Apprendimento di reti bayesiane con distribuzioni note a priori

Edoardo Grassi

L'obiettivo dell'elaborato è quello di cercare di implementare e analizzare la tecnica descritta in un articolo[1] indicato dal professore per l'apprendimento di reti bayesiane. Per fare ciò verrà utilizzata una distribuzione nota a priori, scelta tra quelle disponibili nella libreria `bnlearn`, di media grandezza. In particolare verrà utilizzata la distribuzione `INSURANCE`.

1 Implementazione

L'implementazione è stata realizzata in *Python* cercando di utilizzare il meno possibile librerie esterne. In particolare le librerie utilizzate sono:

- `matplotlib` per la visualizzazione dei grafici delle analisi effettuate;
- `graphviz` per la visualizzazione delle reti bayesiane risultanti.

Nel modulo `structure_learning` è presente il sotto-modulo `networks` usato per gestire le reti bayesiane. In questo sotto-modulo sono presenti le classi:

1. `BayesianNode` che rappresenta un nodo della rete bayesiana e quindi una variabile aleatoria, contenendo le informazioni riguardanti le dipendenze condizionali (e quindi sia i padri che i figli di quel nodo), e la profondità del nodo nella rete;
2. `BayesianNetwork` che rappresenta una rete bayesiana insieme ai suoi nodi; da notare sono i metodi `get_colliders` e `count_diff_colliders`: il primo restituisce i nodi collider della rete bayesiana; il secondo, data un'altra rete bayesiana sotto forma di `BayesianNetwork`, restituisce il numero di collider in eccesso e in difetto della rete data rispetto a quella su cui è stato chiamato il metodo; questo permette di valutare quanto due reti bayesiane siano simili tra loro, confrontando le loro indipendenze condizionali[2]; inoltre il metodo `draw_graph` permette di salvare un'immagine della rete bayesiana utilizzando la libreria `graphviz`;
3. `CPTBayesianNode` estensione della classe `BayesianNode` che aggiunge la tabella delle probabilità condizionali per quel nodo, quindi per ogni possibile combinazione di valori dei padri e della variabile stessa assegna un valore di probabilità;
4. `CPTBayesianNetwork` estensione della classe `BayesianNetwork` che permette di gestire solo nodi di tipo `CPTBayesianNode` e aggiunge il metodo `generate_random_sample` che genera un singolo campione casuale della distribuzione rappresentata dalla rete bayesiana utilizzando la libreria `random`, in maniera tale che ogni campione sia indipendente e identicamente distribuito rispetto agli altri.

Inoltre nello stesso sotto-modulo viene definita una funzione per generare una `CPTBayesianNetwork` a partire da un file di testo in formato `.net`.
Proseguendo ad analizzare gli altri elementi del modulo `structure_learning` si hanno:

5. `Samples` classe contenitore di un insieme di campioni generati da una rete bayesiana, insieme alla rete che li ha generati;
6. `Heuristic` classe per rappresentare la funzione euristica utilizzata per l'apprendimento della struttura, che d'ora in poi prenderà il nome di *euristica fattoriale* indicata nell'articolo come $g(i, \pi_i)$[3]; l'implementazione proposta prevede il precalcolo di tutti i fattoriali fino a $m + r - 1$ dove m è il numero di campioni generati e r la massima dimensione tra i domini delle variabili aleatorie, e questi valori vengono salvati nelle istanze di tale classe; per convenienza tale classe viene anche usata per contenere un'istanza della classe `Samples` usata dall'algoritmo di apprendimento;

[1] Gregory F. Cooper e Edward H. Herskovits. "A Bayesian Method for the Induction of Probabilistic Networks from Data". In: *Machine Learning* 9.4 (1992), pp. 309–347. URL: `https://link.springer.com/content/pdf/10.1007/BF00994110.pdf`.

[2] Judea Pearl e Thomas Verma. "Equivalence and Synthesis of Causal Models". In: (1990), pp. 220–227. URL: `https://arxiv.org/pdf/1304.1108.pdf`.

[3] Gregory F. Cooper e Edward H. Herskovits. "A Bayesian Method for the Induction of Probabilistic Networks from Data". In: *Machine Learning* (1992), p. 321.

7. `LogaritmicHeuristic` estensione della classe `Heuristic` che nell'articolo viene proposta come alternativa per ridurre i tempi di calcolo della funzione euristica: infatti calcola $\log(g(i, \pi_i))$ quindi utilizza somme e sottrazioni al posto di moltiplicazioni e divisioni[4]; rimane comunque il precalcolo dei **logaritmi** dei fattoriali;

8. la funzione `k2` che implementa l'algoritmo *K2*[3] per l'apprendimento della struttura della rete bayesiana, prendendo come parametri un'ordinamento delle variabili aleatorie, l'euristica da utilizzare contenente anche i campioni generati dalla rete bayesiana, e il numero massimo di genitori per ogni nodo; restituisce sempre un'istanza della classe `BayesianNetwork` che rappresenta la rete bayesiana contenente le sole dipendenze condizionali.

