Università degli studi di Padova
Dipartimento di Scienze Statistiche
Corso di Laurea Magistrale in

Scienze Statistiche


Relazione Finale

# DEEP RANDOM FOREST: UN NUOVO METODO DI ENSEMBLE LEARNING

Relatore Prof. Livio Finos
Correlatore Dario Solari
Dipartimento di Scienze Statistiche

Laureando Alessandro Blanda
Matricola N 1084025

Anno Accademico 2014/2015

# Indice

# Elenco delle figure

# Elenco delle tabelle

# Introduzione

L'*ensamble learning* usa differenti algoritmi derivanti dal *machine learning* e dalla statistica per ottenere *performance* previsive più accurate.
È difficile rintracciare il punto d'inizio dell'*ensamble learning* poiché l'idea di sviluppare diversi modelli decisionali per il medesimo scopo è stata utilizzata dalla società umana fin da tempi remoti: già il filosofo greco Epicuro (341-270 a.C.) aveva introdotto il principio di spiegazioni multiple, il quale auspicava di tenere in considerazione tutti i modelli che sono coerenti con i dati osservati (Lucretius Carus, 2012).
L'*ensemble learning* si è poi sviluppato a partire dagli anni 90', quando è stato introdotto lo *Stacking* (Wolpert, 1992), il *Bagging* (Breiman, 1996a), la *Random Forest* (Breiman, 2001) e infine l'*Extra-trees* (Geurts, Ernst e Wehenkel, 2006).
Sebbene l'obiettivo principale dell'*ensemble learning* sia la maggiore accuratezza possibile, i precedenti modelli non riescono a competere con la rete neurale multistrato (*deep neural network*) in vari ambiti, ad esempio la classificazione di immagini (LeCun et al., 1998) e il riconoscimento vocale (Graves, Mohamed e Hinton, 2013).
Dall'altro canto la rete neurale presenta numerosi difetti. Primo, sono usualmente richiesti un grande ammontare di dati per la stima (Goodfellow, Bengio e Courville, 2016), rendendo inadatto il modello su piccoli *dataset*[1]. Secondo, si richiedono ingenti risorse computazionali. Infine, la rete neurale è composta da numerosi *iper*-parametri che influenzano pesantemente la *performance* del modello e cambiano enormemente da un contesto all'altro, rendendo lo

[1]Anche nell'era dei *big data* (Goodfellow, Bengio e Courville, 2016) continuano ad essere a disposizione un numero limitato di osservazioni in certi ambiti dovuti agli alti costi di raccolta

strumento accessibile solamente ad un utente esperto.
Verrà quindi esposto un nuovo metodo *ensemble* (Zhou e Feng, 2017) chiamato *Deep random forest* che si propone di ottenere risultati migliori con un costo computazionale uguale o inferiore e una regolazione dei parametri sostanzialmente nulla, rendendo lo strumento accessibile a ogni tipo di utente.
Nel Capitolo 1 vengono descritti i principali modelli *ensemble* esistenti e proprietà associate.
Il Capitolo 2 è la parte centrale, dove il *Deep random forest* è presentato mediante una formulazione matriciale. Verrà quindi testata quanto la *performance* è influenzata dalla scelta dei parametri di regolazione verificando in particolare la validità delle impostazioni di *default* definita dall'autore in 3 differenti studi di simulazione.
Nel Capitolo 3 vengono analizzati 3 differenti *dataset* appartenenti rispettivamente a un problema di regressione, classificazione e multiclassificazione confrontando la potenza predittiva del *Deep random forest* con i modelli sopra citati. In particolare il 3° studio rientra nel campo del *deep learning* dove la rete neurale ha una prestazione altamente competitiva.
Il Capitolo si conclude con una descrizione delle tecniche con cui è possibile parallelizzare il *Deep random forest* e un resoconto dei tempi computazionali in base alle librerie utilizzate.
Segue una breve conclusione del lavoro svolto.

# Capitolo 1

# I metodi *ensemble*

L'*ensemble learning* stima differenti metodi di apprendimento derivanti dal *machine learning* e dalla statistica per poi combinarli tra loro e ottenere un modello finale (*ensemble model*) dalla potenza predittiva maggiore rispetto ai singoli modelli di partenza (*base learners*).
La figura Figura 1.1 mostra una comune architettura *ensemble*. Quest'ulti-

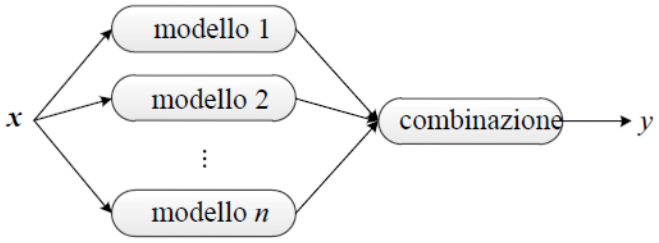


**Figura 1.1:** L'architettura *ensemble*

ma contiene un certo numero di modelli di base (1° strato), seguiti da una procedura che li combina assieme e fornisce la previsione finale. Se *ensemble model* utilizza un unico tipo di *base learner* il modello finale viene definito *omogeneo*, altrimenti si è in presenza di un insieme *eterogeneo*.
Anche utilizzando un insieme omogeneo composto da *base learners* con alto livello interpretativo (ad esempio, l'alberi decisionali), si perderà parte dell'interpretabilità sperando in un aumento Questo porta a definire il dilemma di *Occam* (Breiman, 2002):

> L'accuratezza richiede, generalmente, modelli previsivi maggiormente complessi. Le funzioni semplici e interpretabili non generano le migliori previsioni.

Questo principio potrebbe sembrare un controsenso rispetto al compromesso varianza-distorsione (più il modello è complesso, più la varianza cresce, cfr. Azzalini e Scarpa, 2009, cap. 3), tuttavia Cohen e Jensen (1997) dimostrano che il sovradattamento non nasce per la complessità del modello in sé, ma perché stimare un numero elevato di modelli porta con alta probabilità a trovare un modello che si adatta bene ai dati puramente grazie al caso. Quindi provare 10 modelli complessi comporta un minor rischio di *overfitting* piuttosto che provarne 100 semplici.
I metodi *ensemble* possono essere divisi in 3 categorie:

- Combinazione di modelli.
- Unione di modelli deboli.
- Mistura di esperti.

*Combinazione di modelli* è rappresentata dalla Figura 1.1, precedentemente descritta, la cui caratteristica principale è l'indipendenza tra i modelli di base che possono essere stimati in parallelo.
*Unione di modelli deboli* combina modelli leggermente più accurati di una scelta casuale, cercando di creare un modello finale con alto valore predittivo. Caratteristiche tipiche sono la sequenzialità in cui un modello cerca di correggere gli errori della precedente stima.
*Mistura di esperti* (ME) è un efficace approccio per impiegare molteplici modelli di apprendimento. Se la vera funzione generatrice dei dati è particolarmente irregolare può essere utile dividere lo spazio campionario e stimare un *ensemble* formato da un insieme di *base learners*, definiti esperti, e un sistema di pesi (*gating networks*) che sceglie quale modello deve essere utilizzato in ogni porzione.
Si suppone di volere combinare $K$ esperti, si definisce:

- $\hat{Y}_j \sim \mathbb{P}(y|\mathbf{x}_i, \hat{\boldsymbol{\theta}}_j)$ la distribuzione condizionata della previsione associata al $j$-esimo esperto dato il valore della $i$-esima osservazione e del vettore di parametri stimati $\hat{\boldsymbol{\theta}}_j$.

- $p_j$ il $j$-esimo peso.

La mistura di esperti è pari a:

$$\mathbb{P}(y|\mathbf{x}_i, \Psi) = \sum_{j=1}^{K} \hat{p}_j \mathbb{P}(y|\mathbf{x}_i, \hat{\boldsymbol{\theta}}_j). \tag{1.1}$$

dove $\Psi$ rappresenta l'insieme di tutti i parametri. Spesso $p_j$ è specificato tramite la funzione *softmax*:

$$p_j = \frac{e^{\boldsymbol{\gamma}_j^T \mathbf{x}_i}}{\sum_{k=1}^{K} e^{\boldsymbol{\gamma}_j^T \mathbf{x}_i}} \tag{1.2}$$
$$p_j \geq 0, \sum_{j=1}^{K} p_j = 1$$

dove $\boldsymbol{\gamma}_j$ rappresenta il vettore dei parametri ignoti.
Il metodo più conveniente per massimizzare la log-verosimiglianza associata è tramite l'algoritmo EM (*Expectation-Maximization*, si veda (Hastie, Tibshirani e Friedman, 2009), pag. 329-332).
Le prime 2 categorie di modelli *ensemble* stimano un insieme di modelli per il medesimo problema, quindi i *base learners* saranno altamente correlati e un problema chiave è come aumentare la loro indipendenza. Al contrario, nella mistura di esperti questo aspetto non è importante, il problema chiave è individuare la miglior suddivisione dello spazio campionario, quindi scegliere i modelli appropriati corrispondenti.
Il *Deep random forest* (Cap. 2) è una combinazione delle prime 2 categorie: rispetto alla Figura 1.1, non si ha uno, ma molteplici strati, di cui ognuno viene stimato sulla base dei precedenti e delle variabili iniziali. I modelli di base potranno essere ancora stimati in parallelo all'interno di ogni strato, ma non tra strati differenti.
Di seguito si illustrerà i principali modelli *ensemble* e proprietà associate.

## 1.1 Bagging

Si supponga di dover affrontare un problema di classificazione binaria {1,0}, che la vera funzione generatrice dei dati sia $f(x)$ e che siano stati

stimati $T$ classificatori indipendenti ognuno con errore pari a $\epsilon$, i.e., per il $j$-esimo classificatore $C_j$, valga:

$$\mathbb{P}\{\hat{C}_j(\mathbf{x}_i) \neq f(\mathbf{x}_i)\} = \epsilon \tag{1.3}$$

dove $\mathbf{x}_i$ indica il vettore della $i$-esima osservazione e $\hat{C}_j(\mathbf{x}_i)$ la previsione corrispondente del classificatore $j$-esimo. Dopo aver combinato i $T$ classificatori di base mediante la formula

$$\hat{C}^T(\mathbf{x}_i) = \left\lfloor \frac{1}{T} \sum_{j=1}^{T} \hat{C}_j(\mathbf{x}_i) \right\rceil \tag{1.4}$$

la stima finale sarà pari alla classe maggiormente predetta.
L'insieme $C^T$ commetterà un errore solo se la maggior parte dei classificatori farà un errore. Perciò, attraverso la *disuguaglianza* di *Hoeffding*, l'errore atteso è

$$\mathbb{P}\{\hat{C}^T(\mathbf{x}_i) \neq f(\mathbf{x}_i)\} = \sum_{k=0}^{T/2} \binom{T}{k} (1-\epsilon)^k \epsilon^{T-k} \leq exp\left(-\frac{1}{2} T(2\epsilon - 1)^2\right). \tag{1.5}$$

La formula mostra che l'errore si riduce esponenzialmente all'aumentare di $T$, tendendo a 0 quanto $T$ tende all'infinito.
In pratica è impossibile ottenere modelli indipendenti che sono generati dal medesimo insieme di stima. Si cerca quindi di ridurre la dipendenza utilizzando $T$ diversi campioni di stima generati da un campionamento con reinserimento dei dati originali (*bootstrap*), compare così il *Bagging* (*Bootstrap AGGregatING*), si definisce:

- $\mathbf{x}_i = [x_1 \cdots x_p]$: vettore della $i$-esima osservazione avente $p$ caratteristiche.
- $\mathbf{X}^T = [\mathbf{x}_1^T \cdots \mathbf{x}_N^T]$: matrice degli *input* avente $N$ osservazioni.
- $\mathbf{y}^T = [y_1 \cdots y_N]^T$: vettore della variabile risposta.
- $\mathbf{Z} = [\mathbf{y} \quad \mathbf{X}]$: *dataset* di stima.
- $C$: generico modello di apprendimento.
- $\hat{C}(\mathbf{x}_i)$: stima associata all'$i$-esima osservazione dato il modello $C$.

L'algoritmo *Bagging* effettua $B$ campionamenti *bootstrap* di $\mathbf{Z}$ adattando ad ognuno il modello $C$ scelto. Si ottengono così $B$ nuove stime di previsione $\hat{C}_b^*(\mathbf{x}_i)$ $(b = 1, ..., B)$ della $i$-esima osservazione. Quest'ultime vengono aggregate formando la stima *Bagging*:

$$\hat{C}_{bag}(\mathbf{x}_i) = \frac{1}{B}\sum_{b=1}^{B}\hat{C}_b^*(\mathbf{x}_i). \tag{1.6}$$

Nel caso della classificazione, la previsione sarà 1 se $\hat{C}_{bag}(\mathbf{x}_i) > 0.5$, 0 negli altri casi.
Se il modello $C$ è insensibile a perturbazioni dei dati, tutti i $B$ modelli associati avranno previsioni piuttosto simili, e combinandoli tra loro la *performance* complessiva non si discosterà di tanto rispetto a quella iniziale. Tale modello viene definito stabile, ad esempio il *k-nearest neighbor* e il modello lineare. Al contrario, più il modello è instabile, ad esempio l'albero decisionale, maggiore sarà il miglioramento.
Esplicando la questione in termini formali si dimostra (Hastie, Tibshirani e Friedman, 2009, paragrafo 2.5) che l'errore quadratico medio (MSE) può essere scisso nel seguente modo:

$$\mathbb{E}\{[\hat{Y} - f(\mathbf{x}_i)]^2\} = [\mathbb{E}\{\hat{Y}\} - f(\mathbf{x}_i)]^2 + Var\{\hat{Y}\} = \text{distorsione}^2 + \text{varianza} \tag{1.7}$$

dove $\hat{Y} = \hat{C}_{bag}(\mathbf{x}_i)$ e $f(x)$ è la vera funzione generatrice dei dati.
L'idea essenziale del *Bagging* è combinare diversi classificatori instabili, ma approssimativamente non distorti, per ridurre la varianza. Gli alberi sono candidati ideali per il *Bagging*, poiché possono catturare relazioni complesse nei dati, e se crescono fino alle foglie, hanno bassa distorsione.
Per illustrare il meccanismo di diminuzione della varianza, si pensi che $\hat{C}_1^*(x)$,..., $\hat{C}_B^*(x)$ siano variabili casuali di media $\mu$ e varianza $\sigma^2$, identicamente distribuite ma non necessariamente indipendenti, con indice di correlazione pari a $\rho$. La loro media $\hat{C}_{bag}(x)$ avrà varianza e valore atteso pari a

$$Var\{\hat{C}_{bag}(x)\} = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \tag{1.8}$$

$$\mathbb{E}\{\hat{C}_{bag}(x)\} = \mu. \tag{1.9}$$

All'aumentare di $B$ il secondo termine della 1.8 scomparirà e $\rho\sigma^2 < \sigma^2$, mentre $\mu$ rimane costante. Quindi il miglioramento nelle previsioni ottenute da

un modello *Bagging* sono solamente il risultato di una diminuzione della varianza.
Il valore della Formula 1.8 deve essere confrontato con la varianza dello stimatore iniziale $\hat{C}(x)$, in quanto la distorsione e varianza di $\hat{C}_b^*(x)$ è tipicamente più grande a causa della casualità e della conseguente riduzione dello spazio campionario.
Ad esempio si supponga che il processo generatore dei dati sia $Y \sim \mathcal{N}(\mu, \gamma^2)$. Sia $\bar{Y}$ lo stimatore media campionaria, $Y_i^{(b)}$ la realizzazione $i$-esima del $b$-esimo campione *bootstrap* e $\bar{Y}_b^*$ la media corrispondente. Ora, per le proprietà della funzione di ripartizione empirica (si veda Pace e Salvan, 1996, pag. 67), $\mathbb{E}\{Y_i^{(b)}|Y_1, ..., Y_N\} = \bar{Y}$. Quindi il valore atteso marginale risulta

$$\begin{aligned} \mathbb{E}\{Y_i^{(b)}\} &= \mathbb{E}\{\mathbb{E}\{Y_i^{(b)}|Y_1, ..., Y_N\}\} = \mathbb{E}\{\bar{Y}\} \\ &= \mu \end{aligned} \tag{1.10}$$

e la varianza

$$Var\{Y_i^{(b)}|Y_1, ..., Y_N\} = \frac{\sum_{i=1}^N (Y_i - \bar{Y})^2}{N} = \frac{N-1}{N} S^2 \tag{1.11}$$

dove $S^2 = \sum_{i=1}^N (Y_i - \bar{Y})^2/(N-1)$. Quindi,

$$\begin{aligned} Var\{Y_i^{(b)}\} &= \mathbb{E}\{Var\{Y_i^{(b)}|Y_1, ..., Y_N\}\} + Var\{\mathbb{E}\{Y_i^{(b)}|Y_1, ..., Y_N\}\} \\ &= \gamma^2 \left( \frac{N-1}{N} + \frac{1}{N} \right). \end{aligned} \tag{1.12}$$

Infine la covarianza tra 2 realizzazioni differenti è pari a

$$\begin{aligned} cov(Y_i^{(b)}, Y_j^{(k)}) &= \mathbb{E}\{[Y_i^{(b)} - \mu][Y_j^{(k)} - \mu]\} \\ &= \mathbb{E}\{\mathbb{E}\{Y_i^{(b)} Y_j^{(k)}|Y_1, ..., Y_N\}\} - \mu^2 \\ &= \mathbb{E}\{\bar{Y}^2\} - \mu^2 \\ &= \frac{\gamma^2}{N} \end{aligned} \tag{1.13}$$

con $i \neq j$. La Formula 1.13 vale sia per variabili all'interno dello stesso campione che tra campioni differenti.
Quindi una prima conclusione importante: sebbene le osservazioni siano iid tra loro, le realizzazioni *bootstrap* non condizionate perdono questa proprietà.

Attraverso ulteriori passaggi si ricava la varianza del $b$-esimo stimatore *bootstrap* e la covarianza tra i differenti stimatori:

$$\begin{aligned} Var\{\bar{Y}_b^*\} &= \frac{1}{N}\left(N \cdot Var\{Y_i^{(b)}\} + N(N-1) \cdot cov(Y_i^{(b)}, Y_j^{(b)})\right) \\ &= \gamma^2\left(\frac{2N-1}{N^2}\right) \end{aligned} \tag{1.14}$$

$$cov(\bar{Y}_b^*, \bar{Y}_k^*) = \frac{1}{N^2}\left(\sum_{i=1}^{N}\sum_{j=1}^{N} cov(Y_i^{(b)}, Y_j^{(k)}\right) = \frac{\gamma^2}{N}. \tag{1.15}$$

Si hanno quindi tutti gli elementi della Formula 1.8 per calcolare la varianza dello stimatore *Bagging*:

$$Var\{\bar{Y}_{bag}\} = \frac{\gamma^2}{N}\left(1 + \frac{N-1}{N \cdot B}\right) \tag{1.16}$$

che all'aumentare di B la varianza convergerà a quella dello stimatore $\bar{Y}$. Quindi

$$\lim_{B \to +\infty} \bar{Y}_{bag} = \bar{Y}. \tag{1.17}$$

Più in generale si dimostra (Friedman e Hall, 2007) che lo stimatore *Bagging* avrà varianza minore solamente per le componenti a partire dal secondo ordine, mentre la componente lineare non sarà influenzata.
Ulteriore vantaggio è l'incapacità di sovradattamento all'aumentare di modelli di base $B$ (Breiman, 2001). La Formula 1.6 mantiene tutte le proprietà della media, quindi, sotto miti condizioni di regolarità, secondo la legge dei grandi numeri lo stimatore *Bagging* convergerà in probabilità al valore atteso della distribuzione *bootstrap*, i.e.

$$\lim_{B \to +\infty} \mathbb{P}\{|\hat{C}_{bag}(x) - \mathbb{E}\{\hat{C}_b^*(x)\}| < \epsilon\} = 1 \tag{1.18}$$

dove $\epsilon$ indica un valore infinitesimale.

### 1.1.1 Campioni Out-of-bag

A causa del campionamento casuale con reinserimento, in ciascun campione *bootstrap* una parte dei dati dell'insieme originale di stima viene esclusa, l'insieme di osservazioni omesse viene definito campione *out-of-bag* (OOB).

Si denoti $OOB_i$ l'insieme dei modelli dove $\mathbf{x}_i$ è presente nel campione OOB e $|OOB_i|$ la cardinalità dell'insieme, allora la stima OOB per l'osservazione $\mathbf{x}_i$ è pari a

$$\hat{C}_{OOB}(\mathbf{x}_i) = \frac{\sum_{b \in OOB_i} \hat{C}_b^*(\mathbf{x}_i)}{|OOB_i|}. \tag{1.19}$$

Nel caso della classificazione, la classe OOB predetta sarà 1 se $\hat{C}_{OOB}(\mathbf{x}_i) > 0.5$, 0 altrimenti.
Qualora il modello di apprendimento $C$ scelto sia pari all'albero decisionale, si può mostrare che l'errore OOB tende all'errore *leave-one-out cross validation* (vedi Paragrafo 1.4) quando $B$ tende all'infinito.
Infatti sebbene un campione *boostrap* contenga osservazioni replicate, quest'ultime risultano ininfluenti durante la crescita dell'albero (si veda Azzalini e Scarpa, 2009, pag. 96 e 147), quindi data la definizione precedente di previsione OOB, basta calcolare in quanti campioni OOB, in media, farà parte ciascuna osservazione.
Inizialmente bisogna calcolare la probabilità che l'osservazione $\mathbf{x}_i$ appartenga al campione *bootstrap* $b$, ovvero:

$$\begin{aligned} \mathbb{P}\{\text{osservazione } \mathbf{x}_i \in \text{campione } \textit{bootstrap } b\} &= 1 - \left(1 - \frac{1}{N}\right)^N \\ &\approx 1 - e^{-1} \\ &= 0.632 \end{aligned} \tag{1.20}$$

Quindi $|OOB_i|$ è una variabile casuale distribuita come:

$$|OOB_i| \sim Binomiale(B, 1 - 0.632)$$

con

$$\mathbb{E}\{|OOB_i|\} = 0.368 \cdot B \tag{1.21}$$

quindi l'errore OOB è pari all'errore *leave-one-out cross validation* utilizzando il 37% degli alberi.

## 1.2 Random forest

Il *Bagging* si è visto che combina diversi modelli facendo variare in ognuno l'insieme delle unità statistiche su cui viene effettuata la stima.

Un'altra possibilità è considerare per ogni *base learner* un diverso sottoinsieme delle variabili esplicative.
La *Random forest* (Breiman, 2001) combina questi 2 approcci: scegliendo come modello di apprendimento $C$ l'albero decisionale, ogni albero viene fatto crescere selezionando ad ogni nodo $m < p$ caratteristiche e utilizzando un campione *bootstrap* come insieme di stima. Le previsioni saranno poi aggregate secondo la Formula 1.6.
Se precedentemente si è mostrato come il *Bagging* abbia una varianza ridotta rispetto al singolo classificatore, la *Random Forest* riduce ulteriormente questa varianza facendo diminuire $\rho$ (Formula 1.8). A tale scopo, Ricordando la Formula 1.7 la distorsione aumenterà, al contempo però con la diminuzione della varianza.
Oltre alle proprietà già citate del *Bagging* (assenza di *overfitting* e riduzione della varianza rispetto ai modelli di base), la *Random forest* possiede ulteriori proprietà verificate sperimentalmente (Breiman, 2001):

- Migliore accuratezza e tempi computazionali inferiori rispetto al *Bagging*.
- Robustezza rispetto a perturbazioni della variabile risposta [1].
- Insensibilità rispetto all'aggiunta di variabili incorrelate con la risposta.
- Nei problemi di regressione all'aumentare di $m$ la correlazione $\rho$ cresce più lentamente rispetto ai problemi di classificazione.
- Con un aumento di variabili incorrelate con la risposta, il valore ottimale di $m$ sale in quanto c'è una maggiore probabilità di filtrare le variabili irrilevanti, portando a una significativa diminuzione della distorsione che ripaga l'aumento della varianza.

La *Random Forest* ha 3 parametri di regolazione: numero di alberi $B$, minima ampiezza dei nodi terminali $n_{min}$ e numero di variabili campionate in ogni nodo $m$. L'assenza di *overfitting* rende i primi 2 parametri importanti solo

[1] Per il problema di classificazione, Breiman (2001) ha testato che l'errore della *Random Forest* subisce piccole variazioni cambiando la classe della variabile risposta ogni 20 osservazioni (5% di rumore).

dal punto di vista computazionale, mentre l'errore di previsione è piuttosto insensibile a $m$ [2], di *default* si seguono le seguenti linee guida:

- Nei problemi di classificazione si imposta $m = \sqrt{p}$ e minima ampiezza dei nodi terminali pari a 1.
- Per la regressione, si imposta $m = p/3$ e minima ampiezza dei nodi terminali pari a 5.
- Numero di alberi non inferiore a 128 (Oshiro, Perez e Baranauskas, 2012).

Infine Breiman (2001) afferma che in presenza di variabili categoriali già codificate con *I-1 dummy*, dove $I$ è il numero di categorie, il valore di $m$ deve essere di conseguenza raddoppiato o triplicato, in modo che la probabilità di scegliere tutte le *I-1* categorie sia non troppo inferiore a scegliere la singola variabile non codificata.

## 1.3 Extra-trees

Nel paragrafo 1.1 si è valutata la possibilità di introdurre casualità nel modello attraverso il campionamento delle osservazioni, mentre nel paragrafo 1.2 è stato esposto un modo di campionare le variabili esplicative.
Anche se colonne o righe di una matrice di dati vengono estratte casualmente, l'algoritmo di crescita dell'albero rimane deterministico nelle sue caratteristiche chiavi.
Un differente approccio è rendere casuale l'intero algoritmo in modo che vengano generate partizioni casuali dello spazio campionario, indipendenti dalla variabile risposta.
L'algoritmo *Extra-Trees* (*EXTRemely rAndomized Trees*, Geurts, Ernst e Wehenkel, 2006) costruisce un *ensemble* di alberi decisionali non potati. Rispetto ai 2 metodi classici, le principali differenze sono la scelta parzialmente causale degli *split* e l'utilizzo dell'intero campione per la crescita degli alberi.

[2] Secondo Breiman (2001) la differenza percentuale tra l'errore associato a $m$=1 e quello associato a $\log_2 m + 1$ è inferiore all'1%.

**Algoritmo 1.1** Extra-trees per la classificazione e regressione.

1. Ciclo da $b = 1$ a $B$:
   (a) Costruire un *Extra-Trees* ripetendo in maniera ricorsiva il seguente *step* per ogni nodo terminale dell'albero finché la minima ampiezza $n_{min}$ è raggiunta.
      i. Selezionare casualmente $m$ variabili tra le $p$ a disposizione.
      ii. Ciclo da $j = 1$ a $m$:
         A. Se la variabile è continua:
            - Generare casualmente lo *split* da un'uniforme $U \sim (a_{min}, a_{max})$, dove $a_{max}$ e $a_{min}$ sono rispettivamente il massimo e il minimo della $j$-esima variabile.
         B. Se la variabile è categoriale:
            - Determinare lo *split* scegliendo una categoria da una multinomiale equiprobabile.
         C. Scegliere tra gli $m$ *split* quello che rende minima la funzione di perdita.
2. Restituire l'insieme di alberi $\{\hat{C}_b^*\}_1^B$. Le previsioni risultanti saranno aggregate mediante la Formula 1.6.

La procedura è descritta in dettaglio nell'algoritmo 1.1: ad ogni nodo vengono scelte casualmente $m$ variabili, si generano rispettivamente $m$ *split* casuali e, in maniera deterministica, viene scelto il migliore tra l'insieme formato.
Aggregando poi i $B$ alberi stimati secondo la Formula 1.6, l'*Extra-Trees* mantiene la proprietà di assenza di *overfitting* ad eccezione di quando $m = 1$: in questo caso l'*Extra-trees* viene denominato *Complete-Random trees* e gli *split* sono totalmente casuali, quindi l'errore del modello non converge ad un valore stabile e l'equazione 1.20 non è più rispettata.
Dal punto di vista del compromesso varianza-distorsione (Formula 1.7), la logica dietro il metodo *Extra-trees* è che l'esplicita casualità dei punti di taglio creerà alberi maggiormente incorrelati sperando che la conseguente diminu-

zione di varianza (Formula 1.8) copra l'aumento di distorsione, contrastata a sua volta dall'uso del campione di stima originale. Risultati sperimentali hanno mostrato le seguenti proprietà:

- Migliore accuratezza del singolo albero decisionale e del *Bagging*, ma peggiore della *Random forest*.
- Capacità di approssimare funzioni non parallele agli assi cartesiani (Liu et al., 2008).
- Sensibilità rispetto a variabili irrilevanti e perturbazione dei dati.

I parametri sono gli stessi della *Random forest*: numero di alberi $B$, minima ampiezza dei nodi terminali $n_{min}$ e numero di variabili campionate in ogni nodo $m$. Notare che per raggiungere la medesima ampiezza nei nodi terminali $n_{min}$, l'*Extra-trees* dovrà effettuare un numero maggiore di *split* rispetto al *Bagging* e alla *Random Forest* a causa della casualità nel processo di crescita. Geurts, Ernst e Wehenkel (2006) indicano i seguenti valori di *deafult*:

- Per la classificazione $m = \sqrt{p}$ e $n_{min} = 1$.
- Per la regressione $m = p$, mentre $n_{min} = 5$.
- L'errore converge con un numero di alberi $B > 50$.
- Con l'aumento del rumore nella variabile risposta il valore ottimale di $n_{min}$ cresce.
- Con un aumento di variabili incorrelate con la risposta il valore ottimale di $m$ aumenta di conseguenza.

Le ultime 2 considerazioni valgono anche per la *Random forest*, ma in questo caso, dato la casualità nell'algoritmo di crescita, l'errore è maggiormente sensibile alla scelta dei parametri.
Per quanto riguarda l'ultimo punto si pensi al caso estremo del *Complete-Random trees*: anche se sono presenti variabili irrilevanti nel *dataset*, queste potranno essere selezionate con la stessa probabilità delle altre. Dividere un nodo tramite una variabile non correlata con la risposta è equivalente a dividere casualmente il *dataset* di stima. Quindi variabili irrilevanti comportano

una riduzione delle dimensione del campione e un aumento della distorsione e della varianza dei singoli alberi. Quest'ultima sarà poi compensata dalla media finale, ma resterà invariata la distorsione.

## 1.4 Stacking

Finora tutti i metodi di *ensemble* si sono basati sulla media per combinare tra loro i modelli di base.
Tuttavia la procedura può essere generalizzata sostituendo alla media un qualsiasi modello di apprendimento. Si definisce quindi lo *Stacking* (Wolpert, 1992) dove i modelli di base sono chiamati *learners* di primo livello, mentre il modello di unione viene detto *learner* di secondo livello, o *meta-learner*. L'idea di base è stimare il primo livello utilizzando il *dataset* originale, per poi utilizzare l'*output* risultante come *input* del secondo livello, tenendo invariata la variabile risposta. Spesso è altamente consigliabile utilizzare un primo livello eterogeneo per incoraggiare la diversità.
Nella fase di stima, tuttavia, il rischio di *overfitting* è piuttosto alto se vengono utilizzate le previsioni grezze per il secondo livello. Una buona pratica è impiegare invece le previsioni ottenute tramite una *K-folds cross validation* (*K-folds* CV): ogni gruppo viene predetto da *K-1* gruppi uniti, con ogni gruppo che ricoprirà una volta il ruolo di insieme di verifica e *K-1* volte il ruolo di insieme di stima. Formalmente, sia:

- $N$: numero di osservazioni.
- $K$: numero di gruppi con cui partizionare il *dataset*.
- $k : \{1, ..., N\} \mapsto \{1, ..., K\}$ l'indice che specifica la partizione in cui ogni osservazione $i$ è stata allocata casualmente.
- $m$: numero di modelli di primo livello.
- $C_j$: $j$-esimo modello di primo livello.
- $\hat{C}_j^{-k}(x)$: corrispondente funzione stimata, calcolata avendo rimosso il $k$-esimo gruppo.

- $\hat{C}_j^{-k(i)}(\mathbf{x}_i)$: corrispondente previsione della $i$-esima osservazione.

Ogni riga del *dataset* che verrà utilizzato al secondo livello come *input* è pari a

$$\mathbf{x}_i^{(1)} = [\hat{C}_1^{-k(i)}(\mathbf{x}_i)\ \hat{C}_2^{-k(i)}(\mathbf{x}_i) \cdots \hat{C}_m^{-k(i)}(\mathbf{x}_i)]. \tag{1.22}$$

Dopo aver stimato il *meta-learner*, è consigliabile ristimare i modelli di primo livello sull'intero *dataset* di stima.
Per scegliere il valore migliore di $K$ bisogna riconnetersi all'equazione 1.7: se $K = N$ il metodo viene definito *leave-one-out cross validation* (LOOCV) ed è uno stimatore asintoticamente non distorte per il vero errore di previsione atteso, ma può avere alta varianza dato che gli $N - 1$ gruppi di stima sono simili tra di loro avendo escluso una singola osservazione. Al contrario, dato $K = 3$, la *cross validation* ha minore varianza, ma fornisce una stima distorta. Un buon compromesso tra varianza e distorsione è dato dalla 5- o 10-*fold* CV (Breiman e Spector, 1992).
Scegliendo come *meta-learner* $C^{(2)}$ un modello lineare la previsione finale sarà pari a:

$$\hat{C}^{(2)}(\mathbf{x}_i^{(1)}) = \sum_{j=1}^{m} \hat{\alpha}_j \hat{C}_j(\mathbf{x}_i) \tag{1.23}$$

dove i coefficienti $\alpha$ sono stimati minimizzando la somma dei quadrati residua

$$\hat{\alpha} = \underset{\alpha}{\operatorname{argmin}} \sum_{i=1}^{N} \left( y_i - \sum_{j=1}^{m} \alpha_j \hat{C}_j^{-k(i)}(\mathbf{x}_i) \right)^2. \tag{1.24}$$

Sotto il vincolo ulteriore che $\alpha_j \geq 0$ e $\sum_{j=1}^{m} \alpha_j = 1$ si dimostra (Breiman, 1996b) che l'errore di $\hat{C}^{(2)}(x)$ non può essere superiore al minor errore dei singoli *learners* di primo livello:

$$\sum_{i=1}^{N} \left( y_i - \sum_{j=1}^{m} \alpha_j \hat{C}_j^{-k(i)}(\mathbf{x}_i) \right)^2 \leq \sum_{i=1}^{N} \left( y_i - \hat{C}_j^{-k(i)}(\mathbf{x}_i) \right)^2. \tag{1.25}$$

Come nel paragrafo 1.1, si calcola quindi il valore atteso e la varianza del *meta-learner*. In questo caso si rilassa l'ipotesi di identica distribuzione e correlazione tra i *base learners*, definendo $\mu_j$ e $\sigma_j^2$ la media e varianza del

$j$-esimo *learners* di primo livello, mentre $\rho_{jh}$ la correlazione tra il *learners* $j$-esimo e il *learners* $h$-esimo, allora

$$Var\{\hat{C}^{(2)}(x^{(1)})\} = \sum_{j=1}^{m} \hat{\alpha}_j^2 \sigma_j^2 + 2 \sum \sum_{h<j} \hat{\alpha}_j \hat{\alpha}_h \rho_{jh} \sigma_j \sigma_h \tag{1.26}$$

$$\mathbb{E}\{\hat{C}^{(2)}(x^{(1)})\} = \sum_{j=1}^{m} \hat{\alpha}_j \mu_j. \tag{1.27}$$

Quindi la distorsione diminuisce dando maggiore peso ai migliori modelli. La Formula 1.26 al contrario non è di semplice interpretazione come la Formula 1.8, tuttavia si può notare che adesso si richiede che i *learners* di primo livello siano correlati negativamente tra di loro, ovvero che ognuno riesca a predire bene differenti porzioni di dati. In questo modo l'*ensemble* risultante avrà una varianza notevolmente diminuita.
Ting e Witten (1999) raccomandano di utilizzare, nei problemi di classificazione, le stime della probabilità di ogni classe come *input* del secondo livello in modo da tenere conto non solo delle previsioni finali, ma anche del grado di fiducia con cui vengono fatte.
Notare che la semplice media è un caso particolare dello *Stacking*, di cui però non vi è dimostrazione che il modello *ensemble* risultante sarà migliore dei singoli modelli di base. Questo però non significa che il modello lineare è definitivamente migliore rispetto alla semplice media in quanto i risultati sperimentali sono piuttosto contrastanti (Xu, Krzyzak e Suen, 1992). Una possibile ragione è che i dati a disposizione sono usualmente insufficienti e perturbati, così che i coefficienti stimati dalla 1.24 sono inaffidabili. Inoltre lo *Stacking*, anche utilizzando per la stima le previsioni *K-folds* CV, soffre dell'*overfitting* con un largo numero di modelli di base, al contrario della semplice media, come si può intuire dall'equazione 1.23. L'opinione accettata è che si debba utilizzare la media semplice per combinare modelli con simili *performance*, in caso contrario lo *Stacking* può produrre un migliore *ensemble*.

