Progetto per l'esame di Ingegneria della Conoscenza

Fabio Nardelli
mat. 655172
f.nardelli9@studenti.uniba.it

Indice

1 Introduzione e motivazioni

Ho scelto di implementare un articolo scientifico, "Learning Optimal Decision Trees with SAT" (Narodytska et al. 2018[1]) che riguarda l'apprendimento di alberi di decisione binari ottimali (nel numero di nodi) mediante soddisfacibilità booleana.
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 di mera curiosità, la query *"xai OR explainable ai OR explainable artificial intelligence"* su Google Scholar mostra circa 25000 risultati solo nel 2021, 17000 nel 2020, 9000 nel 2019 e 4500 nel 2018. Si tratta quindi di un trend in crescita.
Alberi di decisione di dimensione minima, oltre ad essere preferibili perché meno a rischio di *overfitting* – e, marginalmente, 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 sempre 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.

2 Prerequisiti

2.1 Il problema della classificazione binaria

Consideriamo un insieme di feature $\mathcal{F} = f_1, \ldots, f_k$, ciascuna della quali assume valore nell'insieme $\{0, 1\}$, e un insieme di esempi di training $\mathcal{E} = e_1, \ldots, e_M$ partizionato in $\mathcal{E}^+$ e $\mathcal{E}^-$, ossia tale per cui ciascun elemento e_i appartiene alla classe positiva o a quella negativa. Dal momento che le feature sono binarie, rappresentiamo un letterale associato alla feature r-*esima* con f_r se $f_r = 1$, o con $\neg f_r$ se $f_r = 0$.
Rappresentiamo un generico esempio $e_q \in \mathcal{E}$ con la coppia $(\mathcal{L}_q, c_q)$, dove $\mathcal{L}_q$ è l'insieme dei letterali associati all'esempio, e $c_q \in \{0, 1\}$ è la classe alla quale l'esempio appartiene. Si ha che $c_q = 1$ se $e_q \in \mathcal{E}^+ e c_q = 0$ se $e_q \in \mathcal{E}^-$.
Denotata con ϕ la funzione che associa a ciascun esempio la classe di appartenenza, il nostro obiettivo è quello di apprendere una funzione $\hat{\phi}$ che approssima ϕ 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 $\hat{\phi}$ significa apprendere un albero di decisione binario.

2.1.1 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 coerente con i dati di training, intendendo quello con il minor numero di nodi o la minore profondità. 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[1], 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*.

2.2 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 Y del CSP nella rappresentazione booleana si utilizzano tante variabili indicatrici quanti sono i

[1] Per 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.

valori v_i del dominio di Y, tali che $Y_i = true$ se $Y = v_i$. A ciascuna di queste variabili si assocerà un letterale y_i, per rappresentare il fatto che la variabile Y assume o non assume il valore corrispondente.
Ad esempio, dato $dom(Y) = \{v_1, \ldots, v_k\}$, lo convertiamo dapprima nelle k variabili indicatrici $\{Y_1, \ldots, Y_k\}$, e successivamente rappresentiamo queste ultime con gli atomi $y_i, \ldots, y_k$. Per "costringere" una variabile ad assumere esattamente un valore alla volta, aggiungiamo i vincoli $y_i \vee \ldots \vee y_k$ (la variabile deve assumere almeno un valore) e $\neg y_i \vee \neg y_j, \forall i, j \in \{1, \ldots, k\}, i < j$ (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 y, si può assegnare $true$ a Y).

3 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 N nodi e un generico nodo i, i suoi figli possono essere numerati nel range di numeri naturali da $i + 1$ a $min(2i + 1, N)$.
Di seguito elenchiamo i letterali 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.

