**Caso di studio per l’esame di Ingegneria della Conoscenza**

Fabio Nardelli mat. 655172

Ho scelto di implementare un articolo scientifico, “Learning Optimal Decision Trees with SAT” (Narodytska et al. 2018) che riguarda l’apprendimento di alberi di decisione binari ottimali (nel numero di nodi) mediante soddisfacibilità booleana.

**Motivazioni**

Gli alberi di decisione sono una delle tecniche di classificazione più utilizzate nell’ambito del machine learning (d’ora in poi ML). Benché si tratti di uno strumento semplice se paragonato ad altri tipi di classificatore, gli alberi di decisione presentano il vantaggio di poter spiegare le predizioni effettuate (basta “visitare” l’albero dalla radice fino alla foglia di destinazione per ottenere una regola di classificazione), pertanto risultano quantomai attuali, dal momento che il ML è sempre più pervasivo, e in molti contesti (specie se *safety-critical* o *mission-critical*) si desidera poter motivare le decisioni prese. A tal proposito, recentemente ha assunto sempre più importanza la XAI (eXplainable Artificial Intelligence), che si pone proprio l’obiettivo di realizzare sistemi intelligenti in grado di spiegare le decisioni prese. A titolo meramente orientativo, una rapida ricerca su Google Scholar mostra che solo nel 2020 sono stati pubblicati circa 5.590 articoli sulla XAI, contro i 3.940 del 2019 e i 2.270 del 2018. Si tratta quindi di un tema estremamente attuale. Alberi di decisione di dimensione minima, oltre ad essere più efficienti nelle predizioni, perché più rapidi da visitare, e ad occupare meno memoria, permettono di ottenere spiegazioni più concise e facili da comprendere. Di qui l’esigenza di trovare un metodo per apprendere alberi di decisione di dimensione minima. Tuttavia, si tratta di un problema NP-Hard, pertanto nella pratica si utilizzano algoritmi euristici che non garantiscono l’ottimalità in termini di dimensioni. Di seguito si propone un metodo per trovare alberi di decisione ottimali (ovvero di dimensione minima) attraverso un problema di soddisfacibilità booleana.

**Il problema della classificazione binaria**

Consideriamo un insieme di feature , ciascuna della quali assume valore nell’insieme , e un insieme di esempi di training} partizionato in e , ossia tale per cui ciascun elemento appartiene alla classe positiva o a quella negativa. Dal momento che le feature sono binarie, rappresentiamo un letterale associato alla feature con se , o con se . Rappresentiamo un generico esempio con la coppia , dove è l’insieme dei letterali associati all’esempio, e è la classe alla quale l’esempio appartiene. Si ha che se e se . Denotata con la funzione che associa a ciascun esempio la classe di appartenenza, il nostro obiettivo è quello di apprendere una funzione che approssima in modo che coincida con essa sui dati di training e che sia in grado di generalizzare *sufficientemente bene* su dati non visti. In questo contesto, apprendere significa apprendere un albero di decisione binario.

**Alberi di decisione**

Un albero di decisione è un albero in cui ogni nodo interno è etichettato con una condizione, ovvero una funzione booleana su una feature (o il valore della feature, se booleana o binaria), e ha esattamente due figli, uno corrispondente al valore *true* della funzione, l’altro al valore *false* (ovvero, i due archi che collegano il genitore ai figli sono etichettati rispettivamente con *true* e *false*). Ogni nodo foglia è etichettato con una classe, ovvero un elemento del dominio della feature target che si vuole predire. In questo contesto ci occupiamo di classificazione binaria, pertanto ci saranno sempre due classi, che chiameremo rispettivamente *positiva* e *negativa*, dato che lavoreremo con dataset binari. La classificazione, o predizione della classe di un esempio, consiste in una visita dell’albero: partendo dalla radice, a ogni nodo interno viene valutata la condizione corrispondente e, sulla base della verità di questa, ci si dirige verso il figlio sinistro o destro. Alla fine, si arriverà a una foglia che indicherà la classe di attribuzione per quel particolare esempio. Un albero di decisione corrisponde al costrutto della selezione (*if-then-else*) tipico della programmazione strutturata. Dato un insieme di esempi di training, è possibile costruire diversi alberi di decisione *consistenti* con essi (ovvero, in grado di classificare correttamente gli esempi di training), perciò occorre effettuare una scelta: un approccio consiste nel preferire il più piccolo albero di decisione consistente con i dati di training, intendendo quello con il minor numero di nodi o la minore altezza. L’apprendimento si configura quindi come un problema di ricerca di un siffatto albero. Poiché lo spazio di ricerca può essere molto vasto, tipicamente gli algoritmi di apprendimento di alberi di decisione effettuano una ricerca *greedy*, con l’obiettivo di minimizzare l’errore, consistente nello scegliere, come condizione di split (ovvero di *branching* nell’albero), quella che produrrebbe la migliore classificazione se fosse ammesso solo uno split. Per questa ragione, si parla anche di split *myopically optimal*. Peri valutare le predizioni, si utilizza in genere la *likelihood* (o la *log-likelihood*), come criterio da massimizzare, oppure la *log-loss* (la likelihood negata divisa per il numero di esempi di training), come misura di errore da minimizzare.

