Apprendimento statistico e machine learning

Metodi ad albero (CART) e splines multivariate (MARS)

Leonardo Egidi

Novembre 2024

Università di Trieste

Alberi di regressione e classificazione (CART)

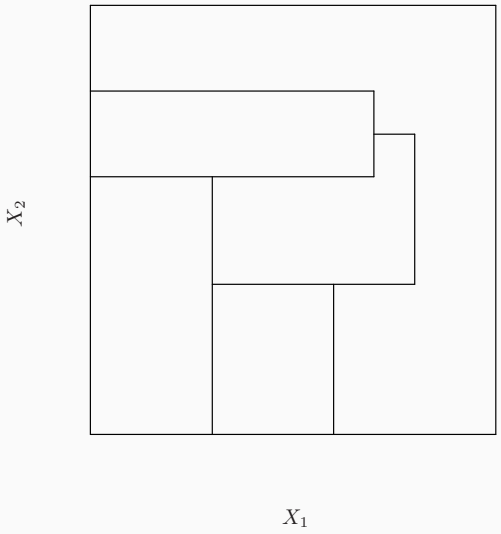
Metodi al albero (tree-based methods)

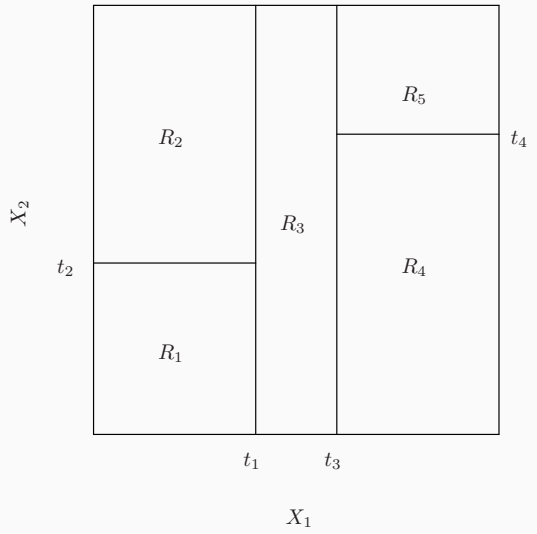
- Descriviamo adesso un altro metodo di regressione, i cosiddetti *metodi ad albero (tree-based)*, che ci saranno molto utili tuttavia anche in ambito di *classificazione*.
- Si tratta di metodi *non parametrici* che *stratificano/segmentano* lo spazio dei predittori in un numero di regioni semplici e rettangolari.
- Siccome l'insieme delle regole di segmentazione delle covariate può essere sintetizzato graficamente da un albero, questi metodi si chiamano per l'appunto *alberi di decisione*.
- Si usa di solito l'acronimo CART: *Classification And Regression Trees*.

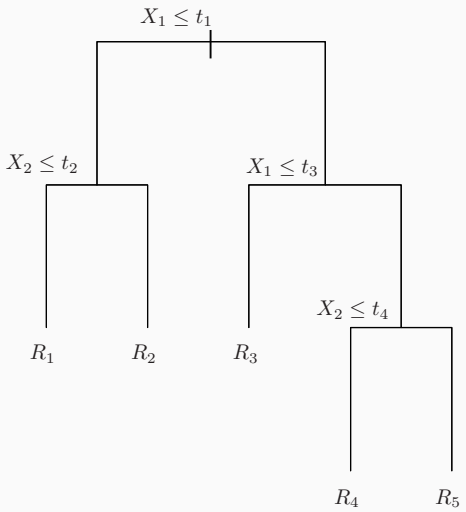
Come funziona un albero di regressione

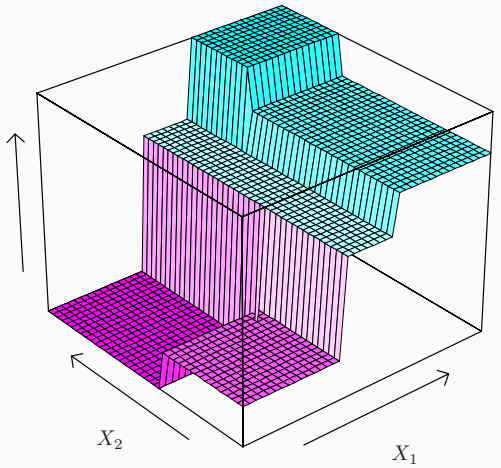
- Consideriamo per semplicità un caso con una variabile risposta continua Y e covariate X_1 e X_2, ognuna delle due con valori in [0,1].
- Introduciamo la cosiddetta tecnica di *partizione ricorsiva binaria*, tipicamente utilizzata nei CART. Dapprima splittiamo lo spazio dei predittori in due regioni, e modelliamo la variabile risposta come la media di Y in ognuna delle regioni. Scegliamo la variabile e il punto di divisione che ci forniscono il miglior fit.
- Dopodiché una o entrambe queste regioni vengono splittate in due regioni a loro volta, e questo processo continua fino a quando non viene applicata qualche regola di arresto.
- Per esempio, X_1 è splittata in $X_1 = t_1$. Poi la regione $X_1 \leq t_1$ è splittata in $X_2 = t_2$, mentre la regione $X_1 > t_1$ è splittata in $X_1 = t_3$. Infine, la regione $X_1 > t_3$ è splittata in $X_2 = t_4$. Il risultato è una partizione in cinque regioni, $R_1, R_2, \ldots, R_5$ (vedi slide seguente).

