Confronto tra la mia implementazione di Random Forest e quella presente nella libreria di sci-kitlearn.

Elaborato Intelligenza Artificiale

Gianni Moretti

A.A. 2022-2023

UNIVERSITA' DEGLI STUDI DI FIRENZE
Facolta di Ingegneria
Corso di Laurea in Ingegneria Informatica

1 Introduzione

Per il mio progetto di Intelligenza Artificiale ho implementato l'algoritmo top-down per l'apprendimento di alberi di decisione descritto in classe ed in R&N 2010 e l'ho modificato per implementare Random Forest come esposto brevemente in classe ed in maggiore dettaglio in (Hastie et al. 2017, 15.2). L'algoritmo è implementato solo per lo scopo della classificazione. Ho verificato il funzionamento dell'algoritmo sviluppato comparando i risultati con quelli ottenuti tramite l'implementazione disponibile in scikit-learn, su almeno tre data sets scelti a piacere dall'UCI Machine Learning Repository. Infine ho riportato i risultati dei confronti delle due implementazioni.

2 Obiettivi

L'obbiettivo che mi sono posto è quello di cercare di creare un algoritmo che abbia prestazioni simili a quelle della libreria di sklearn, utilizzando gli stessi parametri e metodi.

3 Implementazione

Di seguito spiego il mio approccio al progetto e come ho implementato i vari algoritmi.

3.1 Decision Tree Classifier

Per prima cosa ho iniziato a implementare la mia versione di Decision Tree, inizialmente ho seguito l'algortitmo descritto in R&N. Successivamente ho modificato l'algoritmo in modo da farlo comportare il più possibile come quello presente nella libreria di sklearn, cercando di replicare tutti i funzionamenti e parametri. Durante l'implementazione, tuttavia, è emersa una differenza sostanziale: il decision tree presente nella libreria di sklearn accetta solo dati di tipo numerico. Per questo motivo ho deciso di differenziare la mia implementazione aggiungendo un parametro chiamato "categorical column", che rappresenta la lista degli indici degli attributi che devono essere trattati come variabili categoriche quando si effettua una decisione sull'albero. L'algoritmo offre la possibilità di specificare un numero che rappresenta il numero di attributi da considerare quando si decide quale variabile utilizzare per la divisione dei sample. Se il numero specificato è inferiore al numero totale di attributi, gli attributi vengono scelti casualmente con distribuzione uniforme tra quelli disponibili. Per verificare le differenze tra i due alberi, ho creato una funzione per visualizzare il mio albero utilizzando la libreria graphviz. In questo modo ho potuto confrontare i due alberi e individuare eventuali differenze (che non si presentano nel caso di database contenenti solo dati numerici e utilizzando gli stessi parametri di input). Di seguito gli esempi di alberi visualizzati con graphviz.

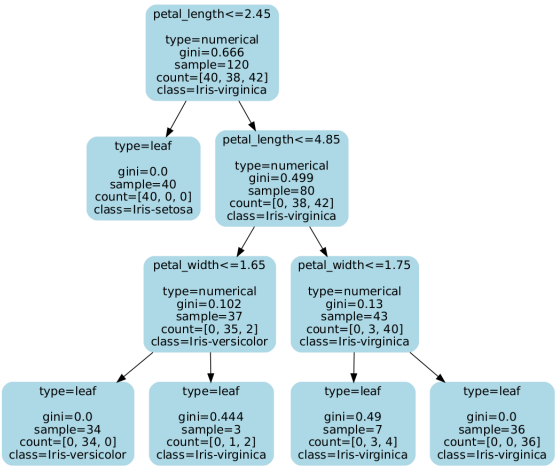

Figure 1: La mia implementazione decision tree classifier nel database Iris con depth = 3

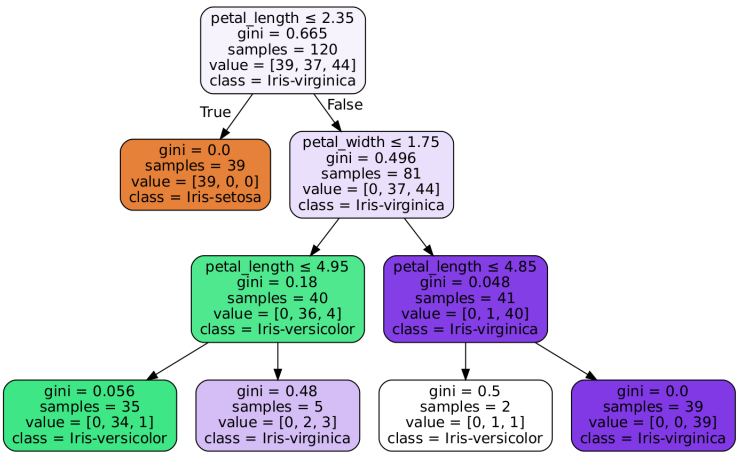

Figure 2: SKLearn decision tree classifier nel database Iris con depth = 3

Questo invece è un esempio di albero costruito a partire da dati di tipo misto

(numerici e categorici).

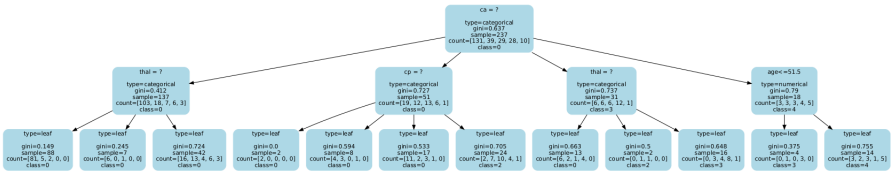

Figure 3: La mia implementazione decision tree classifier nel database Heart Disease con depth = 3

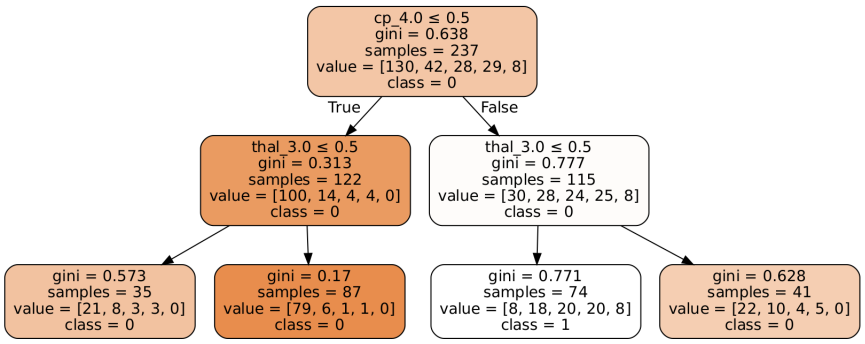

Figure 4: SKLearn decision tree classifier nel database Heart Disease con depth = 3

3.2 Random Forest

Per quanto riguarda l'algoritmo Random Forest, ho cercato di replicare il maggior numero possibile di parametri dell'implementazione presente in sklearn. Nella fase di training dell'algoritmo, viene utilizzata la tecnica del bootstrap .632, che ho implementato all'interno della classe RandomForestClassifier con il nome di "resample".

3.3 Database

I tre database scelti dal sito UCI Machine Learning Repository sono:

- Heart Disease (versione con soli 14 attributi)
- Covertype (versione con 54 attributi)
- Wine Quality (solo vini binachi)

I database sono stati modificati in modo che tutte le righe contenenti dei valori mancanti vengano cancellate. Inoltre, nel caso degli algoritmi di sklearn, che utilizzano solo variabili numeriche, i database che contengono attributi categorici sono stati modificati utilizzando la tecnica del One-Hot-Encoding.

3.3.1 Data Visualizzation

Prima di utilizzare i database per l'apprendimento, ho creato una piccola libreria Python in cui ho implementato alcuni metodi che utilizzo per controllare la qualità dei dati presenti nei database. Utilizzando lo script datavisualization.py e cambiando i parametri con i dati del database che si vuole visualizzare, è possibile ottenere informazioni utili sul database.

3.3.2 File "Database.txt"

All'interno di questo file sono presenti tutti i dati relativi ai tre database scelti.

4 Risultati

Di seguito riporto i grafici che confrontano le accuratezze nei vari database. Per riprodurre i seguenti risultati ho impostato i seguenti parametri fissi per ogni database:

- `n_estimetors` = 100
- criterion = 'gini'
- `min_samples_split` = 2

Mentre gli altri due parametri (`max_features`, `max_depth`) cambiano in base al database. `max_features` viene impostato come l'intero inferiore della radice del numero di attributi totali, mentre per `max_depth` ho deciso il valore da dargli sulla base di varie prove effettuate precedentemente.

4.1 Heart Disease

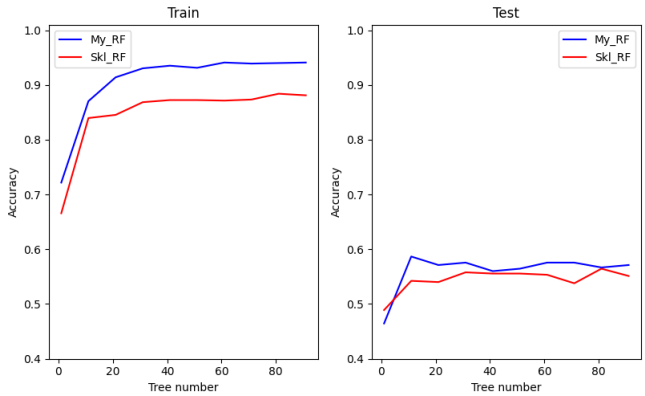

Figure 5: Accuratezza al variare del numero di sample con `max_features` = 4, `max_depth` = 5.

