Confronto tra metodi basati su alberi decisionali: Un'applicazione sui dati Boston Housing

Gaetano Tedesco

1 Introduzione e Obiettivo

L'obiettivo dell'applicazione è confrontare tra loro vari metodi di apprendimento basati su alberi decisionali. In particolare, si è confrontato un semplice modello ad albero di regressione con versioni *enseble* di questo in merito a performance e interpretabilità dei risultati.

La valutazione dei metodi si è svolta sul dataset *"Boston Housing"* del 1970, spesso utilizzato come *benchmark* per valutare algoritmi di apprendimento automatico su problemi di regressione. Per condurre l'analisi in modo replicabile è stato creato lo script python `AutoLib.py` contenente tutte le funzioni e le classi utilizzate.

2 Metodi basati su Alberi Decisionali

2.1 Alberi di Regressione (CART)

I modelli ad albero partizionano lo spazio delle covariate in regioni distinte, adattando un modello semplice su ognuna di queste, in modo da ottenere regioni di previsione costante. Per semplicità e facilità d'interpretazione si utilizzano *"tagli"* paralleli agli assi andando a formare regioni p-dimensionali rettangolari.

L'algoritmo di stima per alberi di regressione è conosciuto con il nome di *recursive binary splitting* e si compone di due step:

1. Dividere lo spazio delle covariate in J regioni non sovrapposte $R_1, R_2, ..., R_J$ tali che sia minima la RSS data da: $\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$.
2. Per ogni osservazione che cade nella regione R_j assegnare la stessa previsione, ovvero la media della risposta delle osservazioni interne alla j-esima partizione: $\hat{f}(x) = \hat{y}_j = ave(y_i | x_i \in R_m)$.

Nella maggior parte dei casi è computazionalmente infattibile calcolare ogni possibile partizione dello spazio in J *"scatole"* e si prosegue con un approccio *greedy* dall'alto verso il basso.

Si parte con tutte le osservazioni appartenenti ad un'unica regione, si considera una variabile X_j come asse e un valore s del suo supporto come punto di *split*, tali che la partizione nelle due regioni $R_1(j, s) = \{X | X_j \leq s\}$ e $R_2(j, s) = \{X | X_j > s\}$ porti alla maggior riduzione possibile in termini di RSS allo step corrente piuttosto che in uno step futuro.

Tra tutti i predittore $X_1, X_2, ..., X_p$ e tutti i possibili valori s per ognuno di essi, viene selezionato, dunque, il predittore e il rispettivo valore che minimizzano l'equazione: $\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$. L'operazione si ripete in modo da dividere ulteriormente i dati minimizzando la RSS su tutte le regioni risultanti, finchè non viene raggiunta una certa condizione d'arresto.

2.1.1 Potatura

Un processo come quello appena descritto è particolarmente avvezzo a cadere nella trappola dell'*overfitting*, ovvero ad adattarsi così bene ai dati da non riuscire a generalizzare su dati esterni a quelli con cui è stato addestrato. L'albero risultante potrebbe, infatti, essere così complesso da cogliere fluttuazioni locali e casuali come *pattern* sistematici del processo che ha generato i dati. Al contrario, alberi troppo piccoli e semplici potrebbero non essere in grado di catturare importanti strutture del fenomeno oggetto di studio.

In questo senso la dimensione dell'albero rappresenta un parametro di *tuning* che ci permette di governare il *trade-off* tra la complessità generale del modello (da intendere come precisione dello stesso nel prevedere il fenomeno) e la variabilità delle stime. La strategia adottata è quella di far crescere un albero molto grande T_0 fino a che non raggiungiamo, ad esempio, un numero minimo di unità per nodo e successivamente *potare* l'albero in modo da trovare il *sotto-albero* ottimale; tale procedura è conosciuta con il nome di *cost-complexity pruning*.

Definiamo sotto-albero ogni albero $T \subset T_0$ che può essere ottenuto da T_0 collassando un qualsiasi numero di nodi interni e consideriamo la sequenza di alberi indicizzata dal parametro (non negativo) α.

Per ogni valore di α abbiamo un albero $T \subset T_0$ tale che minimizza: $\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T|$.

Ora, preso $|T|$ il numero di foglie dell'albero T, R_m la partizione corrispondente all'm-esima foglia e $\hat{y}_{R_m}$ la previsione associata a tale partizione è facile notare che l'equazione che regola la potatura altro non è che una versione penalizzata della RSS. Per $\alpha = 0$ l'albero ottimale sarà di fatto l'albero completo T_0 che minimizza l'errore sui dati di addestramento e al crescere di α la RSS penalizzata tenderà ad essere minima per alberi più piccoli, fino eventualmente a raggiungere l'albero *nullo*.

