# Alberi decisionali

Iacopo Erpichini

28 Marzo 2018

**Abstract**

In questo articolo analizzo il funzionamento del mio programma che sviluppa il codice per l'apprendimento di alberi di decisione, utilizzando l'entropia come misura di impurità. Si implementa quindi una semplice strategia di pruning sulle regole corrispondenti all'albero (DNF dei cammini dalla radice alle foglie) basata sull'errore sul validation set.

**Keywords** — Alberi decisionali, pruning, validation set

## 1 Introduzione teorica

Nel *Machine Learning* uno dei metodi di apprendimento conosciuti è quello degli alberi di decisione[1][2]. Esso consiste nel costruire un albero di decisione a partire da un insieme di dati iniziali (dataset: nel mio caso reperiti da MlData), che verrà poi usato poi come strumento di predizione per dati futuri.

Questo tipo di struttura presenta però un problema, infatti essa aumenta esponenzialmente con l'aumento degli attributi e dei relativi possibili valori; si cerca allora un modo per minimizzare il numero di nodi creati durante la costruzione dell'albero, in modo da avere più velocità e minor spazio occupato.

Si introducono allora due concetti:

- il **costo** è una misura dell'impurità del dataset (rispetto all'attributo target, il quale rappresenta una certa classe che vogliamo studiare e predire)
- il **gain** invece è una misura che indica l'attributo migliore da cui continuare (o iniziare) la costruzione dell'albero. Esso, dato un dataset D ed un certo attributo A è defnito come segue:

$$Gain\ (S, A) = Entropy\ (S) - \sum_{v \in A} \frac{|S_v|}{|S|} Entropy\ (S_v)$$

Figure 1: Formula guadagno

Il valore del gain cambia a seconda della funzione che viene utilizzata, nel caso specifico dell'elaborato viene utilizzato l' algoritmo id3[3] per la creazione

di alberi di decisione che utilizza l'entropia come misura di impurità, con una piccola variazione che mi serve per eseguire il pruning.

$$Entropy(S) = \sum_{i=1}^{C} p_i \log_2 p_i$$

Figure 2: Formula entropia

Un' altro aspetto analizzato sono i problemi relativi all' **overfitting**[2] e all' underfitting di un albero. Di solito un algoritmo di apprendimento viene allenato usando un certo insieme di esempi (il training set appunto), ad esempio situazioni tipo di cui è già noto il risultato che interessa prevedere (output). Si assume che l'algoritmo di apprendimento (il learner) raggiunga uno stato in cui sarà in grado di predire gli output per tutti gli altri esempi che ancora non ha visionato, cioè si assume che il modello di apprendimento sia in grado di generalizzare. Tuttavia, soprattutto nei casi in cui l'apprendimento è stato effettuato troppo a lungo o dove c'era uno scarso numero di esempi di allenamento, il modello potrebbe adattarsi a caratteristiche che sono specifiche solo del training set, ma che non hanno riscontro nel resto dei casi; perciò, in presenza di overfitting, le prestazioni (cioè la capacità di adattarsi/prevedere) sui dati di allenamento aumenteranno, mentre le prestazioni sui dati non visionati saranno peggiori.

Per prevenire ed evitare l'overfitting è necessario mettere in atto particolari accorgimenti tecnici, come la tecninca **validation-set**[2][4] e l'arresto anticipato, che indicano quando un ulteriore allenamento non porterebbe a una migliore generalizzazione

# 2 Implementazione

Si implementa quanto descritto mediante il linguaggio di programmazione Python, versione 2.7.

Il *codice è molto commentato*, per renderlo più leggibile ad un possibile lettore. A seguito è riportata una panoramica delle classi presenti nel progetto e le principali funzionalità di cui si occupano. Ogni riga sarà quindi un dizionario dove ad ogni attributo sarà associato un certo valore

- Nel file **importa_file.py** è presente una funzione che si occupa semplicemente di leggere un file **.csv***(comma separated values)* ritornando un dataset, una lista di attributi e l'attributo target.
- Nel file **grafico.py** sono presenti due semplici funzioni, che fanno uso della libreria *matplotlib.pyplot* per stampare delle curve di apprendimento degli alberi da me presi in analisi.
- Nel file **albero_di_decisione.py** troviamo le funzioni *entropy & gain*. Tutte fanno uso di una importante sotto funzione che ritorna un dizionario

dove ad ogni possibile valore di un certo attributo all'interno del dataset è associato il numero di volte che esso compare. Questa informazione infatti è fondamentale per il calcolo degli indici. La funzione **create_decision_tree** si occupa di creare l'albero, utilizzando id3, inoltre si occupa anche dell' operazione di pre-pruning.

- Nel file **validation_set.py** è presente la funzione principale che si occupa di eseguire il test a seconda dei parametri in ingresso esegue un test dividendo il dataset in due parti, in base a una percentuale e esegue un numero di test multipli per avere delle accuratezze migliori. Le due parti sono train set e test set, il train set è diviso a sua volta in due parti (con un euristica 80%train:20%validation che è un buon compromesso) di cui una si occupa del validation set nel caso del pruning per determinare la profondità migliore per costruire l'albero prunato.

  Il pruning implementato per questi algoritmi è un pre-pruning che si occupa solo di arrestarsi a determinate profondità(determinate dal validation-set), per i miei test non era molto rilevante applicare un pruning anche al numero minimo delle foglie campione dell' albero e quindi ho lasciato questo valore a 1.

  Alla fine vengono ritornati i valori relativi a un dataset delle accuratezze di predizione sul trainset(train+validation) e testset di una colonna target specificata dalla funzione.

