Apprendimento della struttura di reti Bayesiane

Lorenzo Serchi Masini

Settembre 2021

Procedura generale

L'implementazione dell'apprendimento della struttura di reti Bayesiane di seguito segue l'approccio descritto in (Heckerman 1997).
Date $X_1, ..., X_N$ variabili aleatorie categoriche e un dataset D di n campioni i.i.d. completo (i.e. ciascuna N-upla non ha valori mancanti), per apprendere la struttura S^h di una rete Bayesiana che sia rappresentativa di D, avendo scelto come distribuzione a priori dei parametri $\Theta_{X_i|Pa(X_i)=j}$ delle $Dirichlet(\alpha_1, ..., \alpha_{ijk}, ..., \alpha_{d_i})$, effettuiamo una ricerca nello spazio dei possibili DAG massimizzando la seguente metrica:

$$Score(S^h) = \prod_{i=i}^{N} \prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij} + n_{ij})} \prod_{k=1}^{d_i} \frac{\Gamma(\alpha_{ijk} + n_{ijk})}{\Gamma(\alpha_{ijk})} \quad , \sum_{k=1}^{d_i} \alpha_{ijk} = \alpha_{ij} \ , \sum_{k=1}^{d_i} n_{ijk} = n_{ij} \tag{1}$$

dove d_i è il numero di configurazioni possibili del nodo X_i e q_i è il numero di configurazioni possibili dei padri $Pa(X_i)$, i conteggi n_{ijk} sono le statistiche sufficienti derivate dal dataset, contando le occorrenze in D delle configurazioni $x_i = k \wedge Pa(x_i) = j$. Questa metrica è localmente decomponibile, ossia una modifica di una parte della struttura su cui è calcolata si ripercuote su pochi fattori della stessa (il che rende più efficiente l'esplorazione dello spazio dei DAG per aggiunta/modifica/inversione di archi). In più si può mostrare (Heckerman 1995) che due DAG che codificano le stesse assunzioni di indipendenze condizionate sono equivalenti sotto questa metrica (hanno lo stesso punteggio: si parla di *likelihood equivalence*)*se* si scelgono in maniera appropriata gli pseudo-conteggi α_{ijk}; in particolare, scegliendo ad esempio:

$$\alpha_{ijk} = \alpha * P(X_i = k, Pa(X_i) = j|S) = \frac{\alpha}{d_i q_i}$$

si ha garanzia che valga questa proprietà. Il parametro α è detto equivalent sample size, e si sceglie tipicamente pari a $\alpha = 1$ (come lo scegliamo noi). Questa metrica si chiama BDeu (cioè **B**ayesian-**D**irichlet likelihood-**e**quivalent **u**niform metric) ed è un caso particolare della BDe (per cui vale l'equivalenza delle likelihood) in cui i prior sono scelti con distribuzione uniforme.

La ricerca nello spazio dei possibili DAG è fatta partendo da un DAG generato casualmente e spostandosi con (a) aggiunta, (b) rimozione o (c) inversione di uno degli archi; scelto il DAG con punteggio migliore, si ripete la procedura fino a che non c'è più un miglioramento (Hill Climbing). In realtà, essendo lo spazio dei possibili DAG molto grande, l'implementazione usa l'algoritmo Beam Search partendo da b (*beam size*) DAG generati casualmente, elencando tutti i DAG ad essi vicini e scegliendo i b migliori fino a che non si ha più un miglioramento.

Per confrontare la struttura S^h del modello risultante con la struttura S generante il dataset D, si applica la seguente metrica di valutazione:

$$accuracy(S^h, S) = \frac{TP + TN}{TP + FP + FN + TN} = \frac{\#(archi\ uguali)}{\#(archi\ in\ un\ DAG\ con\ N\ vertici)}$$

alcune delle strutture S sono prese da reti bayesiane ('ASIA', 'CANCER' con 8 e 5 nodi rispettivamente) disponibili a (Bayesian Network Repository).

Documentazione essenziale

Sono fornite 3 reti Bayesiane già pronte per generare un dataset D. Quindi non è necessario leggere come costruire una rete Bayesiana se (a) si intende usare una di quelle fornite o (b) si ha già a disposizione un dataset (in quest ultimo caso è comunque necessario leggere la documentazione relativa ai nodi, in quanto è necessaria per far interpretare correttamente il dataset all'algoritmo di apprendimento).

Costruire una rete Bayesiana & generare il Dataset

Una rete Bayesiana comprende un insieme di nodi/variabili, ciascuno con una CPT (Conditional Probability Table) associata:

- **class Node**
  - **def __init__(self, name, configurations, thetas)**
    per ogni nodo X va specificato il nome e:
    * i valori che può assumere ($dom(X) = \{x_1, x_2, ..., x_d\}$)
    * i parametri della CPT: $P(X \mid Pa(X))$

    per farlo, ad esempio per la variabile booleana X con CPT a lato (dove $Pa(X) = \{Y, Z\}$):

    * configurations = ["1","0"]
    * thetas = ["00":[0.3,0.7],"01":[0.4,0.6]],"10":[0.2,0.8],"00":[0.1,0.9]

    dove le configurazioni dei padri (es. "00","01",...) seguono l'ordine di inserimento degli archi (es. se si aggiunge prima (Y,X) di (Z,X) lo "0" di "01" si riferisce a Y).

