Alberi di decisione con attributi continui e categorici

David Megli

Per implementare l'algoritmo richiesto ho implementato una struttura dati ***Node*** con i seguenti attributi e metodi (descrivo solo quelli non banali):

- ***attribute***: contiene il nome dell'attributo del dataset su cui effettuare il confronto, nel caso in cui il nodo non è una foglia. Se il nodo rappresenta una foglia, questo campo contiene invece un valore di classe (o *label*).
- ***threshold***: se *attribute* è "continuo" contiene il miglior threshold per l'attributo, ovvero quello che dividendo il dataset fornisce un *information gain* maggiore.
- ***children***: dizionario usato per referenziare i nodi figli, le chiavi corrispondono ai possibili assegnamenti di valori di *attribute* nel caso in cui esso rappresenti valori non continui. Se l'attributo è continuo le chiavi sono 0 e 1, e referenziano i figli sinistro e destro, corrispondenti ai valori di *attribute* < e >= di *threshold* rispettivamente.

Ho implementato l'algoritmo di apprendimento dell'albero di decisione tramite una funzione ***learnDecisionTree***, la quale iterativamente sceglie il "migliore" attributo, ovvero quello che separa meglio gli esempi. Tale attributo è quello che massimizza l'*information gain*, ovvero la differenza fra *impurità* dell'intero dataset e la somma delle impurità dei sottoinsiemi generati dall'assegnazione di ciascun valore dell'attributo, pesate dalla frazione di esempi con tale valore rispetto al dataset.
Per il calcolo dell'impurità ho implementato una funzione ***entropy***, che calcola appunto l'entropia di un insieme di esempi.
La funzione, dopo aver diviso l'insieme di esempi sulla base dell'attributo scelto come migliore, crea un nodo con tale attributo, a cui aggiunge un figlio per ogni possibile valore (2 nel caso di attributi continui), creato con la chiamata iterativa a learnDecisionTree, a cui passo il sottoinsieme degli esempi in cui l'attributo ha il valore corrispondente, e a cui rimuovo l'attributo appena valutato come migliore al passo attuale.
In questo modo mi ritrovo un albero in cui gli attributi che meglio classificano il dataset si trovano in "alto".
Quando ad una certa iterazione gli attributi, escluso l'attributo target (ovvero la classe), sono terminati associo gli esempi rimanenti alla classe più comune. Se gli esempi rimanenti appartengono ad una sola classe associo questi esempi a tale classe.

Ho utilizzato la struttura dati *dataframe* di *pandas* per estrarre i dati dalla *UCI Machine Learning Repository*, per questo ho utilizzato tale struttura anche all'interno dell'algoritmo.
Per valutare se un attributo sia continuo ho usato il confronto *attribute.dtype==float*, inoltre ho deciso di associare un threshold anche agli attributi la cui cardinalità dei valori superasse una certa soglia.

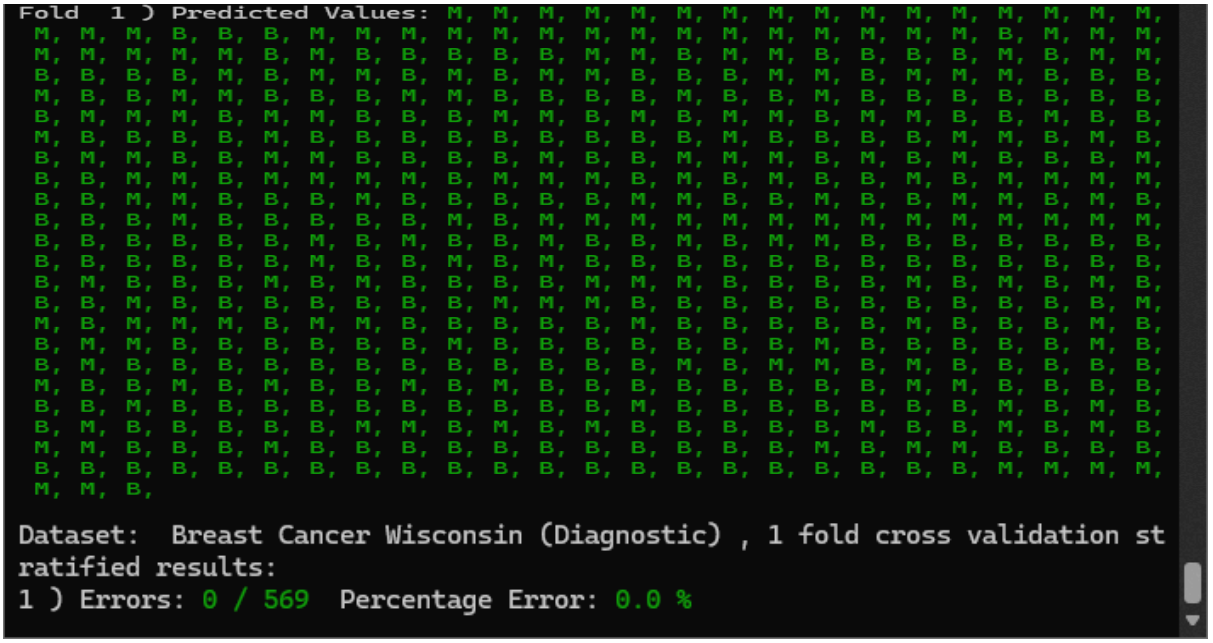
La funzione ***predict*** prende in input un singolo esempio ed effettua confronti sull'attributo del nodo attuale, scendendo lungo l'albero dalla radice a una foglia. Se l'attributo nel nodo è continuo si effettua un confronto con disuguaglianza, e se il valore è minore il nuovo nodo da considerare è il figlio sinistro, altrimenti il destro. Se l'attributo non è continuo si scende nel figlio la cui chiave è uguale al valore dell'attributo corrispondente. Una volta raggiunta una foglia ne restituisco il valore, che corrisponde alla classificazione finale dell'esempio.

La funzione ***predictAll*** prende in input un insieme di esempi e li classifica richiamando su ognuno di essi la funzione *predict*.

La classe ***DecisionTreeClassifier*** è quella che viene usata nel programma principale.
La funzione ***train*** genera *k folds*, ed effettua una *cross validation stratificata* nel seguente modo: aggiunge ad ogni *fold* una porzione uguale di righe del dataset (esempi) per ogni valore assunto dal campo target, in modo da mantenere (grossomodo) uguale la proporzione di esempi in ogni *fold*.
Successivamente, per ogni *fold* genera un sottoinsieme di esempi chiamato *validationSet* e uno chiamato *trainingSet*. Il primo è costituito da 1 *fold* ed è usato per testare l'albero di decisione, mentre l'altro è formato dai restanti k-1 *fold* ed è usato per far apprende l'albero. Per ogni *fold* calcola il numero di errori e l'errore relativo. Le prime 2 liste che restituisce contengono errori assoluti e relativi, le restanti contengono le dimensioni dei *validation set*, e le liste dei valori di classe originali e di quelli predetti dall'algoritmo (servono per la stampa a video).

Per utilizzare l'algoritmo è sufficiente eseguire ***main.py***, scegliere un dataset dalla lista (o un altro qualsiasi dalla *UCI Machine Learning Repository* inserendone l'id), ed inserire il valore corrispondente al numero di *folds* desiderate per la k–fold cross validation stratificata.
Il programma stampa a video la lista delle classi e delle predizioni, evidenziando in rosso gli errori, e riportando per ogni *fold* il numero di errori commessi e gli errori relativi.
Nota: ho scelto di mostrare direttamente i nomi delle classe e non personalizzare il programma per i soli dataset da me testati, in modo da poter usare il programma su qualsiasi altro dataset proveniente dalla *UCI Machine Learning Repository*.