3.1 Letterali

Dati N nodi, K feature binarie, e definiti, per $1 \leq i \leq N$, gli insiemi:

- $LR(i) = even([i + 1, min(2i, N - 1)])$
- $RR(i) = odd([i + 2, min(2i + 1, N)])$

introduciamo i letterali mostrati nella Tabella 1 (adottiamo i valori 1 e 0 in luogo di $true$ e $false$ per brevità, e per rispettare la notazione utilizzata dagli autori dell'articolo):

Tabella 1: Letterali utilizzati per la codifica del problema

Simbolo	Descrizione
Letterali per la rappresentazione di un albero binario completo	
v_i	1 sse il nodo i è una foglia, $i = 1, \ldots, N$
$l_{i,j}$	1 sse il nodo i ha il nodo j come figlio sinistro, $i = 1, \ldots, N, j \in LR(i)$
$r_{i,j}$	1 sse il nodo i ha il nodo j come figlio destro, $i = 1, \ldots, N, j \in RR(i)$
$p_{j,i}$	1 sse il nodo j ha come genitore il nodo i, $j = 2, \ldots, N, i = 1, \ldots, N-1$
Letterali per l'apprendimento di un albero di decisione	
$a_{r,j}$	1 sse la feature f_r è assegnata al nodo j, $r = 1, \ldots, K, j = 1, \ldots, N$
$u_{r,j}$	1 sse la feature f_r è discriminata dal nodo j, $r = 1, \ldots, K, j = 1, \ldots, N$
$d^0_{r,j}$	1 sse la feature f_r è discriminata per il valore 0 dal nodo j o da uno dei suoi antenati, $r = 1, \ldots, K, j = 1, \ldots, N$
$d^1_{r,j}$	1 sse la feature f_r è discriminata per il valore 1 dal nodo j o da uno dei suoi antenati, $r = 1, \ldots, K, j = 1, \ldots, N$
c_j	1 sse la classe del nodo foglia i è 1, $i = 1, \ldots, N$

3.2 Vincoli

3.2.1 Vincoli per la rappresentazione di un albero binario completo

La radice non deve essere una foglia:

$$\neg v_1 \tag{1}$$

Se un nodo è una foglia, non ha figli:

$$v_i \rightarrow \neg l_{i,j} \qquad j \in LR(i) \tag{2}$$

I figli sinistro e destro dell'i-esimo nodo sono numerati consecutivamente, rispettivamente:

$$l_{i,j} \leftrightarrow r_{i,j+1} \qquad j \in LR(i) \tag{3}$$

Un nodo non foglia deve avere un figlio:

$$\neg v_i \rightarrow \left(\sum_{j \in LR(i)} l_{i,j} = 1 \right) \tag{4}$$

Se l'i-esimo nodo è un genitore, deve avere un figlio sinistro e uno destro:

$$\begin{aligned} p_{j,i} &\leftrightarrow l_{i,j} \qquad j \in LR(i) \\ p_{j,i} &\leftrightarrow r_{i,j} \qquad j \in RR(i) \end{aligned} \tag{5}$$

Tutti i nodi tranne la radice devono avere un genitore:

$$\sum_{i=\lfloor \frac{j}{2} \rfloor}^{min(j-1,N)} p_{j,i} = 1 \qquad j = 2, \ldots, N \tag{6}$$

3.2.2 Vincoli per l'apprendimento di un albero di decisione

Per discriminare una feature per il valore 0 al nodo j:

$$d^0_{r,j} \leftrightarrow \left(\bigvee_{i=\lfloor \frac{j}{2} \rfloor}^{j-1} \left(\left(p_{j,i} \wedge d^0_{r,i} \right) \vee \left(a_{r,i} \wedge r_{i,j} \right) \right) \right) ; d^0_{r,1} = 0 \qquad j = 2, \ldots, N \tag{7}$$

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

$$d^1_{r,j} \leftrightarrow \left(\bigvee_{i=\lfloor \frac{j}{2} \rfloor}^{j-1} \left(\left(p_{j,i} \wedge d^1_{r,i} \right) \vee \left(a_{r,i} \wedge l_{i,j} \right) \right) \right); d^1_{r,1} = 0 \qquad j = 2, \ldots, N \tag{8}$$

Uso della feature r al nodo j, con $r = 1, \ldots, K,\ j = 1, \ldots, N$:

$$\begin{gathered} \bigwedge_{i=\lfloor \frac{j}{2} \rfloor}^{j-1} \left(u_{r,i} \wedge p_{j,i} \rightarrow \neg a_{r,j} \right) \\ u_{r,j} \leftrightarrow \left(a_{r,j} \vee \bigvee_{i=\lfloor \frac{j}{2} \rfloor}^{j-1} \left(u_{r,j} \wedge p_{j,i} \right) \right) \end{gathered} \tag{9}$$

Per un nodo interno j, è usata esattamente una feature:

$$\neg v_j \rightarrow \sum_{r=1}^{k} a_{r,j} = 1 \qquad j = 1, \ldots, N \tag{10}$$

Per un nodo foglia j, non è usata alcuna feature:

$$v_j \rightarrow \sum_{r=1}^{k} a_{r,j} = 0 \qquad j = 1, \ldots, N \tag{11}$$

Sia $e_q \in \mathcal{E}^+$, e sia $\sigma(r,q) \in \{0,1\}$ il segno del letterale relativo alla feature f_r per l'esempio e_q. Per ogni nodo foglia j:

$$v_j \wedge \neg c_j \rightarrow \bigvee_{r=1}^{K} d^{\sigma(r,q)}_{r,j} \qquad j = 1, \ldots, N \tag{12}$$

ovvero, ogni esempio positivo deve essere discriminato se la foglia j è associata alla classe negativa.

Sia $e_q \in \mathcal{E}^-$, e sia $\sigma(r,q) \in \{0,1\}$ il segno del letterale relativo alla feature f_r per l'esempio e_q. Per ogni nodo foglia j:

$$v_j \wedge c_j \rightarrow \bigvee_{r=1}^{K} d^{\sigma(r,q)}_{r,j} \qquad j = 1, \ldots, N \tag{13}$$

ovvero, ogni esempio negativo deve essere discriminato se la foglia j è associata alla classe positiva.

3.2.3 Vincoli aggiuntivi

Definiamo 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 $k < i$, se il letterale v_k 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 i. Analogamente, se il letterale v_k 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 i.
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 $k < 3$ (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 $\lambda_{t,i}$ il numero di foglie fino al nodo i compreso.
Fissato $0 \leq t \leq \lfloor \frac{i}{2} \rfloor$, $\lambda_{t,i}$ è definito induttivamente come segue:

1. $\lambda_{0,i} = 1, \qquad 1 \leq i \leq N$
2. $\lambda_{t,i} \leftrightarrow (\lambda_{t,i-1} \vee \lambda_{t-1,i-1} \wedge v_i), \qquad i = 1, \ldots, N, t = 1, \ldots, \lfloor \frac{i}{2} \rfloor$

Analogamente, denotiamo con $\tau_{t,i}$ il numero di nodi interni fino al nodo i compreso.
Fissato $0 \leq t \leq i$,$\tau_{t,i}$ è definito come:

1. $\tau_{0,i} = 1, \qquad 1 \leq i \leq N$
2. $\tau_{t,i} \leftrightarrow (\tau_{t,i-1} \vee \tau_{t-1,i-1} \wedge \neg v_i), \qquad i = 1, \ldots, N, t = 1, \ldots, i$

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

$$\lambda_{t,i} = 1 \rightarrow l_{i,2(i-t+1)} = 0 \wedge r_{i,2(i-t+1)+1} = 0, \qquad 0 < t \leq \left\lfloor \frac{i}{2} \right\rfloor.$$

$$\tau_{t,i} = 1 \rightarrow l_{i,2(t-1)} = 0 \wedge r_{i,2t-1} = 0, \qquad \left\lceil \frac{i}{2} \right\rceil < t \leq i$$

Oltre a questi vincoli, presenti nell'articolo, ho formulato altri tre vincoli, sempre con lo scopo di ridurre lo spazio di ricerca. La numerazione segue quella dei vincoli originali così come presentati nell'articolo, e rispetta la divisione tra vincoli adibiti alla rappresentazione di alberi binari completi (6.1 e 6.2), e vincoli per l'addestramento (13.1).

Ogni figlio sinistro/destro deve avere esattamente un genitore:

$$\begin{aligned} l_{i,j} &\rightarrow \sum_{i=1}^{N} p_{j,i} = 1 \qquad j = 2, \ldots, N \\ r_{i,j} &\rightarrow \sum_{i=1}^{N} p_{j,i} = 1 \qquad j = 2, \ldots, N \end{aligned} \tag{6.1}$$

Nodi sullo stesso livello devono essere numerati consecutivamente:

$$\begin{aligned} l_{i,j} &\rightarrow l_{h,j-2} \qquad i = 1, \ldots, N,\ j = 2, \ldots, N,\ h < i \\ r_{i,j} &\rightarrow r_{h,j-2} \qquad i = 1, \ldots, N,\ j = 2, \ldots, N,\ h < i \end{aligned} \tag{6.2}$$

Solo un nodo foglia può essere associato a una classe:

$$c_i \rightarrow v_i \qquad i = 1, \ldots, N \tag{13.1}$$

In particolare, la necessità di aggiungere i vincoli 6.1 e 6.2 è scaturita dal fatto che, in una prima implementazione del programma, quando avevo inserito solo i primi sei vincoli (dediti alla sola rappresentazione di alberi binari completi), ho constatato che venivano generati anche alberi non validi (in particolare con più genitori per uno stesso nodo) oppure doppioni di uno stesso albero (ovvero, alberi con la stessa topologia ma nodi numerati diversamente, violando la regola secondo cui i nodi devono essere numerati consecutivamente da sinistra a destra). Questi problemi sparivano una volta aggiunti i vincoli per l'apprendimento ma, testando il sistema, una volta completato, ho verificato che i vincoli da me aggiunti portano a un miglioramento delle prestazioni, perché di fatto realizzano un ulteriore pruning dello spazio di ricerca.

4 Implementazione

Per l'implementazione ho utilizzato il linguaggio Python (è necessaria almeno la versione 3.7 perché ho utilizzato il costrutto dataclass per rappresentare i nodi dell'albero). Per semplicità, anziché utilizzare un SAT solver puro, che avrebbe richiesto di convertire i vincoli in CNF, ho scelto di utilizzare un SMT (Satisfiability Modulo Theory) solver. Si tratta di strumenti atti alla risoluzione di CSP che ammettono un linguaggio più espressivo rispetto ai SAT solver puri. Ad esempio, è possibile rappresentare vincoli di cardinalità o quantificatori (sfociando nel prim'ordine). In particolare, ho utilizzato e confrontato tre diversi SMT Solver *state-of-the-art*: Google OR-Tools[2], Microsoft Z3[3] e Yices 2.0[4].

4.1 Struttura del sistema

Il sistema si compone dei seguenti moduli:

- Il modulo `core` contiene le implementazioni del csp con OR-Tools, Z3 e Yices
- Il modulo `decisiontree` contiene l'implementazione dell'albero di decisione con i relativi metodi `fit` e `fit_optimal`(per l'addestramento) e `predict` (per classificare un esempio). Il metodo `fit` costruisce un albero di decisione con un determinato numero di nodi, passatogli in input; il metodo `fit_optimal` costruisce un albero di decisione con il minor numero di nodi possibile.
- Il modulo `utils` contiene una helper function per la conversione di un generico dataset a feature discrete in binario con la tecnica one-hot-encoding, e la classe `ResultSet` e il metodo `get_mean_scores` per il calcolo delle metriche usate per la valutazione del sistema
- Il modulo `preprocess` consente di caricare un dataset e di convertirlo in binario, utilizzando la helper function del modulo precedente

[2] `https://developers.google.com/optimization`
[3] `https://github.com/Z3Prover/z3`
[4] `https://yices.csl.sri.com/`

4.2 Requisiti e dipendenze

Per utilizzare il sistema occorre installare i seguenti software:

- Python 3.7+
- scikit learn[5], per effettuare il campionamento e lo splitting dei dataset in training e test set, nonché per calcolare le metriche di valutazione
- Pandas[6] e numpy[7] per caricare e manipolare i dataset
- I già citati OR-Tools, Z3 e Yices 2.0 con il relativo binding[8] per Python per l'implementazione e risoluzione del csp

