Progetto Intelligenza Artificiale: Pruning delle regole dell'Albero Decisionale

Lorenzo Gianassi

5 settembre 2020

1 Introduzione

Lo scopo del progetto è quello di andare a creare un albero decisionale utilizzando l'algoritmo ID3 con misura di impurità l'entropia. Una volta aver creato l'albero decisionale sarà attuata la strategia di Pruning, quindi sarà necessario trasformare l'albero in *regole* che rappresentano tutti i cammini dalle radici alle foglie. La strategia di Pruning sarà eseguita valutando l'errore sul ValidationSet. Andremo infine a eseguire i confronti sulle accuracy prima e dopo l'operazione di Pruning. Il dataset utilizzato sarà Adult dall'archivio UCI.

2 Descrizione del Progetto

Il progetto si basa sull'utilizzo del DataSet Adult, verrà diviso in TrainSet e TestSet (80-20). Il TrainSet verrà a sua volta diviso per ottenere il ValidationSet. Tramite l'algoritmo ID3 andremo a costruire l'albero di decisione utilizzando gli esempi contenuti nel TrainSet. Per scegliere l'attributo da utilizzare come nodo verranno calcolati il *Guadagno di Informazione* e l'*Entropia*, come misura di impurità. Si sceglierà l'attributo che porta al maggiore *Guadagno di Informazione*. L'obbiettivo di questo Progetto sarà quello di analizzare le accuracy ottenute dall'albero sui tre Set. In particolare, il caso su cui ci focalizzeremo, sarà la presenza o meno dell'Overfitting. L'albero creato dall'algortimo di apprendimento deve essere in grado di *Generalizzare*, cioè riuscire a prevedere la label di esempi che non sono stati utilizzati nel processo di allenamento (nel nostro caso gli esempi del TestSet). L'Overfitting avviene quando l'albero si adatta troppo alle caratteristiche del TrainSet su cui si è allenato portando ad aver una accuracy molto alta su quest'ultimo, ma ottenendo una peggiore accuracy sul TestSet. L'obbiettivo, perciò, è quello di avere una accuracy sul TestSet che si avvicini a quella sul TrainSet. Per ottenere questi valori sarà nessario trasformare il nostro albero in regole (in particolare DNF), per poi effettuare il Pruning. Questa trasformazione ci permette una lettura più semplice dell'albero e di potere eliminare qualsiasi elemento di una regola che apporta il maggior guadagno di accuracy.

Il Pruning consiste nell'eliminare un elemento alla volta da ogni regola selezionando, successivamente, l'eliminazione che ha portato al maggior miglioramento di accuracy calcolata sul ValidationSet (in modo Greedy). Successivamente effetturemo l'eliminazione, aumentando l'accuracy. Nel Progetto andrò a guardare le prestazioni al variare della profondità dell'albero e della dimensione del dataset che utilizzo, in modo tale da avere una panoramica completa del problema. Infine, andrò ad effettuare il Pruning e a valutare le prestazioni prima e dopo il suo utilizzo.

3 Componenti del progetto

Di seguito verranno illustrati i file che compongono il programma per l'esecuzione dell'esperimento.

3.1 DataSet.py

In questo file sono definiti il metodo che si occupa di leggere il file 'adult.csv' e di eliminare gli esempi con valori assenti(come da richiesta) e il metodo che si occupa dello split del DataSet secondo una percetuale. Il DataSet Adult presenta 48842 esempi con 14 attributi e la label 'income' che è il valore da prevedere.

3.2 Learning.py

All'interno di questo file si trovano tutti metodi ausiliari per l'algoritmo di apprendimento dell'albero come , ad esempio, il calcolo del *Guadagno di Informazione* e dell'*Entropia*. Naturalmente è presente anche l'algoritmo ID3 in cui ho aggiunto anche un check sulla profondità dell'albero che userò per eseguire i test sull'Overfitting.

3.3 Rules.py

In questo file saranno presenti tutti i metodi che si occuperanno di trasformare l'albero in regole, trovando tutti i cammini dalla radice alle foglie.

3.4 Utility.py

File usato per i metodi di calcolo dell'accuracy sia dell'albero, cioè scorrendo il dizionario che lo rappresenta, che delle regole.

3.5 Pruning.py

Nel File Pruning.py vengono definiti i metodi necessari per l'esecuzione del pruning. L'algoritmo scorre le regole e per ciascuna di esse prova a eliminare tutti gli elementi, uno alla volta, calcolando l'accuracy con le regole ottenute.

Ad ogni iterazione seleziono la potatura che mi dà maggiore guadagno di accuracy rispetto al valore precendente, se non trovo nessun miglioramento, mi fermo. Ho introdotto, inoltre, un controllo sull'accuracy del TrainSet, per cui se la precisione sul Validation dovesse superare quella sul Train l'algoritmo si fermerrebe. Questa scelta è dovuta al fatto che la dimensione del ValidationSet è molto inferiore rispetto a quella del TrainSet quindi la scelta dei letterali (elementi delle regole) da eliminare potrebbe adattarsi troppo alle caratteristiche del Validation sfavorendo il TrainSet.
Un'altra alternativa sarebbe potuta essere quella di individuare un limite di accuracy per il Validation in modo tale da mantenere la generalizzazione oppure mantenere una differenza in percentuale dall'accuratezza sul TrainSet.

3.6 Main.py

In quest'ultimo file è dove sostanzialemnte i test sono esguiti, presenta i metodi per fare il plot dei grafici e per eseguire il confronto a seguito del *Pruning*. Possono essere modificati i valori per eseguire dei test differenti.

4 Esecuzione e Risultati

Per esaminare al meglio il problema dell'*Overfitting* e dell'utilizzo del *Pruning* ho svolto più test su diversi aspetti, cercando di sottolineare la differenza fra le accuracies su TrainSet e TestSet.

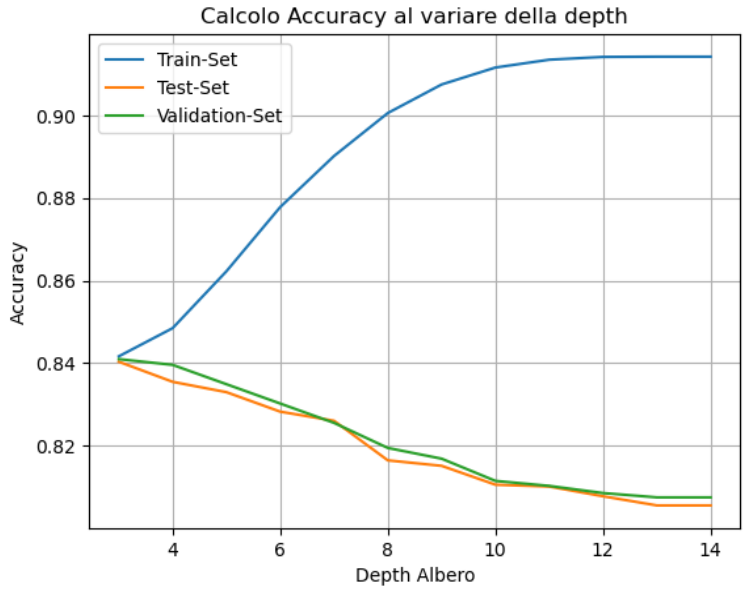

Figura 1: Accuracies al variare della Depth dell'Albero

Come si può notare, fissata la dimensione del dataset utilizzato, con l'aumentare della depth dell'albero, del numero di nodi e di conseguenza dei letterali usati nelle Regole la differenza di accuracy sul TrainSet e sul ValidationSet e il TestSet aumenta. Si nota che alla profondità circa di 8/9 i valori di accuracy iniziano a stabilizzarsi e la differenza si assesta.
Adesso analizzeremo invece come variano le accuracies al variare del numero di esempi del DataSet utilizzati.

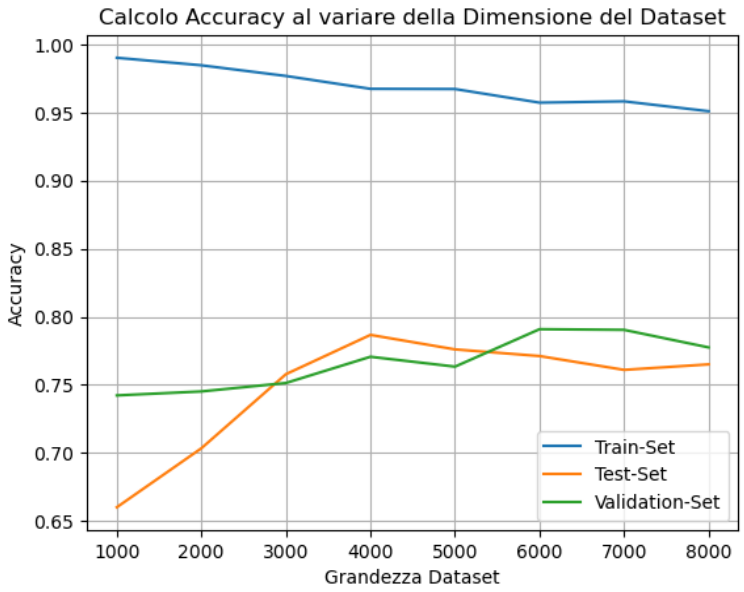

Figura 2: Accuracies al variare della Dimensione

Ho scelto queste quantità perchè si poteva apprezzare maggiormente il cambiamento di accuracies.
In particolare si può notare che all'aumentare del numero di esempi utilizzati la differenza fra le accuracies diminuisce. A dimensioni basse l'albero creato tenderà a non generalizzare bene e ad adattarsi alle caratteristiche del TrainSet.
Dati i valori appena analizzati, per avere una buona rappresentazione della utilità del Pruning ho effettuato l'algoritmo su un insieme di esempi pari a 8000 e con una profondità dell'albero pari a 8. In questo modo ho un numero di esempi che possa essere valido e una profondità che possa far occorrere l'Overfitting.
Riporto di seguito i risultati:

Accuracies Pre e Post Pruning		
DataSet	Pre-Pruning	Post-Pruning
TrainSet	92 %	100 %
ValidationSet	78 %	100 %
TestSet	79 %	99 %

5 Conclusione

Come possiamo notare dai dati ottenuti tramite l'utilizzo del Pruning siamo riusciti ad avvicinare di molto l'accuracy su TestSet, riducendo di molto la differnza con quella sul TrainSet . In questo modo le regole risultanti sono in grado di generalizzare correttamente su un numero di esempi molto maggiore rispetto a prima.

Riferimenti bibliografici

[1] Norvig, Peter (2010), *Artificial Intelligence: A Modern Approach*, Pearson, New Jersey.

[2] Tom M. Mitchell(1997),*Machine Learning*, McGraw-Hill.