Alberi di decisione

Lorenzo Nuti

L'obiettivo di questo elaborato è di implementare l'algoritmo per l'apprendimento di alberi di decisione utilizzando l'entropia come misura di impurità e come criteri per terminare la ricorsione l'errore di classificazione p e il numero massimo di errori m.

Descrizione elaborato

L'elaborato è stato implementato in Python e suddiviso in vari **file**, descritti di seguito:

- MAIN.PY: è il file da eseguire, e contiene le impostazioni del programma e le istruzioni che richiamano le funzioni dei file DATASET.PY e TEST.PY per caricare i 3 data set scelti ed eseguire i test in base alle impostazioni selezionate.
- DATASET.PY: in questo file sono implementate la classe *Data* che associa un'istanza di dati al label corrispondente, la classe *Attr* che contiene il nome e il dominio di un attributo, e la classe *Label* che contiene il nome e il dominio dei label. Sono presenti anche le funzioni *load_plant_data*, *load_chess_data* e *load_poker_data* per caricare i 3 data set scelti, riordinare i dati in modo casuale, definire i domini degli attributi e dei label e separare il training set dal test set.
- DECISIONTREE.PY: questo file contiene le definizioni delle classi per gestire la struttura dell'albero decisionale. In particolare sono presenti la classe *Node*, che rappresenta un nodo dell'albero e ha la responsabilità di gestire la lista dei figli e tenere un riferimento all'attributo che contiene, e la classe *Leaf*, che tiene memoria del label che rappresenta. Entrambe le classi hanno anche l'attributo *attr_value*, che rappresenta il valore dell'attributo presente nella classe padre che permette di arrivare a tale figlio. Il file contiene inoltre la classe *TreePrinter*, che contiene alcuni metodi ripresi da https://github.com/jml/tree-format e poi adattati e modificati per stampare l'albero decisionale composto dalle classi precedentemente descritte.
- DTLEARN.PY: è il file che contiene il nucleo del programma, ovvero l'algoritmo di apprendimento e le funzioni ad esso associate. La funzione *dt_learn* implementa l'algoritmo, mentre le funzioni *cost*, *gain* e *max_gain* implementano rispettivamente l'entropia, il guadagno e la scelta dell'attributo che massimizza il guadagno. La classe *Distribution* gestisce i parametri p e m: l'algoritmo ad ogni chiamata ricorsiva crea un oggetto di questo tipo, che analizza il data set passato e associa ogni label al numero di volte che compare all'interno di tale data set. Fornisce quindi i metodi per calcolare il label più comune, la percentuale di ogni label e determinare se quel data set è una foglia oppure no in base ai valori di p e m precedentemente settati.
- TEST.PY: in questo file sono presenti le funzioni *risk* e *find_y* per calcolare l'errore sul test set e altre funzioni che richiamano quelle dei file sopra descritti per eseguire l'algoritmo e i test: le funzioni *print_p_table* e *print_m_table* per stampare la tabella con il tempo di apprendimento e gli errori su training e test set al variare rispettivamente di p e m; le funzioni *export_p_table* e *export_m_table* per stampare il valore del parametro e l'errore sul test set al variare rispettivamente di p e m secondo il formato a coppie (x, y) utile per esportare i dati; infine la funzione *print_tree* che stampa l'intero albero di decisione creato, il numero di nodi interni e di foglie, il tempo di apprendimento e gli errori su training e test set per i valori dei parametri assegnati.

In particolare come criteri per terminare la ricorsione nell'algoritmo di apprendimento sono stati usati i parametri p e m, che devono essere definiti in modo esclusivo (xor): quando viene settato il parametro p viene creata una foglia se la percentuale del label più frequente nel data set testato è maggiore o uguale a $1-p$; quando viene settato il parametro m viene creata una foglia se il numero di volte che compare il label più frequente nel data set testato è maggiore o uguale al numero di istanze del data set $- m$. Quindi p indica la percentuale massima di errori sul training set accettata all'interno di una foglia e m indica il numero massimo di errori sul training set accettati all'interno di una foglia.

Esecuzione test

Per eseguire il programma si devono selezionare le impostazioni desiderate all'interno del file MAIN.PY ed eseguirlo per visualizzare su terminale i risultati dei test richiesti. Il programma carica i 3 data set e prosegue a seconda della **modalità** selezionata:

- MODE.PRINT_TABLE: chiama su ogni data set le funzioni *print_p_table* e *print_m_table*, generando quindi due tabelle per data set che contengono il valore del parametro, il tempo utilizzato per eseguire l'algoritmo di apprendimento e gli errori su training e test set al variare rispettivamente di p e m. Questa è la modalità di default, che stampa i risultati dei test in un formato leggibile, evidenziando in particolar modo tramite la colorazione dell'output le eventuali variazioni dell'errore sul test set al variare dei parametri rispetto all'errore senza terminazione anticipata (parametri a 0): grigio se è uguale, blu se è migliore, giallo se è peggiore di meno dell'1% e rosso se è peggiore di più dell'1%.
- MODE.EXPORT_TABLE: chiama su ogni data set le funzioni *export_p_table* e *export_m_table*, generando quindi due righe di output per data set che contengono il valore del parametro e l'errore sul test set al variare rispettivamente di p e m secondo il formato a coppie (x, y) utile per esportare i dati. Questo output è lo stesso che viene generato nella modalità precedente, ma con un diverso formato che risulta difficile alla lettura diretta, ma comodo per esportare i dati verso l'esterno: è stato ad esempio usato per esportare i dati necessari alla creazione dei grafici sotto riportati.
- MODE.PRINT_TREE: chiama su un data set la funzione *print_tree* che stampa l'intero albero di decisione creato, insieme al numero di nodi interni e di foglie, il tempo necessario per eseguire l'algoritmo di apprendimento e gli errori su training e test set. In questa modalità è possibile selezionare quale tra i 3 data set usare e il valore del parametro p oppure del parametro m utilizzato per la creazione dell'albero.

Essendo i dati riordinati in modo casuale, al momento del caricamento è inoltre possibile impostare un seed specifico (di default è 1) in modo da ottenere dei risultati riproducibili.

Analisi risultati

Di seguito sono riportate le descrizioni dei data set scelti e l'analisi dei risultati dell'algoritmo di apprendimento, riportando nei grafici il valore dell'errore sul test set (in percentuale) al variare dei parametri. In particolare nello stesso grafico sono mostrati sia i risultati al variare di p che di m per poterli facilmente confrontare e sull'asse x vengono rappresentate le variazioni dei parametri, che si distinguono in questo modo: per ogni unità sull'asse x viene aumentato il parametro p dello 0.5% e il parametro m di 1. Quindi i parametri variano da un minimo di 0 (senza terminazione anticipata) per il quale l'algoritmo crea l'albero completo, fino a un massimo del 10% per quanto riguarda p e di 20 per quanto riguarda m.

Plant data set

Il primo data set selezionato è stato scaricato dal sito MLData e contiene 2691 istanze, delle quali 1883 (il 70%) usate per il training set e 808 (il 30%) usate per il test set. Ogni istanza rappresenta una pianta e contiene 6 attributi categorici interi che indicano l'*habitat* (da 1 a 6), il *colore* (da 1 a 4), il *tipo di foglia* (da 1 a 4), la *larghezza della foglia* (1 o 2), la *lunghezza della foglia* (da 1 a 4), l' *altezza* (da 1 a 3) ed è associata ad un label anch'esso categorico intero che indica se la pianta è *commestibile* oppure no (1 o 2). Dai risultati riportati in Figura 1 si vede bene come l'errore sul test set venga ridotto attraverso il metodo della terminazione anticipata, infatti si riesce a evitare l'overfitting che è presente quando i parametri sono impostati a 0, tuttavia per valori alti di p e m l'approssimazione è troppo alta e l'errore aumenta. Gli errori minimi per ogni tabella si raggiungono per p compreso tra 3.5% e 5.5% e per $m = 1$, e in questo caso si riesce a ottenere il risultato migliore usando il parametro p.