# Capitolo 2

# Un nuovo modello

Dopo avere presentato i principali metodi di *ensamble learning* e alcune loro proprietà, in questo capitolo verrà descritto in dettaglio il *Deep random forest*, termine utilizzato per indicare 2 modelli *ensemble* associati: il *Cascade Forest* e il *Multi-Grained Cascade Forest*. Tuttavia, data la recente comparsa, il materiale empirico che ne attesti l'efficacia è ancora esiguo. Verranno pertanto eseguiti 3 studi di simulazione per descriverne la sua capacità di adattamento e previsione.

## 2.1 Cascade Forest

Si procede quindi a una descrizione formale del *Cascade Forest*, la notazione utilizzata è basata sulla pubblicazione di Miller et al. (2017). Si definisce quindi:

- $\hat{y}_i$: valore dell'*output* per la $i$-esima osservazione.
- $\mathbf{x}_i^{(0)} = [x_1 \cdots x_p]$: valore degli *input* per la $i$-esima osservazione.
- $L$: numero di livelli del *Cascade forest*, senza comprendere lo strato di *input* (che si denota come livello 0) e lo strato di *output*.
- $m_l$: numero di modelli di apprendimento dell'$l$-esimo livello, $l = 1, ..., L$.
- $C_j^{(l)}$: $j$-esimo modello di apprendimento dello strato $l$-esimo, $j = 1, ..., m_l$.

- $\hat{C}_j^{(l)}(\mathbf{x}_i)$: previsione associata all'$i$-esima osservazione dato il modello $C_j^{(l)}$ (singolo valore nel caso della regressione o vettore delle probabilità $1 \times K$ nel caso della classificazione a K categorie).

Al primo strato si ottiene quindi una vettore $1 \times (Km_1)$, denominato $\mathbf{x}_i^{(1)}$:

$$\mathbf{x}_i^{(1)} = [\hat{C}_1^{(1)}(\mathbf{x}_i^{(0)})\ \hat{C}_2^{(1)}(\mathbf{x}_i^{(0)}) \cdots \hat{C}_{m_1}^{(1)}(\mathbf{x}_i^{(0)})] \tag{2.1}$$

Il vettore $\mathbf{x}_i^{(1)}$ viene concatenato al vettore $\mathbf{x}_i^{(0)}$, formando:

$$\mathbf{X}_i^{(1)} = [\mathbf{x}_i^{(0)}\ \mathbf{x}_i^{(1)}] \tag{2.2}$$

un vettore $1 \times (p + Km_1)$ che è l'*imput* per lo strato successivo. In generale, la relazione tra lo stato $l$-esimo e lo strato *(l-1)*-esimo è definita come:

$$\mathbf{x}_i^{(l)} = [\hat{C}_1^{(l)}(\mathbf{X}_i^{(l-1)})\ \hat{C}_2^{(l)}(\mathbf{X}_i^{(l-1)}) \cdots \hat{C}_{m_{l-1}}^{(l)}(\mathbf{X}_i^{(l-1)})] \tag{2.3}$$

con

$$\mathbf{X}_i^{(l-1)} = [\mathbf{x}_i^{(0)}\ \mathbf{x}_i^{(1)} \cdots \mathbf{x}_i^{(l-1)}] \tag{2.4}$$

All'ultimo strato $L$ si ha quindi il vettore $\mathbf{x}_i^{(L)}$.
Nel caso della regressione la previsione finale viene ottenuta mediante la media:

$$\hat{y}_i = \frac{1}{m_L}\mathbf{x}_i^{(L)}\mathbf{1} \tag{2.5}$$

con $\mathbf{1}$ vettore $m_L \times 1$.
Di conseguenza, la completa relazione che lega il vettore di *input* $\mathbf{x}_i^{(0)}$ con quello dell'*output* $\hat{y}_i$ risulta essere:

$$\hat{y}_i = \frac{1}{m_L}[\hat{C}_1^{(l)}([\mathbf{x}_i^{(0)} \cdots [\hat{C}_1^{(l-1)}(\cdots[\hat{C}_1^{(1)}(\mathbf{x}_i^{(0)}) \cdots \hat{C}_m^{(1)}(\mathbf{x}_i^{(0)})])\cdots]]) \cdots \hat{C}_m^{(l)}(\mathbf{X}_i^{(l-1)})]\mathbf{1} \tag{2.6}$$

Nel caso della classificazione, $\mathbf{x}_i^{(L)}$ contiene $L$ probabilità previste per ogni categoria, quindi si devono fare $K$ medie:

$$\hat{\mathbf{y}}_i = \frac{1}{m_L} vec_{K,m_L}^{-1}(\mathbf{x}_i^{(L)^T})^T\mathbf{1} \tag{2.7}$$

Dove $\hat{\mathbf{y}}_i$ è un vettore $K \times 1$ contenente le probabilità previste delle $K$ classi e $vec_{K,m_L}^{-1}$ l'operazione inversa alla vettorizzazione; la classe predetta è quella

a cui corrisponde probabilità più alta. La procedura descritta viene rappresentata in Figura 2.1.
Si noti come la struttura del *Cascade forest* sia molto simile a quella del *feed-forward neural networks* (si veda Aere, 2017, pag. 20), dove ogni livello produce un *output* che viene utilizzato dal livello successivo e così via fino alla previsione finale.
Ogni livello è un *ensemble* di foreste, ovvero un *ensemble* di *ensembles*. Per incoraggiare la diversità ogni livello è formato da 2 tipi di foreste: 2 *Random Forest* (blu) e 2 *Complete-random-trees* (nero).

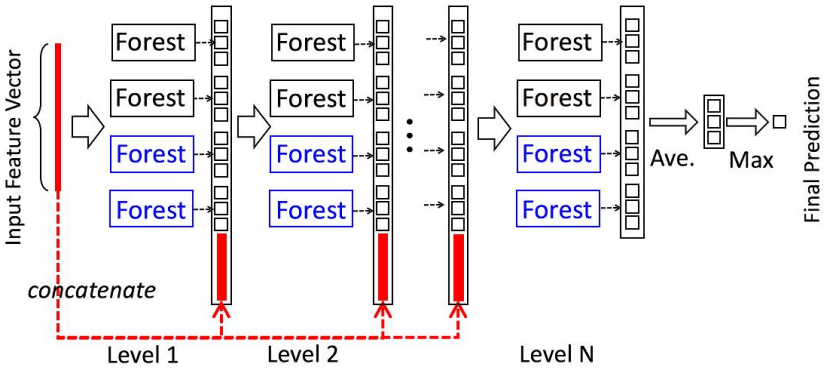


**Figura 2.1:** Illustrazione della struttura del *cascade forest*. Ogni livello del *Cascade forest*(blu) è costituito da 2 *Random Forest* e 2 *Complete-random trees*(nero). In questo caso si tratta di un problema di classificazione con 3 classi.

Per evitare il rischio di *overfitting* e in analogia allo *Stacking*, in fase di stima le previsioni che vengono attaccate allo strato successivo sono generate da una *K-folds cross validation* (*K-folds* CV). Dopo aver creato un nuovo livello, viene calcolato l'errore di verifica (*test error*) e la procedura terminerà se non ci sarà una diminuzione significativa rispetto all'errore associato al precedente livello: in questo modo il numero di livelli viene automaticamente determinato.
Per scegliere la composizione di ogni livello bisogna tenere in considerazione 2 aspetti presentati nel capitolo precedente: correlazione e distorsione. Ad esempio, se il *Cascade Forest* è formato da un singolo livello, la previsione finale non sarà nient'altro che la media delle previsioni associate ai singoli

modelli di base. La media non agisce sulla distorsione, ma solo sulla varianza in maniera inversamente proporzionale alla correlazione tra i modelli.
Per mediare questi 2 fattori Zhou e Feng (2017) scelgono di inserire la *Random Forest* e il *Complete-random trees* in egual proporzione all'interno di ogni livello: il primo ha bassa distorsione e diversità, il secondo, per la sua natura casuale, ha alta diversità e distorsione. Allo stesso modo il *Cascade Forest* con 2 livelli e con $m_2 = 1$ può essere paragonata a uno *Stacking* che utilizza sia le previsioni del precedente livello che le variabili esplicative iniziali.

## 2.2 Multi-Grained Scanning

Certi tipi di reti neurali sono particolarmente adatte per gestire particolari relazioni, ad esempio *convolutional neural networks* sono efficaci su insiemi di immagini dove le relazioni spaziali tra i *pixels* sono cruciali (LeCun et al., 1998 e Krizhevsky, Sutskever e Hinton, 2012), *recurrent neural networks* sono specializzate su dati sequenziali dove bisogna tenere conto dell'ordine di esecuzione (Graves, Mohamed e Hinton, 2013 e Cho et al., 2014). Prendendo spunto da questi metodi, Zhou e Feng (2017) hanno voluto sfruttare lo stesso concetto per potenziare il *cascade forest* mediante una procedura di *Multi-Grained Scanning*.
Come illustrato dalla Figura 2.2, una finestra mobile è utilizzata per mappare le variabili iniziali. Si supponga di avere 400 variabili esplicative e di utilizzare una finestra di ampiezza 100. Per dati sequenziali (pannello superiore) è generato un vettore 100-dimensionale ogni volta che la finestra procede di un'unità; in totale sono prodotti 301 vettori. Se invece le variabili grezze hanno una relazione spaziale (pannello inferiore), ad esempio una immagine $20 \times 20$ per un totale di 400 *pixel*, allora una finestra $10 \times 10$ produrrà 121 sottoimmagini. Quest'ultime saranno poi utilizzate come *input* di una *Random Forest* (*Forest B*) e un *Complete-random trees* (*Forest A*). Supponendo che sia in un problema di classificazione a 3 classi, ogni foresta produce un vettore 3-dimensionale di probabilità stimate, per un totale di 121 vettori. Infine i vettori vengono concatenati insieme, formando 1806 variabili derivate

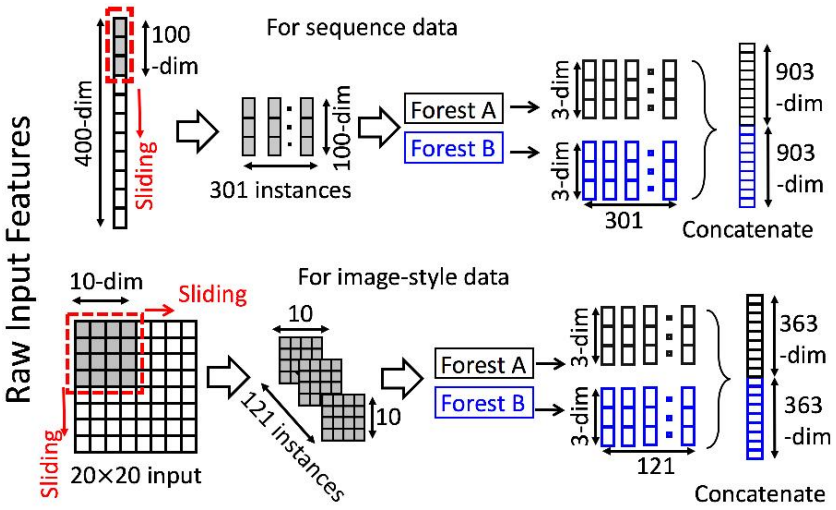


**Figura 2.2:** Illustrazione di variabili pre-processate utilizzando una finestra mobile. Si è supposto un problema di classificazione a 3 classi, numero di variabili esplicative pari a 400 e una finestra mobile di dimensione 100.

dalle iniziali 400.

La Figura 2.3 descrive come il *Multi-Grained Scanning* venga unito al *Cascade Forest* per formare il *gcForest* (*Multi-Grained Cascade Forest*). Si supponga di essere ancora in un problema di classificazione a 3 classi, che i dati siano sequenziali e composti da 400 caratteristiche originali, e siano utilizzate 3 finestre di ampiezza diversa. Per ogni osservazione, finestre con ampiezza 100, 200 e 300 genereranno rispettivamente 1806, 1206 e 606 vettori. I 3 gruppi verranno inseriti rispettivamente nel 1°, 2° e 3° livello del *Cascade Forest*, denominati gradi. La procedura viene ripetuta fino alla convergenza dell'errore di verifica; la previsione finale sarà ottenuta come la media degli ultimi 4 vettori 3-dimensionali di probabilità.

Riassumendo il *gcForest* può essere visto come un insieme di $n$ *Cascade Forest* e $n$ *Multi-Grained scanning*, dove $n$ corrisponde al numero totale di finestre utilizzate.

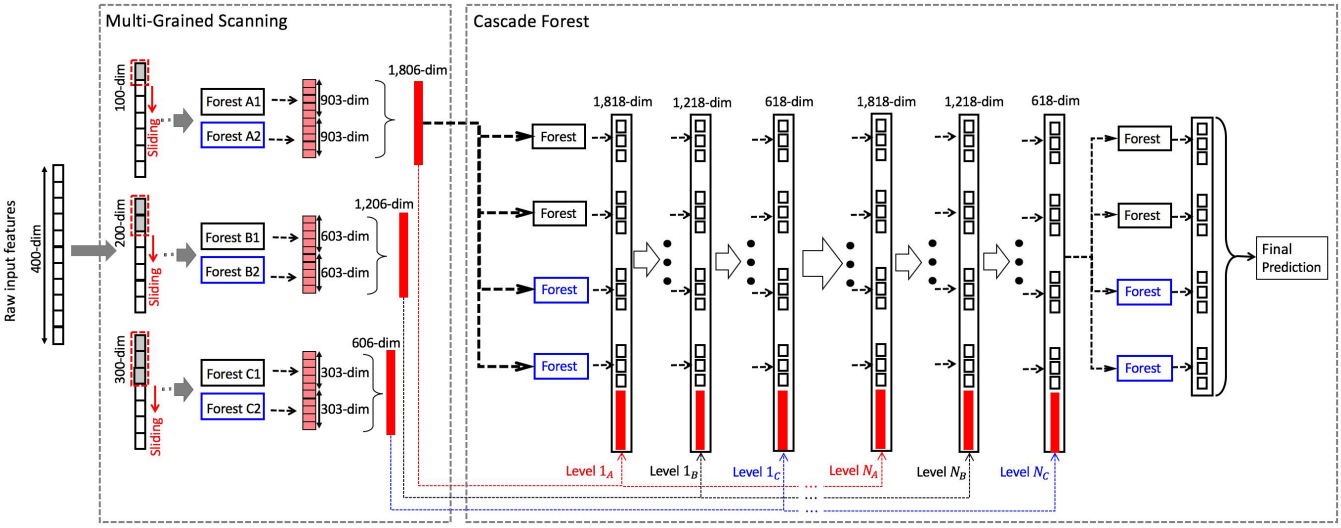


**Figura 2.3:** La procedura complessiva del *gcForest*. Si suppone che ci sono 3 classi da predirre, dati sequenziali composti da 400 variabili e 3 finestre di diversa ampiezza utilizzate.

## 2.3 Studio di simulazione

Si supponga che i dati vengano generati dalla funzione $Y = f(X) + \epsilon$. Si denoti $X \in \mathbb{R}^p$ la matrice casuale dei dati di *input* e $Y \in \mathbb{R}$ il vettore casuale dei dati di *output*. Si cerca quindi uno stimatore $\hat{f}(X)$ per predire $Y$ dati i valori di $X$. A tal scopo si definisce la funzione di perdita $L(Y, \hat{f}(X)) = (Y - \hat{f}(X))^2$ per penalizzare gli errori di previsione, detta errore quadratico. Questo porta ad un criterio oggettivo per scegliere $\hat{f}$:

$$MSE(\hat{f}) = \mathbb{E}\{[Y - \hat{f}(X)]^2\} \tag{2.8}$$

chiamato errore quadratico medio. Condizionandosi a $X = x$ e minimizzando la funzione 2.8 rispetto a $\hat{f}(x)$, si ottiene la soluzione [1]

$$\hat{f}(x) = \mathbb{E}\{Y|X = x\} \tag{2.9}$$

ovvero il valore atteso condizionato, anche conosciuto come funzione di regressione. Quindi la migliore previsione di $Y$ dato i punti $X{=}x$ è il valore

[1] Per la dimostrazione completa si veda Hastie, Tibshirani e Friedman (2009), pag. 18 e 21.

atteso condizionato. Sostituendo la 2.9 al vero modello generatore dei dati si ottiene:

$$\begin{aligned} MSE(f) &= \mathbb{E}\{[Y - \mathbb{E}\{Y|X = x\}]^2\} \\ &= Var\{Y|X = x\} \\ &= Var\{\epsilon\}. \end{aligned} \tag{2.10}$$

Quindi com'era logico aspettarsi, anche conoscendo la vera funzione generatrice dei dati l'errore quadratico medio non potrà mai essere uguale a 0.
In realtà esistono diverse funzioni di perdita, e.g $L(Y, \hat{f}(X)) = |Y - \hat{f}(X)|$, in questo caso la soluzione al problema di minimizzazione è la mediana condizionata: $\hat{f}(x) = median(Y|X = x)$.
Quando l'*output* è una variabile categoriale $G$ (problema di classificazione) si deve sostituire la funzione di perdita 2.8. Data una stima $\hat{G}$, essa assumerà valori in $\zeta$, l'insieme delle possibili classi. La funzione di perdita sarà rappresentata da una matrice $\mathbf{L}_{K\times K}$, dove $K = card(\zeta)$. $\mathbf{L}$ avrà valori nulli sulla diagonale e non negativi altrove, dove $L(k, l)$ è il prezzo pagato nel classificare un'osservazione appartenete alla classe $\zeta_k$ come $\zeta_l$. In questo contesto verrà utilizzata la funzione di perdita *zero-uno*, ovvero ogni classificazione errata verrà pagata una singola unità.
L'errore previsto atteso (*expected prediction error*) è

$$\begin{aligned} EPE &= \mathbb{E}\{L(G, \hat{G}(X))\} \\ &= \mathbb{E}\left\{\sum_{k=1}^{K} L(\zeta_k, \hat{G}(X))\right\} \end{aligned} \tag{2.11}$$

detto errore medio di errata classificazione. Come nel caso precedente, condizionandosi a $X = x$ e minimizzando la funzione 2.11 rispetto a $\hat{G}(x)$, si ottiene la soluzione

$$\hat{G}(x) = \zeta_k \text{ se } \mathbb{P}\{\zeta_k|X = x\} = \max_{g\in\zeta} \mathbb{P}\{g|X = x\}. \tag{2.12}$$

La soluzione è conosciuta come il classificatore di *Bayes* e dice che si sceglierà la classe più probabile, usando la distribuzione condizionata $\mathbb{P}\{G|X = x\}$.
La Figura 2.4 mostra la frontiera di *Bayes* ottimale per i dati oggetto di studio del Paragrafo 2.3.3.

