# Structure Learning in Bayesian Network

Giacomo Vitangeli

## 1 Introduzione

Nella seguente relazione verrà illustrato il funzionamento di un software per l'apprendimento della struttura di reti Bayesiane; tale struttura verrà appresa da un dataset, generato a partire dalle probabilità condizionate dei nodi della rete.
La rete Bayesiana è caratterizzata da un insieme di variabili che hanno una certa probabilità di verificarsi, il condizionamento probabilistico tra le variabili è rappresentato dagli archi che collegano i nodi del grafo.

## 2 Cenni Teorici

Alla base di questo progetto ci sono dei concetti teorici che verranno descritti rapidamente prima di passare agli aspetti prettamente pratici e sperimentali.
L'ordine di presentazione di questi cenni teorici corrisponde all'ordine di esecuzione del software, ogni parte è condizionata dalla precedente.

### 2.1 Ordinamento Topologico

L'ordinamento topologico è indispensabile per il corretto funzionamento del programma; tale funzionalità si occupa di ordinare i nodi secondo un cammino che va dalla radice alle foglie, con un metodo detto visita in profondità o **DFS** (*depth first search*).
Vengono salvati i tempi di fine scoperta di ogni nodo, in modo che nessun nodo possa venir analizzato prima dei propri genitori; inoltre l'ordinamento può non essere unico.

### 2.2 Dataset

Il dataset consiste in una matrice che ha per righe un insieme di casistiche e per colonne i nodi della rete.
Vengono generati in modo pseudocasuale dei valori compresi tra 0 e 1, quindi ogni valore viene poi confrontato con i valori del campo probabilistico `.cpt` del nodo in esame data l'eventuale configurazione dei genitori, perciò è indispensabile un precedente ordinamento topologico dei nodi.

### 2.3 Structure Learning

L'apprendimento della struttura consiste nell'analizzare un dataset, da questo ottenere la struttura (grafo rappresentato da una matrice di adiacenza) della rete bayesiana che lo ha generato.
Prima di tutto vanno fatte delle **assunzioni** al fine del corretto funzionamento dell'algoritmo di apprendimento:

- Le variabili aleatorie sono discrete e interamente osservabili: supponiamo che non ci siano dati mancanti.

- I record del dataset sono indipendenti tra loro.
- I nodi sono ordinati secondo l'ordine topologico.
- Viene definito inizialmente un numero massimo di genitori che ogni nodo può avere.

Esistono due **metodi principali** per l'apprendimento della struttura:

- **Search-score**: vengono usate delle funzioni score che permettono di confrontare l'adeguatezza tra le possibili strutture di una rete.
- **Constraint-based**: in questo caso si utilizzano delle misure per indagare l'esistenza di eventuali dipendenze o indipendenze condizionate tra le variabili aleatorie.

#### 2.3.1 K2

L'algoritmo K2 è un algoritmo greedy per l'apprendimento della struttura di reti bayesiane di tipo score-based, la sua efficienza dipende direttamente dall'ordine con cui gli vengono fornite le variabili in input; se non viene rispettato l'ordinamento topologico, la struttura appresa non sarà corretta.
L'obiettivo di questo metodo è selezionare la rete a cui è associato lo score più alto, in particolare la ricerca delle dipendenze tra le variabili aleatorie avviene mediante l'uso di una struttura ipotetica di dipendenze e tramite l'ausilio di un calcolatore viene individuata la probabilità della distribuzione a posteriori.
Può accadere che la struttura ottenuta non sia perfettamente identica a quella di partenza, ma che differisca per un numero di archi in eccesso e/o in difetto.

# 3 Codice

Di seguito verrà illustrata la soluzione implementativa adottata, i test ed i relativi risultati.

## 3.1 Implementazione

Il linguaggio di programmazione scelto per questo progetto è stato il **Python**.
Prima di tutto viene istanziata la rete bayesiana tramite il file *BayesianNet.py* che imposta la rete, in particolare utilizza un altro file (*Earthquake.py*) per ottenere i dati specifici della rete scelta. Dopo aver impostato la rete viene generato il dataset nel file *Dataset.py*, dentro al quale viene ordinato l'array di nodi chiamando la funzione `dfs`.

Infine si ha l'esecuzione del metodo `k2()` per l'apprendimento della struttura; tale metodo prende in input: un dataset, un vettore di nodi (ordinati topologicamente) e il numero massimo di genitori che può avere ogni nodo.
Dopo aver preso i dati richiesti in input viene resettato il campo dei genitori per ogni nodo, in modo da poter apprendere i genitori in base allo studio del dataset.
Un passaggio fondamentale di quest'algoritmo è dato dal confronto dello score mediante il metodo `maximize_score()`, a tale metodo vengono passati, oltre al dataset ed il nodo in analisi, anche un nodo dall'insieme dei predecessori del nodo considerato ad esclusione dei genitori già affermati con lo score massimo assegnati precedentemente; viene così trovato il predecessore che massimizza la funzione score, e che quindi è il candidato migliore ad essere padre del nodo considerato. Questa funzione viene iterata finchè non

si raggiunge il numero massimo di padri che può avere un nodo (`upper_bound`).
Il metodo centrale è `score()`, che implementa la seguente funzione:

$$f(i, \pi_i) = \prod_{j=1}^{q} \frac{(r_i - 1)!}{(N_{\text{ij}} + r_i - 1)!} \prod_{k=1}^{r} a_{\text{ijk}}!$$

All'interno del metodo `score()` viene chiamata la funzione `count_case()`, la quale si occupa di contare il numero di occorrenze delle diverse configurazioni (proposte dal `cartesian_product()` sotto forma di combinazioni di 0 e 1 sul dataset), ritornando così $a_{ijk}$.
Il metodo `k2()` infine ritornerà come output la matrice di adiacenza della rete appresa.