X	Y	Z	$P(X\|Y,Z)$
0	0	0	0.7
0	0	1	0.6
0	1	0	0.8
0	1	1	0.9
1	0	0	0.3
1	0	1	0.4
1	1	0	0.2
1	1	1	0.1

  - **def generate_sample(parents_config)**
    è il metodo che genera un campione per il nodo in base alla sua CPT e alla configurazione attuale dei suoi padri.

questi nodi, inseriti in una lista, sono usati per istanziare la classe "DAG". Una volta istanziata, si aggiungono gli archi.(**N.B.** Se si ha intenzione di usare questa rete per generare campioni, l'ordine in cui si aggiungono gli archi è importante: devono essere aggiunti in modo coerente con le CPT dei singoli nodi - vedi *thetas*)

- **class DAG**
  una rete Bayesiana è rappresentata univocamente dalla sua matrice di adiacenza, dal suo insieme di nodi (avendone stabilito un ordine) e dalle CPT associate ad essi. In questa implementazione, l'ordine dei nodi è quello nella lista passata al momento dell'instanziazione del DAG (in base a questo ordine, l'associazione tra nodi e matrice di adiacenza è univocamente determinata).
  - **def __init__(self,nodes,randomInit=False)**
    avendo passato la lista dei nodi, (di default) l'istanziazione consiste semplicemente nel creare una matrice di adiacenza vuota (quindi nessun arco). Eventualmente si può optare per aggiungere casualmente degli archi (mantenendo l'aciclicità).

  una volta istanziato, si possono aggiungere gli archi (specificati tenendo conto sia dell'indice del nodo - stabilito dal suo ordine nella lista *nodes* - e aggiunti con un ordine compatibile all'ordine scelto dei padri durante la specifica della CPT di ciascun nodo) con il metodo:

  - **def add_edge(self,edge)**
    edge è una tupla (u,v) con u/v indice del rispettivo nodo nella lista *nodes*. Restituisce *False* se non è possibile aggiungere tale arco.

  fatto questo, la rete è pronta per essere campionata invocando il metodo:

  - **def generate_sample(self)**
    genera un campione dalla rete Bayesiana

  così facendo si può generare il dataset.

per mostrare la rete Bayesiana (solo la struttura) costruita si possono invocare i metodi print() o draw(), rispettivamente per mostrare la sua matrice di adiacenza o una sua rappresentazione grafica.

L'algoritmo di apprendimento

Per trovare la struttura che con maggiore probabilità ha generato il dataset D ci basiamo sul principio max. a posteriori, ossia *si assume* che la struttura che massimizza $P(S^h \mid D)$ sia l'unica ad avere probabilità non nulla. Con questa assunzione, possiamo trovare una tale struttura massimizzando la verosimiglianza del dataset D data la struttura S^h (ossia la metrica introdotta inizialmente). In realtà massimizziamo la log-verosimiglianza per semplificare i conti:

$$\log(Score(S^h)) = \sum_{i=i}^{N} \sum_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij} + n_{ij})} \sum_{k=1}^{d_i} \frac{\Gamma(\alpha_{ijk} + n_{ijk})}{\Gamma(\alpha_{ijk})} \tag{2}$$

L'algoritmo (già anticipato) è implementato come segue:

- **def beam_search(b, nodes, dataset)**

si passa il "beam size" b, la lista di variabili *nodes* (che può essere prodotta senza specificare le CPT: è sufficiente specificare i nomi dei nodi) e il *dataset* di dimensioni nxN con n il numero di esempi e N il numero di variabili/nodi (chiaramente è necessario che *nodes* e *dataset* siano compatibili nel senso inteso dal dataset: la colonna i-esima deve contenere i campioni relativi al nodo in posizione i-esima di *nodes*).
Vengono generati casualmente b DAGs. Per ciascuno di essi si calcola:

- *count*: cioè si contano (dal *dataset*) gli $n_{ijk} = \#(x_i = k \wedge Pa(x_i) = j)$, salvandoli nella tabella hash *count* (chiave: 'i,j,k', valore: n_{ijk});
- *score*: cioè si calcola il punteggio dell'intero DAG secondo la metrica (2) usando i prior α_{ijk} uniformi (per il motivo già detto). Il punteggio complessivo è ottenuto come somma del punteggio dei singoli nodi:
  - *def node_score(graph, node, count)*

  il motivo per cui il calcolo del punteggio di un singolo nodo è incapsulato in una funzione a parte è perché (come detto) la metrica (2) è localmente decomponibile, pertanto quando si (a) aggiunge, (b) rimuove o (c) inverte un arco non è necessario ricalcolarla tutta, bensì solo cambiare il punteggio di un solo (due nel caso c) nodo (nodi).