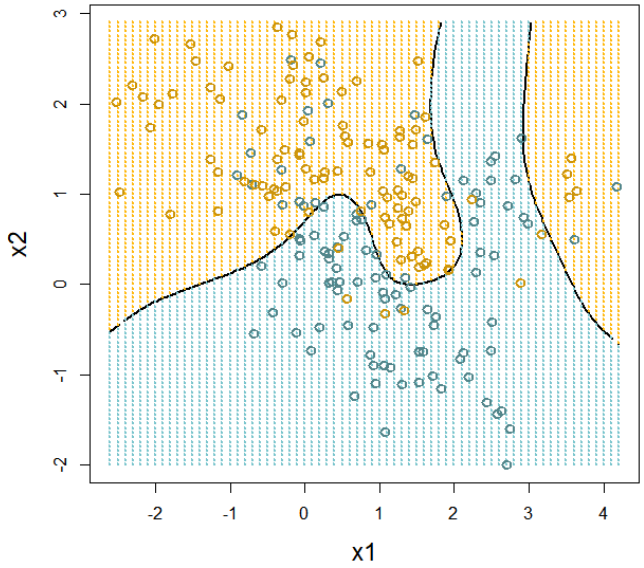


**Figura 2.4:** La frontiera di *Bayes* ottimale per i dati nel 3 studio di simulazione.

Il tasso di errore del classificatore di *Bayes* (2.12) e $MSE(f)$ (2.10) vengono detti errori di *Bayes*.
Per quantificare quanto il *Cascade Forest* si avvicini ai rispettivi errori di *Bayes*, come già anticipato, verranno effettuati 3 studi di simulazione: 2 di regressione e 1 di classificazione. Lo scopo è di confrontare la capacità di adattamento, cambiando il valore dei parametri.
Il *Cascade Forest* ha 3 parametri di regolazione fondamentali:

- Quale tipo di previsione attaccare ad ogni livello successivo.
- Composizione di ogni livello.
- Numero di livelli.

Secondo Zhou e Feng (2017) le previsioni da attaccare ad ogni strato successivo dovrebbero essere generate da una 3-*folds* CV, mentre ogni livello dovrebbe contenere 8 foreste (4 *Random forest* e 4 *Complete-Random trees*) e il numero di livelli dovrebbe essere auto-determinato dall'errore di verifica.

Quindi le previsioni 3-*folds* CV saranno confrontate con le previsioni OOB (Paragrafo 1.1.1) e le previsioni 5-*folds* CV (Paragrafo 1.4), mentre si utilizzerà gli errori associati a questi 3 metodi per capire se riescono a determinare il numero corretto di livelli senza dover utilizzare un insieme di verifica.
Inizialmente si simula 100 volte da una distribuzione predeterminata i valori di $X$ e si calcola i corrispondenti valori di $f(x)$; per ogni campione vengono stimate 3 *Cascade Forest* composte ciascuna da 10 livelli, con 4 *Random Forest* e 4 *Complete-random trees* per livello e 200 alberi per ogni foresta. Vengono infine utilizzate rispettivamente le previsioni OOB[2], 3- e 5-*folds* CV da attaccare ad ogni strato successivo.
Successivamente alla scelta dei migliori parametri la simulazione sarà ripetuta cambiando il numero totale di foreste ad ogni livello (1, 2, 4, 8, 12 e 16), lasciando la proporzione (*n. Random Forest*)/ (*n. Totale di foreste*) costante al 50%. Nei primi due studi [3] i dati verranno generati da un modello additivo $Y = f(X) + \epsilon$, dove $X = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_p]$ e ogni $\mathbf{x}_j$ viene generato da una normale standard.
L'errore $\epsilon$ ha distribuzione Gaussiana, a media nulla e varianza scelta in modo che il rapporto segnale-rumore (*signal-to-noise ratio*)

$$\frac{Var\{f(X)\}}{Var\{\epsilon\}} = 4. \tag{2.13}$$

Il campione di stima avrà 200 osservazioni, quello di verifica $10^4$.

### 2.3.1 Prima simulazione

Nel primo caso di studio la funzione (Figura 2.5) da cui verrà generata la variabile risposta sarà pari a

$$Y = \sigma(Xa_1) + \sigma(Xa_2) + \epsilon_1 \tag{2.14}$$

dove $\sigma(v) = 1/(1 + e^{-v})$, $a_1^T = (3, 3)$ e $a_2^T = (3, -3)$.

[2]Nel *Complete-Random trees*, dovendo utilizzare l'intero *dataset*, le previsioni saranno calcolate con una *3-folds* CV

[3]Vengono ripresi gli stessi studi di simulazione utilizzati per le reti neurali in Hastie, Tibshirani e Friedman (2009), pag. 399, 401-404.

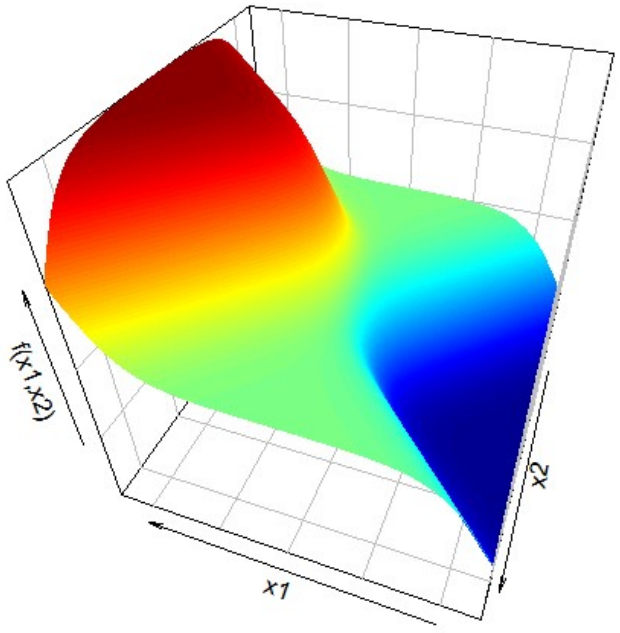


**Figura 2.5:** Grafico della funzione del primo studio di simulazione

I risultati sono rappresentati in Figura 2.6. Nel pannello di destra si vede che il primo livello raggiunge la miglior *performance*. Con un numero maggiore sopraggiunge velocemente l'*overfitting* e l'errore comincia ad aumentare per poi stabilizzarsi dal 5° livello su un valore costante (una sorta di *plateau*). I valori sono riscalati per l'errore di *Bayes* (0.08), ottenuto tramite la Formula 2.13 e ripetendo la simulazione $10^4$ volte.
Le zone ombreggiate rappresentano gli intervalli alla *Wald* al 95%, in questo caso non vi è una differenza significativa tra i 3 modelli. Nel pannello di sinistra è presente l'andamento degli errori OOB, 3- e 5-*folds* CV (rispettivamente oob, cv3 e cv5). Solamente quest'ultimi 2 individuano il corretto numero di livelli; il primo, benché rimanga costante fino al secondo livello, ha un brusco calo al livello 3 per poi rimanere costante.
Infine la Figura 2.7 mostra i *boxplot* dell'errore di verifica per 6 differenti numeri di foreste utilizzando un singolo livello. Notare che quando il numero di foreste è pari a 1 si ricade nella *Random Forest*. Dato il basso numero di dimensioni e la funzione generatrice dei dati non particolarmente comples-

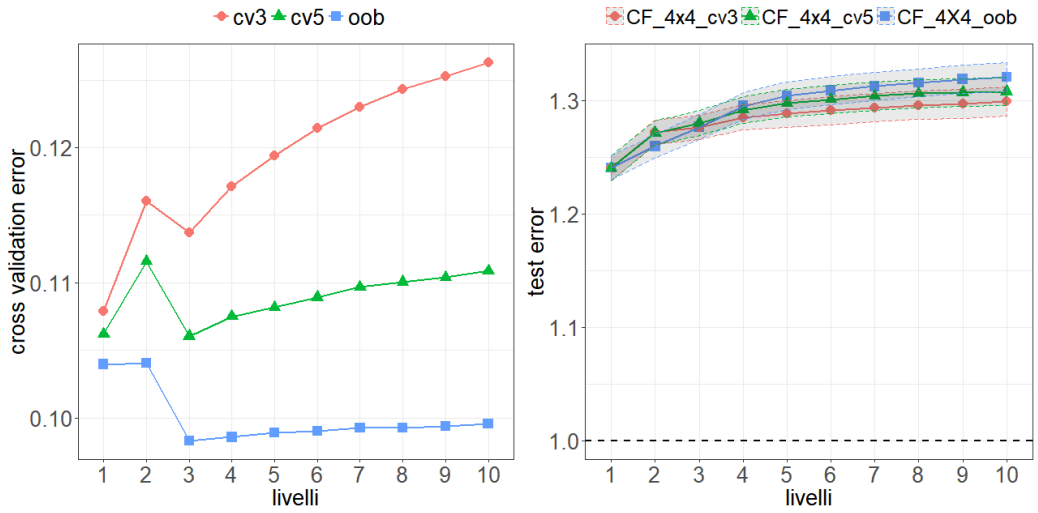


**Figura 2.6:** Primo studio di simulazione. Andamento degli errori oob, 3- e 5-*folds cross-validation* (cv3 e cv5) (pannello di sinistra) e dei corrispondenti errori di verifica, divisi per l'errore di *Bayes* (linea orizzontale tratteggiata) pari a 0.08. Le aree ombreggiate rappresentano i relativi intervalli di confidenza al 95%.

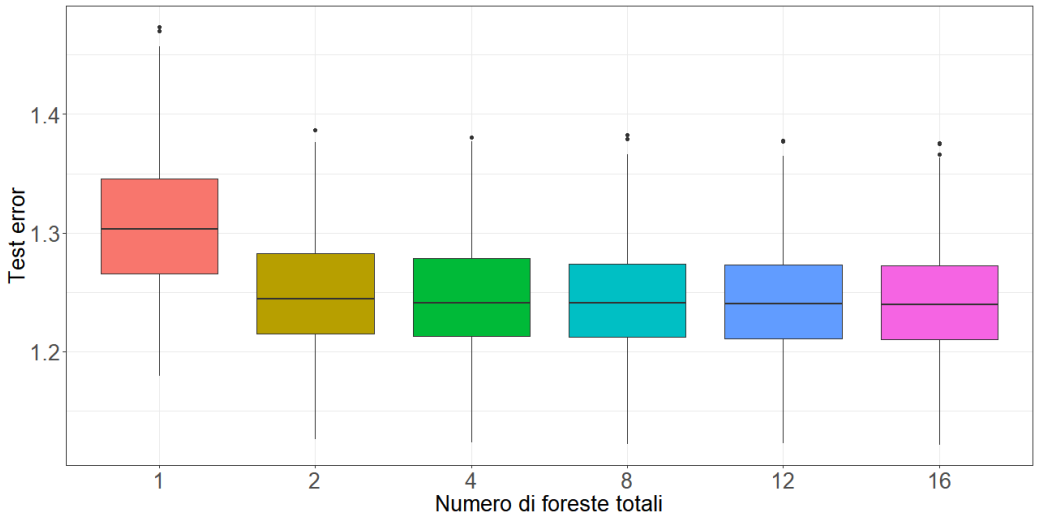


**Figura 2.7:** *Boxplot* degli errori di verifica divisi per l'errore di *Bayes*. L'errore è mostrato per 6 differenti numeri di foreste utilizzando un singolo livello.

sa, è sufficiente una *Random Forest* e un *Complete-random trees* (numero di foreste uguale a 2) per raggiungere l'ottimalità.

### 2.3.2 Seconda simulazione

Nel secondo caso di studio si hanno 10 variabili esplicative generate da una normale multivariata a componenti indipendenti. La funzione 2.8 da cui è stata generata la variabile risposta è:

$$Y = \prod_{m=1}^{10} \phi(X_m) + \epsilon_2 \tag{2.15}$$

con

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}.$$

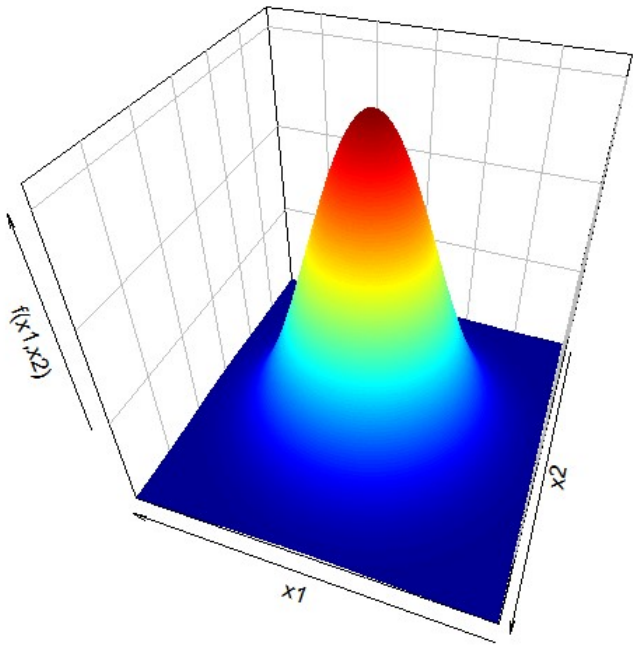


**Figura 2.8:** Grafico della funzione marginale bivariata del secondo studio di simulazione.

I risultati ottenuti sono illustrati nella Figure 2.9 in cui si è utilizzata la stessa

notazione precedente. In questo caso sono necessari 2 strati per raggiungere il risultato migliore. La funzione $f(X)$ è molto più difficile da stimare, sia per il numero maggiore di variabili, sia per la sua forma campanulare difficilmente approssimabile da una funzione costante a tratti. Nel pannello di destra si vede che l'errore di verifica sta ben al di sopra dell'errore di *Bayes* (notare la differente scala sull'asse delle ordinate). La zona ombreggiata ha il medesimo significato della Figura 2.6, ma in questo caso, al secondo livello, l'errore associato al *Cascade Forest* ottenuto tramite le previsioni OOB (CF_4×4_oob) è significativamente più basso dell'errore del *Cascade Forest* ottenuto tramite una 3-*folds* CV (CF_4×4_cv3).
Nel pannello di sinistra è presente l'andamento degli errori OOB, 3- e 5-*folds* CV. Rispetto al caso precedente l'errore OOB e l'errore cv3 raggiungono il minimo a 2, mentre l'errore cv5 richiede un livello in più.

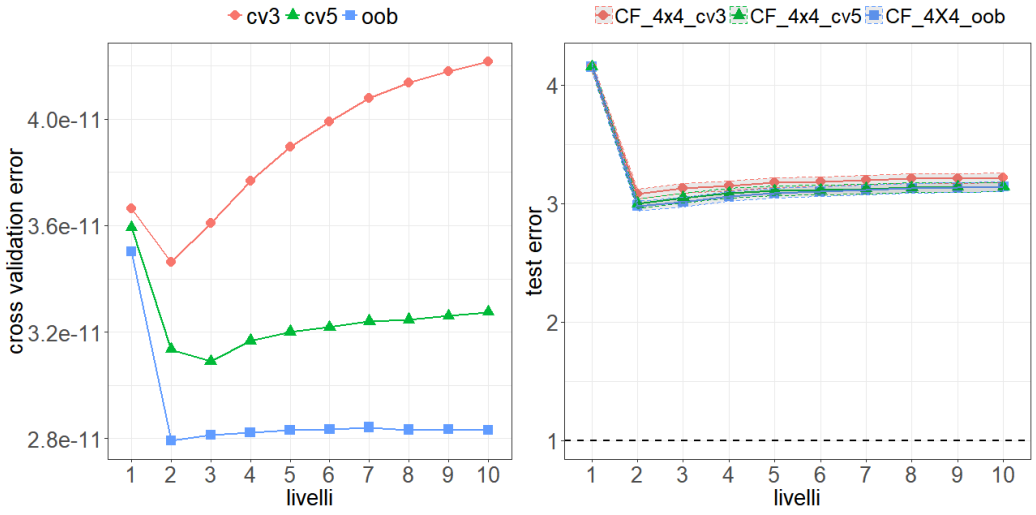


**Figura 2.9:** Secondo studio di simulazione. Andamento degli errori OOB, 3- e 5-*folds cross-validation* (pannello di sinistra) e dei corrispondenti errori di verifica, divisi per l'errore di *Bayes* (linea orizzontale tratteggiata) pari a $8.19 \cdot 10^{-12}$. Le aree ombreggiate rappresentano i relativi intervalli di confidenza al 95%.

