Corso di Laurea Triennale in Ingegneria Informatica

Elaborato di Intelligenza Artificiale

ANDREA PUCCIA

Anno Accademico 2018-2019

Testo del Problema

Potatura di alberi di decisione
Nella prima parte di questo elaborato si sviluppa del codice (in un linguaggio di programmazione a scelta) per l'apprendimento di alberi di decisione come esposto in classe e descritto in R&N 2009 §18.3, utilizzando l'entropia come misura di impurità. Si implementa quindi una semplice strategia di pruning basata sull'errore sul validation set (si veda Mitchell 1997, cap. 3).
Nella seconda parte, si applica il codice per l'apprendimento di alberi di decisione ad almeno tre data sets scelti a piacere dal repository MLData, confrontando i risultati ottenuti prima e dopo il pruning.

Implementazione

L'implementazione si articola in 6 file .py ognuno dei quali implementa funzioni diverse.

1. Node.py
2. Tree.py
3. Learning.py
4. Testing.py
5. Pruning.py
6. test.py

Node.py

Definisce la classe node la quale rappresenta la struttura dati per rappresentare i nodi dell'albero di decisione.
Questa classe risulta avere diversi attributi:

- *Y*: indica l'etichetta di un nodo, cioè il valore con il quale è stato classificato il nodo in base ai dati utilizzati.
- *attribute*: rappresenta l'attributo che viene testato nel nodo. In base al valore di questo attributo verrà selezionato uno dei suoi figli.
- *children*: rappresenta un dizionario <chiave-valore> in cui la chiave risulta essere un particolare valore dell'attributo indicato nel campo *attribute*, mentre il valore risulta essere un altro oggetto node che rappresenta il figlio.
- *leaf*: è un attributo booleano utilizzato per distinguere se un nodo è una foglia oppure no. Ha valore True se il nodo è una foglia, cioè un nodo senza figli, ha valore False in caso contrario.
- *error:* rappresenta il numero di errori di classificazione fatti nel nodo risulterà utile per implementare la strategia di pruning.

La classe è inoltre dotata di alcuni metodi che risultano utili per l'utilizzo degli oggetti di tipo node.

- *getChild(value)*: ritorna il valore associato alla chiave passata come parametro, ritorna dunque il figlio associato al particolare valore passato come parametro.
- *countNodes()*: ritorna il numero di nodi del sotto-albero che ha per radice il nodo self.
- i*ncError()*: incrementa il valore dell'attributo *error*.
- *childrenLeaf()*: ritorna il valore True in caso i figli siano tutti delle foglie
- *childrenError()*: ritorna il numero di errori commessi dai figli.

Tree.py

Rappresenta la struttura dati necessaria per rappresentare un albero. Ha due attributi uno identifica la radice dell'albero, l'altro il numero di nodi che contiene l'albero. La classe ha un metodo il quale esegue il conteggio dei nodi e provvede ad aggiornare il relativo campo.

Learning.py

In questo file viene implementato l'algoritmo *Decision-Tree-Learning(Data, Attrs, pData)* visto a lezione. Si utilizza come misura di impurità l'entropia, tale misura indica quanto è diversificato il Data-Set. Utilizzando questa misura è possibile calcolare il guadagno di ciascun attributo cioè quanto un attributo riesce a rendere il dataset puro, viene selezionato l'attributo che rende massimo il guadagno. Dopo di che viene calcolato il valore dell'etichetta che compare più spesso nel Data-Set e infine viene creato un nodo con le caratteristiche appena citate.
Si passa poi ad assegnare ricorsivamente i figli del nodo appena creato richiamando la funzione *Decision-Tree-Learning(Data, Attrs, pData)* con gli adeguati parametri. Un nodo ha tanti figli quanti sono i valori del dominio dell'attributo. La funzione ritorna un nodo, il nodo che viene ritornato dalla prima chiamata risulta essere la radice dell'albero di decisione.

Testing.py

E' implementata la funzione necessaria per testare l'albero di decisione su un insieme di dati. La funzione *TestData(tree, data)* non fa che analizzare ad una ad una le tuple di data utilizzando la funzione *TestQuery(node, data)* la quale naviga all'interno dell'albero di decisione, partendo dalla radice, fino a che non raggiunge una foglia e a quel punto restituisce il valore dell'etichetta contenuto nella foglia e la confronta con il reale valore della tupla per contare gli errori. La funzione *TestData(tree, data)* viene chiamata passando come parametro l'albero di decisione e i dati da utilizzare e ritorna il numero di errori commessi.