4.2 Covertype

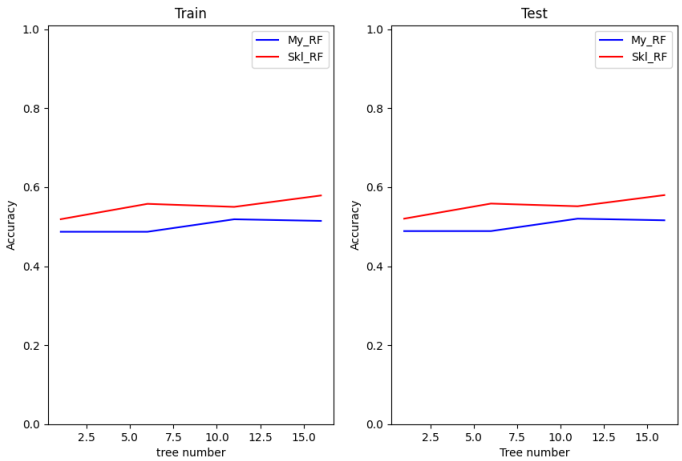

Figure 6: Accuratezza al variare del numero di tree con `max_features` = 7, `max_depth` = 4.

4.3 Wine Quality

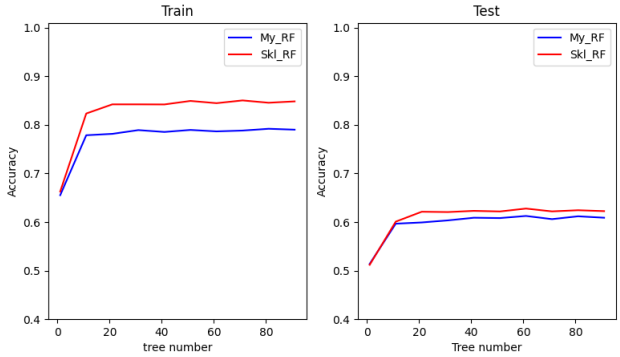

Figure 7: Accuratezza al variare del numero di sample con `max_features` = 3, `max_depth` = 10.

4.4 Conclusioni

Dall'analisi dei grafici, è evidente che l'accuratezza migliora all'aumentare del numero di stimatori utilizzati. Tuttavia, è probabile che l'utilizzo di parametri ottimizzati per gli algoritmi possa ulteriormente migliorare i risultati. È interessante notare che l'implementazione da me realizzata (rappresentata dal colore blu nei grafici) non si discosta significativamente dall'implementazione di sklearn (rappresentata in rosso). In particolare, nel caso del database Heart Disease, si osserva un comportamento particolarmente diverso, che probabilmente è attribuibile al modo diverso di gestire le variabili categoriche. Nei due altri database, infatti, vengono considerate solo variabili numeriche. Nella situazione specifica del database Heart Disease, la mia implementazione ottiene un'accuratezza migliore sia nel training set che nel test set. Tuttavia, in base alle prove effettuate, sembra che questo comportamento non segua il pattern classico, anzi, spesso sembra portare a un aumento dell'overfitting.

4.4.1 Performance

Infine, è importante evidenziare una differenza significativa tra i due algoritmi in termini di tempo di esecuzione e utilizzo della memoria. Inizialmente, con database di dimensioni ridotte come iris, heart disease e wine quality, non mi ero accorto delle differenze di prestazioni rispetto all'implementazione di sklearn. Tuttavia, con il database covertype, la differenza è diventata evidente. Il mio algoritmo richiedeva circa 500 secondi solo per creare il primo albero, mentre sklearn riusciva a creare l'intera foresta in meno di 1 secondo. Dopo un'analisi più approfondita del codice della libreria sklearn, ho apportato alcune modifiche che mi hanno consentito di creare l'intera foresta in circa 15 secondi. Nonostante ciò, la mia implementazione rimane comunque lenta, ma può essere eseguita in un tempo ragionevole.

5 Fonti

[R&N] Russell, S. J., and Norvig, P. (2020). Artificial Intelligence: A Modern Approach. Pearson education limited.

T. Hastie, R. Tibshirani, and J. Friedman. The Elements of Statistical Learning. Data Mining, Inference, and Prediction. 2nd edition. Springer, 2009