Infine la Figura 2.10 mostra i *boxplot* dell'errore di verifica per 6 differenti numeri di foreste utilizzando 2 livelli. In realtà con un numero di foreste pari a 1 ci si riferisce ancora alla *Random Forest* per poter quantificare il

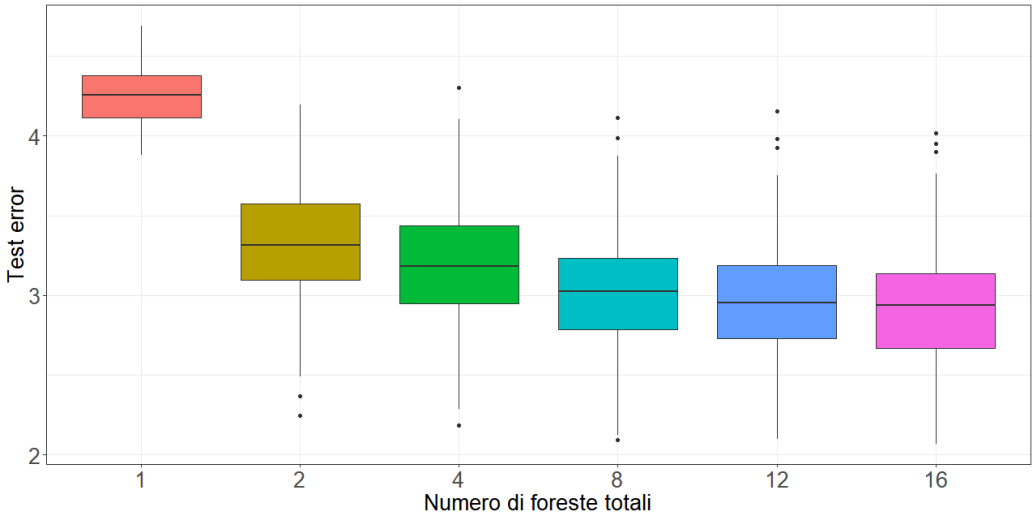


**Figura 2.10:** *Boxplot* degli errori di verifica divisi per l'errore di *Bayes*. L'errore è mostrato per 6 differenti numeri di foreste tenendo costante il numero di livelli pari a 2.

miglioramento. Rispetto al caso precedente servono almeno 8 foreste affinché la mediana si stabilizzi, con un lieve miglioramento tra 8 e 12 foreste.

### 2.3.3 Terza simulazione

In quest'ultimo caso ci si sposta su un problema di classificazione, in cui ogni classe viene da una mistura di 10 distribuzioni Gaussiane ognuna con varianza di 0.2 e con medie individuali distribuite anch'esse come una normale standard (Figura 2.4 mostra 200 osservazioni simulate), in formula:

$$f(\mathbf{x}|\boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_{10}) = \sum_{i=1}^{10} \frac{1}{10} \phi_2 \left( \frac{1}{\sqrt{0.2}} (\mathbf{x} - \boldsymbol{\mu}_i) \right) \tag{2.16}$$

con

$$\begin{aligned} \mathbf{x}^T &= [x_1, x_2] \\ \boldsymbol{\mu}_i | Y = 0 &\sim \mathcal{N}_2(\boldsymbol{\gamma}_1, \boldsymbol{\Sigma}) \\ \boldsymbol{\mu}_i | Y = 1 &\sim \mathcal{N}_2(\boldsymbol{\gamma}_2, \boldsymbol{\Sigma}) \\ \boldsymbol{\gamma}_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \boldsymbol{\gamma}_2 &= \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \boldsymbol{\Sigma} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \end{aligned}$$

I risultati sono mostrati in Figura 2.11. Benché il problema sia completamente diverso in confronto alla prima simulazione, si nota una certa somiglianza rispetto alla Figura 2.6: il minor errore di verifica viene raggiunto con il singolo livello, ma l'andamento dell'errore OOB (pannello di sinistra) continua a diminuire fino al terzo livello.
In questo caso la differenza tra il modello CF_4×4_cv3 e il modello CF_4×4_oob risulta significativa dal livello successivo al terzo: il primo si stabilizza al secondo livello, mentre il secondo continua a salire fino al settimo.

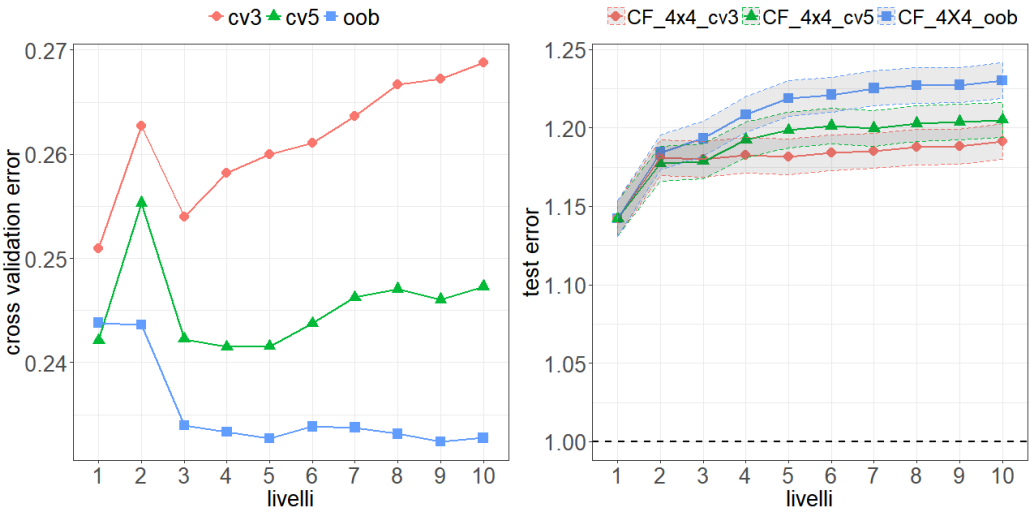


**Figura 2.11:** Terzo studio di simulazione. Andamento degli errori OOB, 3- e 5-*folds cross-validation* (pannello di sinistra) e dei corrispondenti errori di verifica, divisi per l'errore di *Bayes* (linea orizzontale tratteggiata) pari a 0.21. Le aree ombreggiate rappresentano i relativi intervalli di confidenza al 95%.

Successivamente si è proceduto come nei precedenti 2 studi a variare il numero totale di foreste. Anche in questo caso la Figura 2.12) è molto simile a quello della Figura 2.7, con l'ottimalità raggiunta immediatamente con un numero di foreste pari a 2 e un evidente miglioramento rispetto alla *Random Forest* (numero di foreste pari a 1).
Avendo solamente 2 variabili esplicative è stato inoltre possibile disegnare la frontiera decisionale del *Cascade forest* sui dati già presentati nella Figura 2.4, composto da un livello e un numero di foreste pari a 8, e confrontarla

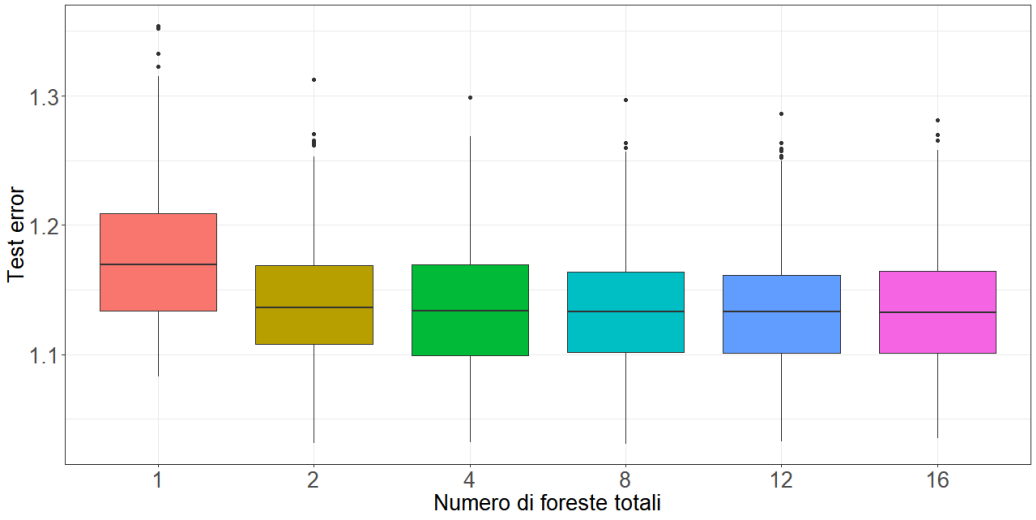


**Figura 2.12:** *Boxplot* degli errori di verifica divisi per l'errore di *Bayes*. L'errore è mostrato per 6 differenti numeri di foreste tenendo costante il numero di livelli pari a 1.

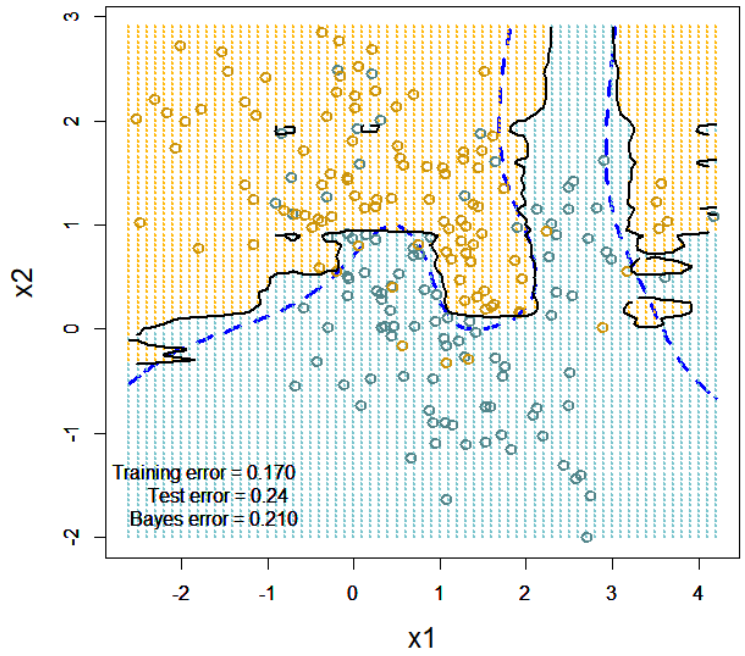


**Figura 2.13:** Frontiera decisionale del *Cascade forest* con un livello e 8 foreste. La linea tratteggiata blu rappresenta la frontiera di *Bayes*.

con la corrispondente frontiera di *Bayes* (linea tratteggiata blu nella Figura 2.13): le 2 curve seguono un andamento speculare, anche se la prima ha un andatura piuttosto frastagliata.

## 2.4 Riassunto

Nella seconda parte del capitolo si è cercato di definire i parametri di *default* del *Cascade Forest* mediante 3 studi di simulazione. Si è provato quindi a confrontare il modello utilizzando le previsioni 3-*folds* CV (CF_4×4_cv3) con il modello associato alle previsioni OOB (CF_4×4_oob) e alle previsioni 5-*folds* CV (CF_4×4_cv5). Il primo confronto presenta risultati discordanti: nel primo studio i 2 modelli non mostrano una differenza significativa, nel secondo il modello CF_4×4_oob è quello che possiede l'errore minore, situazione invertita nel terzo studio per un numero di livelli superiori a 3. Al contrario il CF_4×4_cv5 in tutti e 3 gli studi si posiziona in mezzo tra CF_4×4_oob e CF_4×4_cv3, anche se in tutti i casi le differenze con il migliore non sono significative.
Quindi la migliore scelta è utilizzare le previsioni fornite da una 5-*folds* CV da attaccare a ogni strato successivo, in analogia a quanto detto per lo *Stacking* (Paragrafo 1.4). Se non si hanno sufficienti risorse computazionali si può utilizzare il metodo stima-verifica con le previsioni OOB per capire il numero di livelli ottimale.
Il numero di foreste per strato sembra non portare a problemi di sovradattamento, ma solamente ad un aumento dei costi computazionali, in tutti e 3 gli studi un numero maggiore di 8 non porta un miglioramento evidente.
Infine dai 3 studi si può vedere che il *Cascade Forest* tende all'*overfitting* all'aumentare dei livelli: in tutti i casi CF_4×4_oob è quello che varia maggiormente, con un peggioramento del 6%, 5% e 8% rispettivamente nel primo, secondo e terzo studio.

# Capitolo 3

# Risultati sperimentali

Nei capitoli precedenti è stato descritto il nuovo metodo dal punto di vista teorico e i vari metodi a esso associato. In questo capitolo verranno confrontati per l'analisi di *dataset* nel campo della regressione, classificazione binaria e multiclassificazione.
Per i primi 2 casi di studio è stato possibile scaricare i *dataset* da `bee-viva.com`, il sito di competizioni di statistica dell'Università degli Studi di Padova: i dati erano già divisi nella parte di stima e in quella di verifica, ma la variabile risposta della seconda parte era oscurata. È stato comunque possibile calcolare gli errori di verifica caricando le previsioni in formato *.txt* nella pagina dedicata.
Il 3° *dataset* è disponibile all'indirizzo `yann.lecun.com/exdb/mnist` dove sono inoltre riportati tutti i classificatori sperimentati, nonché gli errori di verifica e gli articoli corrispondenti.
Per analizzare i primi 2 *dataset* è stato utilizzato esclusivamente il linguaggio di programmazione **`R`**, mentre nell'ultimo caso di studio è stato utilizzato il linguaggio **`Python`** per stimare il *Multi-Grained Scanning* in tempi ragionevoli. I pacchetti sono disponibili ai siti `github.com/ablanda/deepForest` e `github.com/kingfengji/gcForest`, per maggiori dettagli si rimanda al Paragrafo 3.4 e all'Appendice A.

## 3.1 Regressione

Il *dataset* analizzato proviene da un noto gruppo automobilistico ed è stato utilizzato nel corso dell'evento *Business game 2017* (`www.unipd.it/ilbo/business-game-selezione-data-scientist`).
I dati si riferiscono ad interventi di assistenza da parte di un noto gruppo automobilistico italiano. La *Business Unit After Sales* gestisce queste richieste attraverso un sistema di *ticketing* identificati da un Id (*Id_ ticket*), in particolare un intervento coinvolge diversi utenti con ruoli differenti e può seguire diverse strade, a ciò vengono aggiunte un certo numero di variabili di contorno che descrivono meglio il processo. Le prime corrispondono a:

- *Variants*, le attività coinvolte all'interno del processo, codificate come 15 variabili binarie denominate *Apertura*, *Assegnato*, *Attesa ricambi*, *Attesi di conferma*, *Attivazione specialista*, *Caso riaperto*, *Caso singolo* , *Cet*, *Confirmed*, *Escalation*, *Feedback_ negativo*, *Secondo livello di uscita*, *Prima_ attivazione_ secondo_ livello*, *Riapertura*, *Soluzione non efficace*.

- *Avaruser*, gli identificativi concatenati di tutti gli addetti coinvolti nel processo (2481 modalità).

- *Avarrole*, i ruoli concatenati che i corrispondenti addetti occupano (2 modalità).

Le seconde corrispondono a:

- *Gravità*, la gravità dell'anomalia (4 modalità).

- *Dea*, il committente che ha aperto la pratica di *ticket* (9029 modalità).

- *Mcall*, il motivo dell'intervento di assistenza (16 modalità).

- *Cod_ pr*, tipo di committente da cui parte l'intervento di assistenza (11 modalità).

- *Mercato*, area in cui viene risolto l'intervento (20 modalità).

- *Marca*, marca del veicolo (8 modalità).

- *Tkt_ type*, tipologia dell'intervento di assistenza (4 modalità).
- *Dinterv*, difficoltà dell'intervento (3 modalità).
- *Modello*, modello del veicolo (135 modalità).
- *Serie*, serie del veicolo (19 modalità).

Lo scopo dell'analisi è prevedere la durata totale di ogni processo.

### 3.1.1 Operazioni preliminari

Sono state fatte alcune operazioni preliminari prima di effettuare la fase di modellazione statistica:

- La variabile risposta *Target* è strettamente positiva e ha un *range* elevato (0.01-6981.87), perciò per tutti i modelli eccetto quello per dati di durata verrà applicata la trasformata logaritmica per liberare il suo dominio da vincoli e rendere meno influenti i valori anomali.
- La variabile *DEA* è stata rimossa avendo troppe modalità.
- Le variabili *Modello* e *Serie* non trovavano corrispondenza biunivoca rispetto alla *Marca*, perciò si suppone che non abbiamo un chiaro significato sulle caratteristiche del veicolo e sono state perciò eliminate.
- La variabile *Avaruser* è stata trasformata nel conteggio del numero di utenti coinvolti all'interno di un processo per ridurre il numero elevato di modalità.
- *Gravità* e *Marca* hanno rispettivamente il 47% e 1% di valori mancanti: nella prima variabile verrà creata la categoria *NA* in quanto potrebbe avere senso che la carenza del dato sia dipendente dal valore sottostante, nella seconda i valori mancanti verranno sostituiti dalla categoria più frequente, ovvero *FIAT*.

### 3.1.2 Modellazione statistica

Di seguito vengono elencati i modelli utilizzati assieme a una breve descrizione della procedura utilizzata per la scelta dei parametri di regolazione, non è stata eseguita una fine calibrazione in quanto uno degli aspetti che si vuole tenere in considerazione e la semplicità di regolazione:

- *Benchmark*: la media condizionata alla *Marca* e al *Mercato*, ovvero quelle variabili che descrivono l'ambiente circostante in cui si sviluppa l'intervento di assistenza.

- *Lasso rilassato* utilizzando una convalida incrociata 5-*folds* per scegliere il parametro di regolazione *lambda* (Meinshausen, 2007).

- *Modello di Cox*: i valori temporali sono stati approssimati a numeri interi, quindi per ogni osservazione del *dataset* di verifica è stata stimata la funzione di sopravvivenza, infine si è calcolata la mediana per ognuna delle curve come valore predetto finale.

- *Random forest* con 200 alberi con numero di variabili campionate per ogni *split* pari a 10 (scelto tra 6, 8, 10) e ampiezza minima dei nodi è pari a 5 (seguendo le indicazioni di Hastie, Tibshirani e Friedman, 2009, pag. 592).

- *Extreme gradient boosting* (XGBoost) con 100 alberi, profondità massima pari a 10 (scelto tra 6, 8, 10), valore del *learning rate* pari a 0.08 (scelto tra 0.02, 0.04 e 0.08), sottocampione utilizzato per ogni albero pari al 50% (scelto tra 50% e 100%) e la porzione di variabili campionate per ogni *split* pari al 66,66% (per una guida sulla regolazione si veda il sito `www.slideshare.net/ShangxuanZhang` e Friedman, 2001).

- *Feed-forward neural network* (FFNN) con 2 strati latenti da 821 e 164 nodi rispettivamente; per risolvere il problema dei minimi locali le stime sono state rieseguite 3 volte facendo la media delle previsioni ottenute (Ripley, 2007), per ogni rete è stato applicato un *early stopping* alla $10^a$ epoca e una penalità *weight decay* pari a 0.0001 (scelta tra 0.01, 0.001 e 0.0001), per la scelta sull'architettura della rete sono state seguite

le indicazioni di Huang (2003) e Bengio e Delalleau (2011), infine è stata scelta la funzione di attivazione $ReLU$ per la presente e tutte le successive reti neurali.

- *Feed-forward neural network* (FFNN+dropout) con 2 strati latenti da 821 e 164 nodi rispettivamente; è stato applicato il *dropout*, con probabilità di conservare i nodi pari a 50% per gli strati latenti e 80% per gli strati di *input*, un *early stopping* alla $20^a$ epoca ed una penalità *weight decay* pari a 0.0001.

- *Feed-forward neural network* (FFNN+MC-dropout) con gli stessi parametri precedenti, ad eccezione di applicare il *Monte Carlo dropout* al posto del semplice *dropout*[1].

- *Cascade forest* composto da 2 livelli, con 4 *Random forest* e da 4 *Complete-Random trees* per livello e utilizzando le previsioni da attaccare ad ogni strato successivo OOB (CF_4×4_oob), 3- e 5-*fold* CV (CF_4×4_cv3 e CF_4×4_cv5). Ogni foresta ha un numero di alberi pari a 200, mentre minima ampiezza dei nodi pari a 5 in accordo con i paragrafi 1.2 e 1.3.

- *Vincitore BG*: per avere un termine di paragone più chiaro, è stato riportato il punteggio migliore raggiunto durante la manifestazione, ottenuto da un *Random forest* con 500 alberi e numero di variabili campionate per ogni *split* pari a 8.

Le previsioni ottenute sono state poi riportate alla scala originaria mediante la trasformazione esponenziale, per valutare la capacità previsiva dei modelli statistici sopra descritti si è utilizzato l'errore medio assoluto per la sua maggiore semplicità interpretativa. I risultati sono mostrati nella Tabella Tabella 3.1.
*BeeViva* mostra nell'interfaccia utente il punteggio parziale e solo alla fine della competizione rivela il punteggio finale. Se gli errori finali sono superiori a quelli parziali, si è verificato il problema dei test multipli accennato nel

[1]Per una descrizione sul *dropout* e *Monte Carlo dropout*, si veda Aere (2017).

**Tabella 3.1:** Risultati sulle previsioni del dataset *Business Game* 2017.

| | **Errore assoluto medio** | |
|---|---|---|
| **Modello** | Parziale | Finale |
| Benchmark | 169 | 168 |
| Lasso rilassato | 164 | 164 |
| Cox | 155 | 151 |
| Vincitore BG | 139 | 141 |
| FFNN | 141 | 140 |
| Random Forest | 138 | 136 |
| XGBoost | 138 | 136 |
| FFNN+dropout | 136 | 135 |
| FFNN+MC-dropout | 134 | 132 |
| CF_4×4_oob | 134 | 132 |
| CF_4×4_cv5 | 134 | 131 |
| CF_4×4_cv3 | 133 | 131 |

Capitolo 1 (Cohen e Jensen, 1997): effettuare un numero eccessivo di *submission* ha portato a scegliere il modello che si è adattato maggiormente ai dati solamente grazie al caso.
Si noti come i 3 tipi di *Cascade Forest* si collocano nelle prime 3 posizioni, segue il *Feed-forward neural network* con MC-*dropout*. CF_4×4_oob ha una previsione leggermente peggiore rispetto alle altre 2 versioni, ma con un costo computazionale decisamente inferiore (si veda il Paragrafo 3.4).

### 3.1.3 Regolazione Cascade forest

Per la regolazione del *Cascade Forest* si sono seguite le conclusioni riportate al Paragrafo 2.4. Per selezionare il livello ottimale sono stati impiegati la 5-*folds* CV utilizzando l'intero *dataset* di stima e l'approccio stima-verifica dividendo ulteriormente il *dataset* disponibile in 2 sotto-campioni di dimensioni pari a 60618 e 20207. Il primo verrà utilizzato come insieme di stima, il secondo come insieme di verifica.
Avendo scelto quale *K-folds* CV utilizzare, rimangono 2 parametri di regolazione:

- Numero di livelli, in quanto un eccessivo valore porta all'*overfitting*.
- Numero di foreste in ogni livello, non tanto per un problema di sovradattamento, ma per ridurre al minimo i costi computazionali.

I risultati sono illustrati nella Figura 3.1. L'errore di verifica (test_cv5) segnala 2 livelli, mentre quello mediante 5-*folds* CV (cv5) si stabilizza attorno a 5. In questo caso si predilige la scelta più parsimoniosa. Nel pannello di destra si vede come varia l'errore di verifica all'aumentare del numero di foreste tenendo costante il numero di livelli pari a 2: in questo caso il minimo è 12, ma già con 8 foreste la diminuzione non copre i maggiori oneri computazionali (notare la scala delle ordinate).

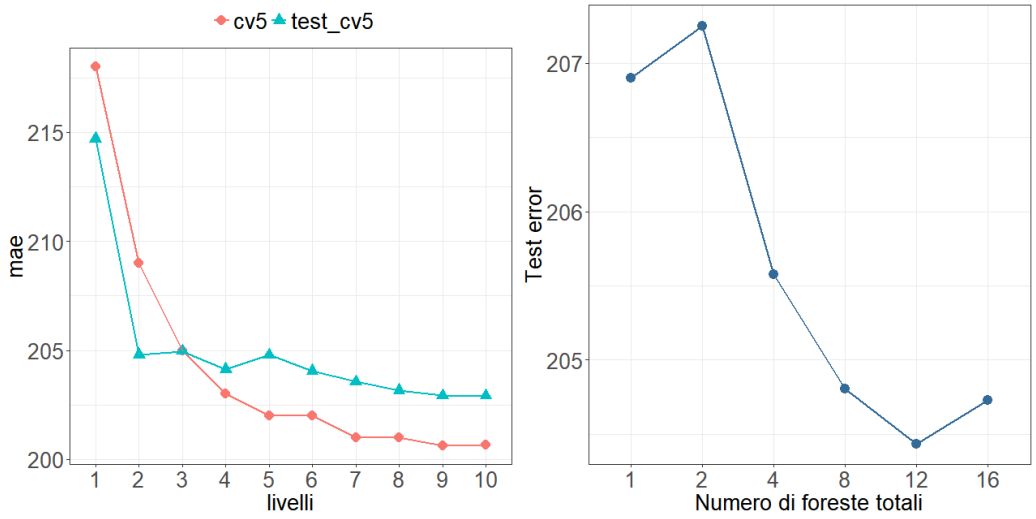


**Figura 3.1:** Andamento dell'errore 5-*folds* CV (cv5) e dell'errore di verifica (test_cv5) all'aumentare del numero di livelli (pannello di sinistra) e andamento dell'errore di verifica all'aumentare del numero di foreste totali tenendo costante il numero di livelli pari a 2 (pannello di destra).

## 3.2 Classificazione

Il *dataset* analizzato proviene dal gruppo bancario *Findomestic* ed è stato utilizzato nel corso dell'evento *Stat under the stars 3* (i risultati sono disponibili al sito `/local.disia.unifi.it/sus3`).

*Findomestic* vuole compiere un'azione promozionale dei suoi prestiti rivolta ai suoi 200000 clienti ai quali si rivolge in modo individuale, mediante il canale telefonico. Per motivi di costo, si decide di effettuare un numero limitato di chiamate, e quindi sorge il problema di selezionare i clienti da chiamare.
Per ottenere un *dataset* di stima *Findomestic* seleziona 40000 clienti a caso e aspetta il mese successivo per verificare i risultati della campagna promozionale. Grazie alle informazioni ottenute vuole selezionare i 10000 clienti tra i restanti 160000 maggiormente propensi ad accettare l'offerta; il metro di valutazione sarà perciò la percentuale dei clienti che accetteranno la proposta tra i 10000 selezionati.
Le informazioni a disposizioni possono essere ricondotte a 3 macro categorie:

- *caratteristiche socio-demografiche*, che includono:
  - *AGE* (età), *ANZ_BAN* (anzianità bancaria), *ANZ_RES* (anzianità residenza) e *ANZ_PROF* (anzianità professionale) espresse in anni.
  - *IMP_RD* (reddito richiedente) e *IMP_FAM* (reddito famigliare) espresso in euro.
  - *COD_RES*, tipologia residenza (6 modalità).
  - *COD_STAT_CIV*, stato civile (6 modalità).
  - *NUM_FIGLI*, numero di figli categorizzato (3 modalità).
  - *FLG_SEX*, sesso (2 modalità).
  - *PROF*, professione (9 modalità).
- *Equipaggiamento del cliente*, che include:
  - Il numero dei prestiti in corso di diversa tipologia (10 variabili) e il loro ammontare (14 variabili).
  - Se il cliente è in possesso di carta di credito (CRT_PRE_C_FLG_PRE) e relativo ammontare (CRT_TODU_REV).
- *Storico del cliente*, tra cui:

– Richiesta di prestito personale negli ultimi 18 mesi, mesi trascorsi (*FIND_PPQ18SS_MONTH_DAT_MAX_FIN*) e numero di richieste (*FIND_PPQ18SS_NUM_PRA*).

– Descrizione dei prestiti saldati negli ultimi 18 mesi (24 variabili).

– Pagamenti effettuati con la carta di credito negli ultimi 18 mesi, suddivisi per tipo di rimborso (9 variabili).

### 3.2.1 Operazioni preliminari

Sono state fatte alcune operazioni preliminari precedenti alla modellazione statistica.

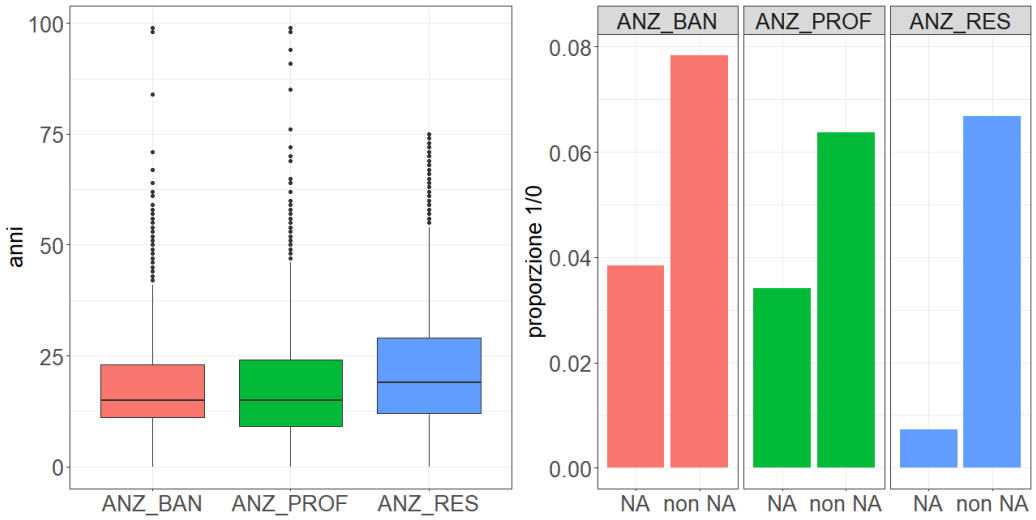


**Figura 3.2:** Distribuzione variabili socio-demografiche

Come si può vedere dal pannello di sinistra della figura Figura 3.2, Le variabili *ANZ_BAN*, *ANZ_PROF* e *ANZ_RES* hanno molti *ouliers*. Seguendo un ragionamento logico il cliente può essere residente in una certa zona da 75 anni, ma certo non può avere un'anzianità professionale (compresi gli anni pensionistici) di 100 anni. Assumendo un errore nel *database* aziendale, tutti i valori di *ANZ_BAN* e *ANZ_PROF* superiori a 75 verranno contrassegnati come valori mancanti.
Nel pannello di destra si è calcolata la proporzione tra i clienti che hanno

sottoscritto il prestito e quelli che non l'hanno sottoscritto condizionata alla presenza o meno di valori mancanti per ognuna delle 3 variabili. Il grafico risultante mostra un evidente cambiamento di distribuzione, perciò verranno create 3 variabili dicotomiche che segnaleranno la presenza o meno di valori mancanti, le variabili continue tuttavia verranno mantenute riempendo i valori mancanti col valor medio corrispondente.
Infine la colonna *FIND_PPQ18SS_MONTH_DAT_MAX_FIN* ha il 98% di valori mancanti, tuttavia in questo caso essi indicano l'assenza della richiesta di finanziamento da parte del cliente, quindi la variabile in questione verrà trasformata in una variabile dicotomica indicante la richiesta o meno di un prestito personale.

### 3.2.2 Modellazione statistica

Di seguito i modelli stimati secondo le linee guida del precedente paragrafo:

- *Random*: 10000 clienti scelti a caso. Non conoscendo il numero totale di clienti che hanno accettato la proposta tra i 160000 la percentuale ottenuta dai modelli successivi potrebbe essere forviante e perciò si utilizzerà questo risultato come termine di paragone.

- *Naive Bayes*, assumendo per le variabili continue una distribuzione normale multivariata (Hastie, Tibshirani e Friedman, 2009, pag. 210).

- *Lasso rilassato* utilizzando una convalida incrociata 5-*folds* per scegliere il parametro di regolazione *lambda*.

- *Random forest* con 200 alberi, con numero di variabili campionate per ogni *split* pari a 8 (scelto tra 6, 8, 10) e ampiezza minima dei nodi è pari a 1 (seguendo le indicazioni di Hastie, Tibshirani e Friedman, 2009, pag. 592).

- *Extreme gradient boosting* con 100 alberi, profondità massima pari a 10 (scelto tra 6, 8, 10), valore del *learning rate* pari a 0.08 (scelto tra

0.02, 0.04 e 0.08) e porzione di variabili campionate per ogni *split* pari al 11%.

- *Feed-forward neural network* con 4 strati latenti da 19 nodi ciascuno; per risolvere il problema dei minimi locali le stime sono state rieseguite 3 volte facendo la media delle previsioni ottenute, per ogni rete è stato applicato un *early stopping* alla $10^a$ epoca e una penalità *weight decay* pari a 0.0001 (scelta tra 0.01, 0.001 e 0.0001).

- *Feed-forward neural network* con 2 strati latenti da 577 e 115 nodi rispettivamente; è stato applicato il *dropout*, con probabilità di conservare i nodi pari a 50% per gli strati latenti e 80% per gli strati di *input*, un *early stopping* alla $20^a$ epoca ed una penalità *weight decay* pari a 0.0001.

- *Feed-forward neural network* con 2 strati latenti da 577 e 115 nodi rispettivamente; è stato applicato il *Monte Carlo dropout*, con probabilità di conservare i nodi pari a 50% per gli strati latenti e 80% per gli strati di *input*, un *early stopping* alla $20^a$ epoca ed una penalità *weight decay* pari a 0.0001.

- *Cascade forest* composto da 2 livelli, con 4 *Random forest* e da 4 *Complete-Random trees* per livello e utilizzando le previsioni da attaccare ad ogni strato successivo OOB (CF_4×4_oob), 3- e 5-*fold* CV (CF_4×4_cv3 e CF_4×4_cv5). Ogni foresta ha un numero di alberi pari a 200, mentre minima ampiezza dei nodi pari a 5 in accordo con i paragrafi 1.2 e 1.3.

- *Vincitore SUS*: anche in questo caso viene riportato il la miglior previsione ottenuto durante la competizione. Si tratta di un *ensemble model* combinando le previsioni di 3 modelli mediante la media aritmetica: un *Random forest* con 200 alberi e variabili campionate per ogni *split* pari a 8; un *Extreme gradient boosting* con 2000 alberi, profondità massima pari a 3 e valore del *learning rate* pari a 0.01; una *Feed-forward neural network* con 2 strati latenti da 110 e 50 nodi rispettivamente.

Come spiegato sopra, per valutare la capacità previsiva dei modelli statistici descritti precedentemente si è utilizzato la percentuale di clienti che hanno accettato l'offerta tra i 10000 selezionati. I risultati sono mostrati nella tabella Tabella 3.2.

**Tabella 3.2:** Risultati sulle previsioni del dataset *Stat under the stars*.

| | **Percentuale clienti previsti** | |
|---|---|---|
| **Modello** | Parziale | Finale |
| Random | 5.45% | 5.89% |
| Naive bayes | 39.50% | 38.91% |
| Lasso rilassato | 49.50% | 49.16% |
| FFNN | 50.80% | 49.65% |
| FFNN+dropout | 50.30% | 50.46% |
| FFNN+MC-dropout | 50.60% | 50.46% |
| Random forest | 52.25% | 51.61% |
| CF_4×4_oob | 52.70% | 51.64% |
| CF_4×4_cv5 | 52.65% | 51.78% |
| CF_4×4_cv3 | 52.75% | 51.92% |
| XGBoost | 52.25% | 51.96% |
| Vincitore SUS | 52.70% | 52.36% |

In questo caso il *Cascade Forest* non è riuscito a eguagliare il miglior punteggio, ma soprattutto rispetto alla *Random forest* vi è una diminuzione dell'errore troppo modesta. Si potrebbe ipotizzare che il *Complete-random trees* non è adeguato per eventi rari (la variabile risposta ha il 6% di 1) data la casualità degli *split*. Rispetto al caso precedente abbiamo un fallimento delle reti neurali, mentre le 3 versioni di *Cascade Forest* ottengono errori quasi identici.

### 3.2.3 Regolazione Cascade forest

Si procede come nel Paragrafo 3.1.3 a esaminare da un lato l'andamento dell'errore 5-*folds* CV e di verifica all'aumentare dei livelli, dall'altro aumentando il numero di foreste per livello.

Per il calcolo dell'errore di verifica il *dataset* di stima è stato ulteriormente diviso in 2 sotto-campioni di dimensioni pari a 30000 e 10000 e come errore di misura è stata utilizzata la media tra l'errore di 1° e di 2° tipo , ovvero la media tra i falsi positivi e falsi negativi, preferita rispetto all'errore totale in quanto la variabile risposta è fortemente sbilanciata. Dalla Figura 3.3 si

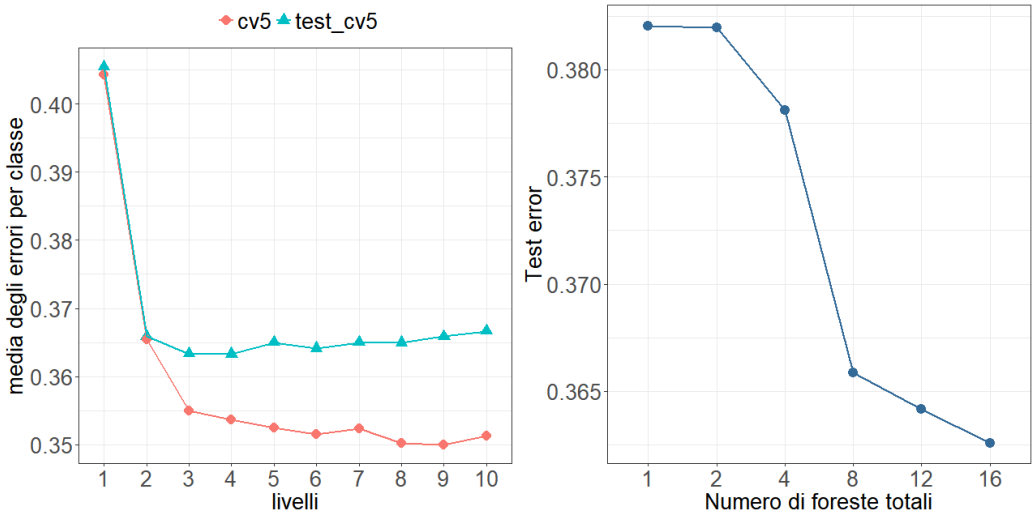


**Figura 3.3:** Andamento dell'errore di classificazione medio stimato da una 5-*folds* CV (cv5) e dall'errore di verifica (test_cv5) all'aumentare del numero di livelli (pannello di sinistra) e andamento dell'errore di verifica all'aumentare del numero di foreste totali tenendo costante il numero di livelli pari a 2 (pannello di destra).

vede (pannello di sinistra) che l'andamento dell'errore di verifica (test_cv5) produce un gomito in corrispondenza di 2 livelli, mentre l'errore tramite 5-*folds* CV (cv5) si stabilizza a 3.
Il pannello di destra mostra l'andamento dell'errore di verifica all'aumentare del numero di foreste dato un numero di livelli pari a 2. Il minimo viene raggiunto in corrispondenza di un numero di foreste pari a 16, anche se il guadagno di precisione da 8 in poi è irrisorio.

## 3.3 Multiclassificazione

I dati provengono dal *dataset MNIST* (*Modified National Institue of Standards and Technology*). Esso è composto da immagini in bianco e nero, ognuna delle quali rappresenta una cifra da "0" a "9" scritta a mano. Ogni immagine ha grandezza $28 \times 28$ pixel, ognuno dei quali assume un valore da 0 a 255, a seconda della tonalità di grigio.

### 3.3.1 Modellazione statistica

Considerando ognuno dei *pixel* come una variabile esplicativa, sono stati stimati i seguenti modelli:

- *Naive Bayes*, assumendo per le variabili continue una distribuzione normale multivariata.
- *Lasso rilassato* utilizzando una convalida incrociata 5-*folds* per scegliere il parametro di regolazione *lambda*.
- *Random forest* con 200 alberi; il numero di variabili campionate per ogni albero è impostato a 30 (scelto tra 26, 28, 30, 32, 34) e ampiezza minima dei nodi è pari a 1.
- *Extreme gradient boosting* con 100 alberi, con profondità massima pari a 6 (scelto tra 6, 8, 10), valore di *learning rate* pari a 0.08 (scelto tra 0.02, 0.04 e 0.08) e porzione di variabili campionate per ogni *split* pari al 30% (scelta tra i valori 10%, 20%, 30%, 50% e 100%).
- *Feed-forward neural network* con 2 strati latenti da 1024 e 512 nodi rispettivamente; è stato applicato un *early stopping* alla $20^a$ epoca ed una penalità *weight decay* pari a 0.001 (scelta tra 0.01, 0.001 e 0.0001).
- *Feed-forward neural network* con 2 strati latenti da 1024 per ogni strato; è stato applicato il *dropout* solo agli strati latenti, con probabilità di conservare i nodi pari a 50%. Altri parametri sono un *early stopping* alla $20^a$ epoca ed una penalità *weight decay* pari a 0.001.

- *Feed-forward neural network* con 3 strati latenti da 1024 nodi ognuno; è stato applicato il *Monte Carlo dropout*, con probabilità di conservare i nodi pari a 50% solo agli strati latenti, un *early stopping* alla $20^a$ epoca ed una penalità *weight decay* pari a 0.001.

- *LeNet-5*: una versione moderna della *Lenet*[2] (LeCun et al., 1998) con un *dropout* applicato agli strati latenti del 50% e funzione di attivazione $ReLU$.

- *Cascade forest* composto da 5 strati ognuno con 4 *Random forest* e da 4 *Complete-Random trees*, le previsioni da attaccare ad ogni strato successivo sono stati ottenuti da un 3-*fold* CV; il numero di alberi di ogni foresta è pari a 200.

- *Multi-Grained Cascade Forest* utilizzando 3 finestre mobili di grandezza $7 \times 7$, $9 \times 9$ e $14 \times 14$, ogni finestra verrà processata da un *Complete-Random tree* e da una *Random forest* da 200 alberi ciascuno; le previsioni ottenute saranno poi utilizzate come *input* del *Cascade Forest* con la medesima configurazione precedente.

- *Ensemble di CNN*: a cui corrisponde il minor errore ottenuto nel corso degli anni. Si tratta di 5 *Convolutional neural network*, ognuna formata da 6 strati convoluzionali da 784, 50, 100, 500, 1000 e 10 nodi rispettivamente. Le previsioni sono state poi aggregate mediante la media aritmetica.

Per valutare la bontà di previsione dei modelli 60000 immagini verranno utilizzate come insieme di stima, 20000 come insieme di verifica e si calcolerà l'errore totale di errata classificazione. I risultati sono rappresentati in tabella Tabella 3.3.

Nel corso degli anni il *dataset* è stato analizzato in vari modi. Di questi sono stati riportati solamente *LeNet-5* e l'*ensemble di CNN* come rappresentanti delle *convolutional neural networks*, mentre le reti neurali invece sono state

[2] *Lenet* è una *Convolutional neural network* composta da 2 strati convoluzionali e due strati di *pooling*, disposti in sequenza alternata, e seguiti da uno strato composto da 500 nodi.

**Tabella 3.3:** Risultati sulle previsioni del *dataset* MNIST.

| **Modello** | **Errore di classificazione** |
|---|---|
| Naive Bayes | 12.09% |
| Lasso rilassato | 7.19% |
| Random forest | 3.20% |
| Cascade Forest | 1.98% |
| FFNN | 1.79% |
| XGBoost | 1.81% |
| FFNN+dropout | 1.71% |
| FFNN+MC-dropout | 1.43% |
| CNN (LeNet-5) | 0.95% |
| GcForest | 0.74% |
| Ensemble di CNN | 0.21% |

ristimate per mantenere le stesse analisi dei precedenti *dataset*. Comunque sul sito `yann.lecun.com/exdb/mnist/` le reti neurali hanno un errore compreso tra 4.7% e 1.6% senza effettuare alcun pre-processamento ai dati, mentre utilizzando il processo di *elastic distorsion*[3] l'errore scende a un punteggio di 0.35 % con 6 strati composti rispettivamente da 84-2500-2000-1500-1000-500. Il *gcForest* si colloca in mezzo tra *LeNet-5* e l'*ensemble* di CNN.

### 3.3.2 Regolazione gcForest

Tenendo il numero di foreste per livello pari a 8, si procede a scegliere il numero di *grains* e il numero di strati.

Nella Figura 3.4 (pannello di sinistra) si vede che la *performance* del *gcForest* può essere ulteriormente migliorata aumentando il numero di finestre (*grains*), mentre l'errore da 5 livelli in poi rimane pressoché costante. Purtroppo non è stato possibile esplorare anche l'effetto di cambiamenti simultanei dei parametri a causa delle risorse computazionali limitate.

[3]Modo di emulare la casuale oscillazione dei muscoli delle mani mentre le persone scrivono (Simard, Steinkraus e Platt, 2003).

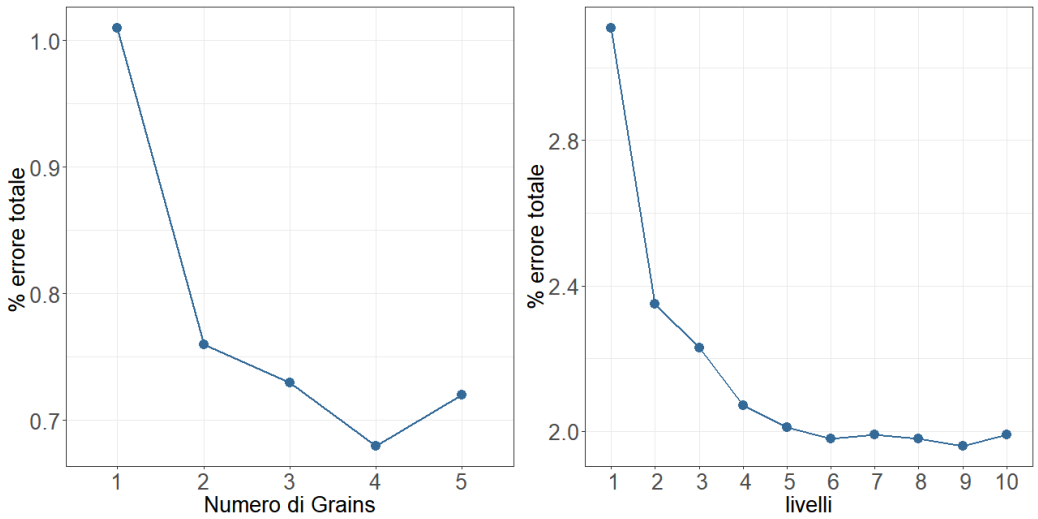


**Figura 3.4:** Andamento dell'errore totale modificando il numero di *grains* (pannello di sinistra) o il numero di foreste fissando il precedente parametro a 3 (pannello di destra).

## 3.4 Aspetti computazionali

Nel Paragrafo 1.1 sono state presentate 2 possibili caratteristiche degli algoritmi *ensemble*: algoritmo sequenziale, dove i modelli di base sono generati sequenzialmente (e.g. *AdaBoost* e *Gradient Boosting*), e algoritmo *ensemble* parallelo, dove i modelli di base sono generati contemporaneamente, come il *Bagging* e di conseguenza la *Random forest* e il *Complete-Random trees*. La seconda categoria è facilmente parallelizzabile, tanto da essere definita nel linguaggio della programmazione *embarrassingly parallel*.

### 3.4.1 Embarrassingly parallel

Parallelizzare un processo significa suddividere le operazioni computazionali e affidarle a più processori (*cores*). Così facendo il tempo computazionale nell'eseguire il calcolo sequenzialmente viene moltiplicato per un fattore pari a $1/c$, dove $c$ è il numero dei processori.
La parola *embarrassingly* allude al fatto che il problema affrontato è così facile da parallelizzare che non c'è bisogno di alcuna ricerca supplementare, in

quanto la soluzione è banale. Ad esempio, non è *embarrassingly parallel* l'ordinamento di un vettore, in quanto il compito a prima vista non può essere parallelizzato, ma richiede uno specifico algoritmo denominato *Merge sort*. Condizione necessaria affinché il processo sia *embarrassingly parallel* è la possibilità di essere suddiviso in sotto-compiti indipendenti fra loro, condizione rispettata per tutti i modelli *ensemble* paralleli.
Nel *Cascade Forest* e nel *Multi-Grained Scanning* il calcolo parallelo può essere applicato su 3 livelli:

- All'interno di ogni livello.
- All'interno di ogni modello *ensemble*.
- All'interno di ogni albero.

Parallelizzare ogni livello significa che i modelli che lo compongono verranno stimati da processori differenti. I risultati saranno poi trasferiti al processore iniziale che li aggrega e fa ripartire il processo.
Parallelizzare ogni modello ha la stessa logica precedente, ma invece di suddividere i metodi *ensemble*, vengono suddivisi gli alberi associati.
Infine, si ricordi l'algoritmo di crescita di un generico albero di regressione utilizzando solo variabili quantitative: ad ogni nodo, ciascuna variabile esplicativa viene ordinata in base al valore delle osservazioni, si suddivide quindi lo spazio campionario in ogni possibile partizione condizionatamente alla singola variabile esplicativa e si sceglie infine lo *split* che massimizza la diminuzione di devianza. Il costo computazionale di ordinare un singolo vettore è dell'ordine di $O(n\log n)$, mentre la ricerca del miglior *split* ha un costo lineare $O(n)$.
Un albero può essere quindi parallellizzato in 2 modi:

- Parallelizzando la ricerca degli *split* associati ai vari nodi.
- Parallelizzando la ricerca dello *split* all'interno di un singolo nodo.

Il primo metodo può essere applicato quando il numero di nodi è maggiore di 1 e non permette una suddivisione equa dei compiti, infatti un nodo può contenere un numero molto più elevato di osservazioni rispetto agli altri.

Il secondo metodo è molto più efficiente perché permette la suddivisione dei compiti all'interno di un singolo nodo, ovvero ogni processore calcola separatamente le suddivisioni associate a differenti variabili. Tuttavia nei piccoli nodi i vantaggi non coprono i costi: quando un nodo contiene poche osservazioni e ha un costo computazionale esiguo, i vantaggi del calcolo parallelo non coprono i costi dati dal trasferimento di informazioni da un processore all'altro.
Per un ulteriore vantaggio si potrebbe ordinare le variabili esplicative all'inizio del processo di crescita e salvare gli indici corrispondenti a ogni osservazione, per non dover rifare l'operazione ad ogni nodo dell'albero, tuttavia questo comporta un maggior dispendio di memoria e i 2 aspetti devono essere bilanciati.
In questa tesi sono state utilizzate per semplicità le ultime 2 parallelizzazioni grazie alla funzione del pacchetto **ranger** (*RANdom forest GENerator*, Wright e Ziegler, 2015) che a seconda della dimensione dei nodi sceglie se salvare gli indici in memoria o ricalcolarli allo *split* successivo.

### 3.4.2 Librerie utilizzate

L'autore ha messo a disposizione la libreria **gcForest** in linguaggio **Python** all'indirizzo `github.com/kingfengji/gcForest` che utilizza a sua volta la libreria **sklearn**. Le istruzioni riportate sul sito sono solamente per i sistemi operativi **Linux** e le funzioni associate possono essere eseguite direttamente da terminale.
In **R** non erano presenti funzioni soddisfacenti, quindi si è scelto la libreria **ranger** come base per l'implementazione (per dettagli sulla selezione dei candidati, si veda l'Appendice A). La libreria risultante (**deepForest**) ha l'opzione di poter utilizzare le previsioni OOB o le previsioni tramite *K-folds cross validation*, disponibile all'indirizzo `github.com/ablanda/deepForest`.

In Tabella 3.4 sono riportati i tempi computazionali delle 2 librerie, stimando un *Cascade Forest* composto da 4 livelli, 2 *Random forest* e 2 *Complete-Random trees* per livello e utilizzando le previsioni OOB (ranger_OOB) e le previsioni 3-*folds* CV (ranger_cv3 e gcForest). Con **Python** si ha una diminu-

**Tabella 3.4:** Tempi di esecuzione di un *Cascade Forest* con 4 livelli e 4 foreste per livello (2 *Random forest* e 2 *Complete-Random trees*) sul *dataset* MNIST utilizzando la libreria **gcForest** e la libreria **deepForest** utilizzando le previsioni OOB e le previsioni 3-*folds* CV.

| **Libreria** | **Software** | **Minuti** |
|---|---|---|
| ranger_cv3 | R | 45 |
| ranger_OOB | R | 26 |
| gcForest | Python | 18 |

zione del 30% in confronto a ranger_OOB e del 60% rispetto a ranger_cv3, questo è dovuto a 2 fattori:

- La libreria **sklearn** è scritta quasi interamente in linguaggio **Python**, mentre **ranger** è implementato in **C++** come linguaggio base e utilizzando l'ambiente **R** come linguaggio di collegamento le operazioni computazionali rallentano.
- **R** ha una cattiva gestione della memoria che comporta un generale rallentamento del computer.

# Conclusioni

In questa tesi si è cercato di analizzare dal punto di vista prevalentemente empirico il nuovo modello *ensemble* chiamato *Deep random forest*.
Nel Capitolo 1 si è spiegato il meccanismo con cui i modelli *ensemble* migliorano l'accuratezza dei *base learner*. Inizialmente si è mostrato che l'errore quadratico medio può essere scisso in 2 componenti (Formula 1.7): varianza e distorsione. Il *Bagging* e i modelli associati migliorano la previsione solamente grazie ad una diminuzione della varianza, lasciando inalterata la distorsione, mentre lo *Stacking* cerca al contempo di diminuire entrambe le componenti. Per capire il funzionamento dello *Stacking* è stato considerato solamente il modello lineare come *meta-learner*, ma la procedura può essere generalizzata a un qualsiasi modello statistico e di *machine learning*.
Il Capitolo 2 si apre con la descrizione dettagliata del *Deep random forest*, termine utilizzato per indicare il *Cascade forest* e il *gcForest*. Il primo non è altro che una generalizzazione dello *Stacking* in cui vi possono essere un numero di livelli maggiore di 2, mentre il secondo cerca di gestire le relazioni spaziali tra i *pixel* in un problema di classificazione di immagini e le relazioni sequenziali in un problema di riconoscimento vocale.
Nella seconda parte del capitolo è stato effettuato uno studio di simulazione per poter definire dei parametri di *default* del *Cascade Forest* e specificare la procedura per una scelta corretta dei parametri di regolazione. I risultati sono che la miglior previsione da attaccare allo strato successivo è data da una 5-*folds* CV: l'errore associato ha identificato in tutti i 3 studi di simulazione il numero corretto di livelli. Se i tempi computazionali risultassero eccessivi si può comunque utilizzare il metodo stima-verifica utilizzando le previsioni OOB, di cui si è dimostrato al Capitolo 1 che sono asintoticamente pari alle

previsioni ottenute tramite una LOOCV.
Il Capitolo 3 è dedicato alle applicazioni pratiche. Zhou e Feng (2017) utilizzavano il nuovo modello *ensemble* esclusivamente nell'ambito del *deep learning*, tuttavia si è cercato di sperimentarlo anche su *dataset* con variabili reali, non fittizie.
Solamente nel primo *dataset* il *Cascade Forest* raggiunge il risultato migliore, mentre nel secondo e terzo viene superato da altri modelli *ensemble*. Tuttavia il metodo funziona in quanto, nei 3 *dataset*, si è riusciti a diminuire l'errore del *Random forest* rispettivamente del 3.7%, 0.54% e 38%, segno inoltre che la composizione dei singoli livelli formata dal *Random forest* e *Complete-Random trees* in egual proporzione è stata studiata appositamente per il *deep learning*. Quindi un logico sviluppo futuro è sperimentare differenti composizioni, cercando di delineare regole per ogni specifico problema.
Inoltre si potrebbe effettuare un'analisi teorica nello stesso modo in cui è stata effettuata per lo *Stacking*.
Per quanto riguarda il *gcForest*, termine per indicare la procedura *Multi-Grained Scanning* unita al *Cascade Forest*, si è semplicemente provato a modificare il numero di *Grains* e di livelli, vedendo che l'errore può essere ulteriormente diminuito fino a 0.68% nel terzo problema. Anche in questo caso sarebbe interessante sperimentare differenti modelli nel *Multi-Grained Scanning*.
Per quanto riguarda l'obiettivo iniziale di cercare un'alternativa alla rete neurale, la conclusione è negativa, piuttosto la rete neurale potrebbe essere utilizzata come modello di base dell'*ensemble*.
Infine viene presentato un confronto tra il linguaggio **`Python`** e **`R`** mostrando che il primo comporta un minor dispendio computazionale.