Si può dimostrare che la procedura di potatura è *annidata* e che per ogni valore di α si ha uno ed un solo albero ottimo. All'atto pratico, dunque, calcoliamo l'intero percorso di potatura e selezioniamo il valore ottimale del parametro di complessità tramite *cross-validation* prendendo quello associato all'albero che minimizza l'errore di validazione.

2.1.2 Vantaggi e Svantaggi

I modelli ad albero sono ottimi per problemi di regressione in cui la relazione tra la risposta e le covariate non è lineare; essi permettono, infatti, di approssimare una qualsiasi funzione liscia attraverso una *funzione a scalini*. In secondo luogo, tali modelli possono essere visualizzati graficamente e sono di facile interpretazione anche per non esperti (specialmente se piccoli).

L'assenza di qualunque assunzione distributiva per la risposta li rende particolamente flessibili (a patto di possedere dati sufficienti) ed è proprio per questo motivo che alcuni alberi decisionali possono dimostrarsi particolarmente *non-robusti*; in altre parole, cambiamenti anche minimi nei dati possono portare a grandi cambiamenti nelle stime finali fornite dal modello, sinonimo di un'*alta variabilità generale* delle stime prodotte da questo metodo.

2.2 Metodi *ensemble*

I metodi di **apprendimento d'insieme** si basano sulla combinazione di molti modelli più semplici (a cui ci si rifersce come *weak learners*) al fine di ottenere una migliore prestazione predittiva rispetto a quella ottenuta dagli stessi applicati singolarmente. Tra i metodi che utilizzano gli alberi di regressione come blocchi costituenti troviamo il *bagging*, *random forest* e il *boosting*.

2.2.1 Bagging

Boostrap aggregating o *bagging* è una procedura generale atta a ridurre la varianza di metodi di apprendimento automatico. L'idea dietro al metodo è che dato un campione di n osservazioni indipendenti $Z_1, Z_2, ..., Z_n$, ognuna con varianza σ^2, la varianza della media campionaria delle osservazioni $\bar{Z}$, data da σ^2/n, sia minore della varianza delle singole osservazioni.

Un modo per ridurre la varianza delle stime è quello di prendere una moltitudine di insiemi di stima (*training-set*) dalla popolazione, costruire un modello di previsione su ciascun insieme e prendere come stima finale la media delle previsioni dei singoli modelli. Poichè normalmente si dispone di un unico campione si utilizza il *bootstrap* per generare molteplici training-set da questo.

Nella pratica vengono campionati B differenti insiemi di stima, per ognuno di questi si allena un singolo modello in modo da ottenere la stima $\hat{f}^{*b}(x)$ e si prende come previsione finale la media di tutte le previsioni:

$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x)$.

Ora, dal momento che la "debolezza" più grande degli alberi di decisione è proprio l'alta variabilità delle stime, tale metodo risulta particolarmente utile nel migliorare le performance di questi modelli. Vengono adattati B alberi di regressione *indipendentemente* sui B training-set; gli alberi vengono lasciati liberi di crescere in profondità, senza alcuna potatura, e questo fa si che ogni albero produca stime poco distorte ma con alta variabilità (fortemente dipendenti dai dati su cui è stato allenato), la media delle stime dei B alberi riduce, poi, tale varianza.

Il miglioramento nell'accuratezza delle previsioni avviene a discapito dell'interpretabilità; l'elevata complessità dei singoli alberi, infatti, rende estremamente complicata l'interpretazione dei risultati. Una soluzione a questo problema può essere ottenuta andando a valutare per ogni variabile del modello la sua importanza generale, calcolata come la media della diminuzione totale di RSS derivante dagli *splits* effettuati su tale variabile. Un valore elevato di tale metrica indica che il predittore in questione spiega gran parte della variabilità della risposta.

2.2.2 Random Forest

Il *random forest* costituisce un miglioramento rispetto al metodo precedente. Rappresenta una soluzione per *decorrelare* i singoli alberi decisionali e, quindi, diminuire ulteriormente la varianza di stima.

Anche in questo caso si adatta un albero per ognuno dei B insiemi di stima ottenuti tramite bootstrap, tuttavia, nell'adattare tali alberi, ogni volta che una partizione viene considerata, si estrae un nuovo campione casuale (proprio da qui deriva l'eccezione *random* nel nome del metodo) di m predittori, rispetto ai p disponibili, come candidati per lo split.

Così facendo il metodo evita che le stime risultanti dai singoli alberi siano correlate a causa della struttura simile di questi (a livello di predittori) massimizzando, quindi, la riduzione di varianza operata dalla media sui B modelli.

La differenza tra i metodi risiede nella scelta della dimensione m del sottoinsieme di predittori estratti ad ogni iterazione, scelte comuni sono: $m = \sqrt{p}$, $m = \log_2 p$ o $m = p/3$; con $m = p$ stiamo nuovamente utilizzando un approccio bagging.

2.2.3 Boosting

Come il bagging anche il *boosting* è una procedura generale che può essere applicata per migliorare le performance di molti metodi di regressione. Se nel bagging gli alberi vengono cresciuti in modo indipendente gli uni dagli altri (*parallelamente* potremmo dire), nel boosting questo avviene in maniera *sequenziale* e ogni albero è cresciuto su una versione dei dati originali modificata sfruttando le informazioni degli alberi stimati precedentemente. Invece di adattare ai dati un unico grande albero decisionale, che possa potenzialmente fare overfitting (tecnica descritta in gergo come *hard fitting*), l'approccio boosting si basa su un *apprendimento lento*.

In pratica, facciamo crescere un nuovo albero di regressione usando i *residui* del modello adattato all'iterazione precedente come risposta, piuttosto che i valori della variabile Y. Successivamente aggiungiamo il nuovo albero decisionale al resto degli alberi già adattati in modo da aggiornare i residui e ripetiamo l'operazione. Ognuno degli alberi può essere piuttosto piccolo, ammettendo solo pochi nodi foglia, in modo da andare a migliorare $\hat{f}$ lentamente nelle aree dove la stima è più distante dalla realtà. L'*aggiunta* del nuovo albero di regressione al modello stimato fino a quel momento viene ulteriormente controllata dal parametro di regolarizzazione (*shrinkage*) λ che permette a più alberi, possibilmente con forme diverse, di *attaccare* i residui.

L'output del modello finale risulta del tipo: $\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$

2.2.4 Ottimizzazione degli iper-parametri

Se nell'albero di regressione semplice l'unico parametro che regola il compromesso tra complessità del modello e accuratezza delle previsioni viene *"imparato"* dai dati, lo stesso non può essere detto per i metodi di apprendimento d'insieme.

Tali metodi presentano, infatti, una serie di parametri di controllo, che pur non potendo essere stimati dai dati, se ottimizzati, portano a miglioraramenti più o meno notevoli delle performance.

In particolare abbiamo:

- B: numero di replicazioni bootstrap per i metodi bagging e random forest; numero di alberi cresciuti sequenzialmente per l'approccio boosting. Se per i primi due metodi valori elevati di tale parametro non portano un sovra-adattamento del modello ai dati di training, lo stesso non si può dire per il boosting. Il valore idoneo va selezionato attraverso convalida incrociata;
- m: dimensione del sottoinsieme di predittori estratti come candidati per ogni possibile partizione durante la crescita dell'albero (utile a selezionare il metodo più idoneo tra bagging e random forest);
- λ: parametro di regolarizzazione del boosting, controlla il *"peso"* di ogni nuovo albero aggiunto all'insieme, più piccolo è il valore meno la stima risultante dall'albero corrente influirà sull'adattamento generale del metodo;
- d: numero di split di ogni albero aggiunto all'insieme. Tale parametro concorre a controllare la complessità del modello finale; anche in questo caso il parametro è utile principalemente al boosting, dove controlla l'ordine d'interazione tra le variabili, stabilendo il numero massimo di variabili per ogni singolo albero.

3 Analisi e Confronto in Python

L'implementazione in python delle funzioni e delle classi utili all'analisi e il confronto dei metodi è contenuta nel file `AutoLib.py` (per una miglore riproduzione dell'analisi viene allegato anche un notebook `BostonHousing.ipynb`). Lo script fa uso delle librerie NUMPY e PANDAS per calcoli matriciali e manipolazioni dei dati, MATPLOTLIB e SEABORN per la parte grafica e SCIKIT-LEARN per le implementazioni di modelli e metodi di apprendimento automatico. L'idea dietro alle funzioni e alle classi definite è quella di rendere l'analisi dei dati e il confronto tra i metodi replicabile. (Per una spiegazione dettagliata dell'implementazione in python si rimanda alla documentazione del codice.)

3.1 Presentazione Dati e Analisi Esplorativa

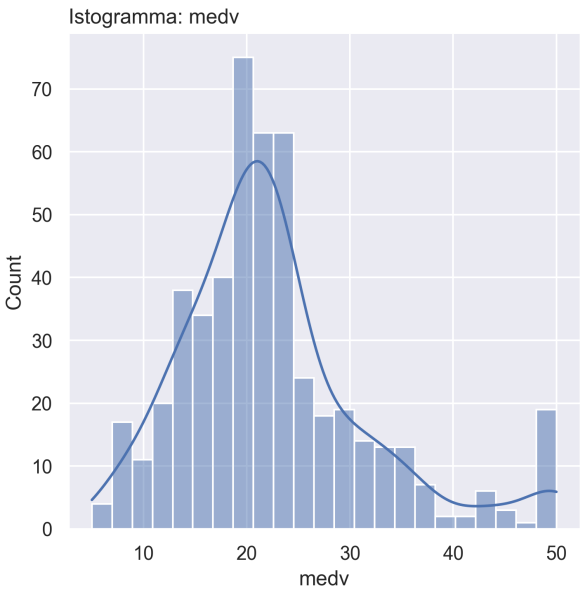
Il dataset "Boston Housing" è un dataset classico nel campo dell'apprendimento automatico e dell'econometria; introdotto per la prima volta nel 1978 da Harrison e Rubinfeld contiene informazioni su 506 abitazioni della Boston Standard Metropolitan Statistical Area, Massachusetts, negli Stati Uniti.

Il dataset è stato creato per esaminare la relazione tra il valore mediano delle abitazioni e una serie di variabili descrittive, tra cui: il tasso di criminalità per persona in un quartiere (**crim**), la proporzione di terreno residenziale assegnato per lotti di oltre 25.000 piedi quadrati (**zn**), la proporzione di ettari dedicati alle attività commerciali non al dettaglio per città (**indus**), l'informazione se la proprietà si affaccia o meno sul fiume Charles (**chas**), la concentrazione di ossidi di azoto in parti per 10 milioni (**nox**), il numero medio di stanze per abitazione (**rm**), la proporzione di unità occupate costruite prima del 1940 (**age**), le distanze ponderate dai cinque centri di impiego di Boston (**dis**), l'indice di accessibilità alle strade principali (**rad**), il tasso di imposta sulla proprietà a valore pieno per 10.000 dollari (**tax**), il rapporto studenti-insegnanti per quartiere (**ptratio**) e la percentuale di persone a basso reddito (**lstat**).

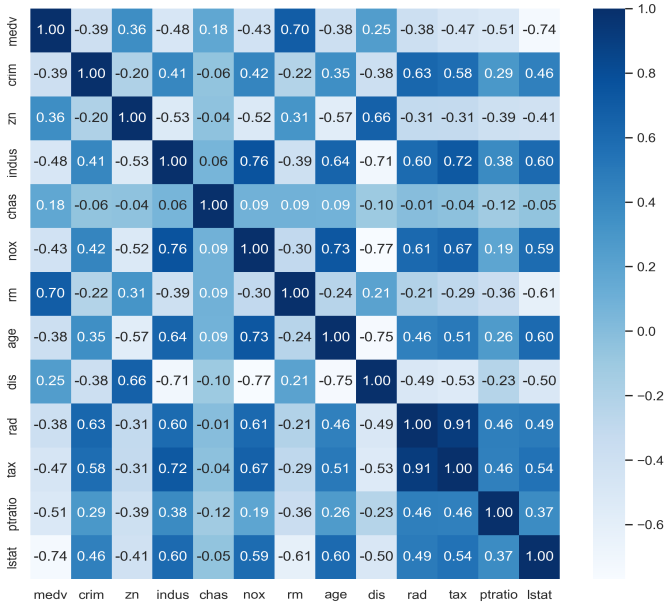
La variabile presa come risposta in questa applicazione è il valore mediano delle abitazioni (**medv**), espresso in migliaia di dollari.

I dati vengono inizialmente caricati in memoria con la funzione `read_data` che, ricevendo come parametri il nome del file e il percorso della cartella in cui questo è contenuto, permette la lettura di dati dal formato csv e la conversione di questi in un oggetto `DataFrame`.

Successivamente, attraverso la funzione `auto_eda` viene evidenziato che la risposta ha una distribuzione bimodale, asimmetrica a destra, dove la maggior parte delle osservazioni è concentrata attorno alla media e una quantità ridotta, ma comunque non trascurabile, presenta un valore notevolmente più elevato. Questo ci suggerisce che la variabile risposta non è normalmente distribuita e che, quindi, non sarebbe del tutto corretto l'utilizzo di modelli lineari che richiedono la normalità dei dati sottostanti. Impostando i parametri `plot_density` e `cor_plot` entrambi uguali a `True` si ottengono i seguenti grafici:



(a) Distribuzione della risposta **'medv'**



(b) Matrice di correlazione

Le variabili più correlate sono: **lstat, rm** e **ptratio**.

Tra queste l'unica ad avere una correlazione positiva è **rm**, mentre **lstat** e **ptratio**, correlate positivamente tra loro, correlano negativamente con la risposta.

Dall'analisi si conclude che la forma che meglio esprime la relazione tra la risposta e le variabili sopra citate è **non lineare** e questo, in associazione con la natura della variabile risposta, suggerisce la necessità di una famiglia di modelli più flessibile, come gli alberi di regressione.

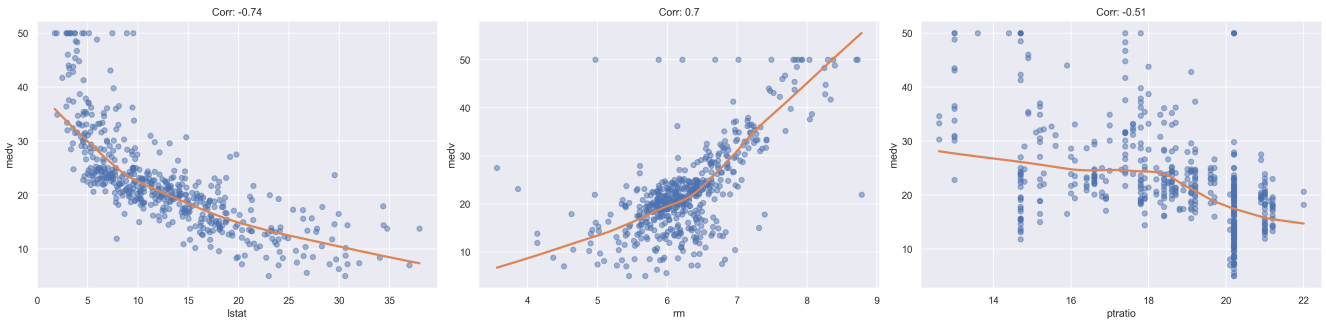


Figura 2: Variabili maggiormente correlate con la risposta.

3.2 Modelli e Metodi

Prima di procedere all'adattamento dei vari modelli si utilizza la funzione `model_matrix` per *aggiustare* i dati e ottenere la matrice delle covariate (in pieno stile `R`) così come richiesto dalle implementazioni `sklearn` dei vari stimatori. Successivamente il dataset viene diviso in *training e test set* utilizzando un *wrapper* della funzione `train_test_split`. Il 25% dei dati viene destinato all'insieme di verifica, passando il valore alla funzione come numero decimale attraverso il parametro `size`.

L'inizializzazione dell'albero di regressione avviene definendo un oggetto della classe `AutoTree` impostando il parametro `obj = 'regressione'` e i restanti come da default (profondità massima pari a 5). Attraverso il metodo `auto_fit` il modello viene adattato ai dati, ottenendo un primo albero con un numero di nodi terminali pari a 22 e come variabili più importanti quelle maggiormente correlate con la risposta: **lstat** e **rm**. La valutazione delle performance in termini di errore di generalizzazione viene eseguita con l'ausilio della funzione `perf` che si occupa, dato il vettore con i valori osservati per la risposta e il vettore dei valori predetti, di calcolare le metriche: errore quadratico medio (**MSE**), la sua variante sotto radice quadrata (**RMSE**), l'errore medio assoluto (**MAE**) e l'errore medio assoluto percentuale (**MAPE**).

Basandoci sulla metrica RMSE sbagliamo nel predire il prezzo di una casa di $\$\pm 5460$ circa.

Albero	**MSE**	**RMSE**	**MAE**	**MAPE**
Normale	29.832	5.462	3.228	0.145
Potato	28.481	5.337	3.304	0.148

Tabella 1: MSE, RMSE, MAE, MAPE pre e post potatura

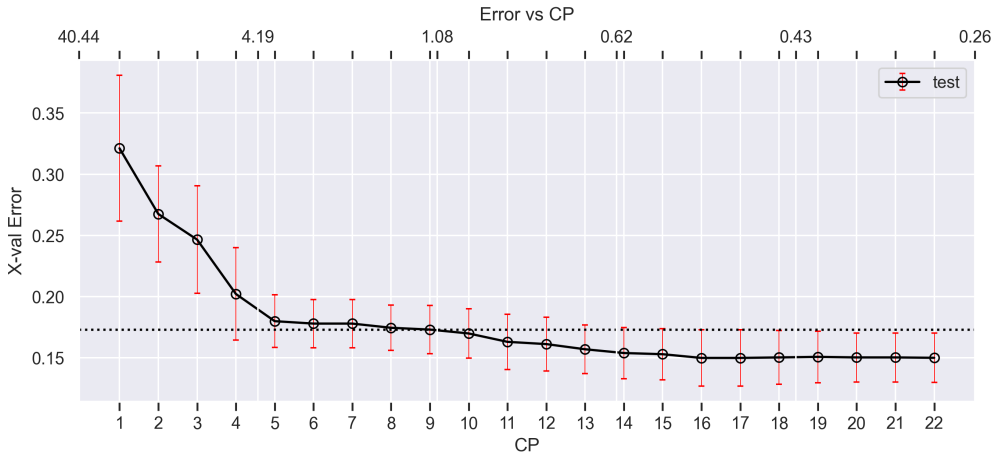


Figura 3: Andamento errore di generalizzazione.

Utilizzando il metodo `auto_prune` viene potato l'albero scegliendo, attraverso il confronto tra il percorso di potatura e l'andamento dell'errore al variare del parametro di complessità (calcolato tramite *cross-validation*), un valore di $\alpha = 0.551$ associato al sotto albero con 11 foglie come si vede in Figura 3. Si ottiene così un miglioramento delle previsioni di circa \$100.

Se in termini di performance, il guadagno prodotto dall'operazione di potatura è solo marginale , si può notare come il numero di nodi terminali sia stato dimezzato, portando ad un notevole miglioramento in termini di interpretabilità.

In particolare, l'albero finale prevede il prezzo medio più basso in corrispondenza di abitazioni che si trovano in distretti con un elevato tasso di persone a basso reddito e un moderato tasso di criminalità: **lstat** > 15 e **crim** > 5.8; al contrario, le case più costose sono quelle in distretti con un tasso di persone a basso reddito inferiore al 9%, un numero di stanze (**rm)** superiore a 7 e un rapporto studenti-insegnati inferiore al 17%, come mostrato in Figura 5.

Per cercare di migliorare ulteriormente le performance predittive si procede con un approccio *ensemble* o *d'insieme* combinando molteplici alberi di regressione attraverso un oggetto della classe `AutoEnsemble`. Tale classe implementa in modo unificato i metodi di *bagging e random forest*, per cui viene inizializzato nell'attributo `model` lo stimatore `RandomForestRegressor`. Il comportamento di default seleziona la variabile split dall'insieme di tutti i p predittori facendo, dunque, crescere gli alberi senza alcun vincolo sulle variabili considerate ad ogni partizione (approccio *bagging*).

Il modello produce un miglioramento importante in termini d'errore, penalizzando, tuttavia, l'interpretazione che viene ora eseguita andando a valutare l'importanza media delle variabili in tutti i singoli stimatori attraverso il metodo `auto_features`. Il metodo seleziona i predittori più rilevanti, utilizzando come `threshold` di default la mediana della distribuzione della diminuzione di RSS causata da ognuno di questi.

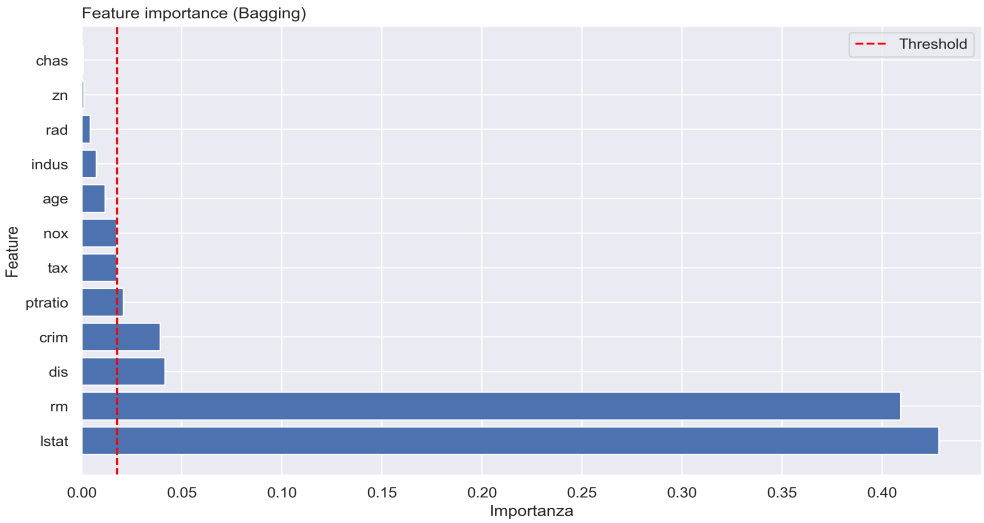


Figura 4: Importanza dei predittori (Bagging).

Poichè il numero di feature "importanti" risulta essere minore del numero di totale di predittori, viene testata la performance del metodo *random forest* inizializzando un oggetto `AutoEnsemble` a cui viene attribuito 6 come valore del parametro `max_features`. Tuttavia, il metodo non comporta alcun miglioramento apprezzabile in termini di performance, rispetto al bagging applicato in precedenza. Tale mancanza è dovuta al fatto che, come visto dalla matrice di correlazione (Figura 1b), le variabili **lstat** e **rm**, fortemente correlate con la risposta, sono correlate con le altre *features* e, quindi, *"riassumono"* in un certo senso tutta l'informazione sull'abitazione. Nonostante il metodo permetta ad altre variabili (meno importanti) di entrare nei singoli alberi, la *decorrelazione* delle stime ottenuta non è così efficace come quella derivante dall'operazione di media.

Si ha, invece, un miglioramento evidente ricorrendo al *boosting*, inizializzabile passando `'gb'` come valore del parametro `method`. Nonostante le performance iniziali siano peggiori del singolo albero di regressione, in seguito all'ottimizzazione stocastica degli iper-parametri, eseguita passando il dizionario dei parametri da ottimizzare al metodo `auto_tuning` con `randomized = True`, otteniamo l'errore di previsione più basso dell'intera analisi.

Il metodo, infatti, utilizzando 1000 alberi di regressione, con una profondità massima pari a 3 e un *learning rate* $\lambda = 0.2$, prevede il prezzo delle abitazioni dell'insieme di verifica in un range di \$±3686.

Ottimizzazione	**Bagging**	**Random Forest**	**Boosting**
No	4.129	4.583	8.435
Si	4.056	4.546	**3.686**

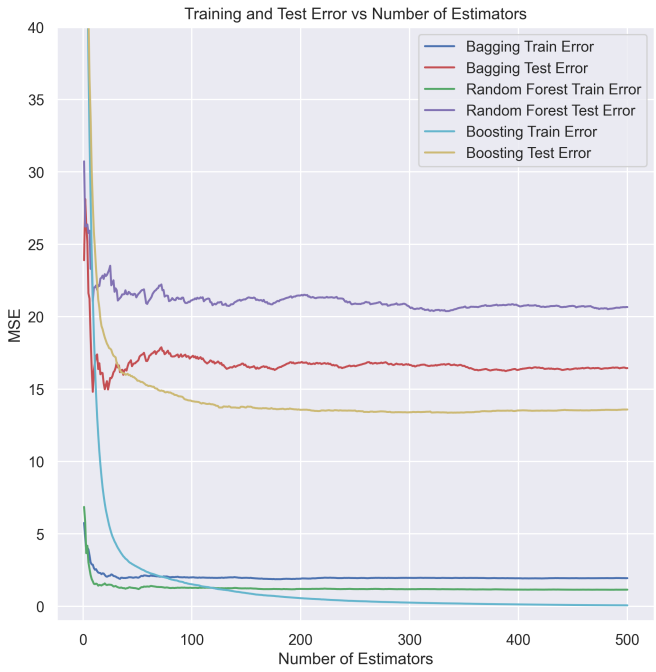
Tabella 2: Confronto metodi *enseble* su dai Boston Housing (RMSE)

4 Conclusione

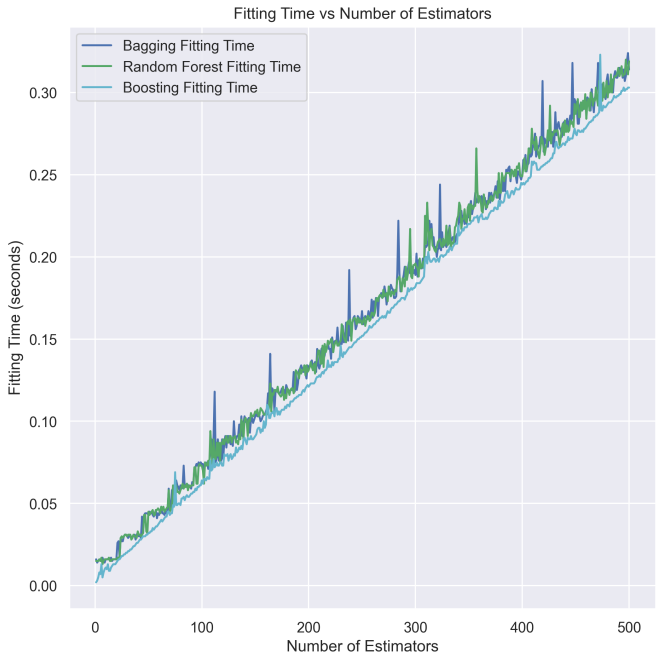
Dal confronto tra i vari metodi emerge che il singolo albero di regressione produce i risultati migliori in termini di **interpretabilità**. Il modello, infatti, con i dovuti accorgimenti, quali la modifica dei criteri di arresto dell'algoritmo di crescita e l'applicazione di tecniche di potatura più o meno aggressive, produce regole di decisione "chiare", facili da comunicare e replicabili. Tali regole risultano essere molto legate ai dati di addestramento e questo conferisce alle previsioni **alta variabilità di stima**, ossia ne penalizza la "precisione".

Per ovviare al problema vengono combinati insieme più alberi di decisione, addestrati su versioni più o meno modificate dei dati di partenza. Questo migliora (in alcuni casi notevolmente) la **precisione** delle stime andando a sfruttare al massimo l'informazione a disposizione e "slegando" le previsioni dal singolo insieme di dati su cui il modello viene allenato. Il prezzo da pagare per tale aumento di performance può risultare più o meno alto. Infatti, se indipendentemente dal metodo utilizzato l'interpretazione dei risultati è nettamente più complicata da esporre, il tempo impiegato per l'adattamento, il **costo computazionale**, il numero di **iper-parametri** da ottimizzare e la relativa riduzione di errore non sono uguali per i tre *enseble* confrontati.

Si può osservare come l'errore diminuisca all'aumentare della complessità per tutti i metodi testati, ma il boosting sembra il metodo che, a parità di numero di stimatori, produce le stime più accurate. Occorre, tuttavia, ricordare che i metodi confrontati in Figura 6a sono quelli con parametri ottimizzati e che proprio il boosting si è dimostrato quello più sensibile a tale tipo di messa a punto, passando da un errore quadratico medio pari a 71.14 *pre-tuning* a solo 13.59 *post-tuning*. Non si nota, invece, alcuna differenza rilevante nell'andamento del tempo di adattamento rispetto all'aumentare del numero di stimatori, che sembra essere lineare per tutti e tre i metodi (Figura 6b).



(a) MSE in funzione del N. di stimatori.



(b) Fitting time Vs. N. di stimatori.

Il metodo che performa meglio *off-the-shelf* risulta essere il bagging, come si può osservare dai risultati ottenuti in Tabella 2.

Tale metodo produce un miglioramento significativo nella qualità delle stime, al netto del costo computazionale necessario per l'ottimizzazione dei parametri.

Inoltre, dal momento che i singoli alberi vengono fatti crescere *indipendentemente* sui campioni bootstrap, il processo può essere **parallelizzato** su sistemi distribuiti e questo porta un enorme vantaggio computazionale rispetto a procedure sequenziali (come il boosting), soprattutto in campo *big data*.

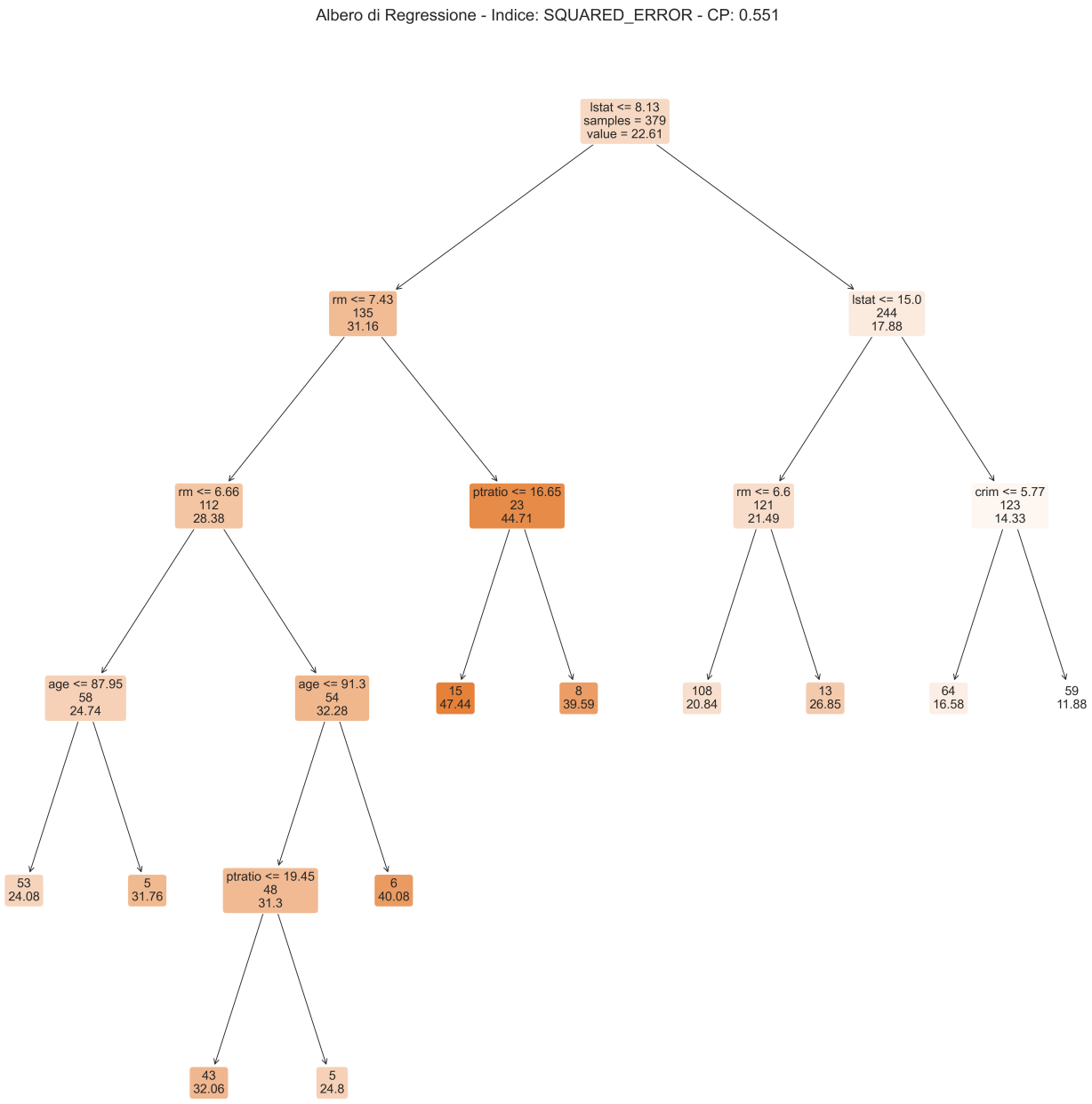


Figura 5: Albero potato.