Come funziona un albero di regressione









Come funziona un albero di regressione

- Il corrispondente modello di regressione prevede Y con una costante c_j nella regione R_j:

$$\widehat{f}(X) = \sum_{j=1}^{5} c_j I\{(X_1, X_2) \in R_j\}.$$

- Nella figura precedente:
  - nel panel in alto a sinistra vi è una segmentazione dello spazio dei predittori che non può essere ottenuta tramite un metodo di partizionamento binario ricorsivo.
  - Nel panel in basso a sinistra vi è la tipica rappresentazione ad albero corrispondente alla partizione dei predittori del pannello in alto a destra.
  - Nel pannello in basso a destra vi è un cosiddetto *perspective plot* della superficie di previsione.

Dettagli del processo di costruzione di un albero

1. Dividiamo lo spazio dei predittori - ovverossia, l'insieme dei possibili valori per $X_1, X_2, \ldots, X_p$ - in J distinte e non sovrapposte regioni, $R_1, R_2, \ldots, R_J$, e modelliamo la risposta come una costante c_j in ciascuna regione:

$$f(x) = \sum_{j=1}^{J} c_j I(x \in R_j).$$

2. Per ogni osservazione che appartiene alla regione j-esima otteniamo la stessa previsione, semplicemente la media della variabile risposta per le osservazioni di training in R_j:

$$\hat{c}_j \equiv \hat{y}_{R_j} = \text{ave}(y_i | x_i \in R_j).$$

Costruzione ottimale delle regioni

Ma come costruiamo di fatto le regioni $R_1, R_2, \ldots, R_J$ allo step 1?

- Le regioni potrebbero in principio avere qualunque forma. Tuttavia, decidiamo di dividere lo spazio dei predittori in *rettangoli multi-dimensionali*, o *scatole (boxes)*, per facilitare l'interpretazione del modello.
- L'obiettivo è reperire quelle regioni $R_1, R_2, \ldots, R_J$ che minimizzino la RSS, data da:

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

dove $\hat{y}_{R_j}$ è la risposta media per le osservazioni di training nella j-esima scatola.

Altri dettagli dell'algoritmo ad albero

- Sfortunatamente è computazionalmente non percorribile considerare ogni possibile partizione dello spazio dei predittori in J regioni.
- Per questo motivo optiamo per un metodo grezzo di tipo *top-down* che è il partizionamento binario ricorsivo.
- Si dice approccio *top-down* perché inizia dal vertice alto dell'albero e successivamente splitta di volta in volta lo spazio dei predittori; ogni nuovo split è indicato come una *ramificazione* in due nuove metà del precedente ramo.
- Si tratta di un approccio *grezzo* perché ad ogni passo della costruzione dell'albero lo split migliore è ottenuto in quel particolare step, piuttosto che guardare in avanti e scegliere uno split che condurrà ad un albero migliore in qualche passo ulteriore e futuro.

Altri dettagli dell'algoritmo ad albero

- 1a. Prima selezioniamo il predittore X_j e il punto di divisione s tale da suddividere lo spazio dei predittori nelle regioni $R_1(j,s) = \{X|X_j \leq s\}$ e $R_2(j,s) = \{X|X_j > s\}$ che forniscono la più grande riduzione possibile in RSS. Quindi, cerchiamo i valori j e s che minimizzano l'equazione:

$$\min_{j,s}\left[\min_{c_1}\sum_{i:x_i\in R_1(j,s)}(y_i-c_1)^2+\min_{c_2}\sum_{i:x_i\in R_2(j,s)}(y_i-c_2)^2\right].$$

  Per ogni scelta j, s, la minimizzazione interna è realizzata da:

$$\hat{c}_1 \equiv \hat{y}_{R_1} = \text{ave}(y_i|x_i \in R_1(j,s)), \text{ e } \hat{c}_2 \equiv \hat{y}_{R_2} = \text{ave}(y_i|x_i \in R_2(j,s)).$$

- 1b. Poi ripetiamo il processo, cercando il miglior predittore e il miglior *cutpoint* per splittare i dati ulteriormente in modo da minimizzare la RSS all' interno di ciascuna delle risultanti regioni. Tuttavia questa volta, invece di splittare l'intero spazio dei predittori, splittiamo una delle due regioni precedentemente identificate. Abbiamo adesso tre regioni.

Altri dettagli dell'algoritmo ad albero

- 1c. Ancora, splittiamo una di queste tre regioni in base alla minimizzazione della RSS. Il processo continua fino a quando una soglia di arresto non viene applicata: per esempio, possiamo continuare fino a quando nessuna regione contiene più di 5 osservazioni ciascuna.
- 2 Una volta ottenute le regioni $R_1, R_2, \ldots, R_J$, prevediamo la variabile risposta per un test set usando la media delle osservazioni di training per la regione alla quale l'osservazione di test appartiene.

Nella figura in slide 4 su dati artificiali abbiamo ottenuto una segmentazione in 5 regioni.

Quanto far crescere l'albero?

- Il processo descritto può fornire buone previsioni per il training set, ma un albero molto largo rischia di *overfittare* i dati, dando quindi una scarsa accuratezza predittiva sul test set.
- Un albero più piccolo, con meno splits, quindi meno regioni $R_1, R_2, \ldots, R_J$, può portare a meno varianza e miglior interpretazione al costo di un maggior bias.
- Un'alternativa è quella di far crescere l'albero fino a quando la riduzione in RSS dovuta ad ogni split ecceda una data soglia.
- Questa alternativa produce alberi più piccoli, ma è un po' *miope*: un apparentemente inutile split potrebbe infatti essere seguito da uno split molto buono e utile - ovverossia, uno split che più avanti riduce la RSS di molto.