# Appendice A

# Implementare Cascade Forest

Le librerie prese in considerazione per implementare il *Cascade Forest* sono state le seguenti:

- **randomForest**.
- **extraTrees**.
- **xgboost**.
- **h2o**.
- **ranger**.

La libreria **randomForest** è stata sviluppata da Breiman e Cutler (*Random Forests*) in linguaggio **Fortran 77**, portata su **R**, come linguaggio di collegamento, da Liaw (*Package randomForest*).
Purtroppo la libreria è estremamente inefficiente nell'allocazione della memoria e solamente nel 2° *dataset* (3.2) si è riuscito a stimare il modello con un massimo di 200 alberi utilizzando un PC $Intel^{R}Core^{TM}i7-6500U$ (4 *Cores* e 16 GB di RAM). Inoltre la funzione corrispondente non è parallelizzata, quindi, come si vede dalla Tabella A.1, il tempo di stima è nettamente il più lungo, rendendola utilizzabile solamente per piccoli-medi *dataset*.
**ExtraTrees** è il pacchetto inizialmente utilizzato in coppia con **Random Forest** per generare *Complete-Random trees*. Il codice sorgente è scritto in **Java** e il modello può essere calcolato in parallelo scegliendo un numero di *Cores*

**Tabella A.1:** Precisione associata alle 4 librerie utilizzando il *dataset Stat under the stars*.

| Libreria | Percentuale clienti previsti | Secondi |
|---|---|---|
| randomForest | 51.90% | 335 |
| ranger | 51.70% | 40 |
| h2o | 51.65% | 28 |
| xgboost | 50.50% | 24 |
| extraTrees | 48.95% | 16 |

maggiore di 1 (opzione di *deafault*) mediante l'opzione `numThreads`. Come si vede dalla Tabella A.1, grazie al calcolo parallelo e agli *split* casuali, il tempo di stima è il migliore a discapito della bontà di adattamento.
Su *GitHub* ad Aprile 2017 è stata implementata una funzione utilizzando la libreria **`xgboost`**, disponibile al profilo *Laurae2/Laurae: Advanced High Performance Data Science Toolbox for R by Laurae*. Essa utilizza **`C++`** come linguaggio di base, vanta una parallelizzazione degli alberi a livello dei singoli nodi (Chen e Guestrin (2016)) ed è utilizzato principalmente per stimare il *Gradient Boosting*.
Inoltre per diminuire ulteriormente i tempi computazionali, la funzione `xgb.train` permette una approssimazione dell'algoritmo dove, per ogni variabile continua, si traccia la funzione di ripartizione empirica e si valutano gli *split* posti a intervalli di percentili equispaziati.
La funzione `xgb.train`, come spiegato al sito `xgboost.readthedocs.io/en/latest/parameter.html`, permette di implementare una *Random Forest* impostando i seguenti parametri:

- `nrounds=1`, il *gradient boosting* è un algoritmo iterativo, dove ogni albero è stimato sui residui del modello della iterazione precedente, in questo modo tutti gli alberi vengono stimati sui dati originali;
- `num_parallel_tree`, numero di alberi da far crescere in questa singola iterazione;
- `sampleby_level`, proporzione di colonne campionate a ogni nodo dell'albero, nel rispetto dei consigli del paragrafo 1.2, si consiglia il valore

$\sqrt{p}/p$ per i problemi di classificazione e 1/3 per quelli di regressione;

- `subsample=0.632`, `xgb.train` permette solo un campionamento senza reinserimento, tuttavia, come dimostrato nell'equazione 1.20, il numero atteso di osservazioni originali in un campione *bootstrap* è pari al 63.2% dell'intero campione;

- `max_depth = 99999`, le impostazioni del paragrafo 1.2 indicano per i problemi di classificazione una minima ampiezza dei nodi terminali pari a 1, tuttavia in questo caso la crescita dell'albero è regolata dalla profondità, pari al maggior numero di *split* tra i nodi e la radice. Non essendoci una chiara relazione tra la profondità e la minima ampiezza dei nodi il numero deve essere arbitrariamente grande, stesso discorso per i problemi di regressione;

- `max_leaves = 99999`, massimo numero di foglie permesse, numero arbitrariamente grande per i motivi precedenti.

Anche se possiede un basso costo computazionali, presenta importanti difetti. Primo, non è previsto lo *split* casuale dei nodi. Secondo, il *dataset* di *input* deve essere obbligatoriamente un oggetto in formato `matrix`, dove le variabili categoriali non sono ammesse e devono essere codificate in qualche modo (si veda `github.com/Laurae2/CategoricalAnalysis` per una relazione sulle migliori codifiche da utilizzare in un albero decisionale). Se sono trasformate in *I-1 dummy*, dove $I$ è il numero di categorie, allora il numero di variabili selezionate a ogni nodo dell'albero deve essere raddoppiato o triplicato secondo Breiman (2001). Anche con questo accorgimento i risultati sono notevolmente peggiori rispetto alle altre librerie, rendendo utilizzabile la funzione solamente in *dataset* dove il numero di variabili categoriali non sia troppo elevato.
La seconda in ordine di rapidità è la libreria **`h2o`**.
Essa utilizza **`Java`** come linguaggio di base, ma può essere eseguita anche in molti altri linguaggi ad esso collegati come, ad esempio, **`Python`** e **`R`**. Richiede una classe speciale di *dataset*, il formato **`h2o`**. La logica è di salvare nel *workspace* di **`R`** degli oggetti **`h2o`** che non sono altro che collegamenti al *dataset* reale caricato in **`Java`**. Questo ha l'indiscusso vantaggio di poter analizzare

*dataset* che non potrebbero nemmeno essere caricati in **R** nella maniera tradizionale. Permette *split* casuali nella crescita degli alberi mediante l'opzione `histogram_type = "Random"` e salva in memoria le previsioni associate alla *cross validation*, semplificando la creazione dello strato successivo nel *Cascade Forest*. Per una guida esaustiva visitare il sito `www.h2o.ai` e le varie guide ad esso collegate (*H2O Tutorials* e *DeepLearningBookletV1*).
Tuttavia, una volta implementata la funzione `h2o.cascadeforest`, per stimare lo stesso modello della Tabella 3.4, il tempo computazionale era pari a 335 minuti. Questo perché La funzione `h2o.randomforest`, per aumentare la velocità computazionale e avere un minor dispendio di memoria, gestisce certe operazioni in maniera *lazy* (letteralmente *pigra*), i.e. vengono eseguite solamente quando sono esplicitamente richieste parti dell'*output* (*H2O Module — H2O documentation*). Dato che l'algoritmo *Cascade Forest* incolonna le previsioni dei precedenti modelli, questo crea un percorso tortuoso con risultati pessimi dal punto di vista computazionale.
Infine si è utilizzata la funzione `ranger` della omonima libreria di cui si è parlato al paragrafo 3.4. Viene quindi riportato il codice per fare la stima e la previsione del modello descritto in Tabella 3.4. Per evitare di riempire la RAM, ogni modello di base stimato verrà salvato sul disco fisso del computer, per poi essere ricaricato durante le previsioni.

**Codice A.1:** Stima e previsione di un *Cascade Forest*

```
library(devtools)
install_github('ablanda/deepForest')
library(deepForest)
dati<-read.csv('mnist_train.csv',header=F)
dativ<-read.csv('mnist_test.csv',header=F)

# Cascade Forest con 10 livelli, 8 foreste per livello e una 3-folds CV (
    se k=NULL utilizza le previsioni OOB)
m<-power_ranger(y=1,training_frame = dati,validation_frame = dativ,n_
    forest=8,random_forest = 4,early.stop=10,k=3)

# Previsioni del quinto livello
pred<-matrix(0,nrow(dativ),nlevels(dati[,1])-1)
```

```
for(h in 1:(nlevels(dati[,1])-1)){
pred_level<-sapply(1:8,function(j) m$pred_val[[5]][[j]][,h])
pred[,h] <-rowMeans(pred_level)
}
```

# Bibliografia


Aere, Alessandro (2017). «Proprietà statistiche di modelli per il deep learning». Tesi di laurea mag. Università degli studi di Padova.

Azzalini, Adelchi e Bruno Scarpa (2009). *Analisi dei dati e data mining*. Springer Science & Business Media.

*BeeViva*. URL: http://www.bee-viva.com/.

Bengio, Yoshua e Olivier Delalleau (2011). «On the expressive power of deep architectures». In: *Algorithmic Learning Theory*. Springer, pp. 18–36.

Breiman, Leo (1996a). «Bagging predictors». In: *Machine learning* 24.2, pp. 123–140.

— (1996b). «Stacked regressions». In: *Machine learning* 24.1, pp. 49–64.

— (2001). «Random forests». In: *Machine learning* 45.1, pp. 5–32.

— (2002). «Manual on setting up, using, and understanding random forests v3. 1». In: *Statistics Department University of California Berkeley, CA, USA* 1.

Breiman, Leo e Adele Cutler. *Random Forests*. URL: https://www.stat.berkeley.edu/~breiman/RandomForests/.

Breiman, Leo e Philip Spector (1992). «Submodel selection and evaluation in regression. The X-random case». In: *International statistical review/revue internationale de Statistique*, pp. 291–319.

Breiman, Leo et al. (2001). «Statistical modeling: The two cultures (with comments and a rejoinder by the author)». In: *Statistical science* 16.3, pp. 199–231.

*Business game 2017*. URL: http://www.unipd.it/ilbo/business-game-selezione-data-scientist.

Chen, Tianqi e Carlos Guestrin (2016). «Xgboost: A scalable tree boosting system». In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, pp. 785–794.

Cho, Kyunghyun et al. (2014). «Learning phrase representations using RNN encoder-decoder for statistical machine translation». In: *arXiv preprint arXiv:1406.1078*.

Cohen, Paul R, David Jensen et al. (1997). «Overfitting explained». In: *Preliminary Papers of the Sixth International Workshop on Artificial Intelligence and Statistics*, pp. 115–122.

*Deep Forest by ranger*. URL: `https://github.com/ablanda/deepForest`.

*DeepLearningBookletV1*. URL: `https://github.com/h2oai/h2o-2/blob/master/docs/deeplearning/DeepLearningBookletV1.pdf`.

Deng, Houtao, George Runger e Eugene Tuv (2012). «System monitoring with real-time contrasts». In: *Journal of Quality Technology* 44.1, p. 9.

Domingos, Pedro (1998). «Occam's two razors: The sharp and the blunt». In: *KDD*, pp. 37–43.

Efron, Bradley e Robert Tibshirani (1997). «Improvements on cross-validation: the 632+ bootstrap method». In: *Journal of the American Statistical Association* 92.438, pp. 548–560.

Freund, Yoav e Robert E Schapire (1997). «A desicion-theoretic generalization of on-line learning and an application to boosting». In: *European conference on computational learning theory*. Springer, pp. 23–37.

Friedman, Jerome, Trevor Hastie, Robert Tibshirani et al. (2000). «Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors)». In: *The annals of statistics* 28.2, pp. 337–407.

Friedman, Jerome H (2001). «Greedy function approximation: a gradient boosting machine». In: *Annals of statistics*, pp. 1189–1232.

Friedman, Jerome H e Peter Hall (2007). «On bagging and nonlinear estimation». In: *Journal of statistical planning and inference* 137.3, pp. 669–683.

Geurts, Pierre, Damien Ernst e Louis Wehenkel (2006). «Extremely randomized trees». In: *Machine learning* 63.1, pp. 3–42.

Giacomini, Elia (2017). «Convolutional neural networks per il riconoscimento di nudità nelle immagini». Tesi di laurea mag. Università degli studi di Padova.

*GitHub*. URL: https://github.com/.

Goodfellow, Ian, Yoshua Bengio e Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Graves, Alex, Abdel-rahman Mohamed e Geoffrey Hinton (2013). «Speech recognition with deep recurrent neural networks». In: *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, pp. 6645–6649.

*H2O Module — H2O documentation*. URL: http://h2o-release.s3.amazonaws.com/h2o/master/3052/docs-website/h2o-py/docs/h2o.html.

*H2O Tutorials*. URL: docs.h2o.ai/h2o-tutorials/latest-stable/H2OTutorialsBook.pdf.

*H2O.ai*. URL: https://www.h2o.ai.

Hansen, Lars Kai e Peter Salamon (1990). «Neural network ensembles». In: *IEEE transactions on pattern analysis and machine intelligence* 12.10, pp. 993–1001.

Hastie, Trevor, Robert Tibshirani e Jerome Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Second. Springer-Verlag New York.

Huang, Guang-Bin (2003). «Learning capability and storage capacity of two-hidden-layer feedforward networks». In: *IEEE Transactions on Neural Networks* 14.2, pp. 274–281.

Jacobs, Robert A et al. (1991). «Adaptive mixtures of local experts». In: *Neural computation* 3.1, pp. 79–87.

James, Gareth et al. (2013). *An introduction to statistical learning*. Vol. 112. Springer.

Kochina, Elena et al. (2017). «Deep Learning Summit». In: *London*.

Krizhevsky, Alex, Ilya Sutskever e Geoffrey E Hinton (2012). «Imagenet classification with deep convolutional neural networks». In: *Advances in neural information processing systems*, pp. 1097–1105.

*Laurae2/CategoricalAnalysis: Analysis of Categorical Encodings for dense Decision Trees*. URL: https://github.com/Laurae2/CategoricalAnalysis.

*Laurae2/Laurae: Advanced High Performance Data Science Toolbox for R by Laurae*. URL: https://github.com/Laurae2/Laurae.

LeCun, Yann et al. (1998). «Gradient-based learning applied to document recognition». In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Lev, Utkin (2017). «Deep Forest: About a new deep learning model proposed by Zhi-Hua Zhou and Ji Feng». In: *Peter the Great Saint-Petersburg Polytechnic University Polytech Machine Learning Lab*.

Liaw, Andy. *Package randomForest*. URL: https://cran.r-project.org/web/packages/randomForest/randomForest.pdf.

Liu, Fei Tony et al. (2008). «Spectrum of variable-random trees». In: *Journal of Artificial Intelligence Research* 32, pp. 355–384.

Lucretius Carus, Titus (2012). *Of the nature of things*. Resounding Wind Publishing.

Maree, Raphael et al. (2005). «Random subwindows for robust image classification». In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. IEEE, pp. 34–40.

Matloff, Norman (2011). *The art of R programming: A tour of statistical software design*. No Starch Press.

Meinshausen, Nicolai (2007). «Relaxed lasso». In: *Computational Statistics & Data Analysis* 52.1, pp. 374–393.

*Merge sort*. URL: https://it.wikipedia.org/wiki/Merge_sort.

Miller, Kevin et al. (2017). «Forward Thinking: Building Deep Random Forests». In: *arXiv preprint arXiv:1705.07366*.

*MNIST handwritten digit database*. URL: http://yann.lecun.com/exdb/mnist/.

*Official implementation for the paper 'Deep random forest'*. URL: https://github.com/kingfengji/gcForest.

Oshiro, Thais Mayumi, Pedro Santoro Perez e José Augusto Baranauskas (2012). «How many trees in a random forest?» In: *MLDM*. Springer, pp. 154–168.

Pace, Luigi e Alessandra Salvan (1996). *Introduzione alla statistica: Inferenza, verosimiglianza, modelli.-2001.-xvi, 422 p*. Cedam.

*Parallel Gradient Boosting Decision Trees*. URL: http://zhanpengfang.github.io/418home.html.

Ripley, Brian D (2007). *Pattern recognition and neural networks*. Cambridge university press.

Schapire, Robert E (1990). «The strength of weak learnability». In: *Machine learning* 5.2, pp. 197–227.

Simard, Patrice Y, David Steinkraus, John C Platt et al. (2003). «Best practices for convolutional neural networks applied to visual document analysis.» In: *ICDAR*. Vol. 3, pp. 958–962.

*Stat under the stars 3*. URL: http://local.disia.unifi.it/sus3/.

Ting, Kai Ming e Ian H Witten (1999). «Issues in stacked generalization». In: *J. Artif. Intell. Res.(JAIR)* 10, pp. 271–289.

*Winning data science competitions, presented by Owen Zhang*. URL: https://www.slideshare.net/ShangxuanZhang/winning-data-science-competitions-presented-by-owen-zhang.

Wolpert, David H (1992). «Stacked generalization». In: *Neural networks* 5.2, pp. 241–259.

Wright, Marvin N e Andreas Ziegler (2015). «ranger: A fast implementation of random forests for high dimensional data in C++ and R». In: *arXiv preprint arXiv:1508.04409*.

*XGBoost Parameters — xgboost 0.6 documentation*. URL: http://xgboost.readthedocs.io/en/latest/parameter.html.

Xu, Lei, Adam Krzyzak e Ching Y Suen (1992). «Methods of combining multiple classifiers and their applications to handwriting recognition». In: *IEEE transactions on systems, man, and cybernetics* 22.3, pp. 418–435.

Zhou, Zhi-Hua (2012). *Ensemble methods: foundations and algorithms*. CRC press.

Zhou, Zhi-Hua e Ji Feng (2017). «Deep forest: Towards an alternative to deep neural networks». In: *arXiv preprint arXiv:1702.08835*.