

Analisi Comparativa di KAN, MLP, Random forest e XGBoost con Tecniche di Ottimizzazione

Martin Tomassi
Università di Bologna

Corso di Laurea in Ingegneria e Scienze Informatiche

Settembre 2025

Indice

1 Multi-Layer Perceptron (MLP)	9
1.1 Introduzione	9
1.2 Fondamenti matematici	10
1.2.1 Enunciato formale	10
1.2.2 Teorema di approssimazione universale	10
1.2.3 Significato di $\sup_{x \in K}$	11
1.2.4 Ipotesi e Precisazioni	12
1.3 Struttura delle MLP	13
1.3.1 Strati: input, nascosti, output	13
1.3.2 Funzioni di attivazione comuni	14
1.4 Procedura di forward pass	21
1.4.1 Calcolo delle attivazioni	21
1.4.2 Propagazione ed output	22
1.5 Algoritmo di backpropagation	22
1.5.1 Derivazione del gradiente	22
1.5.2 Aggiornamento dei pesi	23
1.5.3 Tecniche di regolarizzazione	24
1.6 Vantaggi e limiti	25
1.6.1 Flessibilità e capacità di generalizzazione	25
1.6.2 Problemi di vanishing/exploding gradient	26
1.6.3 Efficienza computazionale	26
2 Kolmogorov–Arnold Networks (KAN)	27
2.1 Introduzione	27

2.2	Fondamenti matematici	28
2.2.1	Enunciato del teorema di Kolmogorov–Arnold	28
2.2.2	Sulla natura "non costruttiva" delle dimostrazioni	29
2.3	Architettura delle KAN	30
2.3.1	Funzioni univariate parametriche	32
2.3.2	B-spline	33
2.3.3	Scaling laws e Curse of dimensionality	35
2.4	Funzionamento operativo	36
2.4.1	Calcolo del mapping input–hidden–output	36
2.4.2	Processo di training e Calcolo dei pesi	37
2.5	Confronto con MLP tradizionali	38
2.5.1	Architettura a confronto	38
2.5.2	Complessità computazionale	39
2.5.3	Interpretabilità e flessibilità locale	39
2.5.4	Precisione controllabile tramite grid extension	41
2.5.5	Dipendenza dalla struttura compositiva	41
2.5.6	Irregolarità nella rappresentazione di Kolmogorov	42
2.5.7	Overhead computazionale e scelta della struttura	42
2.5.8	Sensibilità al rumore e necessità di regolarizzazione	42
3	Random forest	43
3.1	Introduzione	43
3.2	Concetti fondamentali	44
3.2.1	Alberi di decisione: Criteri di splitting (Gini, Entropia, Gain ratio)	44
3.2.2	Ensemble learning e Bagging	46
3.3	Architettura e Costruzione	49
3.3.1	Bootstrapping e Feature bagging	49
3.3.2	Aggregazione delle predizioni	50
3.4	Vantaggi	51
3.5	Svantaggi	52
3.6	Vantaggi	52
3.7	Svantaggi	53

4 eXtreme Gradient Boosting (XGBoost)	55
4.1 Introduzione	55
4.2 Introduzione al Gradient boosting	56
4.2.1 Principi iterativi e Weak learners	57
4.2.2 Funzione di perdita e Discesa del gradiente	58
4.3 Miglioramenti di XGBoost rispetto al Gradient boosting tradizionale	59
4.3.1 Regolarizzazione e Tree pruning	59
4.3.2 Gestione dei valori mancanti	61
4.3.3 Ottimizzazioni (Parallelismo, Cache-awareness)	62
4.4 Parametri chiave e Tuning	63
4.5 Vantaggi	64
4.6 Svantaggi	65
5 Convolutional Neural Networks (CNN)	67
5.1 Introduzione	67
5.2 Principi fondamentali delle CNN	68
5.2.1 Convoluzione: kernel, stride, padding	68
5.2.2 Feature maps e profondità dei canali	69
5.2.3 Convoluzioni 1D, 2D e 3D	70
5.3 Pooling e normalizzazione	70
5.3.1 Max pooling vs average pooling	70
5.3.2 Global pooling	71
5.3.3 Batch, Layer e Group normalization	71
5.4 Data augmentation: rotazioni, zoom, colour jitter	72
5.5 Funzionamento delle CNN	73
5.5.1 Forward pass	73
5.5.2 Strati di convoluzione	73
5.5.3 Funzioni di attivazione	74
5.5.4 Strati di pooling	74
5.5.5 Gerarchia delle caratteristiche	74
5.5.6 Strati fully-connected	74
5.5.7 Flusso informativo e Trasformazioni progressive	75

5.6	Vantaggi	75
5.7	Svantaggi	76
6	Ottimizzazione degli iperparametri	78
6.1	Introduzione	78
6.2	Cross-Validation (CV)	79
6.2.1	Time Series Cross-Validation (TSCV)	80
6.3	Nested Cross-Validation (NCV)	81
6.4	Grid search (GS)	83
6.4.1	Spiegazione dell'algoritmo	83
6.4.2	Vantaggi	84
6.4.3	Limiti	84
6.5	Random Search (RS)	85
6.5.1	Spiegazione dell'algoritmo	85
6.5.2	Vantaggi	86
6.5.3	Limiti	86
6.6	Bayesian optimization (BO)	87
6.6.1	Spiegazione dell'algoritmo	87
6.6.2	Vantaggi	88
6.6.3	Limiti	89
6.7	Genetic algorithm (GA)	89
6.7.1	Spiegazione dell'algoritmo	90
6.7.2	Vantaggi	92
6.7.3	Limiti	92
6.8	Confronto pratico	92
6.8.1	Criteri per la scelta	92
6.8.2	Tabella riassuntiva comparativa	93
6.8.3	Matrice decisionale per la scelta del metodo	93
6.8.4	Scelta per i casi studio: Random search	93
6.8.5	Ottimizzazione del numero di iterazioni nel Random search	94

7 Studio di ablazione e Pruning post-training	96
7.1 Studi di ablazione	96
7.1.1 Definizione	96
7.1.2 Benefici	97
7.2 Pruning post-training	97
7.2.1 Definizione	97
7.2.2 Benefici	98
7.2.3 Trade-off bias-varianza	98
7.3 Pruning L1 post-training per MLP e KAN	99
7.3.1 Definizione	99
7.3.2 Considerazioni specifiche per le KAN	99
7.4 Pruning per Ensemble: Rank-based pruning per Random forest	100
7.4.1 Principio fondamentale	100
7.4.2 Criterio di ranking basato sulla feature importance .	100
7.4.3 Procedura di selezione	101
7.5 Pruning per Ensemble: Cumulative pruning per XGBoost .	101
7.5.1 Criterio di pruning cumulativo	101
7.5.2 Procedura di selezione	101
8 Primo Caso Studio: Regressione su emissioni di automobili	103
8.1 Progettazione del caso di studio	103
8.1.1 Tecnologie e librerie	103
8.1.2 Ambienti di sviluppo e infrastruttura	104
8.1.3 Linguaggi, automazione e pianificazione del workflow	104
8.1.4 Script di sottomissione (Cluster GPU)	104
8.1.5 Scelte architetturali ed iperparametri	105
8.2 Data preparation	106
8.2.1 Nota sull'origine e stato del dataset	106
8.2.2 Riassunto sintetico delle principali trasformazioni .	107
8.2.3 Ruolo nella tesi	107
8.3 Addestramento modelli	107
8.3.1 Features e variabile target	107

8.3.2	Pipeline di preprocessing	108
8.3.3	Split dei dati	108
8.3.4	Metriche e intervalli di confidenza	109
8.3.5	Strategia di training comune e griglie di iperparametri	109
8.4	Valutazione dei modelli	112
8.4.1	Selezione del miglior modello	113
8.5	Studio di ablazione	115
8.5.1	Ablation study: L1 pruning su MLP e KAN	115
8.5.2	Ablation study: ensemble pruning su Random Forest e XGBoost	117
8.5.3	Ablation study — Confronto complessivo (Neural Networks vs Ensemble)	119
9	Secondo Caso Studio: Classificazione di PM2.5	122
9.1	Data preparation	122
9.1.1	Fonti e descrizione generale del dataset	122
9.1.2	Caricamento dati e organizzazione iniziale	123
9.1.3	Indicizzazione temporale	123
9.1.4	Riduzione e unificazione di feature ridondanti	124
9.1.5	Verifica e gestione dei valori mancanti	126
9.1.6	Analisi esplorativa (EDA) e selezione delle feature rilevanti	127
9.1.7	Ricampionamento ed aggregazione a livello statale .	131
9.1.8	Rilevamento e rimozione degli outlier	132
9.1.9	Feature engineering ed arricchimento temporale . . .	133
9.1.10	Creazione di lag-features e categorizzazione di PM2.5	137
9.1.11	Categorizzazione PM2.5	137
9.2	Addestramento dei modelli	138
9.2.1	Pipeline di preprocessing	138
9.2.2	Split dei dati	139
9.2.3	Metriche e intervalli di confidenza	139
9.2.4	Random Forest	140
9.2.5	XGBoost	140

9.2.6	MLP	141
9.2.7	KAN	141
9.3	Valutazione dei modelli	142
9.3.1	Selezione del miglior modello	143
9.4	Studio di ablazione	146
9.4.1	Ablation study: L1 pruning su MLP e KAN	146
9.4.2	Ablation study: ensemble pruning su Random Forest e XGBoost	148
9.4.3	Ablation study — Confronto complessivo (Neural Networks vs Ensemble)	150
10	Terzo Caso Studio: Classificazione di età tramite immagini	154
11	Discussione comparativa sui Risultati dei casi studio	155
12	Conclusioni	156

Introduzione

La presente tesi si propone di analizzare, confrontare e valutare diverse architetture di Machine e Deep learning: le Kolmogorov–Arnold Networks (KAN), i Multi-Layer Perceptron (MLP), le Random forest e XGBoost.

L’obiettivo principale è quello di fornire, da un lato, un’analisi teorica esaustiva di ciascun modello, includendo i fondamenti matematici e le architetture; dall’altro, valutare sperimentalmente le prestazioni dei modelli su tre casi di studio: regressione sulle emissioni di automobili, classificazione dell’inquinamento atmosferico (PM2.5) e riconoscimento di immagini mediante CNN abbinate a MLP e KAN.

Oltre allo studio comparativo dei modelli, la tesi include un’ampia indagine sui metodi di Hyperparameter Tuning: Grid search, Random search, Bayesian optimization e Genetic algorithms, selezionati in base ad uno studio preliminare di confronto.

A completamento, sono stati condotti due studi di ablazione post-training per ogni caso di studio:

- **L1 Pruning** per KAN e MLP.
- **Ensemble Pruning** dove:
 - **Rank-Based Pruning** per Random forest.
 - **Cumulative Pruning** per XGBoost.

Capitolo 1

Multi-Layer Perceptron (MLP)

1.1 Introduzione

Questo capitolo presenta una descrizione del Multi-Layer Perceptron (MLP), un'architettura fondamentale delle reti neurali. L'obiettivo è esplorare la sua struttura e le sue capacità di modellare relazioni complesse, a partire dai principi matematici. Verranno analizzate le sue componenti principali, tra cui la distinzione tra gli strati e le funzioni di attivazione, che sono cruciali per l'apprendimento di relazioni non lineari. Il testo spiegherà poi il meccanismo attraverso cui la rete impara, descrivendo i passaggi di calcolo che avvengono durante il forward pass e l'algoritmo di backpropagation che aggiorna i parametri del modello. Verranno inoltre affrontate le tecniche di regolarizzazione, indispensabili per evitare che la rete memorizzi i dati di addestramento invece di imparare a generalizzare. Infine, il capitolo riassume i vantaggi ed i limiti delle MLP, ponendo l'accento sulla loro flessibilità e sui problemi legati alla profondità della rete, come il vanishing/exploding gradient.

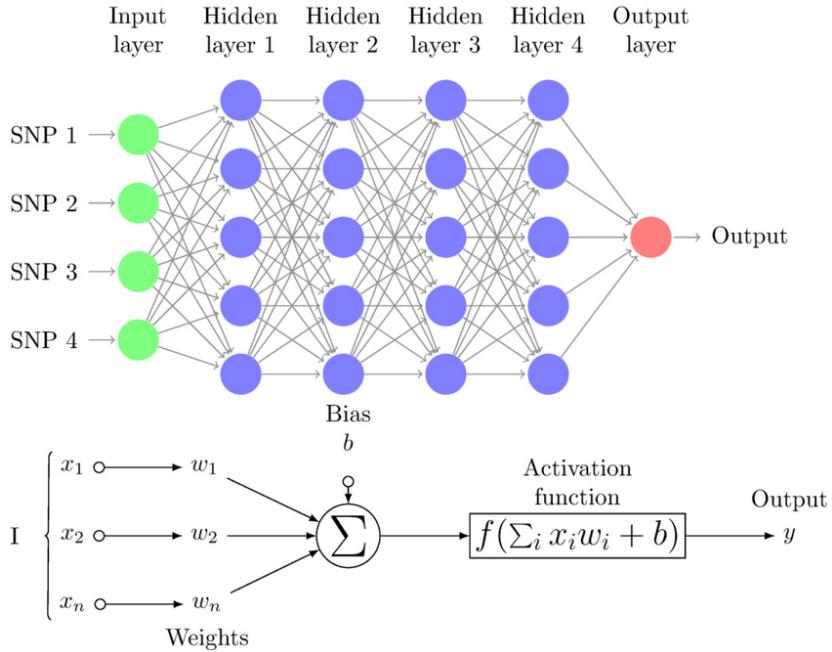


Figura 1.1: Multi-Layer Perceptron.

1.2 Fondamenti matematici

1.2.1 Enunciato formale

Sia $K \subset \mathbb{R}^n$ uno spazio compatto e sia $C(K)$ lo spazio delle funzioni continue su K munito della norma uniforme $\|\cdot\|_\infty$. Sia $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ una funzione di attivazione che soddisfa una delle seguenti ipotesi:

- (A₁) σ è continua e sigmoide, cioè $\lim_{t \rightarrow -\infty} \sigma(t) = a$ e $\lim_{t \rightarrow +\infty} \sigma(t) = b$ con $a \neq b$ (Cybenko);
- (A₂) σ è continua e non polinomiale (Leshno et al.).

Allora vale il seguente risultato di approssimazione universale.

1.2.2 Teorema di approssimazione universale

Per ogni $f \in C(K)$ e per ogni $\varepsilon > 0$, esistono un intero $N \in \mathbb{N}$ (indica il numero di neuroni nello strato nascosto), coefficienti scalari $c_i \in \mathbb{R}$, vettori

$w_i \in \mathbb{R}^n$ e bias $b_i \in \mathbb{R}$ tali che la funzione a singolo strato

$$\hat{f}_N(x) = \sum_{i=1}^N c_i \sigma(w_i \cdot x + b_i)$$

soddisfa

$$\|f - \hat{f}_N\|_\infty = \sup_{x \in K} |f(x) - \hat{f}_N(x)| < \varepsilon.$$

In altre parole, lo span delle funzioni elementari (cioè l'insieme di tutte le possibili combinazioni lineari di queste funzioni) $x \mapsto \sigma(w \cdot x + b)$ è denso in $C(K)$ rispetto alla norma uniforme. Ciò significa che per ogni funzione continua $f \in C(K)$ e per ogni $\varepsilon > 0$ esiste una combinazione lineare finita di blocchi attivati da σ (cioè una rete a singolo strato nascosto) che approssima f uniformemente su K con errore massimo minore di ε . Formalmente, la chiusura (nell' $\|\cdot\|_\infty$) dello spazio generato dalle funzioni elementari coincide con l'intero $C(K)$.

1.2.3 Significato di $\sup_{x \in K}$.

La notazione $\sup_{x \in K}$ denota l'estremo superiore di un insieme di reali. Nella norma uniforme

$$\|f - \hat{f}_N\|_\infty = \sup_{x \in K} |f(x) - \hat{f}_N(x)|$$

il valore indicato è l'errore massimo di approssimazione su tutto il dominio K . Poiché nel teorema K è assunto compatto e la funzione $x \mapsto |f(x) - \hat{f}_N(x)|$ è continua, l'estremo superiore coincide con il massimo:

$$\sup_{x \in K} |f(x) - \hat{f}_N(x)| = \max_{x \in K} |f(x) - \hat{f}_N(x)|.$$

Esempio Se $K = [0, 1]$ e $f(x) = \sin(2\pi x)$, affermare che esiste N tale che

$$\sup_{x \in [0,1]} |f(x) - \hat{f}_N(x)| < 0.01$$

significa che con quel numero di neuroni si può costruire \hat{f}_N che differisce dalla sinusoide al più di 0.01 in ogni punto dell'intervallo $[0, 1]$.

1.2.4 Ipotesi e Precisazioni

- **Compattezza del dominio K .** Il teorema è enunciato per funzioni continue definite su un insieme compatto $K \subset \mathbb{R}^n$ (ad es. l'intervallo chiuso $[0, 1]^n$). La compattezza garantisce che la norma uniforme $\|g\|_\infty = \sup_{x \in K} |g(x)|$ sia ben definita e che l'estremo superiore sia effettivamente un massimo raggiunto su K . Su domini non limitati (per es. \mathbb{R}^n) la formulazione uniforme non è direttamente applicabile.
- **Ipotesi sulla funzione di attivazione σ :** la validità del risultato dipende dalle proprietà di σ . Due formulazioni tipiche sono:
 - **Sigmoide limitata e continua (Cybenko):** dove σ ha limiti finiti agli estremi e cambia valore tra $-\infty$ e $+\infty$.
 - **Funzione continua non polinomiale (Leshno et al.):** condizione più generale che garantisce densità dello span.

Queste ipotesi escludono funzioni che non introducono la non linearità richiesta per generare uno spazio denso in $C(K)$. Per attivazioni moderne (ad esempio, ReLU) il teorema rimane valido ma con enunciati e ipotesi tecniche differenti.

- **Natura esistenziale del risultato:** il teorema è di tipo qualitativo: afferma che esiste un numero finito di neuroni N e parametri (c_i, w_i, b_i) tali che l'approssimazione uniforme è ottenuta entro qualsiasi tolleranza prefissata ε . Non fornisce però una procedura esplicita per la ricerca dei parametri ed un bound quantitativo generale che esprima N in funzione di ε per una data f .
- **Non implica una fase di training facile:** anche se esiste una rete che approssima f , nella pratica:

- gli algoritmi numerici di ottimizzazione (SGD, Adam, ecc.) non sono garantiti a trovare quei parametri ottimali: la funzione di perdita è non convessa e può avere molteplici ottimi locali o regioni piatte;
- la buona approssimazione teorica non assicura buona generalizzazione se i dati a disposizione sono scarsi: quindi è necessario usare tecniche di regolarizzazione, validazione e controllo dell'overfitting.

1.3 Struttura delle MLP

1.3.1 Strati: input, nascosti, output

Le Multi-layer Perceptron (MLP) sono una tipologia di reti neurali feed-forward, costituite da più strati (layer) di neuroni: uno strato di input, che riceve i dati iniziali; uno o più strati nascosti; ed uno strato di output che genera le previsioni finali. Ogni neurone, appartenente ad uno strato, è connesso a tutti i neuroni di quello successivo (architettura fully connected). Questo significa che ogni input viene trasformato dallo strato di input ai layer nascosti intermedi ed infine allo strato di output, con ogni collegamento caratterizzato da un peso w . Il numero di neuroni, in ciascun layer, è un iperparametro da scegliere: tipicamente lo strato di input ha tante unità quanti sono i parametri in ingresso, gli strati nascosti possono variare da pochi a molti nodi, a seconda del problema, e lo strato di output ha un neurone per ogni valore target.

In ogni neurone (esclusi quelli di input), si calcola una somma pesata degli input e di un termine di bias, per poi applicare una funzione di attivazione. L'output di un neurone i nel layer l è dato da:

$$z_i^{(l)} = \sum_{j=1}^{N_{l-1}} w_{ij}^{(l)} a_j^{(l-1)} + b_i^{(l)}$$

$$a_i^{(l)} = \sigma(z_i^{(l)})$$

Dove $z_i^{(l)}$ è la somma pesata, $w_{ij}^{(l)}$ è il peso della connessione, $a_j^{(l-1)}$ è l'output del neurone precedente, $b_i^{(l)}$ è il bias, e σ è la funzione di attivazione non lineare.

1.3.2 Funzioni di attivazione comuni

Le funzioni di attivazione sono cruciali nelle reti neurali, poiché introducono la non linearità necessaria per modellare relazioni complesse tra input e output. In assenza di tali funzioni, una rete multi-layer si ridurrebbe ad una trasformazione lineare. Di seguito vengono descritte alcune delle funzioni di attivazione più diffuse, con le loro proprietà matematiche, i pro e contro.

Proprietà rilevanti Quando si valuta una funzione di attivazione, conviene considerare la sua **differenziabilità**, che è fondamentale per la backpropagation; la **boundedness dell'output e zero-centering**, cioè se l'output è centrato attorno a 0; la **saturazione**, che può causare il *vanishing gradient*; la **sparsità** ed il **costo computazionale**.

1. Sigmoide

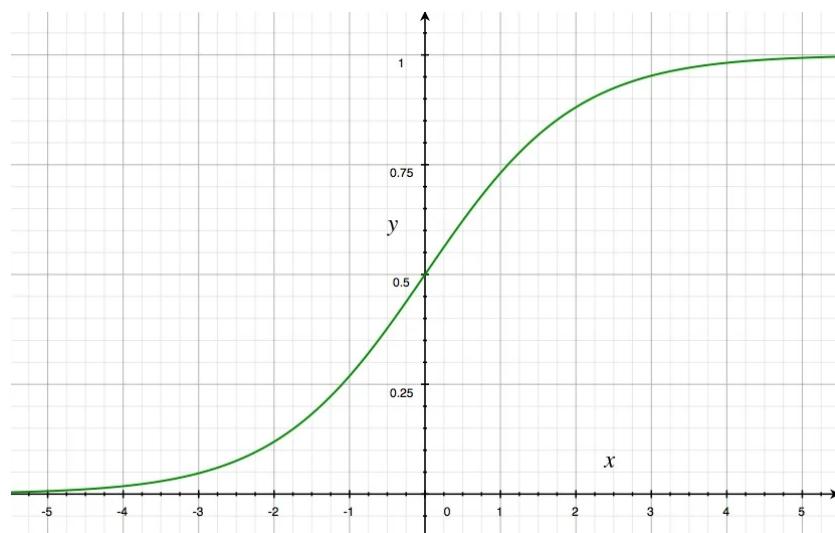


Figura 1.2: Grafico della funzione di attivazione Sigmoide

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \quad \sigma'(x) = \sigma(x)(1 - \sigma(x)).$$

Questa funzione ha un output che si colloca nell'intervallo $(0, 1)$ ed una caratteristica forma a "S", rendendola utile per rappresentare la probabilità (quindi un output normalizzato); la sua derivata è semplice da calcolare. Tuttavia, la sua principale debolezza è la saturazione per $x \rightarrow \pm\infty$, dove il gradiente tende a zero. Questo fenomeno porta al problema del vanishing gradient nei layer profondi. È tipicamente utilizzata nello strato di output per la classificazione binaria, in combinazione con la binary cross-entropy.

2. Tangente iperbolica (\tanh)

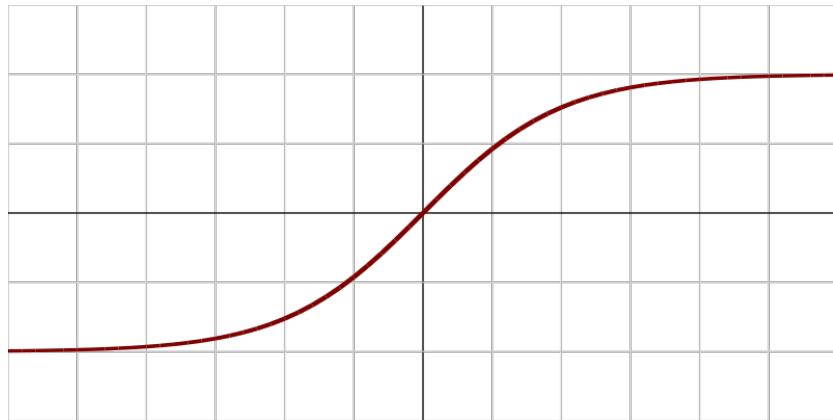


Figura 1.3: Grafico della funzione di attivazione Tangente iperbolica

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad \tanh'(x) = 1 - \tanh^2(x).$$

La funzione \tanh ha un output compreso tra $(-1, 1)$ ed è centrata in 0, il che, quando i dati sono normalizzati, porta ad una convergenza migliore rispetto alla sigmoide. Nonostante questo vantaggio, rimane una funzione saturante per valori estremi, e di conseguenza può soffrire ancora del problema del vanishing gradient. Il suo uso tipico è nei layer nascosti di reti poco profonde o quando è desiderabile un output centrato.

3. Rectified Linear Unit ReLU (ReLU)

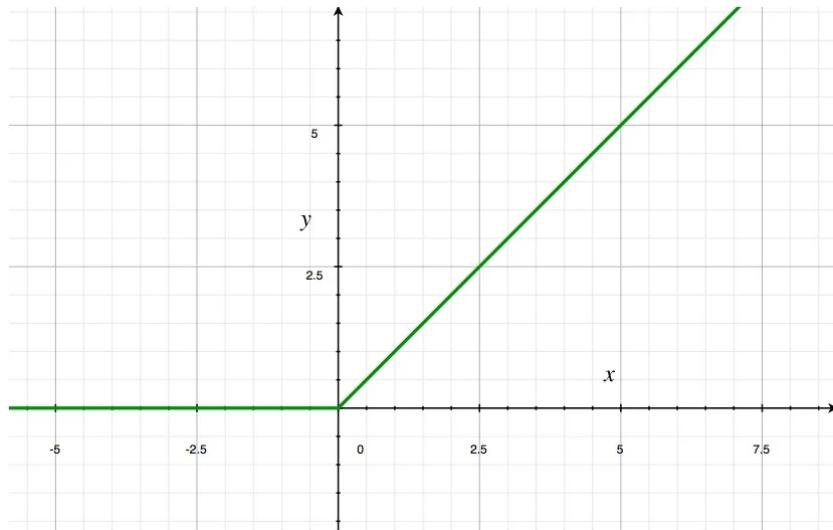


Figura 1.4: Grafico ReLU

$$\text{ReLU}(x) = \max(0, x), \quad \text{ReLU}'(x) = \begin{cases} 0 & x < 0, \\ 1 & x > 0, \end{cases}$$

La ReLU è una funzione semplice e computazionalmente efficiente, che, nella maggior parte dei casi, evita il problema del vanishing gradient sulle porzioni attive e favorisce la sparsità delle attivazioni. Il principale svantaggio è il problema della "dying ReLU": i neuroni possono diventare permanentemente inattivi se ricevono input negativi. Un neurone si considera "morto" se, per tutte (o quasi) le istanze del dataset, l'input $x \leq 0$, dato che in tal caso l'output è sempre nullo. Poiché la derivata di ReLU è zero per $x < 0$, il gradiente non si propaga a ritroso, e il neurone non riceve più aggiornamenti dei pesi, rimanendo inattivo per tutta la durata dell'allenamento. Viene ampiamente utilizzata nei layer nascosti in quasi tutte le architetture di reti neurali.

4. Leaky ReLU (LReLU) / Parametric ReLU (PReLU)

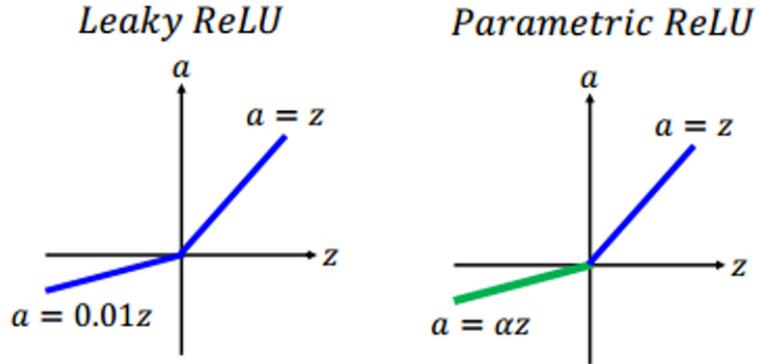


Figura 1.5: Grafico LReLU vs PReLU

$$\begin{aligned} \text{LReLU}(x) &= \begin{cases} 0.01x & x \leq 0, \\ x & x > 0, \end{cases} & \text{LReLU}'(x) &= \begin{cases} 0.01 & x < 0, \\ 1 & x > 0, \end{cases} \\ \text{PReLU}(x) &= \begin{cases} \alpha x & x < 0, \\ x & x \geq 0, \end{cases} & \text{PReLU}'(x) &= \begin{cases} \alpha & x < 0, \\ 1 & x > 0, \end{cases} \quad \alpha \in (0, 1) \end{aligned}$$

PReLU apprende il parametro α durante il training. Entrambe le varianti mantengono un piccolo gradiente per $x < 0$, riducendo significativamente il problema dei "dead neurons". L'introduzione (o l'apprendimento) di un iperparametro è un potenziale svantaggio, ed il loro comportamento non è sempre superiore a quello della semplice ReLU. Sono impiegate dove si vuole evitare il problema della "dying ReLU", mantenendo la semplicità.

5. Exponential Linear Unit (ELU)

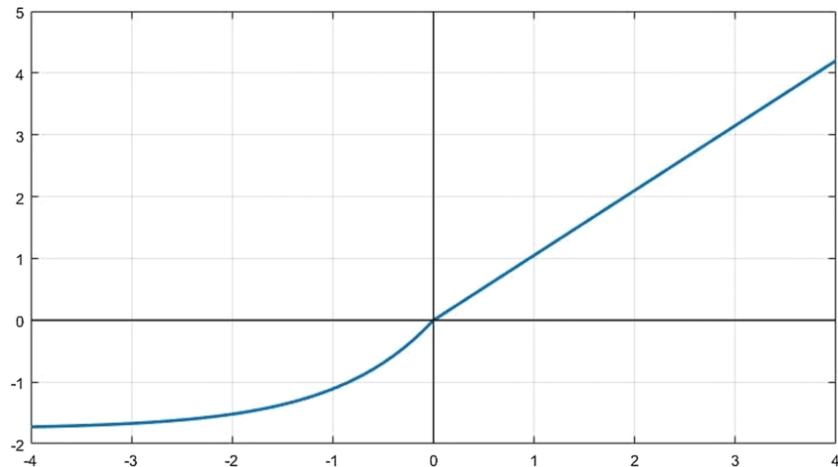


Figura 1.6: Grafico ELU

$$\text{ELU}(x) = \begin{cases} x & x \geq 0, \\ \alpha(e^x - 1) & x < 0, \end{cases} \quad \text{ELU}'(x) = \begin{cases} \alpha(e^x) & x < 0, \\ 1 & x > 0, \end{cases} \quad \alpha > 0$$

L'ELU produce un output più centrato rispetto alla ReLU e ha un gradiente non nullo per $x < 0$, che in alcuni casi può portare a una migliore convergenza. Tuttavia, è leggermente più costosa a livello computazionale a causa della funzione esponenziale ed introduce un parametro α da ottimizzare.

6. Softplus

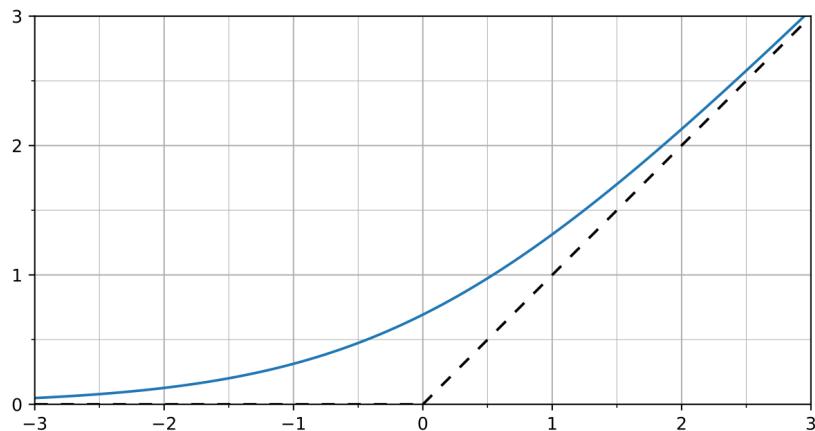


Figura 1.7: Grafico Softplus

$$\text{softplus}(x) = \log(1 + e^x), \quad \text{softplus}'(x) = \frac{1}{1 + e^{-x}}$$

La Softplus è una versione continua e completamente differenziabile della ReLU. Nonostante questa proprietà, è più costosa dal punto di vista computazionale e meno sparsa rispetto alla ReLU.

7. Gaussian Error Linear Unit (GELU)

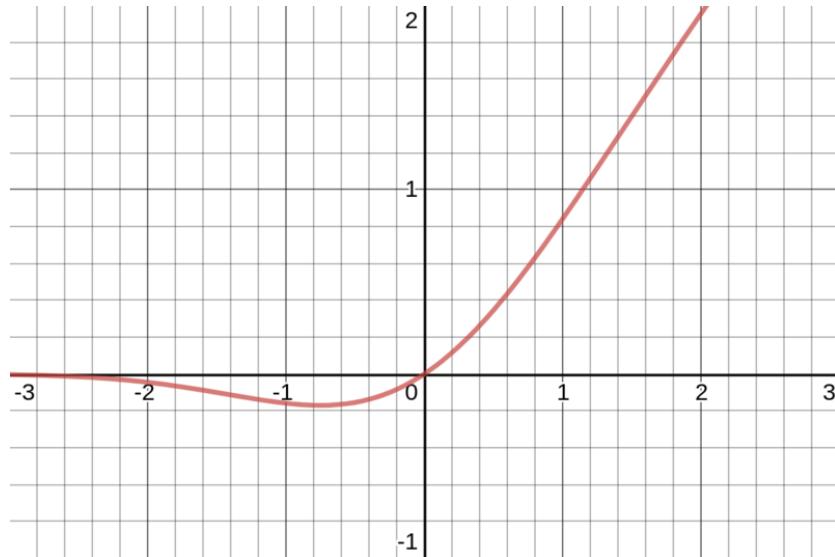


Figura 1.8: Grafico GELU

$$\text{GELU}(x) = x \cdot \Phi(x)$$

dove $\Phi(x)$ è la funzione di distribuzione cumulativa (CDF) della distribuzione normale standard e 'erf' è la funzione degli errori di Gauss:

$$\Phi(x) = \frac{1}{2} \left[1 + \text{erf} \left(\frac{x}{\sqrt{2}} \right) \right]$$

$$\text{GELU}'(x) = \Phi(x) + \frac{1}{2} x \phi(x)$$

dove

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

La GELU ha dimostrato un eccellente comportamento empirico, specialmente nei modelli di linguaggio, ed è considerata una funzione di attivazione "soft" che combina linearità e gating stocastico. Il suo principale svantaggio è il costo computazionale più elevato. Viene ampiamente utilizzata nelle architetture Transformer.

8. Softmax

Per un vettore $x \in \mathbb{R}^K$:

$$\text{softmax}(x)_j = \frac{e^{x_j}}{\sum_{k=1}^K e^{x_k}}.$$

Questa funzione viene utilizzata per trasformare i logits (gli output grezzi di un layer) in una distribuzione di probabilità. È normalmente combinata con la funzione di perdita di categorical cross-entropy per problemi di classificazione multi-classe.

1.4 Procedura di forward pass

1.4.1 Calcolo delle attivazioni

Durante la fase di propagazione in avanti (forward pass), i dati attraversano la rete dallo strato di input a quello di output. Ogni neurone calcola prima un ingresso pesato sommandolo con il bias. Dato un neurone j dello strato nascosto o di output, l'eccitazione, o attivazione lineare, è:

$$z_j = \sum_i w_{ji}x_i + b_j,$$

dove x_i sono gli output (o input iniziali), w_{ji} i pesi di connessione, e b_j il bias. In seconda battuta, si applica la funzione di attivazione ϕ per ottenere l'uscita del neurone:

$$a_j = \phi(z_j).$$

Ad esempio, con $\phi = \sigma$ (sigmoid), avremmo $a_j = 1/(1 + e^{-z_j})$. Questo processo viene eseguito strato per strato. Ogni layer trasforma in modo non lineare i dati in ingresso, permettendo alla rete di apprendere composizioni funzionali complesse.

1.4.2 Propagazione ed output

Dopo aver calcolato le attivazioni in tutti gli strati intermedi, l'uscita dello strato finale (\mathbf{a}_{out}) costituisce la previsione della rete. Se il problema è di regressione, l'ultima funzione di attivazione può essere identità (o lineare); se è di classificazione binaria, si può usare la sigmoid; se è multi-classe, si usa tipicamente la softmax. Ad esempio, in una classificazione a K classi lo strato di output contiene K neuroni con softmax, e ciascuna uscita $a_k \in (0, 1)$ rappresenta la probabilità assegnata alla classe k . L'output finale è dunque un vettore di predizioni che dipende dalle scelte di pesi, bias e funzioni di attivazione attraverso la rete. Infine, confrontando \mathbf{a}_{out} con il valore target (ground truth) del training si calcola una funzione di perdita che misura l'errore di previsione (ad esempio, MSE per regressione o cross-entropy per classificazione). Questa funzione di perdita viene poi utilizzata nell'allenamento per aggiornare i pesi tramite backpropagation.

1.5 Algoritmo di backpropagation

L'algoritmo di backpropagation è fondamentale per l'addestramento delle reti neurali, poiché calcola come i pesi della rete devono essere modificati per minimizzare l'errore. Questo processo si basa sull'applicazione della regola della catena (chain rule) per derivare il gradiente della funzione di perdita L rispetto a ogni peso w .

1.5.1 Derivazione del gradiente

Il processo inizia con la definizione dell'errore, calcolato attraverso una funzione di perdita L , che misura la differenza tra il valore previsto dalla rete e il valore target effettivo. Ad esempio, per la regressione si può usare l'errore quadratico medio (MSE), mentre per la classificazione si ricorre spesso alla cross-entropy.

Una volta definito l'errore, l'algoritmo procede calcolando la derivata parziale della funzione di perdita L rispetto all'attivazione netta di ciascun

neurone. Questo valore è noto come "errore locale" o δ_j per il neurone j , ed è una misura di quanto l'errore di output sia influenzato dall'input di quel neurone prima dell'applicazione della funzione di attivazione.

$$\delta_j = \frac{\partial L}{\partial z_j}.$$

Per un neurone nello strato di output, questo calcolo è diretto, ma per i neuroni nei layer nascosti, δ_j viene determinato propagando a ritroso gli errori dei neuroni successivi. In pratica, ogni neurone riceve contributi di errore da tutti i neuroni a cui è connesso nel layer seguente. Questo meccanismo di propagazione a ritroso è il cuore della backpropagation. L'applicazione della chain rule permette di ottenere il gradiente di L rispetto ad ogni peso w_{ji} che connette il neurone i al neurone j :

$$\frac{\partial L}{\partial w_{ji}} = a_i \delta_j,$$

dove a_i è l'output del neurone i nel layer precedente e δ_j è l'errore locale del neurone j . In parole semplici, l'algoritmo calcola l'errore di output e lo distribuisce a ritroso attraverso i layer, moltiplicandolo per le derivate delle funzioni di attivazione. Si ottiene così un vettore di gradienti che indica la direzione in cui i pesi devono essere aggiornati per ridurre l'errore complessivo.

1.5.2 Aggiornamento dei pesi

Una volta noti i gradienti parziali $\frac{\partial L}{\partial w}$, i pesi vengono aggiornati solitamente tramite discesa del gradiente (gradient descent). Con un learning rate η , l'aggiornamento è dato da:

$$w \leftarrow w - \eta \frac{\partial L}{\partial w}.$$

Questo modifica ogni peso nella direzione negativa del gradiente per ridurre la perdita. In termini di formula, per l'esempio di un singolo campione e

peso w_{ji} :

$$\Delta w_{ji} = -\eta \frac{\partial L}{\partial w_{ji}} = -\eta \delta_j a_i.$$

Questa è la regola standard di backpropagation con discesa del gradiente. In pratica, si iterano più epoch di allenamento aggiornando i pesi in base a molti esempi, eventualmente con varianti come la discesa del gradiente stocastico (SGD) oppure con algoritmi avanzati (momentum, Adam, ecc.). Tra un passo di forward e il successivo di backward, possono essere applicate tecniche di batching: l'errore può essere aggregato su minibatch di esempi per stabilizzare l'aggiornamento. In ogni caso, il principio fondamentale è che i pesi vengono "aggiustati" proporzionalmente al proprio contributo all'errore complessivo, come descritto nelle sezioni precedenti.

1.5.3 Tecniche di regolarizzazione

Affinché la rete abbia un'ottima capacità di generalizzazione (non si limiti a riprodurre il rumore dei dati di training), si usano tecniche di regolarizzazione:

- **Dropout:** durante la fase di training, in ciascuna iterazione si disattiva casualmente una parte di neuroni in alcuni layer, impostando le loro attivazioni a zero. Questo costringe la rete a non fare affidamento eccessivo su singoli neuroni, favorendo lo sviluppo di rappresentazioni ridondanti e riducendo l'overfitting. Ad esempio, con una dropout probability p , un neurone viene disattivato con probabilità p e gli altri sono scalati di $1/(1-p)$ per compensazione.
- **Regolarizzazione L1 (LASSO):** si aggiunge alla loss un termine proporzionale alla somma dei valori assoluti dei pesi:

$$L'(w) = L(w) + \lambda \|w\|_1 = L(w) + \lambda \sum_i |w_i|,$$

con $\lambda > 0$. La funzione di penalità ℓ_1 induce sparsità: molte componenti dei pesi vengono impostate a zero, facilitando la selezione di feature e l'interpretabilità del modello.

- **Regolarizzazione L2 (Ridge):** si aggiunge il quadrato della norma dei pesi:

$$L'(w) = L(w) + \frac{\lambda}{2} \|w\|_2^2 = L(w) + \frac{\lambda}{2} \sum_i w_i^2.$$

Il termine ℓ_2 non produce soluzioni esattamente sparse ma riduce la magnitudine di tutti i pesi verso lo zero.

- **Combinazione L1+L2 (Elastic Net):** combina entrambe le precedenti penalità:

$$L'(w) = L(w) + \alpha \left(\lambda_1 \|w\|_1 + \frac{\lambda_2}{2} \|w\|_2^2 \right),$$

e viene scelta per ottenere sia sparsità (L1) sia stabilità (L2) quando le features sono correlate. Elastic Net è spesso preferibile quando il numero di variabili supera il numero di osservazioni o quando ci sono gruppi di variabili fortemente correlate.

1.6 Vantaggi e limiti

1.6.1 Flessibilità e capacità di generalizzazione

Le reti MLP offrono grande flessibilità: grazie alla combinazione di pesi e attivazioni non lineari, possono modellare relazioni complesse e non lineari tra input e output. Possono apprendere sia compiti di regressione che di classificazione (binarie o multi-classe) e sono in grado di approssimare praticamente qualsiasi funzione continua. Tale capacità di rappresentazione rende le MLP potenti modelli predittivi in molti ambiti. Inoltre, in presenza di dati adeguati e con l'uso di tecniche di regolarizzazione, le MLP tendono ad avere una buona capacità di generalizzazione, ossia sono in grado di fare previsioni corrette su dati non visti.

1.6.2 Problemi di vanishing/exploding gradient

Uno dei limiti più importanti delle MLP (soprattutto se possiedono molti hidden layer) riguarda il problema del vanishing gradient. Poiché, durante la backpropagation, i gradienti vengono moltiplicati per le derivate delle funzioni di attivazione in ogni layer, se queste derivate sono piccole (come in sigmoid o tanh, che assumono valori entro $(0, 1)$), il prodotto dei gradienti tende a diminuire esponenzialmente con la profondità. Di conseguenza, i pesi nei primi strati (vicini all'input) ricevono gradienti quasi nulli e la rete impara molto lentamente le rappresentazioni nei layer bassi. Al contrario, se derivate o pesi sono grandi (> 1), può manifestarsi un exploding gradient, dove i gradienti crescono esponenzialmente e portano ad instabilità numeriche (pesanti oscillazioni o overflow). Per questo si usano funzioni, come le ReLU, che hanno derivate più stabili, normalizzazione dei dati, inizializzazioni specifiche dei pesi, o architetture speciali per alleviare il fenomeno.

1.6.3 Efficienza computazionale

Dal punto di vista computazionale, le MLP possono diventare costose da addestrare se il numero di layer o di neuroni è elevato. Ogni propagazione in avanti ed indietro richiede calcoli intensivi e su set di dati di grandi dimensioni l'allenamento può richiedere molto tempo e risorse computazionali. Il costo cresce con il numero di connessioni del modello. Inoltre, le MLP richiedono un tuning accurato degli iperparametri (learning rate, struttura della rete, regolarizzazione, ecc.) per ottimizzare performance ed efficienza.

Capitolo 2

Kolmogorov–Arnold Networks (KAN)

2.1 Introduzione

Il presente capitolo introduce le Kolmogorov–Arnold Networks (KAN), una nuova classe di reti neurali che si ispira direttamente al teorema di approssimazione di Kolmogorov-Arnold. L'obiettivo è esplorare la loro architettura innovativa, che differisce dalle tradizionali reti MLP nel modo in cui gestiscono le funzioni di attivazione. Verranno esaminati i fondamenti matematici che giustificano la loro efficacia, in particolare come le KAN si propongono di superare la "maledizione della dimensionalità" (curse of dimensionality) grazie alla loro capacità di approssimare funzioni complesse. Il capitolo si concentra sul funzionamento operativo, descrivendo come le funzioni parametriche basate su B-spline sostituiscono i pesi scalari e le attivazioni fisse degli MLP. Infine, verranno discussi in dettaglio i vantaggi e gli svantaggi delle KAN, come la loro interpretabilità e maggiore flessibilità locale rispetto agli MLP, ma anche la loro maggiore complessità computazionale e la dipendenza dalla struttura del problema.

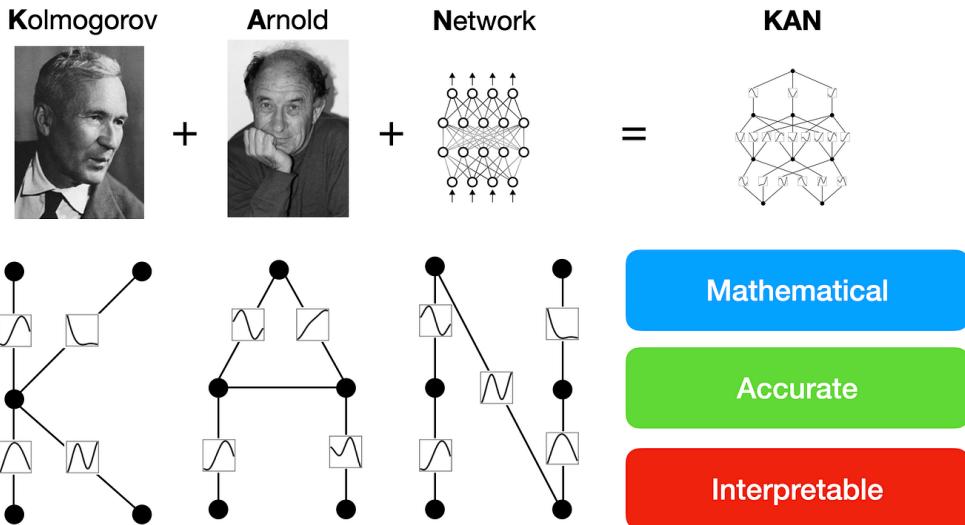


Figura 2.1: Kolmogorov–Arnold Networks

2.2 Fondamenti matematici

2.2.1 Enunciato del teorema di Kolmogorov–Arnold

Il teorema di Kolmogorov–Arnold (KART) stabilisce che ogni funzione continua multivariata (cioè su più variabili), su un intervallo compatto, può essere espressa come una combinazione di somme di funzioni univariate (cioè su una variabile). In forma esplicita, per una funzione continua $f : [0, 1]^n \rightarrow \mathbb{R}$ esistono funzioni continue univariate $\phi_{q,p}$ e Φ_q tali che

$$f(x_1, \dots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^n \phi_{q,p}(x_p) \right),$$

per $q = 1, \dots, 2n + 1$.

In altre parole, ogni funzione multivariata può essere ricondotta a combinazioni di funzioni unidimensionali tramite opportune funzioni interne $\phi_{q,p}$, che agiscono come "feature extractors" individuali, cioè estraggono informazioni rilevanti da ciascuna dimensione dei dati in modo indipendente,

ed esterne Φ_q , che agiscono come un "classificatore" di queste features, cioè combinano le caratteristiche estratte per prendere una decisione. Questo risultato afferma che qualsiasi funzione continua di più variabili può essere completamente rappresentata tramite combinazioni di funzioni continue di una sola variabile.

2.2.2 Sulla natura "non costruttiva" delle dimostrazioni

Le dimostrazioni originali del KART, eseguite da Kolmogorov nel 1957 e successivamente Arnold nel 1967, sono di natura esistenziale: garantiscono l'esistenza delle funzioni $\phi_{q,p}$ e Φ_q , ma non forniscono una procedura esplicita o una formula chiusa per costruirle. Pertanto il teorema è fondamentale dal punto di vista teorico ma, senza ulteriori risultati costruttivi, ha un'utilità pratica limitata per la costruzione diretta di architetture neurali basate su tali funzioni, spingendo i ricercatori a privilegiare le reti neurali multistrato (MLP) che, pur con i loro limiti, erano più facili da implementare.

2.3 Architettura delle KAN

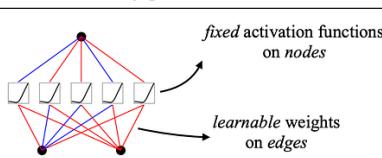
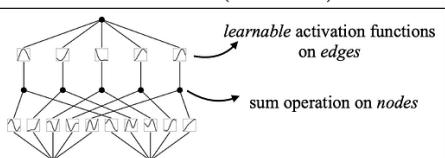
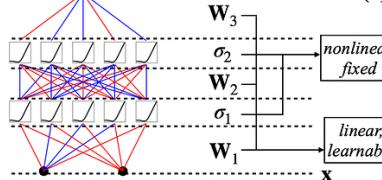
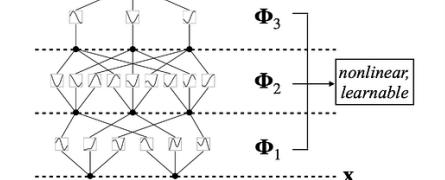
Model	Multi-Layer Perceptron (MLP)	Kolmogorov-Arnold Network (KAN)
Theorem	Universal Approximation Theorem	Kolmogorov-Arnold Representation Theorem
Formula (Shallow)	$f(\mathbf{x}) \approx \sum_{i=1}^{N(e)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^n \phi_{q,p}(x_p) \right)$
Model (Shallow)	(a) 	(b) 
Formula (Deep)	$\text{MLP}(\mathbf{x}) = (\mathbf{W}_3 \circ \sigma_2 \circ \mathbf{W}_2 \circ \sigma_1 \circ \mathbf{W}_1)(\mathbf{x})$	$\text{KAN}(\mathbf{x}) = (\Phi_3 \circ \Phi_2 \circ \Phi_1)(\mathbf{x})$
Model (Deep)	(c) 	(d) 

Figura 2.2: Confronto tra un Multi-Layer Perceptron (MLP) ed una Kolmogorov–Arnold Network (KAN).

Una Kolmogorov–Arnold Network (KAN) è strutturalmente simile ad una rete feedforward completamente connessa, simile ad una MLP, ma differisce in modo sostanziale nell’uso delle funzioni di attivazione: ogni arco (collegamento) tra i neuroni di strati consecutivi porta con sé una funzione univariata parametricamente definita (spesso una B-spline), anziché un peso scalare come nelle MLP. Ciascun neurone di uno strato riceve gli output dei collegamenti in ingresso e calcola semplicemente la somma di tali output, senza l’uso di pesi lineari o di funzioni di attivazione non lineari applicate ai singoli nodi stessi.

Il modello generale si descrive così: se lo strato $(\ell - 1)$ ha $d_{\ell-1}$ neuroni e lo strato ℓ ne ha d_ℓ , allora esiste una matrice di funzioni unidimensionali $\{f_{ij}^{(\ell)}\}_{i=1,\dots,d_{\ell-1}, j=1,\dots,d_\ell}^{j=1,\dots,d_\ell}$ tale che, dati gli output degli $d_{\ell-1}$ neuroni precedenti $x_i^{(\ell-1)}$,

l'uscita $x_j^{(\ell)}$ del j -esimo neurone del livello ℓ è:

$$x_j^{(\ell)} = \sum_{i=1}^{d_{\ell-1}} f_{ij}^{(\ell)}(x_i^{(\ell-1)}) .$$

In forma matriciale si può scrivere $x^{(\ell)} = f^{(\ell)}(x^{(\ell-1)})$, dove $f^{(\ell)}$ è l'insieme delle funzioni collegate a quello strato. L'output complessivo della KAN è quindi dato dalla composizione degli strati successivi:

$$y = x^{(L)} = f^{(L)}(f^{(L-1)}(\dots f^{(1)}(x^{(0)}) \dots)),$$

dove $x^{(0)}$ è il vettore di input della rete.

Questa architettura combina, in un'unica funzione f_{ij} per ogni arco, le trasformazioni lineari e non lineari, permettendo alla rete di apprendere la forma esatta della funzione di attivazione necessaria per ogni arco di connessione.

Si noti che un MLP applica funzioni di attivazione predefinite (ReLU, sigmoid, ecc.) sui singoli neuroni e moltiplica gli input per pesi scalari; al contrario, una KAN utilizza funzioni parametriche sugli archi e non applica ulteriori attivazioni non lineari sui neuroni.

2.3.1 Funzioni univariate parametriche

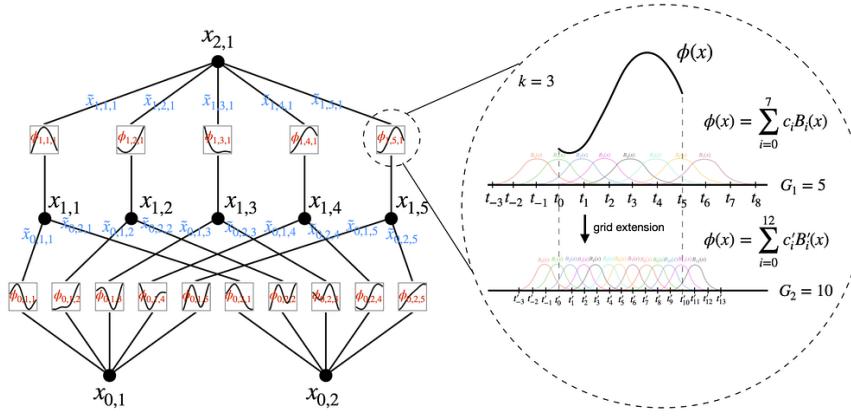


Figura 2.3: Funzioni univariate parametriche.

Le funzioni di attivazione utilizzate nelle KAN sono funzioni univariate parametriche che possono apprendere flessibilmente la forma durante il training. Nell'implementazione classica proposta da Liu et al. (2024), queste funzioni sono rappresentate tramite B-spline (polinomi a tratti di basso grado), che offrono un buon trade-off tra flessibilità locale e complessità di calcolo.

Ciascuna funzione di attivazione su un collegamento in una KAN è quindi espressa nella forma

$$f_{ij}(t) = t + g_{ij}(t),$$

dove $g_{ij}(t)$ è una combinazione lineare di B-spline:

$$g_{ij}(t) = \sum_{k=1}^{G+p} c_k B_{k,p} \left(\frac{t - t_{\min}}{t_{\max} - t_{\min}} \right).$$

Il termine lineare t garantisce un comportamento affine iniziale, migliorando la stabilità dell'ottimizzazione durante il training, mentre il contributo spline introduce la non linearità appresa dalla rete, permettendo di model-

lare funzioni di attivazione flessibili e adattabili.

Un aspetto importante è la definizione della spline grid, ovvero la suddivisione dell'asse degli input in nodi che delimitano gli intervalli su cui sono definiti i singoli segmenti delle B-spline. Il numero di nodi G e la loro posizione influenzano la capacità espressiva e la risoluzione locale delle funzioni attivazione. In generale, oltre alle B-spline, si possono utilizzare anche altre famiglie di funzioni unidimensionali parametriche, come i polinomi di Chebyshev o altre basi ortogonali, in base alle esigenze specifiche del problema. Infatti, le reti KAN sono in grado di adattare i coefficienti di queste basi durante il training, permettendo alla rete di apprendere funzioni di attivazione ottimali per ciascun collegamento.

2.3.2 B-spline

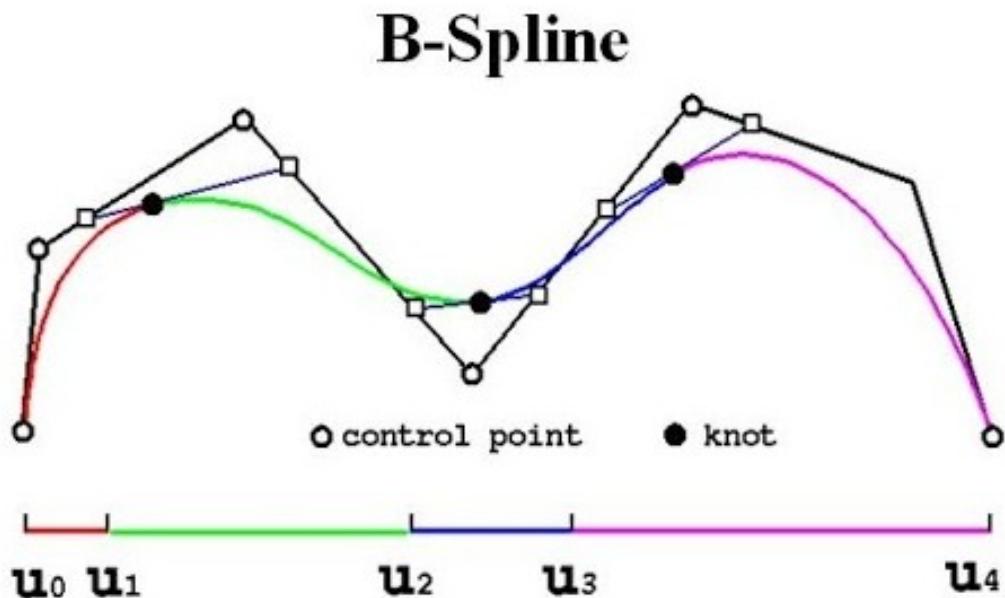


Figura 2.4: Struttura B-spline.

Definizione

Le B-spline sono funzioni polinomiali a tratti che costituiscono la base per la rappresentazione di funzioni spline di un dato grado. Una B-spline di ordine $p + 1$ (ovvero di grado p) è definita ricorsivamente come segue:

$$B_{i,0}(t) = \begin{cases} 1 & \text{se } t_i \leq t < t_{i+1} \\ 0 & \text{altrimenti} \end{cases}$$

e per $p > 0$:

$$B_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} B_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} B_{i+1,p-1}(t),$$

dove $\{t_i\}$ è il vettore dei nodi (knot vector), che suddivide il dominio della funzione in intervalli. Ogni B-spline $B_{i,p}(t)$ è diversa da zero solo sull'intervallo $[t_i, t_{i+p+1}]$, conferendo proprietà di supporto locale.

Implementazione delle B-spline nelle KAN

Nelle implementazioni standard delle KAN, le funzioni di attivazione sui collegamenti sono parametrizzate come combinazioni lineari di B-spline cubiche ($p = 3$):

$$f_{ij}(x) = w_0 x + \sum_{k=1}^{G+3} c_k B_{k,3} \left(\frac{x - x_{\min}}{x_{\max} - x_{\min}} \right),$$

dove:

- w_0 è il termine residuo lineare che garantisce un comportamento iniziale affine,
- $\{c_k\}$ sono i coefficienti addestrabili della spline,
- G è il numero di intervalli della griglia di nodi,
- $B_{k,3}$ sono le funzioni base B-spline di grado p .

Il termine residuo lineare w_0x stabilizza l'ottimizzazione, permettendo alla funzione di partire come lineare e apprendere non linearità tramite la componente spline.

Adattamento della griglia

Un aspetto importante delle KAN è l'adattamento dinamico della spline grid. Inizialmente, la griglia è generalmente uniforme, ma può essere ampliata aumentando il numero di nodi. Durante il training, i nodi possono essere riposizionati strategicamente per concentrare la risoluzione nelle aree in cui la funzione varia maggiormente, migliorando così la capacità di modellazione locale della rete. Per mantenere la stabilità durante queste modifiche alla griglia, si utilizzano tecniche di interpolazione lineare sui parametri dell'ottimizzatore, garantendo transizioni fluide e controllate nella definizione della griglia.

2.3.3 Scaling laws e Curse of dimensionality

Come suggerito dal teorema matematico KART, le KAN godono di proprietà di approssimazione analoghe ma più raffinate rispetto alle MLP. In particolare, il teorema di Kolmogorov-Arnold garantisce che ogni funzione continua multivariata definita su un dominio compatto possa essere rappresentata esattamente come composizione di funzioni univariate e somme, realizzabile da una KAN con 2 strati e larghezza proporzionale all'input n (in particolare, larghezza $2n + 1$). Di conseguenza, per ogni tolleranza $\epsilon > 0$ esiste una KAN sufficientemente ampia che approssima la funzione f entro errore ϵ , cioè

$$\|f_{\text{KAN}} - f\| < \epsilon.$$

Questa capacità permette alle KAN di superare il problema noto come curse of dimensionality, a condizione che la funzione da approssimare possieda una struttura additivo-composizionale sufficientemente regolare. In particolare, l'errore di approssimazione di una KAN dipende principalmente dalla risoluzione della griglia spline utilizzata nelle funzioni univariate,

risultando approssimativamente indipendente dalla dimensione dell'input. Questa proprietà si traduce in scaling laws più favorevoli rispetto alle MLP tradizionali, per le quali il numero di parametri necessari per garantire una certa accuratezza generalmente cresce esponenzialmente con la dimensione dell'input. Il vantaggio teorico delle KAN deriva dal fatto che esse, a differenza delle MLP, apprendono non solo la struttura compositiva della funzione, ma sono anche in grado di modellare con elevata precisione le funzioni univariate interne, grazie all'uso di attivazioni parametriche basate su spline.

2.4 Funzionamento operativo

2.4.1 Calcolo del mapping input–hidden–output

Nel funzionamento operativo di una KAN, i dati scorrono in avanti attraverso gli strati esattamente come in una normale MLP, ma usando le funzioni di attivazione sugli archi. Dato un vettore di input $x^{(0)}$, il calcolo procede layer dopo layer: per ciascun neurone nel primo strato nascosto si valuta la somma delle funzioni dei collegamenti in ingresso applicate alle componenti di $x^{(0)}$, ottenendo $x^{(1)}$ e così via. Al livello successivo si ripete lo stesso procedimento prendendo $x^{(1)}$ come input, e così via fino allo strato di output. Formalmente, ogni layer ℓ esegue la trasformazione $x^{(\ell)} = f^{(\ell)}(x^{(\ell-1)})$, e l'output finale è $y = x^{(L)}$.

Poiché tutte le operazioni, cioè l'applicazione delle funzioni univariate e le somme, sono differenziabili, l'intero modello è addestrabile tramite backpropagation. Questo significa che i coefficienti delle funzioni di attivazione (ad esempio, delle spline) possono essere ottimizzati tramite discesa del gradiente, minimizzando una funzione di perdita, allo stesso modo di quanto avviene in un MLP.

2.4.2 Processo di training e Calcolo dei pesi

Il training di una KAN segue la procedura standard di addestramento supervisionato con discesa del gradiente. Si parte da un dataset di training ed una funzione loss, quindi si aggiornano iterativamente i parametri delle funzioni di attivazione. Nelle KAN, i "pesi" da addestrare sono i coefficienti che definiscono le funzioni unidimensionali sui collegamenti. Per esempio, una spline di ordine 3 su r intervalli ha $r + 3$ coefficienti; ciascun coefficiente è un parametro della rete. È comune includere un termine base lineare (di solito la stessa identità), come in

$$f_{ij}(t) = t + g_{ij}(t),$$

dove g_{ij} è la spline addestrabile. Questo facilita la convergenza iniziale. Durante l'ottimizzazione si può anche aggiornare adattivamente la griglia di definizione delle spline, così da coprire automaticamente i nuovi intervalli di attivazione che emergono durante il training (grid extension). In pratica, ogni volta che un valore di attivazione supera la griglia corrente, si estende dinamicamente il supporto della spline per mantenere il dominio di apprendimento adeguato. In sintesi, il flusso di calcolo è identico ad un MLP: si effettua forward pass, si calcola la loss, e poi si retropropaga l'errore calcolando gradienti rispetto ai coefficienti delle funzioni g_{ij} .

2.5 Confronto con MLP tradizionali

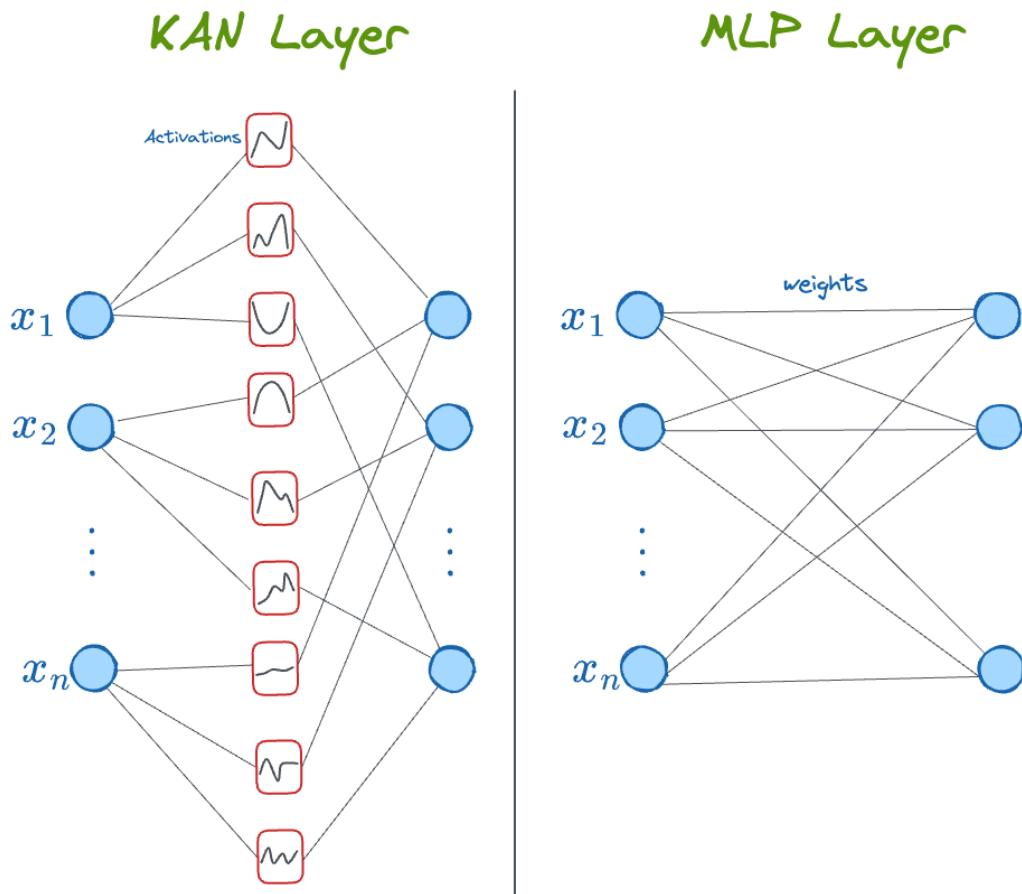


Figura 2.5: Differenze tra Layer MLP e KAN.

2.5.1 Architettura a confronto

Dal punto di vista architettonale, l'architettura di base di una KAN è feedforward e totalmente connessa come quella di un MLP. La differenza fondamentale risiede nel collocamento delle non-linearietà e nell'assenza di matrici di pesi lineari. Come abbiamo visto, in un MLP ogni neurone applica una funzione di attivazione fissa, dopo una combinazione lineare dei suoi input, mentre in una KAN ogni collegamento possiede direttamente una funzione di attivazione addestrabile. Di conseguenza, una KAN "unisce" le trasformazioni lineari e non-lineari in un'unica funzione f_{ij} per ogni arco,

anziché trattarle separatamente come in un MLP.

In termini pratici, una MLP a L strati alterna operazioni $x \mapsto Wx + b$ e $x \mapsto \sigma(x)$, mentre una KAN sostituisce ogni prodotto $W_{ij}x_i$ con $f_{ij}(x_i)$. Questo implica che una KAN può essere vista come un MLP "con pesi che variano in modo non-lineare col valore dell'input".

2.5.2 Complessità computazionale

Dal punto di vista parametrico, una KAN può avere un numero di parametri superiore rispetto ad una MLP di dimensioni simili. Ad esempio, supponiamo una KAN con L strati, ciascuno di larghezza m , che usa spline di ordine p su r intervalli. Allora il numero totale dei parametri della rete KAN cresce come $O(L m^2 p r)$, mentre una MLP con L strati e larghezza m avrebbe circa $O(L m^2)$ pesi scalari. In teoria quindi le KAN appaiono meno efficienti in termini di numero di parametri. Tuttavia, empiricamente si osserva che spesso basta una KAN con dimensioni molto più piccole per eguagliare le prestazioni di una MLP molto più grande. Dal punto di vista computazionale, l'impiego di funzioni parametriche sugli archi comporta un overhead rispetto alle semplici moltiplicazioni peso-input di una MLP. In pratica, valutare una B-spline su ogni collegamento è più costoso del prodotto scalare in una MLP, soprattutto se la rete è profonda o le spline sono molto finemente discretizzate (cioè utilizzano un numero elevato di punti di controllo). Quindi, l'addestramento di una KAN, può essere più lento, con stime che indicano un tempo di training circa 10 volte superiore rispetto alle MLP a parità di condizioni.

2.5.3 Interpretabilità e flessibilità locale

Uno dei principali vantaggi delle KAN è la loro interpretabilità. Poiché ogni arco implementa una funzione univariata ben definita, è possibile visualizzare direttamente le forme delle attivazioni apprese. Questo permette di comprendere i singoli contributi delle variabili di input e, in alcuni casi,

di dedurre formule simboliche sottostanti, oltre a rendere più semplice il debug e la semplificazione del modello.

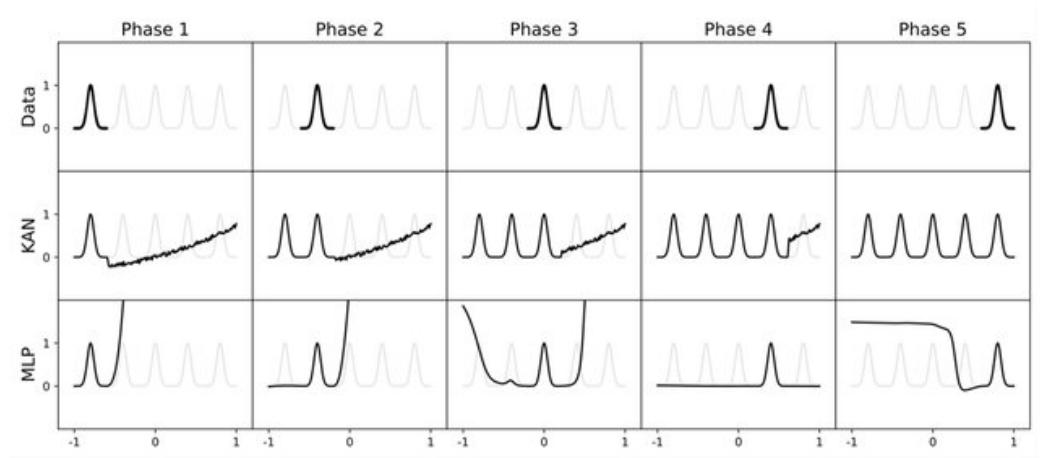


Figura 2.6: Differenza nel Catastrophic forgetting di MLP e KAN.

Un altro vantaggio importante è la flessibilità locale che deriva dalla natura delle spline. A differenza delle MLP che usano funzioni di attivazione globali (come ReLU o Tanh), una KAN modifica solo una piccola regione di input quando apprende una nuova informazione. Ciò riduce significativamente il rischio di "catastrophic forgetting", un fenomeno in cui l'addestramento su nuovi dati può distruggere le informazioni precedentemente apprese.

2.5.4 Precisione controllabile tramite grid extension

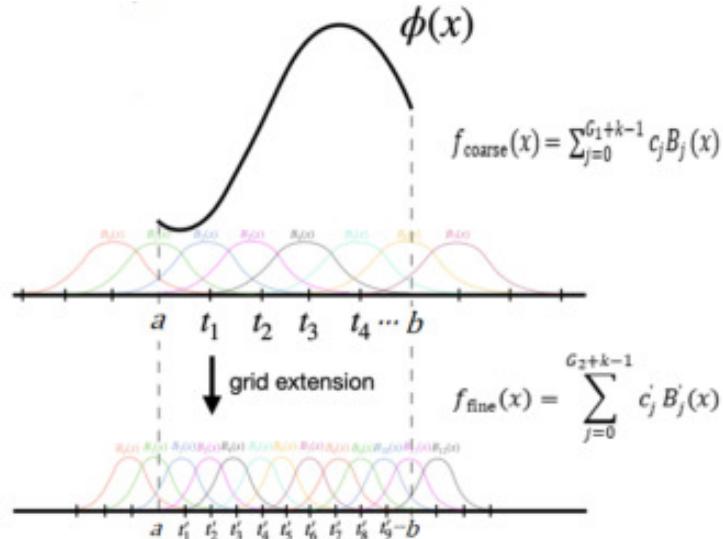


Figura 2.7: Grid extension della spline grid delle KAN.

Le funzioni univariate nelle KAN sono parametrizzate tramite B-spline definite su una griglia. È possibile aumentare la risoluzione della griglia ("grid extension") per incrementare la precisione in modo controllato: partendo da spline grossolane si possono ottenere spline più fini con una procedura di inizializzazione che conserva la continuità e permette rapide riduzioni della loss senza costi computazionali esponenziali.

2.5.5 Dipendenza dalla struttura compositiva

I vantaggi più evidenti delle KAN si manifestano quando la funzione target da approssimare ha una struttura che si avvicina ad una decomposizione in somme di funzioni univariate, cioè può essere sufficientemente rappresentata come somma di trasformazioni su singole variabili. Quando la funzione è intrinsecamente non decomponibile o presenta forti interazioni multivariante, la rappresentazione KAN perde efficacia e può risultare meno efficiente rispetto ad una parametrizzazione densa tipica delle MLP.

2.5.6 Irregolarità nella rappresentazione di Kolmogorov

Il teorema di Kolmogorov–Arnold garantisce l'esistenza di una rappresentazione, ma non la regolarità delle funzioni intermedie $\varphi_{q,p}$. In casi pratici, queste funzioni possono essere non-smooth o altamente oscillanti; per approssimarle con spline possono essere necessarie griglie molto fitte, annullando i vantaggi teorici in termini di parametri e costo computazionale.

2.5.7 Overhead computazionale e scelta della struttura

Parametrizzare ogni arco come funzione spline introduce overhead in memoria ed in tempo di calcolo (valutazione e aggiornamento di B-spline, gestione di griglie differenziate). Inoltre, la scelta automatica della topologia (numero di rami, profondità, risoluzione delle griglie per ciascuna spline) non è banale e richiede procedure di pruning o ricerca strutturale che aumentano la complessità del workflow.

2.5.8 Sensibilità al rumore e necessità di regolarizzazione

In presenza di dati molto rumorosi, una parametrizzazione spline troppo fine tende al sovraffitting locale. È quindi necessario un attento tuning degli iperparametri (ordine della spline, numero di nodi, termine di regolarizzazione, smoothing), e in alcuni casi una MLP ben regolarizzata può mostrare maggiore robustezza.

Capitolo 3

Random forest

3.1 Introduzione

In questo capitolo viene descritto il Random forest, un algoritmo di Machine learning molto versatile, in grado di gestire sia compiti di classificazione che di regressione. Il suo funzionamento si basa sull'idea dell'ensemble learning, combinando la forza di più alberi di decisione per ottenere un modello finale più robusto e preciso. L'introduzione del capitolo si concentra sui concetti fondamentali che guidano la costruzione del modello, come i criteri di divisione dei dati e la tecnica di bagging, che sfrutta il campionamento casuale per ridurre la varianza. Si approfondisce poi l'architettura specifica del Random forest, che aggiunge un ulteriore livello di casualità nella selezione delle variabili per ogni albero, rendendo la "foresta" più diversificata e meno soggetta ad overfitting. Il capitolo si conclude con una valutazione dei vantaggi e degli svantaggi dell'algoritmo, evidenziando la sua robustezza e la sua capacità di generalizzazione, ma anche i suoi requisiti in termini di risorse computazionali e la minore interpretabilità rispetto ad un singolo albero.

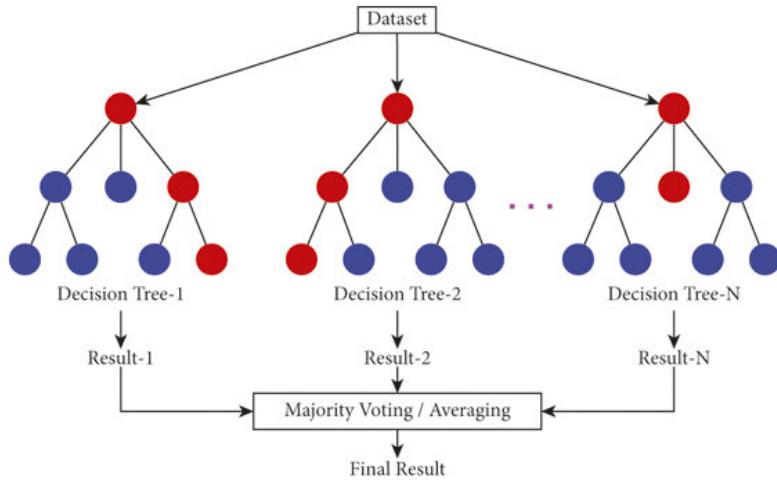


Figura 3.1: Random forest.

Il Random forest è un algoritmo di Machine learning ampiamente utilizzato, noto per la sua robustezza e versatilità sia in problemi di classificazione e regressione. La sua efficacia deriva dalla combinazione di più alberi di decisione "deboli", formando un "ensemble" che supera le prestazioni di un singolo albero, mitigando le debolezze di ciascuno.

3.2 Concetti fondamentali

3.2.1 Alberi di decisione: Criteri di splitting (Gini, Entropia, Gain ratio)

Gli alberi di decisione costituiscono i "learner deboli" fondamentali all'interno di un Random forest. La loro costruzione implica la divisione iterativa dei dati basata sulle features per creare sottoinsiemi sempre più omogenei. La qualità di queste divisioni è misurata da specifici criteri di impurità o Information gain. Il Gini impurity (o Gini index) è una misura di non-omogeneità ampiamente utilizzata negli alberi di classificazione. Essa quantifica la probabilità che un elemento scelto casualmente da un set venga erroneamente etichettato, se classificato in modo casuale, secondo la

distribuzione delle classi nel sottoinsieme. Un valore di 0 indica purezza perfetta, dove tutti gli elementi appartengono alla stessa classe, mentre un valore massimo di $1 - \frac{1}{n}$, dove n è il numero di classi, indica la massima impurità, con le classi equamente distribuite. La formula per il Gini impurity è data da:

$$Gini = 1 - \sum_{i=1}^n (p_i)^2$$

dove p_i rappresenta la proporzione delle istanze della classe i nel set. Ad esempio, se un nodo contiene 50 campioni, di cui 25 di una classe e 25 di un'altra, l'impurità di Gini sarebbe 0.5, indicando la massima incertezza. Dopo uno split, l'algoritmo seleziona la variabile che produce la maggiore diminuzione dell'impurità di Gini, portando a nodi più puri.

L'Entropia (Entropy) misura il grado di disordine, imprevedibilità o incertezza in un dataset. Un'entropia di 0 indica un set perfettamente puro, mentre un valore di $\log_2(n)$ indica la massima incertezza. L'Information gain misura la riduzione dell'entropia ottenuta da uno split, indicando quanta "informazione" viene acquisita sulla variabile target. La formula per l'Entropia è:

$$Entropy = - \sum_{i=1}^n p_i \log_2(p_i)$$

dove p_i è la probabilità della classe i .

Il Gain ratio è stato introdotto per mitigare un problema dell'Information gain, che ha un bias verso attributi con un gran numero di valori. Questi attributi tendono a creare molti nodi piccoli e puri, il che può condurre a un potenziale overfitting. Il Gain ratio normalizza l'Information gain, penalizzando gli split che creano molti sottoinsiemi. La sua formula è:

$$Gainratio = \frac{Informationgain}{Splitinformation}$$

dove lo *Split information* è l'entropia dello split stesso:

$$SplitInformation(S, A) = - \sum_{i=1}^v \frac{|S_i|}{|S|} \log_2\left(\frac{|S_i|}{|S|}\right)$$

dove S è il set di dati del nodo, A è la feature su cui si sta splittando, v è il numero di valori unici della feature, $|S_i|$ è il numero di istanze nel sottoinsieme i e $|S|$ è il numero totale di istanze.

Un confronto tra Gini impurity, Entropia e Gain ratio rivela differenze importanti. Il Gini impurity ha un intervallo di valori compreso tra $[0, 1 - \frac{1}{n}]$, mentre l'Entropia ha un intervallo tra $[0, \log_2(n)]$. Dal punto di vista computazionale, il Gini index è generalmente più efficiente da calcolare rispetto all'Entropia, poiché quest'ultima richiede l'uso di logaritmi. La scelta tra i criteri non è arbitraria, ma implica un compromesso. Il Gini, essendo computazionalmente più veloce, è meno incline a produrre alberi di decisione molto profondi, poiché privilegia split che generano nodi più bilanciati. Al contrario, l'Entropia, sebbene più onerosa, tende a generare alberi che massimizzano la riduzione dell'incertezza, ma il suo bias può portare a preferire caratteristiche con molte categorie, aumentando il rischio di overfitting. Per mitigare ciò, il Gain Ratio si dimostra più robusto.

Per dataset molto grandi o per applicazioni con stringenti requisiti di velocità, il Gini potrebbe essere la scelta preferibile. Per contro, in contesti dove la massima purezza dei nodi è cruciale, l'Entropia (o, più precisamente, il Gain ratio) potrebbe rivelarsi più efficace, a condizione che il rischio di overfitting venga gestito adeguatamente. Questa decisione fondamentale, a livello del singolo albero, si ripercuote sulla performance e sulla struttura complessiva del Random forest. Un albero "più debole" ma più rapido, generato con il Gini, può essere efficacemente compensato dall'approccio Ensemble, mentre alberi "più forti" ma più lenti, derivanti dall'Entropia, potrebbero non scalare con la stessa efficienza.

3.2.2 Ensemble learning e Bagging

Il principio alla base dell'Ensemble learning è spesso descritto come la "saggezza della folla": un gruppo di learner deboli, che individualmente

potrebbero non performare in modo ottimale, a causa di alta varianza o alto bias, può, quando le loro previsioni vengono aggregate, formare un "learner forte" con prestazioni notevolmente migliorate.

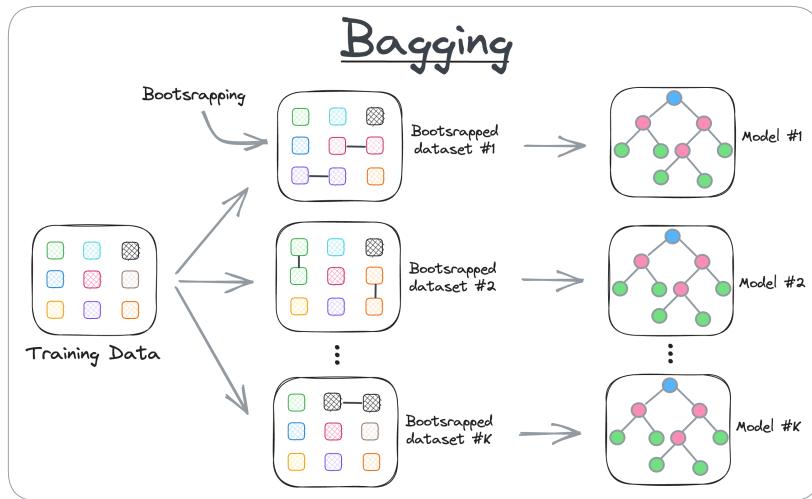


Figura 3.2: Bagging.

Il Bagging è un metodo di Ensemble learning il cui scopo principale è ridurre la varianza all'interno di un dataset "rumoroso", migliorando così la capacità di generalizzazione del modello e mitigando l'overfitting. Il processo di Bagging si articola in tre fasi fondamentali:

1. **Bootstrapping:** questa tecnica di ricampionamento genera diversi sottoinsiemi del training set. Il campionamento avviene selezionando istanze in modo casuale con re-immissione, ciò significa che una singola istanza può essere scelta più volte all'interno dello stesso sottoinsieme. Questo processo è cruciale per creare diversità tra i campioni su cui verranno addestrati i modelli individuali.
2. **Addestramento parallelo:** i campioni bootstrap così generati vengono utilizzati per addestrare, in modo indipendente e parallelo, una serie di learner deboli, che nel contesto di Random forest sono tipicamente alberi di decisione.

3. **Aggregazione:** una volta che i modelli individuali hanno prodotto le loro previsioni, queste vengono combinate. Per i problemi di regressione, si utilizza un processo noto come Soft voting (cioè la media di tutti gli output previsti dai singoli learner), mentre, per i problemi di classificazione, si utilizza l'Hard o Majority voting (cioè viene scelta la classe più votata).

Il beneficio più significativo è la riduzione della varianza, particolarmente utile con dati ad alta dimensionalità o in presenza di valori mancanti, dove un'alta varianza può rendere il modello più incline all'overfitting. La diversità introdotta nei dati di training per ciascun modello contribuisce a ridurre la varianza nelle previsioni finali. Questo processo mitiga l'influenza del rumore nei dati e degli outlier, producendo un modello aggregato più stabile ed affidabile. Tuttavia, il Bagging può portare ad una perdita di interpretabilità, rendendo difficile estrarre intuizioni precise a causa del processo di media delle previsioni. È anche computazionalmente costoso, rallentando e diventando più intensivo all'aumentare del numero di iterazioni. Infine, è meno flessibile con algoritmi già stabili o con un alto bias, poiché i benefici, in termini di riduzione della varianza, sono meno visibili.

Il Bagging non è semplicemente un metodo per combinare modelli, ma agisce come una forma intrinseca di regolarizzazione. Addestrando modelli su sottoinsiemi diversi del dataset, ottenuti attraverso il campionamento di tipo bootstrap, ciascun modello apprende una prospettiva leggermente differente del problema. Quando le previsioni di questi modelli, sebbene diversi, sono aggregate, gli errori casuali e la varianza intrinseca di un singolo modello tendono a compensarsi reciprocamente. Ciò porta a una previsione finale che è più stabile e meno sensibile al rumore oppure agli outlier. Questo meccanismo è la ragione fondamentale per cui il Bagging è così efficace nel ridurre l'overfitting, specialmente per learner ad alta varianza, come gli alberi di decisione profondi. Questa capacità di ridurre la varianza senza introdurre un bias significativo rende il Bagging una base così potente per algoritmi come Random forest, che altrimenti sarebbero molto inclini all'overfitting. È una dimostrazione del principio che la "di-

versità" all'interno di un ensemble conduce a una maggiore robustezza del modello complessivo.

3.3 Architettura e Costruzione

Random forest estende il concetto di Bagging introducendo un ulteriore livello di casualità, rendendo l'Ensemble ancora più robusto e meno propenso all'overfitting.

3.3.1 Bootstrapping e Feature bagging

La costruzione di un Random forest si basa su due fonti principali di casualità, essenziali per garantire che gli alberi individuali siano il più possibile non correlati tra loro.

La prima fonte di casualità è il campionamento di tipo bootstrap. Per ogni albero che fa parte della foresta, viene estratto un campione casuale di dati dal set di training originale con re-immissione. Questo sottoinsieme di dati è noto come Bootstrap sample. Una caratteristica importante di questo processo è che circa un terzo dei dati originali non viene selezionato per un dato bootstrap sample; questi dati non utilizzati sono chiamati campioni "out-of-bag" e possono essere impiegati per la validazione interna del modello. Questo tipo di campionamento assicura che ogni albero sia addestrato su un sottoinsieme leggermente diverso dei dati originali, promuovendo una diversità fondamentale tra gli alberi.

La seconda fonte di casualità è legata alle features, nota come feature bagging o random subspace method. Ad ogni split dei nodi, all'interno di un albero di decisione, Random forest non considera tutte le feature disponibili, ma seleziona solo un sottoinsieme casuale di esse. Questa è una differenza importante rispetto agli alberi di decisione standard, che valuterebbero tutte le feature possibili per trovare lo split migliore. L'introduzione di questa casualità nella selezione delle feature aggiunge ulteriore diversità al dataset per ogni albero e riduce significativamente la correlazione tra gli alberi di decisione individuali.

La casualità introdotta nel Random Forest, tramite bootstrapping e feature bagging, non è ridondante, ma complementare e strategica. Il Bootstrapping riduce la varianza generale del modello assicurando che ogni albero acquisisca una prospettiva leggermente diversa del dataset complessivo. Il Feature bagging, d'altra parte, previene che caratteristiche particolarmente forti o dominanti influenzino la costruzione di tutti gli alberi. Se una singola caratteristica fosse sempre scelta come il miglior punto di split, tutti gli alberi all'interno della foresta risulterebbero molto simili e altamente correlati, rendendo inutili i benefici derivanti dall'approccio Ensemble. Introducendo la casualità nella selezione delle feature, si forza ogni albero ad esplorare diverse combinazioni di caratteristiche, riducendo ulteriormente la loro correlazione e, di conseguenza, la varianza dell'intero Ensemble.

3.3.2 Aggregazione delle predizioni

Una volta che tutti gli alberi di decisione sono stati costruiti e addestrati sui rispettivi sottoinsiemi di dati e feature, le loro previsioni vengono combinate per ottenere il risultato finale del Random forest. Il metodo di aggregazione dipende dalla natura del problema:

- **Regressione:** le previsioni numeriche generate da ciascun albero individuale vengono semplicemente mediate. Il valore medio di tutte le previsioni degli alberi costituisce la previsione finale del modello.
- **Classificazione:** la classe finale viene determinata attraverso un processo di voto di maggioranza. Ogni albero nella foresta produce una previsione e la classe che riceve il maggior numero di "voti" (cioè, la più scelta) tra tutti gli alberi viene accettata come previsione finale del Random forest.

Questo processo di aggregazione trasforma un insieme di learner deboli in un modello forte. Sebbene gli alberi di decisione individuali possano presentare un'alta varianza, la media o il voto di maggioranza sulle loro previsioni riduce significativamente la varianza complessiva e rende il modello meno sensibile ad errori ed outlier. Questo processo mitiga la tendenza

del singolo albero a sovrastimare o sottostimare i valori, producendo una previsione finale più stabile e robusta. Questo meccanismo di aggregazione è ciò che consente al Random forest di raggiungere un'elevata accuratezza mantenendo al contempo una solida capacità di generalizzazione. È un esempio pratico dell'applicazione del principio della "saggezza della folla" nel Machine learning, dove la combinazione di molteplici opinioni individuali, seppur imperfette, conduce ad un risultato finale superiore.

3.4 Vantaggi

- **Riduzione della varianza:** è intrinsecamente meno propenso all'overfitting rispetto a un singolo albero di decisione. Questo è dovuto alla sua natura ensemble, che aggrega le previsioni di molti alberi diversi, ed all'introduzione di casualità sia nel campionamento dei dati che nella selezione delle features.
- **Robustezza agli outlier ed ai dati rumorosi:** il modello è meno sensibile all'influenza degli outlier e del rumore nei dati, poiché le previsioni vengono mediate su più alberi, diluendo l'impatto di singole osservazioni estreme.
- **Gestione dei valori mancanti:** Random forest può gestire efficacemente i valori mancanti nel dataset, rendendo la fase di pre-processing dei dati più semplice.
- **Gestione dei dati sbilanciati:** ha la capacità di gestire dataset in cui le classi sono sbilanciate.
- **Facilità di determinazione dell'importanza delle feature:** l'algoritmo rende relativamente semplice valutare il contributo di ciascuna variabile al modello.
- **Parallelizzabile:** i singoli alberi, all'interno del Random forest, possono essere addestrati in parallelo. Questa caratteristica contribuisce

significativamente alla sua velocità di addestramento, specialmente quando si lavora con dataset di grandi dimensioni e hardware multi-core.

3.5 Svantaggi

- **Costo computazionale e Tempo di addestramento:** l'addestramento di un Random forest può essere lento, in particolare con un numero molto elevato di alberi o su dataset di grandi dimensioni, poiché la costruzione di ogni albero è un'operazione computazionalmente intensiva.
- **Requisiti di risorse:** poiché elabora dataset potenzialmente grandi e costruisce molti alberi, richiede più risorse di memoria per archiviare i dati rispetto ad un singolo albero.
- **Complessità e Perdita di interpretability (rispetto a un singolo albero):** la previsione di una "foresta" di centinaia o migliaia di alberi diventa molto più difficile da interpretare a livello globale.
- **Mancanza di regolarizzazione esplicita:** A differenza di altri algoritmi ensemble, Random forest non include internamente tecniche di regolarizzazione dirette. Si affida principalmente alla sua natura ensemble e all'introduzione di casualità per prevenire l'overfitting.

3.6 Vantaggi

Il Random Forest si distingue per la sua robustezza e flessibilità, offrendo diversi vantaggi che lo rendono una scelta popolare nel machine learning. Un punto di forza principale è la **riduzione della varianza**, che lo rende intrinsecamente meno propenso all'overfitting rispetto a un singolo albero di decisione. Questo risultato è ottenuto grazie alla sua natura ensemble, che aggrega le previsioni di molti alberi indipendenti, ed all'introduzione di casualità sia nel campionamento dei dati (bootstrap) sia nella selezione

delle feature per ogni singolo albero. La sua robustezza non si limita alla varianza. L'algoritmo mostra anche una notevole **robustezza agli outlier**, poiché l'impatto di singole osservazioni estreme viene diluito e mediato tra i vari alberi. Il Random Forest eccelle anche nella **gestione dei valori mancanti**, semplificando notevolmente la fase di pre-elaborazione dei dati. Allo stesso modo, è in grado di gestire efficacemente **dataset sbilanciati**. Dal punto di vista della usabilità, il Random Forest facilita la **determinazione dell'importanza delle feature**, offrendo un modo diretto per valutare il contributo di ogni variabile al modello. La sua architettura è inoltre **parallelizzabile**, il che significa che i singoli alberi possono essere addestrati in parallelo. Questa caratteristica contribuisce in modo significativo alla sua velocità di addestramento, in particolare con dataset di grandi dimensioni e hardware multi-core.

3.7 Svantaggi

Nonostante i numerosi vantaggi, il Random Forest presenta alcune limitazioni, principalmente legate ai requisiti computazionali e alla complessità. Uno svantaggio notevole è il **costo computazionale ed il tempo di addestramento**. L'addestramento può essere lento, soprattutto con un numero elevato di alberi o su dataset molto grandi, poiché la costruzione di ogni singolo albero è un'operazione computazionalmente intensiva. Di conseguenza, il modello ha anche **requisiti di memoria** più elevati rispetto a un singolo albero, dato che deve memorizzare la struttura di tutti gli alberi che compongono la foresta. Un'altra debolezza significativa è **la complessità e la perdita di interpretabilità**. A differenza di un singolo albero di decisione, la cui logica è facile da visualizzare e comprendere, una "foresta" composta da centinaia o migliaia di alberi rende la previsione molto più difficile da interpretare a livello globale. Infine, il Random Forest **non include una regolarizzazione esplicita** nel suo nucleo. A differenza di altri algoritmi ensemble che possono incorporare tecniche di regolarizzazione dirette, il Random Forest si affida principalmente alla sua natura ensemble ed

all'introduzione di casualità per prevenire l'overfitting.

Capitolo 4

eXtreme Gradient Boosting (XGBoost)

4.1 Introduzione

Questo capitolo presenta eXtreme Gradient Boosting (XGBoost), un algoritmo che ha rivoluzionato il Machine Learning per la sua efficacia e velocità. L'obiettivo è analizzare come XGBoost, pur basandosi sul Gradient Boosting, lo superi grazie ad una serie di importanti ottimizzazioni. Verranno esaminati i miglioramenti che lo rendono così performante, come l'uso di una funzione di perdita avanzata ed una gestione più efficiente della regolarizzazione e dei dati mancanti. Il testo spiegherà come la sua architettura sfrutti il parallelismo e l'uso intelligente della cache per velocizzare i calcoli, rendendo l'addestramento più rapido. Si affronterà anche il tema degli iperparametri, che offrono una grande flessibilità per personalizzare il modello, sebbene richiedano un'attenta calibrazione. Il capitolo si conclude con un riassunto dei suoi vantaggi, come la robustezza e l'efficienza, e dei suoi svantaggi, tra cui la complessità e la maggiore richiesta di risorse per il tuning.

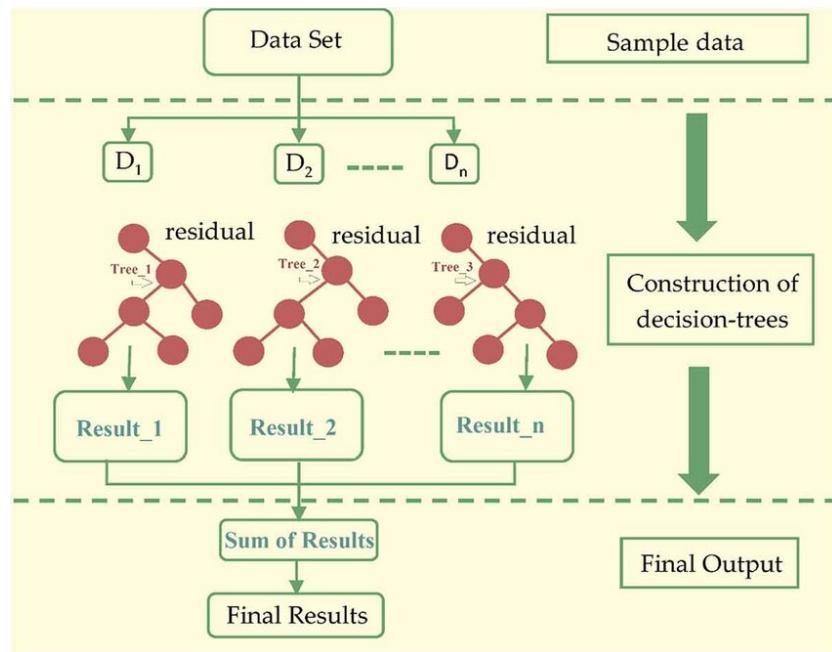


Figura 4.1: XGBoost.

4.2 Introduzione al Gradient boosting

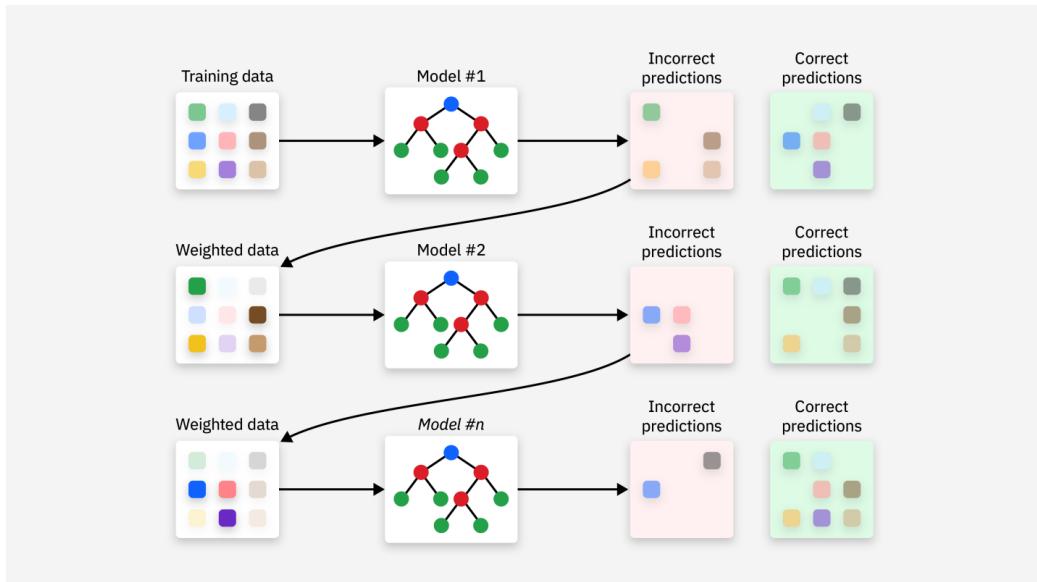


Figura 4.2: Gradient boosting.

Il Gradient boosting è una tecnica di Ensemble learning che costruisce un modello predittivo forte combinando iterativamente i risultati di numerosi "learner deboli". A differenza del Bagging, che addestra i modelli in parallelo, il Boosting adotta un approccio sequenziale, dove ogni nuovo modello cerca di correggere gli errori commessi dai modelli precedenti.

4.2.1 Principi iterativi e Weak learners

Il gradient boosting si fonda sull'idea di migliorare progressivamente un modello addestrando nuovi learner per correggere gli errori commessi dai precedenti. Questo processo è intrinsecamente iterativo e sequenziale. A differenza del Bagging, dove i learner sono addestrati in modo parallelo ed indipendente, il gradient boosting costruisce un modello additivo passo dopo passo. Ogni nuovo "albero" (che funge da learner debole) viene costruito specificamente per ridurre gli errori, o più precisamente gli "pseudo-residui", generati dalle previsioni degli alberi addestrati nelle iterazioni precedenti. I "learner deboli", in questo caso, sono modelli che, se utilizzati singolarmente, classificano o predicono i dati in modo scarso e presentano un alto tasso di errore. Nel framework del gradient boosting, i learner deboli sono tipicamente alberi di decisione semplici. Possono essere molto semplici, a volte ridotti ad un singolo split, in tal caso sono noti come "decision stumps". L'obiettivo generale del gradient boosting è minimizzare una funzione di perdita predefinita, aggiungendo iterativamente funzioni (i learner deboli) che puntano nella direzione del gradiente negativo di tale funzione. Mentre il Bagging mira a ridurre la varianza addestrando modelli indipendenti e poi mediandone i risultati, il Boosting si concentra sulla riduzione del bias del modello. Addestrando sequenzialmente nuovi learner per correggere gli errori dei precedenti, il modello impara a concentrarsi sulle istanze più difficili da classificare o predire. Questo processo iterativo consente al modello di adattarsi in modo più efficace alle relazioni complesse presenti nei dati, riducendo il bias complessivo e portando spesso a una maggiore accuratezza predittiva. Questa differenza fondamentale nell'approccio, ovvero la riduzione della

varianza contro la riduzione del bias, spiega perché il Bagging ed il Boosting eccellono in contesti diversi.

4.2.2 Funzione di perdita e Discesa del gradiente

Il processo di apprendimento nel gradient boosting è formalizzato come un algoritmo di discesa del gradiente nello spazio delle funzioni, dove l'obiettivo è trovare la funzione che minimizza la funzione di perdita, che quantifica quanto bene il modello sta eseguendo le previsioni sui dati forniti. La scelta della funzione di errore dipende dalla natura del problema: ad esempio, per problemi di regressione si potrebbe usare l'errore quadratico medio (MSE), mentre per problemi di classificazione si potrebbe usare la log-loss. L'algoritmo di gradient boosting mira a minimizzare questa funzione di perdita. In ogni iterazione, il modello calcola i cosiddetti "pseudo-residui", che non sono i residui tradizionali (differenza tra valore osservato e previsto), ma piuttosto i gradienti negativi della funzione di perdita rispetto alle previsioni attuali del modello. Il nuovo learner debole (tipicamente un albero di decisione) viene quindi addestrato per predire questi pseudo-residui, imparando così a correggere gli errori del modello ensemble cumulativo. Il processo di aggiunta di nuovi alberi può essere interpretato come un passo nella direzione del gradiente negativo della funzione di perdita nello spazio delle funzioni.

Un'innovazione significativa in XGBoost rispetto al gradient boosting tradizionale è l'utilizzo di un'approssimazione di Taylor del secondo ordine nella funzione di perdita. Questo approccio collega il processo di ottimizzazione di XGBoost al metodo Newton-Raphson, che è più robusto e può portare a una convergenza più rapida rispetto alla discesa del gradiente del primo ordine. L'utilizzo di un'approssimazione di Taylor del secondo ordine nella funzione di perdita distingue XGBoost dal gradient boosting tradizionale, che si basa sul gradiente del primo ordine. Ciò implica che XGBoost considera, non solo la direzione di discesa più ripida (data dal gradiente), ma anche la curvatura della funzione di perdita (data dall'hessiana). Ciò consente al modello di compiere passi più informati e potenzialmente più

ampi verso il minimo della funzione di perdita, portando a una convergenza più rapida e stabile. Di conseguenza, XGBoost risulta meno sensibile a problemi come i minimi locali rispetto agli algoritmi che utilizzano esclusivamente il gradiente del primo ordine. Questa modifica è uno dei motivi principali della superiorità prestazionale di XGBoost in numerose competizioni di machine learning ed in svariate applicazioni reali. Dimostra come un'implementazione più avanzata dei principi fondamentali possa tradursi in miglioramenti significativi in termini di prestazioni ed efficienza del modello.

4.3 Miglioramenti di XGBoost rispetto al Gradient boosting tradizionale

XGBoost è riconosciuto come un'evoluzione del gradient boosting, grazie all'integrazione di una serie di ottimizzazioni e tecniche di regolarizzazione che ne migliorano significativamente prestazioni, velocità e robustezza.

4.3.1 Regolarizzazione e Tree pruning

XGBoost si distingue per l'integrazione di meccanismi di regolarizzazione configurabili, che sono cruciali per prevenire l'overfitting e migliorare la sua capacità di generalizzazione. Il modello incorpora termini di regolarizzazione L1 (Lasso) e L2 (Ridge) direttamente nella sua funzione obiettivo. Questi termini penalizzano la complessità del modello e i pesi di grandi dimensioni. In particolare, la regolarizzazione L1, che si basa sulla somma del valore assoluto dei pesi:

$$\Omega(w) = \lambda \|w\|_1 = \lambda \sum_j |w_j|$$

incoraggia la sparsità, spingendo i pesi meno importanti verso lo zero. Al contrario, la regolarizzazione L2, che si basa sulla somma dei quadrati dei

pesi:

$$\Omega(w) = \lambda \|w\|_2^2 = \lambda \sum_j w_j^2$$

incoraggia pesi più piccoli e distribuiti.

Una componente fondamentale della regolarizzazione nel gradient boosting è il learning rate (o shrinkage). Questo parametro riduce il contributo di ogni nuovo albero aggiunto al modello. L'uso di learning rate piccoli (ad esempio, 0.01 o 0.1) migliora notevolmente la capacità di generalizzazione del modello, sebbene ciò comporti un aumento del numero di iterazioni e, di conseguenza, del tempo di calcolo.

XGBoost implementa anche tecniche avanzate di tree pruning. A differenza di alcuni algoritmi che costruiscono alberi fino alla massima profondità, XGBoost pone gli alberi in base a un "guadagno" del nodo. La decisione di effettuare una ulteriore divisione su un nodo foglia dipende dal raggiungimento di un guadagno minimo, specificato dall'iperparametro γ (gamma). Un valore più grande di γ rende l'algoritmo più conservativo, limitando la crescita dell'albero e aiutando a prevenire l'overfitting. Il guadagno di un nodo in XGBoost quantifica il miglioramento nella funzione obiettivo che si ottiene dividendo un nodo. Questo guadagno deve superare una soglia minima, definita dall'iperparametro γ , per autorizzare la divisione. La formula per il guadagno è:

$$\text{Gain} = \frac{1}{2} \left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma$$

Dove I_L e I_R sono gli insiemi di istanze (dati di addestramento) nei nodi figli sinistro e destro, mentre I rappresenta l'insieme di istanze nel nodo padre. In questa formula, g_i è la derivata prima della funzione di perdita rispetto all'output dell'istanza i , mentre h_i è la derivata seconda della funzione di perdita rispetto allo stesso output. Il parametro λ è un termine di regolarizzazione L2 che penalizza i pesi elevati, e γ è la soglia di guadagno minima richiesta per la divisione. Se il guadagno calcolato è inferiore a γ , la divisione viene annullata.

Un altro parametro cruciale per la regolarizzazione è la somma minima dei pesi delle istanze, noto come "min_child_weight". Questo iperparametro agisce come un criterio di regolarizzazione per controllare la crescita degli alberi. Un nodo figlio può essere creato solo se la somma dei pesi delle istanze al suo interno supera un valore soglia predefinito. Questa somma è calcolata usando l'**hessiana**, e la condizione per uno split può essere espressa come:

$$\sum_{i \in I_L} h_i \geq \text{min_child_weight} \quad \text{e} \quad \sum_{i \in I_R} h_i \geq \text{min_child_weight}$$

Se una delle somme è inferiore al valore di "min_child_weight", la partizione viene annullata. Un valore più grande di questo iperparametro rende l'algoritmo più conservativo, riducendo la complessità degli alberi individuali e aiutando a prevenire l'overfitting.

4.3.2 Gestione dei valori mancanti

XGBoost è in grado di lavorare efficacemente anche quando nel dataset ci sono dati assenti o non registrati per alcune feature. XGBoost incorpora internamente un meccanismo che permette di decidere automaticamente, durante la costruzione degli alberi, come trattare i valori mancanti. Durante la costruzione degli alberi di decisione, quando viene incontrato un valore mancante per una determinata feature, XGBoost non si limita a ignorare o richiedere una rimozione preliminare di essa. Invece, l'algoritmo impara quale direzione (ramo dell'albero) seguire per le istanze con valori mancanti per ottimizzare le performance. Internamente, durante la fase di training, XGBoost tratta la "mancanza" del dato come una caratteristica informativa a sé stante, apprendendo la direzione ottimale per i dati mancanti in modo da ottimizzare le suddivisioni. Questa capacità semplifica la pipeline di preparazione dei dati, evitando bias o rumori introdotti da imputazioni errate o rimozioni arbitrarie, e rende il modello più robusto e affidabile in scenari del mondo reale, dove i dati spesso presentano valori mancanti. Questa funzionalità rende XGBoost particolarmente efficiente e pratico.

per dataset reali, che spesso contengono valori mancanti, riducendo la necessità di avere una fase di pre-processing complessa, migliorando così l'affidabilità del modello in scenari del mondo reale.

4.3.3 Ottimizzazioni (Parallelismo, Cache-awareness)

Una delle ottimizzazioni chiave di XGBoost è il parallelismo. Sebbene il gradient boosting sia intrinsecamente sequenziale nella costruzione degli alberi (ogni albero corregge gli errori del precedente), XGBoost introduce il parallelismo, nella costruzione dei singoli alberi, in particolare sui livelli dell'albero o di split. Questo significa che, anziché costruire gli alberi in modo strettamente sequenziale, XGBoost può scansionare i valori del gradiente ed utilizzare somme parziali per valutare la qualità degli split in parallelo. Il sistema sfrutta tutti i core della CPU disponibili su una singola macchina e può operare in modalità distribuita, massimizzando l'utilizzo della potenza di calcolo. Questo parallelismo su larga scala accelera significativamente il processo di addestramento.

Un'altra ottimizzazione interessante è il Cache-awareness. XGBoost è progettato per utilizzare in modo intelligente la cache della CPU per accelerare l'accesso ai dati. Durante l'addestramento, memorizza nella cache i calcoli intermedi e le statistiche importanti, evitando così di ricalcolare gli stessi valori ripetutamente. Questo riduce i ritardi nel recupero dei dati tra la CPU e memoria principale, portando ad un'elaborazione e previsioni molto più veloci.

Infine, XGBoost beneficia della GPU acceleration, che velocizza significativamente l'addestramento del modello e contribuisce a migliorare l'accuratezza delle previsioni. L'algoritmo sfrutta il calcolo parallelo per eseguire operazioni veloci, per ripartizionare i dati e costruire gli alberi un livello alla volta, elaborando l'intero dataset contemporaneamente sulla GPU.

4.4 Parametri chiave e Tuning

XGBoost offre un'ampia e dettagliata lista di iperparametri che possono essere ottimizzati per personalizzare il comportamento del modello e massimizzare le sue prestazioni. Questa flessibilità, sebbene potente, richiede una comprensione approfondita ed un'attenta strategia di tuning. I parametri per il Tree booster controllano la costruzione dei singoli alberi all'interno dell'Ensemble:

- **learning_rate** (o **eta**): questo è il tasso di apprendimento, un parametro cruciale che controlla la dimensione del passo di shrinkage per prevenire l'overfitting. Valori più piccoli di **eta** rendono il processo di boosting più conservativo, riducendo il rischio di overfitting ma richiedendo più iterazioni. Il suo intervallo è $(0, 1]$.
- **max_depth**: definisce la profondità massima di un albero. Aumentare questo valore rende il modello più complesso e potenzialmente più incline all'overfitting. Un valore di 0 indica nessuna limitazione di profondità, ma ciò può portare a un elevato consumo di memoria. L'intervallo è $[0, \infty]$.
- **min_child_weight**: specifica la somma minima del peso delle istanze (basata sull'hessiana) necessaria in un nodo figlio per consentire un ulteriore split. Un valore più grande rende l'algoritmo più conservativo, limitando la crescita dell'albero. L'intervallo è $[0, \infty]$.
- **subsample**: rappresenta la frazione di osservazioni (istanze) campionate casualmente per la costruzione di ogni albero. Questo campionamento avviene una volta per ogni iterazione di boosting ed aiuta a prevenire l'overfitting. L'intervallo è $(0, 1]$.
- **colsample_bytree**: indica la frazione di features campionate casualmente per la costruzione di ogni albero. Questo parametro contribuisce anch'esso a prevenire l'overfitting introducendo casualità nella selezione delle caratteristiche. L'intervallo è $(0, 1]$.

- `lambda` (o `reg_lambda`): è il termine di regolarizzazione L2 sui pesi. Aumentare questo valore rende il modello più conservativo, penalizzando i pesi grandi. L'intervallo è $[0, \infty)$.
- `alpha` (o `reg_alpha`): è il termine di regolarizzazione L1 sui pesi. Un valore più grande rende il modello più conservativo e incoraggia la sparsità, spingendo i pesi meno importanti verso lo zero. L'intervallo è $[0, \infty]$.
- `objective`: definisce la funzione obiettivo che il modello mira a minimizzare. Ad esempio, `reg:squarederror` per problemi di regressione, `binary:logistic` per classificazione binaria e `multi:softprob` per classificazione multclasse.
- `eval_metric`: specifica la metrica di valutazione da monitorare durante l'addestramento. È possibile specificare più metriche.

A differenza di Random forest, che tende ad avere meno parametri da ottimizzare, XGBoost richiede una comprensione più approfondita ed una sperimentazione più estesa per raggiungere le sue prestazioni ottimali. Questo implica che, sebbene XGBoost possa teoricamente superare Random forest in termini di accuratezza, raggiungere tale superiorità richiede un investimento maggiore in termini di tempo e risorse per il tuning.

4.5 Vantaggi

L'algoritmo XGBoost è rinomato per le sue prestazioni e si distingue per una serie di vantaggi chiave che lo rendono uno standard nel machine learning competitivo. Uno dei suoi punti di forza principali è la **robustezza all'overfitting**, garantita da varie tecniche di regolarizzazione integrate. Questo controllo granulare sulla complessità del modello è un fattore determinante per la sua notevole capacità di generalizzazione. XGBoost è stato progettato per la scalabilità e l'efficienza, permettendogli una gestione efficiente di grandi dataset, dati sparsi e valori mancanti.

La sua architettura ottimizzata consente di elaborare volumi di dati che per altri algoritmi sarebbero difficili da gestire. Nonostante la sua natura sequenziale, **la velocità di addestramento** di XGBoost è eccezionale; può superare quella del Random Forest, specialmente quando si sfruttano le sue capacità di parallelismo e l'accelerazione GPU, oltre al supporto per sistemi distribuiti.

Un altro vantaggio significativo è **la sua flessibilità e personalizzazione**. L'algoritmo offre un'ampia gamma di iperparametri che consentono un fine-tuning profondo, adattando il modello alle specifiche esigenze di ogni problema e dataset. Inoltre, XGBoost è particolarmente efficace nella **gestione di dati sbilanciati**, un problema comune in molti contesti di classificazione.

4.6 Svantaggi

Nonostante i suoi punti di forza, XGBoost presenta alcune limitazioni, principalmente legate alla sua complessità. **Il processo di addestramento sequenziale** può renderlo intrinsecamente più lento del Random Forest nella costruzione completa degli alberi, dato che ogni albero dipende dal precedente. Sebbene siano state introdotte ottimizzazioni per il parallelismo interno, la sua natura sequenziale rimane una potenziale barriera, a meno che non si sfruttino appieno le sue capacità di parallelismo e le accelerazioni hardware.

La complessità e la necessità di tuning rappresentano un'altra sfida. XGBoost è un algoritmo più complesso da comprendere e implementare rispetto al Random Forest. La sua vasta gamma di iperparametri richiede una maggiore conoscenza ed esperienza per un fine-tuning efficace, rendendo il processo più dispendioso in termini di tempo e risorse.

Inoltre, XGBoost può risultare **meno interpretabile del Random Forest**. Sebbene fornisca l'importanza delle feature, la complessità dell'ensemble di alberi sequenziali può rendere le sue decisioni specifiche più difficili da interpretare. Spesso sono necessari strumenti aggiuntivi per la spiegabilità,

a differenza del Random Forest che, in alcuni casi, può essere più trasparente.

Infine, il **consumo di memoria** può essere una limitazione significativa. XGBoost può consumare una notevole quantità di memoria, in particolare quando si addestrano alberi molto profondi. Questo può rappresentare un problema per dataset estremamente grandi su hardware con risorse limitate.

Capitolo 5

Convolutional Neural Networks (CNN)

5.1 Introduzione

Questo capitolo si occupa di introdurre le Convolutional Neural Networks (CNN), un’architettura di rete neurale che eccelle nell’analisi di immagini e di dati con una struttura a griglia. L’obiettivo è esplorare come queste reti riescano ad estrarre automaticamente le caratteristiche rilevanti, superando i limiti delle reti tradizionali. Il testo descriverà il funzionamento dei principali blocchi costruttivi, come la convoluzione e il pooling, che permettono alla rete di identificare forme e pattern in modo gerarchico e di ridurne la complessità. Verrà poi spiegato come, attraverso una sequenza di operazioni, un’immagine viene trasformata in rappresentazioni via via più astratte fino a giungere ad una decisione finale. Infine, il capitolo riassume i vantaggi delle CNN, come la loro efficienza e la capacità di riconoscere oggetti a prescindere dalla loro posizione, e i loro svantaggi, tra cui la necessità di enormi quantità di dati e il loro costo computazionale.

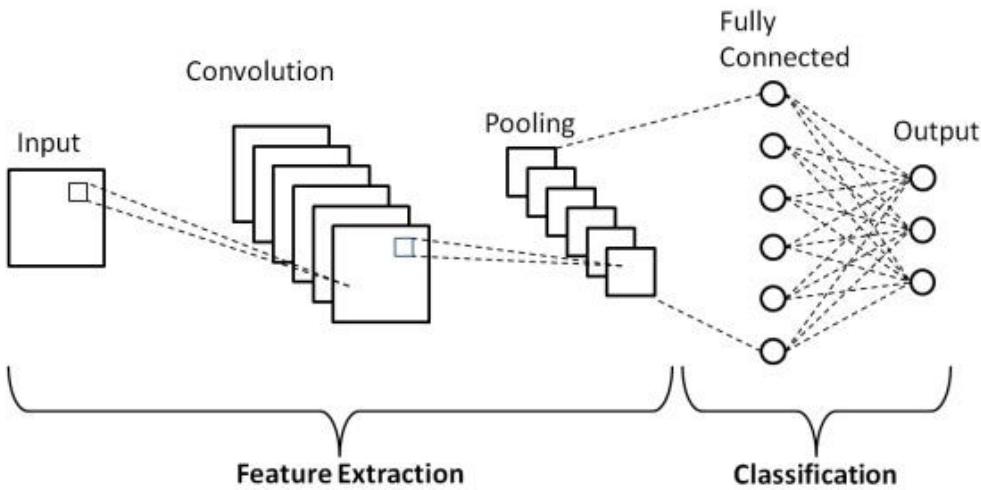


Figura 5.1: Convolutional Neural Networks.

5.2 Principi fondamentali delle CNN

5.2.1 Convoluzione: kernel, stride, padding

Un kernel (o filtro) è una matrice di piccole dimensioni, tipicamente 3×3 , 5×5 o 7×7 , contenente pesi apprendibili che vengono moltiplicati elemento per elemento con porzioni locali dell'input. Durante l'operazione di convoluzione, il kernel viene fatto scorrere attraverso l'intera immagine di input, calcolando il prodotto scalare tra i pesi del filtro e i valori corrispondenti dell'input in ogni posizione.

Lo stride rappresenta la grandezza in pixel di cui il kernel si sposta ad ogni passo durante la convoluzione. Uno stride di 1 significa che il filtro si muove di un pixel alla volta, producendo un output con dimensioni spaziali simili all'input. Uno stride maggiore (ad esempio 2 o 3) riduce significativamente le dimensioni dell'output, fornendo un effetto di downsampling.

Il padding è una tecnica che consiste nell'aggiungere pixel (solitamente con valore zero) ai bordi dell'immagine di input. Esistono due tipi principali di padding:

- Valid padding: nessun padding viene aggiunto, l'output risulta più piccolo dell'input.
- Same padding: viene aggiunto padding sufficiente per mantenere le stesse dimensioni spaziali dell'input.

La formula per calcolare le dimensioni dell'output di una convoluzione è:

$$O = \frac{W - K + 2P}{S} + 1$$

dove W è la dimensione dell'input, K è la dimensione del kernel, P è il padding e S è lo stride.

5.2.2 Feature maps e profondità dei canali

Le feature maps rappresentano l'output prodotto dall'applicazione di filtri convoluzionali all'input. Ogni feature map corrisponde ad un filtro specifico e rappresenta la risposta di quel filtro all'immagine di input. Negli strati iniziali della rete, le feature maps possono catturare caratteristiche semplici come bordi, linee e angoli, mentre negli strati più profondi possono rappresentare pattern più complessi come forme, texture o addirittura oggetti interi.

La profondità dei canali si riferisce al numero di feature maps prodotte da uno strato convoluzionale. Aumentare il numero di feature maps consente alla rete di apprendere caratteristiche più complesse e astratte, ma incrementa anche il costo computazionale e può portare ad overfitting se la rete è troppo grande per i dati disponibili. Un aspetto cruciale è che la profondità di un filtro deve corrispondere alla profondità dell'input. Ad esempio, per un'immagine RGB (3 canali), ogni filtro deve avere profondità 3. L'output di ogni convoluzione è una feature map 2D, indipendentemente dalla profondità dell'input.

5.2.3 Convoluzioni 1D, 2D e 3D

Le CNN possono operare su dati di diversa dimensionalità, richiedendo tipi specifici di convoluzione:

- **Convoluzione 1D:** viene utilizzata per dati sequenziali come segnali temporali, dati finanziari, segnali biomedici (EEG, ECG) o testi. Il kernel si muove lungo una sola direzione (asse temporale), producendo un output 1D.
- **Convoluzione 2D:** è la più comune nelle CNN per Computer vision. Il kernel si muove in due direzioni (x e y) attraverso l'immagine, producendo feature maps 2D. Anche quando l'input è 3D (come un'immagine RGB), la convoluzione opera ancora in 2D poiché la profondità del filtro deve corrispondere al numero di canali dell'input.
- **Convoluzione 3D::** è utilizzata per dati volumetrici come video, scansioni MRI o TAC. Il kernel si muove in tre direzioni (x,y,z), producendo un output 3D. È essenziale che la profondità del filtro sia minore di quella dell'input per generare un output volumetrico.

5.3 Pooling e normalizzazione

5.3.1 Max pooling vs average pooling

Il pooling è un'operazione di downsampling che riduce le dimensioni spaziali delle feature maps mantenendo le informazioni più importanti. Questa operazione migliora l'efficienza computazionale e introduce una forma di invarianza alle traslazioni, rendendo la rete meno sensibile a piccoli spostamenti nell'input.

Il max pooling seleziona il valore massimo all'interno di ogni finestra di pooling. È particolarmente efficace nel preservare le caratteristiche più importanti e nell'introdurre invarianza alle traslazioni. Ad esempio, con una finestra 2×2 e stride 2, il max pooling riduce le dimensioni dell'input della metà mantenendo le attivazioni più forti.

L'average pooling calcola la media dei valori all'interno di ogni finestra di pooling. Mentre preserva più informazione locale rispetto al max pooling, può essere meno efficace nel gestire variazioni sottili delle caratteristiche o features significative in certe regioni dell'immagine.

5.3.2 Global pooling

Il global pooling è una variante che riduce ogni feature map ad un singolo valore, eliminando completamente le dimensioni spaziali. Il global max pooling seleziona il valore massimo da ogni feature map intera, mentre il global average pooling calcola la media di tutti i valori in ogni feature map. Questa tecnica è spesso utilizzata prima degli strati fully connected finali nelle architetture CNN per la classificazione, riducendo drasticamente il numero di parametri e prevenendo l'overfitting.

5.3.3 Batch, Layer e Group normalization

La batch normalization è una tecnica introdotta per accelerare l'addestramento delle reti neurali profonde e ridurre la sensibilità all'inizializzazione dei parametri. Normalizza gli input di ogni strato utilizzando la media e varianza del mini-batch corrente durante l'addestramento.

Per ogni canale, durante la fase di apprendimento, la batch normalization calcola:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

dove μ_B e σ_B^2 sono la media e varianza del batch, ed ϵ è una costante piccola per evitare divisioni per zero. Successivamente applica una trasformazione affine apprendibile:

$$y_i = \gamma \hat{x}_i + \beta$$

dove γ e β sono parametri apprendibili.

La layer normalization normalizza tutti i neuroni in un particolare strato per ogni input individualmente, rendendola indipendente dalla dimensione del batch.

La group normalization divide i canali in gruppi e normalizza all'interno di ogni gruppo, offrendo un compromesso tra batch e layer normalization.

5.4 Data augmentation: rotazioni, zoom, colour jitter

La data augmentation è una tecnica che aumenta artificialmente le dimensioni del dataset applicando trasformazioni realistiche agli esempi di training, migliorando la generalizzazione e riducendo l'overfitting.

Le rotazioni ruotano le immagini di angoli casuali (tipicamente tra -50° e 50°), aiutando la rete a riconoscere oggetti indipendentemente dal loro orientamento. Studi hanno dimostrato che la rotazione può migliorare significativamente le prestazioni.

Lo zoom (o scaling) modifica le dimensioni degli oggetti nell'immagine, permettendo alla rete di riconoscere oggetti a diverse scale. Il random crop è una variante che estrae porzioni casuali dell'immagine originale.

Il colour jitter modifica luminosità, contrasto, saturazione e tonalità delle immagini. Questa tecnica varia i canali RGB con valori casuali, producendo cambiamenti casuali nel colore che aiutano la rete a essere invariante alle variazioni di illuminazione e colore.

5.5 Funzionamento delle CNN

5.5.1 Forward pass

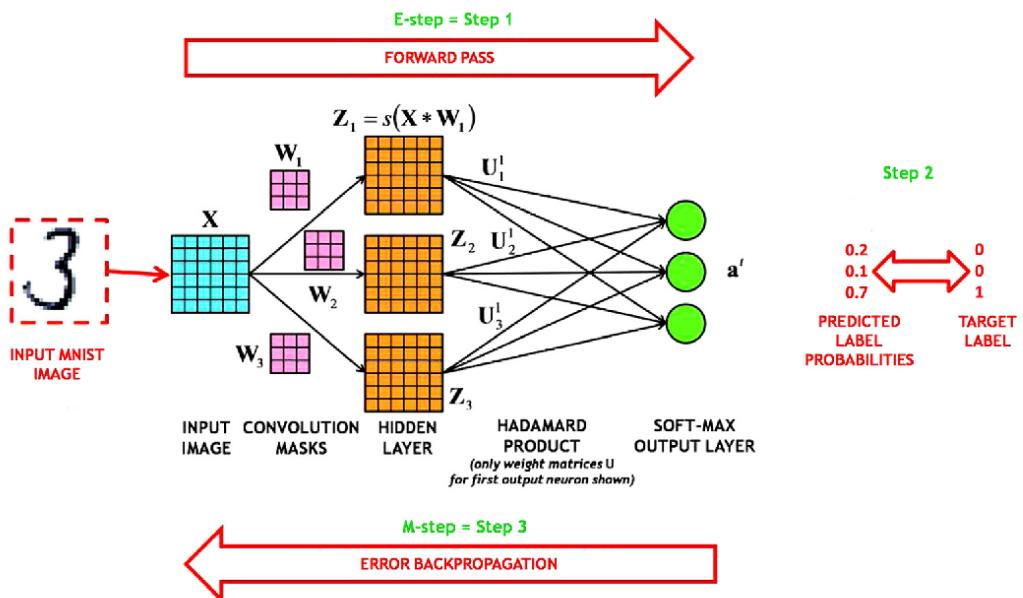


Figura 5.2: Diagramma del forward pass in una CNN.

Il forward pass rappresenta il percorso che i dati seguono dall'input verso l'output attraverso l'intera architettura della rete. Questo processo inizia con l'immagine grezza e termina con la predizione finale, passando attraverso una serie di trasformazioni matematiche che estraggono progressivamente caratteristiche sempre più complesse.

5.5.2 Strati di convoluzione

Il primo stadio, per l'elaborazione delle immagini, coinvolge gli strati convoluzionali, che rappresentano il cuore dell'architettura CNN. Quando un'immagine di input entra nella rete, essa viene elaborata attraverso un insieme di filtri (kernel) che eseguono l'operazione di convoluzione. Ogni filtro è responsabile del rilevamento di una specifica caratteristica: negli

strati iniziali, questi filtri apprendono a riconoscere caratteristiche di basso livello come bordi, linee e angoli. L'operazione di convoluzione produce le feature maps, che rappresentano la risposta di ciascun filtro all'immagine di input. Ogni elemento nella feature map indica l'intensità della presenza di quella specifica caratteristica in quella posizione dell'immagine.

5.5.3 Funzioni di attivazione

Dopo ogni operazione di convoluzione, viene applicata una funzione di attivazione, tipicamente ReLU. Questa fase è cruciale perché introduce non-linearietà nel modello, permettendo alla rete di apprendere relazioni complesse nei dati.

5.5.4 Strati di pooling

Successivamente agli strati convoluzionali, i dati passano attraverso gli strati di pooling. Questi strati eseguono un'operazione di downsampling che riduce le dimensioni spaziali delle feature maps mantenendo le informazioni più importanti.

5.5.5 Gerarchia delle caratteristiche

Man mano che i dati attraversano strati successivi, si verifica un fenomeno fondamentale: la costruzione gerarchica delle caratteristiche. Gli strati iniziali rilevano caratteristiche elementari (bordi, texture), gli strati intermedi combinano queste caratteristiche per formare pattern più complessi (forme geometriche, motivi), mentre gli strati più profondi assemblano questi pattern in rappresentazioni di alto livello che corrispondono a parti di oggetti o oggetti interi.

5.5.6 Strati fully-connected

Dopo l'estrazione gerarchica delle caratteristiche, i dati raggiungono gli strati fully-connected. Prima di entrare in questi strati, le mappe di caratteristiche

multidimensionali vengono convertite in un vettore unidimensionale. Negli strati fully-connected, ogni neurone è collegato a tutti i neuroni dello strato precedente, consentendo alla rete di combinare tutte le caratteristiche estratte per prendere la decisione finale, in base al tipo di problema.

Durante il terzo caso di studio, verranno esplorate le modifiche agli strati fully-connected delle CNN, analizzando due diverse architetture: MLP e KAN.

5.5.7 Flusso informativo e Trasformazioni progressive

Durante tutto questo processo, l'immagine originale, inizialmente rappresentata come una matrice di valori pixel, viene gradualmente trasformata in rappresentazioni sempre più astratte e significative dal punto di vista semantico. Ogni strato della rete contribuisce a questa trasformazione: gli strati convoluzionali estraggono e raffinano le caratteristiche, gli strati di pooling riducono la complessità computazionale e introducono invarianza, mentre gli strati fully-connected integrano tutte le informazioni per la classificazione finale.

Questo design architetturale permette alle CNN di apprendere automaticamente le rappresentazioni ottimali per il compito specifico, eliminando la necessità di progettare manualmente gli estrattori di caratteristiche. La capacità di costruire rappresentazioni gerarchiche rende le CNN particolarmente efficaci per compiti di computer vision, dove la comprensione dell'immagine richiede l'integrazione di informazioni a diversi livelli di astrazione.

5.6 Vantaggi

Le CNN si distinguono per una serie di vantaggi che le rendono lo standard per l'elaborazione delle immagini. Il loro punto di forza principale è il **rilevamento automatico delle caratteristiche**. A differenza delle reti neurali tradizionali che richiedono un'estrazione manuale delle feature, le CNN apprendono e identificano autonomamente le caratteristiche rilevanti

direttamente dai dati grezzi. Questo approccio riduce drasticamente lo sforzo di pre-processing e permette al modello di adattarsi in modo più efficace ai dati.

Un'altra caratteristica distintiva è la loro **efficienza computazionale e riduzione dei parametri**. L'uso del weight sharing (una tecnica che permette ad un singolo filtro di rilevare la stessa caratteristica in qualsiasi posizione dell'immagine utilizzando lo stesso set di pesi) e degli strati di pooling riduce notevolmente il numero di parametri da addestrare rispetto alle reti fully-connected. Questo non solo accelera l'addestramento, ma contribuisce anche a prevenire l'overfitting.

Grazie all'operazione di convoluzione, le CNN offrono **invarianza alla traslazione**. Sono in grado di riconoscere un oggetto indipendentemente dalla sua posizione nell'immagine. Un filtro che ha imparato a riconoscere un occhio, ad esempio, lo riconoscerà sia che si trovi in alto a sinistra che in basso a destra.

Le architetture CNN sono anche estremamente **scalabili**: possono essere adattate facilmente a dataset di grandi dimensioni ed a compiti complessi, aumentando la profondità e la larghezza della rete per gestire immagini ad alta risoluzione o per apprendere pattern più astratti.

5.7 Svantaggi

Nonostante i loro numerosi vantaggi, le CNN presentano alcune limitazioni. La principale è la loro **dipendenza da grandi dataset**. Specialmente le architetture più complesse richiedono enormi quantità di dati etichettati per un addestramento efficace. La mancanza di un dataset sufficientemente grande può portare all'overfitting o ad una scarsa capacità di generalizzazione.

Un altro limite è l'**invarianza limitata**. Le CNN standard sono invarianti alla traslazione, ma non lo sono rispetto ad altre trasformazioni geometriche. Se un'immagine viene ruotata o scalata in modo significativo, il modello potrebbe non riconoscerla correttamente, a meno che non si usino tecniche

di data augmentation per esporre il modello a tali variazioni durante l'addestramento.

Le CNN sono spesso criticate per la loro **mancanza di interpretazione**. È difficile capire il motivo per cui un modello prenda una determinata decisione. Le feature map intermedie possono essere visualizzate, ma il processo decisionale complessivo rimane una "black box", rendendo complicato il debugging e l'adozione in settori critici come la medicina, dove è essenziale la trasparenza.

Infine, l'addestramento di architetture CNN molto profonde ha un elevato **costo computazionale**. Spesso richiede una notevole potenza di calcolo e hardware specializzato come le GPU. Anche l'inferenza su dispositivi a bassa potenza può risultare problematica.

Capitolo 6

Ottimizzazione degli iperparametri

6.1 Introduzione

Questo capitolo presenta una panoramica delle metodologie di ottimizzazione degli iperparametri, essenziali per migliorare le performance dei modelli di Machine e Deep Learning. L'obiettivo è esplorare come queste tecniche permettano di navigare lo spazio delle configurazioni di un modello per trovare la combinazione ideale. Il testo inizia descrivendo diverse forme di Cross-Validation, tra cui la K-fold CV, la Nested Cross-Validation e la Time Series Cross-Validation, pensata specificamente per i dati temporali. Successivamente, il capitolo si concentra sulle strategie di ricerca, a partire dai metodi più semplici come il Grid Search e il Random Search, per poi introdurre approcci più sofisticati come l'Ottimizzazione Bayesiana e gli Algoritmi Genetici. Infine, viene fornito un confronto pratico tra questi metodi per aiutare a comprendere quando applicare ciascuno di essi. Per la sua efficienza e scalabilità, in questa ricerca è stato scelto il Random Search, che è stato ottimizzato utilizzando una formula che utilizza le probabilità per determinare il numero ideale di iterazioni.

6.2 Cross-Validation (CV)

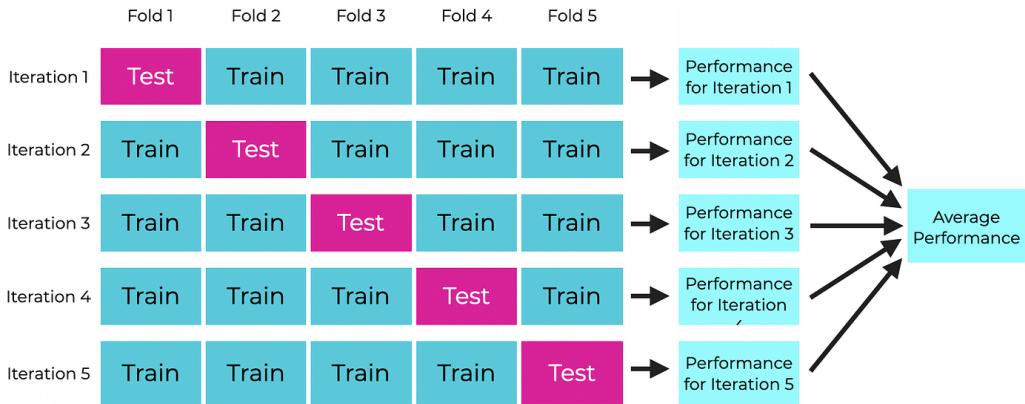


Figura 6.1: Cross-Validation.

La CV è una tecnica statistica per la valutazione delle prestazioni di un modello predittivo. Il suo scopo principale è quello di stimare quanto un modello è in grado di generalizzare su dati indipendenti non visti durante la fase di training. Dato un dataset $D = \{z_1, \dots, z_N\}$ e una procedura di training che produce un modello \hat{f}_D , la K-fold cross-validation divide i dati in K parti disgiunte (fold), dove ognuno ha una dimensione approssimativamente uguale. Il processo si ripete K volte. Per ogni iterazione $k \in \{1, \dots, K\}$:

- Si usano le restanti $K - 1$ parti, che costituiscono il training set $D^{(k)} = D \setminus D_k$, per allenare il modello. Questo produce un modello parziale $\hat{f}_{D^{(k)}}$.
- Si valuta l'errore del modello $\hat{f}_{D^{(k)}}$ sul fold lasciato fuori D_k , che ricopre il ruolo del validation set per questa iterazione. L'errore viene calcolato come $E_k = \text{Errore}(\hat{f}_{D^{(k)}}, D_k)$.

Al termine delle K iterazioni, si ottiene una lista di K errori $\{E_1, E_2, \dots, E_K\}$. La stima finale della performance del modello è data dalla media degli errori calcolati su ogni fold, $\bar{E} = \frac{1}{K} \sum_{k=1}^K E_k$.

6.2.1 Time Series Cross-Validation (TSCV)

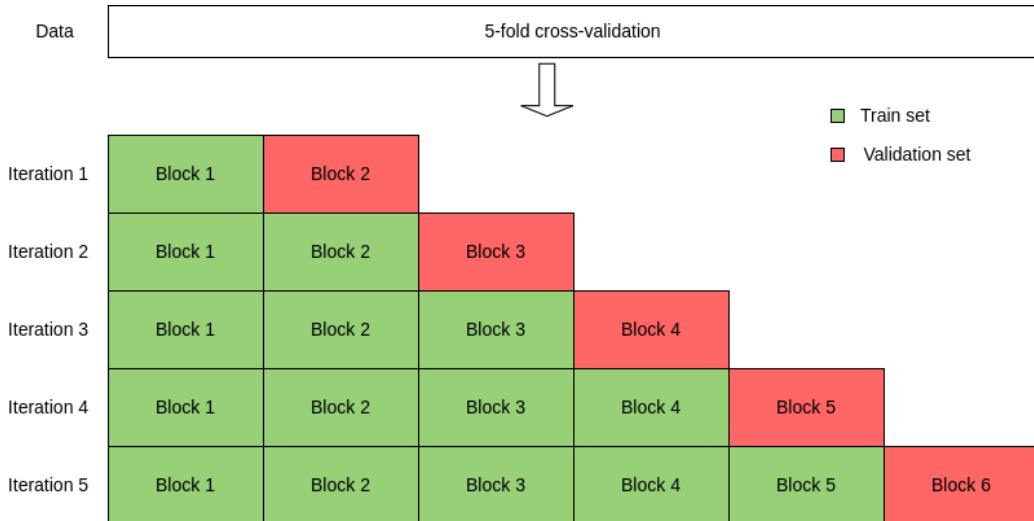


Figura 6.2: Time Series Cross-Validation.

Per i dati che hanno una dipendenza temporale, come le serie storiche, la CV standard non è adatta. Suddividere i dati in fold casuali romperebbe la struttura temporale, e l’addestramento su dati futuri per testare il modello su dati passati (fenomeno noto come data leakage) non avrebbe senso pratico e porterebbe a risultati fuorvianti. La TSCV, sviluppata per risolvere questa problematica, crea un training set che cresce sequenzialmente nel tempo, ed il validation set è sempre un blocco di dati che segue immediatamente il dataset di allenamento. Il processo funziona così:

- **Prima iterazione:** il modello è addestrato sui primi m punti temporali e testato sui successivi n punti.
- **Seconda iterazione:** il modello è addestrato sui primi $m + n$ punti temporali e testato sui successivi n punti.
- **Iterazioni successive:** il processo continua, con il training set che si espande ad ogni passo ed il validation set che avanza nel tempo.

Questo approccio rispecchia fedelmente lo scenario reale in cui un modello di serie storica viene addestrato su dati passati e utilizzato per fare previsioni

su dati futuri non ancora visti. La media degli errori di validazione su tutte le iterazioni fornisce una stima robusta e realistica delle prestazioni del modello.