5 Test

5.1 Condizioni di test

Ho testato il sistema sul dataset "Car" di UCI[9], utilizzato anche dagli autori dell'articolo, previa sua conversione in binario attraverso la nota tecnica *one-hot-encoding*, che prevede l'aggiunta di una feature binaria per ogni elemento del dominio di una data feature del dataset originale. Il dataset presenta feature target con quattro valori ("unacc", "acc", "good" e "vgood"); "unacc" compare il 70% delle volte. Dato che il sistema funziona solo con classi binarie, ho attribuito al valore "unacc" la classe negativa e agli altri tre quella positiva. Ho constatato che lo stesso è stato fatto dagli autori dell'articolo originale, e da Bessiere et al., autori di un precedente lavoro[2]. A tal proposito, ringrazio il dr. Emmanuel Hebrard, coautore di quest'ultimo articolo, il quale mi ha fornito alcuni dataset utilizzati nei due articoli già convertiti in binario, così ho avuto modo di sincerarmi della correttezza del mio encoding.
Il dataset consta di 1729 esempi, ma per effettuare il training ho potuto utilizzare solo una porzione molto piccola (variabile a seconda del campione scelto), in genere non contenente più di 20-30 esempi, a causa della complessità del problema della soddisfacibilità booleana (NP-completo). Per cercare di ovviare al problema, e ottenere dei risultati significativi (e non troppo legati al particolare campione scelto), ho effettuato 40 campionamenti diversi, ciascuno costituito da 20 esempi, avvalendomi della funzione `train_test_split` di sklearn con altrettanti seed diversi scelti casualmente, utilizzando il resto del dataset come test set.
Tutti i test di seguito descritti sono stati effettuati su una macchina con processore AMD Ryzen 5 2400G, 16 GB di ram e sistema operativo Fedora Linux 32.
I tempi si riferiscono all'addestramento con la funzione `fit_optimal`, che cerca un albero di decisione a partire da 3 nodi (il minimo possibile), e incrementa questo valore di due in caso di fallimento, fino a trovare una soluzione (in caso di successo).

[5] `https://scikit-learn.org/stable/`
[6] `https://pandas.pydata.org/`
[7] `https://numpy.org/`
[8] `https://github.com/SRI-CSL/yices2_python_bindings`
[9] `https://archive.ics.uci.edu/ml/datasets.php`

Tabella 2: Metriche utilizzate per la valutazione

Nome	Formula	Descrizione
Precision	$\frac{tp}{tp+fp}$	Esempi realmente positivi sul totale di quegli classificati come positivi
Recall	$\frac{tp}{tp+fn}$	Esempi realmente positivi sul totale di quegli positivi (classificati e non)
Average Precision	$\sum_{k=1}^{n} P_k \left(R_k - R_{k-1}\right)$	Media pesata della precision (P) a ogni livello n, dove n è il numero di predizioni effettuate, usando come peso l'incremento di recall (R) dal livello precedente
F1	$\frac{2\times P\times R}{P+R} = \frac{2tp}{2tp+fp+fn}$	Media armonica di precision (P) e recall (R)
Accuracy	$\frac{tp+tn}{tp+tn+fp+fn}$	È la frazione di predizioni corrette sul totale delle predizioni. Non tiene conto della distribuzione delle classi nel dataset.
MCC (Matthews Corr. Coeff.)	$\frac{tp\times tn - fp\times fn}{\sqrt{(tp+fp)(tp+fn)(tn+fp)(tn+fn)}}$	A differenza dell'accuracy e della F1, tiene conto del bilanciamento di ciascuna classe della matrice di confusione (tp, fp, tn e fn), pertanto è indicativo delle performance del classificatore sia sulla classe negativa che su quella positiva.

5.2 Confronto tra le implementazioni

Ho confrontato i tempi di esecuzione di ciascuna implementazione sul dataset "Car", senza utilizzare i vincoli aggiuntivi (no pruning dello spazio di ricerca). Nella tabella sono mostrati i tempi medi di addestramento su venti esecuzioni, campionando il dataset come descritto nel paragrafo precedente.

