Relazione Intelligenza Artificiale

Emanuele Marcantonio 5777575

April 2021

1 Introduzione

Alla base dell'elaborato vi è la Rete Bayesiana. Questa è una struttura dati utilizzata per rappresentare una distribuzione di probabilità completa, o full joint distribution, mediante un grafo aciclico diretto (DAG) i cui nodi rappresentano delle variabili aleatorie e i cui archi esprimono le relazioni di dipendenza condizionale fra queste. Ognuno dei nodi della rete ha al suo interno una conditional probability table, che esprime la probabilità che il nodo assuma uno dei valori all'interno del suo dominio in base a quelli assunti dagli eventuali genitori. Un possibile campo di utilizzo delle reti bayesiane può essere quello medico, per tenere traccia e verificare le correlazioni fra una patologia e i suoi possibili sintomi.

2 Scopo

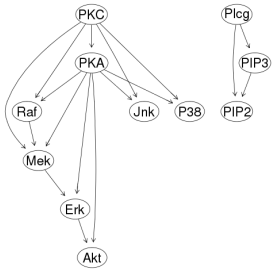
L'obiettivo dell'elaborato è l'implementazione di una metodologia di apprendimento della struttura di reti bayesiane secondo il procedimento descritto da Cooper e Herskovitz e David Heckerman (Heckerman, D.). Questo prevede, avendo un database di assegnazioni di valori dei nodi della struttura bersaglio, l'utilizzo di una funzione di score che calcola il "punteggio" di un'ipotesi di struttura possibile per la rete che lo ha generato. Successivamente si procede a effettuare una greedy hill-climbing search che ammette tre possibili mosse:

- Aggiungere un arco a meno che questo non generi un ciclo;
- Rimuovere un arco;
- Invertire il senso di un arco a meno che questo non generi un ciclo.

Fra i grafi ottenuti attraverso queste tre operazioni, si seleziona quello con lo score maggiore, fino ad ottenere quello con lo score massimo possibile.

3 Codice

L'elaborato è stato scritto in python e il database utilizzato è stato generato attraverso l'utilizzo di Hugins considerando la rete qui riportata. La metodologia sopra riportata è stata implementata all'interno delle seguenti classi:



- **graph.py**: Qui sono raccolte tutte le informazioni sull'implementazione e le funzioni di gestione di un grafo. All'interno sono presenti due classi: Graph e Node, che implementano rispettivamente il grafo e un nodo del grafo. Inoltre, sono presenti le funzioni di gestione degli archi (aggiunta, rimozione, inversione e controllo dei cicli) e due funzioni di appoggio utilizzate per individuare i coefficienti necessari al calcolo dello score.
- **graphScore.py** Qui sono presenti le implementazioni per la funzione di score e la hill-climbing search, oltre a una funzione che permette di applicare le diverse alterazioni permesse dalla ricerca e una che restituisce, ricevendo un grafo in input, una sua permutazione casuale ottenuta introducendo archi in modo casuale. Questa risulta utile a fini di testing.

4 Funzionamento

Per prima cosa viene generato il dataframe contenente i dati ottenuti da Hugin su 100 test utilizzando la libreria Pandas di python. A partire da questo, si realizza una rete di partenza priva di archi e con gli stessi nodi della rete obiettivo. A questa vengono poi aggiunti degli archi casuali utilizzando la funzione relativa presente in graphScore.py e si procede con la hill-climbing search utilizzandola come punto di partenza. Questa inizia calcolando lo score della rete in modo da poterlo confrontare con quello delle permutazioni ottenute aggiungendo, rimuovendo o invertendo un arco. La funzione di score prende in ingresso una rete e il dataframe degli esempi e, per ogni nodo all'interno della rete, realizza dataframe di appoggio per poter individuare gli N_{ij}, ovvero il numero di volte in cui i genitori del nodo i compaiono nella combinazione j, e gli N_{ijk}, ovvero il numero di volte che il nodo i assume valore k con i genitori nella configurazione j. Questi vengono quindi utilizzati per il calcolo secondo la seguente funzione di score:

$$p(D \mid S^h) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij} + N_{ij})} \cdot \prod_{k=1}^{r_i} \frac{\Gamma(\alpha_{ijk} + N_{ijk})}{\Gamma(\alpha_{ijk})}$$

Considerando $\alpha_{ij} = \alpha_{ijk} = 1$ secondo la tecnica del Laplace Smoothing. Una volta calcolati gli score delle reti permutate, viene selezionata quella con lo score maggiore fra loro e lo si confronta con quello della rete di partenza. Se questo è maggiore, la rete iniziale viene sostituita con quella permutata e su questa verrà di nuovo applicato il procedimento di ricerca, altrimenti verrà incrementato un counter che, una volta raggiunto il valore massimo specificato nell'input della funzione, terminerà la ricerca restituendo l'ultima rete considerata come risultato. Nel caso in cui si abbia l'uguaglianza fra i due score confrontati, la funzione permette fino a cinque sidemoves, in modo da evitare eventuali plateau.

5 Conclusioni

L'implementazione dell'algoritmo così effettuata non presenta risultati ottimali, specialmente su dataset di grandi dimensioni: questo perché lo score calcolato utilizzando il metodo riportato arriva ad assumere valori al di sotto del valore minimo rappresentabile dalla macchina in float $2.2250738585072014e-308$, valore che viene raggiunto o superato in seguito alle divisioni per $\Gamma(\alpha_{ij} + N_{ij})$, valore che spesso si rivela molto grande. Per questo motivo, spesso il programma non effettua sostanziali modifiche al grafo di partenza se non quelle permesse dalle sidemoves, mentre in altri casi tende a riportare la rete a una situazione in cui è presente il minor numero di archi possibile.