Relazione AI: Alberi di decisione con attributi mancanti

Marco Solarino

Aprile 2020

1 Introduzione

In questo elaborato è stato implementato in Python l'algoritmo per l'apprendimento di alberi di decisione affrontato in classe, considerando il caso di attributi mancanti nel data set. Sono stati costruiti alberi al variare della probabilità della presenza di attributi mancanti (p = 0, 0.1, 0.2, 0.5), osservando i risultati sul test set.

2 Algoritmo

L'algoritmo mantiene la stessa struttura della versione che non gestisce valori mancanti ma subisce variazioni nelle funzioni che chiama. Le principali differenze si riscontrano in *importance()* che computa l'attributo migliore per dividere il train set nei nodi figli. In essa bisogna infatti gestire i pesi delle istanze del dataset, non più unitari, che variano in base alla presenza di valori mancanti nelle istanze degli esempi al nodo attuale. Il peso rappresenta la "frazione" di istanza che è stata propagata al nodo: è unitario se presenta il valore dell'attributo, viceversa è ottenuto dalla seguente formula

$$w = w \cdot fraction$$

dove fraction è il numero di istanze che presentano quel preciso valore dell'attributo tra tutte le istanze con valore noto e w è il peso iniziale dell'istanza. L'istanza può essere ulteriormente frazionata se presenta altri valori mancanti durante un'altra iterazione. Queste differenze riguardano dunque il calcolo del gain e quindi dell'entropia, che per vogni valore dell'attributo sfrutta i pesi. La costruzione dell'albero di decisione richiede molte risorse con l'aumentare di p: aumentano il numero di nodi e aumenta il tempo necessario ad eseguire *importance()* che deve calcolare, per ogni valore dell'attributo e per ogni attibuto non ancora testato nel ramo, i pesi e restituire quello con gain maggiore.
Altre differenze sono state implementate nella funzione *pick_examples()* che seleziona gli esempi da trasmttere ai nodi figli e che deve dunque considerare anche le istanze con il valore dell'attributo mancante (contrassegnato da '?') e

aggiornare il loro peso. Infine la classe *Node* presenta *plurality_class* che viene usata in fase di test quando l'istanza non può più procedere nell'albero perchè manca il valore dell'attributo del test attuale. In questo caso viene classificata con la classe più frequente negli esempi del nodo, che viene appunto indicata da *plurality_class*, passata al nodo alla sua creazione.

3 Riprodurre i risultati

Per eseguire un test viene chiamata la funzione *test_precision(data_set, p)* dove p rappresenta la probabilità della presenza di attributi mancanti. Il data set deve essere in formato csv con la classificazione come ultimo attributo e gli attributi devono essere categorici. Per leggere e modificare il data set è stata usata la libreria pandas, che ha permesso di vedere in fase di debug il data set e le modifiche che venivano apportate e di accedere facilmente alla struttura dati. La funzione *uniform_deletion(p, data_set)* elimina ogni cella del data set (sostituendo il valore con '?') con probabilità p. Per dividere il data set in train set e test set è stata usata la funzione *train_test_split()* di sklearn con dimensioni rispettivamente dell' 80% e 20%. Infine *test_precision()* chiama *decision_tree_learning()* che costruisce l'albero e per ogni elemento del test set *classifier(tree, example)* che restituisce la classificazione ottenuta dall'albero. Per ottenere la precisione, conta il numero di classificazioni corrette andando a leggere la classificazione riportata nel data set.

4 Risultati

Nei test sperimentali sono stati usati tre data set che rispettano le condizioni già descritte, con numero di istanze e attributi differenti:

- Car evaluation: 1728 istanze, 6 attributi
- Tic tac toe endgame: 958 istanze, 9 attributi
- Nursery: 12960 istanze, 8 attributi

Al variare di p per ciascuno dei tre dataset è stata chiamata *test_precision()* 20 volte ed è stata poi fatta la media .

Le precisioni riportate sono state:

4.1 Car evaluation

- p = 0: 92.8%
- p = 0.1: 78%
- p= 0.2: 73.2%
- p= 0.5: 69.6%

4.2 Tic tac toe

- p = 0: 85%
- p = 0.1: 76.1%
- p= 0.2: 70.1%
- p= 0.5: 65.1%

4.3 Nursery

- p = 0: 98.4%
- p = 0.1: 76.9%
- p= 0.2: 73.6%
- p= 0.5: 54.3%

5 Conclusioni

Come previsto la precisione sul test set cala sensibilmente all'aumentare della probabilità di mancanza di attributi nel data set. Questo è dovuto al fatto che si introduce molto rumore nei dati con conseguente aumento delle dimensioni dell'albero causando dunque overfitting. La mancanza dei valori anche nel test set inoltre, aumenta l'errore sui risultati perchè una volta arrivato al test dell'attributo di cui manca il valore, l'esempio viene classificato secondo la classe più frequente negli esempi di quel nodo. Il progetto ha dunque mostrato come la mancanza anche del solo 10% dei dati possa inficiare sulla precisione dell'algoritmo, e la necessità di gestire l'overfitting, facendo magari post-pruning dell'albero.