è vero".
  (Si noti che la regola termina con un punto. e le virgole sono intese come congiunzioni)

- **fatti**: Un singolo termine (anche composto), senza il segno ":-", viene chiamato fatto. I fatti equivalgono a regole senza corpo, che sono considerate automaticamente vere. Un esempio di fatto è:
  gatto(tommaso).

Questo genere di programmi sono utilizzati per verificare la correttezza di un programma e per ottimizzare programmi di tipo imperativo, infatti, e possibile passare da un programma logico ad uno imperativo che sia più efficiente applicando delle tecniche di trasformazione.

Esempio

Logico:

```
factorial(0,1).
factorial(1,1).

factorial(N,F) :-
    N>0,
    N1 is N-1,
    factorial(N1,F1),
    F is N * F1.
```

Andando ad analizzare il programma che calcola il fattoriale nel paradigma logica si vede che vengono fatte delle " dichiarazioni" sui fatti noti, come ad esempio, che il fattoriale di O ed 1 è uguale ad 1 e successivamente si richiede di calcolare il fattoriale seguendo la regola che ci dice che bisogna calcolare F1* N dove F1 contiene il valore del fattoriale di n-1.

2.5 PPL

Come è stato detto precedentemente, i linguaggi di programmazione probabilistica sono stati introdotti per semplificare la modellizzazione di casi di incertezza e beneficiare degli algoritmi di inferenza che sono stati messi a disposizione.
Come e' stato detto precedentemente, questi linguaggi sono basati su linguaggi gia' affermati, sul quale vengono introdotte primitive per eseguire programmazione probabilistica.
Di linguaggi di questo tipo ne esitono diversi che esprimono un compromesso tra efficienza, espressivita' e perspicacia.
Qui verranno analizzati tre PPl di tipo funzionale: Anglican, Gen e Hakaru e uno di tipo logico, Problog.

2.5.1 Anglican

Anglican è un PPL integrato in Clojoure che a sua volta è un dialetto del linguaggio di programmazione funzionale Lisp.
Anglican è costituito da alcuni costrutti propri che lo rendono un linguaggio probabilistico e all'interno dei quali non vengono riconosciute tutte le funzioni definite in clojure, sebbene sia basato su esso.
Questi costrutti sono:

- **defquery:** E' una macro che abilita il linguaggio anglican all'interno di un modulo clojure. Al suo interno viene rappresentata la probabilità congiunta e si possono trovare i costrutti:
  - **let:** vengono definite le prior probability;
  - **observe:** viene definito il valore delle variabili casuali di cui si è a conoscenza.
  - **valore di ritorno**: valore della distribuzione condizionale della variabile di interesse.
  - sono supportati anche i costrutti clojure **if, when, cond, case, and, or, fn**
- **defm:** permette di definire delle funzioni in anglican all'esterno di defquery e di richiamarle al suo interno
- **sample:** viene usato per "estrarre" un valore dalle distribuzioni associate alle variabili
- **doquery**: usa algoritmi di inferenza per valutare il valore di ritorno richiesto in defquery.

Inoltre, dispone di una vasta gamma di distribuzioni come, ad esempio, quella di Bernoulli, flip, Poisson,

Dalla belief network al codice:

Riprendendo la belief network dell'esempio 2 vediamo come è possibile, a partire da essa, scrivere un programma in anglican che esegua l'inferenza. Un programma anglican

inizia all'interno del costrutto defquery in cui è possibile definire il modello probabilistico tramite il costrutto let. Per prima cosa vengono definite le probabilità a priori, che corrispondono alle variabili indipendenti della nostra belief network ,ovvero ai nodi senza archi entranti. Dopodiche' vengono definite le probabilità condizionate, dove la distribuzione di probabilità da associare alle variabili associando dipende dai valori assunti dalle variabili dei nodi entranti.
A questo punto,il modello è stato definito e si puo' procedere con la definizione delle evidenze. Per fare ciò si usa il costrutto observe," (observe variabile osservazione)" che associa alla variabile il valore di osservazione.
Infine, nel costrutto doquery viene definito l'algoritmo con il quale si vuole fare inferenza sul modello, che può essere del tipo MCMC o Importance Sampling e devono essere specificati il numero di particelle ed il numero di iterazioni da usare all'interno dell'algoritmo. Nel nostro caso useremo un algoritmo MCMC chiamato lmh.

Esempio:

```
(ns bayes-net-raining
  (:require [gorilla-plot.core :as plot]
            [anglican.stat :as s])
  (:use clojure.repl
        [anglican core runtime emit
         [inference :only [collect-by]]]))

(defquery raining-b-net [t-wet-grass j-wet-grass]
  (let [raining  (sample(flip 0.4))

        sprinkler (sample (flip 0.3))
        jack (cond (= raining true)
                        (flip 0.9)
                   (= raining false)
                        (flip 0.2))

        tracey(cond(and (= raining true)
                        (= sprinkler true))
```

```
                        (flip 0.99)
                   (and (= raining false)
                        (= sprinkler false))
                        (flip 0.02)
                   (or  (= raining true)
                        (= sprinkler true))
                        (flip 0.99))]
    (observe tracey t-wet-grass)
    (observe jack j-wet-grass)

    raining))
(->> (doquery :lmh raining-b-net [true false] :number-of-particles 100)
     (take 10000)
     (collect-by :result)
     (s/empirical-distribution)
     (#(plot/bar-chart (keys %) (vals %))))
```

2.5.2 Problog

Problog è un linguaggio probabilistico basato su Prolog, che permette oltre a specificare le relazioni tra i diversi componenti, anche di catturare l'incertezza con il quale essi si possono verificare.
La differenza tra problog e prolog risiede nel tipo di informazione che si vuole ricavare, infatti, mentre in prolog si e' interessati a sapere se una query si verifica o meno in problog si vuole conoscere la probabilita' con il quale essa si verifica.
Problog permette di:

- calcolare qualsiasi probabilità marginale quando ha a diaposizione delle evidenze.
- Imparare i parametri di un programma problog avendo a disposizione delle informazioni solo parziali
- campionare da un programma problog
- risolvere problemi di decisione

Dalla belief network al codice:

Come nel caso di anglican vengono codificati prima i nodi indipendenti della belief network, associandogli una probabilità a priori. Queste corrisponderanno a dei fatti nel linguaggio problog.
Successivamente, si definiscono le probabilità condizionate, che rappresenteranno le regole. Dato che dalla belief network, possiamo notare che T e J sono condizionate da R ed S, lo andremo a tradurre in una regola di questo tipo:

$$0.9 :: traceyWetGrass : -raining, sprinkler.$$

Questa vuol dire che P(T)=0.9 se e solo se sono vere sia R che S, infatti il simbolor ", " corrisponde al connettivo logico "AND", mentre " ; " corrisponde al connettivo logico " OR " e " " corrisponde al "NOT".
All'interno del programma problog, a differenza di anglican, non abbiamo bisogno di esplicitare quale algoritmo di inferenza dobbiamo utilizzare, in quanto sarà il compilatore a decidere quale sia il più adatto al caso. Questo perchè, se ricordiamo la composizione di un algoritmo logico(Paragrafo 2.4), rientra nei compiti del controllo.

Esempio:

```
0.4::raining.
0.3::sprinkler.
0.9::jackWetGrass :- raining.
0.9::traceyWetGrass :- raining, sprinkler.
0.7::traceyWetGrass :- \+raining, sprinkler.
0.7::traceyWetGrass :- raining, \+sprinkler.

evidence(traceyWetGrass,false).
evidence(jackWetGrass,true).

query(raining).
query(sprinkler).
```

TOLGO IL PARAGRAFO SU GEN ED HAKARU?

2.5.3 Gen

Gen e' un linguaggio di programmazione probabilistico che si appoggia sul linguaggio funzionale Julia. Julia e' un linguaggio di programmazione di alto livello e ad alte prestazioni. E' caratterizzato dall' essere un linguaggio di programmazione dinamico, ovvero permette di eseguire operazione che dovrebbero essere fatte a compile-time a run-time. Gen, come gli altri PPL permette di rappresentare modelli probabilistici di tipo stocastico, discreto o continu.Cio' che gli permette di distinguersi dagli altri PPL e' possibilita' che lascia ai programmatori di deffinire algoritmi di inferenza personalizzati.

```
@gen function raining_model()

        #prior of the variables
        sprinkler =@trace(bernoulli(0.3), :sprinkler)

        raining = @trace(bernoulli(0.2), :raining)

        if(raining)
                jackWetGrass  = @trace(bernoulli(0.9), :jackWetGrass )
        else
                jackWetGrass  = @trace(bernoulli(0.2), :jackWetGrass )
        end
        if(raining && sprinkler)
                traceyWetGrass  = @trace(bernoulli(0.9), :traceyWetGrass )
        elseif(!raining && !sprinkler)
                traceyWetGrass  = @trace(bernoulli(0.2), :traceyWetGrass )
        elseif(raining || sprinkler)
                traceyWetGrass  = @trace(bernoulli(0.8), :traceyWetGrass )
        end

        return (sprinkler ,raining )
end

u = choicemap(:traceyWetGrass => true, :jackWetGrass => true)
(tr, _) = generate(raining_model, (), u)
trs, weights, lml = Gen.importance_sampling(raining_model, (), u, 1000)
probBurglar = sum([exp(weights[i]) * trs[i][:raining] for i=1:1000])
```

2.5.4 Hakaru

Hakaru è basato sul linguaggio puramente funzionale Haskell.
Hakaru si differenzia dagli altri linguaggi di programmazione probabilistica per la possibilità di scrivere codice in modo modulare, che rispecchia un pò il ragionamento che viene fatto prima di iniziare a modellare.
Un programma in hakaru prevede una sezione in cui viene presentato il problema dando il modello generale, una in cui vengono associate le prior e conditional distibution attraverso la funzione disintegrate e l'ultima in cui si procede con l'inferenza attraverso la funzione simplify.

```
def bern(p prob):
    x  <~ categorical([p, real2prob(1 - p)])
    return [true, false][x]

raining <~ bern(0.2)
sprinkler <~ bern(0.2)
traceyWetGrass    <~ bern(if raining || sprinkler: 0.9 else:  0.02 )
jackWetGrass <~ bern(if raining: 0.7 else: 0.2)
return ((traceyWetGrass, jackWetGrass) ,raining)
```

Capitolo 3

Risultati e conclusioni

3.1 Introduzione al problema

In questo capitolo si è svolta l'analisi delle prestazioni delle tecniche di inferenza di due dei linguaggi di programmazione probabilistica introdotti, Anglican e Problog.
Per eseguire un'analisi delle prestazione è necessario vedere il loro comportamento su istanze di problemi più complesse di quelle viste nel capitolo precedente. Per questo motivo, si è pensato di provare a risolvere il problema decisionale 3SAT, usando delle sue istanze intrinsecamente difficili da risolvere per la loro struttura, dette appunto, HARD 3SAT.
A prima vista sembrerebbe difficile rappresentare una formula 3SAT in un modello probabilistico, ma come vedremo, è possibile trasformare una formula 3SAT in una Belief Network dal quale poi sarà semplice ricavare il modello probabilistico.
Oltre i concetti trattati precedentemente, per la comprensione di questo capitolo è necessario introdurre SAT e 3SAT.

3.1.1 SAT e 3SAT

Con il termine Sat (satisfiability) si indica il problema decisionale che determina se una formula booleana sia soddisfacibile o meno.
Sia F una formula della logica proposizionale del tipo $F = \{(u1 \vee u2 \wedge \neg u3)\}$, F si dice soddisfacibile se è possibile assegnare alle variabili u1,u2,u3 il valore TRUE o FALSE in

modo tale che l'intera formula F risulti TRUE.
Nel nostro problema si farà uso di formule 3Sat, che non sono altro che una caso particolare di Sat, in cui le formule presentano una forma ben precisa.
Sia F una formula 3Sat essa sarà della forma:

$$F = \{c_1 \wedge c_2 \wedge c_3 \wedge ... \wedge c_n\}$$

con

$$c_i = (u_1 \vee u_2 \vee u_3)$$

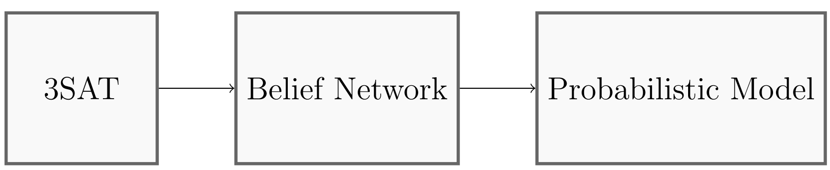
dove c_i è chiamata clausola, mentre u_i è detto letterale ed è associato ad una variabile. Quindi, il letterale u_i è vero se e solo se la variabile u_i a associata è TRUE, invece $\neg u_i$ è vero se e solo se la variabile associata $\neg u_i$ è falsa.

Esempio:

Sia $F = \{(u_1 \vee u_2 \vee u_3) \wedge (\neg u_1 \vee \neg u_2 \vee u_3) \wedge (u_2 \vee \neg u_3 \vee u_4)\}$, un possibile assegnamento di verità per il quale la formula è soddisfacibile potrebbe essere: $u_1 = T$,$u_2 = F$,$u_3 = F$,$u_4 = T$. Per cui diciamo che F è soddisfacibile.

3.2 3SAT Come Belief Network

Per poter analizzare le prestazioni dei nostri linguaggi su un'istanza di 3SAT, abbiamo bisogno di riuscire a descriverla attraverso un modello probabilistico.
Nei capitoli precedenti abbiamo visto come è possibile a partire da una belief network ricavare il relativo modello probabilistico. Questo è un punto che possiamo sfruttare a nostro favore, riducendo il problema iniziale nel creare una belief network che rispecchi la formula. Da qui il modello probabilistico verrà scritto banalmente.

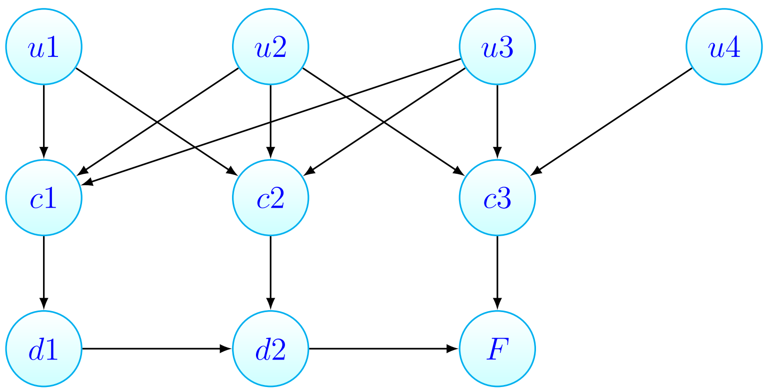


Osservando una formula 3SAT si può vedere che le clausole e la formula sono dipendenti rispettivamente dai letterali e dalle clausole, mentre i letterali sono variabili indipendenti. Ricapitolando, ogni clausola c_i è dipendente dal valore dei letterali da cui è composta, quindi ogni nodo che rappresenta i letterali di c_i dovrà avere un arco uscente verso c_i. La clausola c_i, quindi, sarà TRUE se e solo se il valore di verità di almeno uno dei letterali sarà TRUE. La stessa cosa vale per la formula, dato che il suo valore di verità dipende dal valore di verità di tutte le clausole che la compongono. Per cui ogni nodo c_i dovrà avere un arco uscente verso F che sarà TRUE se e solo se tutte le clausole sono TRUE. Tuttavia per rendere meno complessa la belief network e il modello probabilistico verranno inseriti dei nodi di appoggio (D_i) con il compito di semplificare l'and tra tutte le clausole. Ogni nodo (D_i) possiederà un arco entrante proveniente dal nodo (C_i) ed uno uscente che arriverà nel nodo ($D_i + 1$). L'n-esimo nodo d'appoggio, dove n è uguale alla dimensione della formula, corrisponderà al nodo F. I letterali invece corrispondono alle variabili indipendenti, per cui non avranno alcun arco entrante e saranno TRUE in accordo con la probabilità a priori ad essi associata.[Coo90]

Esempio:

La formula introdotta nell'esempio precedente corrisponderà alla seguente belief network:

$$F = \{(u_1 \vee u_2 \vee u_3) \wedge (\neg u_1 \vee \neg u_2 \vee u_3) \wedge (u_2 \vee \neg u_3 \vee u_4)\}$$



3.2.1 Hard 3SAT

Abbiamo detto che per eseguire un'analisi delle prestazioni abbiamo bisogno di fare inferenza su problemi complessi.
3SAT puó raggiungere grandi dimensioni, per cui sembrerebbe prestarsi bene al nostro scopo, peró vedremo che per essere veramente di difficile risoluzione, non é sufficiente che queste formule siano enormi, ma devono essere di tipo ben preciso.
Uno studio su formule 3SAT di diversa lunghezza sul quale e' stata applicata la procedura Davis-Putnam, ha dimostrato come il rapporto tra numero di clausole e variabili che compongono la formula giochi un ruolo veramente importante per ottenere formule di difficile risoluzione, che chiameremo HARD 3SAT. La procedura Davis-Putnam, assegna ad ogni chiamata, un valore di veritá alle variabili e se non soddisfano la clausola fa backtracking, sino a che la clausola non viene soddisfatta. Il numero chiamate alla procedura corrisponde alla complessita' nel trovare un assegnamento che soddisfi la formula 3SAT. Dallo studio é emerso che formule con poche clausole presentano pochi legami, quindi esistono molti assegnamenti che rendono la formula soddisfacibile ed e' molto probabile che uno di questi assegnamenti venga trovato in poco tempo. Mentre formule con troppe clausole, invece, sono troppo vincolate, per cui e' probabile che all'interno della formula si trovi una clausola e la sua contraddizione; ed appena viene trovata possiamo dire che la formula é insoddisfacibile, si stima che anche questa situazione venga identificata in poco tempo. Diverso é per formule di media lunghezza con un determinato rapporto tra numero di clausole e numero di variabili in quanto il numero di assegnamenti che rendono la formula vera, se esiste é molto basso; per cui é probabile sia necessario scorrere quasi tutti i possibili assegnamenti prima di trovarne uno soddisfacibile. Questo si puó osservare nel grafico della Figura 3.1. Formule 3SAT con un rapporto clausole-variabile di circa 4.3 presenta il maggior numero di chiamate a DP per poter risolvere la formula 3SAT. In quel punto, inoltre, circa il 50% delle é soddisfacibile, mente con un rapporto inferiore é piú probabile siano soddisfacibili e viceversa. [SML96]

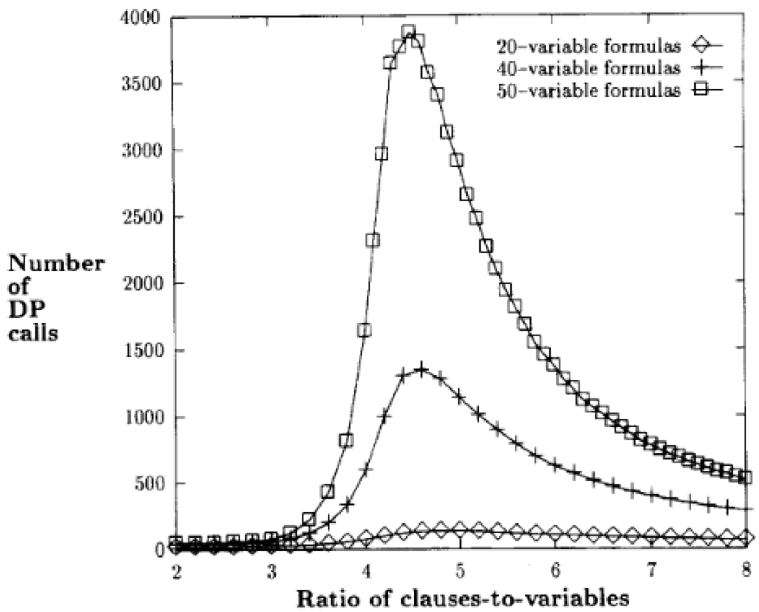


Figura 3.1: Bayes theorem.
Fonte:https://dpzbhybb2pdcj.cloudfront.net/pfeffer/Figures/01fig07alt.jpg

