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).

- Heart failure: `age`, `anemia`, `creatinine_phosphofinase` (indica il livello dell'enzima CPK nel sangue), `diabetes`, `ejection_fraction` (percentuale di sangue che esce dal cuore a ogni contrazione), `high_blood_pressure`, `platelets`, `serum_creatinine` (livello di siero di creatinine nel sangue), `serum_sodium` (livello di siero di sodio nel sangue), `sex`, `smoking`, `time` (periodo di follow-up), `death_event` (se il paziente muore durante il periodo follow-up (target)).

- Wine: `Alcohol`, `Maliacid`, `Ash`, `Alcalinity_of_ash`, `Magnesium`, `Total_phenols`, `Flavanoids`, `Nonflavanoid_phenols`, `Proathocyanins`, `Color_intensity`, `Hue`, `OD280_OD315_of_diluted_wines`, `Proline`, `class` (tipo di vino).

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

Nota: per scegliere `M` è stato analizzata la cardinalità degli insiemi di esempi dei sottoalberi di ciascun dataset

3.1. Dataset Iris

Nel caso del dataset Iris si hanno i seguenti risultati:

Split	P	M	training precision	test precision
30/150	3	0	0.8	0.7
30/150	2	0	1.0	0.8667
30/150	1	0	0.433333	0.3333
30/150	3	50	0.43333	0.3333
30/150	3	3	0.66667	0.5
30/150	3	2	0.86667	0.6667
30/150	2	2	0.9333	0.66667

Dalla seguente tabella si possono notare i seguenti comportamenti:

- nei casi P=1 M=0 e P=3 M=50 accade la stessa cosa in quanto vengono eliminati tutti i sotto-esempi, i quali variano tra [1,11]. Perciò si forma un albero di profondità 1.
- riducendo di 1 la profondità dell'albero aumenta la precisione dell'albero, mentre aumentare il limite minimo dell'insieme di esempi diminuisce la precisione.
- M influenza soprattutto la precisione del training (casi P=2 M=3 e P=2 M=2).

3.2. Dataset Heart Disease

Heart Diesase, rispetto a Iris, presenta più attributi. Nonostatnte ciò, l'albero ha le stesse dimensioni: gli attributi che non sono stati utilizzati, non sono seriviti per la classificazione, e quindi si sono presentati casi in cui gli esempi hanno stesso target.

Split	P	M	training precision	test precision
76/299	3	0	0.78947368	0.21052632
76/299	2	0	0.960526316	0.31578947
76/299	1	0	0.6842105263	0.592105263
76/299	3	50	0.61842105263	0.25
76/299	3	3	0.60526315789	0.18421053
76/299	3	2	0.77631578947	0.28947368

Anche in questo caso, ridurre la profondità dell'albero di una unità ne aumenta la precisione.

3.3. Dataset Wine

Il Wine dataset presenta molti attribute che non vengono utilizzati come nel caso precedente, e i sottoalberi hanno solo due foglie.

Split	P	M	training precision	test precision
34/178	3	0	0.941176	0.1470588
76/299	2	0	1.0	0.176470588
76/299	1	0	0.3529411764	0.3529411764

Siccome i sottoalberi hanno solo due foglie, non avrebbe senso importare M in quanto sarebbe come impostare P a 2. Come per il dataset precedente, anche qui non sono stati usati tutti gli attributi.