Inferenza con MAP e MPE

Progetto di Intelligenze Artificiali e Laboratorio - Parte 3

Emanuele Gentiletti, Alessandro Caputo

Indice

Introduzione

Il progetto prevedeva l'implementazione di inferenza probabilistiche su reti bayesiane tramite gli algoritmi MPE e MAP. Per portare a compimento il progetto, abbiamo fatto uso della libreria `aima-python` per l'implementazione delle reti bayesiane (che abbiamo opportunamente modificato, per permettere l'esecuzione degli algoritmi in reti con variabili non booleane). Abbiamo inoltre implementato un parser per il formato XMLBIF, che abbiamo usato per testare gli algoritmi implementati sulle reti disponibili in BayesianExchange.

In seguito discutiamo delle implementazioni degli algoritmi e delle prove fatte.

Implementazione

MPE

L'implementazione dell'inferenza con MPE è composta da quattro funzioni.

`mpe_ask` crea un fattore per ogni variabile della rete bayesiana e salva tutte le variabili di cui bisogna fare il max out nella lista `to_max_out`, ovvero tutte tranne l'evidenza.

```
def mpe_ask(e, bn: BayesNet):
    factors = [
        make_maxout_factor(var, e, bn) for var in reversed(bn.variables)
    ]
    to_max_out = [var for var in reversed(bn.variables) if var not in e]
    return maxout_all(factors, to_max_out, bn)
```

`maxout_all` richiama semplicemente la funzione `max_out` per ogni variabile all'interno di `to_max_out`.

```
def maxout_all(factors, to_max_out, bn):
    for var in to_max_out:
        factors = max_out(var, factors, bn)

    factor = factors[0]
    return factor.previous_assignments[()], factor.cpt[()]
```

La funzione `max_out` separa i fattori che contengono la variabile da quelli che non la contengono, ed esegue il `pointwise_product` di tutti i fattori che la contengono. Dal fattore risultante, viene eseguito il maxout della variabile richiesto, e vengono infine restituiti i fattori che non contengono la variabile con concatenato il fattore su cui è stato eseguito il maxout.

```
def max_out(var, factors, bn):
    """Eliminate var from all factors by summing over its values."""
    result, var_factors = [], []
    for f in factors:
        if var in f.variables:
            var_factors.append(f)
        else:
            result.append(f)
    result.append(pointwise_product(var_factors, bn).max_out(var, bn))
    return result
```

`max_out` (metodo di `MaxoutFactor`) effettua il max out di una determinata variabile all'interno del fattore. Per ogni evidenza calcola la probabilità condizionata su tutte le variabili salvando il massimo, uscito dal ciclo aggiorna la CPT e salva il valore massimo in un dizionario. Restituisce infine il nuovo fattore ottenuto.

Il fattore restituito è di tipo `MaxoutFactor`, un wrapper attorno alla classe `Factor` della libreria `aima-python` che aggiunge l'operazione di maxout e che mantiene la memoria delle assegnazioni fatte durante questa fase.

```
def max_out(self, var, bn) -> "MaxoutFactor":
    """Make a factor eliminating var by summing over its values."""
    variables = [X for X in self.variables if X != var]
    cpt = {}
    mpe = {}

    for e in all_events(variables, bn, {}):
        cpt_key = event_values(e, variables)
        max_p = -1
        max_value = None
        for val in bn.variable_values(var):
            val_p = self.p({**e, var: val})
            if max_p < val_p:
                max_value = val
                max_p = val_p

        cpt[cpt_key] = max_p
        mpe[cpt_key] = {
            **self.previous_assignments[
                event_values({**e, var: max_value}, self.variables)
            ],
            var: max_value,
        }

    return MaxoutFactor(Factor(variables, cpt), mpe)
```

MAP

Abbiamo implementato il metodo di inferenza MAP sfruttando parte delle funzioni scritte per il metodo MPE.

```
def map_ask(Ms: List[str], e: dict, bn: BayesNet) -> Tuple[dict, float]:
    factors = []
    for var in reversed(bn.variables):
        factors.append(make_factor(var, e, bn))
        if is_hidden(var, Ms, e):
            factors = sum_out(var, factors, bn)

    return maxout_all([MaxoutFactor(f) for f in factors], Ms, bn)
```

`map_ask` crea i fattori per ogni variabile all'interno della rete bayesiana. Se la variabile è una variabile nascosta (in questo contesto, intendiamo non MAP), la funzione effettua il sumout di quella variabile dal fattore. Una volta fatto il sumout di tutte le variabili, la funzione richiama `maxout_all` prendendo

come parametro la lista dei fattori risultanti dal sumout e le variabili MAP. Da questo punto in poi l'algoritmo proseguirà come per MPE per tutte le variabili MAP rimaste.

Test

Per verificare il funzionamento di MPE e MAP abbiamo scritto dei test, disponibili in `project_bn/tests/test_inferences.py`. Abbiamo scritto due test per MAP e uno per MPE (dato che MAP riusa il codice di MPE per il maxout, abbiamo considerato il funzionamento dei test di MAP come una conferma del funzionamento di MPE).

I test usano le reti `SprinklerPlus`, usata negli esempi a lezione, ed `earthquake.xml`, disponibile nella cartella delle risorse (corrispondente all'esempio Burglary del Russell e Norvig).

Attraverso i test abbiamo verificato che i valori restituiti dalle funzioni MAP e MPE siano uguali a quelli restituiti dall'applicativo Samiam.