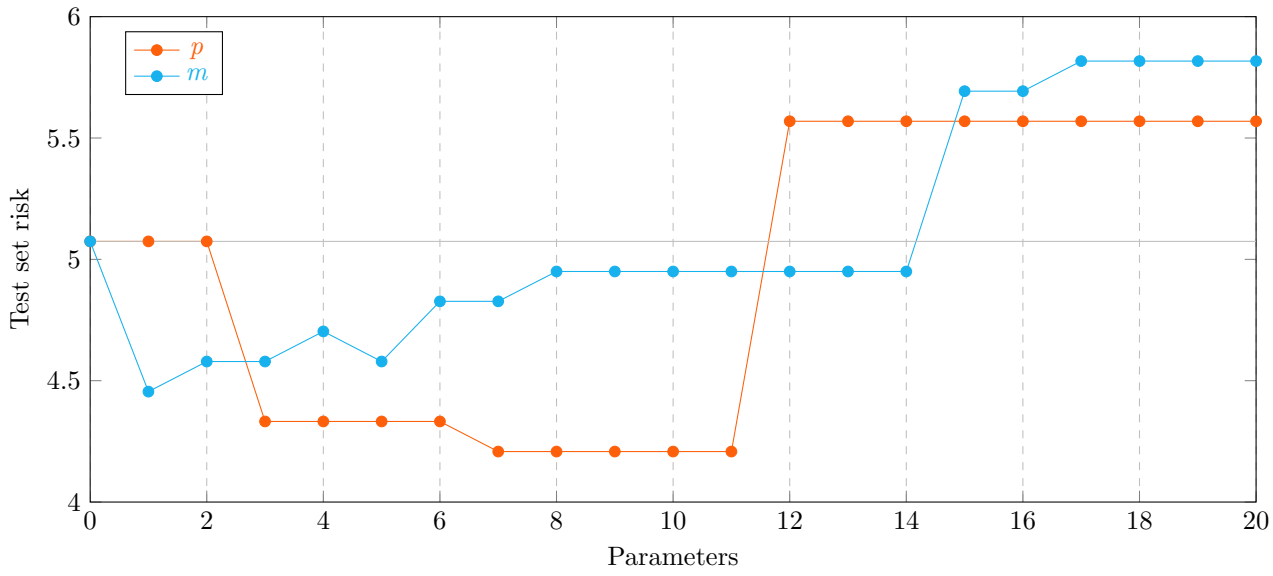

Figura 1: Plant data set

Chess data set

Il secondo data set selezionato è stato scaricato dal sito archive.ics.uci.edu e contiene 28056 istanze, delle quali 19639 (il 70%) usate per il training set e 8417 (il 30%) usate per il test set. Ogni istanza rappresenta le posizioni su una scacchiera di re e torre bianchi contro il re nero e contiene 6 attributi categorici, due per pezzo, che indicano *fila* (un carattere da 'a' a 'h') e *colonna* (un carattere da '1' a '8') di ogni pezzo. Ogni istanza è associata ad un label categorico sotto forma di stringa che indica, considerando le mosse ottimali per entrambi i giocatori, *in quante mosse il bianco riesce a vincere* (da 0 a 16) oppure se pareggiano. Per semplicità le possibili classificazioni sono state ridotte da 18 a 4, ovvero "draw", "low", "medium" e "high". Dai risultati riportati in Figura 2 si vede come in questo caso l'algoritmo si comporti in modo molto diverso in relazione al parametro usato: usando il parametro p l'errore varia leggermente e per p compreso tra 3.5% e 4.5% si ha un miglioramento dell'errore iniziale, mentre usando il parametro m si ha un miglioramento per $m = 1$ e poi peggiora notevolmente all'aumentare di m. In questo caso tuttavia si riesce a ottenere il risultato migliore usando il parametro m impostato a 1.

