Alberi di decisione con dati mancanti

Mirco Ceccarelli

24 gennaio 2019

1 Introduzione

Lo scopo di questo elaborato è quello di valutare le accuratezze ottenute dall'algoritmo di apprendimento per alberi di decisione al variare della percentuale dei dati mancanti nel dataset.

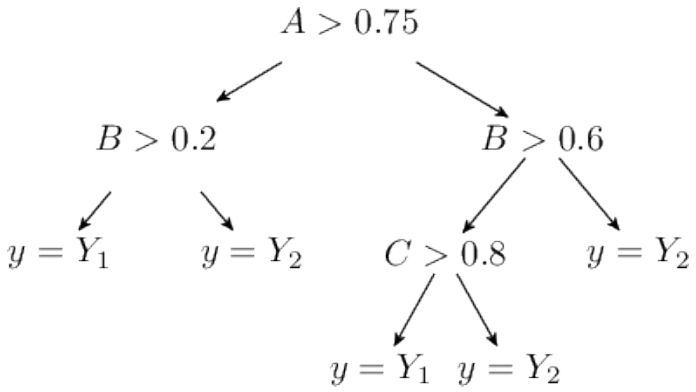

Figura 1: Un esempio di albero di decisione. [*ce.unipr.it*]

1.1 Alberi di decisione

Nell'ambito del *Machine Learning* un metodo semplice ed efficace per risolvere problemi di classificazione è quello di utilizzare gli alberi di decisione.

Un albero di decisione, costruito a partire da un insieme di dati iniziali (**dataset**), è un albero in cui ogni nodo interno è associato ad una particolare "domanda" su una certa caratteristica dei dati. Da questo nodo escono tanti archi quanti sono i possibili valori che la caratteristica può assumere, fino a raggiungere le foglie che indicano la categoria associata alla decisione.

Lo scopo è quello di allenare questo albero su alcuni esempi di dati (*training set*) in modo da poter classificare altri dati non precedentemente visionati (*test set*) con un certo livello di confidenza.

Un problema comune con gli alberi di decisione è che spesso si adattano fin troppo bene ad alcune caratteristiche specifiche solo del *training set*, e questo porta ad un calo delle prestazioni sul *test set*. Si parla in questo caso di ***overfitting***.

Esistono comunque diverse tecniche per prevenire l'*overfitting*: una di queste è l'utilizzo della *cross validation* (vedi sezione 3.2).

1.2 Gestione dei valori mancanti

L'algoritmo standard per l'apprendimento tramite alberi di decisione non gestisce i casi in cui alcuni esempi siano privi di qualche attributo.

Una strategia per la gestione di questi casi è quella di assegnare all'attributo mancante il valore più comune che esso assume rispetto a tutti gli altri esempi che raggiungono il nodo.

Un'altra strategia, più complessa, è quella di far finta che l'esempio abbia *tutti* i possibili valori dell'attributo, pesando però ognuno di questi in base alla sua frequenza negli esempi che raggiungono il nodo. In questo caso l'algoritmo di classificazione dovrebbe quindi seguire tutti i rami uscenti dal nodo per cui manca il valore dell'attributo e moltiplicare i pesi lungo ogni cammino.

Ai fini di questo elaborato è stato scelto di utilizzare la prima strategia.

2 Esperimento

L'esperimento si può dividere in due fasi.

Nella prima fase è stato implementato in Python l'algoritmo di apprendimento per alberi di decisione descritto in classe, ed è stata aggiunta la strategia per gestire valori mancanti, come riportato nella sezione 1.2.

Nella seconda fase è stato applicato l'algoritmo a problemi di classificazione su tre datasets scelti da UCI Machine Learning Repository, calcolandone l'accuratezza con *10-fold cross-validation* dopo aver rimosso, rispettivamente, lo 0%, il 10%, il 20% e il 50% dei dati.

Le accuratezze sono state poi inserite in un grafico, riportato nella sezione 4.

3 Implementazione

Il progetto è stato sviluppato con Python versione 3.7.2 e richiede l'installazione della libreria *matplotlib* per la generazione di grafici in 2d.

3.1 Alberi di decisione

La struttura di un albero di decisione (nodo, rami, foglia) è implementata nei file ***decision_leaf.py*** e ***decision_fork.py***. La classe *DecisionLeaf* ha come unico attributo il risultato della predizione. La classe *DecisionFork*, invece, contiene l'attributo da testare e un dizionario che rappresenta i vari rami, uno per ogni possibile valore dell'attributo stesso.

L'algoritmo standard per l'apprendimento tramite alberi di decisione, implementato in forma ricorsiva, si trova invece nel file ***dt_learner.py*** il quale attinge anche da altre funzioni di aiuto che si trovano in ***utils.py***.

3.2 Cross validation e gestione di attributi mancanti

Per testare l'accuratezza dell'algoritmo e per evitare *overfitting* viene richiesta la tecnica di ***k-fold cross-validation***, implementata nel file ***cross_validation.py***.

Essa consiste nella suddivisione del dataset in k parti uguali e, ad ogni passo, la k-esima parte del dataset si considera come *test set*, mentre la restante parte costituisce il *training set*. Si allena quindi il modello sul *training set* e si cerca di predire il restante gruppo con le conoscenze acquisite. L'accuratezza è il rapporto tra il numero di predizioni corrette e il numero di predizioni totali.