**CSP e soddisfacibilità booleana**

Molti problemi (ad es. di scheduling) possono essere modellati come CSP (Constraint Satisfaction Problem), ovvero problemi di soddisfacimento di vincoli. Un CSP è costituito da un insieme di variabili e da un insieme di vincoli definiti su di esse.

La logica proposizionale è un linguaggio formale con il quale si possono rappresentare vincoli (intensionali) mediante proposizioni. Una proposizione, o formula logica, è una frase scritta in un dato linguaggio, utilizzando opportuni connettivi logici, che ammette un valore di verità (*true*/*false*).

Un CSP finito (con un finito insieme di variabili dal dominio finito) può essere espresso mediante logica proposizionale. Ogni vincolo è rappresentato con una clausola, ovvero un’espressione logica in una forma normale (tipicamente la CNF – *Conjunctive Normal Form* – in cui una clausola è costituita da congiunzioni di disgiunzioni di letterali). Per mappare una variabile del CSP nella rappresentazione booleana si utilizzano tante variabili indicatrici quanti sono i valori del dominio di , tali che se . A ciascuna di queste variabili si assocerà un *letterale* , per rappresentare il fatto che la variabile assume o non assume il valore corrispondente. Ad esempio, dato , lo convertiamo dapprima nelle variabili indicatrici , e successivamente rappresentiamo queste ultime con gli atomi . Per “costringere” una variabile ad assumere esattamente un valore alla volta, aggiungiamo i vincoli (la variabile deve assumere almeno un valore) e (la variabile può assumere al più un valore). I CSP proposizionali (*alias* problemi di soddisfacibilità booleana, noti come *SAT* in letteratura) possono essere risolti efficientemente da algoritmi come il DPLL (*Davis-Putnam-Logemann-Loveland*), che si basa sul pruning dei domini e dei vincoli, sulla separazione dei domini e sull’assegnazione dei letterali puri (ad es. se dopo le semplificazioni compare solo , si può assegnare a ).

**Encoding proposto**

Gli alberi di decisione binari sono alberi binari completi, in cui cioè ogni nodo ha esattamente zero o due figli. Pertanto, il numero di nodi sarà sempre dispari. Per convenzione, numeriamo i nodi in ordine crescente da sinistra a destra, a partire dalla radice (che sarà il nodo *1*). Dati nodi e un generico nodo , i suoi figli possono essere numerati nel range di numeri naturali da a .

Di seguito elenchiamo le variabili e i vincoli utilizzati, suddividendoli tra quelli necessari a rappresentare un valido albero binario completo, e quelli che realizzano l’apprendimento di un albero di decisione.