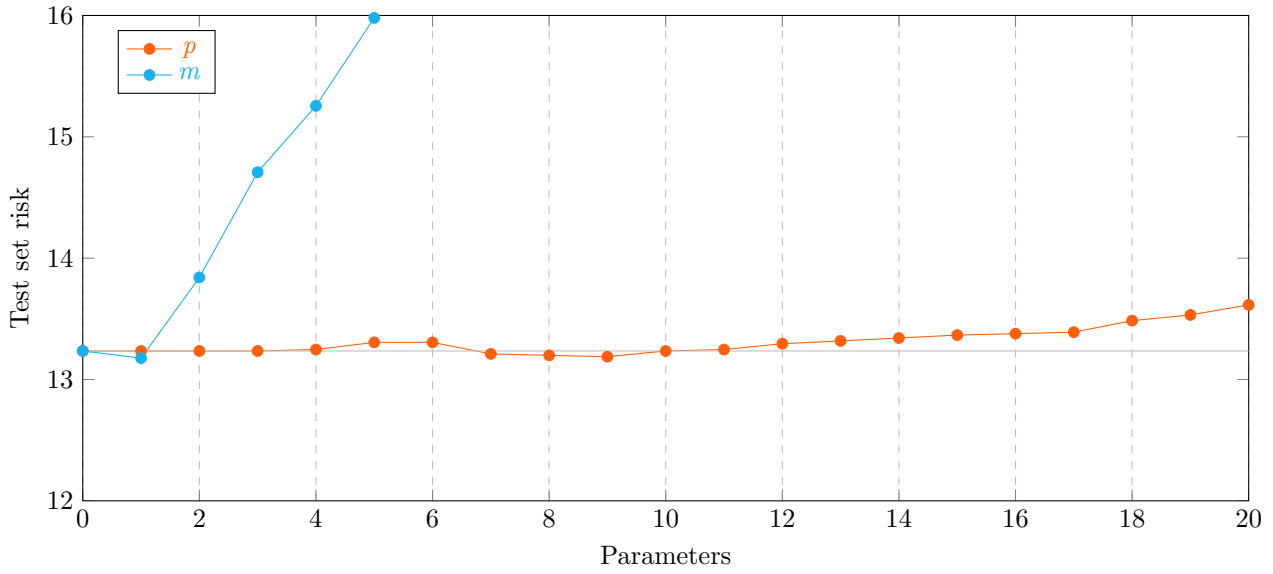

Figura 2: Chess data set

Poker data set

L'ultimo data set selezionato è stato scaricato dal sito archive.ics.uci.edu e contiene 1025010 istanze, delle quali 25010 usate per il training set e 1000000 usate per il test set, come indicato nella documentazione. Ogni istanza rappresenta una mano da 5 carte di poker e contiene 10 attributi categorici interi, 2 per carta, che indicano *seme* (da 1 a 4) e *numero* (da 1 a 13) di ogni carta. Ogni istanza è associata ad un label categorico intero che indica la *mano* (da 0 a 9) di tali carte nel poker: 0 corrisponde a niente, 1 a coppia, 2 a doppia coppia, 3 a tris, ecc. Per determinare la mano del poker l'ordine delle carte nella mano è ininfluente, quindi al momento del caricamento dei dati per ogni istanza le carte vengono riordinate dalla più bassa alla più alta per evitare di riconoscere come un dato diverso una stessa istanza ma con le carte ordinate diversamente; in questo modo è migliorata notevolmente la capacità di previsione dell'algoritmo. Dai risultati riportati in Figura 3 si vede che per questo problema, essendo i dati ben distinti e non soggetti ad errori, l'algoritmo si comporta nel modo ottimale quando viene creato l'albero completo, mentre se vengono usati i parametri p e m l'errore sul test set aumenta.

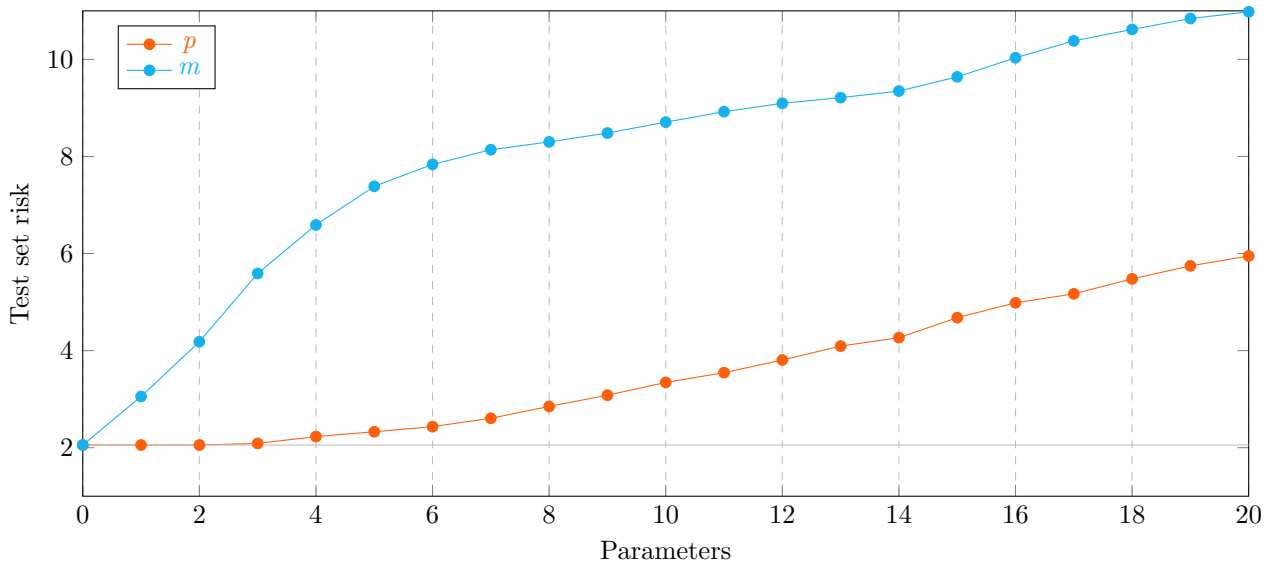

Figura 3: Poker data set