Pruning.py

In questo file è implementa una strategia di pruning basata sull'errore sul Validation-Set. La funzione *Pruning(tree, data)* non fa altro che testare l'albero di decisione su un insieme di dati in maniera del tutto simile a quanto viene fatto con la funzione *TestData(tree, data)* descritta sopra, con la differenza che ad ogni passo oltre a scegliere il nodo successivo si confronta l'etichetta del nodo con quella della tupla di dati che si sta analizzando in caso di errore si incrementa il campo *error* del nodo. Dopo di che viene richiamata la funzione *pruneWalk(node)* che non fa altro che scendere fino alle foglie per poi risalire ed eventualmente effettuare il pruning. Se il numero di errori dei figli risulta essere maggiore degli errori del padre e i figli sono delle foglie viene effettuato il pruning cioè le foglie vengono eliminate e il nodo padre diventa una foglia.

test.py

In questo file viene effettuato il test dell'algoritmo di apprendimento e della strategia di pruning. I dati vengono letti da un file .csv dopo di che vengono permutati casualmente e poi suddivisi in Test-Set, sul quale costruire l'albero, Validation-Set, per realizzare la strategia di pruning, Test-Set per testare le performance dell'albero di decisione creato.
Una volta creato l'albero, viene testato prima e dopo il pruning riportando il numero di nodi dell'albero, il numero di errori e l'errore percentuale.
Questo test viene effettuato utilizzando tre Data-Set diversi.

Dataset

I Data-Set sono memorizzati in dei file .csv i quali vengono letti con appropriate funzioni fornite da alcune librerie python. I Data-Set utilizzati sono 3.

- Cars
  Questo Data-Set contiene 1728 istanze. Sono presenti 6 attributi più il campo che rappresenta la classe. Tutti i campi risultano essere categorici con valori di tipo stringa. Questo Data-Set è stato così suddiviso: Training-Set [0:1000], Validation-Set [1000:1350] e Test-Set [1350:1728]. (https://archive.ics.uci.edu/ml/datasets/Car+Evaluation)
- Nursery
  Il Data-Set contiene 12960 istanze. I dati sono articolati in 8 attributi più il campo che rappresenta la classe. Gli attributi sono tutti categorici con valori di tipo stringa. Il Data-Set è stato suddiviso come segue: Training-Set [0:6400], Validation-Set [6400:9100] e Test-Set [9100:12960]. (https://archive.ics.uci.edu/ml/datasets/Nursery)
- Plant
  Il Data-Set contiene 2961 istanze. Sono presenti 6 attributi più quello relativo a rappresentare la classe. Gli attributi sono categorici e assumono valori interi. Il Data-Set è stato suddiviso: Training-Set [0:1400], Validation-Set [1400:2000] e Test-Set [2000:2961].

Output

Data-Set Plant

Nodi	Errori Training-Set	Errore % Training-Set	Errori Test-Set	Errore % Test-Set
303	36	2,5714	44	6,3676
289	42	3,0	37	5,3546

Data-Set Cars

Nodi	Errori Training-Set	Errore % Training-Set	Errori Test-Set	Errore % Test-Set
314	0	0,0	28	7,4074
295	9	0,9	24	6,3492

Data-Set Nursery

Nodi	Errori Training-Set	Errore % Training-Set	Errori Test-Set	Errore % Test-Set
961	0	0,0	123	3,1865
921	16	0,25	114	2,9534

Questo risulta essere l'output prodotto dall'esecuzione del file test.py. In ogni tabella sono riportati il numero di nodi dell'albero, che comprende sia nodi interni che foglie, il numero di errori e l'errore percentuale sul Training-Set e sul Test-Set.
La prima riga di ogni tabella contiene i dati dell'albero generato da DT-Learn mentre la seconda riga riporta i dati relativi all'albero dopo aver effettuato il pruning.
Come possiamo vedere l'effetto della procedura di pruning porta a ridurre il numero di nodi di ciascun albero con un conseguente aumento dell'errore sul Training-Set ma con una riduzione del tasso di errore sul Test-Set. Nei Data-Set Cars e Nursery si può notare come l'albero si adatti perfettamente ai dati su cui è stato addestrato, infatti testando l'albero sul Training-Set gli errori risultano essere assenti. Testando invece l'albero sul Test-Set si riscontra la presenza di errori. Il pruning riesce ad alleviare il tasso di errore che si ha sul Test-Set, producendo un aumento del tasso di errore sul Training-Set.
Il pruning quindi allevia il problema dell'Overfitting.