Le prove fatte sono:

- `MAP` su `SprinklerPlus`
  - variabili di MAP: $Season$ e $Sprinkler$
  - evidenza $GrassWet = true$ e $RoadWet = false$
  - risultato: $Season = summer$ e $Sprinkler = true$
  - $P(map, e) \approx 0.0992$
- `MAP` su `earthquake.xml`
  - variabili di MAP: $Burglary$, $Earthquake$, $JohnCalls$, $MaryCalls$
  - evidenza: nessuna
  - risultato: tutte le variabili $false$
  - $P(map, e) \approx 0.991$
- `MPE` su `SprinklerPlus`
  - evidenza $GrassWet = true$ e $RoadWet = false$
  - risultato: $Season = summer$, $Sprinkler = true$, $Cloudy = false$ e $Rain = no$
  - $P(map, e) \approx 0.0938$

Analisi

Abbiamo confrontato i due metodi di inferenza prendendo in considerazione le seguenti variabili:

1. Grandezza e complessità della rete bayesiana
2. Numero di evidenze
3. Numero di variabili MAP

Il confronto è stato effettuato su tre domini:

Name	Nodes	Arcs	Parameters
Child	20	25	230
Insurance	27	52	984
Hailfinder	56	66	2656

- Child e Insurance hanno grandezza simile ma complessità differente
- Hailfinder e ha dimensione e complessità maggiori

Per effettuare le prove ci siamo serviti della funzione `create_random_evidence` che assegna a una determinata percentuale di nodi un valore casuale del suo dominio. Lo stesso ragionamento è stato applicato per selezionare le variabili di MAP, data una determinata percentuale la funzione crea una lista di variabili casuali diverse da quelle dell'evidenza.
La quantità di nodi è stata espressa in percentuale al fine di dare alle misurazioni una valenza più oggettiva.

Grandezza

Per confrontare le due inferenze in base alla grandezza del dominio abbiamo deciso di tenere fissa la percentuale dei nodi evidenza al 10% e la percentuale di variabili MAP al 50%. Il tempo è espresso in secondi.

Name	MPE	MAP
Child	0.009964045183221193	0.11299861327879827
Hailfinder	23.262385153	56.359243749

Come possiamo vedere dai risultati le prestazioni di MPE sono molto maggiori rispetto a MAP, in linea con quanto ci aspettavamo. In generale abbiamo notato anche un alto dispendio di memoria, probabilmente in parte dovuto all'implementazione «naive» degli algoritmi.

Complessità

A parità di dimensioni abbiamo confrontato gli algoritmi con reti di complessità differente, ovvero con un numero significativamente diverso di archi. Le percentuali di nodi evidenza e MAP sono sempre fissi al 10% e 50%.

Net	MPE	MAP
Child	0.009964045183221193	0.11299861327879827
Insurance	20.89039581174121	99.12001117187499

Dai risultati si evince che a parità di dimensioni la complessità incide molto sulle prestazioni degli algoritmi.

Evidenza

Un ulteriore analisi è stata effettuata sulla percentuale di nodi evidenza. Nella tabella sottostante vediamo le prestazioni degli algoritmi per percentuali di nodi evidenza crescenti tenendo fisso invece il numero di variabili MAP al 40%.

Net: Child

Evidence percentage	MPE	MAP
10%	0.010847587455143557	0.10104558852493799
25%	0.0063954140551451474	0.08836202614241456
50%	0.002104847541587678	0.0023859823336646034

In questo caso vediamo come all'aumentare della percentuale dei nodi evidenza diminuisce il tempo di esecuzione. La differenza di prestazioni tra MPE e MAP si conserva, anche se la grandezza della rete non ci permette di fare un analisi accurata nonostante i risultati siano stati calcolati su una media di 100 iterazioni.

Variabili MAP

Nell'ultima analisi confrontiamo i metodi di inferenza tenendo fisse le percentuali di nodi evidenza al 10% e aumentando invece la percentuale di variabili MAP.

Net: Child

MAP variables percentage	MPE	MAP
25%	0.011996993480229857	0.027531892575056605
50%	0.009964045183221193	0.11299861327879827
75%	0.021417728341590732	0.6011614255836181

Come possiamo vedere anche in questo caso le differenze prestazionali tra MAP e MPE sono evidenti, notiamo però che all'aumentare delle percentuali l'algoritmo MAP aumenta i tempi di esecuzione. Ipotizziamo che tutto ciò sia dovuto a questa specifica implementazione di MPE. Aumentare le variabili MAP significa effettuare più operazioni di max out attraverso MPE, di conseguenza è molto probabile che la causa di questo fenomeno sia legato all'implementazione di questo algoritmo.

Conclusioni

Abbiamo confrontato gli algoritmi notando delle differenze sostanziali in termini di prestazioni. MAP risulta sempre più lento di MPE al variare dei parametri presi in considerazione. In generale si ha un importante aumento del tempo di esecuzione da parte dei due gli algoritmi in seguito all'aumento di dimensioni e complessità.

Lavorando con domini piccoli in alcuni casi non è stato possibile mettere in evidenza differenze significative. MPE all'aumentare della percentuale di evidenza non sembra diminuire il tempo di esecuzione, ma bisogna prendere in considerazione il fatto che questi intervalli di tempo equivalgono a frazioni di secondo. Nonostante le piccole cardinalità dei domini siamo riusciti ad evidenziare le differenze con MAP.