## 3.2 Test

Per riprodurre i test sarà sufficiente eseguire il file *Main.py*, nel quale viene istanziata la rete e vengono generati 1000 dataset da 150 casi ciascuno. Per ogni dataset generato viene invocato il metodo *k2* per l'apprendimento della struttura.
Le strutture apprese vengono poi sommate tra loro in modo da convergere in un'unica matrice di adiacenza che andrà confrontata con quella della rete bayesiana di partenza.
I dataset sono stati generati di dimensioni ridotte a causa di un **OverflowError** nel metodo *score*, poiché viene effettuato un prodotto con numeri molto grandi, la cui dimensione dipende direttamente dal numero di casi del dataset.
Signatures dei metodi chiamati per eseguire il test di funzionamento:

- `BayesianNet()`: ritorna un istanza di rete Bayesiana.
- `Dataset(bn, num_cases)`: prende in input una rete bayesiana e un numero di test da effettuare, restituisce in output un oggetto di tipo Dataset da cui è possibile recuperare la matrice del dataset accedendo al campo .dataset dell'oggetto.
- `k2(data.dataset, data.ordered_array, 2)`: prende in input la matrice del dataset, una lista di nodi ordinati topologicamente, e un upper bound che identifica il numero massimo di padri che può avere ogni nodo. Rende in output la matrice di adiacenza del dag appreso.

Per testare il codice è stata utilizzata la rete bayesiana in Figura 1.

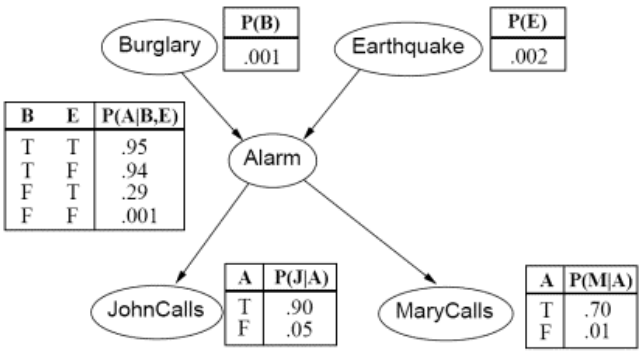


Figura 1: Alarm Model.

### 3.3 Risultati

Sono riportate di seguito due matrici, la prima è la matrice di adiacenza della rete di partenza (Alarm), mentre la seconda è quella ottenuta eseguendo il test descritto al paragrafo 3.2.

| | $B$ | $E$ | $A$ | $J$ | $M$ |
|---|---|---|---|---|---|
| $B$ | 0 | 0 | 1 | 0 | 0 |
| $E$ | 0 | 0 | 1 | 0 | 0 |
| $A$ | 0 | 0 | 0 | 1 | 1 |
| $J$ | 0 | 0 | 0 | 0 | 0 |
| $M$ | 0 | 0 | 0 | 0 | 0 |

| | $B$ | $E$ | $A$ | $J$ | $M$ |
|---|---|---|---|---|---|
| $B$ | 0 | 0 | 769 | 427 | 649 |
| $E$ | 76 | 0 | 222 | 72 | 38 |
| $A$ | 0 | 0 | 0 | 919 | 801 |
| $J$ | 0 | 0 | 0 | 0 | 0 |
| $M$ | 0 | 0 | 0 | 208 | 0 |

Le matrice ottenuta, dopo aver eseguito l'apprendimento della struttura su diversi dataset, mostra chiaramente la **rilevazione di tre archi** della struttura originaria, in particolare: **A**→**J** (92%), **A**→**M** (80%), **B**→**A** (77%); mentre l'arco **E**→**A** (22%) è un risultato più incerto poiché viene rilevato per meno del 50% dei casi.

Dalla seconda matrice è possibile notare che vengono rilevati degli **archi in eccesso**, i più rilevanti sono **B**→**M** (65%), **B**→**J** (43%), **M**→**J** (21%); anche altri tre archi che però sono inferiori al 10%.
Possiamo notare che gli archi in eccesso più rilevanti sono stati quelli che collegano *Burglary* a *MaryCalls* e *JohnCalls*, considerando il modello di partenza sappiamo però che tali archi non sono possibili per la definizione di **d-separation**, infatti la probabilità condizionata di ogni nodo di una rete bayesiana dipende solo dai suoi nodi padri. La causa del rilevamento dei padri in eccesso sulle due variabili di *Call* probabilmente è da attribuirsi al fatto che il numero massimo di padri della rete è 2, perciò oltre ad identificare come padre Alarm, l'algoritmo ha provato a trovare ulteriori padri per le due chiamate, in modo da formare una coppia di padri.

## 4 Conclusione

In conclusione i risultati ottenuti corrispondono al livello di precisione di apprendimento atteso, poiché di tutti gli algoritmi di apprendimento di reti bayesiane k2 non è tra i più precisi.
In termini di **probabilità condizionata** nella struttura appresa sono stati evidenziati un maggior numero di condizionamenti tra le variabili di quanti effettivamente ce ne fossero nella struttura reale, facendo uso di altre euristiche sarebbe stato possibile ottenere risultati più accurati.

## 5 Riferimenti


- Dataset
  *Earthquake*.
- David Heckerman, 1997
  *Bayesian Networks for Data Mining*.
- Stuart J. Russel, Peter Norving
  *Artificial Intelligence: A Modern Approach.*
- Carolina Ruiz, Worcester Polytechnic Institute
  *Illustration of the K2 Algorithm for Learning Bayes Net Structures*.