6.3 Nested Cross-Validation (NCV)

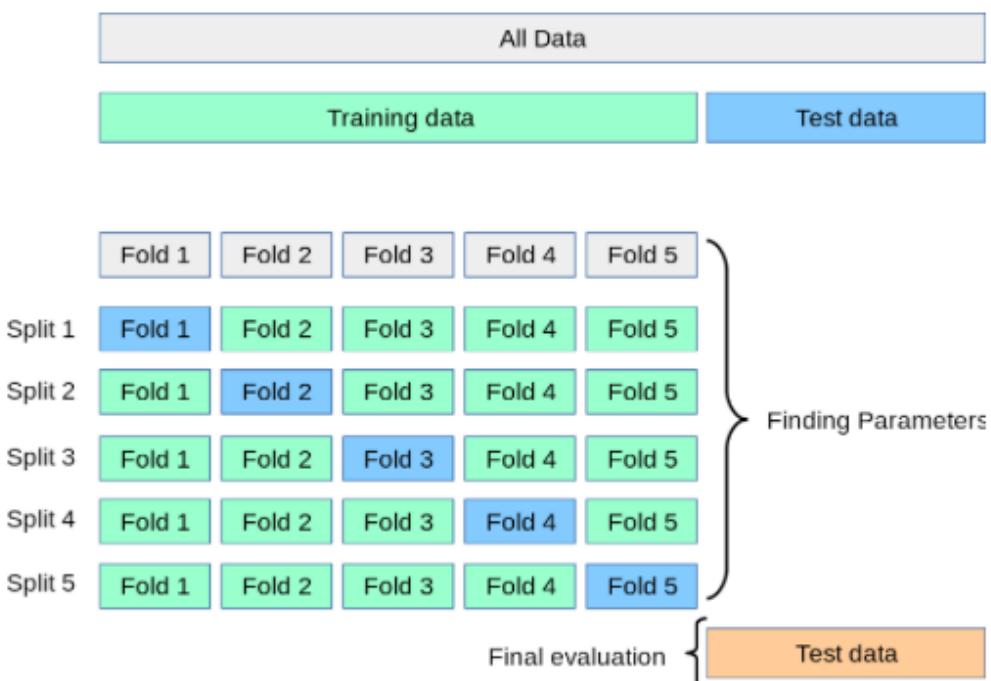


Figura 6.3: Nested Cross-Validation.

La NCV è un'estensione della classica CV. Il suo scopo principale è quello di fornire una stima imparziale ed affidabile dell'errore di generalizzazione di un modello, risolvendo il problema del bias di selezione che può verificarsi quando gli stessi dati vengono utilizzati sia per la scelta degli iperparametri che per la valutazione finale del modello. L'idea alla base della NCV è quella di creare due cicli di CV: un ciclo esterno ed uno interno.

- 1. Ciclo esterno (Outer loop):** ha il compito di stimare l'errore di generalizzazione del modello. Il dataset viene diviso in K fold. Per

ogni iterazione di questo ciclo, vengono utilizzati $K - 1$ parti per costruire il training set esterno ed il restante fold agisce da test set finale, che non verrà mai utilizzato per la selezione degli iperparametri, garantendo una valutazione finale imparziale.

2. **Ciclo interno (Inner loop):** all'interno di ogni iterazione del ciclo esterno, si esegue un altro ciclo di CV (solitamente con L fold) sul training set esterno. Questo ciclo interno è dedicato esclusivamente all'ottimizzazione degli iperparametri. Per ogni combinazione di essi da testare (ad esempio, utilizzando Grid o Random Search), si addestra il modello sui $L - 1$ fold interni e si valuta la sua performance sul fold interno rimanente. La combinazione di iperparametri che ottiene la migliore performance media su tutte le L iterazioni viene selezionata.
3. **Valutazione del modello ottimizzato:** una volta trovata la migliore combinazione di iperparametri nel ciclo interno, il modello viene addestrato nuovamente sull'intero training set esterno utilizzando proprio quella combinazione ottimale. Infine, la performance di questo modello viene valutata sul test set esterno, che è stato lasciato fuori all'inizio dell'iterazione K . L'errore ottenuto in questa fase è la stima delle prestazioni di generalizzazione del modello per quella specifica iterazione del ciclo esterno.

Questo processo viene ripetuto per tutti i K fold del ciclo esterno. La stima finale dell'errore del modello è la media dei K errori ottenuti sui test set esterni.

6.4 Grid search (GS)

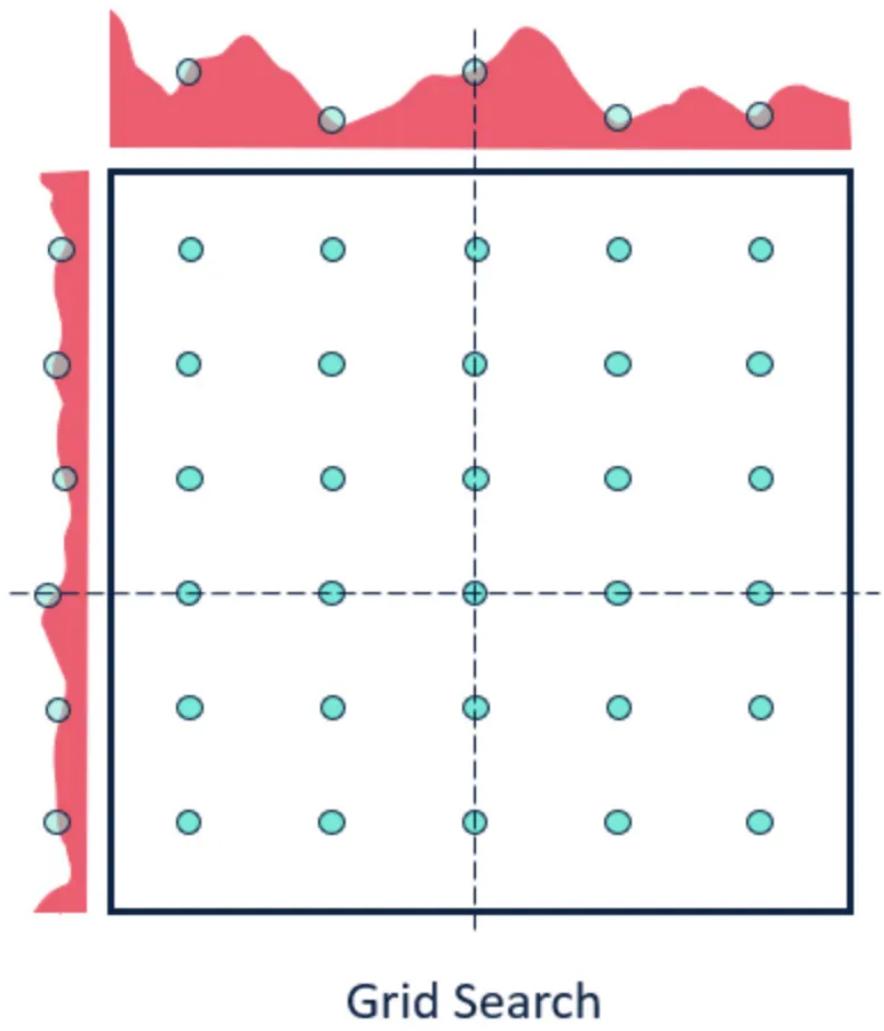


Figura 6.4: Grid search.

6.4.1 Spiegazione dell'algoritmo

Il GS consiste nel definire una griglia discreta di possibili valori per ciascun iperparametro e nell'eseguire una valutazione esaustiva del modello per

ogni combinazione. Alla fine, si seleziona il set di iperparametri che ottimizza la metrica di interesse. Il metodo non introduce casualità, risultando completamente ripetibile e deterministico.

6.4.2 Vantaggi

L'approccio del Grid Search offre diversi vantaggi. Primo fra tutti, la sua natura **esaustiva**: esplora tutte le combinazioni predefinite nello spazio di ricerca, permettendo di trovare l'ottimo globale se questo è incluso nella griglia. Inoltre, è un metodo **deterministico e riproducibile**, dato che l'assenza di casualità assicura che ogni esecuzione dell'algoritmo fornisca risultati replicabili. Infine, la sua **semplicità di implementazione** lo rende ideale in spazi di ricerca ridotti e ben definiti, o come baseline quando si dispone di elevate risorse computazionali.

6.4.3 Limiti

Nonostante i suoi vantaggi, il Grid Search presenta anche dei limiti significativi. Il suo **costo computazionale è esponenziale**, a causa della curse of dimensionality, rendendolo impraticabile per spazi di ricerca ampi o ad alta dimensionalità. Il metodo è spesso **inefficiente**, poiché molte valutazioni potrebbero riguardare regioni poco promettenti, specialmente quando solo alcuni parametri influenzano la performance ottimale. **La discretizzazione e la perdita di ottimi** sono altri problemi rilevanti: la necessità di fissare una griglia per iperparametri continui può portare a saltare valori potenzialmente migliori che non sono inclusi. Infine, non è **adatto a modelli e dataset complessi**, poiché il costo aumenta drasticamente con la complessità e la dimensione del dataset.

6.5 Random Search (RS)

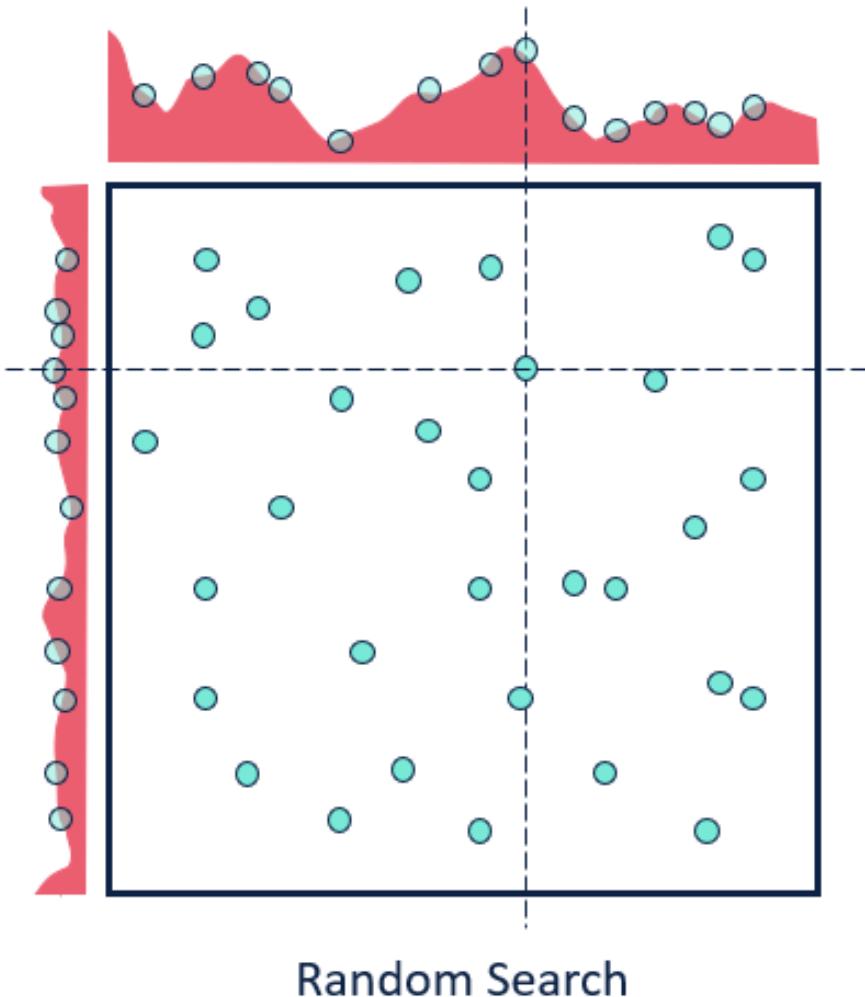


Figura 6.5: Random search.

6.5.1 Spiegazione dell'algoritmo

Il RS seleziona casualmente N configurazioni dagli intervalli o distribuzioni scelte per ciascun iperparametro, valutando il modello solo in quei punti. La campionatura può avvenire secondo distribuzioni uniformi, log-uniformi o

guidate da conoscenze pregresse. Questo approccio si basa, quindi, sulla randomizzazione invece che su una griglia predefinita.

6.5.2 Vantaggi

Il Random Search presenta numerosi vantaggi, tra cui l'**efficienza su spazi estesi**. Campionando casualmente, si ha una maggiore probabilità di individuare combinazioni efficaci, specialmente quando solo pochi parametri sono percepiti come determinanti per le prestazioni. Un altro punto di forza è la sua **scalabilità**: il numero di valutazioni può essere fissato (es. $N=100$), spesso ottenendo performance simili a quelle del Grid Search con molte meno combinazioni. L'algoritmo è anche **facile da parallelizzare**, poiché ogni valutazione è indipendente, consentendo l'esecuzione in parallelo. Infine, è molto **adattabile**, in quanto è possibile scegliere distribuzioni di campionamento informate da conoscenze a priori.

6.5.3 Limiti

Tra i limiti del Random Search, il più evidente è la sua natura **non sistematica**, che non esplora esaustivamente lo spazio delle ipotesi e può quindi saltare l'ottimo globale. Le prestazioni dipendono in gran parte dalla **distribuzione di campionamento** scelta; campionamenti mal scelti possono ignorare regioni promettenti. Inoltre, i risultati **non sono ripetibili** a meno che non si fissi un random seed. Infine, su spazi di ricerca piccoli e ben definiti, il Grid Search può risultare più efficace.

6.6 Bayesian optimization (BO)

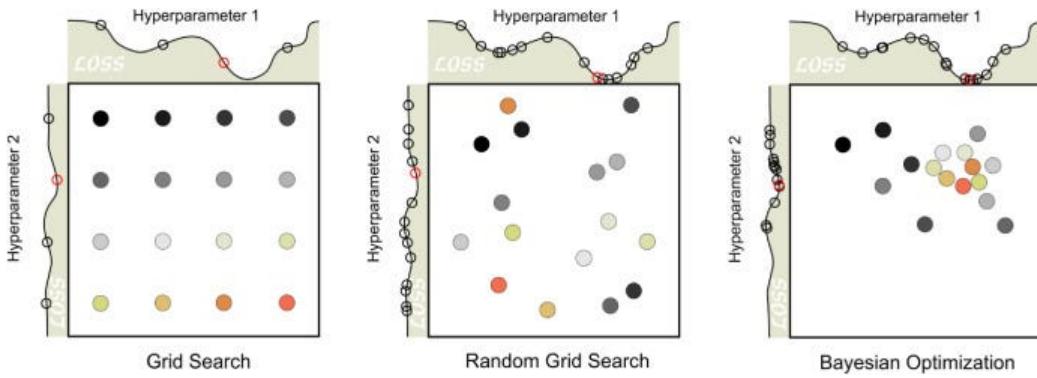


Figura 6.6: Bayesian optimization.

6.6.1 Spiegazione dell'algoritmo

La BO interpreta l'ottimizzazione degli iperparametri come la ricerca del massimo/minimo di una funzione obiettivo costosa ed ignota. Si costruisce un modello surrogato probabilistico (come un Gaussian Process, GP) che stima la funzione obiettivo e quantifica l'incertezza predittiva. Ad ogni iterazione, una funzione di acquisizione determina il prossimo punto da valutare.

Modello surrogato: Gaussian Process (GP)

Un GP definisce, per ogni punto λ , una distribuzione normale per il valore $f(\lambda)$ con media $\mu(\lambda)$ e varianza $\sigma^2(\lambda)$. Dopo aver osservato alcune valutazioni, si aggiorna μ e σ per riflettere ciò che si è imparato.

Le fasi del BO

1. **Campionamento iniziale:** Si comincia con una serie di prove iniziali, selezionando casualmente diverse combinazioni di iperparametri. Per ogni combinazione, si addestra il modello e si calcola una metrica di performance, come la precisione (accuracy) o l'errore quadratico medio (MSE), che funge da risultato della funzione obiettivo.

2. **Modello surrogato:** sulla base dei risultati del campionamento iniziale, si costruisce un modello probabilistico, solitamente un Gaussian process, che approssima la funzione obiettivo. Questo modello non solo predice la performance attesa per una data combinazione di iperparametri, ma fornisce anche un'incertezza sulla predizione.
3. **Funzione di acquisizione:** viene usata per decidere la prossima combinazione di iperparametri da testare. Questa funzione bilancia l'esplorazione (provare combinazioni di cui si sa poco, per ridurre l'incertezza) e lo sfruttamento (provare combinazioni che il modello surrogato ritiene promettenti, per migliorare la performance).
4. **Valutazione della performance:** la nuova combinazione di iperparametri viene utilizzata per addestrare il modello e valutarne le prestazioni. Il risultato di questa valutazione rappresenta il nuovo punto dati che viene aggiunto al nostro set di informazioni.
5. **Aggiornamento del modello surrogato:** viene aggiornato con i nuovi risultati. Questo affina le sue predizioni e riduce l'incertezza, permettendo alla funzione di acquisizione di prendere decisioni più informate nelle iterazioni successive.
6. **Ripetizione:** i passaggi 3, 4 e 5 vengono ripetuti. Il ciclo si ferma quando si raggiunge un criterio predefinito, come un budget di tempo, un numero massimo di iterazioni, o quando la performance del modello smette di migliorare significativamente.

6.6.2 Vantaggi

La Bayesian optimization è un metodo molto **efficiente**, riducendo drasticamente il numero di valutazioni necessarie, il che la rende ideale per funzioni costose. La sua funzione di acquisizione permette un eccellente **bilanciamento tra exploration ed exploitation**, permettendo di esplorare nuove regioni e raffinare quelle già note. Inoltre, il modello surrogato

offre una **modellazione dell'incertezza**, che indirizza la ricerca nelle zone potenzialmente più promettenti.

6.6.3 Limiti

Tra i limiti della BO, si include l'**overhead computazionale** dovuto al mantenimento e all'aggiornamento del modello surrogato. La sua natura sequenziale rende più **complessa la parallelizzazione** rispetto a Random o Grid Search. Infine, la sua **scalabilità è limitata**: pur essendo efficiente su spazi continui di media dimensione, l'ottimizzazione può diventare difficoltosa su spazi molto ampi.

6.7 Genetic algorithm (GA)

Genetic Algorithms

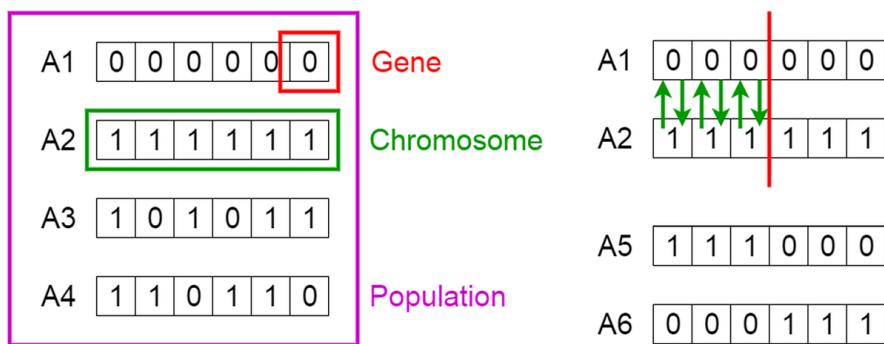


Figura 6.7: Genetic algorithm.

6.7.1 Spiegazione dell'algoritmo

Gli algoritmi genetici (GA) sono metodi di ottimizzazione di iperparametri, ispirati al processo di selezione naturale. Funzionano mantenendo una popolazione di soluzioni e migliorandole nel tempo tramite un processo iterativo. Ad ogni iterazione, detta anche generazione, vengono applicati tre operatori evolutivi principali:

- **Selezione:** gli individui con un punteggio di fitness più alto (le soluzioni "migliori" o più "adatte") vengono scelti per la riproduzione. Questo processo assicura che le caratteristiche positive si diffondano nella popolazione.
- **Crossover:** le soluzioni selezionate vengono combinate per creare nuovi individui "figli". Questo operatore permette di esplorare nuove combinazioni e di mescolare le caratteristiche delle soluzioni migliori.
- **Mutazione:** vengono introdotte piccole, casuali variazioni nelle nuove soluzioni. La mutazione è fondamentale per mantenere la diversità genetica e per evitare che l'algoritmo rimanga bloccato in un'unica soluzione.

Questi operatori permettono, agli algoritmi genetici, di esplorare un vasto spazio di soluzioni (esplorazione globale) ed, allo stesso tempo, di migliorare le soluzioni promettenti (esplorazione locale), migliorandole sempre di più nel corso delle generazioni.

Le fasi del GA

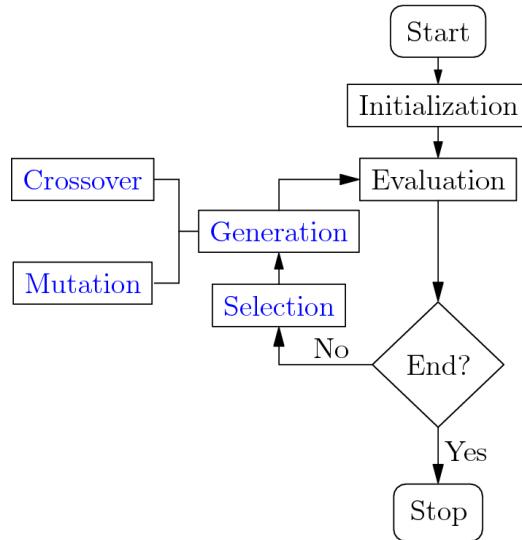


Figura 6.8: Le fasi del processo del Genetic Algorithm.

- **Inizializzazione:**

- Definire lo spazio di ricerca degli iperparametri.
- Generare una popolazione iniziale di N individui casuali.
- Impostare i parametri: dimensione popolazione (N), generazioni massime (G_{max}), tassi di crossover (p_c) e mutazione (p_m).

- **Ciclo evolutivo** (per ogni generazione $g = 1, 2, \dots, G_{max}$):

- **Valutazione:** allenare il modello per ogni individuo e calcolare il fitness $f(x_i)$.
- **Selezione:** scegliere i genitori migliori.
- **Crossover:** combinare coppie di genitori con probabilità p_c per generare figli.
- **Mutazione:** introdurre variazioni casuali nei figli con probabilità p_m .
- **Sostituzione:** formare la nuova popolazione (strategia elitista o generazionale).

- **Terminazione:**
 - Fermarsi quando: raggiunto G_{max} , nessun miglioramento per k generazioni, o fitness target raggiunto.
 - Restituire l'individuo con il miglior fitness come soluzione ottimale.

6.7.2 Vantaggi

Gli algoritmi genetici offrono una **esplorazione robusta** grazie a crossover e mutazioni, il che li rende adatti a spazi complessi e non lineari per trovare ottimi globali. Sono inoltre efficaci nella **gestione di spazi misti**, lavorando bene con iperparametri discreti, continui e categorici. Un altro vantaggio è la loro **parallelizzazione**, in quanto le valutazioni del fitness possono essere distribuite in parallelo.

6.7.3 Limiti

Tra gli svantaggi, si evidenzia il **costo computazionale elevato**, poiché richiedono molte generazioni e valutazioni. I risultati dipendono anche da **iperparametri evolutivi**, come la dimensione della popolazione e i tassi di mutazione/crossover, che devono essere a loro volta ottimizzati. Infine, non c'è una **garanzia di convergenza** all'ottimo globale, dato che l'algoritmo potrebbe bloccarsi in minimi sub-ottimali se la diversità non viene mantenuta.

6.8 Confronto pratico

6.8.1 Criteri per la scelta

Nella scelta del metodo di ottimizzazione degli iperparametri, è fondamentale valutare diversi criteri per identificare la soluzione più adatta ad un problema specifico. I criteri principali includono **l'efficienza**, che si riferisce al numero di valutazioni necessarie per trovare una buona soluzione; **la**

scalabilità, ovvero il comportamento dell'algoritmo all'aumentare della dimensionalità degli iperparametri; **il supporto per vari tipi di variabili**, che possono essere continue, discrete o categoriche; **la parallelizzazione**, ovvero la facilità con cui l'algoritmo può essere eseguito su cluster o GPU; **la robustezza**, ovvero la capacità di gestire rumore e funzioni complesse; e infine **le risorse computazionali richieste**, che considerano l'overhead e i costi aggiuntivi dell'algoritmo.

6.8.2 Tabella riassuntiva comparativa

Metodo	Efficienza	Scalabilità	Parallelizz.	Complessità	Spazi misti
Grid search	Bassa	Pessima	Eccellente	Bassa	Bassa
Random search	Media	Buona	Eccellente	Bassa	Bassa
Bayesian opt.	Alta	Limitata	Moderata	Alta	Media
Genetic alg.	Variabile*	Discreta	Eccellente	Media-Alta	Alta

* può essere molto efficace su spazi complessi/discreti, ma richiede molte valutazioni rispetto a BO in spazi piccoli; meno efficiente se valutazioni sono molto care.

6.8.3 Matrice decisionale per la scelta del metodo

Condizioni	Metodo consigliato	Alternativa	Da evitare
Spazi piccoli (2-3 param.)	Grid Search	Random Search	Bayesian Opt.
Spazi medi/ampi (> 4 param.)	Random Search	Bayesian Opt.	Grid Search
Valutazioni costose, budget limitato	Bayesian Opt.	Random Search	Grid Search
Spazi molto complessi	Genetic Algorithms	Bayesian Opt.	Grid Search
Multi-obiettivo	Genetic Algorithms	-	Grid/Random Search
Risorse limitate	Random Search	Bayesian Opt.	Grid Search

6.8.4 Scelta per i casi studio: Random search

Motivazioni della scelta

Per i casi studio, si è scelto di adottare il Random Search per diverse ragioni. Questo approccio garantisce una notevole **efficienza su spazi ampi**,

offrendo prestazioni superiori al Grid Search quando l'impatto di alcuni iperparametri è marginale. Inoltre, la sua **scalabilità** lo rende immune all'aumento esponenziale della complessità, mantenendo performance elevate anche con un gran numero di iperparametri. A livello implementativo, la sua **semplicità** lo rende un metodo ideale per l'implementazione e la parallelizzazione, riducendo la complessità del codice e facilitando l'esecuzione su cluster e GPU. Un'altra motivazione importante è la sua **flessibilità operativa**, che permette interruzioni anticipate tramite Early stopping ed un facile riutilizzo dei risultati per analisi successive. Infine, il Random Search offre il **miglior rapporto costo-beneficio**, bilanciando efficienza esplorativa e semplicità computazionale, rendendolo la scelta ideale per il contesto di questa tesi.

6.8.5 Ottimizzazione del numero di iterazioni nel Random search

Per massimizzare l'efficienza del Random search, è fondamentale stimare il numero minimo di iterazioni necessarie per garantire un'alta probabilità di trovare configurazioni quasi-ottimali. Questo approccio consente di bilanciare l'efficacia esplorativa con il costo computazionale, evitando valutazioni superflue.

Derivazione teorica della formula

La stima del numero di iterazioni richieste si basa sulla teoria della probabilità discreta. Il processo di derivazione segue una logica chiara, partendo dalla probabilità di fallimento in un singolo tentativo. Se in uno spazio di ricerca con M configurazioni totali ne esistono k che consideriamo "quasi-ottimali", la probabilità di non selezionarne una in un singolo campionamento casuale è:

$$P(\text{fallimento singolo}) = 1 - \frac{k}{M}$$

La probabilità di fallire in tutti gli n tentativi indipendenti è quindi:

$$P(\text{fallimento totale}) = \left(1 - \frac{k}{M}\right)^n$$

Da qui, si ricava la probabilità di successo, ovvero di trovare almeno una configurazione top- k in n tentativi:

$$P(\text{successo}) = 1 - \left(1 - \frac{k}{M}\right)^n$$

Infine, per determinare il numero di iterazioni n necessarie per raggiungere una probabilità di successo P desiderata, si risolve la formula per n :

$$1 - \left(1 - \frac{k}{M}\right)^n = P \implies n = \frac{\ln(1 - P)}{\ln\left(1 - \frac{k}{M}\right)}$$

Quando il numero delle migliori configurazioni (k) è molto più piccolo del numero totale di configurazioni (M), si può usare l'approssimazione $\ln(1 - x) \approx -x$ per x piccolo. In questo caso, la formula si semplifica in:

$$n \approx -\frac{\ln(1 - P)}{k/M}$$

Vantaggi dell'approccio probabilistico

L'approccio probabilistico al Random search offre notevoli vantaggi. Garantisce un'elevata **efficienza computazionale**, riducendo drasticamente il numero di valutazioni necessarie. Fornisce inoltre un forte **controllo statistico**, offrendo una garanzia probabilistica di trovare soluzioni quasi-ottimali. Grazie alla sua **flessibilità**, permette di modulare il trade-off tra accuratezza desiderata (P) e il costo computazionale (n). Infine, il metodo si distingue per la sua **scalabilità**, adattandosi automaticamente alla dimensione dello spazio di ricerca, un aspetto cruciale in contesti con molti iperparametri.

Capitolo 7

Studio di ablazione e Pruning post-training

Gli studi di ablazione rappresentano una metodologia fondamentale nell’analisi e nell’ottimizzazione dei modelli di machine e deep learning, che consiste nella rimozione temporanea di una componente del modello, come un layer o gruppo di parametri, per osservare l’impatto sulle prestazioni. Si tratta di un esperimento diagnostico che aiuta a comprendere quali elementi contribuiscono maggiormente alla capacità predittiva del modello. In questo contesto, il pruning post-training viene utilizzato come una tecnica complementare che permette non solo di comprendere l’importanza delle componenti del modello, ma anche di ottenere versioni più efficienti senza compromettere significativamente le prestazioni.

7.1 Studi di ablazione

7.1.1 Definizione

Per formalizzare matematicamente il concetto di ablazione, consideriamo un modello f_θ dove θ rappresenta l’insieme completo dei suoi parametri, e $L(\theta)$ indica la funzione di perdita del modello. Dato un sottoinsieme

specifico di parametri S , la variazione di performance dovuta all’ablazione può essere quantificata come:

$$\Delta_S = L(\theta_{-S}) - L(\theta)$$

In questa formulazione, θ_{-S} rappresenta il modello con la componente S rimossa. Un valore elevato di Δ_S indica che la componente S fornisce un contributo importante alle prestazioni del modello. Questa definizione matematica fornisce una base quantitativa per valutare l’importanza relativa delle diverse componenti del modello.

7.1.2 Benefici

Gli studi di ablazione offrono diversi benefici significativi. Innanzitutto, permettono l’**identificazione delle componenti critiche**, aiutando a comprendere la sensibilità del modello a specifiche parti della sua architettura e rivelando potenziali vulnerabilità o punti di forza. Inoltre, i risultati di questi studi fungono da **guida per la semplificazione** del modello, fornendo una base empirica per le decisioni di pruning. Infine, l’analisi di come la rimozione di una componente specifica influenzi le previsioni facilita l’**interpretazione del comportamento del modello**, contribuendo a una maggiore comprensione dei meccanismi interni che portano alle predizioni finali.

7.2 Pruning post-training

7.2.1 Definizione

Il pruning post-training può essere considerato come l’applicazione di un operatore \mathcal{P} ad un modello già addestrato, che elimina parametri secondo criteri specifici (come l’ampiezza del parametro, l’importanza stimata, o il contributo marginale) per avere un modello più leggero.

La scelta del criterio di pruning influenza significativamente sia l’efficacia della compressione sia il mantenimento delle prestazioni. I criteri più

comuni includono la magnitudine dei parametri, misure di sensibilità basate sui gradienti, e metriche di importanza derivate dall’analisi della struttura del modello.

7.2.2 Benefici

I benefici del pruning sono molteplici. Il primo e più evidente è la **riduzione della memoria e del tempo di inferenza**: il pruning riduce lo spazio di memoria richiesto e il tempo necessario per le predizioni, aspetti critici per il deployment in ambienti con risorse limitate. L’obiettivo primario è il **mantenimento delle prestazioni**, bilanciando l’efficienza e l’accuratezza del modello in modo che le performance rimangano entro una tolleranza accettabile. Infine, riducendo il numero di elementi attivi, il pruning contribuisce a un aumento dell’**interpretabilità**, facilitando l’analisi e la comprensione del contributo delle parti rimanenti.

7.2.3 Trade-off bias-varianza

La riduzione della complessità del modello, attraverso l’eliminazione di parametri, tende a diminuire la varianza, riducendo il rischio di overfitting sui dati di training. Tuttavia, questa semplificazione può introdurre bias, limitando la capacità del modello di catturare pattern complessi nei dati. L’obiettivo pratico del pruning consiste nel trovare il punto ottimale dove la riduzione della varianza compensa l’aumento del bias, mantenendo prestazioni globalmente superiori. Questo equilibrio è altamente dipendente dalla natura dei dati, dalla complessità del task, e dalle caratteristiche specifiche dell’architettura del modello.

7.3 Pruning L1 post-training per MLP e KAN

7.3.1 Definizione

Dato un insieme di parametri del modello $\Theta = \{\theta_1, \theta_2, \dots, \theta_N\}$, la procedura di L1 pruning opera applicando una trasformazione a ciascun parametro θ_i , che è definita da una soglia di potatura λ , che viene tipicamente determinata a livello globale per il modello o a livello locale per ciascun strato.

La formula è:

$$\theta_i^{\text{pruned}} = \begin{cases} 0 & \text{se } |\theta_i| \leq \lambda \\ \theta_i & \text{se } |\theta_i| > \lambda \end{cases}$$

In questa formulazione, se la magnitudine assoluta ($|\theta_i|$) di un parametro è inferiore o uguale alla soglia λ , esso viene permanentemente impostato a zero. Al contrario, se la sua magnitudine è superiore a λ , il parametro viene mantenuto invariato.

7.3.2 Considerazioni specifiche per le KAN

L'applicazione del L1 pruning alle KAN è abbastanza diverso rispetto a MLP. La motivazione principale risiede nella diversa natura dei parametri che compongono questi modelli. Nelle MLP, i parametri sono pesi e bias che operano su connessioni lineari, mentre nelle KAN sono coefficienti che definiscono funzioni univariate (tipicamente B-spline) su ogni arco della rete.

Il pruning su KAN può essere implementato in due modi. Il primo approccio prevede la rimozione di coefficienti locali delle spline, riducendo la risoluzione locale della rappresentazione funzionale mantenendo la struttura generale. Il secondo approccio elimina intere funzioni su specifici archi della rete, semplificando direttamente l'architettura della KAN.

La scelta tra questi approcci deve considerare l'impatto sulla continuità delle funzioni e sulla loro interpretabilità. La rimozione di coefficienti locali mantiene la struttura generale ma può creare delle discontinuità

o irregolarità indesiderate nella curva che la spline rappresenta, mentre l'eliminazione di intere funzioni preserva la continuità locale ma può alterare significativamente la capacità espressiva del modello.

7.4 Pruning per Ensemble: Rank-based pruning per Random forest

7.4.1 Principio fondamentale

Il rank-based pruning per Random forest si basa sul principio che non tutti gli alberi nell'ensemble contribuiscono equamente alle prestazioni finali. Alcuni alberi possono essere ridondanti o addirittura dannosi per la capacità di generalizzazione dell'ensemble, rendendo la loro rimozione vantaggiosa sia in termini di efficienza e prestazioni. L'approccio implementato definisce per ogni albero m un contributo stimato alle prestazioni, quantificato attraverso la variazione di errore ΔL_m che si osserverebbe rimuovendo l'albero dall'ensemble. Gli alberi con contributo minore sono candidati prioritari per la rimozione durante il processo di pruning.

7.4.2 Criterio di ranking basato sulla feature importance

Invece di valutare l'impatto della rimozione di ogni singolo albero sull'errore complessivo, il criterio di ranking si basa sulle feature importance di ogni singolo albero. Specificamente, l'importanza di un albero m viene calcolata come la somma delle feature importance che l'albero utilizza. La logica sottostante è che un albero che si basa su feature più discriminative, che riducono significativamente l'entropia o l'indice di Gini, è probabile che fornisca un contributo maggiore all'ensemble. Questo metodo offre un vantaggio computazionale significativo rispetto a un'analisi diretta dell'errore.

7.4.3 Procedura di selezione

La procedura di selezione implementa una strategia greedy che ordina gli alberi per importanza decrescente e seleziona i primi k alberi, dove k è determinato dal pruning ratio desiderato. Questa scelta greedy è giustificata dall'assunzione che l'utilità marginale degli alberi decresce monotonicamente con il loro ranking. La validazione empirica di questa assunzione rappresenta un aspetto critico della metodologia, poiché la submodularità della funzione di utilità non è sempre garantita negli ensemble reali. L'implementazione, infatti, include procedure di validazione che verificano che la selezione greedy produca risultati coerenti.

7.5 Pruning per Ensemble: Cumulative pruning per XGBoost

7.5.1 Criterio di pruning cumulativo

Il pruning cumulativo implementato si basa su un principio di selezione basato sull'ordine: vengono mantenuti solo i primi n round di boosting, scartando i successivi. Il presupposto è che le prime iterazioni, che hanno l'obiettivo di ridurre al massimo l'errore iniziale, contribuiscono in modo più significativo e non siano ridondanti come gli alberi meno performanti che si trovano alla fine del processo di training. La percentuale di iterazioni da mantenere è controllata direttamente dal pruning ratio, che determina la frazione di modello da eliminare. In pratica, l'algoritmo mantiene le iterazioni da 1 a n , dove n è calcolato come $(1 - \text{pruning ratio})$ moltiplicato per il numero totale di round di boosting.

7.5.2 Procedura di selezione

A differenza di Random forest, che può semplicemente rimuovere alberi dalla lista degli "estimators", XGBoost richiede un'operazione più delicata a causa della natura additiva delle sue predizioni.

Il processo tecnico consiste in:

1. **Calcolo del numero di iterazioni da mantenere:** si determina il numero di alberi da utilizzare per la predizione. Questo valore viene calcolato in base ad un pruning ratio definito. Nei problemi di classificazione multiclass, ogni round di boosting aggiunge un albero per ogni classe.
2. **Predizione con iterazioni limitate:** invece di modificare la struttura del modello, si sfrutta una funzionalità di XGBoost che permette di specificare il numero di alberi da usare per la predizione. Il modello addestrato mantiene al suo interno tutti gli alberi, ma per calcolare il risultato finale si usa solo l'insieme limitato di alberi specificato.

Se l'obiettivo è creare un modello più leggero da salvare o esportare, è possibile potare il modello in modo permanente. Questo si ottiene limitando l'ensemble agli alberi selezionati e salvando il nuovo modello ridotto. Questo processo è utile per ridurre l'occupazione di memoria e rendere il modello più efficiente per la distribuzione in produzione, una volta che la migliore configurazione è stata identificata tramite lo studio di ablazione.

Capitolo 8

Primo Caso Studio: Regressione su emissioni di automobili

8.1 Progettazione del caso di studio

8.1.1 Tecnologie e librerie

Il workflow sperimentale e di analisi è stato realizzato in Python. Per le reti neurali (MLP e KAN) si è fatto uso di **PyTorch** (moduli `torch.nn`, `torch.optim`, `torch.nn.utils.prune` e utility custom). La famiglia degli ensemble è stata gestita con **scikit-learn** (Random Forest, metriche, pipeline e validazione) e **XGBoost** (`xgboost.XGBRegressor`) per il boosting e la sperimentazione del pruning cumulativo. Per l'esplorazione e la manipolazione dei dati si è utilizzato **pandas** e **numpy**, mentre `scipy.stats` è stata usata per l'analisi statistica e la stima di intervalli di confidenza. La visualizzazione è stata prodotta con **matplotlib** e **seaborn**. Infine, sono stati impiegati tool ausiliari quali `tqdm` per la progress bar, `logging` per il tracciamento degli esperimenti e utility custom per salvataggio/serializzazione dei risultati e calcolo bootstrap al 95%.

8.1.2 Ambienti di sviluppo e infrastruttura

Lo sviluppo è avvenuto in due contesti principali: prototipazione e debug interattivo su **Google Colab**, training e sperimentazione su GPU sul **Cluster HPC** dell’Università di Bologna. Per la riproducibilità delle dipendenze, si sono utilizzati ambienti virtuali (venv) e un file requirements.txt; i risultati sono stati salvati in formati standard (JSON) con metadati (seed, configurazione iperparametri, timestamp).

8.1.3 Linguaggi, automazione e pianificazione del workflow

Il linguaggio principale è Python. L’attività esplorativa e il reporting sono stati condotti tramite Jupyter notebooks, mentre la pianificazione sperimentale è stata automatizzata con piccoli script bash per la sottomissione batch al cluster (SLURM), conversione dei notebook tramite nbconvert e serializzazione dei risultati. Ogni esperimento salva la propria configurazione in JSON (parametri, seed, metadati) ed i run sono riproducibili grazie a seed esplicativi e logging strutturato. Per la sottomissione su cluster si è impiegato uno script SLURM (riportato di seguito) che esegue il notebook e salva il suo output.

8.1.4 Script di sottomissione (Cluster GPU)

Lo script utilizzato per eseguire un notebook Jupyter su SLURM è il seguente:

```
#!/bin/bash
#SBATCH --job-name=Car_Emissions
#SBATCH --mail-type=ALL
#SBATCH --mail-user=martin.tomassi@studio.unibo.it
#SBATCH --time=120:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=8
#SBATCH --mem=60G
```

```

#SBATCH --partition=l40
#SBATCH --output=output_jupyter_exec_job_%j.txt
#SBATCH --chdir=/scratch.hpc/martin.tomassi
#SBATCH --gres=gpu:1

export JUPYTER_CONFIG_DIR="/scratch.hpc/martin.tomassi/jupyter_config_for_job"
export MPLCONFIGDIR="/scratch.hpc/martin.tomassi/matplotlib_cache_for_job_$SLURM_JOB_ID"
mkdir -p "$JUPYTER_CONFIG_DIR"
mkdir -p "$MPLCONFIGDIR"

source venv_car/bin/activate
jupyter nbconvert --to notebook --execute car_emissions.ipynb --output car_emissions_nb.ipynb
jupyter nbconvert --to html car_emissions_trained.ipynb
deactivate

rm -rf "$JUPYTER_CONFIG_DIR"
rm -rf "$MPLCONFIGDIR"

```

8.1.5 Scelte architetturali ed iperparametri

Nella Tabella 8.1 vengono mostrate le scelte finali, dopo l'ottimizzazione degli iperparametri, utilizzate per training, valutazioni comparative e studio di ablazione.

Tabella 8.1: Configurazioni dei modelli.

MLP	<code>input_dim = 1048, hidden_sizes = (128,), dropout = 0.1; ottimizzatore: Adam; lr = 1e-3; l2_lambda = 1e-5; batch_size = 32. Early stopping applicato.</code>
KAN	<code>input_dim = 1048, width = (16,8), grid = 10, k = 4, seed = 0; ottimizzatore: Adam; lr = 1e-3; l2_lambda = 1e-4; batch_size = 32. Early stopping applicato.</code>
Random Forest	<code>n_estimators = 100, max_depth = 30, min_samples_split = 20, min_samples_leaf = 5, max_features = 'sqrt', random_state = 42.</code>
XGBoost	<code>n_estimators = 300, max_depth = 6, learning_rate = 0.1, subsample = 0.8, colsample_bytree = 0.7, reg_alpha = 2.0, reg_lambda = 2.0, random_state = 42.</code>

8.2 Data preparation

Questo caso studio si concentra principalmente sulle fasi di allenamento, valutazione e studio di ablazione; pertanto il dataset utilizzato per l’addestramento è stato fornito già pre-elaborato da terze parti. Di seguito, viene mostrata una sintesi chiara e sintetica delle principali trasformazioni già applicate ai dati, senza entrare nel dettaglio operativo della preparazione dei dati.

8.2.1 Nota sull’origine e stato del dataset

Il dataset finale è il risultato della fusione di tre sorgenti ufficiali su emissioni automobilistiche (Regno unito, Spagna, Canada). La preparazione è stata svolta prima della presente analisi ed ha incluso operazioni di armonizzazione dei nomi di colonna, conversione di unità, pulizia testuale di campi,

rimozione di osservazioni non pertinenti e rimozioni conservative per valori mancanti ritenuti non critici.

8.2.2 Riassunto sintetico delle principali trasformazioni

- **Unione** delle tre sorgenti in un unico dataframe coerente;
- **Omogeneizzazione degli schemi:** mappatura e rinomina delle colonne per ottenere uno schema comune;
- **Pulizia testuale:** standardizzazione e capitalizzazione dei valori testuali e rimozione di suffissi/annotazioni spurie;
- **Imputazione conservativa:** viene usata la mediana (globale o per gruppo/categoria) per riempire i valori mancanti numerici significativi; eliminazioni conservative per categoria quando necessario;
- **Filtri:** eliminazione di righe prive della variabile target (CO2_Emissions) o prive di attributi ritenuti fondamentali per l'allenamento dei modelli.

8.2.3 Ruolo nella tesi

Poiché il preprocessing non rientra nelle attività svolte dall'autore, nelle sezioni successive si assume che il dataset finale sia corretto e pronto per la modellazione; la trattazione si concentra pertanto su:

- la scelta delle architetture e degli algoritmi di regressione;
- la strategia di validazione e splitting utilizzata per evitare data leakage;
- le metriche di valutazione, il confronto tra modelli e lo studio di ablazione.

8.3 Addestramento modelli

8.3.1 Features e variabile target

- Year: anno di produzione del veicolo (numerica);

- **Manufacturer**: casa automobilistica (categorica);
- **Model**: nome del modello (categorica);
- **Engine_cm3**: cilindrata in cm³ (numerica);
- **Transmission_type**: Automatic / Manual (categorica);
- **Fuel_type**: Petrol / Diesel / LPG / ... (categorica);
- **Fuel_consumption**: L/100km (numerica).

La variabile target è **CO2_Emissions** (g/km).

8.3.2 Pipeline di preprocessing

1. **Selezione delle features**: dal dataset finale, sono state mantenute le colonne importanti per la previsione della CO2.
2. **One-hot encoding**: le variabili categoriche sono state codificate con OneHotEncoder.
3. **Standardizzazione**: le variabili numeriche vengono scalate con StandardScaler quando il modello lo richiede.

8.3.3 Split dei dati

- **Train/Test split iniziale**: si ottiene un test set indipendente con il 20% del dataset.
- **Train pool**: dal training residuo, è stato creato un "train pool" di 25000 osservazioni usato per la ricerca di iperparametri. Questo pool è poi suddiviso in:
 - set di training interno: 22500 osservazioni;
 - set di validazione interno: 2500 osservazioni.

- **Validazione incrociata annidata (nested CV) con Random search:** per la selezione degli iperparametri è stato usato il Random search tramite un approccio nested :
 - **Outer CV:** KFold con 5 fold (outer_folds = 5) — per stimare la performance generalizzata;
 - **Inner CV:** KFold con 3 fold (inner_folds = 3) — per comparare le configurazioni campionate dalla Random Search.

8.3.4 Metriche e intervalli di confidenza

- **MSE** (Mean Squared Error): $MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$.
- **MAE** (Mean Absolute Error): $MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$.
- **MAPE** (Mean Absolute Percentage Error): $MAPE = \frac{100}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right|$, calcolata escludendo i casi con $y_i = 0$ o non finiti.
- **R² e R²-adjusted:**

$$R^2 = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2}, \quad R^2_{adj} = 1 - (1 - R^2) \frac{n - 1}{n - k - 1},$$

dove n è il numero di osservazioni e k il numero di predittori.

- **Max Error:** $\max_i |y_i - \hat{y}_i|$.
- **Complessità del modello:** conteggio dei parametri addestrabili per reti neurali; stima del numero di nodi (sommatoria dei nodi degli alberi) per Random Forest / XGBoost.

Sono stati calcolati per MSE, MAE e MAPE gli intervalli di confidenza.

8.3.5 Strategia di training comune e griglie di iperparametri

La strategia di training adottata è sostanzialmente omogenea per tutti i modelli considerati. Qua sotto vengono riportate prima le procedure ed i criteri comuni, poi le griglie di iperparametri specifiche per ogni modello.

Procedure comuni

- **Preprocessing e batching:** i dati categorici sono codificati con `OneHotEncoder` (fit sul training) e le feature numeriche vengono scalate con `StandardScaler`.
- **Protocollo di validazione:** viene impiegata una validazione incrociata annidata (nested CV) con **outer KFold = 5** e **inner KFold = 3**. L'inner loop esegue una Random Search sulle griglie di iperparametri; la migliore combinazione trovata viene riaddestrata sul sottoinsieme interno e valutata sul corrispondente outer test fold.
- **Random Search:** le iterazioni n per ciascun modello sono state scelte con la logica probabilistica descritta in precedenza (obiettivo $P = 0.90$ di includere almeno una delle $k = 10$ migliori configurazioni). I valori adottati sono riportati nella sezione relativa a ciascun modello.
- **Ottimizzazione (reti neurali):** Adam come ottimizzatore, `MSELoss` come funzione di perdita, regolarizzazione L2 opzionale (weight decay o termine esplicito). Epoche massime impostate a 1000 ma con early stopping attivo per interrompere prima il training, se non si osserva un miglioramento significativo.
- **Modelli tree-based:** Random Forest e XGBoost sono addestrati in modo simile alle reti neurali, ma mediante la loro API nativa;
- **Valutazione finale:** per ogni outer fold si raccolgono le metriche (MSE, MAE, MAPE, R^2 , R^2 -adjusted, Max Error) e si calcolano gli intervalli di confidenza al 95% per MSE/MAE/MAPE come descritto nella sezione metriche. È inoltre stimata la complessità del modello.

Griglie di iperparametri per modello

MLP (Random Search: $n = 15$)

- `input_dim`: dimensionalità input
- `hidden_sizes`: [(32,32), (64,64), (128,)]

- `dropout`: [0.1, 0.2, 0.5]
- `lr`: [10^{-3} , 10^{-4}]
- `batch_size`: 32
- `l2_lambda`: [0, 10^{-5} , 10^{-4} , 10^{-3}]

KAN (Random Search: $n = 6$)

- `input_dim`: dimensionalità input
- `width`: [(8,4), (16,8)]
- `grid`: [5, 10]
- `k`: [2, 4]
- `seed`: 0
- `lr`: 10^{-3}
- `batch_size`: 32
- `l2_lambda`: [0, 10^{-5} , 10^{-4} , 10^{-3}]

Random Forest (Random Search: $n = 32$)

- `n_estimators`: [100, 200, 300, 500]
- `max_depth`: [10, 20, 30]
- `min_samples_split`: [10, 20, 30]
- `min_samples_leaf`: [5, 10]
- `max_features`: ['sqrt', 'log2']
- `random_state`: 42

XGBoost (Random Search: $n = 677$)

- `n_estimators`: [100, 200, 300]
- `max_depth`: [3, 4, 5, 6]
- `learning_rate`: [0.01, 0.05, 0.1]
- `subsample`: [0.7, 0.8, 0.9]
- `colsample_bytree`: [0.7, 0.8, 0.9]
- `reg_alpha`: [0.5, 1.0, 2.0]
- `reg_lambda`: [2.0, 5.0, 10.0]
- `random_state`: 42

8.4 Valutazione dei modelli

La figura 8.1 sintetizza le principali metriche di performance con intervalli di confidenza al 95% e la complessità dei modelli.

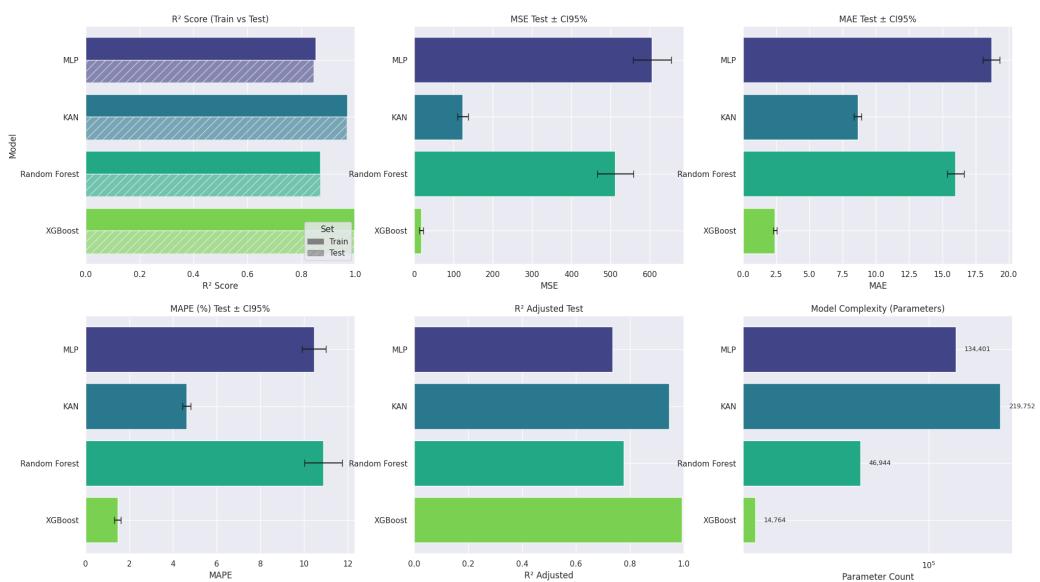


Figura 8.1: Confronto visivo delle prestazioni dei modelli.

8.4.1 Selezione del miglior modello

Per identificare il modello più adatto al problema di regressione delle emissioni di CO₂ è stata adottata una procedura che combina prestazione e complessità dei modelli:

1. Si definiscono le metriche di interesse e la direzione di ottimizzazione:

```
metrics = {
    'MSE_Test'          : 'min',
    'MAE_Test'          : 'min',
    'MAPE_Test'         : 'min',
    'R2_Test'           : 'max',
    'R2_Adjusted_Test' : 'max'
}
```

2. Per ciascuna metrica si calcola il rank dei modelli (rank 1 = migliore) considerando la direzione appropriata (ascendente o discendente).
3. Si calcola un `performance_score` come media dei rank delle metriche di prestazione.
4. Si calcola un `Complexity_rank` in base al numero di parametri/nodi (rank 1 = meno complesso).
5. Si combinano performance e complessità secondo varie strategie:
 - **Equal Weight (1:1)**: `performance_score + Complexity_rank`
 - **Complexity Weighted (1:2)**: `performance_score+2×Complexity_rank`
 - **Extreme Complexity (1:3)**: `performance_score+3×Complexity_rank`
 - **Pareto Approach (40:60)**: normalizzazione di performance e complexity e combinazione con peso 0.4 sulla performance e 0.6 sulla complessità.
6. Per ogni metodo di aggregazione si seleziona il modello con punteggio minimo (migliore).

Risultati

- **Equal Weight (1:1) → XGBoost**
- **Complexity Weighted (1:2) → XGBoost**
- **Extreme Complexity (1:3) → XGBoost**
- **Pareto Approach (40:60) → XGBoost**

Di seguito la tabella riassuntiva con i valori principali usati per il ranking (valori ricavati dall'analisi finale):

Model	Param_Count	Avg_Perf	Compl_Rank	Equal_Rank	Ext_Rank	Pareto_Rank
MLP	134,401	3.83	3	4	3	4
KAN	219,752	2.00	4	3	4	3
XGB	14,764	1.00	1	1	1	1
RF	46,944	3.17	2	2	2	2

Tabella 8.2: Riepilogo ranking: conteggio parametri, performance media (rank-based) e ranks per metodo di aggregazione.

Conclusioni

- **XGBoost** ottiene la performance assoluta migliore sulle metriche di regressione e ha la complessità stimata più bassa: per questi motivi è il modello più adatto per il deployment operativo.
- **KAN** raggiunge un buon compromesso in termini di performance, ma la sua complessità stimata è elevata rispetto al beneficio assoluto di predizione.
- **MLP** mostra performance inferiori in termini di MSE/MAE rispetto a XGBoost pur avendo un elevato numero di parametri.
- **Random Forest** fornisce risultati stabili e prestazioni intermedie, ma non raggiunge XGBoost nel compromesso performance/complessità.

La classifica dei tre migliori modelli, secondo il criterio *complexity-weighted* (dal migliore al peggiore), è la seguente:

1. XGBoost (Params: 14,764 | MSE Test: 17.52)
2. Random Forest (Params: 46,944 | MSE Test: 511.72)
3. MLP (Params: 134,401 | MSE Test: 605.28)

8.5 Studio di ablazione

8.5.1 Ablation study: L1 pruning su MLP e KAN

L'obiettivo è misurare il compromesso tra compressione del modello (numero di parametri attivi / rapporto di compressione) ed il mantenimento delle prestazioni (MSE, MAE, R^2 , MAPE).

Metodologia

- **Tecnica di pruning:** pruning globale L1 applicato ai pesi lineari della MLP e ai coefficienti spline della KAN.
- **Pruning ratios testati:** {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95}.
- **Valutazione:** ogni versione pruned è stata valutata su test set e train set con le metriche MSE, MAE, MAPE, R^2 e max error. Per ciascuna configurazione sono stati calcolati il numero di parametri totali, i parametri attivi dopo pruning e il rapporto di compressione (original params / active params).
- **Definizioni operative:**
 - **baseline:** modello non pruned (pruning ratio = 0.0).
 - **Punto di degrado significativo:** primo pruning ratio che comporta oltre il 5% di perdita relativa in R^2 rispetto alla baseline.
 - **Best trade-off:** punto con massima compressione che causa $\leq 2\%$ di perdita relativa in R^2 .

Figura riassuntiva

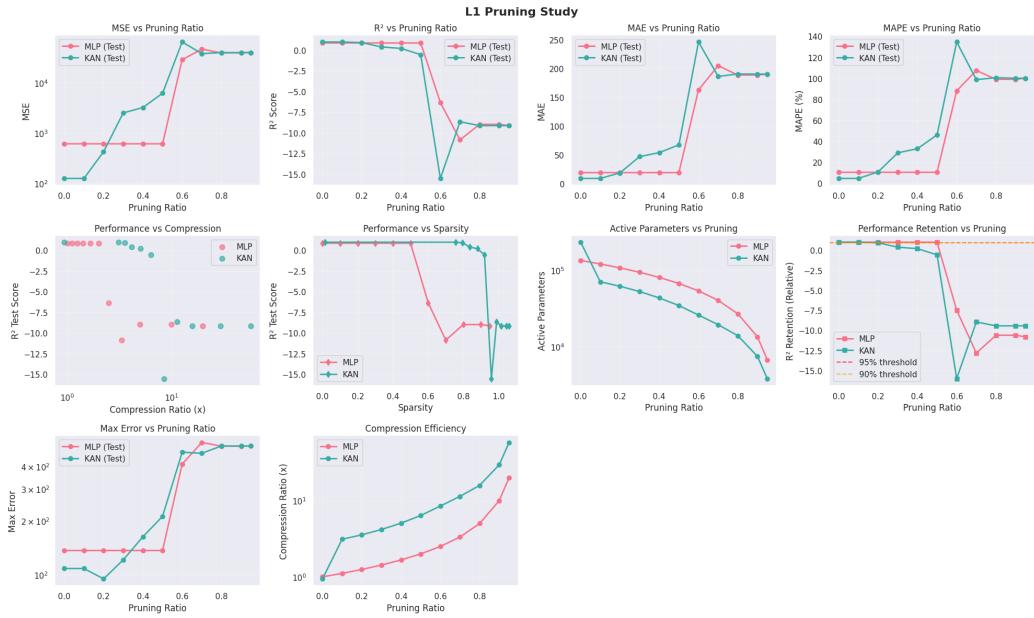


Figura 8.2: Risultati dello studio L1 pruning per MLP e KAN.

Risultati

Model	Total params	Baseline R^2	Best trade-off	Sign. degr.
MLP	134 401	0.8467	50% pruning (compression $\approx 2.0\times$)	60% pruning
KAN	219 752	0.9688	10% pruning (compression $\approx 3.1\times$)	20% pruning

Tabella 8.3: Riepilogo dei punti di trade-off e dei punti di degrado osservati nello studio L1 pruning.

- **MLP:** baseline $R^2 = 0.8467$. Fino a 50% di pruning non si osserva perdita significativa in R^2 (best trade-off: $\approx 2\times$ compression con $\leq 2\%$ perdita relativa). La degradazione significativa del modello si manifesta intorno al 60% di pruning (MSE e MAE aumentano drasticamente, R^2 diventa negativo per pruning più aggressivi). Compressione massima sperimentata: $\approx 20\times$ (pruning 95%), ma a questo livello la performance è inutilizzabile.

- **KAN:** baseline $R^2 = 0.9688$. Il miglior trade-off empirico è stato osservato a soli **10%** di pruning (compression $\approx 3.1\times$) con performance praticamente invariata. La degradazione significativa compare già a circa il **20%** di pruning; oltre tale soglia MSE e MAE aumentano velocemente. Compressione massima sperimentata: $\approx 57.8\times$ (pruning 95%), ma con perdita di performance molto elevata.
- **Osservazione generale:** KAN fornisce una baseline molto migliore (minore MSE, R^2 vicino a 1) ma è più sensibile al pruning rispetto alla MLP; la MLP è più tollerante a pruning moderati fino al 50% e può essere usata per compressioni moderate quando si richiede maggiore robustezza post-pruning.

8.5.2 Ablation study: ensemble pruning su Random Forest e XGBoost

Lo scopo è valutare il compromesso tra riduzione del modello (numero di alberi / rounds) e mantenimento delle prestazioni (qui riportate con metriche di regressione — MSE, R^2 , MAE).

Metodologia

- **Rank-Based Pruning per Random Forest:** per ciascun albero si calcola una metrica di importanza. Gli alberi vengono ordinati per importanza e si rimuovono quelli con importanza minore fino alla frazione indicata dal `pruning_ratio`.
- **Cumulative Pruning per XGBoost:** si mantengono solo le prime iterazioni di boosting (rounds) fino al numero corrispondente a $(1 - \text{pruning_ratio})$ della lunghezza originaria; in pratica si tronca la sequenza di boosting post-training limitando le predizioni ai primi n alberi.
- **Pruning ratios testati:** $\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95\}$.

- **Valutazione:** per ogni versione pruned si calcolano il numero di alberi totali e rimanenti, il rapporto di compressione (total / remaining), e le metriche di performance su train/test (MSE, MAE, MAPE, R^2 , max error).
- **Definizioni operative:**
 - **baseline:** pruning ratio = 0.0 (modello non modificato);
 - **Punto di degrado significativo:** primo pruning_ratio che causa perdita relativa in misura di interesse (qui usiamo R^2) $> 5\%$;
 - **Best trade-off:** massima compressione che causa perdita relativa $\leq 2\%$.

Figura riassuntiva

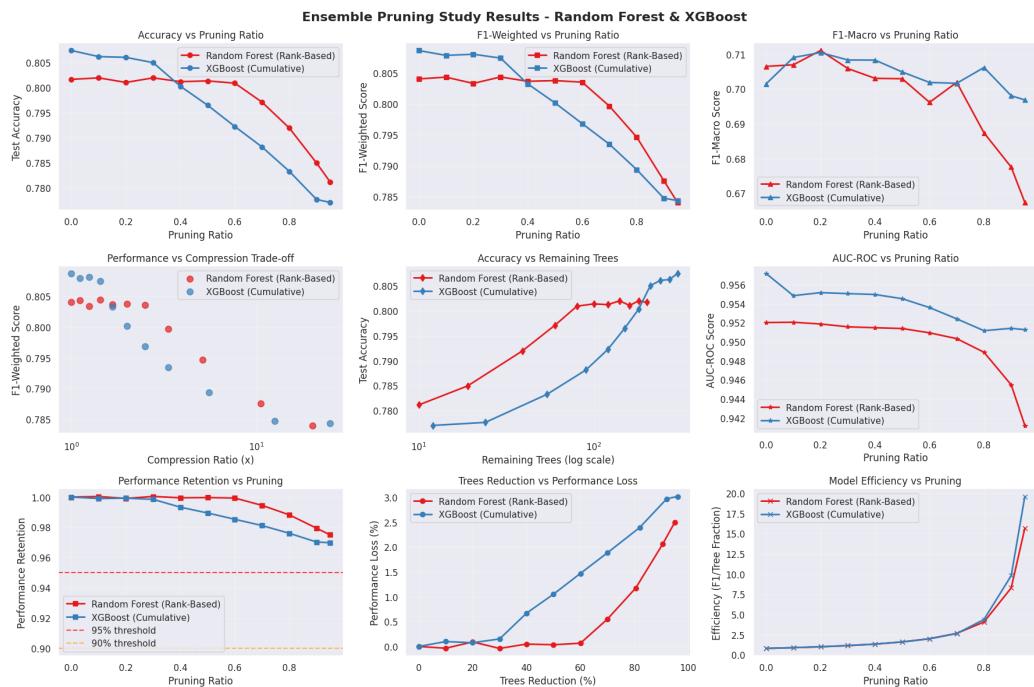


Figura 8.3: Risultati dello studio di pruning per Random Forest (rank-based) e XGBoost (cumulative).

Risultati

Model	Total components	Baseline (R^2)	Best trade-off	Sign. degr.
RF	100 trees	0.8704	95% (compression 20.0x)	no degr. rilevata
XGB	300 trees	0.9956	90% (compression 10.3x)	95% pruning

Tabella 8.4: Riepilogo sintetico dei punti di trade-off osservati per i due ensemble.

- **XGBoost (cumulative):** il modello è molto robusto a pruning moderati: fino al 70–80% di riduzione degli estimators la perdita in R^2 è trascurabile; il best trade-off empirico trovato nello studio è al 90% di pruning (mantenendo 29/300 trees, compression $\approx 10.3\times$) con una perdita relativa di R^2 dell’ordine dell’~ 1.3%. Oltre il 90% (es. 95%) la performance cala rapidamente.
- **Random Forest (rank-based):** la rimozione degli alberi meno performanti tende a mantenere o addirittura migliorare leggermente la generalizzazione (riduzione dell’overfitting). Nel nostro esperimento RF rimane stabile anche con pruning aggressivi: mantenendo solo 5 alberi su 100 (95% pruning, compression 20x) la R^2 test non peggiora e, in alcuni punti, migliora rispetto alla baseline.

8.5.3 Ablation study — Confronto complessivo (Neural Networks vs Ensemble)

Per il confronto, sono stati considerati pruning ratios tipici: 30%, 50%, 70%, 90%.

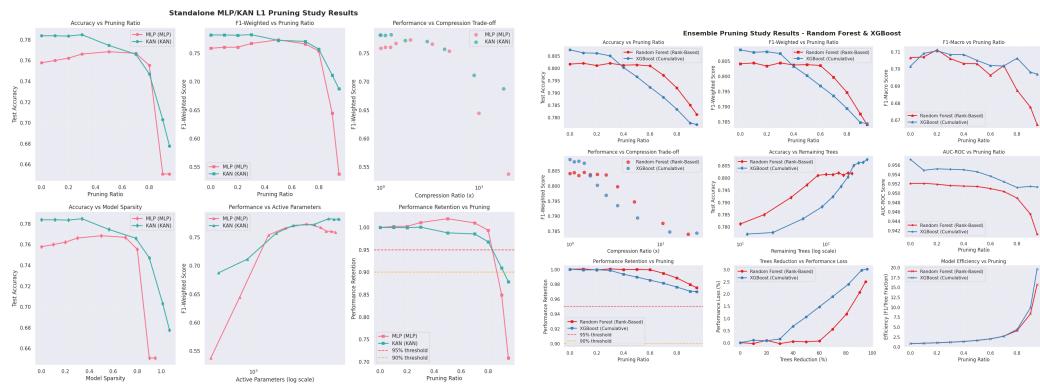
Metriche e output considerati

Per ogni modello sono state raccolte le seguenti misure:

- R^2 (test);
- MSE (test);

- **MAE** (test);
- **MAPE** (test);
- **Compression ratio:** rapporto tra dimensione modello baseline e modello pruned;
- statistiche di performance retention (R^2 pruned / baseline).

Figure riassuntive



A: Ablation MLP vs KAN (L1 pruning). B: Ablation Random Forest (Rank-based) vs XGBoost (cumulative).

Figura 8.4: Risultati sintetici degli studi di pruning.

Riepilogo

Model	Baseline R^2	Best trade-off	Compression
XGB	0.9956	0.9823 @ 90% pr	$\approx 10.3\times$
RF	0.8704	0.9219 @ 95% pr	$\approx 20.0\times$
MLP	0.8467	0.8467 @ 50% pr	$\approx 2.0\times$
KAN	0.9688	0.9688 @ 10% pr	$\approx 3.1\times$

Tabella 8.5: Riepilogo sintetico dei principali punti di trade-off.

Confronto su soglie tipiche (30%, 50%, 70%, 90%)

- **30% pruning:** RF e XGBoost mantengono le prestazioni quasi invariate ($R^2 \approx 0.894$ e 0.995); MLP resta stabile ($R^2 = 0.8467$); KAN degrada molto rapidamente ($R^2 \approx 0.364$).
- **50% pruning:** XGBoost conserva $R^2 \approx 0.994$; RF migliora ulteriormente la generalizzazione ($R^2 \approx 0.904$); MLP resta stabile; KAN crolla sotto lo 0.
- **70% pruning:** XGBoost resta robusto ($R^2 \approx 0.993$, compressione $3.3\times$); RF migliora ancora ($R^2 \approx 0.917$); MLP degrada fortemente ($R^2 < 0$); KAN in pieno collasso prestazionale.
- **90% pruning:** RF mostra la miglior performance in termini di retention ($R^2 \approx 0.9296$, compressione $11\times$); XGBoost mantiene buone prestazioni ($R^2 \approx 0.9823$, compressione $10.3\times$); MLP e KAN risultano compromessi (R^2 negativo).

Conclusioni

1. **XGBoost:** risulta il modello più stabile sotto pruning aggressivo: fino al 90% di riduzione mantiene $R^2 > 0.98$ e compressione $\approx 10.3\times$.
2. **Random Forest:** il pruning rank-based porta a un miglioramento netto della generalizzazione: con 90% di pruning, R^2 sale a 0.9296 (compressione $\approx 11.1\times$).
3. **MLP:** tollera pruning fino al 50% (compressione $2\times$) senza perdita di R^2 , ma degrada rapidamente oltre questa soglia.
4. **KAN:** estremamente sensibile al pruning: già al 20% scende a $R^2 \approx 0.893$, e crolla sotto lo 0 oltre il 40-50%.

Capitolo 9

Secondo Caso Studio: Classificazione di PM2.5

9.1 Data preparation

Questa sezione descrive in dettaglio le operazioni svolte per il caricamento, la normalizzazione, la pulizia e l'arricchimento del dataset utilizzato nel caso studio. Lo scopo principale della fase di Data preparation è trasformare i dati grezzi in una rappresentazione coerente, completa e utilizzabile per la successiva fase di allenamento.

9.1.1 Fonti e descrizione generale del dataset

Il dataset principale utilizzato nello studio raccoglie misurazioni relative alla qualità dell'aria in numerose stazioni di monitoraggio presenti in 453 città indiane per il periodo temporale 2010-2023. Le osservazioni includono sia concentrazioni di inquinanti sia variabili meteorologiche ed ambientali. Un file ausiliario, denominato stations_info.csv, contiene la mappatura tra i file contenenti le misure e informazioni di contesto (stato, città, agenzia responsabile, data di inizio rilevamento), permettendo in tal modo una gestione centralizzata dei metadati associati alle stazioni di monitoraggio.

9.1.2 Caricamento dati e organizzazione iniziale

Il caricamento del dataset è stato effettuato tramite la lettura dei file compressi estratti in una directory dedicata in locale o su Google Colab. Per ogni file contenente le misure di una singola stazione vengono eseguite le seguenti operazioni:

1. estrazione del contenuto dell'archivio ZIP in una directory strutturata per stato;
2. lettura dei metadati (file stations_info.csv) e rimozione di colonne non necessarie per l'analisi, al fine di semplificare la tabella dei metadati;
3. costruzione di un insieme aggregato: per ogni stato, vengono individuati automaticamente tutti i file CSV che condividono il prefisso identificativo dello stato; ogni file viene letto, alla tabella vengono aggiunte le colonne "city" e "state" ed infine tutte le tabelle sono concatenate in un unico DataFrame nazionale.

Il risultato di questa fase è un DataFrame unico contenente le misurazioni orarie con colonne per le concentrazioni degli inquinanti, le variabili meteorologiche e gli identificatori geografici.

9.1.3 Indicizzazione temporale

Nel dataset originale sono presenti due colonne che definiscono la finestra temporale di misurazione: "From Date" e "To Date". Per semplificare la gestione delle serie storiche si è adottata la seguente procedura:

- conversione della colonna "From Date" in tipo datetime;
- rinomina di "From Date" in "datetime" ed impostazione di tale colonna come indice temporale del DataFrame;
- rimozione della colonna "To Date" (ridondante per l'analisi);

Questa trasformazione consente di sfruttare le funzionalità native di raggruppamento e ricampionamento temporale offerte dalle librerie di manipolazione delle serie temporali.

9.1.4 Riduzione e unificazione di feature ridondanti

Dall'esplorazione iniziale, é emersa la presenza di colonne duplicate o varianti dello stesso nome (per esempio, "Ozone (ug/m³)" e "Ozone (ppb)"). Per evitare rappresentazioni inconsistenti della stessa grandezza sono state seguite le operazioni:

1. identificazione di gruppi di colonne equivalenti tramite l'analisi grafica dei trend (andamento delle medie annue) e confronto delle statistiche di base (media, deviazione standard, minimo, massimo);
2. definizione di un mapping di riduzione: ad esempio, tutte le varianti di Xylene sono state aggregate in una colonna comune;
3. per ciascun gruppo di colonne, i valori non nulli provenienti dalle colonne secondarie sono stati trasferiti nella colonna principale; successivamente le colonne ridondanti sono state eliminate.

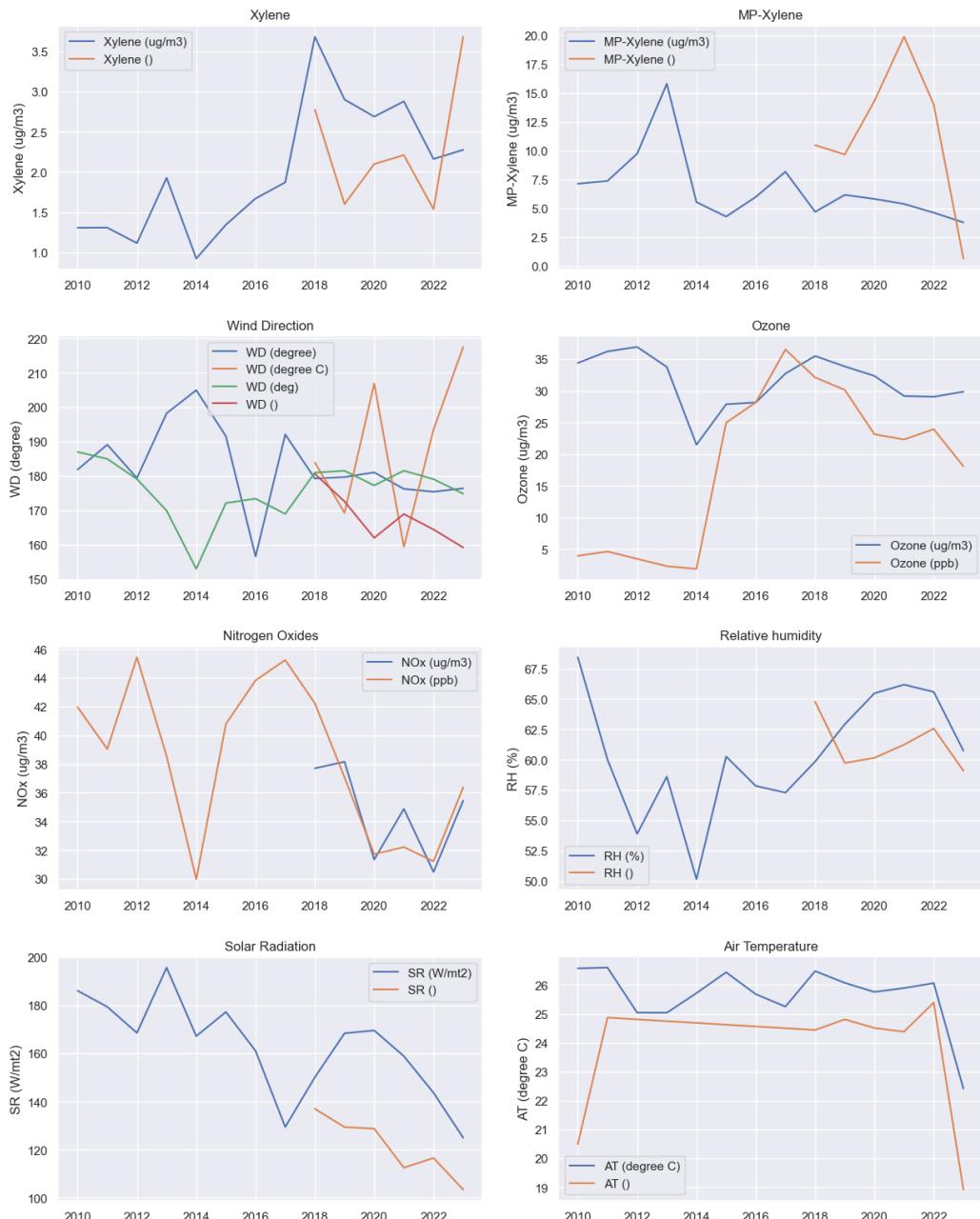


Figura 9.1: Similarità delle features - Analisi medie annue

9.1.5 Verifica e gestione dei valori mancanti

La quantificazione dei valori mancanti è stata effettuata calcolando il numero assoluto e la percentuale di missing per ogni variabile.

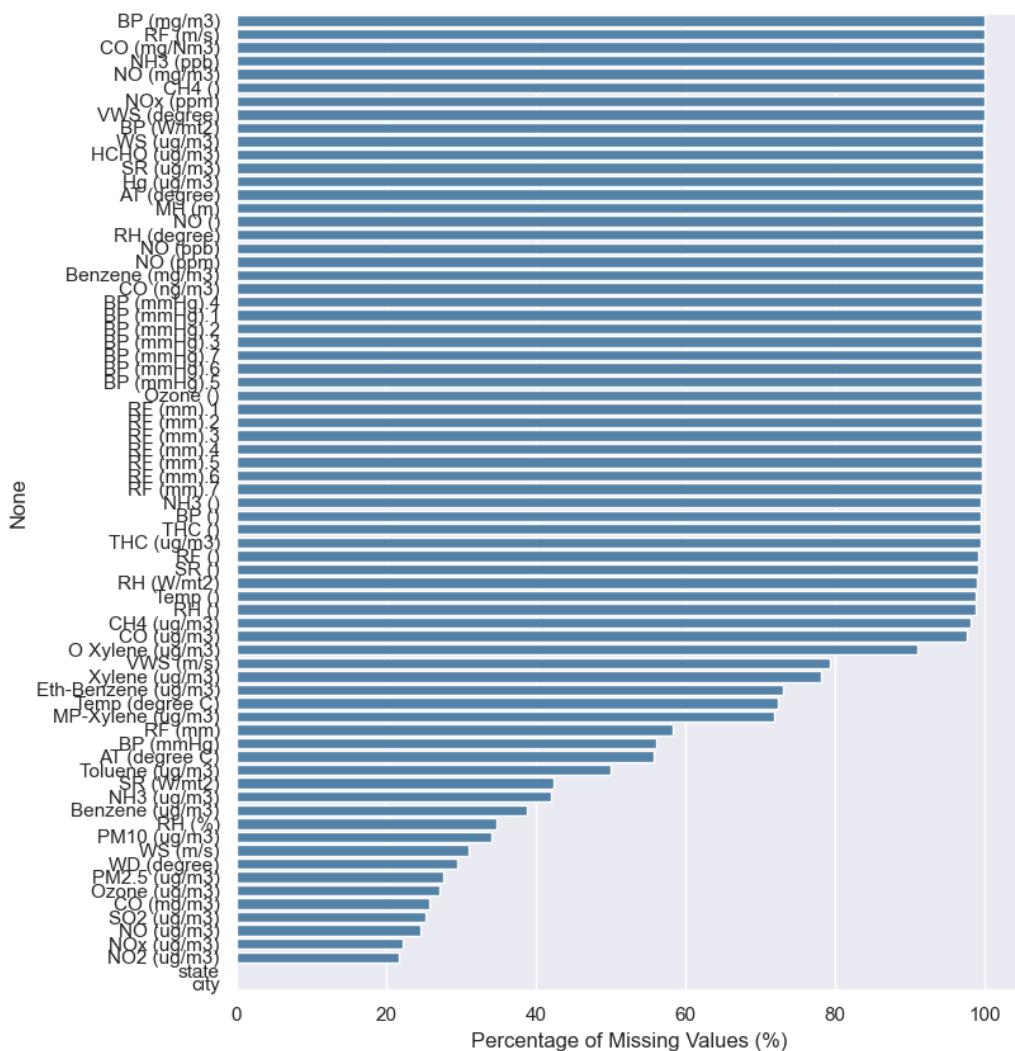


Figura 9.2: Percentuali dei Valori Mancanti.

Sono stati applicati i seguenti criteri:

- rimozione delle osservazioni completamente vuote e delle colonne completamente vuote;

- eliminazione delle colonne con una proporzione di valori mancanti superiore alla soglia del 40% (soglia scelta per bilanciare trade-off tra perdita informativa e robustezza statistica);
- per le restanti colonne numeriche, i valori NaN vengono sostituiti tramite il metodo di interpolazione forward-fill (propagazione dell'ultimo valore valido, non NaN, di quella feature), seguita da una sostituzione tramite media per eventuali rimanenze;

Questa strategia è comunemente adottata quando si lavora con serie temporali, poiché preserva la dinamica locale dei segnali evitando di introdurre forti discontinuità.

9.1.6 Analisi esplorativa (EDA) e selezione delle feature rilevanti

Per esplorare relazioni e correlazioni tra le variabili sono state eseguite le seguenti analisi:

1. calcolo delle medie raggruppate su più frequenze temporali (giorno, mese, anno) e visualizzazione dei trend con grafici a linee;
2. utilizzo di pairplot per ispezionare le relazioni bivariate e le distribuzioni univariate;
3. costruzione della matrice di correlazione e visualizzazione tramite heatmap per quantificare le correlazioni lineari. Sono state identificate, come features potenzialmente rilevanti, quelle con correlazione assoluta maggiore di 0.4 con "PM2.5".

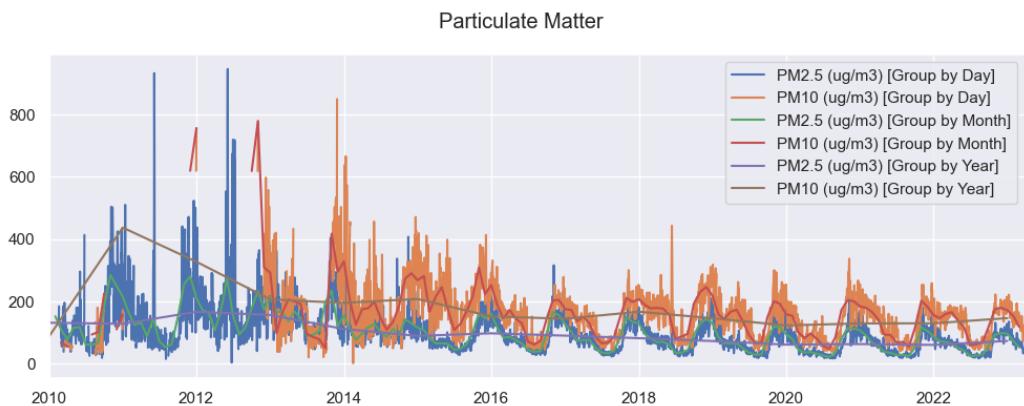


Figura 9.3: Analisi dei trend giornalieri, mensili e annuali per il Particolato.

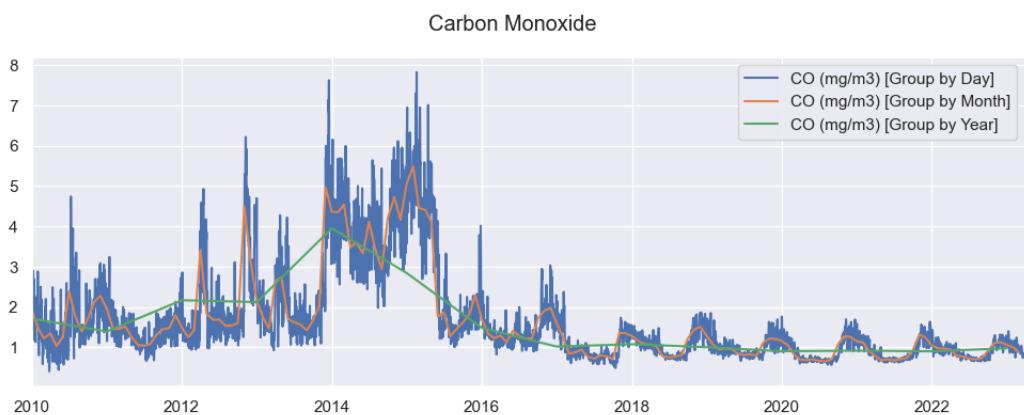


Figura 9.4: Analisi dei trend giornalieri, mensili e annuali per il Monossido di carbonio.

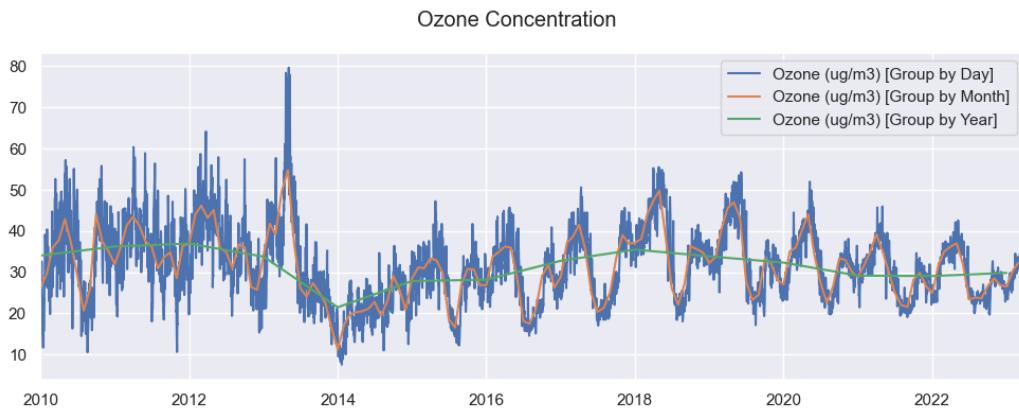


Figura 9.5: Analisi dei trend giornalieri, mensili e annuali per l'Ozono.

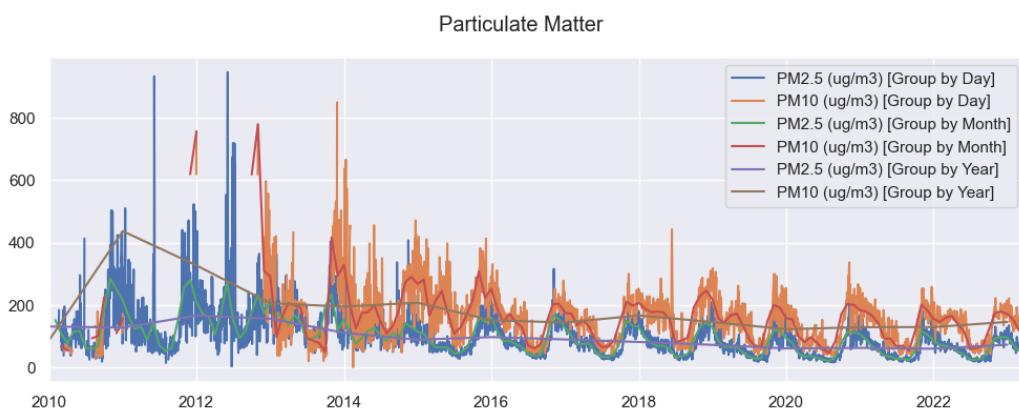


Figura 9.6: Analisi dei trend giornalieri, mensili e annuali per i Composti dell'azoto.

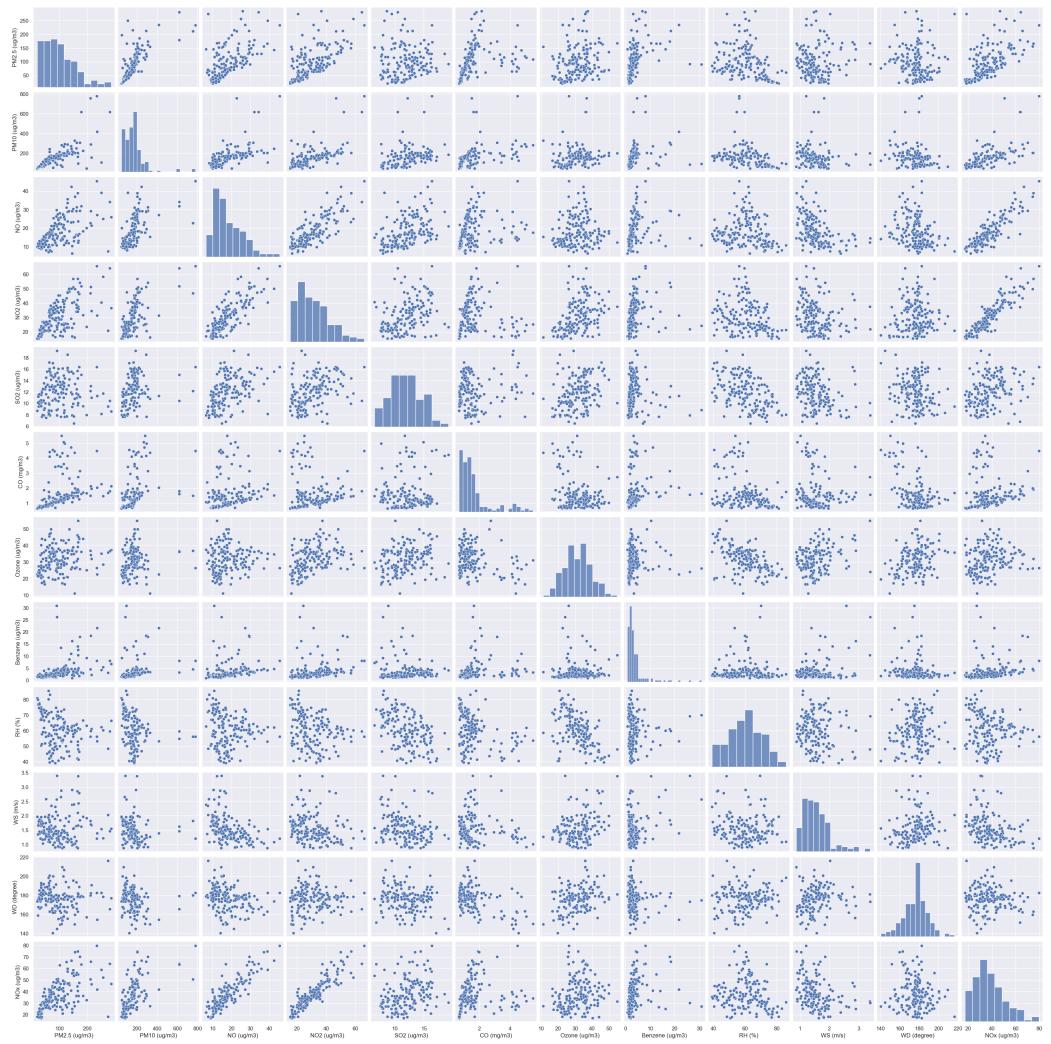


Figura 9.7: Analisi della relazione tra features e le loro distribuzioni univariate.

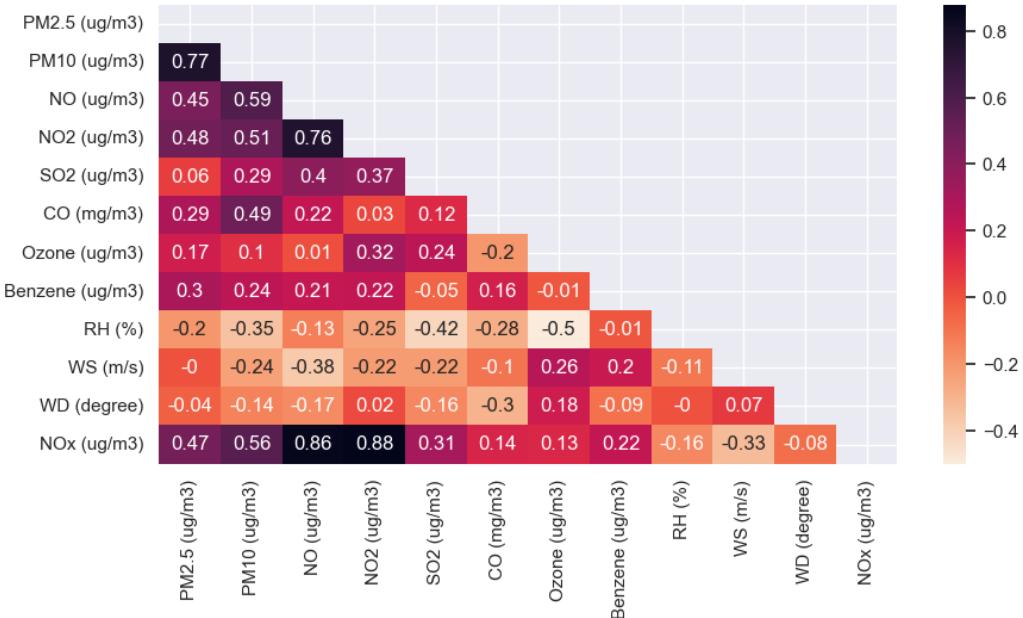


Figura 9.8: Analisi della correlazione tra variabili.

Dall'analisi, si é deciso di mantenere la variabile aggregata "NOx" e rimuovere le componenti ridondanti ("NO", "NO2") per evitare multicollinearitá e semplificare i modelli successivi.

9.1.7 Ricampionamento ed aggregazione a livello statale

Poiché i dati aggregati includevano misurazioni provenienti da più stazioni all'interno dello stesso stato con lo stesso timestamp, si é deciso di ricampionare temporaneamente i dati a frequenza oraria e di aggregare le osservazioni a livello di stato tramite media aritmetica. La procedura implementata é la seguente:

```
df_resampled = df_india
    .groupby('state')
    .resample('60min')
    .mean(numeric_only=True)
    .reset_index()
```

9.1.8 Rilevamento e rimozione degli outlier

Per l'identificazione di outlier nei valori delle concentrazioni degli inquinanti, si è utilizzato l'algoritmo Isolation Forest, un metodo ensemble noto per la sua efficacia nell'identificare valori anomali indipendentemente dalla distribuzione a priori dei dati. I punti principali dell'approccio sono:

- scelta di un insieme limitato di variabili su cui applicare il rilevamento (in questo caso, "PM2.5", "CO", "Ozone", "NOx");
- inizializzazione del modello con un valore di contamination pari a 0.01 (assumendo quindi che circa l'1% delle istanze sia anomala) ed un random_state fissato per la riproducibilità;
- addestramento del modello ed uso del metodo predict per etichettare le istanze anomale con -1 e le normali con 1;
- rimozione delle osservazioni etichettate come outlier e verifica della distribuzione pre/post tramite istogrammi.

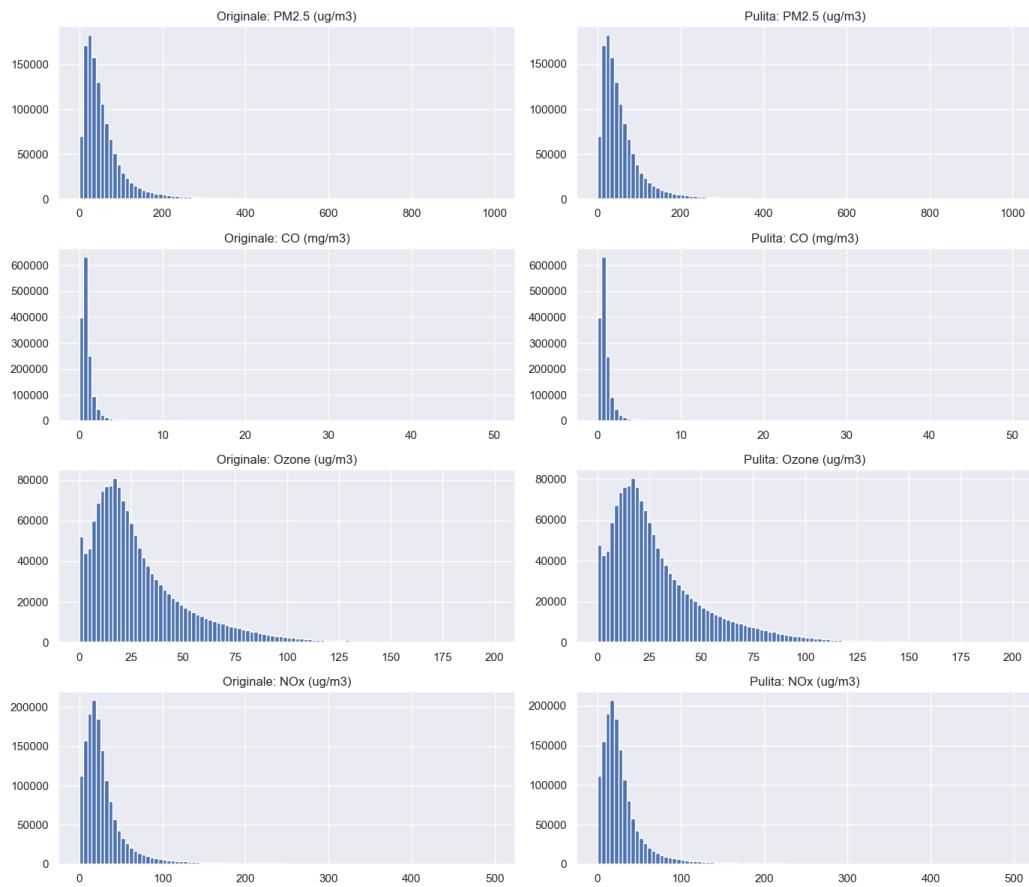


Figura 9.9: Visualizzazione della distribuzione, di ogni variabile esaminata, prima e dopo.

9.1.9 Feature engineering ed arricchimento temporale

Per catturare informazioni temporali rilevanti per aiutare i modelli a predire in modo più accurato, sono state create variabili derivate dall'indice temporale quali: "hour", "dayofmonth", "dayofweek", "dayofyear", "weekofyear", "month", "quarter" e "year". Queste variabili servono a modellare stagionalità, ciclicità giornaliera e pattern settimanali/annuali tipici dei fenomeni atmosferici e delle attività umane legate all'inquinamento atmosferico.

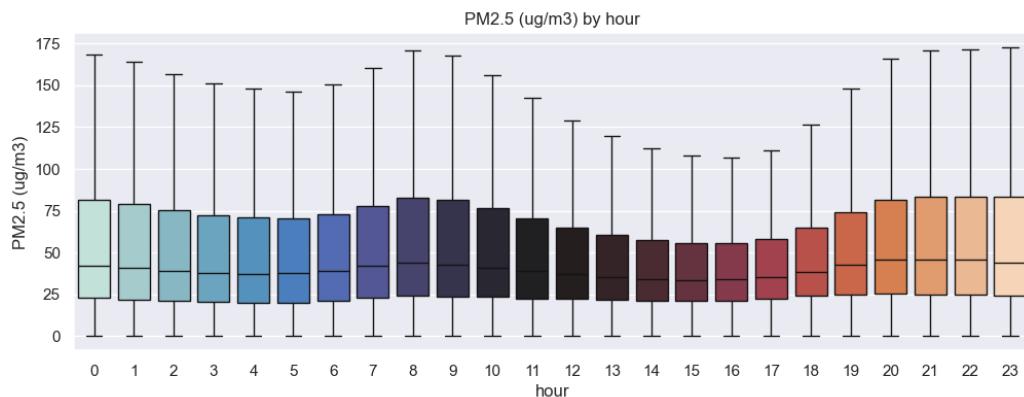


Figura 9.10: Visualizzazione della distribuzione di PM2.5 per ora.

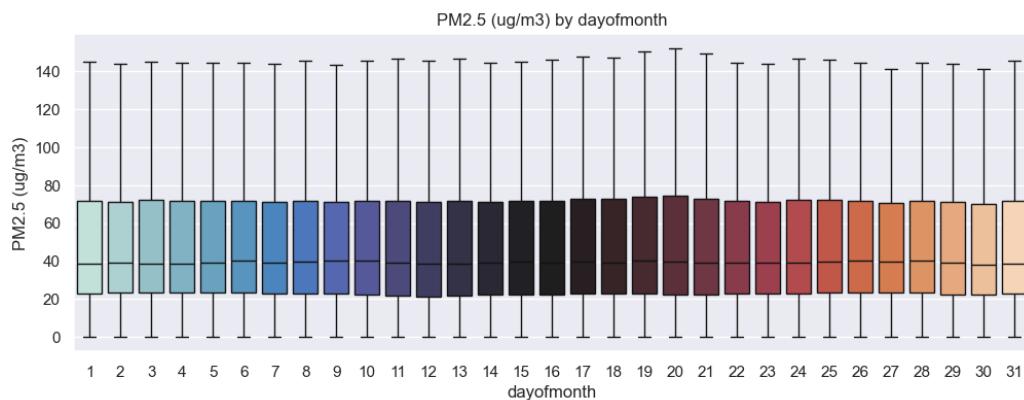


Figura 9.11: Visualizzazione della distribuzione di PM2.5 per giorno del mese.

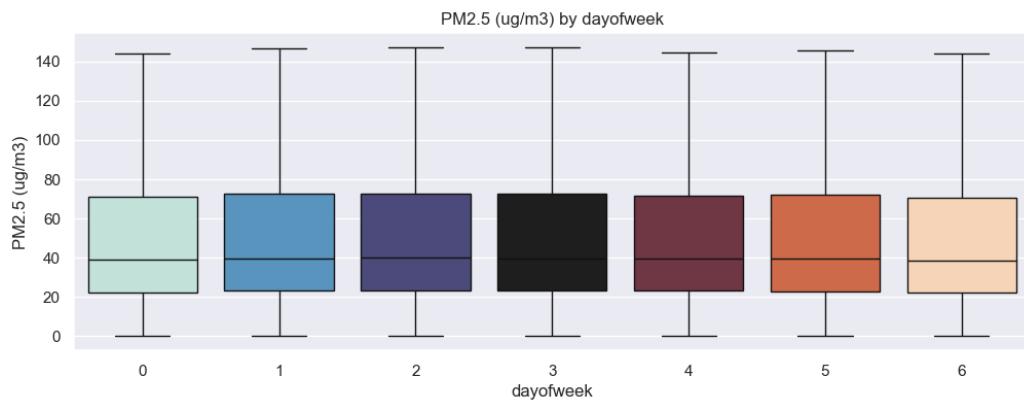


Figura 9.12: Visualizzazione della distribuzione di PM2.5 per giorno della settimana.

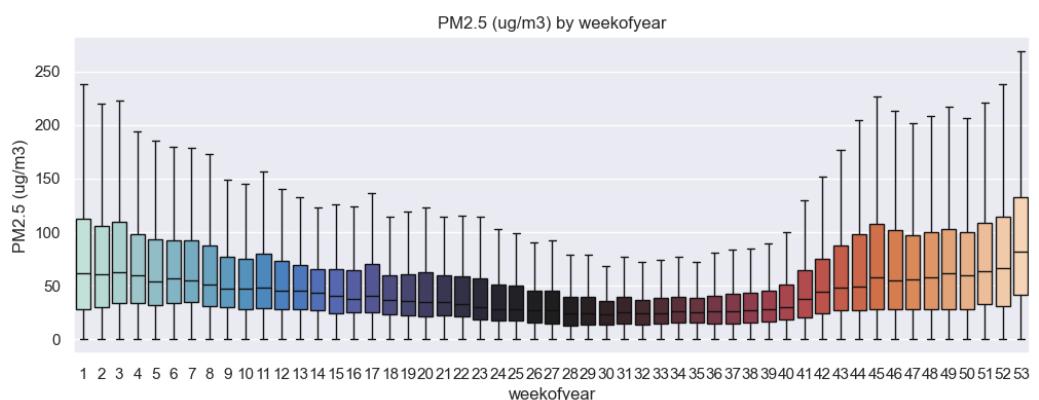


Figura 9.13: Visualizzazione della distribuzione di PM2.5 per settimana dell'anno.

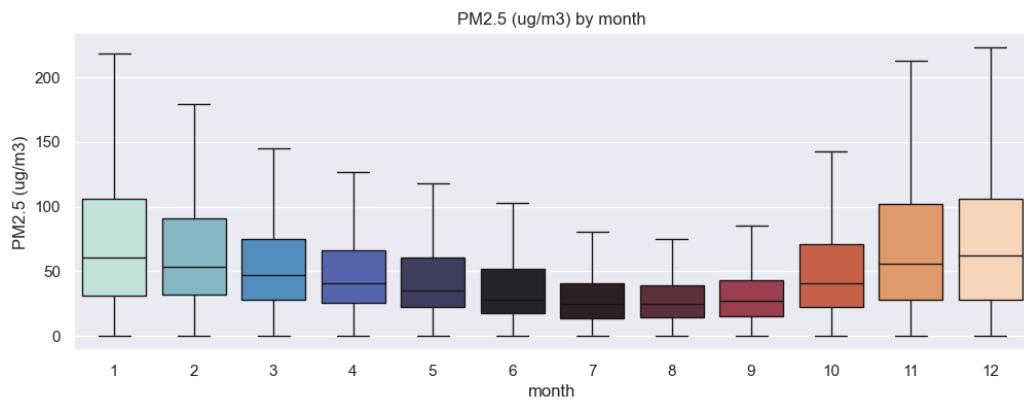


Figura 9.14: Visualizzazione della distribuzione di PM2.5 per mese dell'anno.

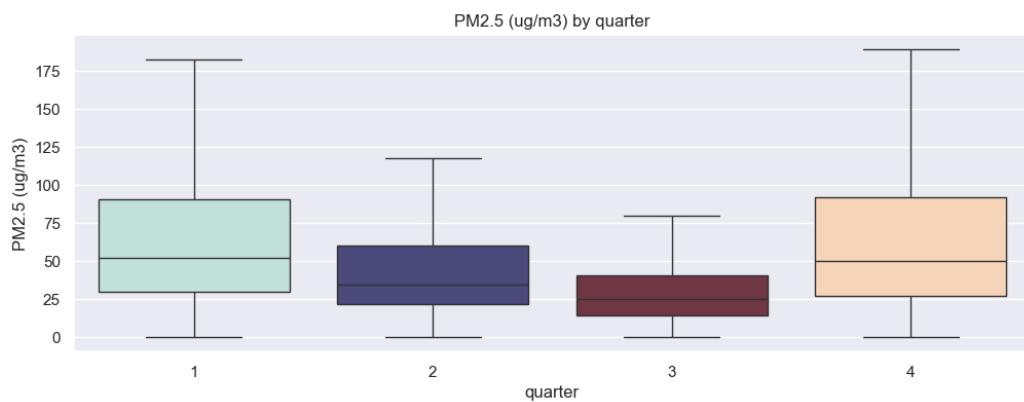


Figura 9.15: Visualizzazione della distribuzione di PM2.5 per trimestre dell'anno.

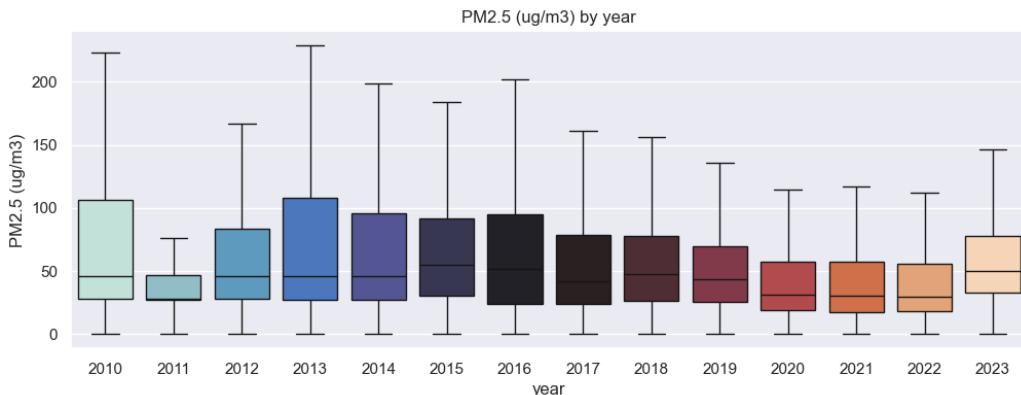


Figura 9.16: Visualizzazione della distribuzione di PM2.5 per anno.