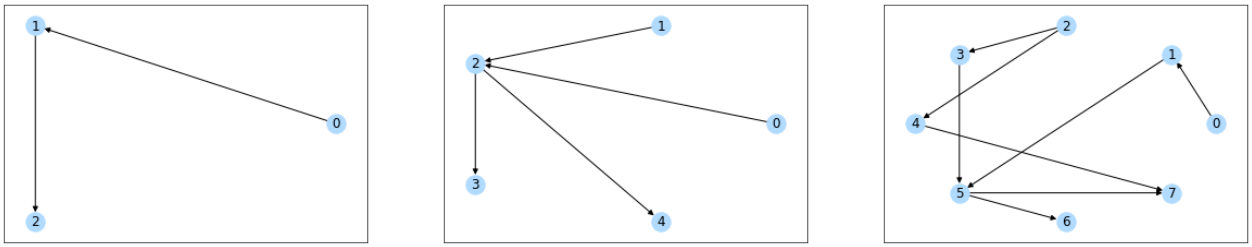
e si generano tutti i DAG ad essi vicini, ossia ottenibili mediante (a) aggiunta, (b) rimozione o (c) inversione di archi (ovviamente laddove queste operazioni non portano a creazione di cicli) e se ne calcolano i rispettivi conteggi e punteggi; conteggi e punteggi che, come detto più volte, sono ottenuti in maniera relativamente semplice a partire dai b DAG iniziali. Infine si selezionano i b DAG migliori in termini di punteggio e si controlla se quello con punteggio massimo tra questi è migliore del corrispondente nella generazione precedente. Se sì, si ripete; altrimenti si termina fornendolo in uscita.

Risultati sperimentali

Come misura di somiglianza tra strutture di reti Bayesiane con N variabili, si usa il rapporto:

$$accuracy(S^h, S) = \frac{\#(archi\ uguali)}{\#(archi\ in\ un\ DAG\ con\ N\ vertici)}$$

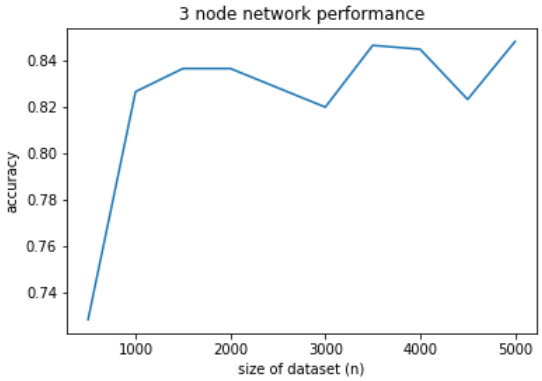
le reti Bayesiane usate hanno rispettivamente 2/5/8 nodi e struttura S come in Figura 1.



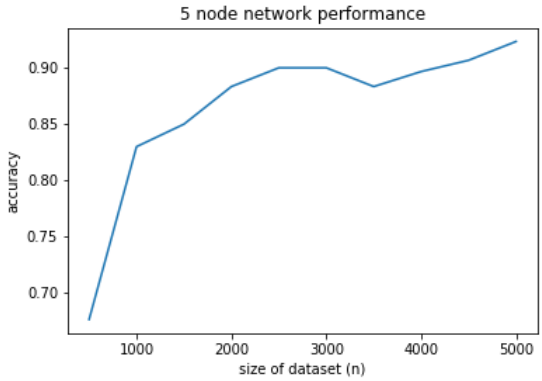
(a) 3 node network (b) 5 node network (c) 8 node network

Figure 1: networks used for testing

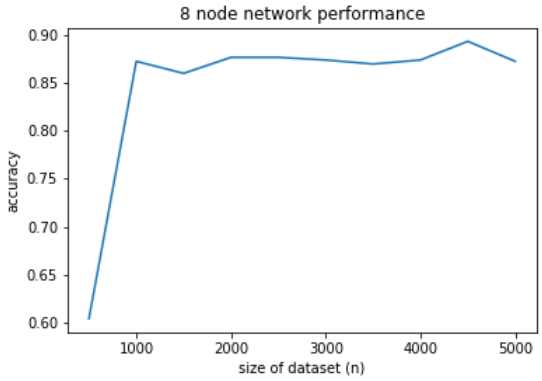
I parametri sono ritrovabili nel codice. I risultati che seguono sono ottenuti con b=10 e rappresentano la media dell'accuratezza di 20 test per ogni dimensione crescente del dataset (10 dimensioni diverse fino a n=5000):



(a) 3 node network



(b) 5 node network



(c) 8 node network

Figure 2: accuracy on 3/5/8 node networks