2 Test svolti

2.1 Primo test

Questo analisi si focalizza sui tempi di esecuzione dell'algoritmo *K2* al variare del numero di campioni (generati utilizzando la distribuzione `INSURANCE`), del numero massimo di dipendenze condizionali per ogni nodo e dell'euristica utilizzata. Ciò viene fatto tramite lo script `main_comparison.py` che genera 10 insiemi di $20 \cdot x$ campioni con $1 \leq x \leq 10$ (quindi $\{20, 40, 60...200\}$) e esegue per tre *round* l'algoritmo *K2* su ognuno di essi, per ogni limite di dipendenze nell'insieme $\{1, 2, 3, 5\}$, individuando il minimo tra tutti i tempi di esecuzione registrati nei vari round. Infine l'ordinamento delle variabili aleatorie è costante e tale per cui ogni nodo è figlio solamente di nodi precedenti rispetto all'ordinamento, quindi garantisce la condizione ottimale per l'algoritmo *K2*.

2.1.1 Risultati

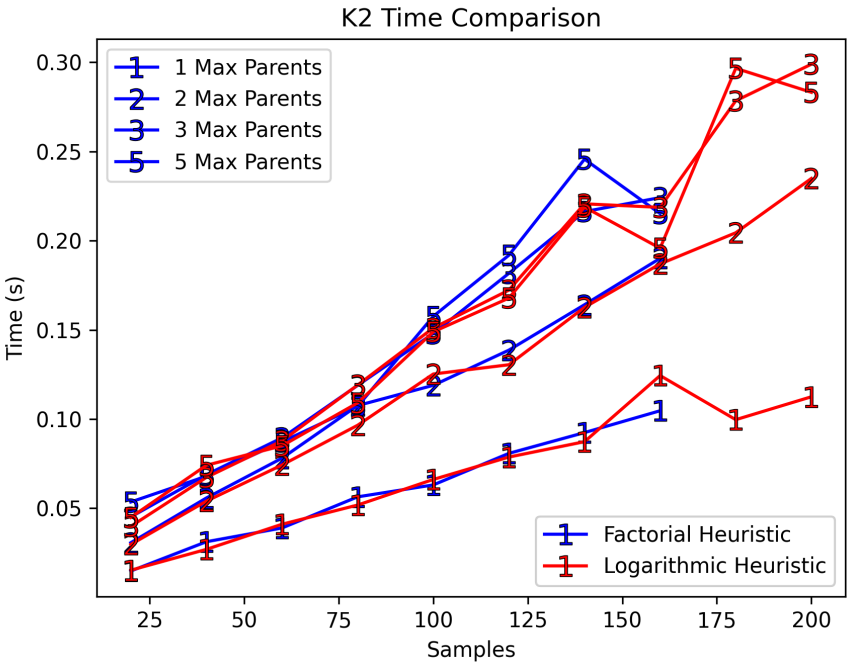


Figura 1: grafico che rappresenta i tempi di esecuzione di *K2*

I risultati sono rappresentati in figura 1 e in linea con quanto riportato nell'articolo[3]: i tempi di *K2* risultano lineari rispetto al numero di campioni, e si può notare che man mano che si aumenta il numero massimo di dipendenze condizionali i tempi aumentano fino a arrivare a un *plateau*: infatti con massimo uno o due dipendenze condizionali per nodo notiamo differenze nei tempi, mentre per le curve con massimo tre e cinque dipendenze non notiamo particolari differenze; questo è dovuto al fatto che la funzione euristica blocca in anticipo l'estensione del numero dei padri osservando valori della funzione euristica peggiori. Per quanto riguarda il tipo

[4]Gregory F. Cooper e Edward H. Herskovits. "A Bayesian Method for the Induction of Probabilistic Networks from Data". In: *Machine Learning* (1992), p. 322.

di funzione euristica applicata non si notano alcune differenze sui tempi, ma è possibile osservare un'altro fenomeno: oltre 160 campioni l'algorimo con l'euristica fattoriale fallisce nell'esecuzione per errori di *overflow*, mentre applicando l'euristica logaritmica non si hanno problemi di questo tipo; questo si spiega dal fatto che la funzione euristica fattoriale genera punteggi molto bassi in modulo che devono essere moltiplicati per fattoriali di numeri molto elevati.

