Decision Tree Learning

Assioma Andrea Aligi

5 febbraio 2024

1. Introduzione

Il seguente elaborato mostra la realizzazione di un albero decisione partendo da quello descritto nel capitolo 19.3 del libro Aritficial Intelligence: A Modern Approach (di Russell e Norvig (edizione 2020)), al quale viene modificato il passo base dello pseudocodice in modo che la ricorsione viene interrotta se la profondità è maggiore di un intero P assegnato oppure se il numero di esempi in examples è minore di un intero M assegnato.

Lo pseudocodice preso in considerazione utilizza come metodi di inserimento di nodi all'interno dell'albero i concetti di entropia e di information gain della teoria della teoria dell'informazione.

1.1. Hardware e sistema operativo utilizzati

Il codice è stato eseguito su un pc fisso con processore i5-10600k e 32gb di ram. Il sistema operativo usato è Arch Linux.

2. Creazione dell'albero di decisione

Per realizzare l'albero di decisione viene estratto il dataset dalla libreria python `ucimlrepo` tramite id (per esempio l'iris ha come id il 53). Successivamente, viene eseguito uno shuffle per randomizzare gli elementi all'interno: essendo gli esempi ordinati in base al target, ciò permette di dividere il seguente dataset in training e test senza che quest'ultimo abbia istanze di un solo target.
Una volta ottenuti i due dataset, viene chiamata la funzione che realizza l'albero: ricorsivamente, viene analizzato ciascun esempio in modo tale da determinare quale attributo inserire prima come nodo dell'albero, oppure viene aggiornata la frontiera aggiungendo foglie all'albero.
Ottenuto l'albero di decisione, viene disegnato graficamente tramite l'utilizzo della libreria `networkx` e ne viene calcolata la precisione testando sia il training dataset che il test dataset.

I dataset utilizzati sono stati reperiti da UCI Machine Learning Repository:

- Iris: classifica le tipologie di iris in base a una serie di attributi. Le variabili sono `sepal length`, `sepal width`, `petal length`, `sepal length` e `class` (setosa, versicolor, virginica).

- Raisin: distingue le tipologie di uvetta, analizzando i pixel di immagini. Le variabili in questo caso sono `Area` (dell'uvetta in pixel), `MajorAxisLength` (misura "dell'ambiente" calcolando distanza tra il contorno dell'uvetta e i pixel che lo circondano), `MinorAxiisLength` (rappresenta la lunghezza del segmento più lungo in pixel che si può creare all'interno dell'uvetta), `Eccentricity` (rappresenta la lunghezza del segmento più corto che si può creare all'interno dell'uvetta), `ConvexArea` (determina l'eccentricità dell'ellisse che approssima l'uvetta), `Extent` (è la dimensione in pixel dell'ellisse che approssima l'uvetta), `Perimeter` (rappresenta il rapporto tra le dimensioni dell'uvetta e quelle "dell'ambiente"), `Class` (Kecimen, Besni).

- Dry Bean: simile al resin, solo che in questo caso vengono analizzati fagioli asciutti. Le variabili sono `Area` (area del fagiolo), `Perimeter` (perimetro del bordo del fagiolo), `MajorAxisLength` (lunghezza del segmento più lungo contenuto dal fagiolo), `MinorAxisLength` (lunghezza massima del segmento che misura il fagiolo quando è in posizione verticale rispetto a uno degli assi), `AspetctRatio` (rapporto tra `MajorAxisLength` e `MinorAxisLength`), `Eccentry` (eccentricità dell'ellisse che approssima il fagiolo), `ConvexArea` (area del poligono (in pixel) che contiene il seme del fagiolo), `EquivDiameter` (diametro cerchio che ha la stessa area del seme del fagiolo), `Extent` (rapporto tra i pixel occupati dal fagiolo e quelli occupati dal riquadro dell'immagine), `Solidity` ($\frac{4\pi A}{P^2}$), `Compactness` (misura la rotondità di un oggetto), `ShapeFactor1`, `ShapeFactor2`, `Shapefactor3`, `Shapefactor4`, `Class` (Seker, Barbunya, Bombay, Cali, Dermosan, Horoz e Sira)

Per la realizzazione dell'albero, sono stati realizzati 5 files differenti:

- `main.py`: si seleziona l'id del dataset, lo split train/test, la massima profondità dell'albero (`P`) e il numero minimo di examples (`M`). Successivamente, chiama la funzione che crea l'albero, lo disegna e stampa la precisione (`create_tree`).

- `tree_creation.py`: definisce la funzione `create_tree` in cui viene inizializzati i dataset e creato l'albero di decisione tramite l'istanziazione di una variabile `DecisionTree`. Vengono inoltre calcolate e stampate le precisioni tramite la chiamata alla funzione `test_tree`, e viene disegnato l'albero tramite la chiamata alla funzione `plot_tree` e rappresenato graficamente grazie alla libreria `matplotlib`.

- `decision_tree.py`: definisce la classe `DecisionTree`.

- `tree_elements.py`: definisce le classi `Branch` e `Node` necessarie per la realizzazione dell'albero di decisione.

- `plot_tree.py`: definisce la funzione che disegna l'albero di decisione.

- `tree_testing.py`: definisce le funzioni che permetto il calcolo della precisione dell'albero (`test_tree` e `tree_precision`).

3. Analisi dei risultati

I risultati dei vari dataset sono stati i seguenti

3.1. Dataset Iris