- Nel file **main.py** è presente una funzione che si occupa di lanciare i test eseguiti, di raccogliere i tempi e di stampare dei grafici per analizzarli in seguito.

# 3 Esperimento

In pratica viene eseguito l' algoritmo validation-set su un insieme di percentuali e viene calcolato lo score di predizione su una data colonna target adottando o no il pruning per confrontare il grado di accuratezza su un test set relativo a ogni dataset.

## 3.1 Analisi dei risultati

I risultati sono espressi sotto forma di tre grafici dove: Il primo si occupa di mostrare le curve di accuratezza del dataset sul trainset e sul test set senza aver applicato il validation set per prunare l'albero, il secondo è speculare al primo solamente che applica il validation set quindi il pruning. Il terzo grafico invece mostra il confronto delle accuratezze dei due metodi sul test set relativo.

Le curve di tutti i dataset sono presenti qui.

- Nel dataset **votes.csv** si nota che passiamo da un overfitting molto alto nel grafico senza pruning rispetto ad uno con un buon scarto quando viene applicato il pruning(poco sopra l'1%), soprattutto vediamo che l'accuratezza

sul testset quando viene effettuato il pruning migliora di molti punti percentuali.

- Nel dataset **carclassifier.csv** si nota che all'aumentare della percentuale di train aumentano il numero di esempi che hanno le stesse righe con target diverso e questo porta ad avere un decadimento dell' accuratezza, con il pruning questo problema viene evitato parzialmente e invece che diminure riesce a migliorare.

- Nel dataset **contraceptive.csv** si vedono meno i miglioramenti rispetto agli altri due esempi ma si può dire che il margine di miglioramento delle predizioni sul test set utilizzando il pruning aumenti di più rispetto a quella senza(data la pendenza della retta), potremmo anche ipotizzare che aumentando il numero di esempi nel dataset migliori. Infatti quando le percentuali di test sono al 90% ä parità di accuratezzac̈on il pruning abbiamo un classificatore più semplice.

```
Analisi dataset:"votes.csv", Percentuale training:90.0% con numero di test per percentuale = 300
Tempi esecuzione senza pruning:['2.1519', '3.8732', '5.4754', '7.6646', '8.7199', '10.5842', '12.1604', '14.0947', '15.7796']
Tempi esecuzione con pruning:['2.0458', '4.3433', '6.5109', '8.4020', '12.2853', '15.7426', '18.4060', '23.2005', '26.7779']
Percentuali di apprendimento:[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
Accuratezza senza pruning:['0.51816327', '0.52356322', '0.52899454', '0.52816092', '0.52902141', '0.53195402', '0.52867684', '0.52904215', '0.53348485']
Accuratezza con pruning:['0.61000850', '0.61486590', '0.61621858', '0.61975734', '0.61798165', '0.61750958', '0.61773537', '0.62233716', '0.61363636']

Analisi dataset:"car-classifier.csv", Percentuale training:90.0% con numero di test per percentuale = 180
Tempi esecuzione senza pruning:['3.0551', '4.2291', '5.5316', '6.8419', '8.1906', '9.0236', '10.3879', '11.5584', '12.7520']
Tempi esecuzione con pruning:['7.5499', '16.9650', '27.7899', '38.0838', '49.9976', '61.0415', '74.2290', '84.2654', '95.3418']
Percentuali di apprendimento:[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
Accuratezza senza pruning:['0.44321265', '0.42454808', '0.40184573', '0.37773492', '0.35473894', '0.33144669', '0.31384072', '0.29725434', '0.28461785']
Accuratezza con pruning:['0.48526492', '0.50095204', '0.51080808', '0.51344691', '0.51628729', '0.51540623', '0.51751231', '0.51332691', '0.50622993']

Analisi dataset:"contraceptive.csv", Percentuale training:90.0% con numero di test per percentuale = 220
Tempi esecuzione senza pruning:['3.1337', '4.9690', '6.9142', '9.1897', '11.2986', '13.3977', '15.5344', '17.6871', '20.2447']
Tempi esecuzione con pruning:['8.6409', '15.1207', '24.0681', '37.1475', '45.5944', '71.7699', '88.8257', '103.8849', '135.8058']
Percentuali di apprendimento:[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
Accuratezza senza pruning:['0.40050391', '0.42012491', '0.43707717', '0.44665261', '0.45397188', '0.45872881', '0.46017071', '0.46275809', '0.46624693']
Accuratezza con pruning:['0.40948512', '0.41914566', '0.42651515', '0.43445084', '0.44378315', '0.45212635', '0.45804196', '0.46311248', '0.46670762']
```

Figure 3: Risultati relativi ai dataset

In questa foto ci sono alcuni dei risultati rilevati dal mio esperimento. i tempi relativi alle percentuali applicando il pruning sono maggiori perchè viene contato anche il tempo in cui l'insieme del validation-set determina l'altezza dell'albero, altrimenti con un altro tipo di test avremmo visto che questi tempi sarebbero stati simili sul calcolo delle accuratezze su un classificatore con o senza pruning.

## 3.2 Utilizzo codice

La **guida** di installazione relativa al mio progetto è presente nel file ReadMe associato al progetto github.

# 4 Riferimenti

[1] Intelligenza Artificiale, Russel & Norving 2009 cap. 18.3
[2] Mitchell, cap 3 ID3 pp. 56, Overfitting & Pruning pp. 66-71
[3] Articolo id3 O'Reilly
[4] Slide alberi di decisione Slide 34