2.2 Secondo Test

Sfruttando lo stesso script *Python* del primo test, si analizzano le capacità dell'algoritmo *K2* nel ricondurre i campioni della distribuzione alla distribuzione originale stessa. Per fare ciò si utilizza la misura di differenza tra i collider delle reti bayesiane definita al punto 1.2, e si analizza la variazione di tale misura al variare del numero di campioni, del numero massimo di dipendenze condizionali per ogni nodo e dell'euristica utilizzata.

2.2.1 Risultati

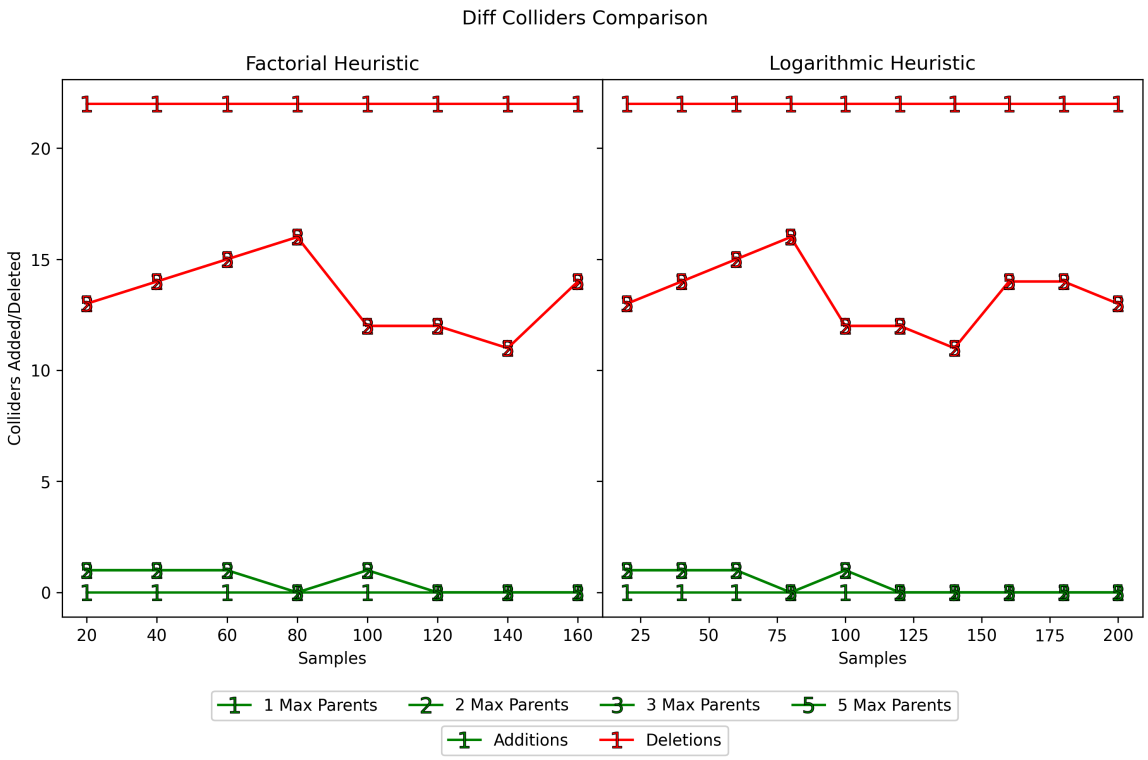
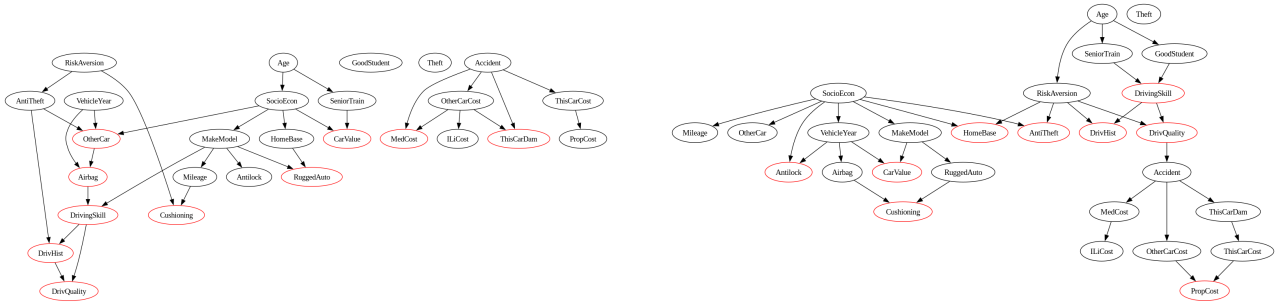


Figura 2: grafico che rappresenta il numero di collider in eccesso e difetto rispetto alla rete originale all'aumentare del numero dei campioni

I risultati rappresentati in figura 2 dimostrano che non c'è alcuna differenza relativamente all'analisi tra l'utilizzo dell'euristica fattoriale e logaritmica, trascurando il fatto che la variante logaritmica non fallisce dopo 160 campioni. In entrambi casi possiamo osservare che aumentando il numero di campioni le discrepanze tra le reti diminuiscono. Analizzando invece il comportamento al variare del numero massimo di dipendenze si nota che con tale limite impostato a 1 si ha la possibilità di ottenere reti con meno collider aggiunti, a discapito di quelle rimosse che risultano maggiori rispetto agli altri limiti; con limiti più alti (2, 3 e 5) si hanno comportamenti simili che tendono a ridurre il più possibile il numero di collider in eccesso/difetto. Notiamo comunque che l'algoritmo fatica molto più ad aggiungere eventuali collider piuttosto che rimuoverne. In figura 3 si possono osservare le reti generate.

2.3 Terzo test

Sulla base dei risultati ottenuti nei test precedenti, si esegue un'ultima analisi per verificare quanti campioni siano necessari per ridurre al minimo la discrepanza tra le reti generate e quella di partenza, misurando anche il tempo necessario. Per fare ciò si utilizza lo script `min_diff.py` che genera insiemi di campioni di cardinalità $1000 \cdot 2^x$ con $1 \leq x \leq 8$ sui quali viene eseguito l'algoritmo di apprendimento tramite l'euristica logaritmica (dato che quella fattoriale fallisce dopo 160 campioni).



(a) Con 20 campioni

(b) Con 200 campioni

Figura 3: Reti bayesiane generate con massimo 5 padri per nodo; i nodi con bordo rosso indicano i collider; è possibile confrontare questi grafi con quello originale disponibile all'indirizzo `https://www.bnlearn.com/bnrepository/insurance/insurance.svg`

2.3.1 Risultati

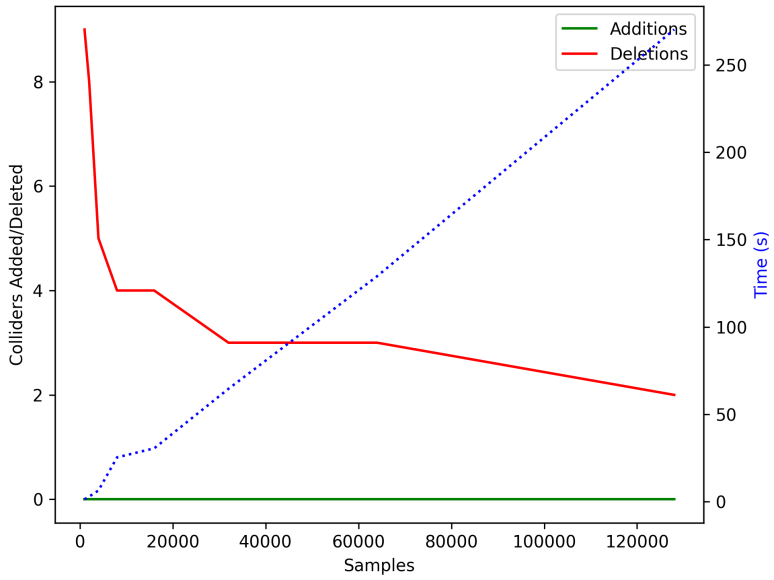


Figura 4: relazione tra il numero di collider errati e il numero di campioni; sull'ordinata destra è indicato il tempo di esecuzione

Si può notare che nonostante un numero esponenziale di campioni, la rete generata non risulta identica a quella di partenza da cui è stata generata la distribuzione, infatti il numero di collider in eccesso raggiunge lo zero, ma il numero di collider in difetto diminuisce in maniera molto lenta. Questo può indicare che l'algoritmo cerca di approssimare la rete ad una rete più "*libera*" che contiene meno dipendenze condizionali. Si è preferito fermarsi a 128000 campioni in quanto oltre tale soglia i tempi di esecuzione diventano superiori ai 5 minuti.

3 Conclusioni

Dobbiamo ribadire che tutti i test svolti assumono che l'ordine delle variabili preso in considerazione dall'algoritmo sia ottimale, ovvero che ogni nodo sia figlio solamente di nodi precedenti rispetto all'ordinamento; questo è un requisito molto forte che non è sempre verificato, infatti in scenari di apprendimento di reti bayesiane spesso abbiamo a disposizione solamente un insieme di campioni i.i.d.. A tal proposito l'articolo propone (in maniera generica) di sfruttare un qualsiasi algoritmo di ricerca locale per individuare tra diversi ordinamenti casuali di variabili il migliore eseguendo *K2* su di esso e determinando un punteggio per la struttura generata[5].

[5]Gregory F. Cooper e Edward H. Herskovits. "A Bayesian Method for the Induction of Probabilistic Networks from Data". In: *Machine Learning* (1992), p. 323.