Elaborato Intelligenza Artificiale
Angelo D'Amante
5756781

APPRENDIMENTO DELLA STRUTTURA BAYESIANA

Introduzione

Il progetto consiste nella costruzione di una rete Bayesiana partendo da un Dataset di una rete generato attraverso il software Hugin Lite. La rete presa in esame è *"Asia"*, che esamina il cancro al polmone. Prelevata dal sito *bnlearn.com.*

L'apprendimento viene effettuato applicando al dataset (dati osservati) un modello "score-based". Lo score descrive la *"bontà di adattamento complessiva"* della rete. A tal proposito viene implementata la formula di *Cooper & Herskovits* che calcola un punteggio attraverso la probabilità a posteriori che utilizza come probabilità a priori la coniugata di Dirichlet.

L'algoritmo che effettua la ricerca del modello ottimale (che massimizza il punteggio) è un algoritmo di ricerca locale. È stato scelto *l'Hill-Climbing*.

Implementazione

Per l'implementazione è stato scelto di voler minimizzare il logaritmo negativo per evitare problemi di underflow.
Applichiamo alla formula di Cooper la funzione di *log-verosomiglianza*.

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

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

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

$$\downarrow$$

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

$$\downarrow$$

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

$$\downarrow$$

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

Score Function

Lo score viene calcolato per ogni nodo, infatti per ottenere lo score di una rete viene effettuata la somma di tutti gli score dei nodi. Ovvero, l'algoritmo chiamerà questa funzione per ogni nodo e alla fine ne fa la somma.

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
- l'insieme *ArrayIndex[]* che memorizza per ogni nodo l'indice nel dataset

La funzione crea due insiemi di oggetti, ciascuno dei quali contenente una coppia di elementi:

- *padri[]* contiene oggetti con due attributi: (la cardinalità di *padri[]* è q_i)
  - configurazione dei padri del nodo (sotto forma di stringa concatenata)
  - il numero di occorrenze di tale configurazione nel dataset (che è N_{ij})
- *padri_e_figlio[]* è analogo al primo ma gli oggetti contengono le configurazioni dei padri più il figlio (nodo stesso)

Questi due insiemi servono per il calcolo della seconda e terza sommatoria, vediamo nel dettaglio il codice:

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

```
# Calcolo della Prima Sommatoria
fattore = len(padri) * log(factorial(Ri - 1))

# Calcolo della seconda sommatoria
for obj in padri:
    fattore = fattore - log(factorial(obj.num_istanze + Ri - 1))

# Calcolo della terza sommatoria
for obj in padri_e_figlio:
    fattore = fattore + log(factorial(obj.num_istanze))
```

Inizializzazione

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".
- *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.

Algoritmo

Questo tipo di procedura si muove nello spazio dei grafi modificando un arco alla volta (se l'arco non è presente, lo aggiunge, in caso contrario, lo inverte o lo elimina). Si arresta quando nessuno di questi cambiamenti produce un incremento (nel mio caso decremento) nello score.

Ad ogni operazione elementare (aggiunta, rimozione e inversione), lo score temporaneo viene aggiornato sottraendo il fattore del nodo nello stato precedente (per far ciò salvo lo score di ogni nodo) 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.
- Operazione elementare eseguita.
- La lista *ArrayScore[]* viene aggiornata.

Inoltre, al termine viene salvata la rete in formato *PNG*.

```
# Ricerca locale per i grafi orientati
while (tipo != 'not found'):
    tipo = 'not found'  # quando lo score non cambia per la prima volta
    c = 0
    print(' ')
    first_score = best_score

    # ciclo in tutti gli archi
    for (u, v) in G.edges():...
    # fine ciclo

    comp = list(complement(G).edges())
    for (u, v) in complement(G).edges():...

    for (u, v) in comp:  # AGGIUNGE
        G.add_edge(u, v)
        c = c + 1
        if is_directed_acyclic_graph(G) == True:
            tmp_score = first_score - ArrayScore[v] + Formula(G, v, dataset, ArrayRi[v],
                                                              ArrayIndex)  # score della rete
            if tmp_score < best_score:
                best_score = tmp_score
                tipo = 'add'
                padre = u
                figlio = v
        G.remove_edge(u, v)
    if (tipo == 'rem'):...

    elif (tipo == 'rev'):...

    elif (tipo == 'add'):...

    # stampo le modifiche
    if (tipo != 'not found'):...
# fine While
```

Gli *if* chiusi sono le fasi di aggiornamento. Per esempio, l'if che si occupa della rimozione, rimuove l'arco e aggiorna lo score nella lista. Quindi aggiorna *ArrayScore[figlio]*.

Risultati

L'algoritmo è stato testato con un dataset di 15000 casi e 8 variabili (ASIA). In alto, la rete reale, mentre in basso la rete generata dall'algoritmo: