

Introduzione ad Alberi di Regressione

Programmazione di Applicazioni Data Intensive

Laurea in Ingegneria e Scienze Informatiche
DISI – Università di Bologna

Alberi di Decisione per la Regressione, Test t-Student,
Foreste di Decisione, Gradient Boosting, XGBoost

Gianluca Moro

Dipartimento di Informatica – Scienza e Ingegneria
Università di Bologna
Via dell’Università, 50 – I-47522 Cesena (FC)
nome.cognome@unibo.it

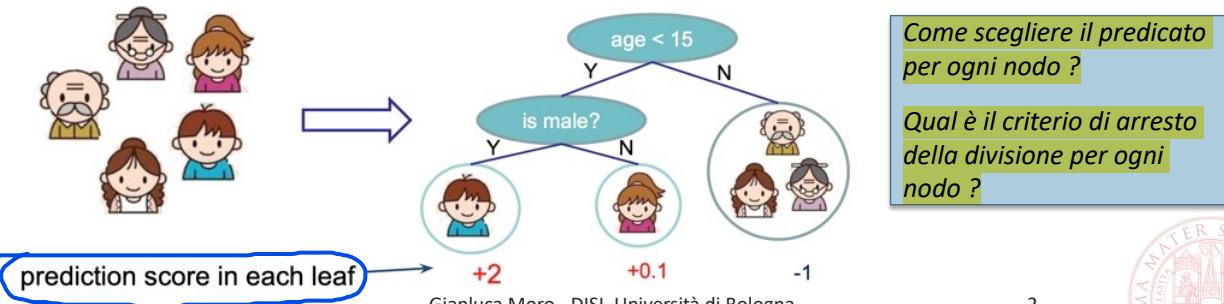
(draft)



Introduzione ad Alberi di Regressione

Alberi di Regressione: Esempio

- per ogni persona sappiamo *età, genere, occupazione* etc. (variabili di input X) e quanto apprezza (Y output) i videogame
- obiettivo: creare una struttura ad albero che predice Y dove
 - ogni nodo intermedio contiene un predicato con una variabile di input
 - il predicato divide le istanze di input in 2 sotto insiemi a cui corrispondono 2 nodi figli con relativi sotto alberi e così via ricorsivamente (e.g. binario)
 - ogni nodo foglia contiene tutte le istanze con apprezzamento Y più simile possibile tra esse, i.e. minimizzando l’errore di regressione in ogni foglia

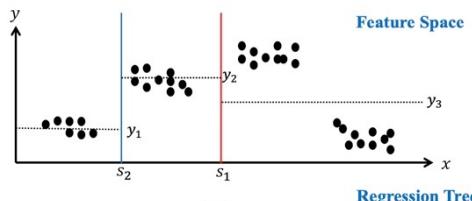
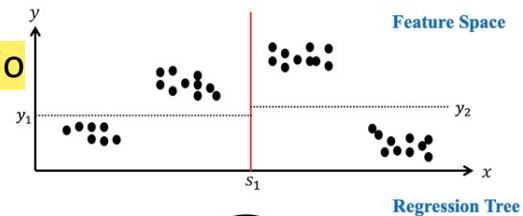


Un albero di regressione suddivide ricorsivamente lo spazio delle feature (in questo caso con una sola variabile X) in sottospazi nei quali assegna un valore di output costante (tipicamente la media del target nei dati in quel sottospazio).

Alberi di Regressione: Divisione dello Spazio (i)

- consideriamo un dataset con una sola variabile X di input ed un albero binario

- la divisione della radice, retta rossa, data dal predicato $x < s_1$ divide in 2 lo spazio in modo t.c. l'errore totale di regressione delle istanze rispetto a y_1 e y_2 sia minimo



- i sottospazi in cui l'errore è insoddisfacente si dividono ricorsivamente con lo stesso criterio;
- la figura di sinistra mostra la divisione del sottospazio di sinistra con $x < s_2$, per cui di nuovo la somma dell'errore di regressione rispetto a y_1 e y_3 nel sottospazio diviso sia inferiore all'errore prima della divisione

Questo processo continua fino a raggiungere:
 - una profondità massima
 - un numero minimo di istanze per nodo
 - oppure l'errore è abbastanza piccolo

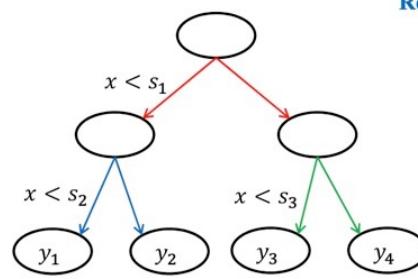
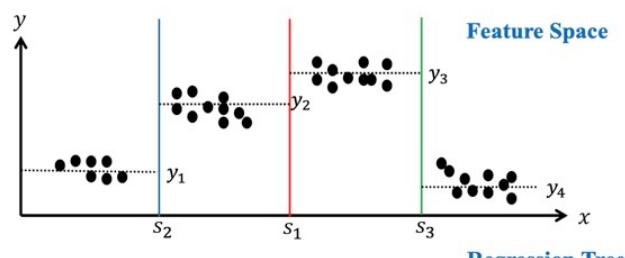
Gianluca Moro - DISI, Università di Bologna

3



Alberi di Regressione: Divisione dello Spazio (ii)

- Analogamente viene diviso il sottospazio di destra, retta verde, con $x < s_3$
- scegliendo s_3 in modo tale che diminuisca l'errore dato dalla somma dell'errore di regressione nei due sottospazi divisi dalla retta verde, rispetto ai valori y_3 e y_4
- Nell'esempio y è l'intervallo di apprezzamento per i videogame ed x è una variabile delle istanze, e.g. l'età delle persone.



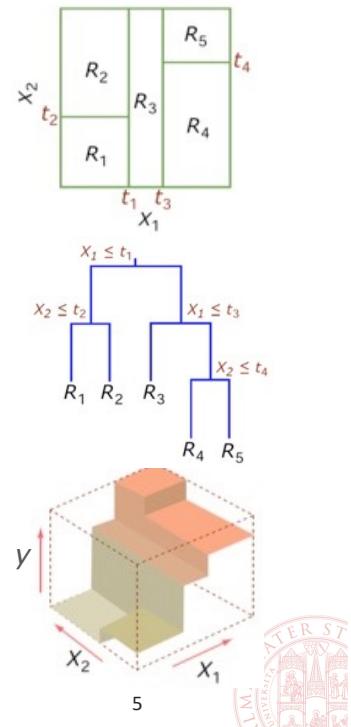
Gianluca Moro - DISI, Università di Bologna

4

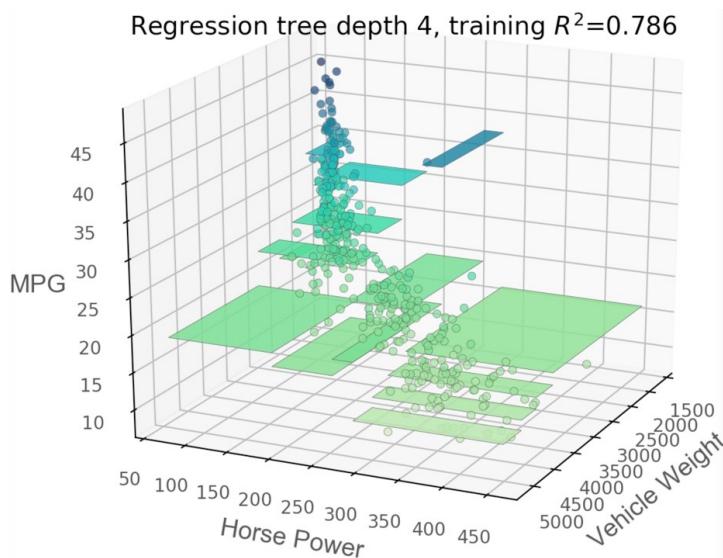


Alberi di Regressione: Esempio in 2 Variabili (i)

- Esempio di partizionamento dello spazio dei dati con due variabili di input x_1, x_2
 - la partizione t_1 su x_1 con $x_1 \leq t_1$ crea 2 sottoalberi
 - la partizione t_2 eseguita su x_2 nel sottoalbero sinistro con $x_2 \leq t_2$ crea 2 sottospazi foglie R_1 ed R_2
 - partizione t_3 su x_1 del sottospazio destro di t_1 con $x_1 \leq t_3$ crea il sottoalbero con la foglia sinistra R_3
 - infine partizione t_4 del sottospazio destro di t_3 su x_2 con $x_2 \leq t_4$ crea 2 sottospazi foglie R_4 ed R_5
- Visualizzazione del partizionamento in 3D
 - con i valori di regressione y predetti, per ciascuna partizione in base ai dati raccolti in ciascuna di esse
 - valori che corrispondono ad iperpiani ortogonali ad y



Alberi di Regressione: Esempio in 2 Variabili (ii)



- previsione del *consumo di carburante (y) miles/gallon* di veicoli con input *peso e potenza (X)* con albero di regressione

Alberi di Regressione: Algoritmo greedy di base

- sia $k = 0$, $S = \mathcal{R}^d$ lo spazio iniziale dei dati, $d = \text{num. variabili di input}$
- def **RegressionTree**(k, S)

$k += 1$

per ogni variabile di input $j = 1, \dots, d$

divide lo spazio S scegliendo per j il valore di separazione $v_j \in \mathcal{R}$ t.c.

$$S_{j,<} = \{ i \in S \mid x_{i,j} < v_j \}; \quad S_{j,\geq} = \{ i \in S \mid x_{i,j} \geq v_j \}$$

$$\bar{y}_{<} = \frac{\sum_{i \in S_{j,<}} y_i}{|S_{j,<}|}; \quad \bar{y}_{\geq} = \frac{\sum_{i \in S_{j,\geq}} y_i}{|S_{j,\geq}|}$$

$$MSE_j = \min \sum_{i \in S_{j,<}} (y_i - \bar{y}_{<})^2 + \sum_{i \in S_{j,\geq}} (y_i - \bar{y}_{\geq})^2$$

sia p la variabile in $1, \dots, d$ che ha ottenuto il minimo MSE_p

se la condizione _di_stop è insoddisfatta

restituisce **RegressionTree**($k, S_{p,<} \cup \text{RegressionTree}(k, S_{p,\geq})$);

altrimenti restituisce $[(p, v_p)]$ # v_p è il valore di separazione per p

- La condizione di stop è data da iperparametri: e.g. max profondità k , num. foglie, num. min di istanze per foglia, un valore minimo di MSE etc.



L'algoritmo:

- Prova tutti gli split possibili per tutte le feature
- Calcola l'MSE per ogni split
- Sceglie lo split che minimizza l'errore
- Si ferma se le condizioni lo richiedono, altrimenti continua ricorsivamente

Complessità del Modello ed Overfitting

- La complessità del modello aumenta all'aumentare della profondità, i.e. dimensione, dell'albero di regressione
 - Regolare la complessità è necessario per adattare il modello alla complessità dei dati, ma la complessità può causare overfitting
 - Regolare la complessità con la regolarizzazione, come abbiamo già visto nelle diverse soluzioni precedenti

- Aggiungiamo la dimensione dell'albero alla loss con peso dato dall'iperparametro C da ottimizzare con nested cross validation
 - $\min \text{ errore-di-traiing(Tree)} + C \times \text{dimensione(Tree)}$

- Strategie alternative di creazione dell'albero

Pre-Crescita

lasciare crescere l'albero da zero e fermarlo quando la loss aumenta

Post-Crescita

lasciare crescere pienamente l'albero e poi potarlo partendo dalle foglie fino a quando la loss torna ad aumentare

la seconda è meno sensibile a scelte greedy non ottimali di predicit

Si costruisce prima un albero massimo (spesso volutamente overfittato al training set).

Poi, si inizia a "tagliare" i rami partendo dalle foglie, risalendo verso la radice. Ad ogni taglio, si valuta se la rimozione di un sotto-albero migliora o mantiene le prestazioni su un set di validazione.

Il processo si ferma quando la rimozione di ulteriori rami inizia a far peggiorare la performance sul set di validazione (o sul set di test, anche se è meglio usare un set di validazione dedicato per il pruning per non inquinare la valutazione finale). Questo approccio è spesso preferito perché una decisione di taglio presa nelle fasi iniziali di costruzione dell'albero (pre-pruning) potrebbe impedire la scoperta di un sotto-albero utile che si trova più in profondità.

Tecniche applicate sia nel Pre-Crescita sia nel Post-Crescita della slide precedente

Il p-value è la probabilità di osservare una differenza tra le medie grande quanto (o più grande di) quella che abbiamo effettivamente misurato nei nostri dati, assumendo che l'ipotesi nulla sia vera (cioè, che non ci sia una vera differenza tra le popolazioni da cui provengono i campioni).

Regole di Potatura (Pruning)

- 1 ▪ Stop quando ogni foglia contiene una sola istanza o un numero di istanze inferiore ad una soglia
- 2 ▪ Stop quando il numero di foglie è inferiore ad una soglia o quando l'errore nella foglia è inferiore ad una soglia
- 3 ▪ Stop quando il p-value della differenza tra le due regressioni ottenute nella divisione di un nodo in 2 foglie è maggiore di una soglia (e.g. 0.05) in base ad un test statistico →
- 4 ▪ E.g. student t-test dei due gruppi, per stabilire se le due partizioni di dati del nodo hanno medie (i.e. regressioni) diverse con significatività statistica data dal p-value, e.g. < 0.05
- 4 ▪ Albero di *classificazione*: Stop quando tutte le istanze in ogni foglia hanno la stessa label

Questa regola mira a garantire che ogni divisione (split) in un albero di regressione sia statisticamente significativa. In altre parole, si vuole essere ragionevolmente certi che la divisione non sia avvenuta per puro caso o a causa di fluttuazioni casuali nei dati di training, ma rifletta una vera differenza tra i sottogruppi di dati creati dalla divisione.

Se p-value > 0.05: Significa che la probabilità di ottenere la differenza osservata per puro caso è alta. Non abbiamo prove sufficienti per rifiutare l'ipotesi nulla. Quindi, concludiamo che la differenza tra le medie dei due gruppi non è statisticamente significativa. In questo caso, la divisione non dovrebbe essere fatta (pre-pruning) o il sotto-albero che ne deriva dovrebbe essere potato (post-pruning), perché non aggiunge valore predittivo robusto.

Gianluca Moro - DISI, Università di Bologna

9

Se p-value <= 0.05: Significa che la probabilità di ottenere la differenza osservata per puro caso è bassa. Rifiutiamo l'ipotesi nulla. Concludiamo che la differenza tra le medie dei due gruppi è statisticamente significativa. In questo caso, la divisione è giustificata e può essere eseguita, poiché riflette una vera distinzione nei dati.

Alberi: Potatura con t-test Student su 2 insiemi

Questa è la formalizzazione dell'ipotesi Nulla per il test t di Student. L'ipotesi nulla assume che non ci sia una differenza reale tra le medie dei valori target nei due gruppi che si formerebbero con la divisione. Qualsiasi differenza osservata nei dati di training è considerata un'aberrazione casuale. L'obiettivo del test è cercare prove sufficienti per rifiutare questa ipotesi nulla. Se non riusciamo a rifiutarla, accettiamo che i due gruppi sono equivalenti (non significativamente diversi).

- test d'ipotesi statistico per decidere se la differenza tra gruppi di dati è significativa o frutto del caso →
 - ← ipotesi da testare, i.e. ipotesi nulla: le medie di 2 gruppi, i.e. insiemi, di dati sono equivalenti ? regressioni
 - t-score (i.e. t-value) è il rapporto tra la differenza tra i due gruppi di dati e la differenza interna a ciascun gruppo
 - maggiore è t-score, minore è la probabilità, i.e. p-value, che la differenza tra i due gruppi sia frutto del caso
 - p-value è calcolato con t-test Student
- fissiamo una soglia di accettazione dell'ipotesi nulla, e.g. 0.05
 - se il corrispondente p-value è superiore alla soglia di accettazione allora assumiamo che la differenza tra i due gruppi sia frutto del caso
 - perciò accettiamo l'ipotesi nulla: i 2 gruppi sono equivalenti e potiamo (annulliamo) il sottoalbero riunendo le due foglie in un nodo foglia

Ogni volta che l'algoritmo di costruzione dell'albero propone di dividere un nodo in due nodi figli (o "foglie" temporanee per la valutazione), esegue un test statistico per determinare se la divisione porta a gruppi di dati che sono veramente diversi tra loro in termini del valore target (la variabile dipendente che stiamo cercando di prevedere).



t-test di Student su 2 insiemi: t-score e p-value

- siano 2 foglie $S_{j,<} , S_{j,\geq}$ date dalla suddivisione rispetto a $v_j \in \mathcal{R}$

$$S_{j,<} = \{ i \mid x_{i,j} < v_j \}; \quad S_{j,\geq} = \{ i \mid x_{i,j} \geq v_j \}; \quad \text{medie } \bar{y}_{<} = \frac{\sum_{i \in S_{j,<}} y_i}{|S_{j,<}|}; \quad \bar{y}_{\geq} = \frac{\sum_{i \in S_{j,\geq}} y_i}{|S_{j,\geq}|}$$

$$\text{varianze } \sigma_{\bar{y}_{<}}^2 = \frac{\sum_{i \in S_{j,<}} (y_i - \bar{y}_{<})^2}{|S_{j,<}|-1}; \quad \sigma_{\bar{y}_{\geq}}^2 = \frac{\sum_{i \in S_{j,\geq}} (y_i - \bar{y}_{\geq})^2}{|S_{j,\geq}|-1}$$

$$\text{t-score} = \frac{\bar{y}_{<} - \bar{y}_{\geq}}{\sqrt{\frac{\sigma_{\bar{y}_{<}}^2}{|S_{j,<}|} + \frac{\sigma_{\bar{y}_{\geq}}^2}{|S_{j,\geq}|}}} \quad \text{p-value} = P(Z > |\text{t-score}|) \text{ con } Z \sim \mathcal{N}(0, 1)$$

Gradi di libertà $v = |S_{j,<}| + |S_{j,\geq}| - 2$

distrib.
normale

- t-score segue approssimativamente la t-distrib. con v gradi di libertà
- E.g. siano 2 foglie con $|S_{j,<}| = |S_{j,\geq}| = 30$, t.c. $\bar{y}_{<} = -1.2$; $\bar{y}_{\geq} = -0.5$; $\sigma_{\bar{y}_{<}}^2 = 23.2$; $\sigma_{\bar{y}_{\geq}}^2 = 10.7$; t-score = -0.66, p-value = 0.51 > 0.05, i 2 gruppi sono equivalenti
- in python: `from scipy import stats ... t_score, p_value = stats.ttest_ind(S_{j,<}, S_{j,\geq})`

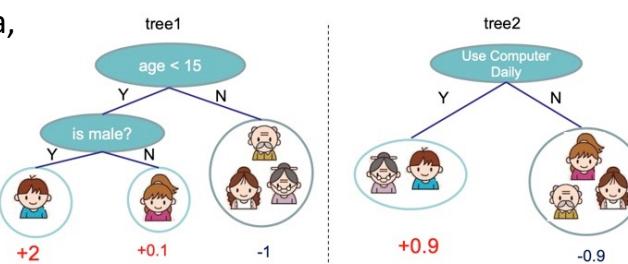


Foreste di Regressione: Esempio

Questi alberi sono spesso "semplici" nel senso che possono essere overfittati o underfittati se considerati individualmente, ma la loro combinazione riduce la varianza (nel caso del Bagging) o il bias (nel caso del Boosting) del modello finale.

- in genere l'errore diminuisce combinando più alberi semplici creati sullo stesso dataset (*Ensemble Learning*), due strategie:

- utilizzando per ciascuno diversi sottoinsiemi casuali di dati non disgiunti, **bagging**, e sottoinsiemi casuali di variabili di input e.g. **random forest**
- alberi in sequenza da errori residui del precedente: **gradient boosting**, **xgboost** ...
- l'output è ottenuto combinando in varie modalità i singoli output
- Esempio precedente in figura, 2 alberi con feature diverse
- La previsione per ogni istanza è la somma delle regressioni predette da ciascun albero
- Aumenta la complessità, diminuisce l'interpretabilità



$$f(\text{boy}) = 2 + 0.9 = 2.9 \quad f(\text{old man}) = -1 - 0.9 = -1.9$$

Slide in parte dagli autori di xgboost
A differenza del Bagging, dove gli alberi sono costruiti in parallelo e indipendentemente, nel Boosting gli alberi sono costruiti in sequenza. Ogni nuovo albero cerca di correggere gli errori (residui) fatti dall'albero o dalla combinazione di alberi precedenti. Gli alberi sono "debolli" e si concentrano sulle istanze che sono state classificate o previste male dagli alberi precedenti.

Decorrelazione degli alberi: Se ci fosse una feature molto forte, tutti gli alberi tenderebbero a dividerla all'inizio, rendendo gli alberi molto simili tra loro (e quindi le loro previsioni correlate). Il campionamento delle feature riduce questa correlazione, assicurando che gli alberi siano più "diversi" l'uno dall'altro. La diversità è fondamentale per il successo dell'Ensemble.

Questo approccio riduce ulteriormente la varianza e prevenire l'overfitting.

Bagging (e Random Forest)

Per la Regressione: L'output finale per una nuova istanza è solitamente la media (average) delle previsioni di tutti i singoli alberi. Questo è il caso più comune per le foreste di regressione (es. Random Forest Regressor). Ogni albero ha lo stesso "peso" nella decisione finale.

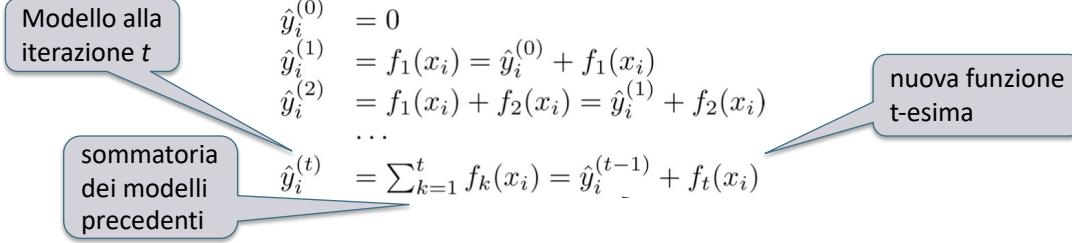
Per la Classificazione: L'output finale è determinato tramite voto a maggioranza (majority voting). Ogni albero predice una classe, e la classe che riceve il maggior numero di voti dagli alberi vince. A volte si usa anche la media delle probabilità predette da ciascun albero per la classe (soft voting).

Introduzione ad Alberi di Regressione

Gradient Boosting Machine: Learning Additivo di più Modelli

- Training additivo di modelli, da qui *boosting*

- inizia con un modello a previsione costante, e.g. che predice sempre 0, e ad ogni iterazione k somma un nuovo modello $\hat{y}_i^{(k)} = f_k(x_i)$ al precedente



- ogni albero aggiunto è addestrato dal dataset degli errori residui del precedente, i.e. dalle coppie $(x_i, y_i - \hat{y}_i^{(k-1)})$ con y_i il valore da predire

Loss da minimizzare $L(y, \hat{y}^{(t)}) = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i^{(t)})^2$ *N num. istanze*

Per ogni istanza i nel dataset di training, l'input dell'albero sarà sempre x_i (le feature). Ma l'output che l'albero cercherà di predire non sarà y_i , bensì la differenza $(y_i - \hat{y}_i^{(k-1)})$.

Gianluca Moro - DISI, Università di Bologna

13



Introduzione ad Alberi di Regressione

Gradient Boosting vs Discesa del Gradiente

- minimizziamo $L(y, \hat{y}^{(t)})$ rispetto ad uno specifico $\hat{y}_j^{(t)}$

$$\begin{aligned} \frac{\partial L(y, \hat{y}^{(t)})}{\partial \hat{y}_j^{(t)}} &= \frac{\partial \sum_{i=1}^N (y_i - \hat{y}_i^{(t)})^2}{\partial \hat{y}_j^{(t)}} \\ &= \frac{\partial}{\partial \hat{y}_j^{(t)}} (y_j - \hat{y}_j^{(t)})^2 \\ &= -2(y_j - \hat{y}_j^{(t)}) \\ \frac{\partial L(y, \hat{y}^{(t)})}{\partial \hat{y}^{(t)}} &= y - \hat{y}^{(t)} \end{aligned}$$

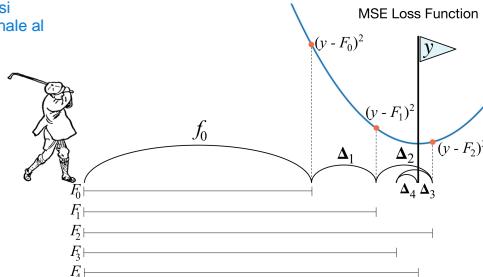
Nel Gradient Boosting, ogni nuovo modello è addestrato per approssimare il gradiente negativo della funzione di perdita (loss function) corrente rispetto alle previsioni del modello cumulativo precedente. Solo nel caso specifico della funzione di perdita MSE, il gradiente negativo si riduce ad essere proporzionale al residuo $(y - \hat{y}^{(t)})$.

Eliminato N dalla sommatoria, il num. di istanze, poiché è costante e $\operatorname{argmin}_x c \cdot f(x) = f(x)$

Eliminata la sommatoria poiché le derivate parziali di L con $i \neq j$ valgono zero

gradiente rispetto al modello t -esimo $\hat{y}^{(t)}$ per tutte le istanze, con la costante 2 analogamente eliminata

Il nuovo modello che viene aggiunto non predice un valore a caso o il target originale. Invece, è un modello addestrato specificamente per predire il gradiente negativo della funzione di perdita corrente. Una volta che $f(x)$ è stato addestrato per approssimare il gradiente negativo, la sua previsione ci dice "per questa specifica istanza x , l'errore attuale è in questa direzione e di questa entità. Aggiungi questo valore alla tua previsione per correggerlo." Quando aggiungiamo $f(x)$ (pesato dal learning rate) alla previsione cumulativa precedente ($\hat{y}^{(t-1)}$), stiamo letteralmente facendo un "passo" nella direzione che minimizza la perdita.



Gianluca Moro - DISI, Università di Bologna

14

il modello GBM è una funzione ricorrente

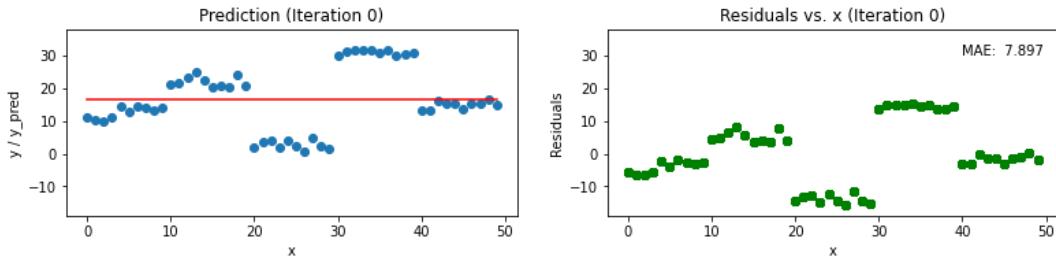
$$\hat{y}^{(t)} = \hat{y}^{(t-1)} + f_t(x)$$

l'aggiunta di un nuovo modello $f_t(x)$ ad ogni iterazione t riduce il residuo, i.e. gradiente, $y - \hat{y}^{(t)}$ infatti la loss è $L(y, \hat{y}^{(t-1)} + f_t(x))$ con discesa del gradiente con η learning rate

$$\hat{y}^{(t)} = \hat{y}^{(t-1)} + \eta(-\nabla L(y, \hat{y}^{(t-1)} + f_t(x)))$$



Esempio di Gradient Boosting: Stato Iniziale (i)



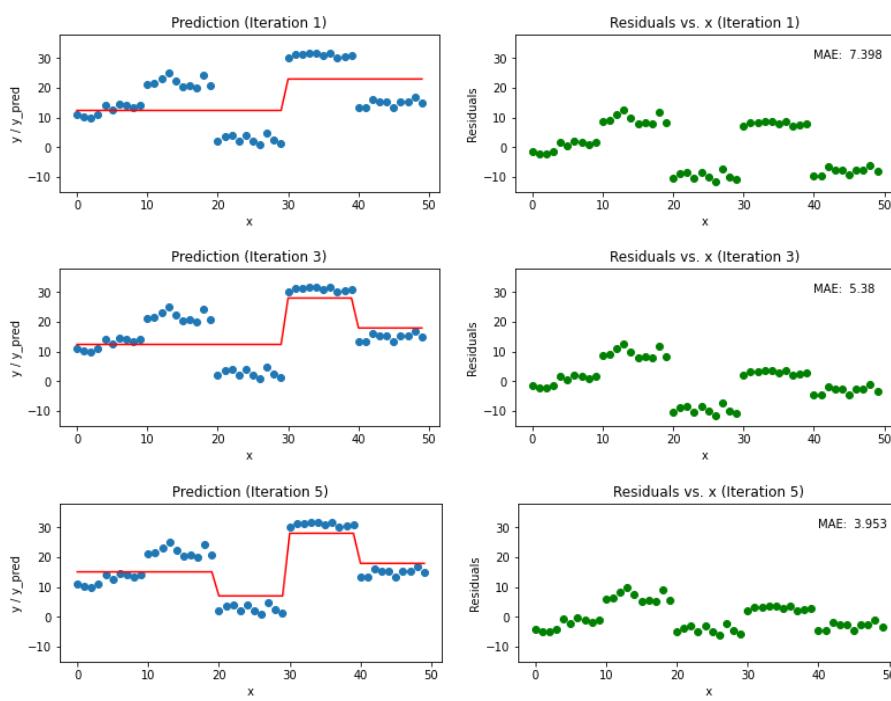
- La figura di sinistra mostra il dataset univariato con istanze (x_i, y_i) , dove x_i è la variabile di input ed y_i la variabile target da predire
- Inizialmente, prima dello split, il modello di previsione $\hat{y}_i^{(0)} = \bar{y}_i$ i.e. valor medio delle istanze y_i (retta rossa), ma può essere qualsiasi valore, anche 0
- La figura di destra mostra gli errori residui $y_i - \hat{y}_i^{(0)}$ per ogni istanza i : l'errore assoluto medio MAE è 7.897, idealmente dovrebbe essere 0
- L'iterazione successiva genera un albero $\hat{y}_i^{(1)}$ dalle istanze $(x_i, y_i - \hat{y}_i^{(0)})$, i.e. dagli errori residui, ed il nuovo modello di previsione diventa $\hat{y}_i^{(0)} + \hat{y}_i^{(1)}$

Gianluca Moro - DISI, Università di Bologna

15



Esempio: Iterazioni e Alberi da 1 a 5 (ii)



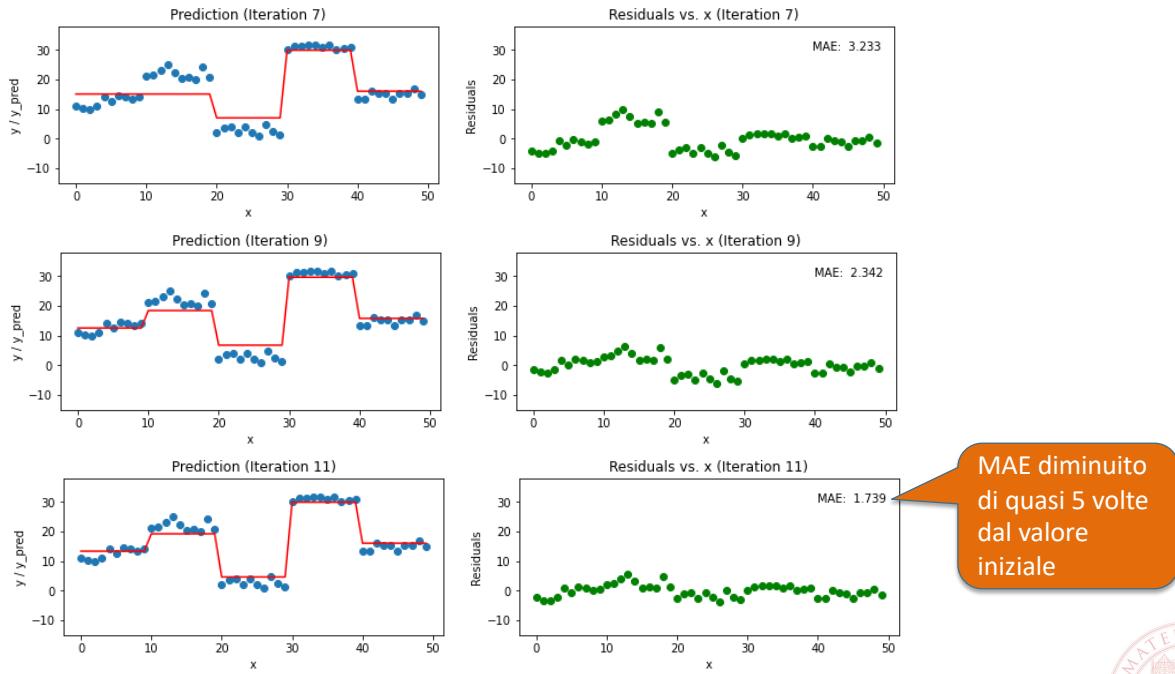
I residui si distribuiscono intorno allo zero in modo NON casuale, quindi c'è margine per creare nuovi alberi che modellino la loro distribuzione non casuale

Gianluca Moro - DISI, Università di Bologna

16



Esempio: Iterazioni e Alberi da 7 a 11 (iii)

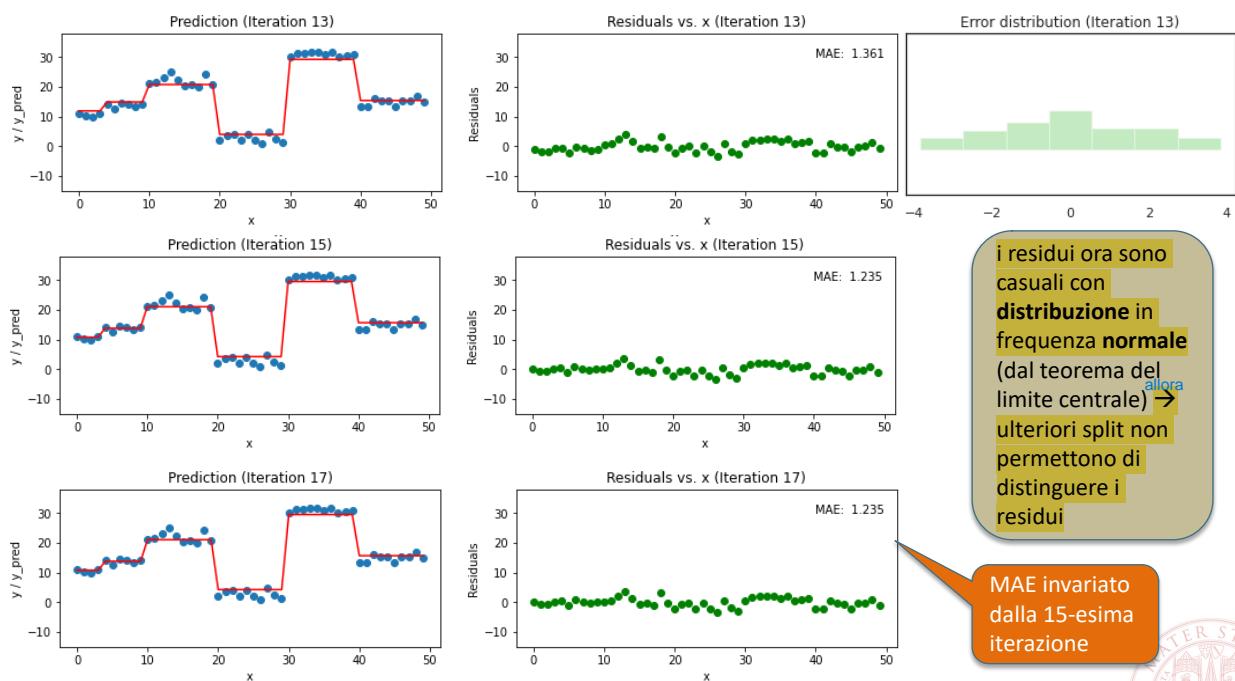


Gianluca Moro - DISI, Università di Bologna

17



Esempio: Iterazioni e Alberi da 13 a 17 (iv)



Gianluca Moro - DISI, Università di Bologna

18



Supervised Learning di un Singolo Modello

- obiettivo del supervised learning di un singolo modello
 - determinare dai dati x_i una funzione f parametrica di regressione per predire i corrispondenti \hat{y}_i $\hat{y}_i = f(x_i)$
 - con un modello lineare con parametri w_j si ha $\hat{y}_i = \sum_j w_j x_{ij}$
 - con d parametri w_j $\Theta = \{w_j | j = 1, \dots, d\}$

- funzione obiettivo da minimizzare: loss L + la regolarizzazione Ω

$$\text{Obj}(\Theta) = L(\Theta) + \Omega(\Theta)$$

In L manca la divisione per N , il numero di istanze, perché $\text{argmin}_x c \cdot f(x) = f(x)$ quando c è costante

- dove $L(\Theta) = \sum_i (\hat{y}_i - y_i)^2$, $\Omega(\Theta) = \lambda \|w\|_2^2$
- loss L definita con la tecnica dei minimi quadrati
 - per determinare i parametri w che minimizzano l'errore di regressione
- regolarizzazione Ω definita dal quadrato della norma L2 pesata per l'iperparametro lambda
 - per ridurre, in valore assoluto, il valore dei w e semplificare il modello f



Supervised Learning Additivo di più Alberi

- regressione con K alberi

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i), \quad f_k \in \mathcal{F}$$

Spazio (foresta) degli alberi di regressione

- dove ogni f_k è un albero da apprendere e la previsione \hat{y}_i è data dalla sommatoria delle previsioni dei K alberi

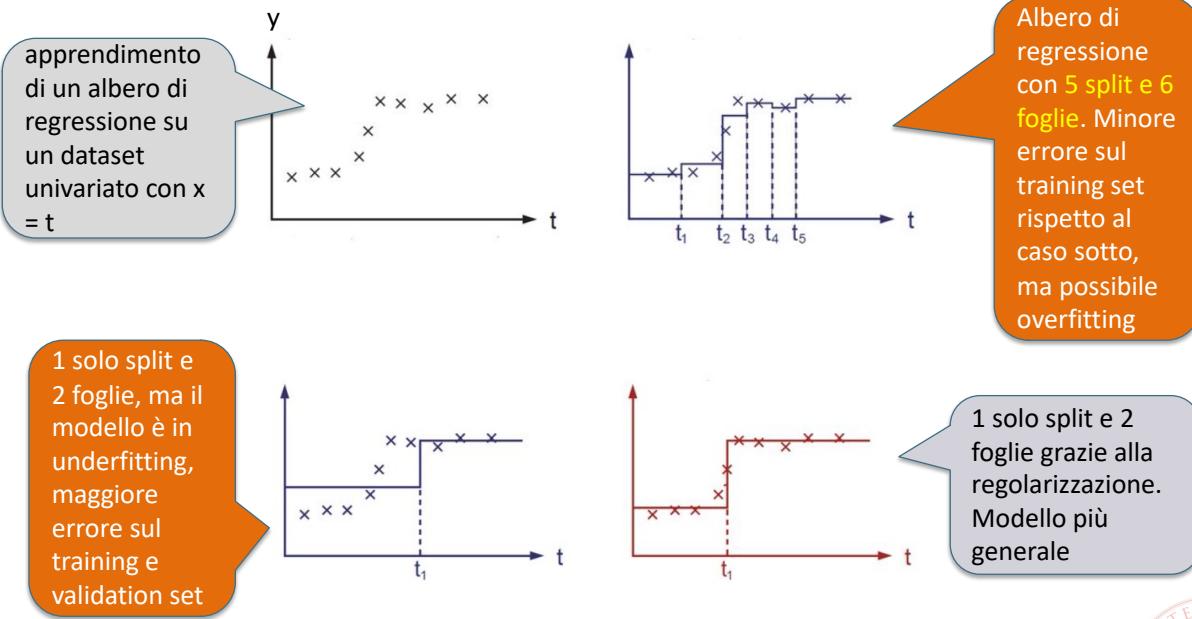
- la loss è la somma delle loss e delle regolarizzazioni dei K alberi

$$\text{Obj} = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

- La minimizzazione delle loss l riduce l'errore sul training set
- La regolarizzazione Ω riduce la complessità degli alberi e tende a rendere i modelli più generali e stabili sui nuovi dati
 - e.g. validation e test set, e sui dati a regime dopo il deployment del modello



Regolarizzazione: Overfitting e Underfitting



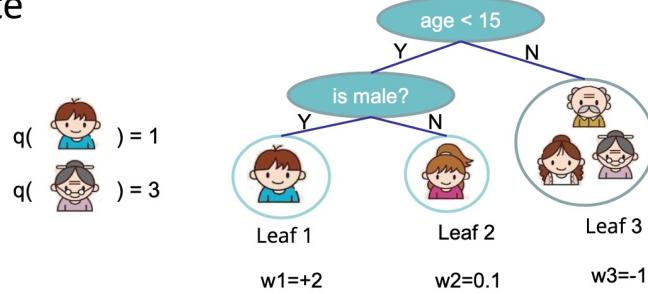
Gianluca Moro - DISI, Università di Bologna

21



Definizione della Complessità di un Albero

- Gli split aumentano la complessità perché generano più foglie e aumentano la profondità dell'albero
- Con la regolarizzazione vogliamo controllarne la complessità
- Sia $f_t(x) = w_{q(x)}$, $w \in \mathbf{R}^T$, $q : \mathbf{R}^d \rightarrow \{1, 2, \dots, T\}$
 - dove f_t è il t -esimo albero di regressione con $w_{q(x)}$ coefficienti di regressione, T numero di foglie, q funzione di struttura dell'albero, d dimensione dei dati di input x
- Esempio con il seguente albero di regressione
 - $q(x_i)$ assegna ogni istanza x_i ad una foglia: la 1a istanza qui alla 1a foglia e la 2a alla terza foglia



Gianluca Moro - DISI, Università di Bologna

22



Complessità degli Alberi in XGBoost:

Extreme Gradient Boost

- Definizione della complessità

$$\Omega(f_t) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$

- dove f_t è il t-esimo albero di regressione
- gamma è l'iperparametro che regola il numero di foglie
- lambda l'iperparametro per la regolarizzazione dei coeff. di regressione

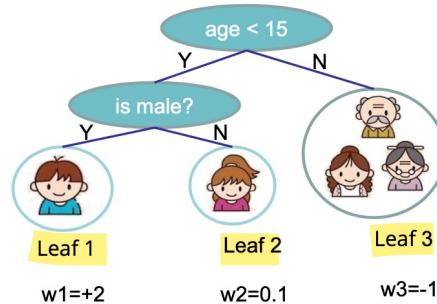
- Per l'albero di esempio con

- $T = 3$ foglie

Coeff di Regressione $w_1 = 2, w_2 = 0.1, w_3 = -1$

- otteniamo

$$\Omega = \gamma 3 + \frac{1}{2} \lambda (4 + 0.01 + 1)$$



Gianluca Moro - DISI, Università di Bologna

23



XGBoost: Apprendere una Foresta di Alberi

- Loss obiettivo

$$\sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_k \Omega(f_k), f_k \in \mathcal{F}$$

- minimizzazione della somma delle loss l → riduce l'errore sul training set
- la regolarizzazione Ω riduce la complessità dei K alberi f_k
- loss obiettivo non differenziabile, i.e. non derivabile

- training additivo di modelli come in GBM, da qui *boosting*

- inizia con una funzione a previsione costante, e.g. restituisce zero, ed aggiunge ad ogni iterazione una nuova funzione

Modello alla iterazione t

$$\begin{aligned}\hat{y}_i^{(0)} &= 0 \\ \hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\ \hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\ &\dots \\ \hat{y}_i^{(t)} &= \sum_{k=1}^t f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)\end{aligned}$$

sommatoria dei modelli precedenti

nuova funzione t-esima

Gianluca Moro - DISI, Università di Bologna

24



XGBoost: Training Additivo

- decide quale funzione aggiungere ottimizzando la loss obiettivo
- la previsione all'iterazione t è $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$ con f_t la nuova funzione da apprendere

$$Obj^{(t)} = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^t \Omega(f_i)$$

- riscritta per esplicitare nella loss l il modello all'iterazione $t-1$ e la f_t da minimizzare

$$Obj^{(t)} = \sum_{i=1}^n l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + \text{constant}$$

sommatoria di regolarizzazione fino a $t-1$ sostituita da una costante

- Con i minimi quadrati minimizza

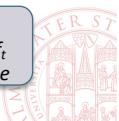
$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^n \left(y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)) \right)^2 + \Omega(f_t) + \text{const} \\ &= \sum_{i=1}^n \left[2(\hat{y}_i^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2 \right] + \Omega(f_t) + \text{const} \end{aligned}$$

Chiamato residuo dalla iterazione precedente

manca $(y_i - \hat{y}_i^{(t-1)})^2$; è eliminato perché non dipende (e.g. moltiplica) f_t , perciò ininfluente nella minimizzazione

Gianluca Moro - DISI, Università di Bologna

25



Approssimazione della Loss con Espansione di Taylor del Secondo Ordine

Opzionale

- Ricordiamo la funzione obiettivo

$$Obj^{(t)} = \sum_{i=1}^n l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + \text{constant}$$

- Approssimazione di una funzione con espansione di Taylor:

$$f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2$$

- Definiamo con g_i e h_i $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$, $h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}^{(t-1)})$

- La funzione obiettivo diventa

$$Obj^{(t)} \simeq \sum_{i=1}^n \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + \text{constant}$$

- Con i minimi quadrati abbiamo

$$g_i = \partial_{\hat{y}^{(t-1)}} (\hat{y}^{(t-1)} - y_i)^2 = 2(\hat{y}^{(t-1)} - y_i) \quad h_i = \partial_{\hat{y}^{(t-1)}}^2 (\hat{y}^{(t-1)} - y_i)^2 = 2$$

Risultato della derivata seconda

Gianluca Moro - DISI, Università di Bologna

26



Nuova Funzione Obiettivo con Approssimazione di Taylor

Opzionale

- funzione obiettivo senza le costanti

$$\sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}^{(t-1)})$$

Il termine $l(y_i, \hat{y}_i^{(t-1)})$ è costante nella minimizzaz. perché dipende dal passo t-1 precedente e non f_t perciò qui è eliminato.

- Definiamo l'insieme delle istanze nella foglia j come I_j

Raggruppamento
per foglie (T foglie)

$$\begin{aligned} Obj^{(t)} &\simeq \sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \\ &= \sum_{i=1}^n [g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^T w_j^2 \\ &= \sum_{j=1}^T \left[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T \end{aligned}$$

- L'obiettivo è una somma di T funzioni quadratiche indipendenti



Funzione Obiettivo Semplificata

Opzionale

- Due risultati noti per funzioni quadratiche in una variabile

$$\operatorname{argmin}_x Gx + \frac{1}{2} Hx^2 = -\frac{G}{H}, \quad H > 0 \quad \min_x Gx + \frac{1}{2} Hx^2 = -\frac{1}{2} \frac{G^2}{H}$$

È la x che minimizza l'espressione

- Definiamo $G_j = \sum_{i \in I_j} g_i \quad H_j = \sum_{i \in I_j} h_i$

È il valore della
espressione
sostituendo la x che
la minimizza

$$\begin{aligned} Obj^{(t)} &= \sum_{j=1}^T \left[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T \\ &= \sum_{j=1}^T \left[G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T \end{aligned}$$

- Supponendo che la struttura dell'albero $q(x)$ sia fissa, i pesi w^* ottimali in ogni foglia ed i valori risultanti della funzione sono

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

$$Obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

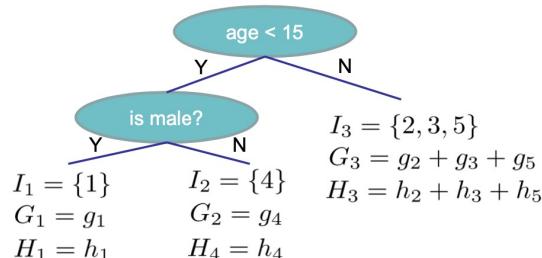
Funz. obiettivo con misura
della complessità dell'albero



Funzione Obiettivo Semplificata: Esempio

- 5 istanze, con indice da 1 a 5, e con I_j e valori G_j e H_j per ciascuna delle 3 foglie

Istanza	Età	Genere	Target	$g_i = \hat{y}_i^{(t-1)} - \text{Target}$	h_i
1	12	M	3	$2.9 - 3 = -0.1$	2
2	16	F	-1.8	$-1.8 - (-1.8) = 0$	2
3	65	M	-2	$-1.9 - (-2) = 0.1$	2
4	14	F	-1	$-0.8 - (-1) = 0.2$	2
5	70	F	0	$0 - 0 = 0$	2



$$Obj = - \sum_j \frac{G_j^2}{H_j + \lambda} + 3\gamma = -0.018 + 3$$

Minore è il valore,
migliore è la capacità
predittiva dell'albero

Gianluca Moro - DISI, Università di Bologna

29



Algoritmo di Ricerca Greedy dell'Albero Migliore

- la ricerca inizia da un albero di profondità zero
- per ogni nodo foglia prova ad eseguire uno split creando due nodi, Left e Right, e calcola il guadagno ottenuto come

$$Gain = \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \gamma$$

Guadagno
prodotto dal
nodo sinistro

Guadagno
prodotto dal
nodo destro

Guadagno
senza eseguire
lo split

Costo
nell'aumento
della complessità
nell'aggiungere
nuove foglie

In sintesi, la formula del Gain è la metrica centrale che guida la costruzione di ogni singolo albero in XGBoost. Permette di valutare non solo quanto bene uno split riduce la perdita, ma anche di penalizzare la complessità, garantendo che gli alberi costruiti siano potenti ma anche generalizzabili.

- come trovare, per ogni foglia, il miglior split ?



Miglior Split, Regolarizzazione e Potatura

- Calcola il guadagno dato da uno split del tipo $x_j < a$
 - nell'esempio l'asse x_j può essere l'età
 - Con le istanze ordinate per età, applica da sinistra a destra una serie di split
 - per ciascuno split calcola g ed h e li somma nelle rispettive due parti G_L e G_R e sceglie lo split che produce il maggior guadagno
- $$G_L = g_1 + g_4$$

$$G_R = g_2 + g_3 + g_5$$
- L'algoritmo termina con una soluzione di trade-off tra semplicità ed errore minimo
- Stop allo split quando il guadagno diventa negativo, anche se può portare a soluzioni migliori successive (approccio greedy)
 - Post potatura: lascia crescere l'albero alla max profondità e ricorsivamente elimina gli split con guadagno negativo

Quando la riduzione della loss di training è inferiore alla regolarizzazione il guadagno è negativo

Gianluca Moro - DISI, Università di Bologna

31



XGBOOST in Python

```

import xgboost as xgb

# read in data
dtrain = xgb.DMatrix('demo/data/agaricus.txt.train')
dtest = xgb.DMatrix('demo/data/agaricus.txt.test')

# specify parameters via map
param = {'max_depth':2, 'eta':1, 'silent':1, 'objective':'binary:logistic' }
num_round = 2
bst = xgb.train(param, dtrain, num_round)

# make prediction
preds = bst.predict(dtest)

```

eta è learning rate
Objective = reg:squarederror per la regressione

- determina anche l'importanza di feature, predice la direzione di discesa nell'albero di istanze con valori mancanti sulla feature di split, mostra la struttura dell'albero per interpretare la conoscenza appresa, automatizza la gestione di dati sparsi

<https://github.com/dmlc/xgboost> <https://xgboost.readthedocs.io/en/latest/index.html>

Gianluca Moro - DISI, Università di Bologna

32



Altri Algoritmi Gradient Boosting

- Light Gradient Boosting Machine
- [LightGBM: A Highly Efficient Gradient Boosting Decision Tree, 2017.](#)
- <https://lightgbm.readthedocs.io/en/latest/Installation-Guide.html>

- CatBoost
- [CatBoost: gradient boosting with categorical features support, 2017.](#)
- <https://catboost.ai/docs/concepts/python-installation.html>