Quanto far crescere l'albero? Potatura (pruning)

- Una strategia migliore è quella di far crescere un albero molto largo T_0, e poi *potarlo* (pruning) a ritroso per ottenere un *sotto-albero*.
- Tale strategia è anche detta *cost complexity pruning*.
- Consideriamo una sequenza di alberi indicizzati mediante un parametro di tuning non-negativo α. Ad ogni valore di α corrisponde un sotto-albero $T \subseteq T_0$ tale che:

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

sia più piccola possibile. Qui $|T|$ indica il numero di nodi terminali per l'albero T, R_m è il rettangolo - il sottoinsieme dello spazio dei predittori - corrispondente all' m-esimo nodo terminale, e $\hat{y}_{R_m}$ è la media delle osservazioni di training in R_m.

Quanto far crescere l'albero? Potatura (pruning)

- Il parametro di tuning α governa il trade-off esistente tra la complessità del sotto-albero e il fit sui dati di training.
- Selezioniamo il miglior $\hat{\alpha}$ tramite cross-validation.
- Torniamo poi all'intero dataset e otteniamo il sotto-albero corrispondente ad $\hat{\alpha}$, quindi l'albero $T_{\hat{\alpha}}$.
  - Valori alti di α producono sotto-alberi T_α più piccoli (meno complessi), e viceversa per valori piccoli di α. Se $\alpha = 0$ la soluzione è l'albero intero T_0.
  - Formulazione simile al lasso!

Algoritmo intero per costruire un albero di regressione

1. Usare la partizione ricorsiva binaria per far crescere un albero largo sui dati di training, stoppandosi solo quando ciascun nodo terminale ha meno di un minimo numero di osservazioni.
2. Applicare la potatura con metodo *costo-complessità* all'albero largo per ottenere una sequenza di sotto-alberi in funzione di α.
3. Uare la K-fold cv per scegliere α. Cioè, dividere il training set in K *folds*. Per ogni $k = 1, 2, \ldots, K$:

   - **(a)** Ripetere steps 1. e 2. su tutti tranne il k-esimo fold dei training data.
   - **(b)** Valutare l'errore di previsione medio quadratico sul k-esimo fold (test set) in funzione di α.
   - **(c)** Fare la media dei risultati per ogni valore di α, e selezionare α per minimizzare l'errore medio.

4. Restituire il sotto-albero dallo step 2. che corrisponde al valore scelto di α.

Esempio: Baseball data

- Data set `Hitters`: prevedere il salario `Salary` di alcuni giocatori in base agli anni `Years` giocati in Major League e numero di battute valide `Hits` realizzate nell'anno precedente.
- Log-trasformiamo il salario, che è misurato in migliaia di dollari, per motivi di simmetria della distribuzione.
- Nella prossima slide vediamo un semplice albero di regressione per il log-salario stimato tramite la funzione `tree` del medesimo pacchetto di `R`.

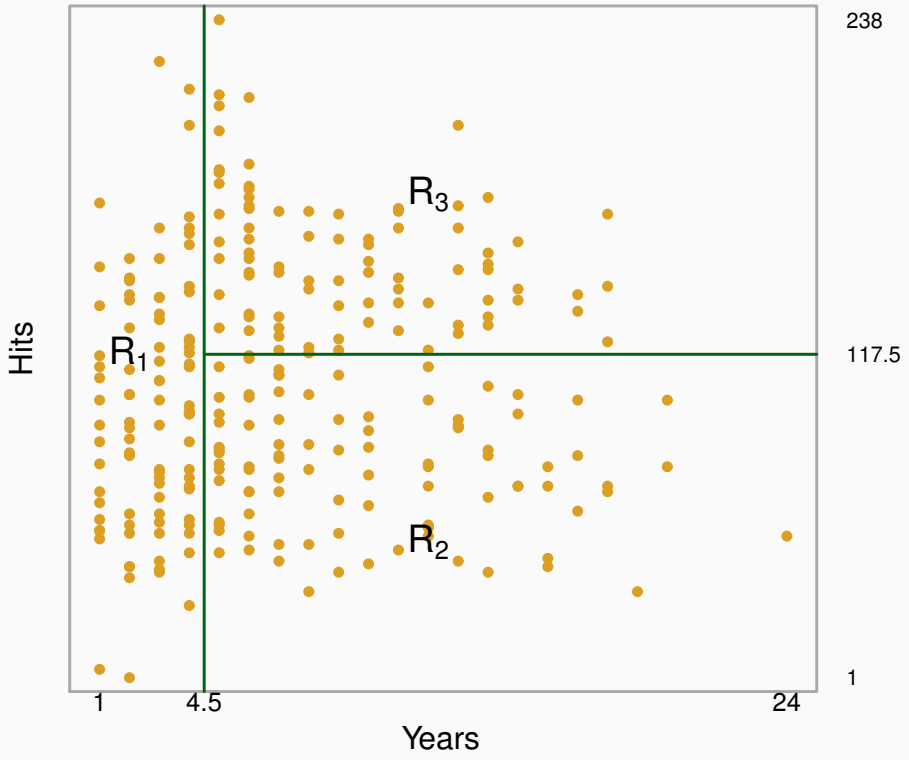
Esempio: Baseball data

Dettagli figura precedente

- L'albero ha due nodi interni, `Years < 4.5` e `Hits < 117.5`, e tre nodi terminali, detti anche *foglie*.
- Il log-salario previsto quando `Years < 4.5` è di 5.11, corrispondente a 165,174\$. Quando invece `Years >= 4.5` dobbiamo vedere cosa accade per la variabile `Hits`.
- In questo secondo caso, se `Hits < 117.5` il log-salario previsto è di 6 (403,428\$), mentre quando `Hits >= 117.5` il log-salario previsto è più alto, 6.74 (845,561\$).
- Otteniamo interpretativamente il risultato che il salario cresce al crescere degli anni giocati, che risulta la variabile più importante. Se un giocatore ha giocato abbastanza anni in Major League, il numero di battute valide dell'anno precedente influenza il salario positivamente. Se invece ha giocato relativamente meno anni, il numero di valide non influenza il salario in modo significativo.
- Modello over-semplificato, ma facile da visualizzare e da spiegare!

Esempio: Baseball data

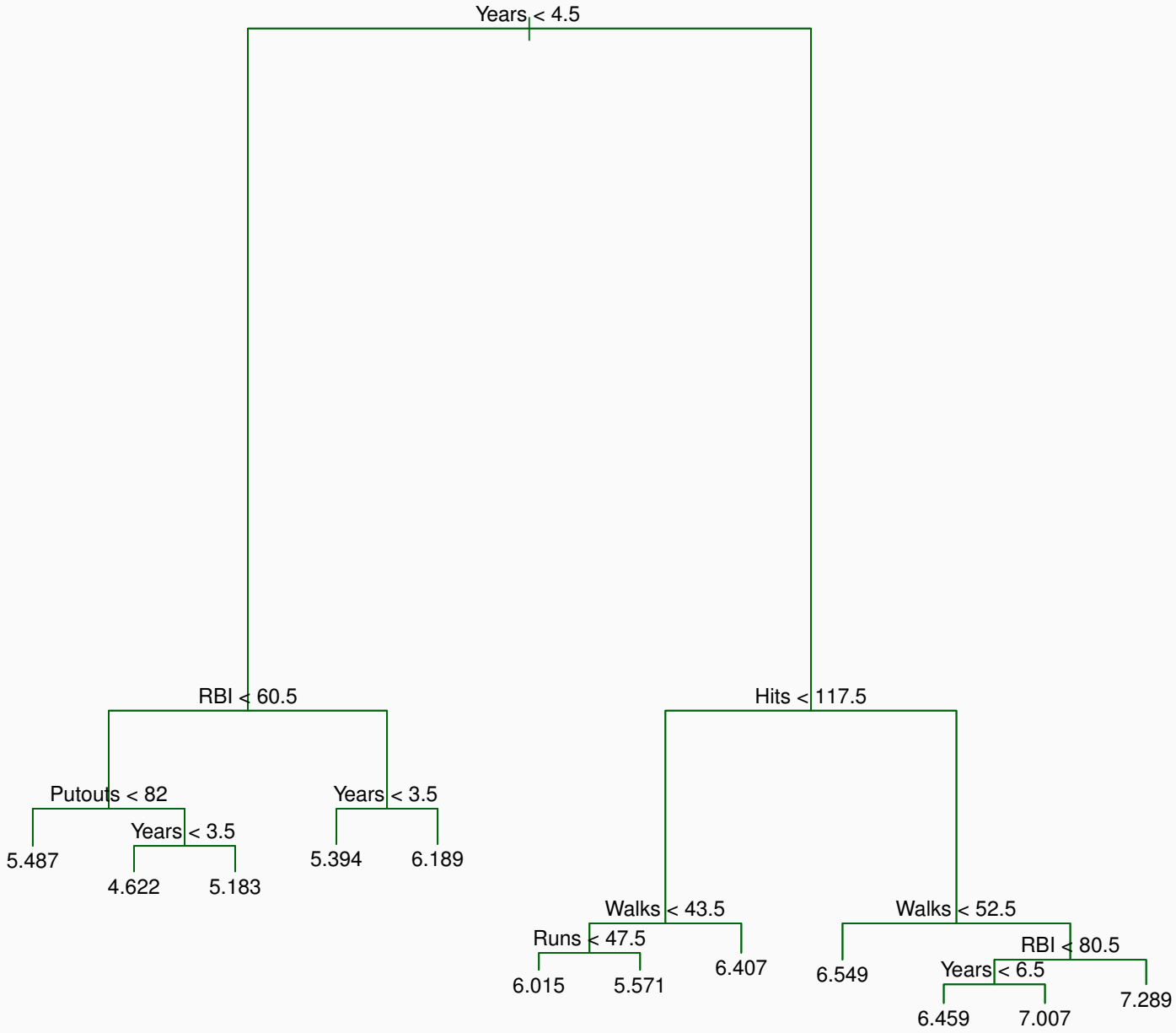
- L'albero *stratifica/segmenta* i giocatori in tre regioni dello spazio dei predittori:
  $R_1 = \{X | \texttt{Years < 4.5}\}$, $R_2 = \{X | \texttt{Years >= 4.5}, \texttt{Hits < 117.5}\}$ and $R_3 = \{X | \texttt{Years >= 4.5}, \texttt{Hits >= 117.5}\}$.



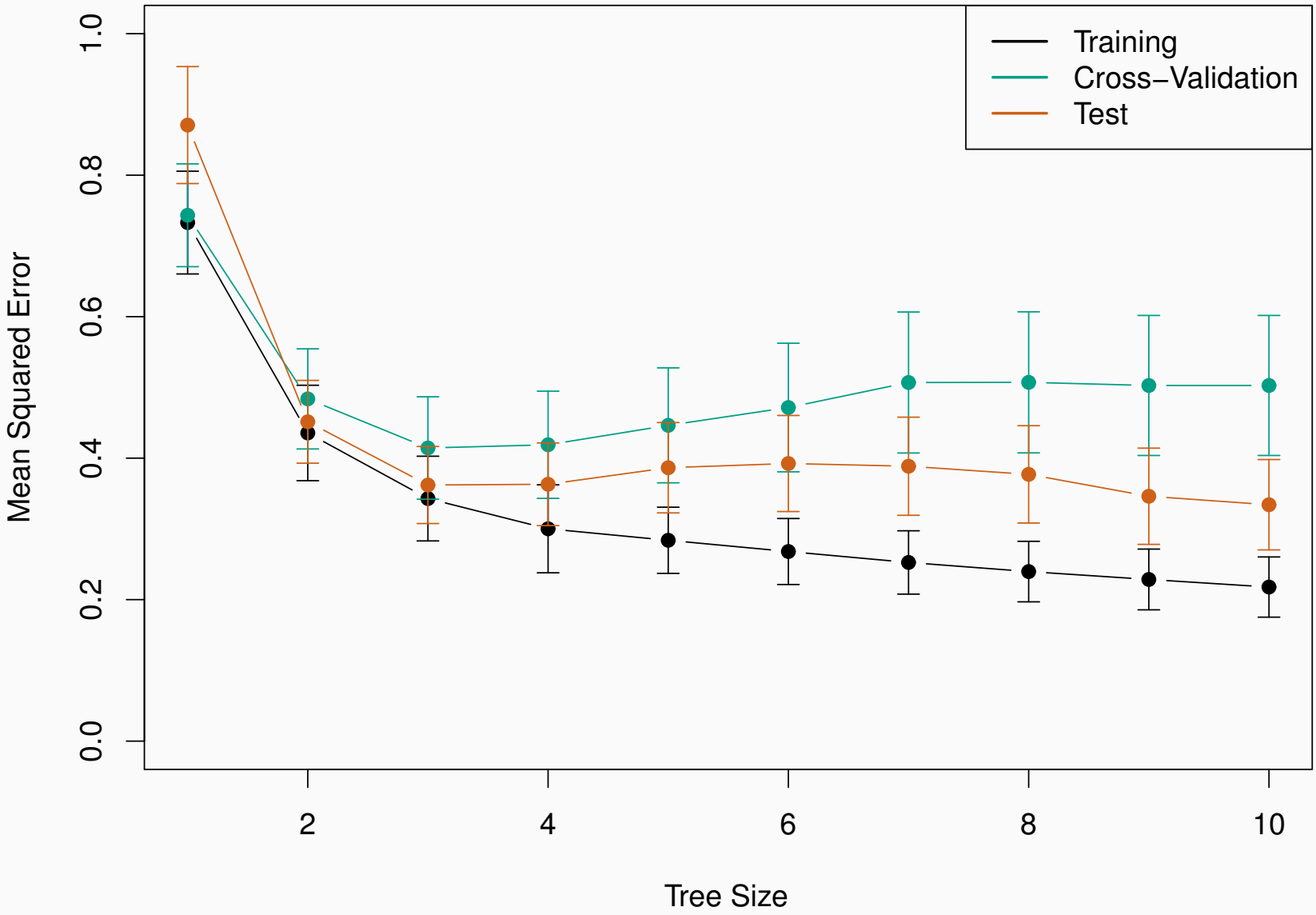
Esempio: Baseball data e pruning

- Prima di tutto dividiamo randomicamente il data set in due metà, con 132 osservazioni nel training set e 131 nel test set.
- Costruiamo un albero grande sui dati di training, usando stavolta 9 predittori, e facciamo variare il parametro α in modo da ottenere sotto-alberi di diversa grandezza (con diversi numeri di nodi terminali).
- Performiamo una 6-fold cv per stimare il cv MSE degli alberi in funzione di α.

Esempio: Baseball data e pruning



Esempio: Baseball data e pruning



Dettagli delle figure precedenti

- Nella penultima figura visualizziamo l'albero intero, non potato.
- La linea verde nell'ultima figura mostra l'errore di CV in funzione del numero di *foglie*, mentre la linea arancione indica l'errore sul test set. Vengono mostrate anche le barre degli standard errors. L'errore di CV è una buona approsimazione dell'errore sul test set: il minimo per la CV si ottiene con tre nodi terminali, mentre il test error minimo si ha per dieci nodi terminali. L'albero potato (o sotto-albero) con 3 foglie è proprio quello mostrato all'inizio dell'esempio.

Vantaggi . . .

- ▼ I metodi ad albero sono semplici e utili a livello interpretativo, nonché facili da spiegare. Addirittura, sono più semplici da spiegare a non esperti di quanto non lo sia la regressione lineare!
- ▼ Alcune persone pensano che i metodi ad albero, più ancora di altri metodi di regressione e classificazione, replichino il meccanismo di decisione umano.
- ▼ Gli alberi possono essere spiegati anche solo graficamente, specie ai non esperti.
- ▼ Gli alberti possono trattare facilmente predittori qualitativi senza il bisogno di creare variabili *dummy*.
- ▼ Scalabili per grandi datasets.

. . . e svantaggi

- ▼ Tuttavia, sfortunatamente spesso non sono competitivi se paragonati ad altri metodi di apprendimento supervisionato in termini di accuratezza predittiva. Per questa ragione, vedremo metodi per *combinare/far crescere più alberi* (metodi di insieme/ensemble) come ad esempio *bagging*, *random forests* e *boosting*. Tali metodi producono spesso un grande miglioramento predittivo fornendo una previsione *media* tra tutti gli alberi.
- ▼ Gli alberi possono essere *non robusti*. In altre parole, una piccola variazione nei dati può causare un grosso cambio nell'albero finale stimato.
- ▼ Non vi sono i canonici strumenti inferenziali, quali stima intervallare, test di ipotesi, selezione di modelli.
- ▼ Gli alberi molto grandi sono difficili da interpretare.

Splines di regressione multivariate adattive (MARS)

Splines di regressione multivariate adattive (MARS)

- Le *splines di regressione multivariate adattive* (MARS) sono una procedura adattiva di regressione idonea per problemi ad alta dimensionalità (alto numero di predittori).
- Possono essere viste come generalizzazione di una regressione lineare a tratti o modificazione di un CART. In questo secondo caso, si può vedere che ne migliorano l'accuratezza predittiva tramite superfici di previsione *più lisce* (smoother).
- Introduciamo le MARS dal primo punto di vista, ma poi facciamo la connessione con i CART. Si tratta di un metodo semi-parametrico (come i GAM) che come i CART usa un algoritmo grezzo e adatta ricorsivamente una curva alla superficie di regressione.
- Ad ogni passo viene scelta una coppia di funzioni di base selezionando ricorsivamente la variabile X maggiormente appropriata e la posizione ottimale del nodo.

Splines di regressione multivariate adattive (MARS)

- Le MARS usano espansioni di funzioni di base lineari a pezzi nella forma $(x-t)_+$, dove il '+' denota la parte positiva:

$$(x-t)_+ = \begin{cases} x-t & \text{if } x > t, \\ 0 & \text{otherwise} \end{cases}, \quad (t-x)_+ = \begin{cases} t-x & \text{if } x < t, \\ 0 & \text{otherwise} \end{cases}.$$

- Ogni funzione è lineare a tratti, con un *nodo* in t: si tratta quindi di *splines lineari*.
- L'idea è di formare delle basi 'riflesse'/simmetriche per ogni X_j con nodi nei valori osservati x_{ij}. La collezione di funzioni di base per ciascun t e j è data da $\mathcal{C} = \{(X_j - t)_+, (t - X_j)_+\}$.
- Il modello funziona come una regressione lineare a pezzi 'in avanti' e ha quindi la forma:

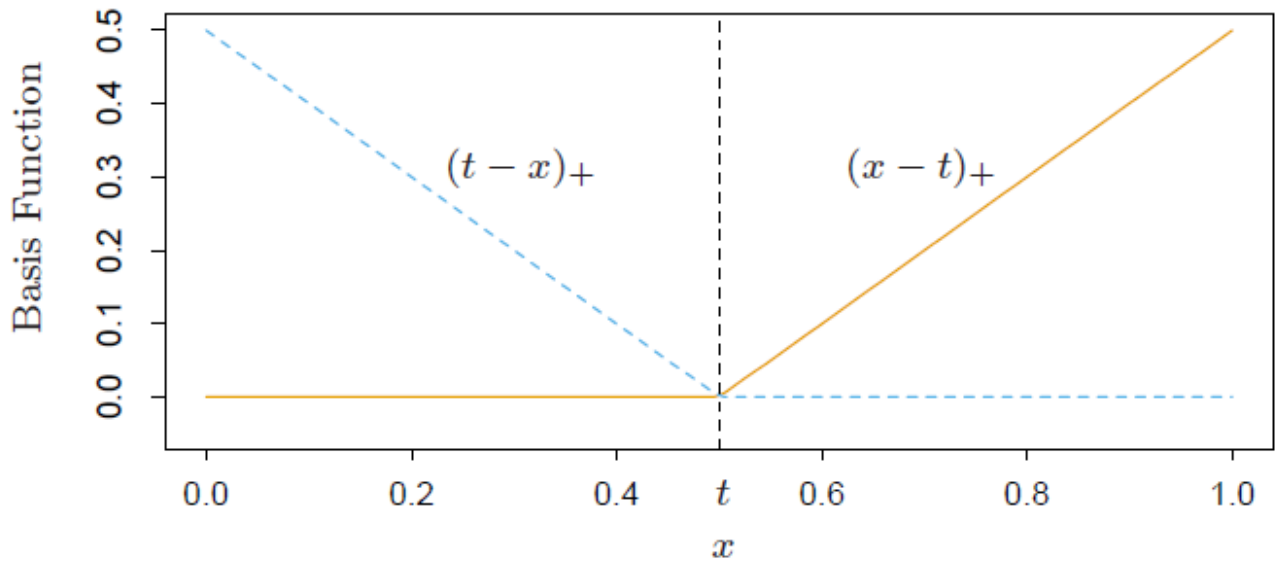
$$f(x) = \sum_{k=0}^{K} \beta_k b_k(x),$$

dove i β_k sono come al solito i coefficienti della curva di regressione e ogni $b_k(x)$ è una funzione di base scelta in $\mathcal{C}$, o un prodotto di due o più di tali funzioni.

Funzioni di base lineari a pezzi

- Ognuna di queste basi $b_k(x)$ può assumere una tra le tre seguenti forme:
  1. Una costante
  2. Una *hinge function*, di fatto quelle nella slide precedente, funzioni del tipo $\max(0, x - const)$, o $\max(0, const - x)$.
  3. Un prodotto di due o più *hinge functions*. Queste funzioni di base possono modellare l'interazione tra due o più variabili esplicative.

Esempio coppia hinge functions

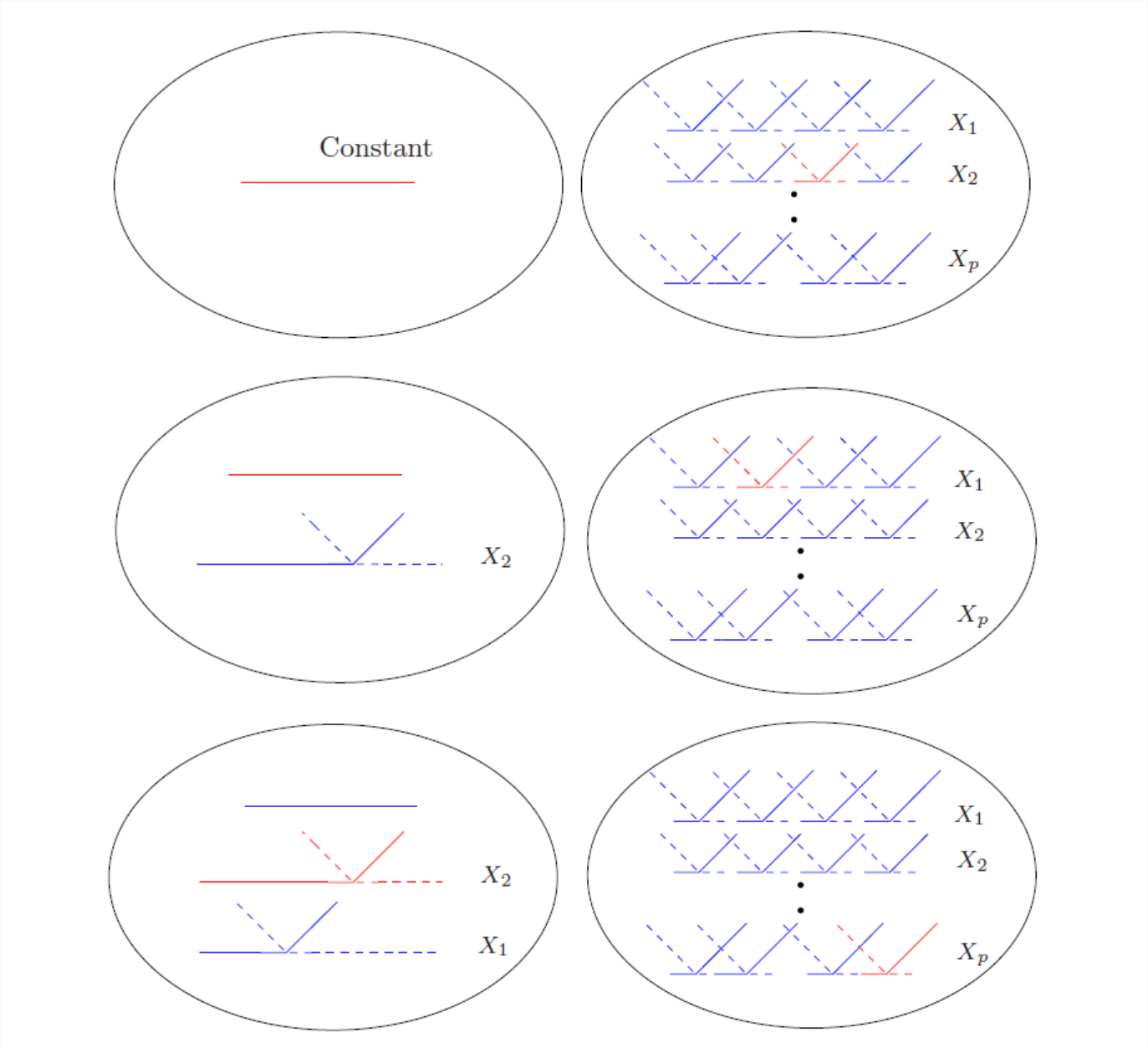


Costruzione adattiva MARS

1. Si inizia con solamente la funzione di base costante, $b_0(x) = 1$.
2. Ad ogni passo consideriamo come una nuova coppia di funzioni di base tutti i prodotti di una funzione di base già nel modello con una delle coppie simmetriche in $\mathcal{C}$, per ogni j.
3. Aggiungiamo al modello corrente il termine nella forma: $\hat{\beta}_{K+1} b_\ell(x)(X_j - t)_+ + \hat{\beta}_{K+2} b_\ell(x)(t - X_j)_+$ che produce la più larga riduzione nell'errore di training. Qui $\hat{\beta}_{K+1}$ e $\hat{\beta}_{K+2}$ sono stimati mediante i minimi quadrati con anche gli altri $K + 1$ coefficienti nel modello.
4. I prodotti 'vincenti' vengono aggiunti al modello e il processo continua fino a quando il modello contiene un numero massimo prefissato di termini.

Alla fine del processo abbiamo un modello per l'appunto della forma: $f(x) = \sum_{k=0}^{K} \beta_k b_k(x)$.

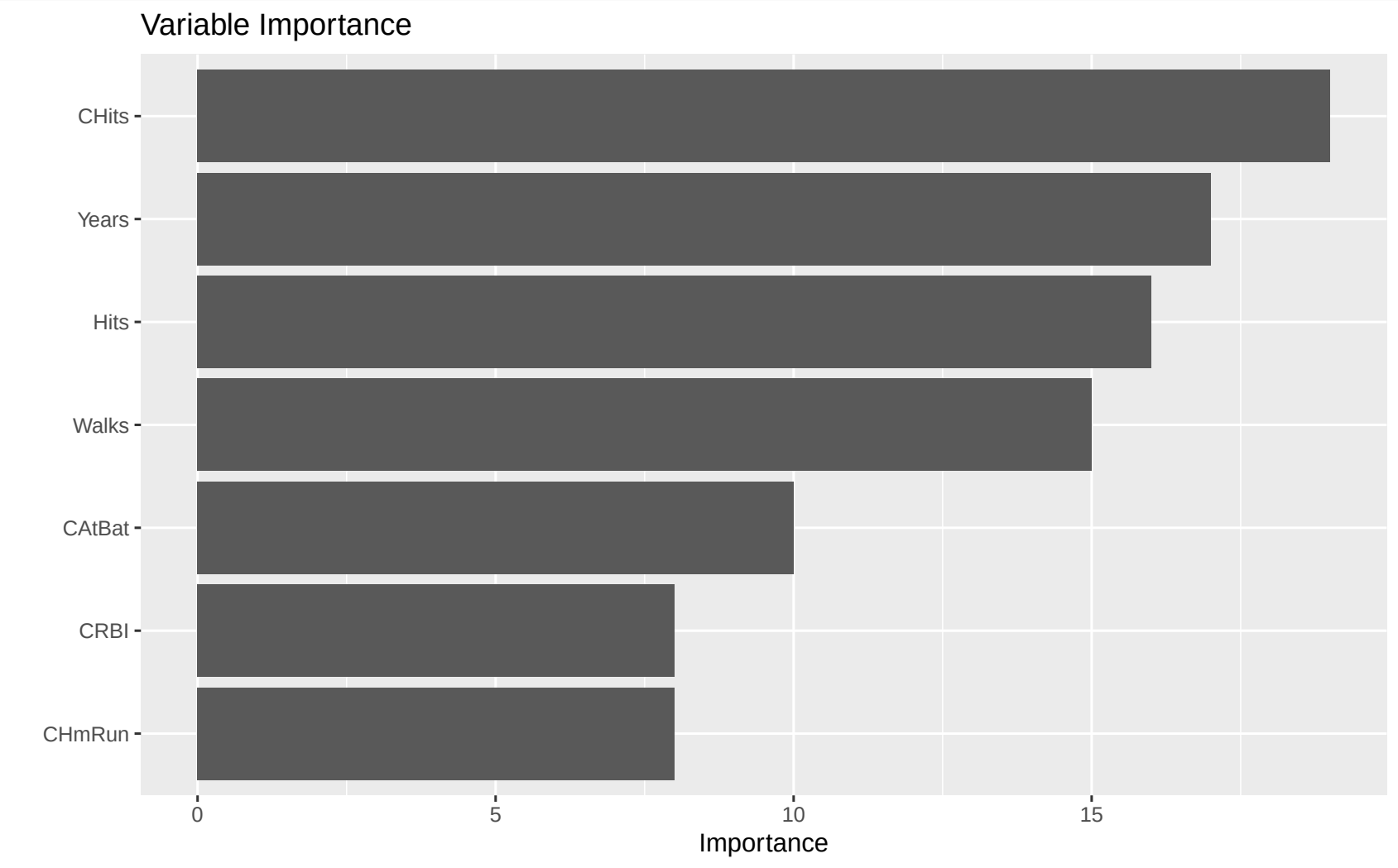
Costruzione adattiva MARS



Qualche commento sulla figura precedente e sulle MARS

- A sinistra: funzioni di base correntemente nel modello (all'inizio $b(x) = 1$). A destra: tutte le candidate funzioni di base, coppie di funzioni di base lineari a pezzi, con nodi t in tutti i valori unici di ciascun predittore X_j. Ad ogni passo consideriamo tutti i prodotti di una coppia candidata con una funzione di base presente nel modello. Il prodotto che decrementa di più l'errore di training viene aggiunto nel modello corrente (funzioni selezionate in arancione).
- Questo modello tipicamente *overfitta* i dati. Il termine la cui rimozione causa il più piccolo incremento nell'errore quadratico viene rimosso dal modello ad ogni passo, producendo un modello migliore stimato del tipo $\hat{f}_\lambda$, dove λ è la dimensione del modello. Per stimare λ si può usare CV.
- ▼ Queste funzioni di base lineari a pezzi operano localmente, quando moltiplicate assieme il risultato è diverso da zero solo in prossimità della piccola parte di spazio dei predittori dove entrambe le componenti sono diverse da zero. La superficie di regressione è dunque più 'liscia'.

Importanza delle variabili in un modello MARS: baseball data



Connessione con i CART

Supponiamo di prendere le MARS e applicare le seguenti modifiche:

- sostituire le funzioni di base lineari a pezzi con funzioni a gradini $I(x - t > 0)$ e $I(x - t \leq 0)$.
- Quando un termine del modello viene moltiplicato per un termine 'candidato', viene sostituito dall'interazione, e quindi non è più disponibile per ulteriori interazioni.

Con queste modifiche le MARS sono la stessa procedura dei CART! Moltiplicare una funzione a gradini per un paio di funzioni a gradino 'simmetriche' (hinge) è difatti equivalente a splittare un predittore ad ogni passo. La seconda restrizione equivale a chiedere che un nodo non possa essere splittato più di una volta, e porta esattamente alla rappresentazione binaria ad albero dei modelli CART. D'altra parte, è proprio questa restrizione che rende difficile modellare strutture additive nei CART. Le MARS superano invece questa difficoltà e possono catturare effetti additivi agevolmente.