3.2.2 Implementazione

Una volta spiegati i punti fondamentali è necessario comporli per poter far analizzare le performance dei linguaggi su questo tipo di problemi d'inferenza. Per la creazione delle formule 3SAT, è stata creata una funzione che dato il numero di variabili e il numero di clausole della formula, crea una formula della lunghezza desiderata, questo ci permette di analizzare i risultati su un ampio range di formule, che va dalle piu semplici, a quelle di tipo HARD 3SAT, fino a quelle con una dimensione molto elevata.

La funzione presi gli input definisce l'insieme delle variabili e in modo random definisce quali variabili appartengono ad una clausola e ancora in modo casuale se sono negate oppure no.

Successivamente, la formula 3SAT generata viene data in pasto ad un funzione che genera la relativa belief network come spiegato nel paragrafo precedente. (3.2) A questo punto, attraverso una visita del grafo(belief network) è possibile ricavarsi il modello probabilistico relativo alla formula 3SAT iniziale. Questo verrà poi eseguito come programma standalone e verranno raccolti i risultati riguardanti la precisione del risultato ed il tempo necessario per raggiungerlo.

Esempio:

In seguito, viene mostrato come viene tradotta una formula 3SAT nel relativo modello probabilistico in anglican e problog:

Problog:

```
0.5::u4.
0.5::u3.
0.5::u2.
0.5::u1.
1.0::c0:-u1;u2;u3.
1.0::c1:-\+u1;\+u2;u3.
1.0::c2:-u2;\+u3;u4.
1.0::d0:-c0.
1.0::d1:-d0,c1.
1.0::y:-d1,c2.
query(y).
```

Anglican :

```
(defquery anglsat []
     (let [
        u4 (sample(flip 0.5))
        u3 (sample(flip 0.5))
        u2 (sample(flip 0.5))
        u1 (sample(flip 0.5))
        c0 (getPrior u1 u2 u3 true true true)
        c1 (getPrior u1 u2 u3 false false true)
        c2 (getPrior u2 u3 u4 true false true)
        d0(cond(= c0 true)
           (sample(flip 1.0))
           (= c0 false)
           (sample(flip 0.0)))
        d1 (getY d0 c1 )
        y (getY d1 c2 )
]y))
```

In cui le funzioni "getY()" e "getPrior()" sono definite in questo modo:

```
(defm getPrior[u1 u2 u3 val1 val2 val3]
    (let[
        c (cond(or
                  (= u1 val1)
                  (= u2 val2)
                  (= u3 val3))
                  (sample(flip 1.0))
              (and(not= u1 val1)
                  (not= u2 val2)
                  (not= u3 val3))
              (sample(flip 0.0)))]
           c))
```

```
(defm getY[u1 u2]
    (let[
        c (cond(and
                  (= u1 true)
                  (= u2 true))
                  (sample(flip 1.0))
                  (or(= u1 false)
                     (= u2 false))
                  (sample(flip 0.0)))]
        c))
```

3.3 Risultati

Bibliografia

[Coo90] Gregory F Cooper. «The computational complexity of probabilistic inference using Bayesian belief networks». In: *Artificial intelligence* 42.2-3 (1990), pp. 393–405.

[SML96] Bart Selman, David G Mitchell e Hector J Levesque. «Generating hard satisfiability problems». In: *Artificial intelligence* 81.1-2 (1996), pp. 17–29.

[Las12] Daniel Lassiter. «Probabilistic reasoning and statistical inference:An introduction (for linguists and philosophers)». In: *NASSLLI 2012 Bootcamp* (2012). DOI: `https://web.stanford.edu/~danlass/NASSLLI-coursenotes-combined.pdf`.