Di seguito riporto dei test da me effettuati. Ho effettuato il primo test su un dataset relativo al cancro al seno (Wolberg,William, Mangasarian,Olvi, Street,Nick, and Street,W.. (1995). Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository. https://doi.org/10.24432/C5DW2B.)
Come primo (banale) test ho usato l'intero dataset sia come training set che come validation set, ed ovviamente l'accuratezza è stata del 100%. (Ho riportato solo le predizioni)



Di seguito riporto altri test effettuati sullo stesso dataset. (Riporto gli errori relativi alle k folds e solo alcune predizioni)

Breast Cancer Wisconsin) Test per k = 3. (*fold* 3 e gli errori per ogni fold)

```
Fold  3 ) Original Values: M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M,
M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M,
M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M,
M, M, M, M, M, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B,
B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B,
B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B,
B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B,
B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B,
Fold  3 ) Predicted Values: M, M, M, M, M, B, M, M, M, M, M, M, M, M, M, M,
 M, M, M, M, M, M, M, M, M, M, M, M, M, B, M, M, M, M, M, M, M, M, M, M, M,
 M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, M, B, M, M, M, M, M, B,
 M, M, M, M, M, M, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B,
 B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, M, B, B, M, B, B,
 B, B, B, B, M, B, B, B, B, B, B, M, B, B, B, B, M, B, B, B, B, B, B, B, M,
 M, B, B, B, B, B, B, B, B, B, B, M, B, M, B, B, B, B, B, B, B, B, B, B, M, B,
 B, B, M, M, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, M, B, B, B,

Dataset:  Breast Cancer Wisconsin (Diagnostic) , 3 fold cross validation st
ratified results:
1 ) Errors: 20 / 187  Percentage Error: 10.7 %
2 ) Errors: 11 / 187  Percentage Error: 5.88 %
3 ) Errors: 18 / 189  Percentage Error: 9.52 %
```

Breast Cancer Wisconsin) k-fold con k = 5:

```
Dataset:  Breast Cancer Wisconsin (Diagnostic) , 5 fold cross validation st
ratified results:
1 ) Errors: 10 / 111  Percentage Error: 9.01 %
2 ) Errors: 11 / 111  Percentage Error: 9.91 %
3 ) Errors: 6 / 111  Percentage Error: 5.41 %
4 ) Errors: 5 / 111  Percentage Error: 4.5 %
5 ) Errors: 8 / 115  Percentage Error: 6.96 %
```

Breast Cancer Wisconsin) k-fold con k = 10:

```
Dataset:  Breast Cancer Wisconsin (Diagnostic) , 10 fold cross validation s
tratified results:
1 ) Errors: 4 / 54  Percentage Error: 7.41 %
2 ) Errors: 5 / 54  Percentage Error: 9.26 %
3 ) Errors: 3 / 54  Percentage Error: 5.56 %
4 ) Errors: 5 / 54  Percentage Error: 9.26 %
5 ) Errors: 0 / 54  Percentage Error: 0.0 %
6 ) Errors: 4 / 54  Percentage Error: 7.41 %
7 ) Errors: 4 / 54  Percentage Error: 7.41 %
8 ) Errors: 5 / 54  Percentage Error: 9.26 %
9 ) Errors: 6 / 54  Percentage Error: 11.11 %
10 ) Errors: 4 / 63  Percentage Error: 6.35 %
```

Test relativi al dataset: Aeberhard,Stefan and Forina,M.. (1991). Wine. UCI Machine Learning Repository. https://doi.org/10.24432/C5PC7J.

Wine) k-fold con k = 1:

```
Dataset:  Wine , 1 fold cross validation stratified results:
1 ) Errors: 0 / 178  Percentage Error: 0.0 %
```

Wine) k-fold con k = 3:

```
Fold  3 ) Original Values: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
Fold  3 ) Predicted Values: 1, 1, 1, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2,
 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,

Dataset:  Wine , 3 fold cross validation stratified results:
1 ) Errors: 11 / 55  Percentage Error: 20.0 %
2 ) Errors: 9 / 55  Percentage Error: 16.36 %
3 ) Errors: 5 / 59  Percentage Error: 8.47 %
```

Wine) k-fold con k = 5:

```
Fold  5 ) Original Values: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
Fold  5 ) Predicted Values: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,

Dataset:  Wine , 5 fold cross validation stratified results:
1 ) Errors: 5 / 31  Percentage Error: 16.13 %
2 ) Errors: 5 / 31  Percentage Error: 16.13 %
3 ) Errors: 4 / 31  Percentage Error: 12.9 %
4 ) Errors: 0 / 31  Percentage Error: 0.0 %
5 ) Errors: 1 / 39  Percentage Error: 2.56 %
```

Wine) k-fold con k = 10:

```
Dataset:  Wine , 10 fold cross validation stratified results:
1 ) Errors: 4 / 13  Percentage Error: 30.77 %
2 ) Errors: 2 / 13  Percentage Error: 15.38 %
3 ) Errors: 3 / 13  Percentage Error: 23.08 %
4 ) Errors: 0 / 13  Percentage Error: 0.0 %
5 ) Errors: 2 / 13  Percentage Error: 15.38 %
6 ) Errors: 2 / 13  Percentage Error: 15.38 %
7 ) Errors: 0 / 13  Percentage Error: 0.0 %
8 ) Errors: 0 / 13  Percentage Error: 0.0 %
9 ) Errors: 1 / 13  Percentage Error: 7.69 %
10 ) Errors: 2 / 31  Percentage Error: 6.45 %
```

Test per dataset: Fisher,R. A.. (1988). Iris. UCI Machine Learning Repository. https://doi.org/10.24432/C56C76.

Iris) k-fold con k = 1

```
Dataset:  Iris , 1 fold cross validation stratified results:
1 ) Errors: 0 / 150  Percentage Error: 0.0 %
```

Iris) k-fold con k = 3

```
Dataset:  Iris , 3 fold cross validation stratified results:
1 ) Errors: 1 / 45  Percentage Error: 2.22 %
2 ) Errors: 3 / 45  Percentage Error: 6.67 %
3 ) Errors: 2 / 51  Percentage Error: 3.92 %
```

Iris) k-fold con k = 5

```
Dataset:  Iris , 5 fold cross validation stratified results:
1 ) Errors: 1 / 27  Percentage Error: 3.7 %
2 ) Errors: 0 / 27  Percentage Error: 0.0 %
3 ) Errors: 2 / 27  Percentage Error: 7.41 %
4 ) Errors: 2 / 27  Percentage Error: 7.41 %
5 ) Errors: 0 / 27  Percentage Error: 0.0 %
```

Iris) k-fold con k = 10

```
Fold  10 ) Original Values: Iris-setosa, Iris-setosa, Iris-setosa, Iris-set
osa, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, Ir
is-virginica, Iris-virginica, Iris-virginica, Iris-virginica,
Fold  10 ) Predicted Values: Iris-setosa, Iris-setosa, Iris-setosa, Iris-se
tosa, Iris-versicolor, Iris-versicolor, Iris-versicolor, Iris-versicolor, I
ris-virginica, Iris-virginica, Iris-virginica, Iris-virginica,

Dataset:  Iris , 10 fold cross validation stratified results:
1 ) Errors: 0 / 12  Percentage Error: 0.0 %
2 ) Errors: 1 / 12  Percentage Error: 8.33 %
3 ) Errors: 0 / 12  Percentage Error: 0.0 %
4 ) Errors: 0 / 12  Percentage Error: 0.0 %
5 ) Errors: 1 / 12  Percentage Error: 8.33 %
6 ) Errors: 1 / 12  Percentage Error: 8.33 %
7 ) Errors: 1 / 12  Percentage Error: 8.33 %
8 ) Errors: 0 / 12  Percentage Error: 0.0 %
9 ) Errors: 0 / 12  Percentage Error: 0.0 %
10 ) Errors: 0 / 12  Percentage Error: 0.0 %
```

Si nota che per k = 1 l'accuratezza è del 100%, dato che il validation set coincide con il training set, che è l'intero dataset. Tuttavia come sappiamo, addestrare un albero decisione sull'intero dataset, in modo che sia estremamente accurato su questo, può causare una peggiore generalizzazione nella valutazione di esempi al di fuori del training set (fenomeno noto come *overfitting*). Per questo usiamo il *k-fold cross validation*, in modo da dividere l'insieme degli esempi in un insieme su cui addestrare l'albero di decisione ed un altro su cui testarne l'accuratezza nella predizione.
Dai test si nota che gli errori diminuiscono all'aumentare di k (anche se non è così evidente, dato il numero limitato di istanze dei dataset), questo perché per k grande aumenta anche la frazione di dataset usata come training set, mentre per k piccolo viene utilizzata una frazione minore del dataset per l'apprendimento.