Per una maggiore precisione questo procedimento può essere ripetuto più volte (è il parametro *trials* della funzione *cross_validation()*), ogni volta rimescolando il dataset, per poi prendere la media dei risultati come stima dell'accuratezza.

La gestione dei valori mancanti avviene nel file ***missing_values.py***. Per prima cosa la funzione *randomly_remove_values()* prende in ingresso un dataset e la probabilità p di rimuovere un elemento, e restituisce un nuovo dataset (lasciando inalterato l'originale), uguale ma con gli elementi rimossi uniformemente e casualmente rispetto a p.

La funzione *handle_missing_values()*, invece, prende in ingresso un dataset e sostituisce tutti gli eventuali valori mancanti di un attributo con il valore più comune che assume quell'attributo rispetto agli altri esempi (vedi sezione 1.2).

3.3 Dataset

Nel file ***dataset.py*** è stata fornita una implementazione della classe *DataSet*. Essa contiene una lista di esempi (righe di valori di attributi) e un target (l'indice di un attributo che l'algoritmo proverà a predire).

Nella cartella *./datasets/* sono presenti tre dataset, reperiti da *UCI ML Repository*: *car.txt, phishing.txt, nursery.txt*. Per maggiori informazioni su questi dataset si rimanda al file *README.txt* all'interno della cartella.

3.4 Test

Nel file ***tests.py*** sono stato eseguiti i seguenti esperimenti:

- Per ognuno dei tre dataset, è stata calcolata l'accuratezza con 10 trials di *10-fold cross-validation*, dopo aver rimosso alcuni attributi uniformemente e casualmente con le probabilità p=[0.0, 0.1, 0.2, 0.5].

- Dopo aver quindi ottenuto quattro accuratezze per ogni dataset, i dati sono stati riportati in un grafico generato con la libreria *matplotlib*.

I risultati sono riportati di seguito.

4 Risultati

Di seguito sono riportati i risultati di una possibile esecuzione dei test.

	0%	**10%**	**20%**	**50%**
car.txt	93.709%	78.593%	72.1881%	67.3322%
phishing.txt	88.3074%	82.114%	78.6847%	70.716%
nursery.txt	98.8657%	81.713%	65.7191%	51.0756%

Tabella 1: Accuratezze rilevate al variare della percentuale di dati mancanti.

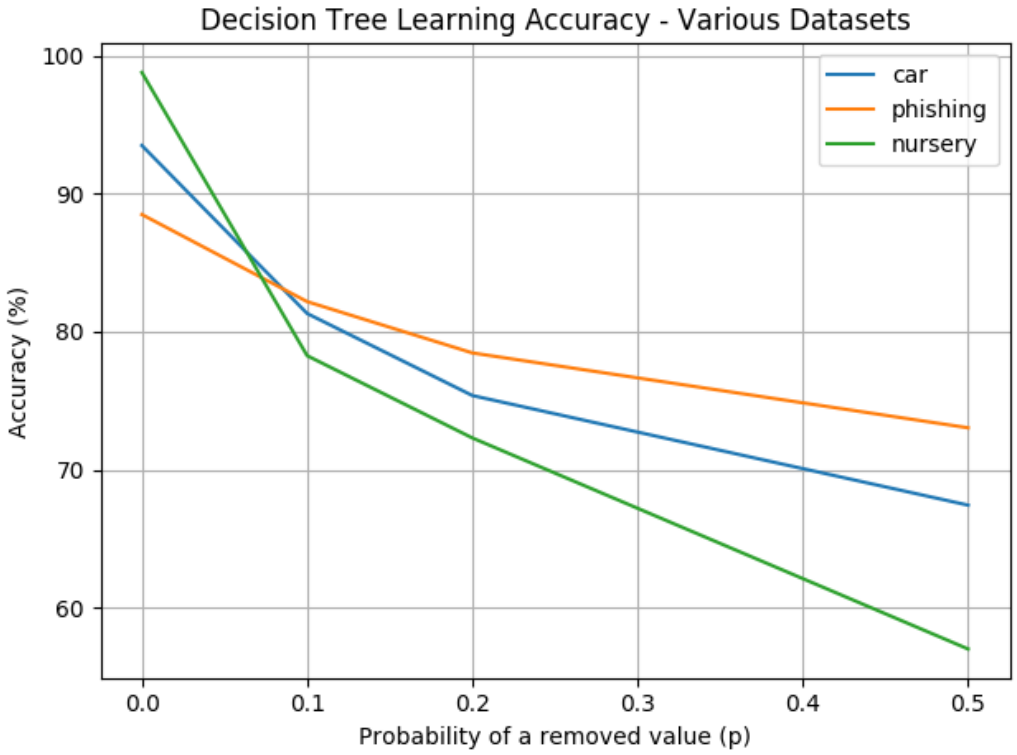

Figura 2: Gli stessi dati della Tabella 1, riportati in un grafico.

Aldilà delle differenze tra i dataset, relative principalmente al diverso numero di esempi e/o di attributi, dal grafico si può osservare come l'andamento delle tre curve sia sostanzialmente lo stesso.

Infatti, come ci si poteva aspettare, l'accuratezza dell'algoritmo di learning risulta inversamente proporzionale al numero di dati mancanti nel dataset. In altre parole, più ci sono dati mancanti nel dataset e più è difficile l'apprendimento, a prescindere dalla bontà del modello utilizzato.