Tabella 3: Tempi di apprendimento sul dataset "Car" con le diverse implementazioni

Yices 2.0	OR-Tools	Z3
2.13	4.89	11.56

L'implementazione che utilizza Yices risulta essere nettamente superiore, perciò i risultati mostrati successivamente si riferiranno tutti a questa.

5.3 Pruning dello spazio di ricerca

Ho confrontato tutte le possibili combinazioni di vincoli aggiuntivi (descritti in precedenza), sia quelli ideati dagli autori dell'articolo (oc nella tabella), che quelli formulati da me (6.1, 6.2 e 13.1). Il confronto è stato effettuato sempre sul dataset "Car".
Come si evince dalla tabella 4, con i vincoli aggiuntivi si riesce ad ottenere un miglioramento di oltre il 50% dei tempi di addestramento.

Tabella 4: Tempi di addestramento sul dataset "Car" con tutte le combinazioni di vincoli aggiuntivi

6.1	6.2	6.1 e 6.2	13.1	6.1 e 13.1	6.2 e 13.1	6.1, 6.2 e 13.1
2.11	1.04	0.99	2.14	1.96	0.98	1.03
oc	6.1 e oc	6.2 e oc	13.1 e oc	6.1, 13.1 e oc	6.2, 13.1 e oc	tutti
1.00	1.02	1.07	1.02	1.20	0.99	1.05

5.4 Risultati

Nella tabella 5 sono mostrati i risultati del test sul dataset “Car”, confrontati con il classificatore DecisionTreeClassifier di sklearn. I valori riportati sono la media di tutti i campionamenti effettuati.
Nelle altre tabelle, sono mostrati i risultati su altri due dataset.

Tabella 5: Media delle metriche misurate su 40 campionamenti casuali del dataset "Car"

	dt	sklearn dt
Nodi	10.65	9.05
Tempo (s)	0.54	0.00
Precision	0.68	0.61
Recall	0.72	0.60
Avg. Precision	0.59	0.50
F1	0.69	0.58
Accuracy	0.81	0.75
MCC	0.57	0.43

Tabella 6: Media delle metriche misurate su 40 campionamenti casuali del dataset "Breast Cancer", 50 esempi di training

	dt	sklearn dt
Nodi	8.95	8.7
Tempo (s)	9.64	0.00
Precision	0.86	0.85
Recall	0.82	0.85
Avg. Precision	0.76	0.78
F1	0.83	0.85
Accuracy	0.89	0.79
MCC	0.75	0.77

Tabella 7: Media delle metriche misurate su 40 campionamenti casuali del dataset "Heart Cleveland", 15 esempi di training

	dt	sklearn dt
Nodi	6.3	6.4
Tempo (s)	1.96	0.00
Precision	0.68	0.61
Recall	0.73	0.70
Avg. Precision	0.65	0.66
F1	0.70	0.69
Accuracy	0.67	0.67
MCC	0.35	0.36

6 Conclusioni

Benché l'apprendimento di alberi di decisione mediante soddisfacibilità booleana sia computazionalmente molto meno efficiente rispetto agli algoritmi tradizionalmente impiegati, tanto da risultare spesso intrattabile, questo esperimento mostra come, su piccoli dataset, si possano ottenere, a parità di condizioni di test, prestazioni predittive pari alle soluzioni *state-of-the-art*, se non addirittura superiori.

Riferimenti bibliografici

[1] Nina Narodytska, Alexey Ignatiev, Filipe Pereira, Joao Marques-Silva. *Learning Optimal Decision Trees with SAT.*
Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, pag. 1362-1368, 2018 International Joint Conferences on Artificial Intelligence Organization
`https://doi.org/10.24963/ijcai.2018/189`

[2] Christian Bessiere, Emmanuel Hebrard, Barry O'Sullivan. *Minimising decision tree size as combinatorial optimization.*
International Conference on Principles and Practice of Constraint Programming, pag. 173-187. Springer 2009.