9.1.10 Creazione di lag-features e categorizzazione di PM2.5

Per catturare l'informazione storica intrinseca nelle serie temporali, sono state create delle lag-features per alcune delle variabili più rilevanti, che consentono ai modelli di sfruttare dipendenze temporali (ad esempio stagionalità annua, effetto di breve periodo (giorni/settimane) e persistenti condizioni mensili) e rappresentano una tecnica consolidata nella preparazione dei dati per serie temporali.

In questo caso, sono state generate le seguenti tipologie di lag:

- a lungo termine: 1 e 2 anni;
- a medio termine: 1 mese;
- a breve termine: 1 settimana ed ultimi 3 giorni dall'osservazione selezionata.

9.1.11 Categorizzazione PM2.5

Per trasformare la variabile target "PM2.5" da continua a discreta, con l'obiettivo di passare da un problema di regressione ad uno di classificazione, si è deciso di mappare i valori di PM2.5 in sei categorie dell'AQI (Good, Moderate, Unhealthy for Sensitive, Unhealthy, Very Unhealthy, Hazardous) usando le soglie standard corrispondenti ai breakpoints AQI per "PM2.5"

(National Ambient Air Quality Standards for PM).

Le soglie utilizzate, misurate in $\mu g/m^3$, sono le seguenti:

- **GOOD:** 0-9.0
- **MODERATE:** 9.1-35.4
- **UNHEALTHY FOR SENSITIVE:** 35.5-55.4
- **UNHEALTHY:** 55.5-125.4
- **VERY UNHEALTHY:** 125.5-225.4
- **HAZARDOUS:** 225.5+

Per facilitare l'analisi e l'addestramento dei modelli, le categorie sono state convertite in etichette intere da 1 (GOOD) a 6 (HAZARDOUS). Questo approccio non solo semplifica la gestione dei dati, ma mantiene anche la relazione ordinale tra le classi.

9.2 Addestramento dei modelli

9.2.1 Pipeline di preprocessing

1. **Selezione di variabili:** le feature indipendenti includono variabili temporali (year, month, dayofmonth, dayofweek, dayofyear, weekofyear, quarter), l'identificativo geografico (state) e le lag-features (p.es. pm_lag_1D, pm_lag_1W, pm_lag_1M, pm_lag_1Y, ecc.).
2. **Rimozione NaN derivanti dalle lag:** per confrontare i modelli in condizioni omogenee si è applicato un "drop" delle righe contenenti valori mancanti generati dalle lag-features.
3. **Trasformazioni:** le colonne numeriche sono standardizzate tramite StandardScaler; la variabile categorica "state" è codificata con OneHotEncoder. Le trasformazioni sono incapsulate in un ColumnTransformer e usate come primo step nelle pipeline di training.

4. **Bilanciamento delle classi in fase di training:** per i modelli che beneficiano di un dataset bilanciato, si applica SMOTE all'interno della pipeline. Lo SMOTE è eseguito esclusivamente sul training fold in ogni ripetizione di cross-validation per evitare data leakage.

Nota su XGBoost XGBoost è in grado di gestire internamente i valori mancanti; tuttavia, per rendere i confronti equi con modelli che non supportano questa funzionalità internamente, si è deciso di rimuovere direttamente tutte le righe con NaN nelle lag-features prima di tutti gli esperimenti.

9.2.2 Split dei dati

Il dataset è stato suddiviso in due partizioni principali nel rispetto dell'ordine cronologico delle osservazioni:

- **Training set (80%):** utilizzato per la fase di ricerca degli iperparametri con Random Search mediante Time Series Cross Validation;
- **Test set (20%):** tenuto separato e mai usato durante la ricerca degli iperparametri; impiegato esclusivamente per la valutazione finale comparativa fra i modelli. Ciò permette di avere una stima realistica delle prestazioni su dati non visti e previene il data leakage.

9.2.3 Metriche e intervalli di confidenza

Vengono utilizzati le seguenti metriche per valutare i diversi modelli sperimentali:

- **Accuracy, F1-score weighted e F1-score macro;**
- **Confusion matrix e Classification report** per analisi di classe;
- **AUC-ROC weighted e AUC-PR weighted** quando disponibili (modelli con predict_proba);

- **Bootstrap di intervalli di confidenza (CI95%)** per F1-weighted, F1-macro e AUC.

9.2.4 Random Forest

Spazio degli iperparametri

- `n_estimators`: [100, 150, 200]
- `max_samples`: [0.5, 0.7, 0.9]
- `max_depth`: [5, 10, 15]
- `min_samples_split`: [2, 5]
- `min_samples_leaf`: [2, 5]
- `max_features`: ['sqrt', 'log2']

9.2.5 XGBoost

XGBoost richiede etichette 0-indexed per la funzione obiettivo "multi:softprob". Le etichette numeriche sono pertanto convertite sottraendo 1 alle classi originali.

Spazio degli iperparametri

- `max_depth`: [3, 5]
- `learning_rate`: [0.01, 0.05, 0.1]
- `n_estimators`: [100, 200, 300]
- `subsample`: [0.7, 0.9]
- `colsample_bytree`: [0.7, 0.9]
- `gamma`: [0, 0.2, 0.4]
- `min_child_weight`: [1, 5]

9.2.6 MLP

Strategia di training

- **Ottimizzatore:** Adam;
- **Funzione di attivazione:** ReLU per gli strati nascosti;
- **Loss:** CrossEntropyLoss (richiede etichette 0-indexed);
- **Regolarizzazione:** L2 opzionale (l2_lambda);
- **Batch size:** 32 (configurabile);
- **Device:** GPU se disponibile (altrimenti CPU);
- **Early stopping:** EarlyStopper con parametri patience e min_delta configurabili per interrompere l'allenamento in caso di mancato miglioramento sulla validation loss;

Spazio degli iperparametri

- **input_dim:** [input_dim] (numero di features dopo il preprocessing);
- **hidden_sizes:** [(64,64), (128,), (128,64), (256,128), (512,256)];
- **dropout:** [0.0, 0.2, 0.5];
- **lr** (learning rate): [1e-3, 1e-4];
- **num_classes:** [num_classes] (numero di classi);
- **l2_lambda** (coefficiente L2): [0.0, 1e-5, 1e-4, 1e-3].

9.2.7 KAN

Strategia di training

L'addestramento segue una strategia simile a quella adottata per la rete MLP, con alcune specificità legate all'architettura:

- **Ottimizzatore:** Adam;
- **Loss:** CrossEntropyLoss (etichette 0-indexed);
- **Regularizzazione:** L2 opzionale (l2_lambda);
- **Batch size:** 32 (configurabile);
- **Device:** GPU se disponibile;
- **Early stopping:** EarlyStopper con patience e min_delta configurabili;

Spazio degli iperparametri

- **input_dim:** [input_dim] (numero di features dopo preprocessing);
- **width:** [(8,4), (16,8), (32,16), (64,32)] (struttura a livelli della rete);
- **grid:** [5, 10, 20] (dimensione della griglia interna);
- **k:** [2, 4] (ordine/complessità della combinazione);
- **seed:** [0] (per riproducibilità);
- **lr:** [1e-3, 1e-4];
- **num_classes:** [num_classes];
- **l2_lambda:** [0.0, 1e-5, 1e-4, 1e-3].

9.3 Valutazione dei modelli

La figura 9.17 sintetizza le principali metriche di performance (Accuracy, F1-weighted, F1-macro, AUC-ROC e AUC-PR) con intervalli di confidenza al 95% (bootstrap) e la complessità dei modelli.

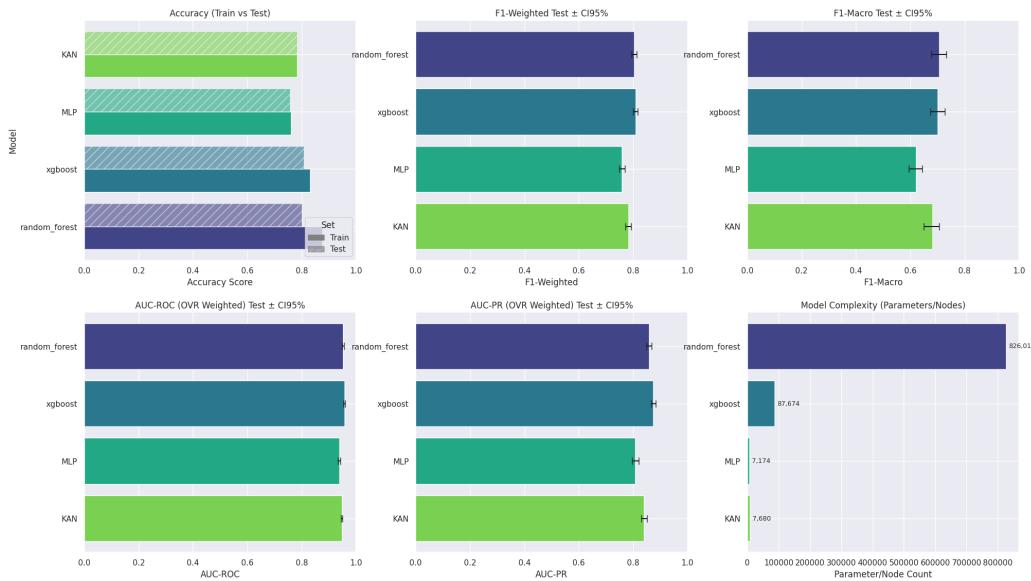


Figura 9.17: Confronto visivo delle prestazioni dei modelli.

9.3.1 Selezione del miglior modello

Per identificare il modello più adatto alla risoluzione del problema, si è deciso di costruire un sistema di classificazione che combina prestazione e complessità di ogni modello. La procedura implementata è la seguente:

1. Si definiscono le metriche di interesse e la direzione di ottimizzazione:

```
metrics = {
    'Accuracy_Test': 'max',
    'F1_Weighted_Test': 'max',
    'F1_Macro_Test': 'max',
    'AUC_ROC_OVR_Weighted': 'max',
    'AUC_PR_OVR_Weighted': 'max'
}
```

2. Per ciascuna metrica si calcola il "rank" dei modelli (rank 1 = migliore) considerando la direzione appropriata (ascendente o discendente).

3. Si calcola un "performance_score" come media dei rank delle metriche di prestazione.
4. Si calcola un "Complexity_rank" in base al numero di parametri/nodi (rank 1 = meno complesso).
5. Si combinano performance e complessità secondo varie strategie:
 - **Equal Weight (1:1)**: $\text{performance_score} + \text{Complexity_rank}$
 - **Complexity Weighted (1:2)**: $\text{performance_score} + 2 \times \text{Complexity_rank}$
 - **Extreme Complexity (1:3)**: $\text{performance_score} + 3 \times \text{Complexity_rank}$
 - **Pareto Approach (40:60)**: normalizzazione di performance e complexity e combinazione con peso 0.4 sulla performance e 0.6 sulla complessità.
6. Per ogni metodo di aggregazione si sceglie il modello con punteggio minimo (migliore).

Risultati

Di seguito sono riassunti i risultati della procedura di ranking: i punteggi minimi identificano il migliore modello per ciascun metodo di aggregazione.

- **Equal Weight (1:1) → XGBoost**
- **Complexity Weighted (1:2) → MLP**
- **Extreme Complexity (1:3) → MLP**
- **Pareto Approach (40:60) → XGBoost**

Di seguito la tabella riassuntiva con i valori principali usati per il ranking (valori ricavati dall'analisi finale):

Model	Param_Count	Avg_Perf	Compl_Rank	Equal_Rank	Ext_Rank	Pareto_Rank
MLP	7,174	4.0	1	2	1	2
KAN	7,680	3.0	2	2	2	3
XGB	87,674	1.2	3	1	3	1
RF	826,018	1.8	4	4	4	4

Tabella 9.1: Riepilogo ranking: conteggio parametri, performance media (rank-based) e ranks per metodo di aggregazione.

Conclusioni

Dalla procedura di ranking multi-criterio:

- **XGBoost** presenta la performance assoluta più alta sulle metriche di classificazione (F1-weighted migliore), ma ha una complessità significativamente maggiore rispetto alle reti neurali;
- **KAN** mostra ottime performance ed un numero di parametri relativamente contenuto, risultando un'opzione solida quando si ricerca maggiore accuratezza senza far esplodere la complessità;
- **MLP** raggiunge un buon compromesso: performance competitive con bassa complessità, risultando il modello raccomandato dal criterio di "complexity-weighted ranking";
- **Random Forest** ottiene metriche competitive ma con una complessità molto elevata, il che ne riduce l'attrattività se si dà peso alla parsimonia del modello.

Classifica dei tre migliori modelli, secondo il criterio "complexity-weighted" (dal migliore al peggiore):

1. MLP (Params: 7,174 | F1-Weighted: 0.7589)
2. KAN (Params: 7,680 | F1-Weighted: 0.7823)
3. XGBoost (Params: 87,674 | F1-Weighted: 0.8087)

Quindi, applicando un bilanciamento tra performance e complessità, MLP risulta la scelta più equilibrata per un deployment efficiente nel contesto considerato. XGBoost è la scelta migliore se il criterio primario è la massima performance, mentre KAN rappresenta una valida alternativa con buon compromesso performance/complessità.

9.4 Studio di ablazione

9.4.1 Ablation study: L1 pruning su MLP e KAN

L'obiettivo è misurare il compromesso tra compressione del modello (numero di parametri attivi / rapporto di compressione) e il mantenimento delle prestazioni (Accuracy, F1-weighted, AUC).

Metodologia

- **Tecnica di pruning:** pruning globale L1 applicato ai pesi lineari della MLP ed ai coefficienti spline della KAN.
- **Pruning ratios testati:** {0.0, 0.1, 0.2, 0.3, 0.5, 0.7, 0.8, 0.9, 0.95}.
- **Valutazione:** ogni versione pruned è stata valutata sul test set e sul train set sulle stesse metriche definite precedentemente. Sono stati calcolati il numero di parametri totali, i parametri attivi dopo pruning ed il rapporto di compressione.
- **Definizioni operative:**
 - **baseline:** modello non pruned (pruning ratio 0.0).
 - **Punto di degrado significativo:** primo pruning ratio che comporta oltre il 5% di perdita relativa in F1-weighted rispetto alla baseline.
 - **Best trade-off:** punto con massima compressione che causa $\leq 2\%$ di perdita relativa in F1-weighted.

Figura riassuntiva

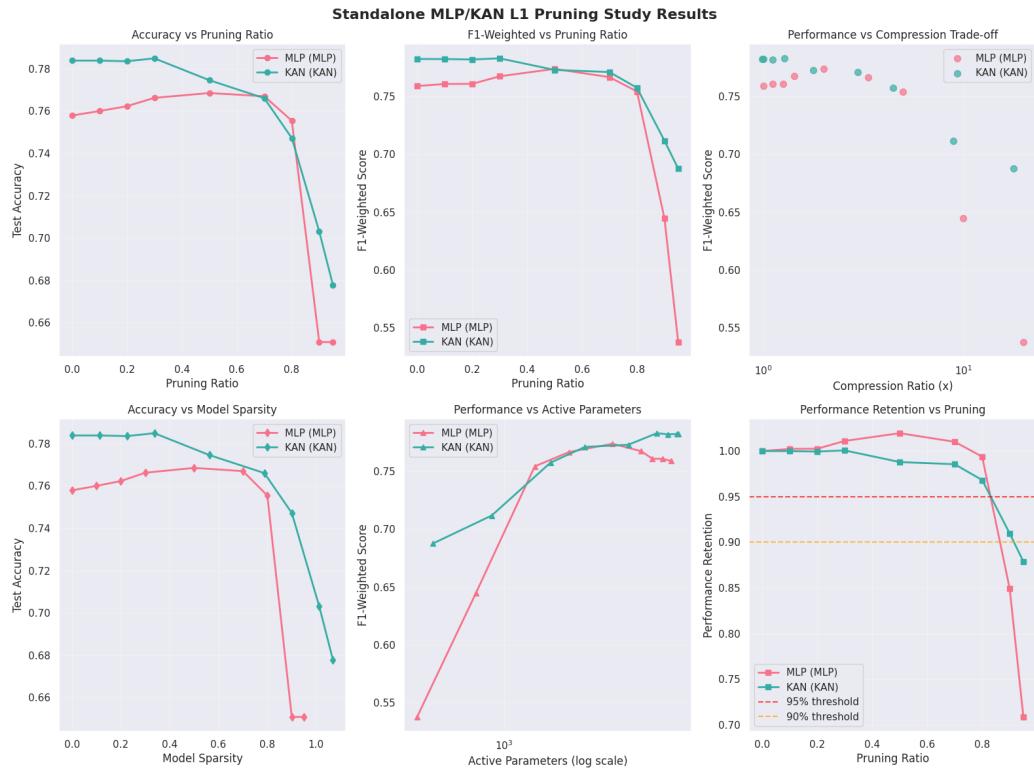


Figura 9.18: Risultati dello studio L1 pruning per MLP e KAN (metriche su test set in funzione del pruning ratio).

Risultati

Model	Total params	Baseline F1	Best trade-off	Sign. degr.
MLP	7,174	0.7589	80% pruning (5.0x compression)	90% pruning
KAN	7,680	0.7823	70% pruning (3.0x compression)	90% pruning

Tabella 9.2: Riepilogo dei punti di trade-off e dei punti di degrado osservati nello studio L1 pruning.

- **MLP:** baseline F1-weighted = **0.7589**. Il miglior punto di trade-off trovato nello studio aggiornato è al **80%** di pruning (compression $\approx 5\times$) con F1-weighted ≈ 0.7542 (perdita relativa $\approx 0.6\%$ rispetto alla

baseline). La degradazione significativa ($> 5\%$ perdita relativa) si osserva a **90%** di pruning; a livelli estremi ($\geq 95\%$) la perdita diventa elevata.

- **KAN:** baseline F1-weighted = **0.7823**. Il modello risulta più robusto per pruning moderati: il miglior trade-off è al **70%** di pruning (compression $\approx 3\times$) con F1-weighted ≈ 0.7710 (perdita relativa $\approx 1.4\%$). La soglia di degrado significativa si posiziona intorno al **90%** di pruning.
- **Compressione massima raggiunta:** rispettivamente $\approx 20\times$ (MLP, pruning 95%) e $\approx 17.8\times$ (KAN, pruning 95%). Tuttavia tali livelli di compressione comportano perdite di performance troppo elevate per essere considerati praticabili in deployment senza ulteriori tecniche.

9.4.2 Ablation study: ensemble pruning su Random Forest e XGBoost

Lo scopo è valutare il compromesso tra riduzione del modello (numero di alberi / rounds) e mantenimento delle prestazioni (F1-weighted, accuracy, AUC).

Metodologia

- **Rank-based pruning per Random Forest:** per ogni albero viene calcolata una metrica di importanza (somma delle feature importances dell'albero). Gli alberi vengono ordinati per importanza e si rimuovono quelli con importanza inferiore fino alla frazione indicata dal pruning ratio.
- **Cumulative pruning per XGBoost:** si mantengono solo le prime iterazioni di boosting (rounds) corrispondenti a $(1 - \text{pruning_ratio})$ della lunghezza originaria della sequenza di rounds; in pratica si tronca la sequenza di alberi di boosting.
- **Pruning ratios testati:** {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95}.

- **Valutazione:** per ogni versione pruned sono state calcolate: numero di alberi totali e rimanenti, rapporto di compressione (total/remaining), Accuracy, F1-weighted, F1-macro, AUC-ROC e AUC-PR (quando calcolabili).
- **Definizioni operative:**
 - **baseline:** pruning ratio 0.0.
 - **Punto di degrado significativo:** primo pruning ratio che causa una perdita relativa in F1-weighted > 5%.
 - **Best trade-off:** massima compressione con perdita relativa in F1-weighted \leq 2%.

Figura riassuntiva

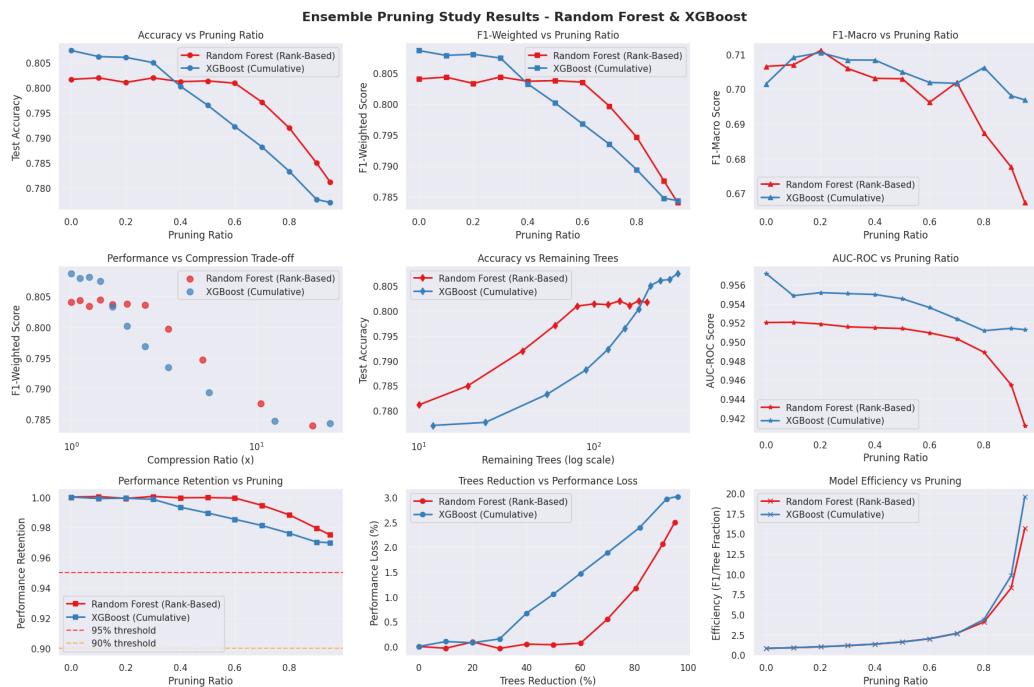


Figura 9.19: Risultati dello studio di pruning per Random Forest (rank-based) e XGBoost (cumulative).

Risultati

Model	Total trees	Baseline F1	Best trade-off (pruning)	Sign. degr.
RF	200	0.8041	80% (5.1× compression)	no degr. rilevata
XGB	300	0.8087	70% (3.3× compression)	no degr. rilevata

Tabella 9.3: Riepilogo sintetico dei punti di trade-off osservati per i due ensemble.

- **XGBoost:** conserva buona parte delle prestazioni anche riducendo il numero di rounds. Il punto di miglior compromesso è al 70% di pruning (mantenendo i primi 90 rounds su 300), con compressione $\approx 3.3\times$ e perdita relativa in F1-weighted $\approx 1.9\%$.
- **Random Forest:** il rank-based pruning riduce significativamente il numero di alberi mantenendo le prestazioni. Nel run corrente il best trade-off risulta al 80% di pruning (39 alberi rimanenti, $\approx 5.1\times$ compression) con F1-weighted ≈ 0.7946 (perdita relativa $\approx 1.2\%$). Non è stata osservata una degradazione significativa ($> 5\%$) nell'intervallo testato.
- **Compressione massima raggiunta:** fino a $\approx 20\times$ per Random Forest (pruning = 0.95) e fino a $\approx 25\times$ per XGBoost (pruning = 0.95), ma questi livelli estremi comportano perdite di performance più marcate e vanno considerati con cautela.

9.4.3 Ablation study — Confronto complessivo (Neural Networks vs Ensemble)

Per il confronto, sono stati estratti i risultati di ogni modello a soglie tipiche: 30%, 50%, 70%, 90%.

Metriche e output considerati

Per ogni modello sono state raccolte le seguenti misure:

- **Accuracy** (test);
- **F1-Weighted** (test);
- **F1-Macro** (test);
- **AUC-ROC e AUC-PR weighted** quando calcolabili;
- **Compression ratio**: rapporto tra dimensione modello baseline e dimensione dopo pruning;
- statistiche di performance retention (pruned / baseline).

Figure riassuntive

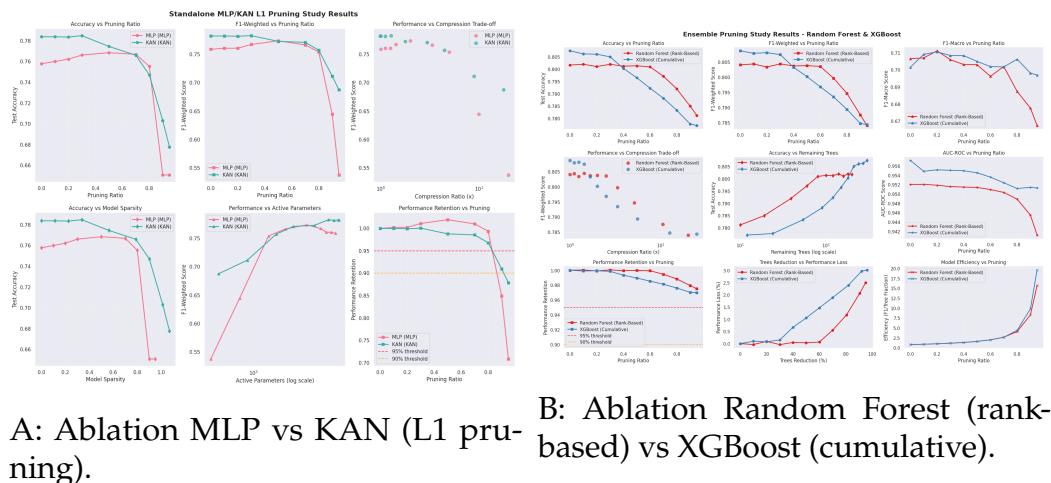


Figura 9.20: Risultati sintetici degli studi di pruning (metriche su test set in funzione del pruning ratio).

Riepilogo

Model	Baseline F1 (weighted)	Best trade-off	Compression
MLP	0.7589	0.7542 @ 80% pr	$\approx 5.0\times$
RF	0.8041	0.7946 @ 80% pr	$\approx 5.1\times$
XGB	0.8087	0.7935 @ 70% pr	$\approx 3.3\times$
KAN	0.7823	0.7710 @ 70% pr	$\approx 3.0\times$

Tabella 9.4: Riepilogo sintetico dei principali punti di trade-off.

Confronto su soglie tipiche (30%, 50%, 70%, 90%)

- **30% pruning:** tutti i metodi mantengono o migliorano lievemente le prestazioni rispetto alla baseline. MLP mostra un piccolo aumento di accuracy/F1 a questa soglia; XGBoost e KAN conservano prestazioni molto alte (F1 praticamente invariata).
- **50% pruning:** MLP mantiene o migliora la F1 (trade-off ottimo a $\sim 2\times$ compression); Random Forest e XGBoost restano stabili con diminuzioni contenute.
- **70% pruning:** XGBoost permette ancora compressioni $\sim 3.3\times$ con perdita relativa contenuta ($\approx 1.9\%$); MLP continua a preservare buona parte della performance fino a soglie elevate; KAN si mantiene relativamente robusto.
- **90% pruning:** le reti neurali mostrano degrado sensibile oltre certe soglie (specialmente MLP a 90% la performance cala in modo marcato), mentre XGBoost e Random Forest conservano ancora ritenzioni accettabili se il pruning sugli alberi è effettuato in modo controllato; Random Forest risulta particolarmente resistente al rank-based pruning fino a livelli molto alti.

Conclusioni

1. **XGBoost:** è la scelta primaria quando la priorità è massimizzare la performance assoluta (F1-weighted baseline: 0.8087). Riducendo

i rounds fino al 50–70% si ottiene un buon compromesso (2–3.3× compression) con perdita contenuta ($\lesssim 2\%$).

2. **MLP / KAN:** L1 pruning consente compressioni significative con perdite limitate. MLP raggiunge un best trade-off operativo a **80%** (5.0× compression, perdita relativa $\approx 0.6\%$), mentre KAN mostra robustezza fino al 70% (3.0×, perdita $\approx 1.4\%$).
3. **Random Forest:** il rank-based pruning consente riduzioni sostanziali del numero di alberi mantenendo la qualità predittiva: il best trade-off è a 80% (39 alberi rimanenti, $\approx 5.1\times$ compression) con F1-weighted ≈ 0.7946 (perdita relativa $\approx 1.2\%$). Questo lo rende interessante quando si può sacrificare qualche albero per risparmio di memoria/latency senza dover ri-addestrare il modello.

Capitolo 10

Terzo Caso Studio: Classificazione di età tramite immagini

Capitolo 11

Discussione comparativa sui Risultati dei casi studio

Capitolo 12

Conclusioni

Bibliografia

- [1] Popescu et al. , *Multilayer perceptron and neural networks*, 2009. https://www.researchgate.net/publication/228340819_Multilayer_perceptron_and_neural_networks
- [2] Hornick et al. , *Multilayer feedforward networks are universal approximators*, 1988. https://www.cs.cmu.edu/~epxing/Class/10715/reading/Kornick_et_al.pdf
- [3] J. Schmidhuber, *Deep learning in neural networks: An overview*, vol. 61, 2015.
- [4] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, vol. 2, 1989. <https://doi.org/10.1007/BF02551274>
- [5] Leshno et al. , *Multilayer feedforward networks with a nonpolynomial activation function can approximate any functionn*, vol. 6, 1993. <https://www.sciencedirect.com/science/article/pii/S0893608005801315>
- [6] I. Goodfellow, Y. Bengio, A.Courville, *Deep Learning*, 2016.
- [7] Liu et al. , *Kan: Kolmogorov-arnold networks*, 2024. <https://arxiv.org/pdf/2404.19756>
- [8] A.N. Kolmogorov, *On the representation of continuous functions of several variables by superpositions of continuous functions of one variable and addition*, *Doklady Akademii Nauk SSSR*, vol. 114, no. 5, 1957.
- [9] V.I. Arnold, *On functions of three variables*, *Doklady Akademii Nauk SSSR*, vol. 141, no. 4, 1963.

- [10] A. Pinkus, *Approximation theory of the mlp model in neural networks*, *Acta Numerica*, vol. 8, 1999.
- [11] T. Poggio, F. Girosi, *Networks for approximation and learning*, *Proceedings of the IEEE*, vol. 78, no. 9, 1990.
- [12] A. Chaudhuri, *B-Splines*, 2021. <https://arxiv.org/pdf/2108.06617.pdf>
- [13] J. Henseler, *Back Propagation*, 1995. <https://link.springer.com/chapter/10.1007/BFb0027022>
- [14] D. Donoho, *High-Dimensional Data Analysis: The Curses and Blessings of Dimensionality*, 2000. https://www.researchgate.net/publication/220049061_High-Dimensional_Data_Analysis_The_Curses_and_Blessings_of_Dimensionality.
- [15] R. French, *Catastrophic forgetting in connectionist networks*, 1999. https://www.researchgate.net/publication/12977135_Catastrophic_forgetting_in_connectionist_networks
- [16] R. Yu, *KAN or MLP: A Fairer Comparison*, 2024. <https://arxiv.org/pdf/2407.16674v1.pdf>
- [17] L. Breiman, *Bagging Predictors*, vol. 24, 1996. <https://link.springer.com/article/10.1007/BF00058655>
- [18] L. Breiman, *Random Forests*, vol. 45, 2001. <https://link.springer.com/article/10.1023/A:1010933404324>
- [19] J.R Quinlan, *C4.5: Programs for Machine Learning*, 1993. https://scholar.google.com/scholar_lookup?&title=C4.5%20Programs%20for%20Machine%20Learning&publication_year=1992&author=Quinlan%2CJ.R.
- [20] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2009. <https://link.springer.com/book/10.1007/978-0-387-84858-7>

- [21] J. H. Friedman, *Greedy function approximation: A gradient boosting machine*, 2001. <https://jerryfriedman.su.domains/ftp/trebst.pdf>
- [22] T. Chen, C. Guestrin, *XGBoost: A Scalable Tree Boosting System*, 2016. <https://arxiv.org/pdf/1603.02754>
- [23] S. Fatima, *XGBoost and Random Forest Algorithms: An in Depth Analysis*, 2023. https://www.researchgate.net/publication/377135877_XGBoost_and_Random_Forest_Algorithms_An_in_Depth_Analysis
- [24] A. E. Hoerl, R. W. Kennard, *Ridge regression: Biased estimation for nonorthogonal problems*, 1970. <https://homepages.math.uic.edu/~lreyzin/papers/ridge.pdf>
- [25] R. Tibshirani, *Regression shrinkage and selection via the lasso*, 1996. [https://webdoc.agsci.colostate.edu/koontz/arec-econ535/papers/Tibshirani%20\(JRSS-B%201996\).pdf](https://webdoc.agsci.colostate.edu/koontz/arec-econ535/papers/Tibshirani%20(JRSS-B%201996).pdf)
- [26] H. Zou, T. Hastie, *Regularization and variable selection via the elastic net*, 2005. <https://academic.oup.com/jrsssb/article-abstract/67/2/301/7109482>
- [27] N. Srivastava et al. , *Dropout: A simple way to prevent neural networks from overfitting*, 2014. <https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf>
- [28] K. O'Shea, R. Nash, *An Introduction to Convolutional Neural Networks*, 2015. <https://arxiv.org/pdf/1511.08458>
- [29] A. Krizhevsky et al. , *ImageNet Classification with Deep Convolutional Neural Networks*, 2012. https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf
- [30] K. Simonyan, A. Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, 2015. <https://arxiv.org/pdf/1409.1556>

- [31] S. Ioffe, C. Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, 2015. <https://arxiv.org/pdf/1502.03167.pdf>
- [32] J.L. Ba et al. , *Layer Normalization*, 2016. <https://arxiv.org/pdf/1607.06450.pdf>
- [33] Y. Wu, K. He, *Group Normalization*, 2018. <https://arxiv.org/pdf/1803.08494.pdf>
- [34] R. Poojary, *Effect of data-augmentation on fine-tuned CNN model performance*, 2020. https://www.researchgate.net/publication/347437393_Effect_of_data-augmentation_on_fine-tuned_CNN_model_performance
- [35] J. Bergstra , Y. Bengio, *Random Search for Hyper-Parameter Optimization*, 2012. <https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf>
- [36] L. Franceschi et al. , *Hyperparameter Optimization in Machine Learning*, 2024. <https://arxiv.org/pdf/2410.22854.pdf>
- [37] J. Snoek et al. , *Practical Bayesian Optimization of Machine Learning Algorithms*, 2012 <https://arxiv.org/pdf/1206.2944.pdf>
- [38] H. Alibrahim, S.A. Ludwig, *Hyperparameter Optimization: Comparing Genetic Algorithm against Grid Search and Bayesian Optimization*, 2021. <https://web.cs.ndsu.nodak.edu/~siludwig/Publish/papers/CEC2021.pdf>
- [39] J. Wainer, G. Cawley, *Nested cross-validation when selecting classifiers is overzealous for most practical applications*, 2018.<https://arxiv.org/pdf/1809.09446.pdf>
- [40] A. Deng, *Time series cross validation: A theoretical result and finite sample performance*, 2023.<https://www.sciencedirect.com/science/article/abs/pii/S0165176523003944>

- [41] G. Tsoumakas, *An Ensemble Pruning Primer*, 1970. https://www.researchgate.net/publication/225606257_An_Elsevier_Pruning_Primer
- [42] S. Vadera, S. Ameen, *Methods for Pruning Deep Neural Networks*, 2021. <https://arxiv.org/pdf/2011.00241.pdf>
- [43] R. Meyers, *Ablation Studies in Artificial Neural Networks*, 2019. https://www.researchgate.net/publication/330672937_Ablation_Studies_in_Artificial_Neural_Networks
- [44] F.T. Liu et al. , *Isolation Forest*, 2009. https://www.researchgate.net/publication/224384174_Isolation_Forest
- [45] N.W. Chawla et al. , *SMOTE: Synthetic Minority Over-sampling Technique*, 2011.<https://arxiv.org/pdf/1106.1813.pdf>