# Relazione per l'esame finale
# Intelligenza Artificiale (2016/2017)

Tommaso Scarlatti
5784154

23 January 2017

## 1 Introduzione

Gli alberi di decisione sono una delle più semplici ed efficaci forme di machine learning e sono collocati nella categoria del cosiddetto "supervised learning". Quest'ultimo è caratterizzato da un agente, il quale osserva una serie di coppie input-output, e produce una funzione che mappa dai primi ai secondi. Nel nostro caso, il problema dell'apprendimento si definisce "classificazione", in quanto i possibili valori degli output sono in numero finito.
Un albero di decisione è quindi una funzione che prende in input un insieme di attributi e restituisce una "decisione", ossia un singolo valore di output. Ogni nodo interno dell'albero corrisponde a testare un particolare attributo, mentre ogni ramo rappresenta un valore assegnato all'attributo del nodo da cui esce. Le foglie rappresentano invece i possibili valori di ritorno della funzione.

## 2 L'algoritmo

### 2.1 Implementazione

L'algoritmo per l'apprendimento di alberi di decisione utilizzato è quello descritto in R&N 2009 §18.3, ed è stato implementato in linguaggio python. Le funzioni per stampare l'albero a video, le strutture dati per memorizzare il dataset e l'organizzazione delle classi sono state riprese dalla aima-python repository su github, il resto è stato sviluppato autonomamente.
Sono state create tre classi: *DataSet*, che rappresenta un insieme di dati con i relativi parametri, *DecisionTree* e *DecisionLeaf*, che rappresentano rispettivamente un albero (o un sottoalbero) e una foglia del decision tree.

### 2.2 Come funziona

L'algoritmo adotta una strategia "divide-et-impera" selezionando per il test l'attributo più "importante" da una lista di disponibili. Il test divide il problema in due sottoproblemi che verranno risolti in modo ricorsivo. Quindi l'albero finale viene costruito in modo bottom-up, partendo dalle foglie, che saranno generate nel passo base dell'algoritmo al verificarsi di una delle seguenti condizioni:

- Tutti gli esempi hanno la stessa classificazione
- Non sono rimasti più esempi
- Non sono rimasti più attributi
- Strategia di pre-pruning

In tutti i casi, fatta eccezione per il primo, si fa ricorso ad una funzione *majority_value* che permette di determinare, tra un insieme di valori, quello da assegnare alla foglia, ossia quello che compare più volte negli esempi (passati come parametro). La funzione *decision_tree_learner*, che al suo interno contiene l'algoritmo ricorsivo *decision_tree_learning*, restituirà l'albero di decisione completo e il numero di nodi interni dell'albero, quantità indicativa della complessità di quest'ultimo.

### 2.3 Entropia e information gain

Che cosa si intende per "importanza" nella selezione di un attributo per il test? Si intende l'attributo che ha il maggiore information gain, un concetto definito in termini di entropia. L'entropia è la misura dell'incertezza di una variaible aleatoria, e per una variabile $V$ con valori $v_k$ ognuno con probabilità $P(v_k)$ è definita come segue:

$$H(V) = -\sum_k P(v_k) \log_2 P(v_k)$$

L'information gain invece è la riduzione di entropia che ci si aspetta avvenga scegliendo un determinato attributo da testare. Quindi ragionevolmente scegliamo l'attributo che ci permette di guadagnare la massima informazione possibile sul target. Formalmente: sia $T$ un insieme di esempi, oguno della forma $(x, y) = (x_1, x_2, \dots x_k, y)$ ove $x_a \in vals(a)$ è il valore dell'$a$-esimo attributo e $y$ è l'attributo target, l'information gain è definito come segue:

$$Gain(T, a) = H(T) - \sum_{v \in vals(a)} \frac{|\{x \in T | x_a = v\}|}{|T|} \cdot H(\{x \in T | x_a = v\})$$

### 2.4 Pruning

Dato un set di esempi, utilizziamo una sola parte di essi per la costruzione dell'albero di decisione (nel mio caso l'80%), mentre destiniamo la restante parte a testare la capacità predittiva di quest'ultimo. Questa particolare suddivisione genera due insiemi di esempi distinti, chiamati rispettivamente training set e test set. Nei casi in cui l'apprendimento è stato effettuato troppo a lungo, o dove c'era uno scarso numero di esempi di allenamento, possono verificarsi degli inconvenienti. Il modello, infatti, potrebbe adattarsi a caratteristiche che sono specifiche solo del training set, ma che non hanno riscontro nel resto dei casi; perciò, le prestazioni (cioè la capacità di adattarsi/prevedere) sui dati di allenamento aumenteranno, mentre le prestazioni sui dati non visionati saranno peggiori. Questo problema è noto con il nome di **overfitting**. L'overfitting è quindi un "eccessivo adattamento" del classificatore al training set. Per evitarlo, vi sono due strategie principali:

- Pre-pruning: fermare la crescita dell'albero durante la costruzione
- Post-pruning: potare l'albero dopo averlo completamente costruito

Nel mio programma ho implementato la prima soluzione. L'algoritmo *decision_tree_learning*, all'interno del file "decisiontreelearner.py" prende in input un parametro intero $m$, che ad ogni iterazione ricorsiva verrà confrontato con il numero di erorri che si ottengono facendo collassare quel particolare nodo in una foglia. La foglia ottiene il valore più comune dell'attributo target su quel determinato set di esempi. Le considerazioni fatte sopra sono state implementate in python nel seguente modo:

```
if misclass_error(examples) < m:
        return majority_value(examples)
```

Dove la funzione *misclass_error* conta il numero di esempi classificati male dall'operazione di pruning del nodo. Ovviamente ci aspettiamo una complessità dell'albero che cresce al diminuire del parametro $m$.

## 3 Testing

### 3.1 Data set utilizzati

Per testare l'algoritmo sono stati utilizzati tre data set presi da UCI Machine Learning Repository. Per praticità d'uso i file .data sono stati convertiti in file .txt. I data set scelti hanno dimensioni distinte, per evidenziare le differenti capacità predittive dell'algoritmo di fronte ad un numero crescente di esempi disponibili. Il data set *balancescale* ha 675 istanze, *carevaluation* ne ha 1728 mentre *nursery* conta ben 12960 esempi. Per informazioni dettagliate sui data set, sugli attributi e sui valori, si rimanda al file "info.txt" presente nella cartella "Project".

## 3.2 Tecnica di testing

La tecnica utilizzata per testare i tre data set è implementata all'interno del file "test.py". La funzione $test\_and\_plot$ costruisce un albero per ogni valore del parametro $m$, il quale varia in un range che dipende a sua volta dalle specifiche del data set. Per ogni albero vengono memorizzati in tre liste distinte il numero di nodi interni, gli erorri sul training set e quelli sul test set. Queste operazioni vengono ripetute per un totale di 10 volte. Ad ogni ciclo si permutano in modo casuale gli esempi del data set tramite l'utilizzo della funzione $random.shuffle()$ della libreria $math$, utilizzando come seme la data corrente. Successivamente viene fatto lo split degli esempi in training set (80%) e test set (20%). Questo rende l'albero di decisione indipendente da qualsiasi particolare suddivisione degli esempi. Al termine delle 10 iterazioni, viene fatta la media sui risultati ottenuti e viene stampato a video un grafico che mostra due curve: una rappresenta la percentuale di errori sul training set in funzione del numero dei nodi interni, l'altra quella sul test set, sempre in funzione del numero di nodi interni.

## 3.3 Risultati ottenuti

In questa sezione vengono mostrati i risultati ottenuti applicando l'algoritmo ai tre data set scelti. Per ogni data set è stato scelto un valore massimo del parametro $m$ tale che fosse possibile raggiungere la configurazione in cui tutto l'albero è collassato in una singola foglia, massimizzando la percentuale di errori (es. 500 esempi con target booleano: 300 positivi e 200 negativi, $m\_range = 200$). Fa eccezione il data set $nursery$, che a causa della sua grande mole non permetteva una computazione in tempi utili con un valore massimo del parametro troppo elevato, per questo è stato scelto $m\_range = 1000$.

### 3.3.1 Car evaluation

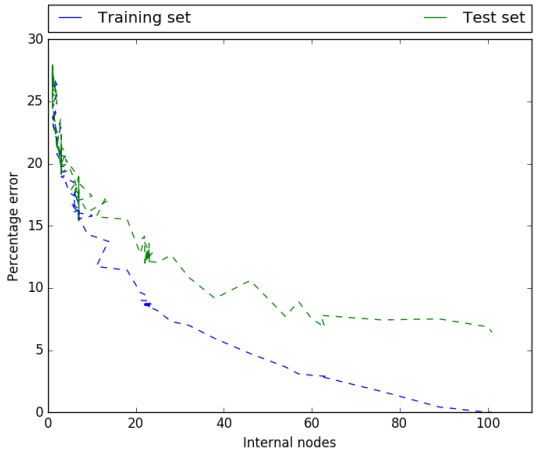


Figure 1: Curve degli errori sul data set car evaluation

| Target | N | N(%) |
|---|---|---|
| unacc | 1210 | 70.023% |
| acc | 384 | 22.222% |
| good | 69 | 3.993% |
| v-good | 65 | 3.762% |

Table 1: Classificazione in base all'attributo target di car evaluation

### 3.3.2 Balance-scale

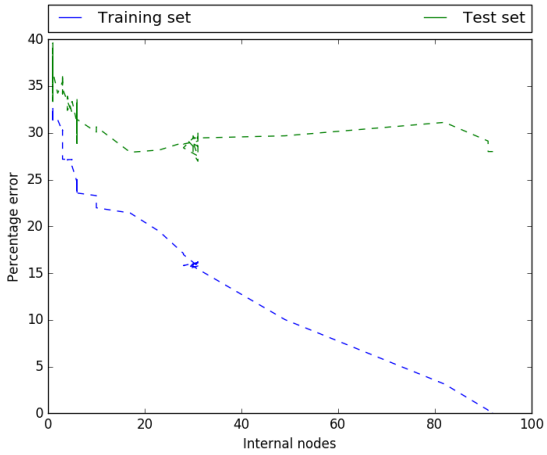


Figure 2: Curve degli errori sul data set balance-scale

| Target | N | N(%) |
|---|---|---|
| L | 288 | 46.08% |
| B | 49 | 07.84% |
| R | 288 | 46.08% |

Table 2: Classificazione in base all'attributo target di balance-scale

### 3.3.3 Nursery

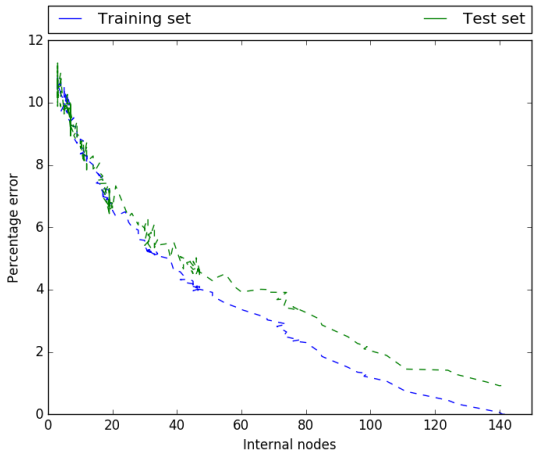


Figure 3: Curve degli errori sul data set nursery

| Target | N | N(%) |
|---|---|---|
| not_recom | 4320 | 33.333% |
| recommend | 2 | 0.015% |
| very_recom | 328 | 2.531% |
| priority | 4266 | 32.917% |
| spec_prior | 4044 | 31.204% |

Table 3: Classificazione in base all'attributo target di nursery