Alberi di decisione con dati mancanti

Edoardo Re

Aprile 2019

1 Decision Tree

In questo progetto è stato implementato l'algoritmo di DecisionTreeLearning descritto nella sezione 18.3 di Russel Norvig. L'algoritmo prende in ingresso un data set e ritorna l'albero di decisione.

2 Data sets utilizzati

I data sets utilizzati reperiti da UCI sono:

1. ***Blood Transfusion:*** size: 510, inputs: 5, classes: 2
2. ***Haberman:*** size: 240, inputs: 4, classes: 2
3. ***Iris:*** size: 150, inputs: 5, classes: 3 (Stringhe)
4. ***Mammographic Masses:*** size: 270, inputs: 6, classes: 2

Tali data sets sono contenuti in file *.txt* e hanno valori continui. Da notare che la size è sempre multiplo di 10, questo per l'utilizzo successivo di *Ten Fold Cross Validation* che divide l'intero data sets in 10 gruppi di size identica.

3 Codice

Il PROGETTO è suddiviso nei seguenti moduli ***.py***:

- ***DecisionTreeLearning.py*** In questo modulo si implementa l'algoritmo omonimo al nome del file *.py*, ripreso dal REPOSITORY pubblico e opportunamente adattato alle nostre esigenze. Per calcolare l'information gain si utilizzano funzioni della libreria Python math. L'algoritmo crea un albero di decisione dato in ingresso un data set con valori di attributi continui, mentre come classificazione per ogni examples ha valori discreti o stringhe. Per creare tale albero su valori continui si utilizza il metodo di split su una threshold, l'algoritmo eseguirà lo split dell'attributo disponendo il sottoalbero con valore di attributo $>$ della threshold sul ramo di destra, mentre per valore $\leq$ della threshold sulla sinistra.

- ***DataSet.py*** Nel seguente modulo si definisce la struttura di classe DataSet e si implementa la funzione *setDataset(file, attrnames, target, values)* che si occupa del parsing del file *.txt*, ovvero per ogni riga crea un example in una lista (semplicemente chiamata examples) e considera le virgole nel .txt per separare i valori di differenti attributi sulla stessa riga. Nel data set Mammographic Masses mancano alcuni valori, si è scelto di saltare le righe con valori mancanti ovvero di non considerare i corrispondenti example.

- ***DecisionTree.py*** Questo modulo serve a definire la struttura dell'albero che ha per ogni nodo un attributo che si dirama in al più due rami che si basano sul valore di threshold. La call è utile per vedere, dato un example in un albero esistente, la classificazione che ne risulta percorrendo il path dalla radice fino alla foglia. Sono inoltre presenti funzioni per aggiungere un sotto albero ad un nodo e due funzioni get per ritornare l'attributo e la soglia di un nodo. Di fondamentale importanza è la funzione GraphViz che visita semplicemente per livelli l'albero (breadth first) e crea automaticamente un file .gv in linguaggio DOT che viene interpretato dalla libreria GraphViz e permette di visualizzarlo in un .pdf che si apre automaticamente durante l'esecuzione. Attenzione: per una seconda esecuzione del Main.py si ricorda di chiudere tale file .pdf per non riscontrare errori di compilazione. La visualizzazione grafica rende intuitiva l'analisi del Decision Tree. Infine viene definita la classe Leaf che appunto definisce le foglie del nostro albero decisionale.

- ***Main.py*** Questo modulo è il fulcro del progetto e contiene i test richiesti, tutte le chiamate a funzioni di altri moduli vengono appunto effettuate da qui. All'avvio dell'esecuzione di Main.py si chiede all'utente quale data set si intende testare, per prima cosa si crea l'albero di decisione sull'intero data set scelto con tree = DecisionTreeLearner(fileDataset), successivamente chiamando tree.GraphViz() si esegue una visita per livelli dell'albero appena creato e si crea in linguaggio DOT l'albero da visualizzare in un file .pdf. Con test(fileDataset) si esegue TenFoldCrossValidation inizialmente sul data set scelto con tutti i valori al completo, successivamente vengono rimossi dei valori casualmente con probabilità uniforme tramite la chiamata di removeValAttrWithProb e si ripete TenFoldCrossValidation dopo aver inserito nuovamente i valori nel dataset secondo il metodo descritto in Mitchell 3.7.4. Il compito di inserire i valori mancanti è affidato alla funzione replaceMissingValue che utilizza la funzione mostCommonValueOfAttrInDataSet che ritorna il valore dell'attributo più comune. Sullo schermo verranno stampati tutti i singoli risultati, la parte più importante è la media che viene eseguita tra risultati in cui si è fatta la TenFoldCrossValidation con la stessa probabilità di rimozione dei valori nel data set. Le medie nei quattro casi ovvero quando $p = [0.0, 0.1, 0.2, 0.5]$ vengono mostrate in un grafico grazie alla libreria matplotlib di Python.

4 Esempio di funzionamento

- Eseguire il file Main.py
- inserire un valore da 1 a 4 per selezionare uno tra i seguenti: 1) Blood Transfusion 2) Iris 3) Haberman 4) Mammographic Masses
- Attenzione: Nel caso in cui non sia presente la libreria GraphViz la compilazione riporta errore ed è dunque richiesto di scaricarla.

5 Risultati

I risultati qui riportati sono stati ottenuti da un laptop con processore $M3-7Y30$ e 4GB di Ram. I test condotti non sono influenzati dalle prestazioni del calcolatore poiché si considerano soltanto le classificazioni corrette. Questi risultati potrebbero essere analoghi a quelli su un calcolatore molto più potente. Di seguito riportiamo la tabella e il grafico corrispondenti al test nei quattro casi, uno per ogni data set. Il tempo di esecuzione del programma su un data set tipicamente è inferiore ai 30 secondi.

Data Set	$p = 0.0$	$p = 0.1$	$p = 0.2$	$p = 0.5$
Blood Transfusion	71%	73%	82%	91%
Iris	91%	77%	63%	71%
Haberman	72%	73%	79%	90%
Mammographic Masses	84%	79%	70%	76%

Table 1: Percentuali classificazioni corrette in una generica esecuzione.

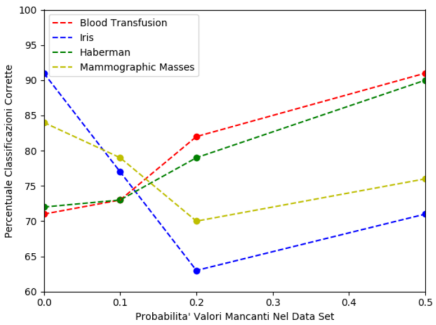

Figure 1: Valori della tabella.

6 Analisi e conclusioni finali

Dai risultati precedenti osserviamo che i quattro data set hanno andamenti differenti fra loro, possiamo però notare che l'algoritmo DecisionTreeLearning opera apprendendo correttamente, fornendo poi classificazioni esatte in un range che va dal 63% al 91%. Notiamo che, se un data set con valori al completo ha percentuale di classificazione abbastanza bassa intorno al 70%, se andiamo a rimuovere i valori e li reinseriamo con il metodo descritto in Mitchell 3.7.4 allora la percentuale di classificazione migliorerà. Al contrario quando in un data set completo le classificazioni sono effettuate quasi tutte correttamente in un range dall'84% a salire, si nota che rimuovendo i valori e inserendo col metodo di Mitchell 3.7.4 si peggiora rispetto al caso iniziale. Questi andamenti caratteristici non sono dovuti dai nostri algoritmi implementati ma sono caratteristiche proprie dipendenti dal data set stesso. La dipendenza è correlata alla size del data set, ovvero all'aumentare di essa, aumenta la precisione nella costruzione dell'albero che descrive le caratteristiche di un oggetto/individuo/fenomeno/malattia... campionato dal data set. Notiamo che in un data set completo vengono classificati molto meglio i data set con size maggiore, minor numero di attributi e minor numero di classi. I risultati così ottenuti confermano la teoria studiata precedentemente e permettono di comprendere al meglio le differenze che si notano tra data set con attributi, size, classi e range di valori di attributi differenti.