**Variabili**

Dati nodi, feature binarie, e definiti gli insiemi e , definiamo le seguenti variabili (di seguito adottiamo i valori e in luogo di e per brevità):

|  |  |
| --- | --- |
| **Variabile** | **Descrizione** |
| *Variabili per la rappresentazione di un albero binario completo* | |
|  | 1 sse il nodo è una foglia, |
|  | 1 sse il nodo ha il nodo come figlio sinistro, , |
|  | 1 sse il nodo ha il nodo come figlio destro, , |
|  | 1 sse il nodo ha come genitore il nodo , , |
| *Variabili per l’apprendimento di un albero di decisione* | |
|  | 1 sse la feature è assegnata al nodo , , |
|  | 1 sse la feature è discriminata dal nodo , , |
|  | 1 sse la feature è discriminata per il valore 0 dal nodo o da uno dei suoi antenati, , |
|  | 1 sse la feature è discriminata per il valore 1 dal nodo o da uno dei suoi antenati, , |
|  | sse la classe del nodo foglia è 1, |

**Vincoli**

**Vincoli per la rappresentazione di un albero binario completo**

1. La radice non deve essere una foglia:
2. Se un nodo è una foglia, non ha figli:
3. I figli sinistro e destro dell’*i-esimo* nodo sono numerati consecutivamente, rispettivamente:
4. Un nodo non foglia deve avere un figlio:

|  |  |
| --- | --- |
|  |  |

1. Se l’*i-esimo* nodo è un genitore, deve avere un figlio sinistro e uno destro:
2. Tutti i nodi tranne la radice devono avere un genitore:

|  |  |  |
| --- | --- | --- |
|  |  |  |

**Vincoli per l’apprendimento di un albero di decisione**

1. Per discriminare una feature per il valore al nodo :

|  |  |  |
| --- | --- | --- |
|  |  |  |

1. Per discriminare una feature per il valore al nodo :

|  |  |  |
| --- | --- | --- |
|  |  |  |

1. Uso della feature al nodo , con :

|  |  |  |
| --- | --- | --- |
|  |  |  |

1. Per un nodo interno , è usata esattamente una feature:

|  |  |  |
| --- | --- | --- |
|  |  |  |

1. Per un nodo foglia , non è usata alcuna feature:

|  |  |  |
| --- | --- | --- |
|  |  |  |

1. Sia , e sia il segno del letterale relativo alla feature per l’esempio . Per ogni nodo foglia :

|  |  |  |
| --- | --- | --- |
|  |  |  |

Ogni esempio positivo deve essere discriminato se la foglia è associata alla classe negativa.

1. Sia , e sia il segno del letterale relativo alla feature per l’esempio . Per ogni nodo foglia :

|  |  |  |
| --- | --- | --- |
|  |  |  |

Ogni esempio negativo deve essere discriminato se la foglia è associata alla classe positiva.

**Vincoli aggiuntivi**

Utilizziamo ulteriori vincoli per effettuare un *pruning* dello spazio di ricerca. L’idea è la seguente: durante la ricerca, viene costruita una struttura parziale dell’albero. Possiamo ridurre lo spazio di ricerca *potando* le estensioni non valide di questo albero parziale.

In particolare, per ogni , se la variabile assume valore *true*, vuol dire che il nodo corrispondente è una foglia, e possiamo ridurre il limite superiore sul numero dei figli sinistro e destro del nodo . Analogamente, se la variabile assume valore *false*, vuol dire che il nodo corrispondente è un nodo interno, e possiamo incrementare il limite inferiore sul numero dei figli sinistro e destro del nodo .

Es.: ipotizzando di cercare un albero con 7 nodi, vogliamo stabilire i possibili figli del nodo 3. In generale, questi possono essere i nodi 4 e 5 (rispettivamente sinistro e destro) o 6 e 7 (idem). Saranno il 4 e il 5 nel caso di un albero perfettamente bilanciato, il 6 e il 7 nel caso di un albero sbilanciato in cui ogni figlio sinistro è una foglia. Se durante la ricerca scopriamo che esiste un nodo foglia (in questo caso solo il nodo 2, perché la radice ovviamente non può essere una foglia), concludiamo che solo i nodi 4 e 5 possono essere figli del nodo 3.

Denotiamo con il numero di foglie fino al nodo compreso.

Fissato , è definito induttivamente come segue:

Analogamente, denotiamo con il numero di nodi interni fino al nodo compreso.

Fissato , è definito come:

Infine, per effettuare il pruning vero e proprio definiamo i seguenti vincoli: