EXPECTATION MAXIMISATION

Cavenaghi Emanuele, Zanga Alessio

MAD

UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

SCELTE IMPLEMENTATIVE

Python

Strutture implementate

Collegamenti alla libreria BNlearn

MAD

SCELTE IMPLEMENTATIVE

Python

È stato scelto di utilizzare il linguaggio Python per svariati motivi, primo tra tutti il solido supporto e il sempre più vasto impiego.

NetworkX[1]

Altra motivazione è la possibilità di avvalersi delle numerose librerie disponibili tra le quali «**NetworkX**» che rende disponibili strutture per la gestione di grafi, grafi orientati, alberi, ecc. Insieme alle strutture è ovviamente disponibile una vasta collezione di algoritmi su grafi con cui poter confrontare i risultati ottenuti dalle nostre implementazioni.

Numba[2]

Infine, è presente la libreria «**Numba**» che consiste in un Just-In-Time compiler per Python in modo da ottenere codice altamente efficiente. Si mantiene così il vantaggio di scrivere codice di alto livello combinandolo con il vantaggio di velocità dato dalla compilazione di codice a basso livello.

Inoltre, dove possibile, rende più facile il compito di parallelizzare le funzioni sia per CPU che per GPU supportando il linguaggio Nvidia CUDA.

SCELTE IMPLEMENTATIVE

Strutture implementate

Sono state implementate diverse strutture generiche nella libreria creata per poter immagazzinare le informazioni richieste durante la costruzione dell'algoritmo Junction Tree.

Graph

La classe **Graph** rappresenta un generico grafo, implementato tramite matrice di adiacenza binaria, contiene tutti i metodi per poter operare e manipolare il grafo. Estendendo questa classe è stata creata la classe **DirectedGraph** che rappresenta in modo specifico grafi orientati, differenziando, per esempio, tra nodi *parent* e nodi *child*, non presenti nella classe **Graph**.

Tree

La classe **Tree** rappresenta un generico albero costituito da nodi di cui si conosce però solo il nodo *root*. Parte fondamentale della classe è la classe **Node** che contiene al suo interno i riferimenti ai propri nodi *child* e al proprio nodo *parent* oltre agli attributi del nodo stesso.

La classe **JunctionTree** è un'estensione della classe **Tree** e ne condivide tutti gli attributi e metodi oltre ad avere una funzione di plot personalizzata e una funzione che consente, dato il DAG di una Bayesian Network, di costruire il corrispondente Junction Tree.

SCELTE IMPLEMENTATIVE

Collegamenti alla libreria BNlearn

La libreria BNlearn[3] resta comunque un punto di partenza per eseguire comparazioni, test e verifiche vista la grande vastità delle risorse disponibili e gli anni di test e miglioramenti apportati.

- Creazione del DAG partendo dalla struttura in stringa (e.g. [A][C][B|A:C])
- Import dei dataset disponibili
- Import dei file delle Bayesian Networks con le *Probability Conditional Table* nei vari formati

EXPECTATION MAXIMISATION

Pseudo-codice

Schema riassuntivo

Impute missing data

MAD

EXPECTATION MAXIMISATION

Pseudo-codice

Il processo che racchiude gli step di Expection e Maximisation viene ripetuto per t volte controllando due condizioni di arresto:

- Raggiunto il massimo numero di iterazioni volute
- Le CPT θ^t hanno un miglioramento trascurabile rispetto alle CPT θ^{t-1} (convergenza)

Nello pseudo-codice Il DAG è indicato con $\mathcal{G}$ mentre il dataset (incompleto) è indicato con $\mathcal{D}$.

```
def Expectation_Maximisation(𝒢,𝒟):

    inizializza le CPT θ^0 con una distribuzione uniforme
    for each t = 0,1,… fino alla convergenza
        // E-step
        Calcola le frequenze attese sul dataset 𝒟 utilizzando
        le CPT θ^(t−1) per imputare i dati mancanti
        // M-step
        Calcola le CPT θ^t del passo t con le frequenze attese
        calcolate nello step di Expectation

return CPT θ^t
```

EXPECTATION MAXIMISATION

Schema riassuntivo

Inizializzazione

Inizializza le $CPT\ \theta^0$ con la distribuzione uniforme

Iterazione

Verifica i due criteri di stop

E-Step

Calcola le ***frequenze attese*** utilizzando le CPT θ^{t-1} per imputare i dati mancanti

M-Step

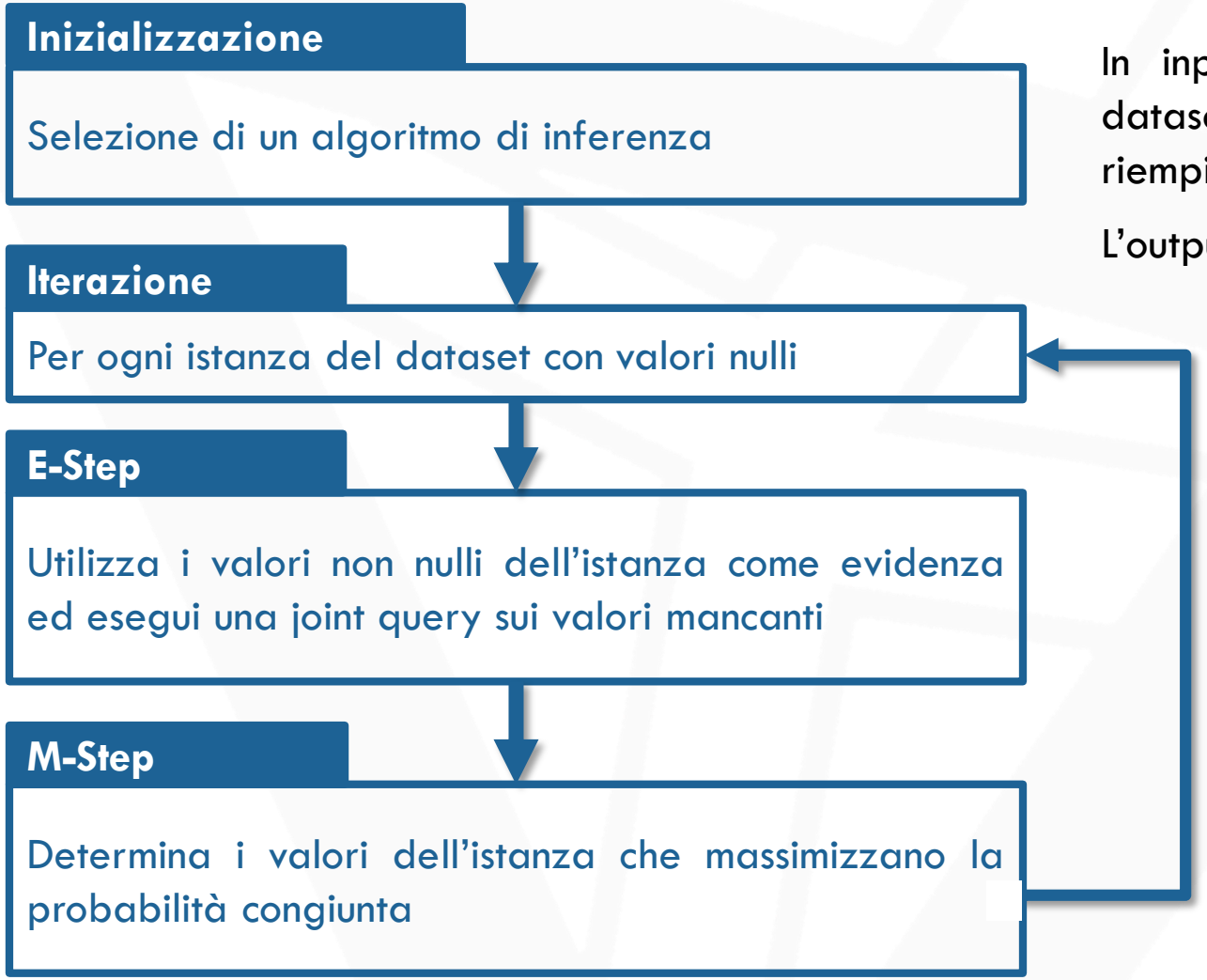
Calcola le CPT θ^t con le frequenze attese calcolate nello step di Expectation

In input vengono passati il DAG della rete e il dataset incompleto che deve essere riempito con i dati imputati.

L'output saranno le CPT θ^t calcolate all'ultimo passo dell'iterazione

EXPECTATION MAXIMISATION

Impute missing data



In input vengono passati la rete e il dataset incompleto che deve essere riempito con i dati imputati.

L'output sarà un dataset completato.

STRUCTURAL EXPECTATION MAXIMISATION

Pseudo-codice

Schema riassuntivo

MAD

STRUCTURAL EXPECTATION MAXIMISATION

Pseudo-codice

L'algoritmo ***Structural EM*** richiede in ingresso solamente il dataset incompleto $\mathcal{D}$ e calcolerà contemporaneamente sia Il DAG $\mathcal{G}$ che le CPT θ durante le iterazioni.

```
def Structural EM(𝒟):

    inizializza le CPT θ^0 con una distribuzione uniforme

    for each t = 0,1,… fino alla convergenza
        // E-step
        Imputa i dati mancanti con le CPT θ^(t−1) e il DAG 𝒢^(t−1) le e riempi i
        valori mancanti nel dataset 𝒟
        // M-step
        Calcola il DAG 𝒢^t con il dataset riempito con i dati imputati
        Calcola le CPT θ^t eseguendo Expectation-Maximisation su 𝒢^t e il
        dataset con i dati imputati

return CPT θ^t
```

STRUCTURAL EXPECTATION MAXIMISATION

Schema riassuntivo

Inizializzazione

Inizializza le $CPT\ \theta^0$ con la distribuzione uniforme

Iterazione

Verifica i due criteri di stop

E-Step

Costruisci il dataset $\mathcal{D}^t$ imputando i dati utilizzando le CPT θ^{t-1} e il DAG $\mathcal{G}^{t-1}$

M-Step

- Costruisci il DAG $\mathcal{G}^t$ usando il dataset imputato $\mathcal{D}^t$
- Esegui EM per calcolare le CPT θ^t

In input viene passato dataset incompleto che deve essere riempito con i dati imputati.

L'output saranno le CPT θ^t calcolate all'ultimo passo dell'iterazione e il DAG $\mathcal{G}^t$

BIBLIOGRAFIA

- [1] NetworkX: https://networkx.github.io/
- [2] Numba: http://numba.pydata.org/
- [3] BNlearn: https://www.bnlearn.com/
- [4] Probabilistic Graphical Models, Principles and Techniques (D. Koller and N. Friedman)

THANK YOU

Cavenaghi Emanuele

Zanga Alessio

MAD

UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA