Algoritmo Random Forest

Pierazzini Tommaso

Aprile 2018

Nella prima parte di questo esercizio si implementa l'algoritmo Random Forest descritto in (Breiman 2001). Nella seconda parte, si applica il codice sviluppato ad almeno 5 data sets scelti a piacere tra quelli elencati in Tabella 1, cercando di verificare i risultati sperimentali riportati in Tabella 2 (limitatamente a random forest).

1 Random Forest

Una Random Forest è un *ensamble learning*, formato da un insieme di alberi di decisione creati sullo stesso dataset, ognuna delle volte randomizzato. La foresta farà passare sotto ciascuno degli alberi (diversi tra loro) che la compongono, un determinato esempio e ognuno di questi esprimerà il suo "voto" di classificazione. Quindi il singolo esempio verrà classificato con il risultato più votato dall'insieme di tutti gli alberi nella foresta.
Questo procedimento fa si che la Random Forest sia più precisa di un singolo albero di decisione in un algoritmo di learning.

2 Data sets utilizzati

I cinque data sets utilizzati, reperiti da UCI, sono i seguenti:

- **Liver**: *train size: 345, inputs: 6, class: 2*
- **Ecoli**: *train size: 336, inputs: 7, class: 8*
- **Vehicle silhouettes**: *train size: 846, inputs: 18, class: 4*
- **Ionosphere**: *train size: 351, inputs: 34, class: 2*
- **Breast Cancer**: *train size: 699, inputs: 9, class: 2*

I seguenti dataset sono costituiti da insiemi di valori continui.
E' quindi importante sottolineare che la soluzione riportata negli script Python di questo progetto è valida per questo tipo di dataset.

3 Documentazione del codice

Il progetto è formato dai seguenti moduli Python:

- ***DataSet.py***

Questo modulo serve a convertire il dataset dal formato *.txt* ottenuto da UCI in un altro con il quale sarà possibile lavorare con il linguaggio di programmazione.
Come indicato nel file README in allegato, tale codice fa riferimento al seguente repository pubblico.

- ***DecisionTree.py***

Serve per creare l'albero di decisione: contiene funzioni per l'inizializzazione, l'aggiunta di sottoalberi a destra/sinistra di un albero e la funzione *call* che, dato un esempio restituirà la *decisione* presa dall'albero navigandolo fino alla foglia.
E' presente anche la classe Leaf, per la creazione di esse.

- ***DecisionTreeLearning.py***

Nel seguente modulo viene implementata la modalità di apprendimento di un singolo albero di decisione, facendo riferimento al metodo descritto in Russell & Norvig 18.3 e ancora una volta al repository pubblico, effettuando delle modifiche necessarie ad alcune parti del codice.
E' presente la funzione *getListValuesForAttribute* che ritorna una lista di liste contenenti i valori di tutti gli esempi, divisi per attributo, non ripetuti. Questa funzione è necessaria per l'apprendimento dei singoli alberi nel trovare gli attributi migliori sui quali eseguire lo *splitting*, determinando anche il valore di soglia con il quale si divideranno le *decisioni*. Questo procedimento è strettamente necessario per via dell'utilizzo di dataset a valori continui.
La funzione *chooseAttribute* ricerca l'attributo migliore sul quale eseguire lo splitting. Per farlo prende la lista dei valori per ogni attributi, dalla funzione sopra descritta, e per ogni attributo calcola l'*information gain*(nel modo descritto in Russell & Norvig 18.3, utilizzando l'*entropia*) più alto tra tutti i suoi valori degli esempi. Fatto questo passa all'attributo successivo ed esegue lo stesso compito. Infine trova l'attributo che rende il guadagno più alto (mostImportanceA) ed imposta la soglia, ovvero il valore del relativo attributo che produce quel guadagno.
La funzione *splittingOnThreshold* separa gli esempi del dataset in due parti, a seconda se il valore di uno specifico attributo in un esempio sia minore/uguale oppure maggiore della soglia (*threshold*) ricavata come descritto in precedenza.
La funzione *allSameClass* controlla se l'insieme degli esempi è puro, ovvero se tutti questi appartengono alla stessa classe.

- ***Helper.py***

Contiene molte funzioni utilizzate dai moduli *RandomForest.py* e *DecisionTreeLearning.py*.
In particolare troviamo la funzione *setForDataset* che legge dal file 'dataset'.txt tutte le righe e le divide in examples, attributes ed input, per poi passarli al modulo *DataSet.py* che creerà la struttura dati per poter lavorare più facilmente con il dataset creato.
Possiamo trovare anche la funzione *divideDataSet* che, preso in input il dataset, lo separa in due parti (Train 90% - Test 10%, come suggerito in Breiman 2001) e crea le due strutture necessarie.
Importante anche sottolineare la funzione *shuffleDataTrain* necessaria a randomizzare la creazione degli alberi nella foresta: questa prende il dataset, mischia gli esempi ed infine ne prende un sottoinsieme, che viene convertito in un oggetto di tipo DataSet, per poi essere passato nell'operazione di creazione dell'albero da inserire nella foresta.

- ***RandomForest.py***

Questo è il modulo principale del progetto. Importa i precedenti due moduli.
Viene implementata la funzione *RandomForest* che andrà a dividere il dataset in train e test, usando il train set per la creazione di N alberi, raccolti poi tutti all'interno di una foresta. Viene poi lanciato il metodo *prediction* per stampare la predizione che ha ottenuto più voti, da tutti gli

alberi, per ogni esempio del test set. Viene infine stampato il test sets error.
Nel seguente script è implementata anche la funzione *mostVotes* che crea una lista contenente a sua volta tante liste quanti sono gli esempi da testare, dove in ognuna delle quali sono presenti i voti espressi da tutti gli alberi della foresta per ciascun esempio. Da questa struttura viene infine calcolato il voto più ricorrente per le singole classificazioni.
Nella parte finale dello script viene implementato un meccanismo per la scelta, da parte dell'utente, del dataset che desidera testare, attraverso un'interazione da console.

4 Esempio di funzionamento

Ai fini di riprodurre i risultati riportati si devono seguire i seguenti passaggi:

- ***1) [opzionale] Impostare il numero di alberi che formeranno la foresta***

Aprire il modulo *RandomForest.py* e, all'ultima riga di codice, se l'utente lo desidera può variare il numero di alberi che formeranno la foresta semplicemente modificando il numero passato all'interno della funzione RandomForest chiamata.
Se per esempio volessimo creare una foresta di 30 alberi:

```
RandomForest(fileDataset, 30)  <-- Modificare questo valore
```

- ***2) Eseguire il modulo RandomForest.py***

Aprire il modulo *RandomForest.py* e lanciare la sua esecuzione.
Comparirà sulla console la possibilità di scelta del dataset da testare: digitare il codice relativo e premere Invio.
Partirà l'esecuzione del codice implementato, che terminerà con l'output dei risultati ottenuti.
Se per esempio volessi utilizzare il dataset *Liver*, formando una foresta di 20 alberi, basterà digitare il codice 1 ed otterremo il seguente output:

```
Select dataset to test:
1. Liver
2. Ecoli
3. Breast Cancer
4. Vehicle
5. Ionosphere
--> type code here:1

Dataset in use:  liver.txt
Dataset divided into Train (90%) and Test(10%)

Creating forest of 20 trees...
Creating tree number: 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 , 11 , 12 , 13 , 14 ,
15 , 16 , 17 , 18 , 19 ,

Data Test classification:
['1', '1', '1', '1', '2', '1', '2', '1', '2', '2', '1', '2', '2', '2', '2', '2', '2',
'1', '1', '1', '1', '1', '1', '2', '2', '2', '2', '1', '1', '2', '2', '1', '2', '2', '1']
Random Forest predicter classification:
['1', '2', '2', '2', '2', '2', '1', '1', '1', '2', '2', '1', '1', '2', '2', '2', '2',
'1', '1', '1', '1', '2', '1', '2', '1', '2', '2', '1', '1', '2', '2', '1', '2', '1', '1']

Test set errors: 34.2857142857 %
```

N.B. L'esecuzione dei dataset *Vehicle* e *Ionosphere* richiede un tempo di attesa maggiore per l'elevata quantità di volte in cui viene eseguita la fase di *splitting* nella creazione di ogni singolo albero di decisione.
In entrambi i casi la creazione di ogni singolo albero decisionale richiede circa venti secondi; di conseguenza a seconda della foresta creata impiegherà un tempo di esecuzione maggiore rispetto agli altri dataset.
E' possibile diminuire i tempi di attesa diminuendo a sua volta il numero di alberi che formeranno la foresta, peccando però sulla precisione della predizione di classificazione.

5 Risultati ottenuti

Nella seconda parte dell'esercitazione dovevano essere confrontate le percentuali di errore, commesse sul Test set dalla foresta, tra Tabella 2 (Breiman 2001) ed i risultati ottenuti.

Table 1. Test set errors (%).

	Tabella2	20 Alberi	50 Alberi
Liver	24.7	34.33	30.6
Ecoli	13.0	17.57	13.94
Vehicle	26.4	27.88	27.97
Ionosphere	7.5	11.42	8.32
Cancer	2.7	12.14	5.67

Le percentuali riportate nella colonna "Tabella2" sono ottenute su foreste di 100 alberi.
Le percentuali riportate nelle colonne 20 e 50 Alberi, sono ricavate dalla media di 10 esecuzioni.
Dall'esecuzione di singole prove effettuate su foreste di 100 alberi, per ognuno dei 5 dataset, ho riscontrato che le percentuali di errore sul test tendono a stabilizzarsi con quelle presenti nella terza colonna (50 Alberi).
Risultano comunque dei risultati molto vicini a quelli da confrontare.

6 Conclusioni

In *Table 1* è possibile notare come al crescere del numero di alberi che compongono la foresta, nella seconda colonna 20 e nella terza 50, la percentuale di errore sulla parte di dataset fornita come test, si avvicini sempre più ai dati della prima colonna, ovvero quelli riportati nella Tabella 2 in (Breiman 2001), ottenuti su 100 alberi decisionali.
Possiamo notare che già creando una foresta di 50 alberi si possono ottenere dei buoni risultati.
Questo risultato conferma la teoria, la quale afferma che l'apprendimento eseguito sulla cooperazione di molteplici alberi di decisione è sicuramente più affidabile di quello ricavato da uno singolo.
Infatti, nel caso in cui un albero di decisione sbagliasse a classificare un esempio, questo fatto potrebbe essere sopperito dai voti degli altri alberi all'interno della foresta.

N.B.
Il moduli Python di cui viene descritto il funzionamento in questa relazione sono disponibili al seguente repository pubblico sulla piattaforma GitHub