Elaborato IA **Tommaso Marigo 5813389**

Introduzione

Il progetto consiste nell'implementazione di un algoritmo di apprendimento di una rete bayesiana dato un dataset.

L'algoritmo parte da una rete senza archi e via via effettua operazioni elementari sul grafo calcolando lo score della rete ottenuta attraverso la formula di Cooper & Herskovitz.

Attraverso una strategia di ricerca locale Hill Climbing sceglie la rete con lo score più alto. Se nessun passo migliora lo score, la ricerca si arresta e restituisce la rete corrente.

Implementazione

Per l'implementazione è stato scelto di minimizzare il logaritmo negativo della formula per evitare problemi di underflow. Inoltre è stato scelto di impostare gli iperparametri con un valore unitario:

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

$$\downarrow \quad \alpha_{ijk}=1$$

$$\prod_{i=1}^{n}\prod_{j=1}^{q_i}\frac{(r_i-1)!}{(N_{ij}+r_i-1)!}\prod_{k=1}^{r_i}(N_{ijk})!$$

$$\downarrow$$

$$\sum_{i=1}^{n}\sum_{j=1}^{q_i}\log\frac{(r_i-1)!}{(N_{ij}+r_i-1)!}+\sum_{k=1}^{r_i}\log(N_{ijk})!$$

$$\downarrow$$

$$\sum_{i=1}^{n}\sum_{j=1}^{q_i}\left(\log(r_i-1)!-\log(N_{ij}+r_i-1)!\right)+\sum_{k=1}^{r_i}\log(N_{ijk})!$$

$$\downarrow$$

$$\sum_{i=1}^{n}q_i\cdot\log(r_i-1)!-\sum_{j=1}^{q_i}\log(N_{ij}+r_i-1)!+\sum_{k=1}^{r_i}\log(N_{ijk})!$$

Score function

L'indice "*n*" è relativo ad ogni nodo della rete.
La subroutine descritta in questo paragrafo calcola il fattore di uno specifico nodo preso in ingresso. Per calcolare lo score di una particolare rete, l'algoritmo chiama questa funzione per ogni nodo e poi somma tutti i valori restituiti.
La subroutine prende in ingresso:

- la rete corrente
- il nodo di cui vogliamo calcolare il fattore
- il dataset
- il valore r_i del nodo in questione (indice della terza sommatoria)
- l'insieme *ArrayIndex[]* che memorizza per ogni nodo l'indice nel dataset

Con questi dati la funzione crea due insiemi di oggetti, ciascuno dei quali contenente una coppia di elementi:

- *padri[]* contiene oggetti contenenti a loro volta una configurazione dei padri del nodo (sotto forma di **stringa** concatenata) e un **intero** rappresentante il numero di occorrenze di tale configurazione nel dataset (la cardinalità di *padri[]* corrisponde all'indice q_i della seconda sommatoria, mentre il numero di occorrenze di ciascuna configurazione corrisponde ai valori N_{ij}*).*
- *padri_e_figlio[]* è analogo con l'unica differenza che gli oggetti contengono le configurazioni dei padri più il figlio (il nodo stesso).

Questi due insiemi servono rispettivamente per il calcolo della seconda e della terza sommatoria:

$$\sum_{i=1}^{n} \underbrace{q_i \cdot \log(r_i - 1)!}_{①} - \underbrace{\sum_{j=1}^{q_i} \log(N_{ij} + r_i - 1)!}_{②} + \underbrace{\sum_{k=1}^{r_i} \log(N_{ijk})!}_{③}$$

① `fattore=len(padri) * log(factorial(Ri-1))`

②
```
for obj in padri:
    fattore=fattore - log(factorial(obj.num_istanze+Ri-1))
```

③
```
for obj in padri_e_figlio:
    fattore=fattore + log(factorial(obj.num_istanze))

return -fattore
```

Inizializzazione

Come prima cosa vengono inizializzati tre insiemi utili per l'Hill Climbing:

- *ArrayIndex[u] =* indice del nodo "u" nel dataset.
- *ArrayScore[u] =* fattore di score riferito al nodo "u" (data la rete corrente e il dataset).
- *ArrayRi[u] =* numero di valori che assume il nodo "u" nel dataset.

La somma di tutti i valori di *ArrayScore[]* costituisce lo score della rete inziale.

L'algoritmo

L'algoritmo è diviso in tre parti: rimozione, inversione e inserimento.
Ad ogni operazione elementare, lo score temporaneo viene aggiornato sottraendo il fattore del nodo nello stato precedete e aggiungendo il fattore dello stesso nodo nel nuovo stato.
Se lo score temporaneo è minore del corrente (quindi migliore), allora lo score corrente viene aggiornato, viene memorizzato il nodo in questione e l'operazione elementare eseguita.
Al termine di un iterazione la lista *ArrayScore[]* dei fattori viene aggiornata in funzione dei dati memorizzati, inoltre viene salvata la rete corrente in formato PNG.

Risultati

L'algoritmo è stato testato con un dataset di 30000 casi e 37 variabili.
In alto la rete reale, mentre in basso la rete generata dall'algoritmo: