POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Meccanica

Tesi di Laurea Magistrale

Sviluppo di un codice Machine Learning per la stima di ossidi di azoto e particolato in motori Diesel

Relatore:
Prof. Daniela Anna Misul
Correlatore:
Ing. Alessandro Falai

Candidato:
Giuseppe Nobile

Aprile 2021

Ai miei genitori, a tutti i loro sacrifici...

A mia mamma Nelly, il mio cuore.
A mio papà Nicola, la mia forza.

"Che tu possa trovare il tuo equilibrio.
Equilibrio è sacrificio di un animo nobile che cerca se stesso...
Equilibrio è solo un minuscolo punto che regge l'infinito
e chi lo trova conosce se stesso e il mondo"
— Nicola Nobile

Indice

Capitolo 1

Abstract

Negli ultimi anni, sia a livello nazionale che a livello internazionale, sono state imposte, in particolare nell'ambito automotive, delle stringenti normative riguardanti l'emissione di sostanze inquinanti come PM (Particulate Matter) e NO_x (ossidi di azoto e miscele), poiché ritenute dannose per l'uomo e per l'ambiente.
In questa direzione si muove il seguente lavoro di Tesi Magistrale dove si sviluppa un sistema automatizzato basato su *Machine Learning* per il calcolo predittivo di inquinanti, con la possibilità in futuro di implementare questi algoritmi nella centralina del motore per rendere possibile un controllo on-board in tempo reale e quindi consentire un ottimizzazione dell'after-treatment.
Il seguente elaborato è stato svolto in collaborazione con il dipartimento di Energia del Politecnico di Torino, il quale ha fornito i dati relativi alle due tipologie di motori Diesel in analisi: un motore da 2.0L ed un Cursor da 11.0L di cilindrata.
In particolare, nel primo caso si hanno 1112 punti di funzionamento motore rpm (giri motore) x pme (pressione media effettiva), mentre nel secondo si hanno 5260 punti rpm x pme suddivisi tra misure effettuate a banco e misure stimate in centralina.
L'obiettivo dell'elaborato è stato quello di realizzare un modello in ambiente Python, implementabile in futuro in ECU (Electronic Control Unit), che stimi con un elevato valore di accuratezza il numero di inquinanti emessi dai motori Diesel.
Il modello implementato in ambiente Python si fonda sui principi di Machine Learning.
Il ML rientra nel mondo dell'Intelligenza Artificiale e utilizza metodi matematico-statistici per simulare l'intelligenza umana, con lo scopo di migliorare autonomamente durante l'allenamento apprendendo dai dati forniti. Il modello in questione è un codice di Machine Learning basato sull'algoritmo GBRT (Gradient Boosted Regression Trees) dove, fornendogli coppie di input (dati raccolti, predittori) e di output (inquinanti prodotti, risultato atteso o target), tramite l'allenamento riesce a trovare, senza l'aiuto umano, una regola

(funzione o modello) con cui generare un output desiderato anche per un valore di input mai visto in precedenza.
Il codice inoltre raccoglie i valori in un foglio di calcolo elaborando i risultati ed individuando i parametri motoristici più influenti per la predizione di un inquinante. Una volta individuate le 7 features più importanti per una "missione" specifica, riproduce lo stesso codice usando questi parametri col fine di ottenere i risultati in un tempo computazionale minore e svincolarsi completamente dalla fenomenologia del caso, pur sacrificando pochi punti percentuale sull'accuratezza e sul rmse.
Per adattare all'algoritmo GBRT i dati *soot* del motore 11.0L ed avere un'efficienza maggiore della predizione, sono state applicate delle trasformate (power transform) alle variabili in gioco con lo scopo di superare i problemi legati al training sulle variabili target e ridurre i coefficienti di asimmetria di Fisher-Pearson della loro dispersione.
La soluzione migliore è stata quella di applicare la trasformata Box-Cox alla variabile target e la trasformata Yeo-Johnson alle features.
Per concludere, è stato realizzato un confronto numerico tra i valori del GBRT e quelli ottenuti con l'algoritmo Random Forest.
La comparazione fra i diversi algoritmi è stata fatta in termini di coefficiente di regressione quadratico r^2 ed errore quadratico medio rmse.

Capitolo 2

Nozioni introduttive

Dall'inizio dalla seconda metà del XX secolo, gli studi sui cambiamenti climatici hanno dato il via ad una lotta concreta per la salvaguardia dell'ambiente, a cui parallelamente si è aggiunta anche la problematica dell'esaurimento dei combustibili fossili.
In figura (2.1) si può notare come ad oggi più della metà della domanda globale di petrolio sia impegnata per il settore dei trasporti[1] e si nota anche come questo trend non è destinato a diminuire nel breve termine.

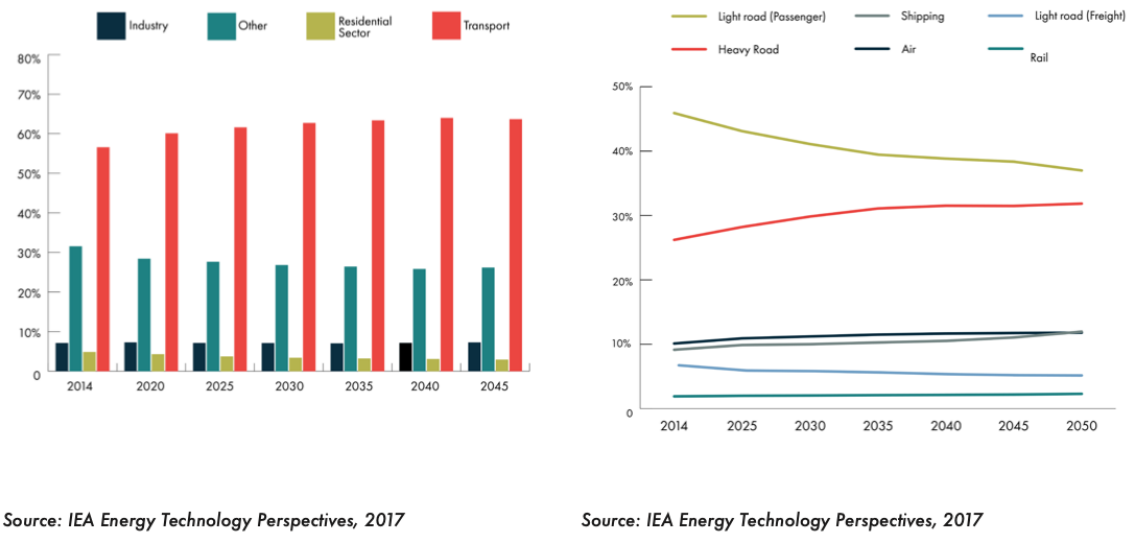


Figura 2.1. Richiesta mondiale di petrolio in funzione dei settori (sinistra), Sotto-settore dei trasporti (destra)

In quest'ambito il trasporto passeggeri su strada rappresenta il 50% del totale perciò oggi si cerca sempre più di ottimizzare l'efficienza energetica dei motori per ridurre il consumo

di combustibile contenendo contemporaneamente anche le emissioni inquinanti.
Negli anni le ricerche e gli studi condotti hanno mirato allo sviluppo di nuove tecnologie e soluzioni che potessero contenere queste problematiche senza però perdere in termini di prestazioni.
L'uso degli HEV o la ricerca di nuovi carburanti biocompatibili hanno portato a soluzioni spesso dispendiose, complesse da realizzare, meno sicure o economicamente non sostenibili.

Col fine di cercare un *trade-off* tra prestazioni ed emissioni, sono stati sviluppati modelli empirici basati sul fenomeno della combustione, ma la complessità del processo non ha permesso un'implementazione on-board a causa dei ristretti tempi computazionali a disposizione, rendendo questo modello inaccurato.
Solo con l'introduzione dei sistemi di *after-treatment* e con l'approfondimento di modelli di Machine Learning è stato possibile affrontare il problema della monitorazione di inquinanti con molta più precisione, senza andare perciò a sacrificare le prestazioni del veicolo.
Con la varia sensoristica si è potuto monitorare i parametri motoristici prevedendo, tramite essi, il valore delle emissioni usando algoritmi indipendenti dalla fenomenologia del caso.
In futuro si vorrà implementare questa soluzione nella centralina del motore ECU per rendere possibile un controllo on-board in tempo reale e ottimizzare consumo di combustibile, emissioni di inquinanti e prestazioni del motore.
I limiti legislativi di emissioni inquinanti sono definiti su cicli guida di omologazione ben precisi, ma le emissioni in uscita dal motore variano notevolmente con la guida real drive del guidatore.
L'idea di questo lavoro di tesi è stata quella di applicare e validare un modello di Machine Learning che predica in maniera accurata la massa di engine-out *soot* e NO_x in funzione dei parametri registrati con la sensoristica.
Il modello in questione è un codice di Machine Learning basato sull'algoritmo GBRT (Gradient Boosted Regression Trees) dove, fornendogli coppie di input (dati raccolti, predittori) e di output (inquinanti prodotti, risultato atteso o target), tramite l'allenamento riesce a trovare, senza l'aiuto umano, una regola (funzione o modello) con cui generare un output desiderato anche per un valore di input mai visto in precedenza.

Capitolo 3

Motori Diesel

Per comprendere a pieno il seguente lavoro può essere necessario affrontare dapprima il modello di combustione interna nei motori ad accensione per compressione: i motori Diesel[3].
Assieme a questo, in un secondo momento, sarà anche utile conoscere i meccanismi con i quali si formano gli inquinanti e i fattori che contribuiscono ad una variazione di essi.
Una volta qualificati e quantificati gli inquinanti si tratteranno infine le metodologie e le strumentazioni per limitarne la presenza allo scarico.
Tutti questi aspetti saranno poi il fulcro del lavoro sperimentale svolto.

3.1 Combustione motori ad accensione per compressione

I motori ad accensione per compressione, anche chiamati ad accensione spontanea, utilizzano combustibili con ritardi di accensione relativamente brevi, ovvero combustibili ad alta reattività come il gasolio o il biodiesel. È chiaro quindi come il processo di combustione sia significativamente diverso rispetto ai motori ad accensione comandata tramite candela in quanto il combustibile non può essere premiscelato con l'aria comburente e compresso senza che questo dia luogo a reazioni di combustioni immediate.
Per controllare tale fenomeno il combustibile deve essere iniettato ad alta pressione ($p_{iniez} = 1000/2000\ bar$) nel cilindro al termine della compressione, cioè nei pressi del punto morto superiore (PMS). Il getto fuoriesce ad una velocità elevata di circa 100 m/s dai fori dell'iniettore e si disintegra in una nube di gocce minutissime ($d \approx 10\ \mu m$). Questo processo prende il nome di "atomizzazione del getto" poiché, non appena il combustibile entra a contatto con l'aria comburente ad alta temperatura ($T_{fin} \cong 900K$) ed alta densità ($20 - 30\ kg/m^3$), vaporizza miscelandosi con l'aria fino a formare una miscela che si

autoaccende spontaneamente senza la necessità di un innesco esterno.
La presenza dei moti turbolenti all'interno della camera favorisce il miscelamento del combustibile con l'aria comburente.

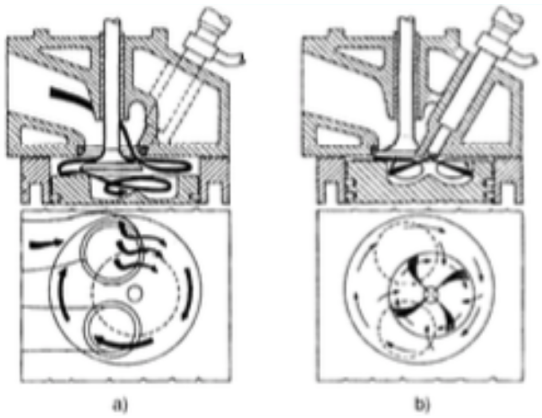


Figura 3.1. Esempio di moti turbolenti di Swirl (a) e di Squish (b)

I combustibili adottati nei motori ad accensione per compressione sono tutti caratterizzati da una molecola lunga e flessibile, caratteristica che ne giustifica l'elevata reattività.
L'idrocarburo rappresentativo per questi combustibili è il Cetano ($C_{16}H_{34}$). In altre parole i combustibili vengono classificati in funzione della maggiore o minore propensione ad autoaccendersi spontaneamente, dove la maggior predisposizione all'autoaccensione viene intesa come un pregio.
Così facendo questa miscela è in grado di accendersi anche con rapporti di aria/combustibile lontani dalle condizioni stechiometriche, muovendosi verso un ambiente più povero.

In figura (3.2) sono riportati gli andamenti nel tempo della pressione nella camera di combustione e delle quantità di combustibile iniettate, accese e bruciate.

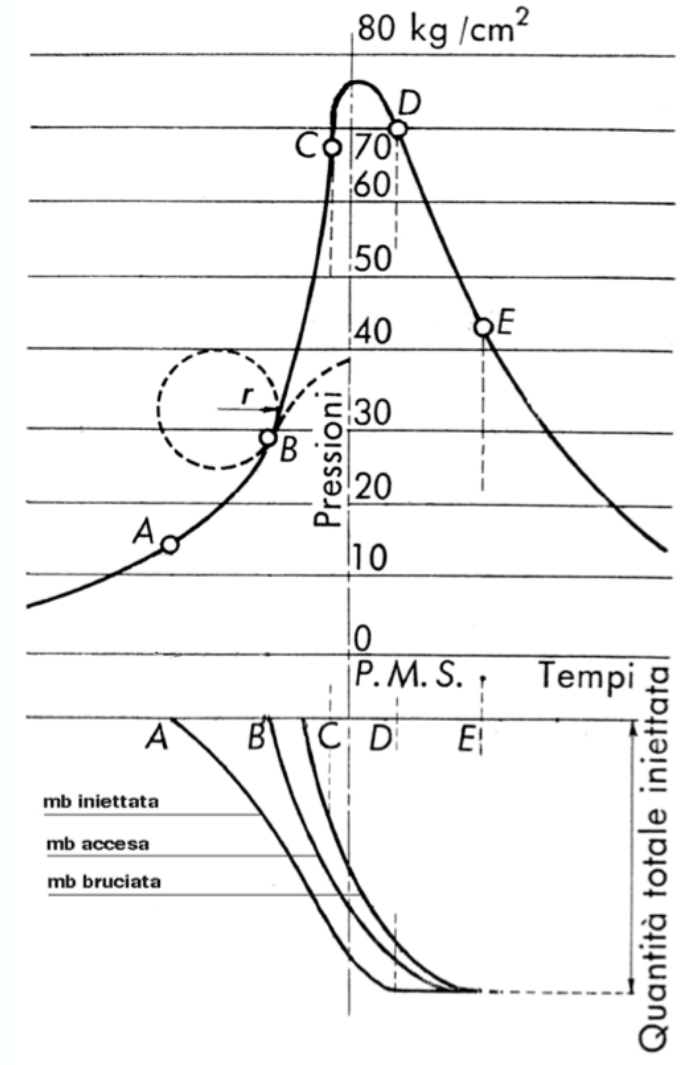


Figura 3.2. Andamento della pressione in camera di combustione e cumulata delle massa di combustibile

Come si evince il processo non comincia precisamente quando le prime goccioline di gasolio vengono iniettate nel cilindro (Punto A - Start of Injection SOI), ma si ha un certo ritardo (AB - Ingnition Delay τ dell'ordine delle frazioni di millisecondi) necessario al riscaldamento, alla vaporizzazione e al miscelamento del combustibile: si può notare un accumulo di combustibile poichè parte del gasolio viene iniettato in questo intervallo di tempo quando la combustione non è ancora iniziata.
Raggiunte le condizioni di autoaccensione (Punto B - Start of Combustion SOC), noteremo come al termine di τ, il combustibile accumulatosi in questo arco di tempo, brucerà

contemporaneamente e in modo quasi isocoro in una fase chiamata *combustione Premiscelata*. Questa fase è caratterizzata da un brusco picco di rilascio termico e ad esso è legato un brusco aumento della pressione in camera che diventa estremamente deleterio per il motore a causa delle vibrazioni che genera ed inoltre diviene il principale responsabile dell'insorgere del "rumore di combustione dei motori Diesel".[1]
L'accumulo di combustibile che si ritrova dopo l'ignition delay deve necessariamente essere tenuto sotto controllo, mantenendolo entro determinati limiti per poter attenuare l'entità del picco di combustione premiscelata. Una diminuzione troppo forte dell'injection rate (portata istantanea immessa in camera) potrebbe portare ad un degrado dell'efficienza della combustione poiché si avrebbe allungamento su un intervallo angolare di manovella più grande e ciò porterebbe ad un rischio di ossidazione durante la fase di espansione.
Per ammortizzare questo fenomeno le principali soluzioni sono:

- *Injection rate shaping*: si tratta di una modulazione della portata iniettata che permette di mantenerla bassa durante le prime fasi in modo tale da accumulare minor quantità di combustibile e poi successivamente, una volta arrivati a condizioni più favorevoli, aumentarla per poter avere efficienza maggiore.

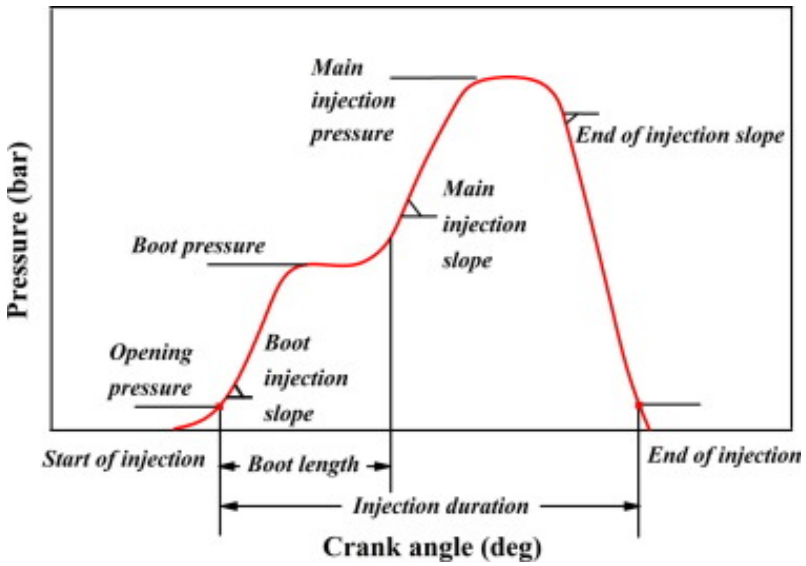


Figura 3.3. Andamento della pressione in un injection rate shaping

[1]Non è il valore massimo della pressione che è legato al rumore quanto il gradiente di essa, ovvero dp/dt

•*Iniezione pilota*: si tratta di una piccola quantità iniettata di combustibile (circa $5 - 10\%$ del totale) che precede l'iniezione main cioè quella principale. Si fa bruciare questa piccola quantità così da innalzare la temperatura localmente e accelerare il processo e permettere di avere una main più reattiva rendendo minore l'accumulo durante l'evento principale. Con questa soluzione si riesce ad attenuare anche il fenomeno del rumore andando a contenere il picco di pressione nella main. Maggiori vantaggi possono essere raggiunti con treni di iniezioni specificamente studiati che portano oltre che alla riduzione di rumore di combustione anche alla diminuzione di idrocarburi prodotti. Nella figura (3.4) viene mostrato uno studio su un motore Fiat 1.9 JTD che mette in risalto i vantaggi generati dall'uso di un Pilot+Pre+Main anziché l'uso esclusivo di una Pilot+Main.

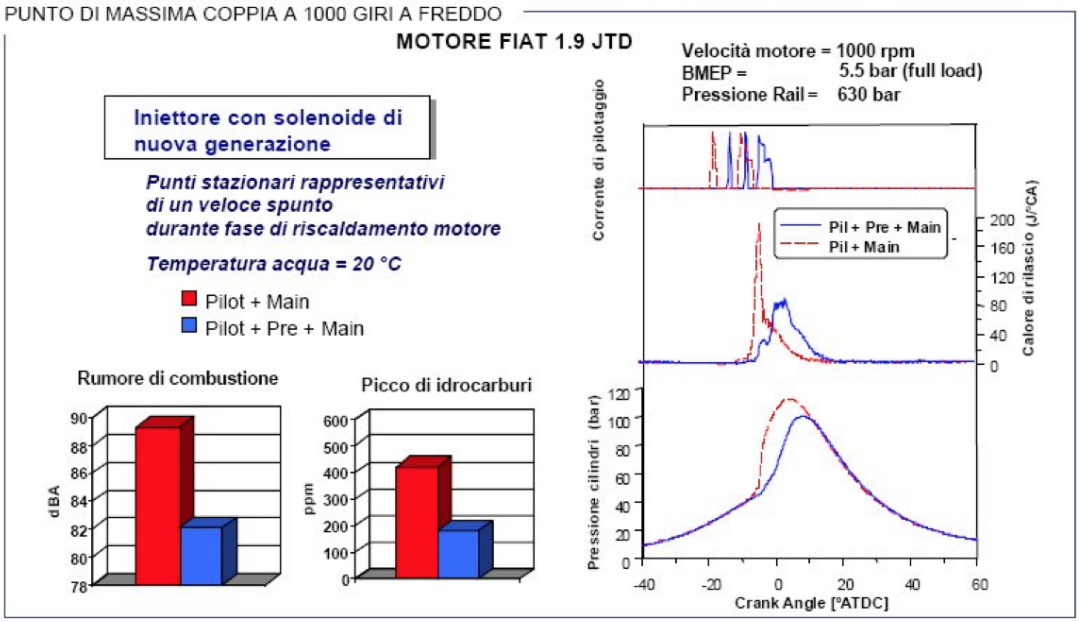


Figura 3.4. Andamento della pressione in un injection rate shaping

Tornando al ritardo di iniezione si mostra che nella parte inferiore della figura (3.2) il distacco fra la curva che descrive il combustibile iniettato e quella che descrive il combustibile acceso tende a diminuire fino a stabilizzarsi col passare del tempo poiché il ritardo τ non è costante: infatti le goccioline iniettate in camera a combustione già avviata sfruttano condizioni di pressione e temperatura sempre più vantaggiose in modo tale che il tempo necessario al loro riscaldamento, vaporizzazione e miscelamento va man mano riducendosi fino a stabilizzarsi ad un valore pressoché costante.
Il τ, come vedremo successivamente, è dovuto sia a *fenomeni fisici* e sia a *fenomeni chimici*.

Sovrapponendo al grafico precedente (3.2) l'andamento della curva HRR (Heat Release Rate) in funzione dell'angolo di manovella possiamo individuare quattro fasi differenti che caratterizzano la combustione nei motori Diesel.

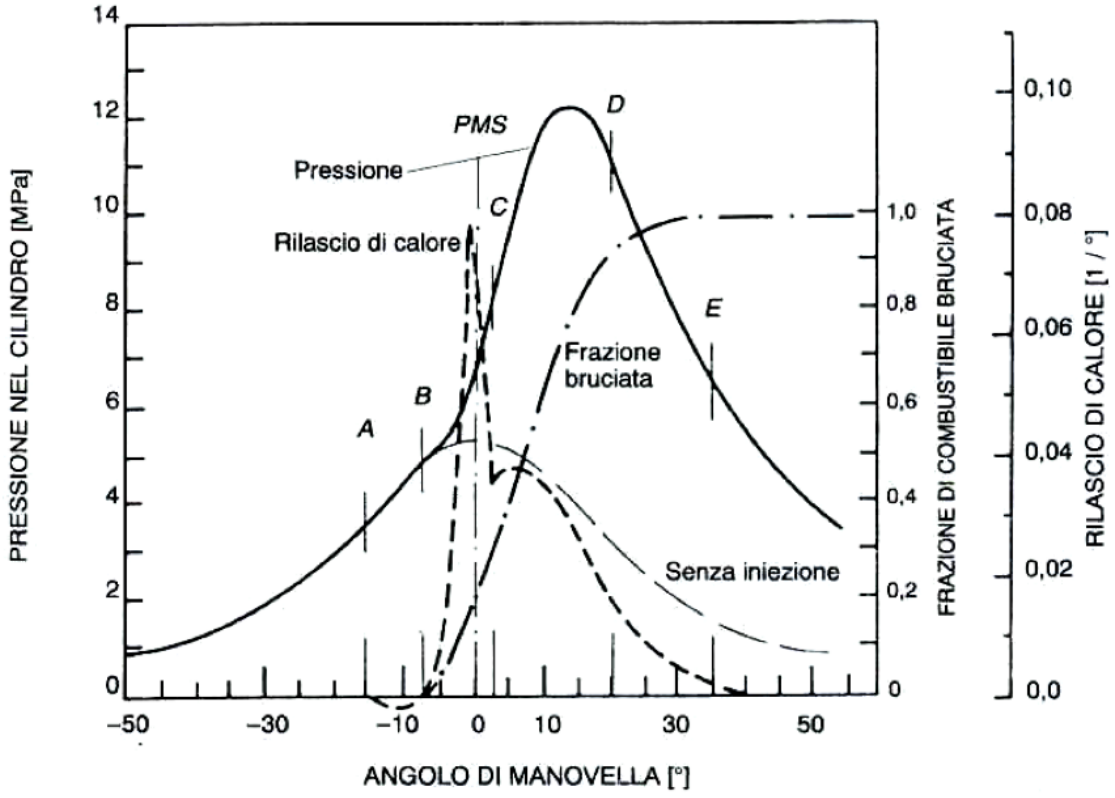


Figura 3.5. Andamento pressione con linea continua, frazione di combustibile bruciata con linea tratto-punto e rilascio di calore (HRR) con linea tratteggiata.

Le fasi identificate nella figura (3.5) sono:

1. AB - *Ignition Delay*, ritardo di accensione;
2. BC - *Premixed phase*, combustione premiscelata;
3. CD - *Mixing Controlled phase*, combustione diffusiva;
4. DE - *Late Combustion phase*, completamento della combustione;

Nei seguenti capitoli si andrà ad approfondire nel dettaglio ogni singola fase guardando agli aspetti più importanti che caratterizzano il processo di combustione.

3.1.1 Ignition delay

Il ritardo di accensione, anche chiamato *Ignition Delay*, in un motore ad accensione per compressione viene definito come l'intervallo di tempo che intercorre dal momento in cui vengono iniettate le prime goccioline di gasolio, start of Injection, all'inizio vero e proprio della combustione, start of Combustion.
Questo intervallo di tempo è dell'ordine dei millisecondi e dipende fortemente da fenomeni sia di natura fisica sia di natura chimica.
•*Fenomeni Fisici*:
-atomizzazione del getto di combustibile iniettato
-evaporazione delle goccioline
-miscelazione dei vapori di combustibile con l'aria comburente
•*Fenomeni Chimici*:
-reazioni con formazione di composti intermedi (radicali, perossidi)
-fenomeni di cracking che precedono gli stadi finali fortemente esotermici del processo di ossidazione
Ad ognuno di questi fenomeni è associato un τ che descrive l'intervallo di tempo che passa a causa di quei fattori. La sommatoria dei due ignition delay corrisponderà al ritardo di accensione totale come mostrato nella seguente equazione (3.1)

$$\tau_{totale} = \tau_{fisico} + \tau_{chimico} \tag{3.1}$$

L'intervallo di tipo fisico risulta essere predominante, anche perchè il combustibile usato presenta un elevata reattività, e ciò si traduce in una alta velocità di reazione che permette di impegnare poco tempo per il $\tau_{chimico}$.
Si analizzeranno dunque dapprima i fenomeni di tipo fisico e come essi contribuiscono al normale processo di combustione nei motori Diesel.

Gli iniettori hanno un numero di fori di iniezione che può andare dai 4 ai 6 di diametro molto piccolo (0.12 a 0.20 mm) in modo da ottenere uno spray di finissime goccioline. Il getto di combustibile che si affaccia nel cilindro si apre in maniera conica (figure (3.6) e (3.7)) principalmente a cause dell'azione combinata fra le forze aerodinamiche che sorgono nel moto relativo getto-gas e gocce-gas e le forze di tensione superficiale. Il getto viene polverizzato in gocce piccolissime[5] di diametro estremamente variabile.

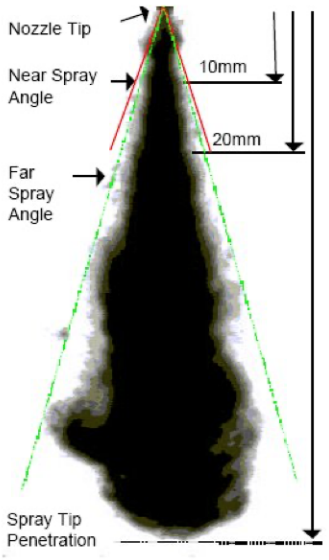


Figura 3.6. Rappresentazione dello spray conico prodotto da un elettroiniettore

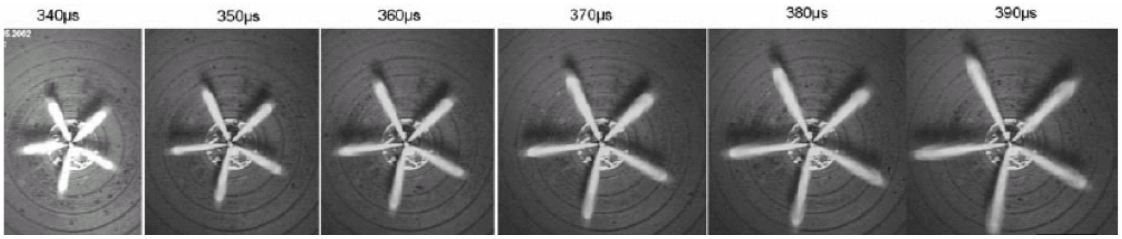


Figura 3.7. Sequenza iniezione common-rail multijet con elettroiniettore VCO a 5 fori

-**Atomizzazione**: L'atomizzazione è governata essenzialmente dai fenomeni fisici come la tensione superficiale, che permette di mantenere la forma sferica delle gocce, e l'intensità delle forza aerodinamiche che permettono di disintegrare il getto stesso.
Il combustibile introdotto in camera deve atomizzarsi perchè in un lasso di tempo molto breve deve potersi miscelare con l'aria ed autoaccendere.

La polverizzazione è legata all'insorgere di oscillazioni sulla superficie della colonna di liquido uscente dell'iniettore che vengono amplificate dalla presenza delle forze aerodinamiche. Quando l'ampiezza delle oscillazioni supera un certo valore critico dal getto si separano legamenti liquidi e gocce (atomizzazione primaria o "break-up" primario, fig. (3.8)).

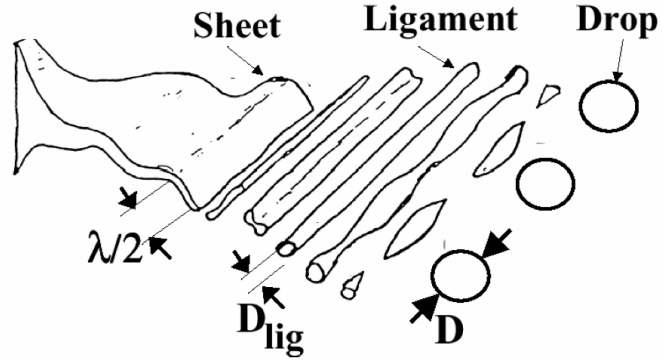


Figura 3.8. Sequenza iniezione common-rail multijet con elettroiniettore VCO a 5 fori

Successivamente le gocce che si sono formate vengono sottoposte ad una progressiva trasformazione, perdendo la forma sferica e dividendosi in una miriade di goccioline di dimensioni microscopiche (rottura o break-up secondario).
Gli studi sperimentali hanno dimostrato che per una goccia che si muove all'interno della camera di combustione a velocità crescente si osserva un deformazione superficiale significativa per We_g (numero di Weber: forze aerodinamiche/tens. superficiale, data dalla formula (3.2)) più alto dell'unità e che oltre un certo valore limite (circa 12 per i comb. liquidi) questa deformazione porta al break-up secondario. Al di sotto del valore $We = 12$ non si avrà una frammentazione del getto.

$$We_g = \frac{\rho_g v^2}{\sigma/D} = \frac{\rho_g v^2 D}{\sigma} \tag{3.2}$$

Con il variare del valore di We_g, in funzione dell'intensità delle forze aerodinamiche, si identificano cinque regimi di atomizzazione[4]:

- regime vibrazionale ($12 < We < 16$);
- regime di rottura "bag" ($16 < We < 45$) (fig. (3.9))
- regime di "stripping" ($100 < We < 1000$) (fig. (3.10));

- regime catastrofico ($We > 1000$) (fig. (3.11)).

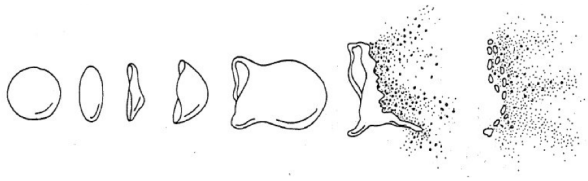


Figura 3.9. Regime di rottura “Bag”

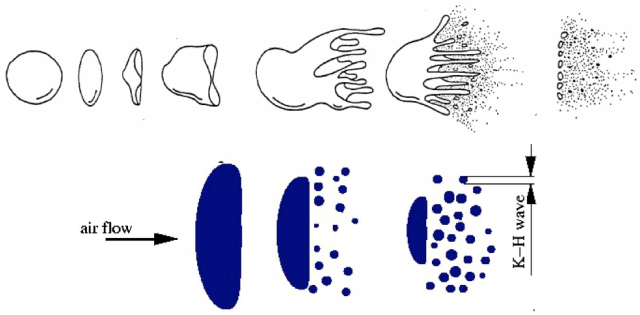


Figura 3.10. Regime di rottura “Stripping”

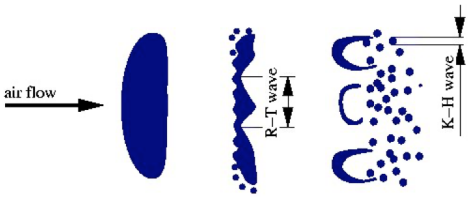


Figura 3.11. Regime di rottura Disastroso

Oltre che dai parametri presenti nella formula del numero di Weber, i fenomeni di atomizzazione del getto dipendono anche dal diametro del foro di efflusso, nonché il diametro iniziale del getto liquido, e dalla pressione di iniezione, quindi dalla velocità del getto rispetto all’aria.
Il diametro del foro dell’iniettore è un limite tecnologico sul decimo di millimetro a causa del coking, fori troppo piccoli potrebbero otturarsi a causa di depositi carboniosi.
La pressione di iniezione ha un andamento simile ad un’iperbole (fig. (3.12)) che correla

perciò un aumento di pressione ad una diminuzione del diametro della goccia. Far crescere la pressione oltre un certo valore però non porta miglioramenti apprezzabili poiché si raggiunge l'asintoto che è circa a $p_{iniez,max} = 2000\ bar$.

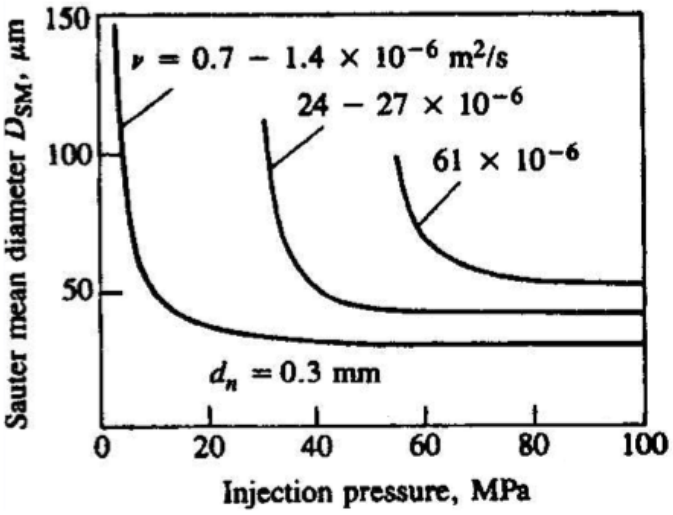


Figura 3.12. Diminuzione del diametro medio di Sauter in funzione della pressione di iniezione

-**Evaporazione del getto**: Avanzando col processo di combustione, la temperatura all'interno della goccia aumenta e a causa del calore ricevuto dall'esterno genera un aumento della rapidità del processo di vaporizzazione.
Ne consegue una riduzione ulteriore della dimensione della goccia, che causa quindi una diminuzione del flusso di calore dall'ambiente circostante ad uno successivo poiché ne assorbe una parte. Allora la velocità con la quale evapora non è costante nel tempo, ma raggiunge un picco per poi decrescere.

-**Miscelazione**: Essendo lo spray piuttosto "denso" non possiamo trascurare gli effetti di mutua interazione tra le gocce: nel dettaglio si possono generare importanti riduzioni locali della temperatura della miscela, nonché condizioni di saturazione dei vapori di combustibile.
Il miscelamento dei vapori di combustibile con l'aria comburente costituisce la fase finale dell'ignition delay, gestendo la successiva parte e dipendendo significativamente dal punto di funzionamento del motore.
Infatti, maggiore sarà il numero di giri del motore e minori saranno le tempistiche a disposizione del processo e perciò dovrà essere maggiore la turbolenza in camera per accelerare

il mixing tra vapori e aria e non avere una riduzione del rendimento.

3.1.2 Premixed phase

Terminato l'ignition delay si ha un innalzamento della temperatura prodotto dalla combustione dei primi nuclei iniettati in camera accelerando così notevolmente il processo di combustione del combustibile accumulatosi nell'intervallo di tempo τ, che brucia tutto insieme in maniera quasi isocora creando un brusco gradiente di pressione, che si traduce in un impennata della pressione come si evince dal grafico (3.5).
Infatti la "Premixed phase" coincide con il tratto BC, che presenta un alto picco di pressione. Questo valore alto può tradursi in un vantaggio in termini di rendimento termodinamico, ma è anche causa della presenza del rumore e dell'insorgenza di vibrazioni dannose per il motore.
Le onde vibrazionali prodotte possono compromettere le strutture meccaniche a causa delle forti forze di inerzia che potrebbero generare.
Durante questa fase inoltre si realizzano le condizioni idonee alla formazione di NO_x, a causa delle alte temperature raggiunte nella combustione premiscelata che si ottiene per miscele ancora localmente ricche.

3.1.3 Mixing controlled phase

Riferendosi sempre all'immagine (3.5) la fase di mixing corrisponde al tratto CD e si presenta dopo la repentina combustione dell'accumulo creatosi in precedenza. La velocità del processo è controllata essenzialmente dalla rapidità con cui i vapori di combustibile e l'aria comburente si miscelano.
Infatti i tempi richiesti per il mixing durante questa fase sono nettamente superiori sia a quelli imprescindibili dell'evaporazione, sia a quelli richiesti dalla cinetica chimica.
Durante questo tratto si raggiungono le pressioni più elevate.
Bisogna però assicurarsi che tutto il combustibile iniettato trovi ossigeno in quantità sufficiente per reagire e procedere con le reazioni chimiche. Con i motori Diesel si opera nel campo della dosatura povera[2].
Durante questa fase si formeranno, per processi di deidrogenazione, condensazione e pirolisi nuclei carboniosi incombusti (soot), costituiti da particelle solide contenenti fino a 10^5

[2]Dosatura povera valore di $\lambda > 1$, Dosature ricche $\lambda < 1$ e per condizioni stechiometriche $\lambda = 1$

atomi di C con $H/C \cong 0.1$.
Questa fase corrisponde alla combustione diffusiva, in cui si brucia circa il 90% di combustibile totale.

3.1.4 Late combustion phase

Nonostante l'iniezione sia ormai giunta al termine, le reazioni chimiche continuano ancora ad esaurirsi in modo graduale.
In questa fase la combustione può chiamare in causa anche i nuclei carboniosi (soot), i PAH (policiclici aromatici), formatisi durante la fase diffusiva precedente.
Fondamentale in questo tratto DE la presenza della turbolenza che permette un rimescolamento della miscela all'interno della camera prima di arrivare allo scarico del motore.
La "Late combustion phase" però non si deve prolungare eccessivamente per non ridurre il rendimento del motore (è noto, infatti, che per avere un alto rendimento occorre concentrare quanto più possibile il rilascio di calore nell'intorno del PMS).

3.2 Emissioni inquinanti nei motori Diesel

La trazione dei veicoli terrestri è negli ultimi anni la principale causa di emissioni di ossidi di azoto e particolato nell'atmosfera. I motori Diesel che andremo a vedere contribuisco a questi dati poiché i motori ad accensione per compressione producono un elevato numero di inquinanti allo scarico.
Nonostante un incremento dei veicoli venduti[6] le emissioni di inquinanti negli ultimi anni ha subito un rallentamento a causa delle nuove tecnologie introdotte utilizzate a valle dello scarico (after-treatment) e a cause delle stringenti normative che col passare degli anni diventano sempre più rigide.
Nel 2016 le percentuali di emissioni provenienti dal traffico[2] stradale rispetto a quelle complessive erano del:
Circa il 50% per gli ossidi di azoto (NO_X)
Circa il 12% per gli idrocarburi (HC)
Circa il 20% per le polveri fini (PM_{10})
Questi valori senza dubbio sono valori allarmanti poiché non influiscono negativamente sulla salute dell'uomo e sull'ambiente circostante.

Grazie allo sviluppo della tecnologia e a l'insinuarsi di nuove strumentazioni nel campo automotive, è stato possibile tramite una strumentazione per l'analisi ottica andare a misurare e valutare gli inquinanti e i meccanismi di formazione. I modelli più recenti sono basati su evidenze scientifiche anziché basati su rilevamenti di pressione e di rilascio del calore come avveniva nel passato.

Per comprendere la formazione delle particelle inquinanti, è importante analizzare il comportamento e le trasformazioni del getto di combustibile che passa attraverso i piccoli fori dell'iniettore. La penetrazione del getto in fase liquida all'interno della camera è contenuta dall'evaporazione del getto stesso, il quale evapora molto velocemente essendo sottoposto a condizioni di temperatura e pressioni elevate.
Nella figura (3.13) si può assistere all'evoluzione di uno dei getti (l'iniettore è multi-foro):

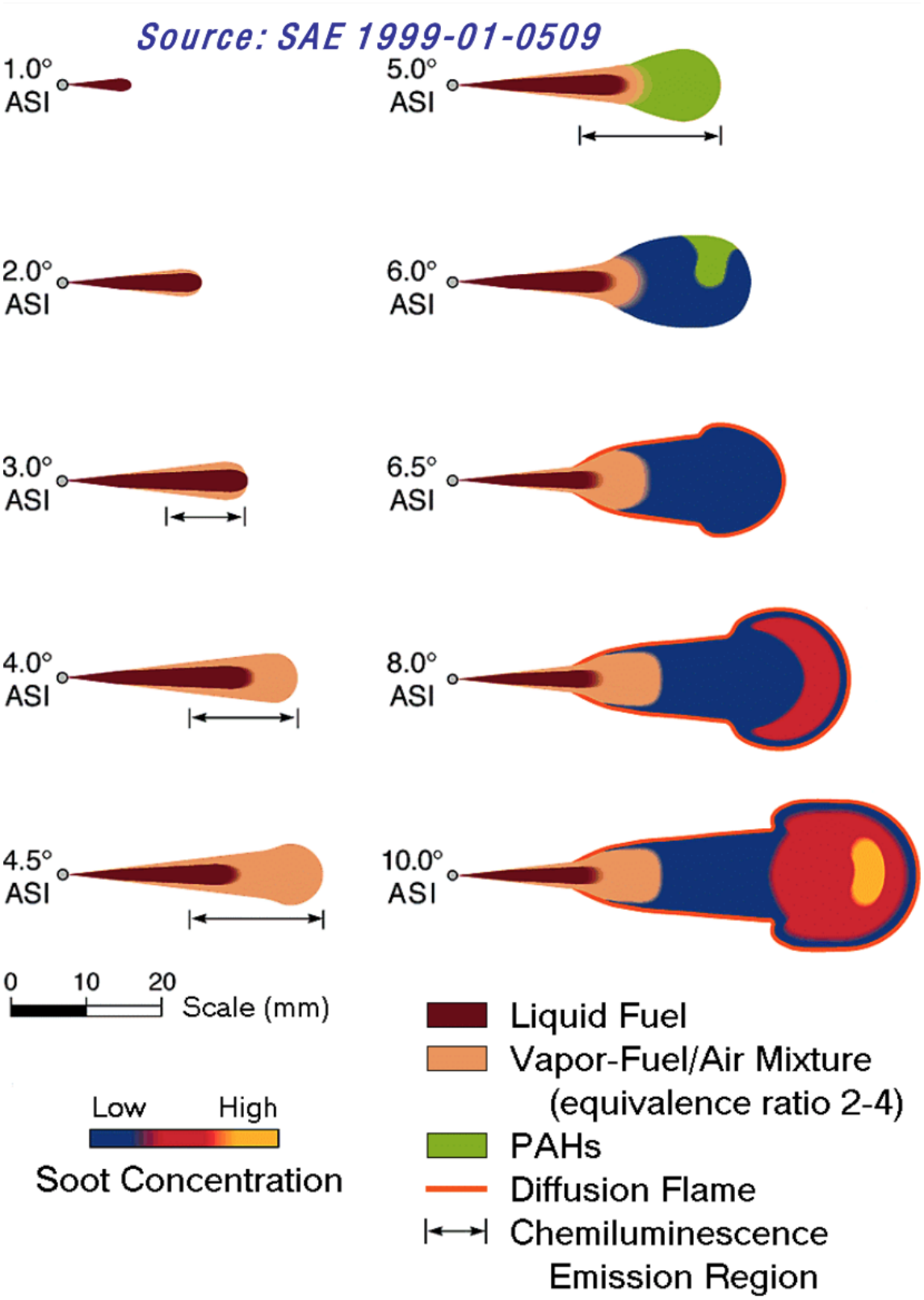


Figura 3.13. Evoluzione di un getto di combustibile iniettato nella camera di un motore Diesel che va dal primo grado ASI fino al decimo grado ASI.

Nella figura è riportata l'evoluzione del combustibile che entra in camera in funzione dell'ASI cioè (after start of injection) a partire dall'avvio idraulico dell'iniezione.
Al 1° ASI il combustibile viene iniettato in forma liquida.
Al 2° ASI non appena il getto entra in contatto con l'aria calda ad alta densità si inizia a creare una formazione di miscela aria/vapori di combustibile che inizia ad avvolgere il getto.
3° ASI il getto continua a penetrare in forma liquida e continua ad avvolgersi sempre più della miscela aria/vapori di combustibile ai bordi di esso.
4° ASI, raggiunta la massima penetrazione, la miscela aria/vapori si concentra sulla punta del getto, creando un'area con dosatura molto ricca: a causa delle temperature di circa 1600 Kelvin la miscela si autoaccende originando la combustione premiscelata.
Tale combustione in eccesso di combustibile non può essere completa, infatti tra i prodotti di combustione si hanno, oltre ad alcuni elementi chimici come CO e H_2, anche elementi del combustibile inalterati e altri che hanno subito solo prereazioni.
Queste ultime creano aldeidi, chetoni e policiclici aromatici, che contribuiscono alla formazione del soot.
Non si può precisamente determinare quando inizia la combustione premiscelata e a quale angolo di manovella accede, ma si prevede che quest'angolo sia compreso tra i 3 e i 5 gradi ASI. Tramite le ultime tecnologie di osservazione però si stabilire e osservare che il processo inizia quando si osserva l'emissione di chemiluminescenza, indice della presenza di radicali.
Andando avanti con i gradi di manovella arriviamo al 6°ASI dove i PAHs nonchè i policiclici aromotici danno vita al soot a seguito della reazione di deidrogenazione.
Il soot, come vedremo, è una frazione che compone il SOL assieme alle ceneri.
A 6,5° ASI: si verificano i fenomeni diffusivi alla base della combustione diffusiva, che si protrae fino a 10° ASI.
Questa combustione avviene nell'intorno dello stechiometrico con $\lambda = 1$ e raggiunge alte temperature fino a circa 2800 Kelvin. A causa di questa alta temperatura raggiunta si incrementa la produzione di NO_X all'interfaccia del pennacchio, subito dopo la fiamma diffusiva. Il successivo rapido raffreddamento non consente il completamento della reazione inversa (ritorno a azoto e ossigeno), lasciando quindi allo scarico molecole incomplete di di NO e NO_2.
La combustione va man mano a ridursi fino all'esaurimento a causa della sempre minor quantità di combustibile e comburente.
Tutto ciò è pericoloso per il soot perchè può essere impedito il corretto mescolamento con l'aria e quindi la corretta combustione, generando quindi grosse particelle carboniose.

Il pennacchio finale (3.14) è quello che rimarrà per tutta la durata dell'iniezione, perciò risulta interessante analizzarne la struttura.

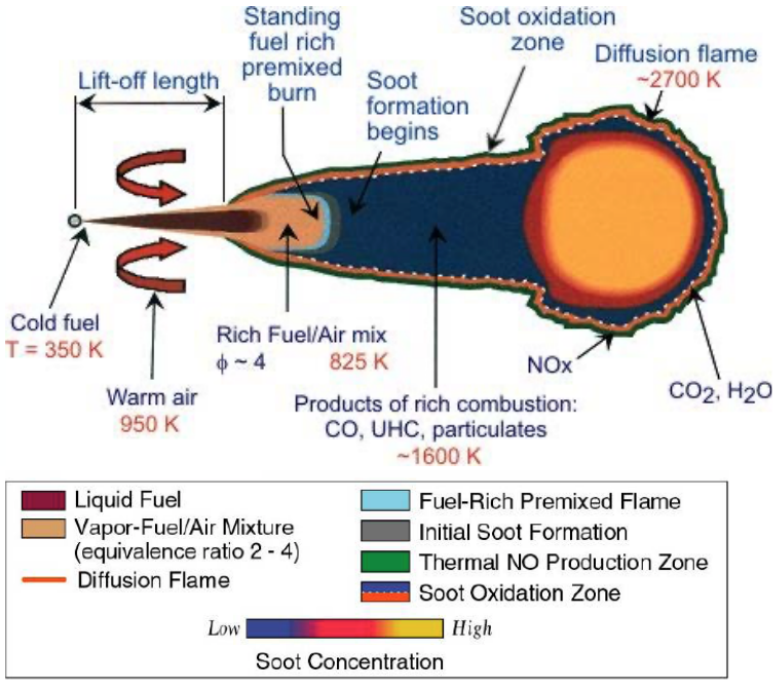


Figura 3.14. Rappresentazione finale del getto di combustibile in camera

Si può notare che gli ossidi di azoto NO_X si formano in condizioni di alte temperature e, diversamente dal *soot*, in presenza di ossigeno.
È la zona della fiamma diffusiva quella migliore per la loro formazione.
Il soot inizia la sua formazione non appena si conclude la combustione premiscelata che genera prodotti di una combustione ricca come HC e PAH nonché policiclici aromatici, precursori del soot.
Soltanto arrivati alla fiamma diffusiva con una temperatura più elevata ed una concentrazione di O_2 maggiore, allora si permetterà la formazione di NO_X e la parziale ossidazione del soot.
Un valido compromesso tra le due sostanze inquinanti può essere trovato tramite il diagramma di Kamamoto-Bae (figura (3.15)) che individua delle precise zone di formazione di soot e NO_X in funzione del Equivalence Ratio (inverso della dosatura) e Temperatura espressa in kelvin.

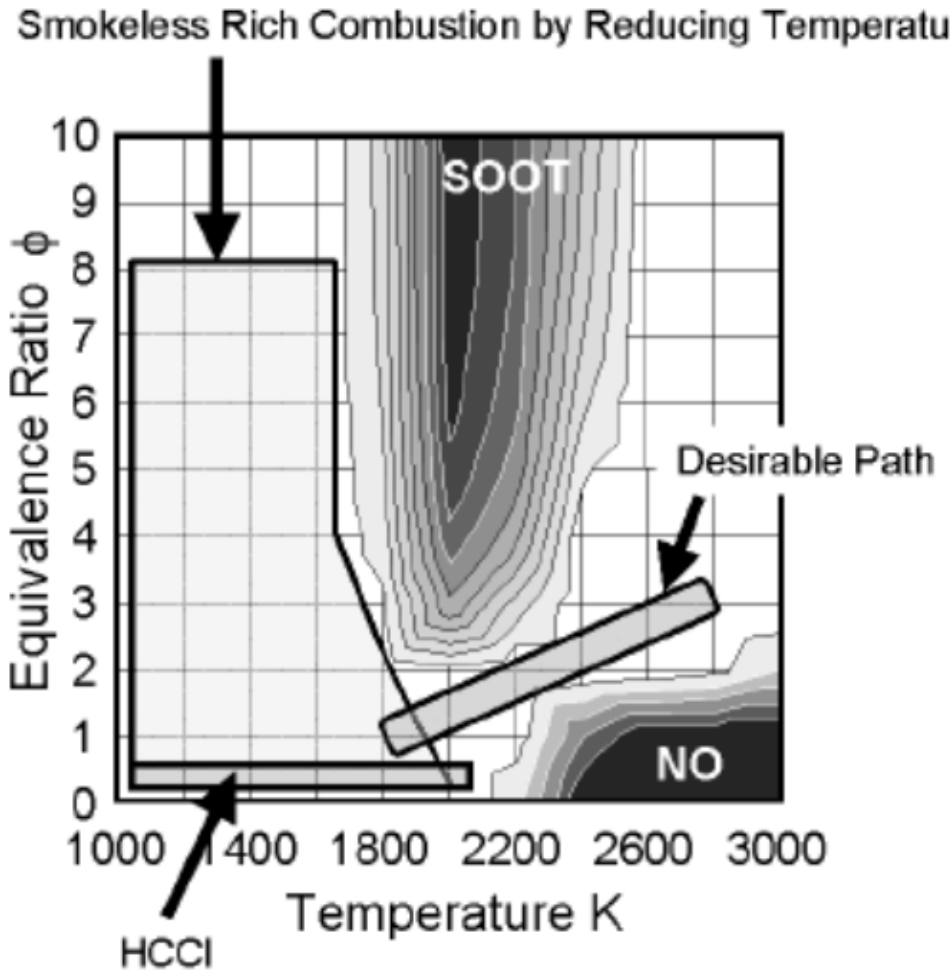


Figura 3.15. Diagramma di Kamamoto-Bae e zone ottimali per il trade-off $NO\_X/soot$

3.2.1 Formazione NOx

Rispetto ai motori SI, nei CI ci sono maggiori probabilità che l'ossidazione dell'azoto possa progredire ulteriormente poiché il tempo e l'ossigeno a disposizione sono maggiori, ed ecco il perché ci sia una maggiore percentuale di NO_2.
Nei Diesel gli NO_X sono composti da circa il 70% di NO e 30% di NO_2.
Il meccanismo principale di formazione degli NO_X è quello termico ed è definito meccanismo di Zeldovich esteso (EZM) dove le prime tre relazioni principali che hanno luogo sono:

$$N_2 + O \leftrightarrow NO + N \tag{3.3}$$

$$N + O_2 \leftrightarrow NO + O \tag{3.4}$$

$$N + OH \leftrightarrow NO + H \tag{3.5}$$

La reazione (3.3) è quella che richiede una E_A maggiore, circa 365 kJ, cioè l'energia di attivazione necessaria per far partire il processo. Si giunge a questa energia grazie alle elevate temperature generate dalla combustione e inoltre è necessaria la presenza di ossigeno per far si che il meccanismo EZM funzioni.
Questa reazione avviene nei prodotti della combustione, unburned gas, e non nella fiamma diffusiva, rimanendo ad alta temperatura per molto tempo da permettere alle reazioni di avvenire, il tempo di residenza garantisce la formazione delle reazioni.

Gli NO_X in realtà si formano generalmente tramite tre meccanismi (immagine (3.16)), quindi oltre quello termico se ne presentano altri due:
1. Istantaneo (Prompt NO_X): Gli NO_X che si formano so principalmente nella parte iniziale della combustione ed hanno un'energia di attivazione molto bassa che permette di non essere dipendente dalla temperatura. In questa fase si è in forte presenza di sostanze intermedie.
Vengono definiti processi istantanei perchè hanno una cinetica di reazione molto veloce.
2. Termico (Thermal NOx): Si formano a partire dall'azoto atmosferico presente in grosse quantità. Questo processo di produzione degli NO_X è consentito per le elevate temperature e quantità d'ossigeno a disposizione dopo la combustione. I motori Diesel operano con dosatura povera e quindi con alte percentuali di ossigeno in camera.
3. Fuel NO_X: Gli NO_X che si formano dall'azoto proveniente dalla struttura chimica del combustibile e contribuiscono in maniera minore alla produzione di inquinanti.
Come detto in precedenza il contributo termico (Meccanismo di Zeldovich) è quello maggiore per la formazione di inquinante NO_X allo scarico dei motori a combustione interna.

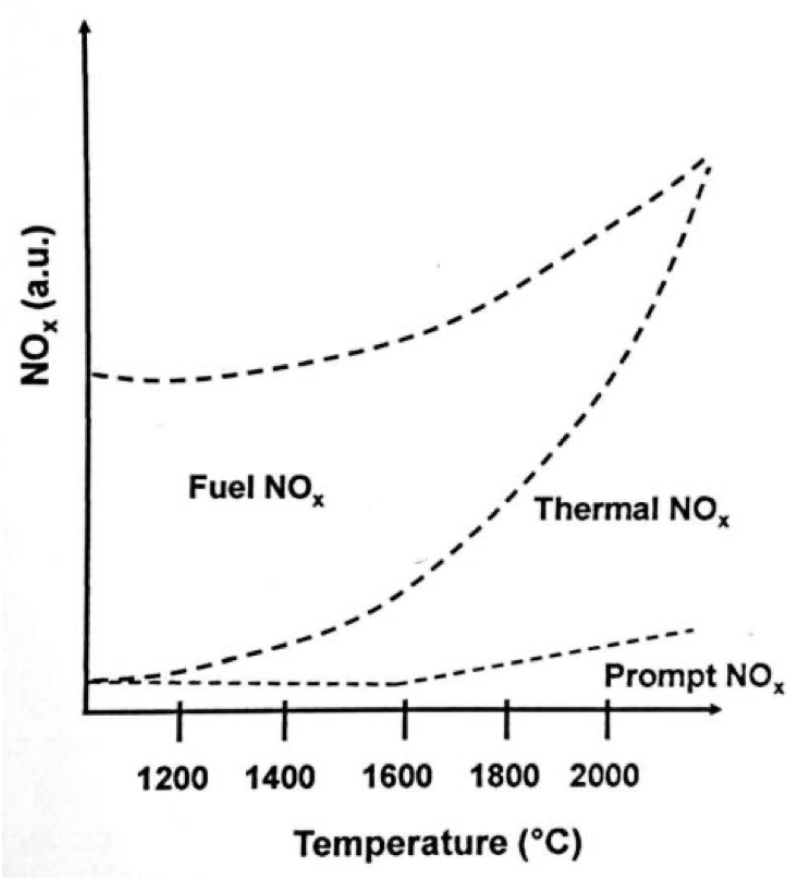


Figura 3.16. Relazione tra i meccanismi di formazione degli NO_X e la temperatura di combustione

Da quanto detto precedentemente i fattori che influenzano maggiormente la produzione di NO_X allo scarico del motore sono:
-La temperatura massima a cui si arriva;
-La presenza di radicali liberi O_2 a disposizione;
-I tempi di residenza per poter far avvenire le reazioni.
Tramite questi fattori viene permesso di superare la soglia della E_A della prima reazione facendo iniziare il processo.

Per capire come si evolve lo sviluppo degli NO_X durante la combustione si vanno quindi ad individuare quei parametri del motore[16] la cui regolazione va ad influenzare questi processi.
Per ridurre i valori di inquinante allo scarico si può intervenire sia su parametri di gestione dell'aria e sia su parametri riguardanti l'iniezione del combustibile.

Ad esempio è fondamentale anche il parametro che controlla l'EGR ciò i gas ricircolati che portano ad una forte diminuzione degli inquinanti prodotti.
Nel caso dell'EGR più vado ad aumentare la percentuale di gas ricircolati più vado ad influire sul parametro della temperatura andando a diluire la miscela. Così facendo si abbassa la temperatura di picco e quindi si riduce il numero di NO_X.
Aumentando la pressione di iniezione del combustibile in camera invece si porta ad un aumento di ossidi di azoto, ma contemporaneamente ad una riduzione di soot. Questo poiché aumentando la pressione di iniezione il combustibile viene introdotto in camera di combustione con una velocità maggiore, generando turbolenze ed aumentando la rapidità del processo permettendo di toccare temperature maggiori e quindi NO_X maggiori.
Anche l'aumento della fasatura dell'iniezione main porta un effetto sull'emissione di ossidi allo scarico del motore Diesel.
Ritardando l'iniezione del carburante in camera si ottiene un effetto di diminuzione del numero di NO_X poiché si consente alla combustione antecedente di avvenire in una maniera più completa.
Al contrario una diminuzione della fasatura significherebbe anticipare l'iniezione e la combustione, causando il rilascio di più incombusti.
Degli studi sperimentali hanno portato perciò ad individuare dei parametri del motore Diesel che calibrati attentamente possono influire in maniera significativa sulla produzione di NO_X allo scarico.
I parametri più influenti sulla gestione dell'aria per la produzione di inquinanti NO_X allo scarico dei Diesel sono:
-Pressione di sovralimentazione: pressione regolata agendo sulla turbina a geometria variabile del turbocompressore.
-Carico del motore.
-Quota di gas combusti ricircolati: regolata agendo sulla valvola EGR.
-Dosatura: rapporto aria su carburante, nei motori Diesel si muove nel campo del povero cioè in presenza di grandi quantità di ossigeno. Gli NO_X sono massimi nel leggermente povero perché la temperatura massima raggiungibile è nel leggermente ricco, ma plafona rimanendo alta anche nel leggermente povero, dove però in aggiunta abbiamo una quantità maggiore di O_2 che favorisce la formazione di ossidi di azoto. Subito dopo il valore di $\lambda = 1.1$ va velocemente a ridursi la formazione di NO_X, poiché l'alta presenza di ossigeno va a mitigare le temperature possibili da raggiungere.
D'altro canto invece i parametri di iniezione del carburante principali sono:
- La pressione di iniezione: la pressione alla quale si introduce carburante nel cilindro, nei dataset che useremo differenzieremo la pressione d'iniezione durante la pilot o durante la

main.
-Fasatura dell'iniezione: come già anticipato la tempistica alla quale avviene l'iniezione, anche qui divese fra pilot e main, può variare il numero di emissione dell'inquinante. Per valori positivi avviene in ritardo rispetto alla iniezione prevista, per valori negativi invece in anticipo.
- La massa di combustibile dell'iniezione: rappresenta la quantità di combustibile iniettata nel cilindro, possiamo riferirci sia alla pilot che alla main.
Sono proprio questi parametri motoristiche che sono stati valutati sul banco prova sperimentale per poter andare a costituire i datasets utilizzati negli studi del seguente lavoro di tesi tramite un algoritmo di Machine Learning.

Dalla figura (3.17), inoltre, si nota come la percentuale di NO_2 sia più alta ai bassi carichi cioè quando c'è una maggiore disponibilità di O_2 e le temperature sono minori. Le percentuali di diossido di azoto sono maggiori anche per basse velocità cioè quando gli scambi termici sono maggiori i quali contribuiscono ad un abbassamento di temperatura in camera.

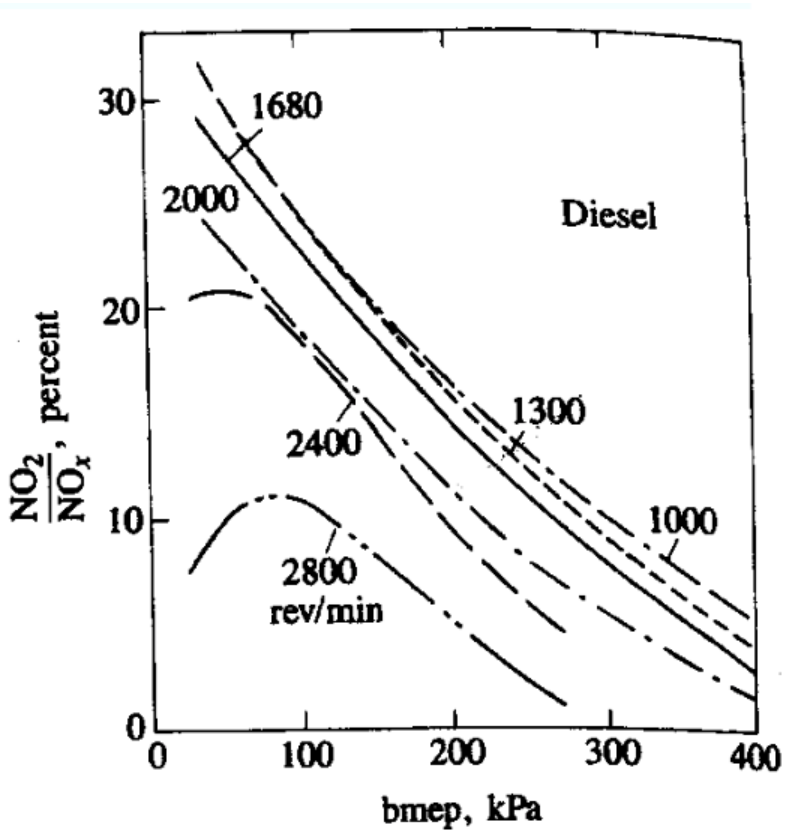


Figura 3.17. Rapporto NO_2 su NO

3.2.2 Formazione PM

Con PM o particolato (particulate matter), si indica l'insieme delle particelle che vengono generate in fase di combustione e successivamente portate in sospensione dai gas di scarico, e formato da[2]:

-Una parte solida (**SOL**): composta da elementi carboniosi e ceneri. Essa è la parte principale ed è chiamata soot.
Durante il processo di combustione, i precursori del particolato i PAH (policiclici aromatici) generano queste particelle carboniose che crescono e si aggregano. Inizialmente hanno strutture esagonali che possono combinarsi sovrapponendosi l'una sull'altra per dare strutture lamellari che talvolta si aggregano nuovamente in particelle più o meno sferiche (figura (3.18)).

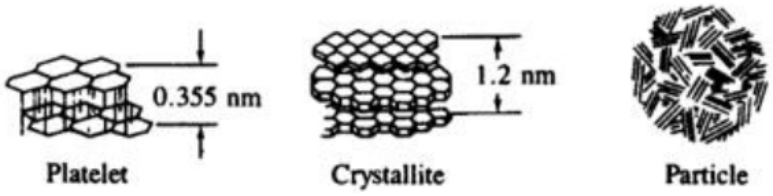


Figura 3.18. Fasi della composizione del soot

Queste particelle presentano delle porosità o degli interstizi non trascurabili. Le ceneri invece, che contribuiscono al SOL, sono sostanze incombustibili che non possiamo sottoporre a rigenerazione e deve richiedono un intervento manuale. Queste sostanze incombustibili sono additivi metallici, ossidi di ferro o altro che non deriva propriamente dalla combustione.

-Una parte solubile (**SOF**): costituita da composti organici (idrocarburi) che possono derivare sia dall'olio usato per la lubrificazione o dal combustibile.

-**Solfati** (SO_4): derivano da acqua o acidi solforici e possono formarsi a valle del processo di combustione, quando si arriva ad ottenere temperature minori, quindi nel condotto di scarico.

I rapporti di queste tre categorie, mostrati in figura (3.19), facenti parte del particolato dipendono dalle condizioni di funzionamento del motore. I valori numerici e la presenza di una rispetto all'altra dipendono molto da velocità di rotazione e carico del motore.

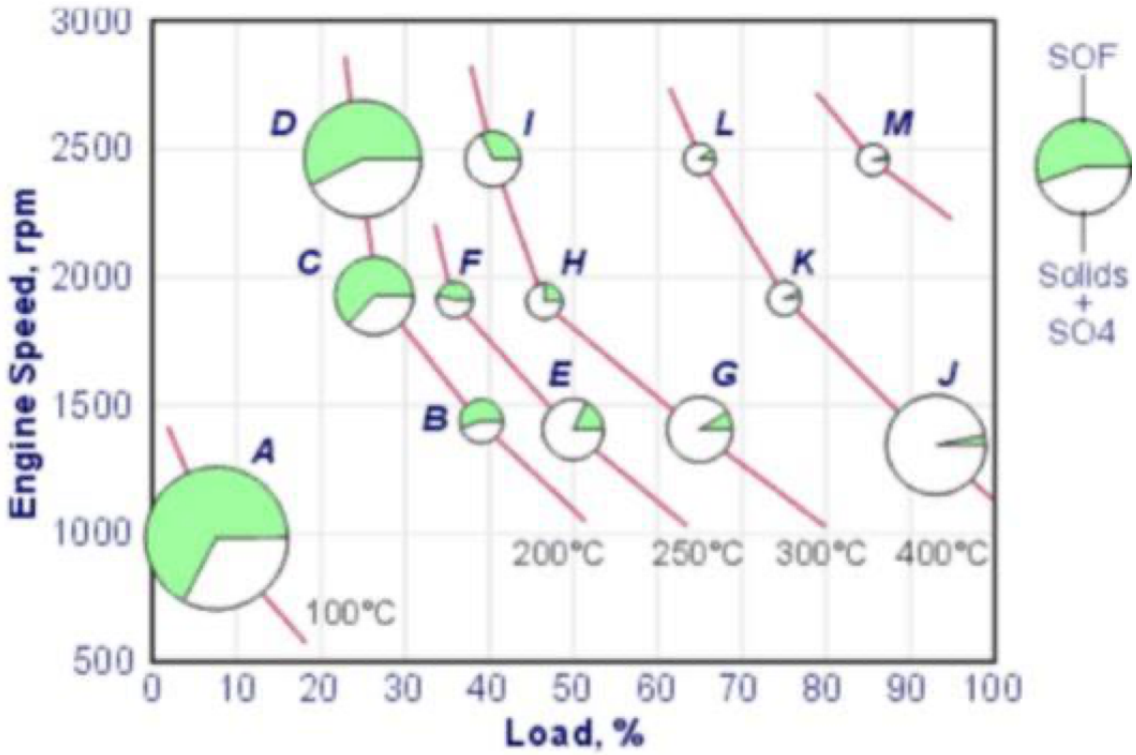


Figura 3.19. Variazioni di concentrazione delle parti costituenti il particolato al variare del Load e dell' Engine Speed

Ci sono due meccanismi con cui si può formare il soot:
1)Reazioni di condensazione degli idrocarburi aromatici del combustibile (assenti nei combustibili di origine vegetale) che portano ad un'agglomerazione degli anelli.
La formazione di soot, in questo caso avviene in condizioni di dosatura locale ricca con temperature relativamente basse (circa 1600 K)
2)Reazione di pirolisi che portano alla frammentazione di aromatici e alifatici: anche in questo caso poi gli scheletri carboniosi si addensano formando soot (a T maggiori, vicino alla zona della fiamma diffusiva).
Nella figura (3.20) si può notare l'intera composizione delle particelle di PM.

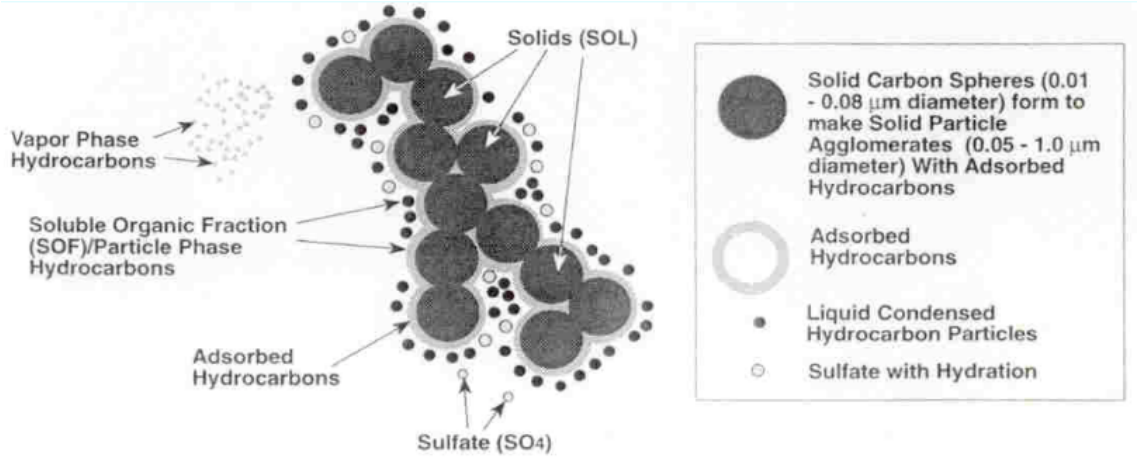


Figura 3.20. Composizione PM

La formazione del soot è stata affrontata all'inizio del capitolo con la figura (3.13) che ha ripercorso passo dopo passo la formazione di soot all'interno del getto di combustibile iniettato all'interno della camera di combustione del motore Diesel.

I dati sperimentali sostengono che determinati parametri motoristici hanno un impatto maggiore sulla formazione di soot nei motori Diesel[16].
Una di queste è l'*equivalence ratio* Φ che dipende dalla quantità di ossigeno e di combustibile introdotto in camera.
Sono importanti alcuni fattori che descrivono la pressione e la temperatura dei gas combusti durante l'iniezione main perché impattano sull'equivalence ratio e sull' air-to-fuel ratio, che tengono sott'occhio la quantità di ossigeno a disposizione. La quantità di ossigeno è direttamente proporzionale all'ossidazione del soot nella fase di fiamma diffusiva a dosatura stechiometrica.

Capitolo 4

Machine Learning

Il *Machine Learning* o apprendimento automatico delle macchine è una branca dell'Intelligenza Artificiale ed è il campo dell'informatica che fornisce ai computer la capacità di imparare a svolgere un compito senza essere stati esplicitamente programmati per la sua esecuzione, apprendendo dai dati forniti[7].
Perciò il Machine Learning utilizza metodi matematico-statistici per simulare l'intelligenza umana con lo scopo di migliorare da solo con l'allenamento apprendendo dai dati che gli forniamo.
Il machine learning tramite la costruzione di algoritmi che permettono l'apprendimento di informazioni a partire da dati disponibili, forniscono la capacità di predire nuove informazioni alla luce di quelle apprese in precedenza.
Attraverso la costruzione di un modello che impara automaticamente a predire nuovi dati a partire da osservazioni, questi algoritmi superano il classico paradigma delle istruzioni strettamente statiche.

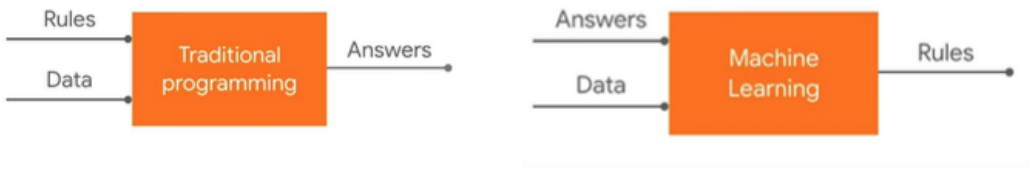


Figura 4.1. Confronto tra "Traditional Programming" e "Machine Learning Programming"

• Nella **programmazione tradizionale**, tutta la conoscenza sul mondo è codificata all'interno del programma. Quindi si dovrà specificare tutti i casi. Questo semplifica sicuramente il processo di elaborazione, ma nel caso in cui una regola non sia stata descritta,

il computer non riuscirà a svolgere il compito.
• Nell'**apprendimento automatico (machine learning)**, al contrario, alla macchina si passano esempi e, da questi, la macchina costruisce le regole che descrivono gli esempi e capirà se un da sola se un nuovo caso risponde o meno alla regola che ha ricavato.
Il machine learning trova il suo impiego principale in quell'insieme di problemi di computazione in cui la progettazione e l'implementazione di algoritmi ad-hoc non è praticabile o è poco conveniente.
Esempi di applicazioni si possono trovare nei motori di ricerca, nel riconoscimento ottico di caratteri (OCR) o in generale tra le applicazioni della computer vision, ampiamente sfruttati nel campo della robotica, della medicina o nel networking.
Questo mondo, resta però ancora parzialmente inesplorato nel settore automotive, in cui solo negli ultimi anni ci sono stati studi sull'argomento con lo scopo di consentire maggiori sicurezze o minori emissioni di sostanze inquinanti allo scarico delle automobili.
Questo strumento in questo settore è stato preso in considerazione solo recentemente a causa delle normative sempre più stringenti riguardo alle emissioni degli autoveicoli e alla salvaguardia dell'ambiente.

L'obiettivo di questo lavoro di tesi magistrale è usare e testare un codice di Machine Learning, scritto in ambiente Python, che sfrutta un noto algoritmo chiamato **GBRT** (Gradient Boosted Regression Trees) per sviluppare un sistema per il calcolo predittivo di inquinanti (PM e NO_x) in due motori Diesel differenti. L'algoritmo è stato implementato in ambiente Python tramite la libreria open-source "Scikit-Learn".

4.1 Classificazione degli algoritmi

Gli algoritmi possono essere suddivisi in funzione di come viene svolto l'allenamento del Machine Learning[8]:
1. Supervised
2. Unsupervised
3. Semisupervised
4. Reinforcement Learning

Questa classificazione copre tantissime realtà nel mondo Machine Learning come mostrato nella figura (4.2).

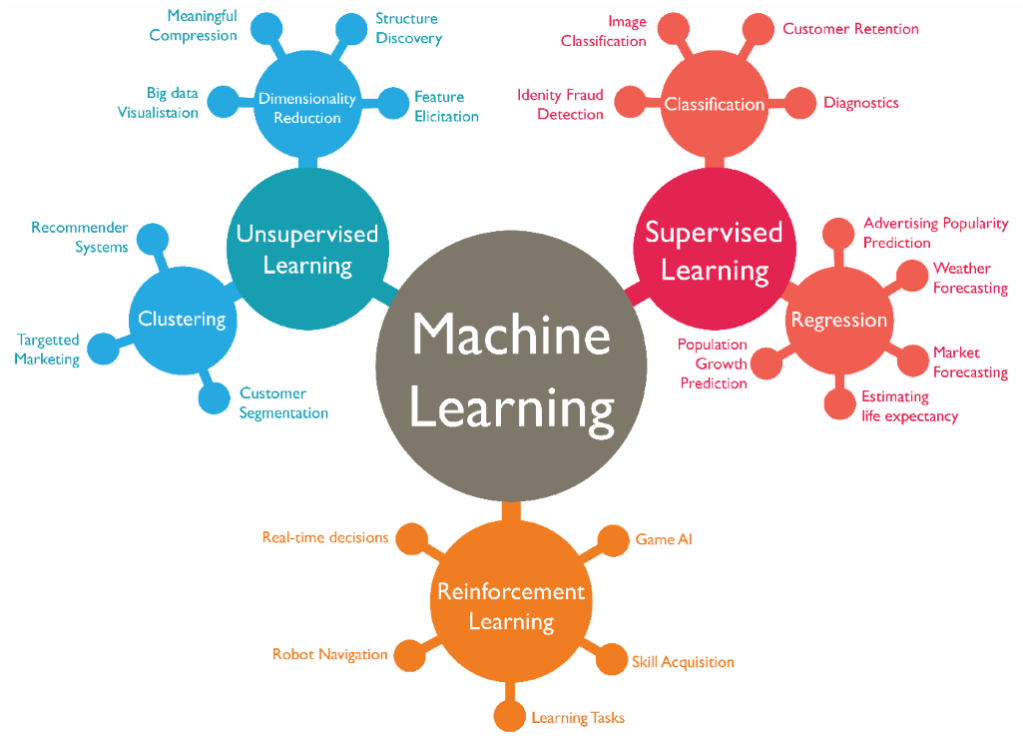


Figura 4.2. Alcuni algoritmi di machine learning, la loro classificazione e i principali utilizzi.

Ulteriori classificazioni possono essere fatte anche in base a:
-Se imparano o no incrementalmente durante l'applicazione.
-Se lavorano comparando i nuovi dati a quelli già conosciuti o se rilevano i modelli tramite i dati di training per poi costruire un modello predittivo.
Queste classificazioni non si escludono l'un l'altra e possono venire combinate per costruire un modello più solido.
Per comprendere la enormi possibilità che il machine learning fornisce affronteremo rapidamente la suddivisione degli algoritmi in base allo svolgimento dell'apprendimento per poi addentrarci più nello specifico nel modello utilizzato con il codice GBRT.

4.1.1 Supervised Learning

Gli algoritmi di maggior successo nell'ambito dell'apprendimento automatizzato sono quelli che generalizzano una regola partendo da esempi noti.
Nell'apprendimento supervisionato vengono forniti all'algoritmo coppie di input (dati raccolti) e di output (inquinanti prodotti, risultato atteso) e l'algoritmo, tramite l'allenamento, riesce a trovare una regola (funzione o modello) con cui generare un output desiderato anche per un valore di input che non aveva mai visto in precedenza senza l'aiuto umano.

Gli algoritmi di apprendimento supervisionato possono essere usati sia per compiti di classificazione e sia per compiti di regressione:

•**Classificazione** nel supervised learning: l'algoritmo è allenato con diversi dati conosciuti e ciascuno con la loro classe[1] di appartenenza, vengono «etichettati».
L'algoritmo poi analizza i dati di esempio e ricava una regola generale grazie alla quale, al presentarsi di un nuovo caso non etichettato, riuscirà a classificare l'oggetto etichettandolo.

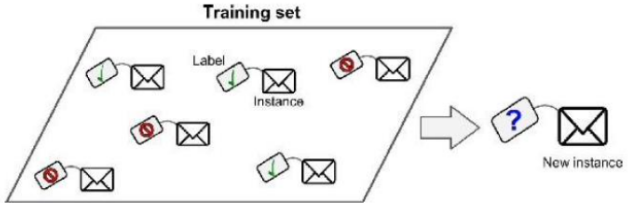


Figura 4.3. Un training set dotato di labels e un nuovo dato da classificare

•**Regressione** nel supervised learning: l'algoritmo prevede un certo valore numerico dopo essersi allenato su dati input-output conosciuti, chiamati predittori e target.
Risolvere un problema di regressione supervisionata significa apprendere una funzione che approssima le coppie di dati input- output che vengono fornite.

[1]La classe un'insieme dei dati aventi proprietà comuni.

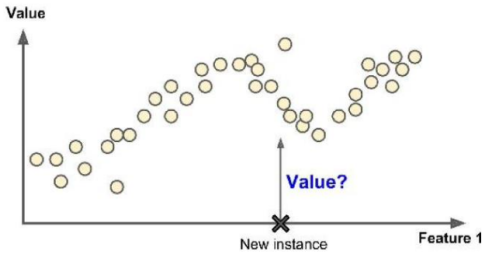


Figura 4.4. Regressione dei dati conosciuti e predizione di una nuova istanza

Nella seguente tesi verrà utilizzato un algoritmo di regressione, *Gradient Boosted Regression Trees*, che come si evince dal nome è un algoritmo di regressione supervisionato che sfrutta il pensiero che c'è dietro al più noto algoritmo *Decision Trees*.
I più importanti algoritmi di apprendimento supervisionato sono:
- Decision Trees
- K-Nearest Neighbors
- Linear Regression
- Logistic Regression
- Support Vector Machines
- Neural networks o reti neurali

Nel prossimo capitolo verranno trattati gli algoritmi e gli iter che hanno portato alla modellazione del GBRT.

4.1.2 Unsupervised Learning

A differenza del caso precedente il training set non è etichettato, cioè non possiedo output per gli input che fornisco.
Perciò non sono note le classi dei pattern[2] utilizzati per l'addestramento (non sono etichettati i dati).
Alla macchina viene chiesto, quindi, di estrarre una regola che raggruppi i casi presentati secondo caratteristiche che ricava dai dati stessi. Per questo è anche definito apprendimento di caratteristiche (feature learning).
Tra le soluzioni più importanti in questa categoria troviamo:

[2]Il termine pattern si usa per riferirsi ai dati.

•**Clustering** nell'unsupervised learning: individua dei gruppi (cluster) fra i dati, figura (4.5), raggruppando quelli con caratteristiche simili. Trovando così delle classi per un problema che inizialmente non le fornisce.
Non conosco il numero di cluster a priori, ma una volta individuati dall'algoritmo posso usare quelli come output per risolvere il problema.

Figura 4.5. Esempio di Clustering

•**Riduzione di dimensionalità** nell'unsupervised learning: riduce il numero di dimensioni dei pattern in input, senza perdere troppe informazioni, figura (4.6).
Certamente perderò alcune informazioni, ma l'obbiettivo è quello di mantenere le più importanti per il problema.
Risulta molto utile per problemi con dimensioni molto elevate, scartando quelle ridondanti o instabili.

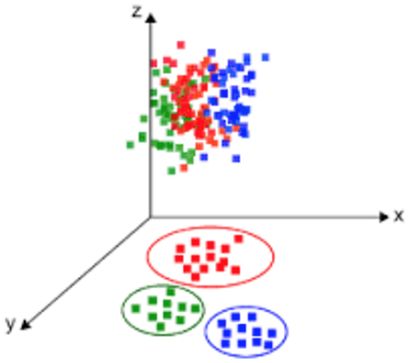


Figura 4.6. Esempio della riduzione di dimensionalità

4.1.3 Semisupervised Learning

Posto a metà fra i due metodi precedenti c'è l'apprendimento parzialmente supervisionato che si basa su dati misti in cui una minima parte è già etichettata, ma una larghissima maggioranza è costituita da dati non etichettati.
Questo approccio viene utilizzato per migliorare le previsioni fatte dalla macchina sui dati non etichettati.

4.1.4 Reinforcement Learning

Il Reinforcement Learning o Apprendimento con rinforzo Ha come obiettivo l'apprendimento di un comportamento ottimale a partire dalle esperienze passate.
Il computer interagisce con un ambiente dinamico nel quale deve raggiungere un determinato obiettivo (figura (4.7)).
Man mano che il computer esplora il dominio del problema, gli vengono restituiti dei feedback in termini di ricompense,nel caso in cui il sistema riesca a raggiungere un obiettivo, o punizioni, nel caso in cui vengano commessi degli errori, in modo da poterlo indirizzare verso la soluzione migliore.
Perciò si punta a realizzare sistemi in grado di apprendere ed adattarsi ai cambiamenti dell'ambiente in cui sono immersi attraverso la distribuzione di una "ricompensa" detta rinforzo, data dalla valutazione delle prestazioni.

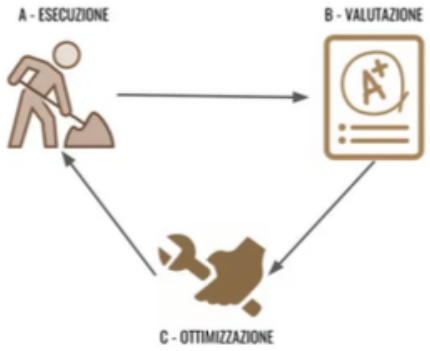
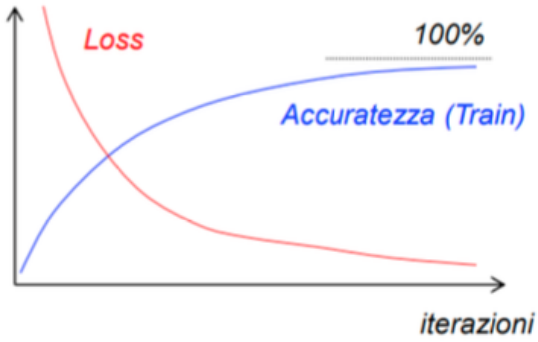


Figura 4.7. Processo base del Reinforcement Learning

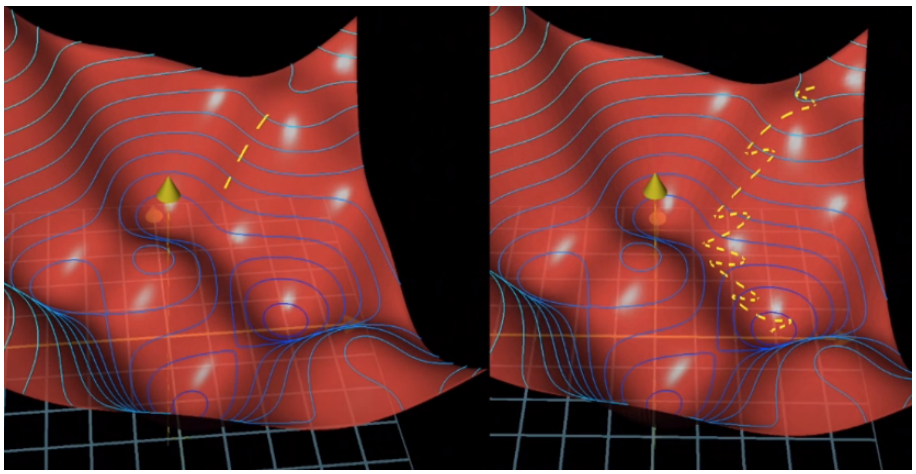
4.2 Apprendimento Machine Learning

Il comportamento dell'algoritmo di Machine Learning è caratterizzato dal valori specifico di un set di parametri che devono essere ottimizzati per rendere più accurato l'apprendimento.
L'allenamento consiste fondamentalmente nel:

1) **Accuratezza** della soluzione da massimizzare.
2) **Errore** o perdita (loss-function) da minimizzare.



Quando svolgo il Train test (come vedremo in seguito) voglio che l'errore (*Loss Funcion*) si minimizzi e quindi voglio che il gradiente della funzione diminuisca (da qui GBRT).
Come vediamo in figura a seconda della scelta degli iperparametri andremo verso punti di minimo più o meno velocemente.



Prendendo in considerazione l'iperparametro «passo» vedo nella figura (4.2) come con un

passo troppo piccolo ci metto molto tempo per arrivare al minimo della funzione e posso anche fermarmi in altri punti di minimo locale con derivata uguale 0. Mentre con un passo grande rischio di oltrepassare la zona di minimo e trovarmi direttamente su un altro picco.

Dunque prima dell'apprendimento bisogna definire questi parametri definiti "*iperparametri*" che sono appunto dei parametri che si ottimizzano durante l'allenamento e che permettono di ridurre il gradiente della loss function riducendo l'errore e migliorando l'accuratezza della soluzione. Si sceglieranno i valori che producono prestazioni migliori.
Una volta scelti si esegue l'apprendimento per ciascuno, andando a capire quali e come impattano di più e forniscono una prestazione migliore.

4.2.1 Ottimizzazione iperparametri

Come vedremo in seguito sarà fondamentale ottimizzare gli iperparametri per permettere di ridurre l'rmse. Per l'ottimizzazione degli iperparametri si deve impostare il valore dei parametri che l'algoritmo non può apprendere da solo.
Il problema è che non si conoscono i valori da dover impostare, pertanto, il processo comunemente utilizzato è fare in modo che l'algoritmo utilizzi diverse combinazioni di valori finché non trova i valori migliori per il modello.
Detto questo, ci sono diversi iperparametri che possiamo regolare nel codice e alcuni esempi possono essere:
-Number of estimators (Numero di stimatori)
-Learning rate (tasso di apprendimento)
-Subsample (sottocampione)
-Max depht (profondità massima)
Bisogna creare un'istanza di *GradientBoostingRegressor*. Successivamente, creeremo la nostra griglia con i vari valori per gli iperparametri. Prenderemo quindi questa griglia e la posizioneremo all'interno della funzione GridSearchCV (che è una variabile di tipo dizionario) in modo che possiamo prepararci a eseguire il nostro modello.
All'interno di questa funzione verranno ottimizzati i valori degli iperparametri scelti e come vedremo in seguito sfrutteremo anche il concetto della Cross-validation.

4.2.2 Valutazione delle prestazioni

La valutazione delle prestazioni di un algoritmo possono essere quantificate direttamente tramite la funzione obiettivo.

Solitamente però per dare un volto alle prestazioni di questi algoritmi si usano parametri matematico-statistici utilizzati nel campo della statistica e della probabilità[10].
Ci sono delle distinzioni in base al tipo di sistema, cioè se affronto un problema discreto o uno continuo.

Discreto solitamente un problama di classificazione, l'accuratezza che va da 0% a 100% è la percentuale di pattern correttamente classificati, mentre l'errore è il complementare.

$$Accuratezza = \frac{pattern\ correttamente\ classificati}{pattern\ classificati} \tag{4.1}$$

$$Errore = 100\% - Accuratezza \tag{4.2}$$

Continuo (problema di regressione), viene valutato il rmse (root mean square error) cioè la radice della media dei quadrati degli scostamenti tra il valore vero e il valore predetto. Questo parametro fornisce un'idea di quanto il sistema, con la sua predizione, si allontana dalla realtà dei dati ed è espresso nell'equazione (4.3).

$$rmse = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(pred_i - true_i)^2} \tag{4.3}$$

L'rmse è preferibile come misura delle prestazioni in un algoritmo di regressione, ma talvolta si può vedere usata anche un'altra funzione chiamata mean absolute error (mae) espressa nell'equazione (4.4):

$$mae = \frac{1}{N}\sum_{i=1}^{N}|pred_i - true_i| \tag{4.4}$$

Anche se meno usato, sia rmse che mae sono entrambi delle strade per calcolare la distanza tra due vettori, quello delle predizioni e quello dei valori di target[3].

4.2.3 Training, Validation e Test

Per capire come il modello si adatta ai nuovi dati è necessario provarlo sulle istanze stesse. Il Dataset che viene fornito all'algoritmo supervisionato (input e output) viene suddiviso in due parti: Training e Testing.
Inoltre il primo si divide in training set e validation set, figura (4.8).
Training:
- *Training set*: insieme di pattern tramite il quale si allena l'algoritmo trovando il valore ottimale dei parametri
- *Validation set*: insieme di pattern su cui il sistema tara gli iperparametri.

Testing:
Insieme di pattern (in cui gli output sono incogniti) su cui l'algoritmo valuta le prestazioni finali.

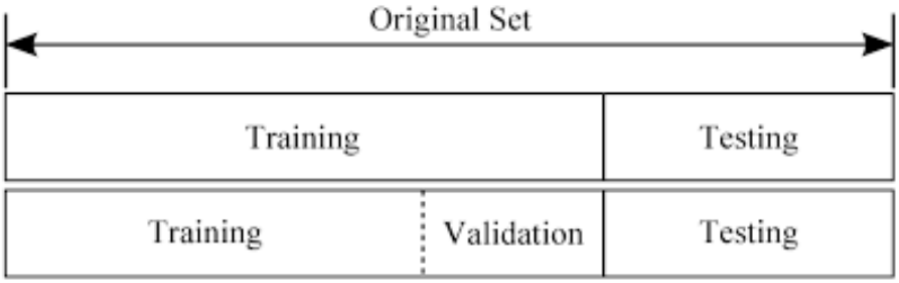


Figura 4.8. Schema di suddivisione del Dataset completo fornito all'algoritmo

Per essere sicuri che l'algoritmo non sia influenzato dalla suddivisione dei dati nei dataset di training, validation e testing, si utilizza la *cross-validation*.

[3]Per "$true_i$" si intende l'i-esimo valore reale (misurato) e per "$pred_i$" l'i-esimo valore predetto.

4.2.4 Cross-validation

L'insieme dei dati di training viene suddiviso in sottoinsiemi complementari, dopo di che il modello è allenato su una parte di questi e validato sulla rimanente. Tale operazione è ripetuta per diverse combinazioni dei sottoinsiemi. Una volta definita quella che restituisce le prestazioni migliori e selezionati gli iperparametri, il modello finale viene allenato sul training set completo e successivamente applicato al test set per fornire l'errore generalizzato[11].
Grazie alla cross validation il diagramma di flusso delle operazioni svolte normalmente viene leggermente modificato, figara (4.9), andando a trovare in un primo momento i valori ottimi degli iperparamatri che verranno successivamente usati nell'algoritmo.

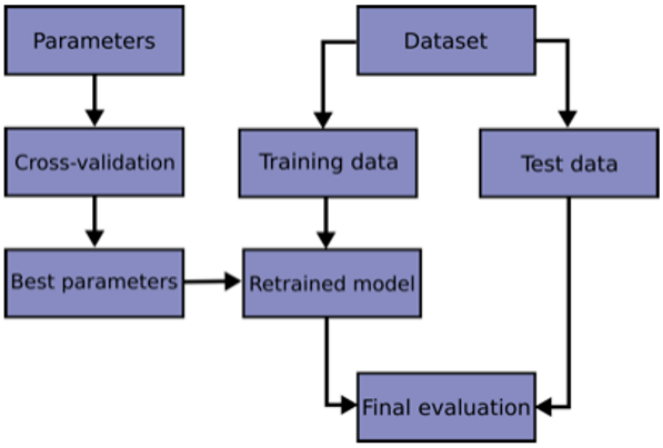


Figura 4.9. Diagramma di flusso con l'implementazione della CV

Suddivido il training set in k parti di uguale dimensione, figura (4.10). In genere, si divide in 5 o 10 parti.
Poi seleziono una parte 1/k per utilizzarla come validation set. Le restanti parti k-1/k invece continuano a comporre il training dataset.

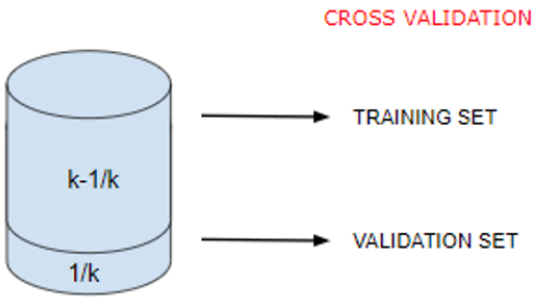


Figura 4.10. Suddivisione k-fold

Poi avvio il processo di training sul dataset residuo (k-1/k).
Ripeto la stessa procedura per k volte, selezionando ogni volta un sottoinsieme diverso come validation set. Nella seguente figura (4.11) il dataset è diviso in 5 fold: 4 per il training e 1 per il validation. Alleno su 4 e testo su 1 e riporto i risultati nei vettori CV test score e CV train score. CV test score contiene l'accuratezza sul validation dataset, CV train score l'accuratezza sul train dataset.

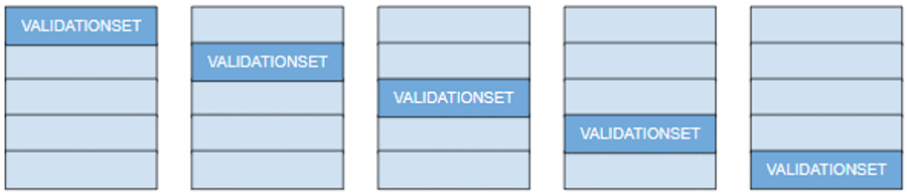


Figura 4.11. Suddivisione per le diverse iterazioni

4.2.5 Overfitting e Underfitting del Training Data

Come detto in precedenza uno dei primi obiettivi prefissati durante l'addestramento è la convergenza sul train set[9].
Se considerassimo un elaborazione in cui l'addestramento prevede un processo iterativo, si ottiene convergenza quando:
-La loss function ha andamento decrescente rispetto al numero di iterazioni; -L'accuratezza ha andamento parabolico crescente rispetto al numero di iterazioni. Nella figura (4.2) si vedono indicativamente gli andamenti qualitatitivi delle due curve per un modello ben costruito.
Se gli andamenti dovessero discostarsi da quelli in figura potremmo essere incappati in

errori computazionali. I due errori più frequenti e importanti che potrebbero verificarsi durante l'addestramento del training data sono *Overfitting e Underfitting*.
Nel seguente diagramma (figura (4.12)) possiamo semplificare il ragionamento per cui si incorre nei seguenti errori:

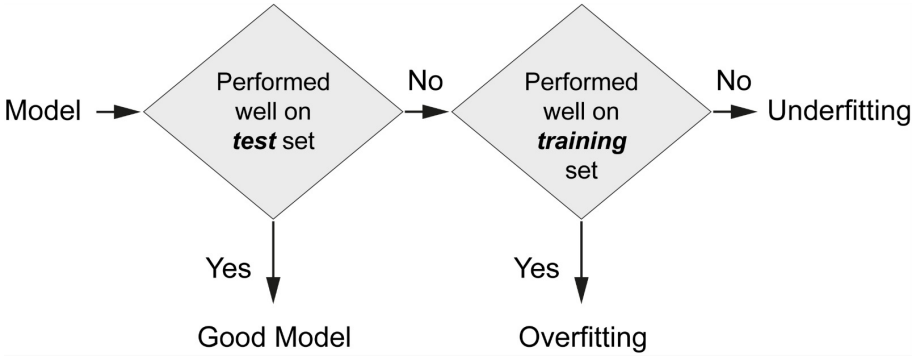


Figura 4.12. Suddivisione per le diverse iterazioni

Nel caso in cui l'apprendimento viene effettuato per troppo tempo o nel coso in cui avevo uno scarso numero di esempi di allenamento, il modello potrebbe adattarsi a caratteristiche che sono specifiche solo del training set, ma che non hanno riscontro nel resto delle casistiche.
Ciò significherebbe non avere una buona generalizzazione del modello, cioè non avere una buona capacità di trasferire l'elevata accuratezza raggiunta sul train alla fase di validazione.
Di seguito in figura (4.13) vengono mostrati gli andamenti della funzione obiettivo nel caso in cui si incorra in errore.

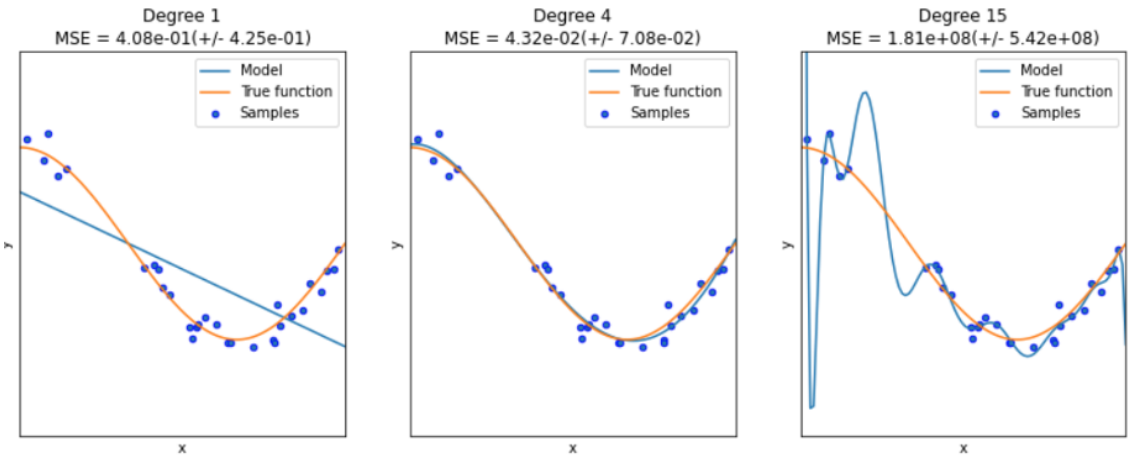


Figura 4.13. Suddivisione per le diverse iterazioni

Overfitting: quando il modello ha un eccessivo numero di gradi di libertà, raggiunge un alta accuratezza sul train ma non sulla validation (scarsa generalizzazione).
Solitamente si parte con pochi gradi di libertà, controllabili attraverso gli iperparametri, andando poi via via ad aumentarli monitorandone l'accuratezza.
Underfitting: è l'opposto del caso precedente, si verifica quando il modello è troppo semplice per apprendere dalla struttura dei dati forniti.

Capitolo 5

Gradient Boosting Regression Tree

In questo capitolo si vuole mostrare una panoramica di come è strutturato e sviluppato un modello semi-empirico con approccio DoE (Design of Experiment), che combina alcuni parametri riguardanti la fisica e la chimica del problema identificati come variabili di input che rappresentano i parametri motoristici più influenti sulla formazione e ossidazione di PM e di NO_X.
L'obiettivo del seguente lavoro è quello creare e ottimizzare un modello di apprendimento automatico Machine Learning che sia svincolato totalmente dall'aspetto fenomenologico della formazione degli inquinanti e sia in grado di sottolineare i parametri motoristici più influenti sulla formazione di soot e degli ossidi di azoto, tramite datasets di parametri appartenenti a metriche diverse: Engine Control Unit (ECU) e test bench (banco prova).
L'algoritmo **Gradient Boosting Regression Tree** (GBRT) è stato implementato in ambiente Python tramite la libreria open-source "Scikit-Learn".
A seguito di un Training, l'algoritmo si propone di predire nella fase di Testing nuovi valori obiettivo a partire da dati mai visti prima. Si tratta di un algoritmo di regressione supervisionato.
Al completamento delle predizioni, il codice restituisce su di un foglio di calcolo Excel il valore dell'accuratezza, rmse, sia per il dataset completo che per le features importance oltre che al valore che gli iperparametri assumono per quel determinato caso studio.
Oltre questi dati numerici restituisce quattro grafici che descrivono precisamente l'andamento delle predizioni confrontando con i dati presenti nel dataset. Affrenteremo nel sotto capitolo (5.3.1) i grafici utilizzati analizzandone le caratteristiche.

Prima di passare alla descrizione della teoria delle decisioni associata al GBRT è opportuno dare un'idea ben chiara dei modelli utilizzati per arrivare alla generazione dell'algoritmo Gradient Boosting Regression Tree.

Questo perchè il GBRT è costituito da un insieme di algoritmi più semplici, *Decision Trees* e *AdaBoost* che possono svolgere compiti sia di classificazione che di regressione.
All'interno della libreria open source Scikit-Learn sono presenti questi algoritmi oltre che per le librerie utilizzate all'interno del codice per applicare le funzioni. Le librerie che contengono la maggior parte delle operazioni effettuate nel codice sono SciPy e NumPy. Quindi per poter analizzare i risultati predetti per i vari casi bisogna prima capire il funzionamento dell'algoritmo.

5.1 Decision Tree

Il Decision Tree è una delle più comuni tecniche di classificazione e regressione utilizzate. Nella teoria delle decisioni, un albero di decisione è un grafo delle decisioni e delle loro possibili conseguenze, utilizzato per creare un 'piano di azioni' mirato ad uno scopo. Un albero di decisione è costruito al fine di supportare l'azione decisionale.
Nella figura (5.1) è mostrata la composizione a blocchi del Decision Tree.

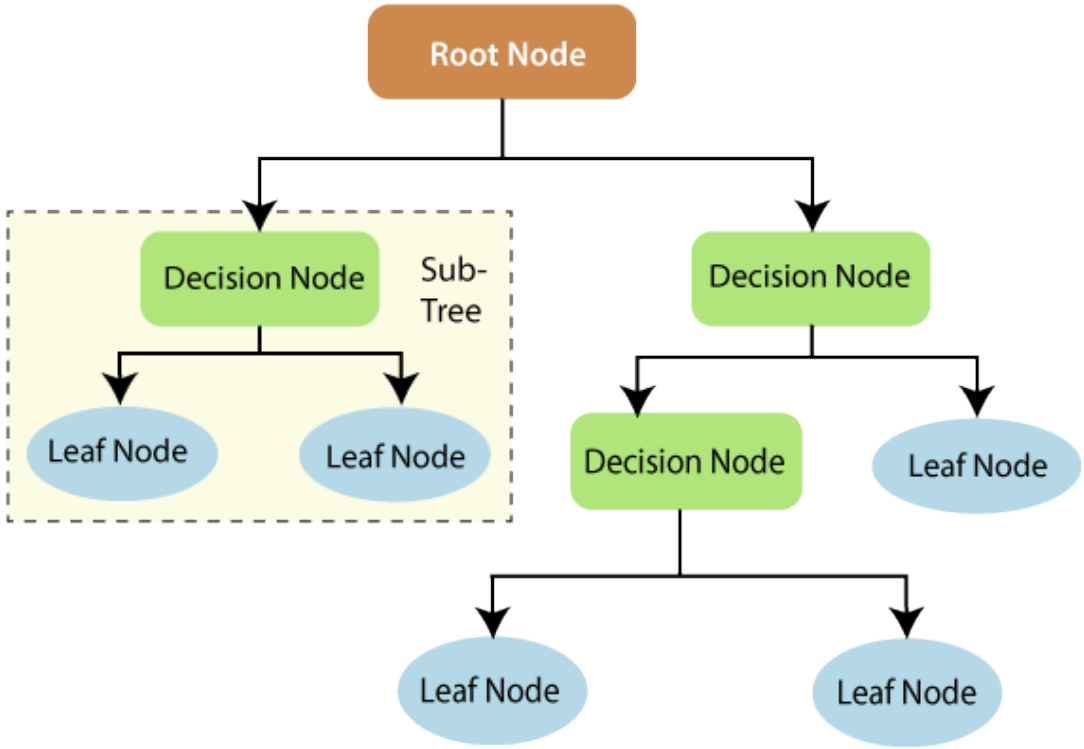


Figura 5.1. Decision Tree

Nel Machine Learning un albero di decisione è un modello predittivo, dove ogni nodo interno rappresenta una variabile, un arco verso un nodo figlio rappresenta un possibile valore per quella proprietà e una foglia il valore predetto per la variabile obiettivo a partire dai valori delle altre proprietà, che nell'albero è rappresentato dal cammino (path) dal nodo radice (root) al nodo foglia(leaf).
-Nodo: rappresenta una variabile come feature (o attributo);
-Ramo o cammino verso il nodo figlio: rappresenta un possibile valore per la feature;
-Foglia: valore predetto per la variabile target.
Questo algoritmo può prendere un set di dati non familiari ed estrarne una serie di regole tramite le quali rendere capace la macchina di comprendere il problema e i risultati, ed presenta i seguenti vantaggi e svantaggi:
Vantaggi:
Basso tempo computazionale
Facile comprensione dei risultati appresi
Permette di trattare features non rilevanti senza presentare problemi
Svantaggi:

Soggetto all'Overfitting

Per la costruzione dell'albero decisionale si parte fornendo all'algoritmo il dataset di punti contenenti tutti i valori associati ad ogni attributo (feature), e i valori di target.
I dati vengono suddivisi ogni volta nei nodi secondo i valori assunti dalle variabili features. Per determinare ciò, si prova ogni feature e si misura quale split (suddivisione) fornisce i risultati migliori.
Dopo di questo, viene diviso il dataset in sottoinsiemi, i quali sono fatti passare attraverso i primi rami del primo nodo decisionale: se i dati analizzati rispettano la stessa condizione imposta alla feature viene raggiunto il nodo foglia, altrimenti si prosegue verso un altro nodo decisionale in cui vengono divisi secondo la condizione nel nuovo nodo. Ogni nodo foglia finale definisce una certa regione, entro le quali saranno poi valutate le nuove variabili obiettivo.
Il criterio di split (suddivisione) con cui crea un nuovo nodo si basa sul massimo guadagno di informazione (info gain). In pratica sceglie l'attributo che riesce a dividere "meglio" le istanze appartenenti a classi diverse (detto anche criterio di massima riduzione di incertezza). Quando tutti gli elementi in un nodo hanno la medesima classe, l'algoritmo non procede con ulteriore split (criterio di stopping). Per evitare overfitting, l'algoritmo inizia una eventuale fase di pruning (potatura): individua gli attributi che non hanno contribuito ad una consistente suddivisione delle istanze ed elimina i rispettivi nodi riunendo le istanze al livello superiore. Quando l'algoritmo termina, è possibile percorrere l'albero dalla radice e, seguendo il percorso risultante dai singoli test presenti su ogni nodo interno, si ottiene la classificazione dell'istanza (nodo foglia).

In questo esempio (5.2) gli alberi decisionali vengono utilizzati per adattare una curva sinusoidale di un'osservazione rumorosa. Di conseguenza, apprende le regressioni lineari locali che si avvicinano alla curva sinusoidale.
Possiamo vedere che se la profondità massima dell'albero (controllata dal parametro max_depth) è impostata troppo alta, gli alberi decisionali apprendono dettagli troppo fini dei dati di allenamento e assecondano troppo il rumore, cioè si adattano eccessivamente.

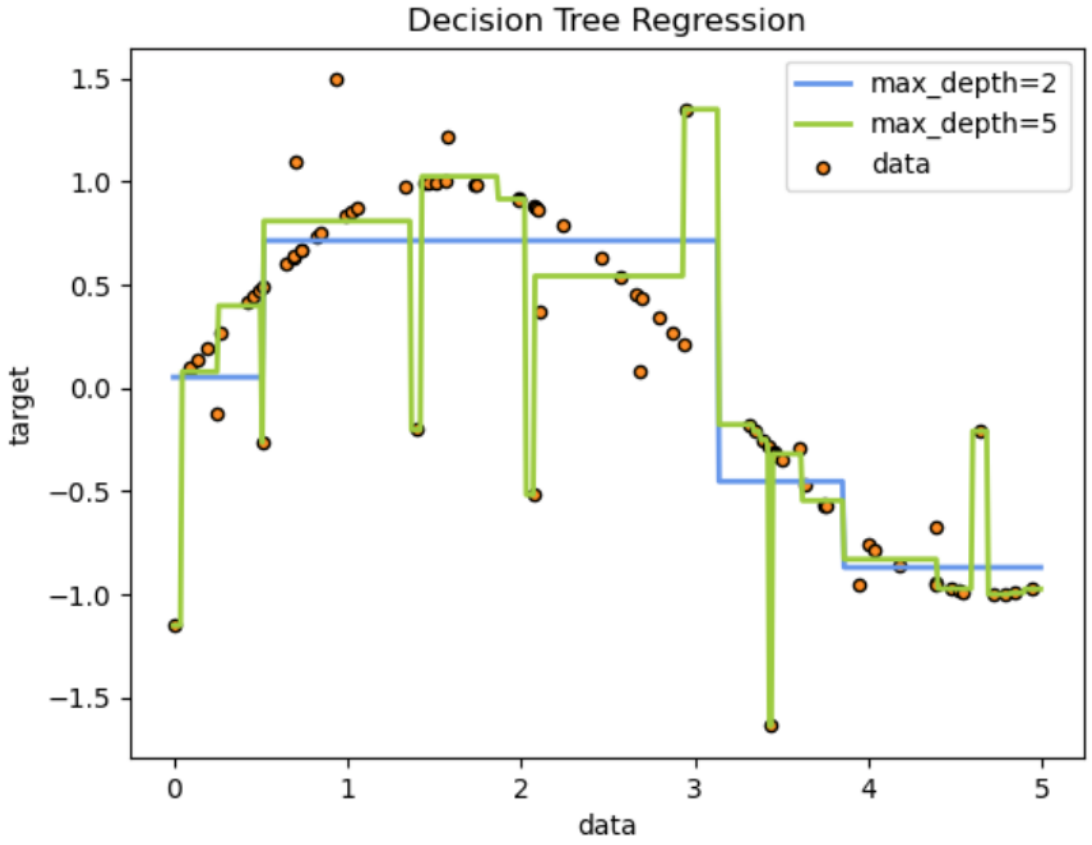


Figura 5.2. Decision Tree di regressione

Da quest'esempio possiamo notare l'importanza degli iperparametri.
Per evitare questo problema di overfitting è necessario limitare la libertà del Decision Tree durante l'allenamento. Tale processo è chiamato regolarizzazione degli iperparametri.
I principali iperparametri che controllano la crescita della struttura e della forma del Decision Tree sono:
max_depth: indica la massima profondità del Decision Tree, definita in livelli;
min_samples_split: numero minimo di istanze che un nodo deve avere prima di essere diviso;
min_samples_leaf: numero minimo di istanze che un nodo foglia deve avere;
min_weight_fraction_leaf: è equivalente al precedente ma espresso come frazione sul numero totale di istanze pesate;
max_leaf_nodes: massimo numero di nodi foglia;
max_features: massimo numero di features valutate per dividere ogni nodo.
Incrementando quelli che esprimono un valore minimo o diminuendo quelli che esprimono

un valore massimo si può regolarizzare il modello.
I parametri più largamente usati per le condizioni di split sono:
-*Indice di Gini* (Gini index): utilizzato da CART (Classification and Regression Trees). L'indice di Gini raggiunge il suo minimo (zero) quando il nodo appartiene ad una singola categoria. In entrambe le formule f rappresenta la frequenza del valore j nel nodo i.

$$I_G(i) = 1 - \sum_{j=1}^{m} f(i,j)^2$$

Figura 5.3. Indice di Gini

-*Variazione di Entropia* (entropy deviance): è basata sul concetto di entropia

$$I_E(i) = -\sum_{j=1}^{m} f(i,j) \log f(i,j)$$

Figura 5.4. Variazione di Entropia

L'indice di Gini e la variazione di entropia sono i parametri che vengono usualmente utilizzati per guidare la costruzione dell'albero decisionale.
Poiché, in generale, in un buon albero di decisione i nodi foglia dovrebbero essere il più possibile puri (ovvero contenere solo istanze di dati che appartengono ad una sola classe), un'ottimizzazione dell'albero consiste nel cercare di minimizzare il livello di entropia man mano che si scende dalla radice verso le foglie. In tal senso, la valutazione dell'entropia determina quali sono, fra le varie scelte a disposizione, le condizioni di split ottimali per l'albero.

Come vedremo a breve il GBRT arriva alla predizione mediando diversi tipologie di decision tree associati al concetto di AdaBoost.

5.2 AdaBoost

GBRT si avvicina molto all'AdaBoost ed è fondamentale dare una panoramica del funzionamento.

Diversamente da altri algoritmi, come il Random Forest, l'AdaBoost solitamente usa alberi composti generalmente da un nodo e due foglie come mostato in figura (5.5).

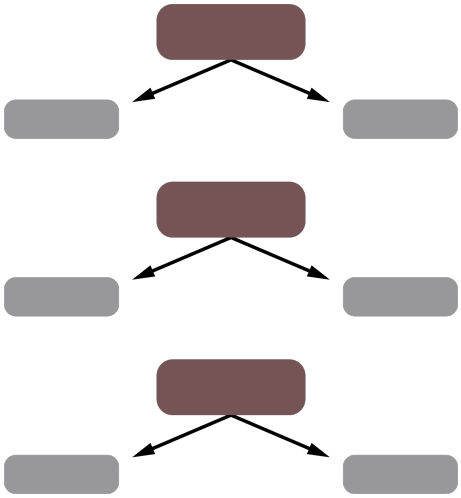


Figura 5.5. Stumps

Alcuni degli Stumps in Adaboost possono dirci di più rispetto ad altri riguardo l'obiettivo finale. Nella figura dimostrativa (5.6) si evidenzia come alcuni possono essere più grandi di altri e perciò valere diversamente.

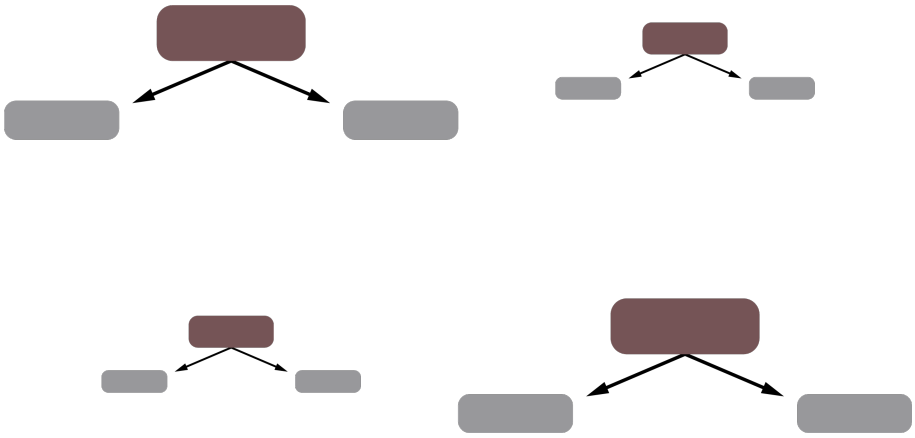


Figura 5.6. Differenti taglie per gli Stumps

Come si evince in questo esempio i ceppi più grandi sono più influenti di quelli piccoli. A differenza del RF negli AdaBoost l'ordine è importante, perchè l'errore che produce il

primo stump influenza il secondo, e così via.
Infatti il terzo punto che caratterizza l'AdaBoost è proprio la concatenazione fra di essi.
Ogni stump è creato partendo dall'errore individuato nello stump precedente proprio come intuitivamente mostrato nella figura (5.7).

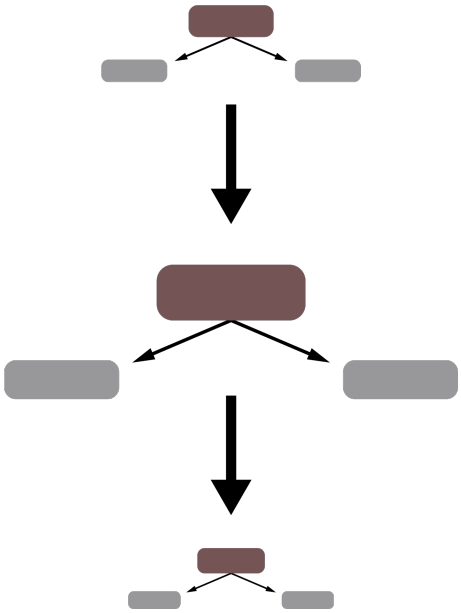


Figura 5.7. Ogni stump risente dell'errore del precedente

5.3 Descrizione algoritmo GBRT

L'algoritmo GBRT costruisce un albero partendo dai dati normalizzati.
Come l'AdaBoost, l'albero del GBRT è basato sull'errore creato dall'albero precedente, ma a differenza dell'AdaBoost questo albero è generalmente più grande dello stump.
In questo semplice esempio (5.8) viene costruito un albero con 4 foglie, ma solitamente si setta il valore dell'iperparametro per in numero di foglie compreso tra 8 e 32.

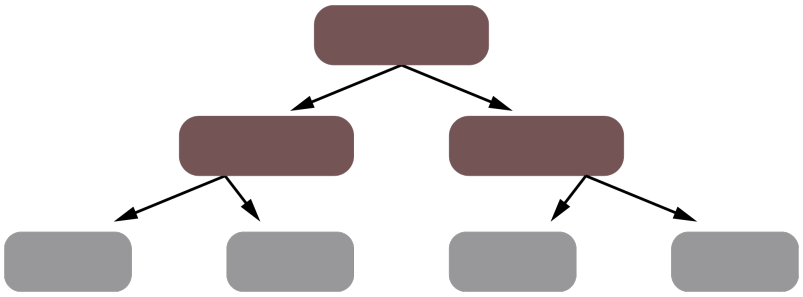


Figura 5.8. Albero Gradient Boosting Regression Tree

Così il GBRT fissa l'albero successivo tramite l'errore del precedente, ma diversamente dallo stump ogni albero è più largo e profondo.
Un'altra similitudine dall'AdaBoost è che è possibile scalare l'albero, ma tutto l'albero viene diminuito o aumentato dello stesso quantitativo.

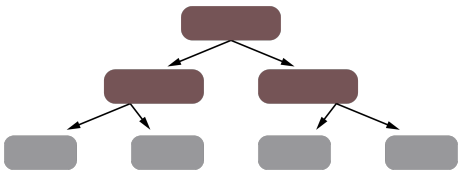


Figura 5.9. Albero Gradient Boosting Regression Tree ridotto

Come nell'AdaBoost, il GBRT successivamente costruisce un albero basato sull'errore e le dimensione del precedente errore, così costruisce il successivo riducendo la scala.
Il GBRT continua a costruire alberi in questa maniera fino a che non si arriva al numero di alberi richiesti tramite il valore dell'iperparametro associato.
Ora conosciamo le maggiori differenze e similitudini fra il GBRT e l'AdaBoost, il quale viene utilizzato per la sua formazione.

Riassumendo, quando il Gradient Boost è usato per la regressione:
-Si inizia con una foglia con il valore medio di ciò che vogliamo predire;
-Dopo si somma un albero basato sul RESIDUALS cioè la differenza tra il VALORE OSSERVATO e il VALORE PREDETTO
-Questa somma viene scalata con il LEARNING RATE
-Successivamente si somma un altro albero in base al nuovo RESIDUO, e così si continua ad aggiungere alberi basati sugli errori fatti precedentemente.
Questo concetto di come si costruisce il GBRT è mostrato nella figura (5.10) tramite l'uso

semplificato di blocchi.

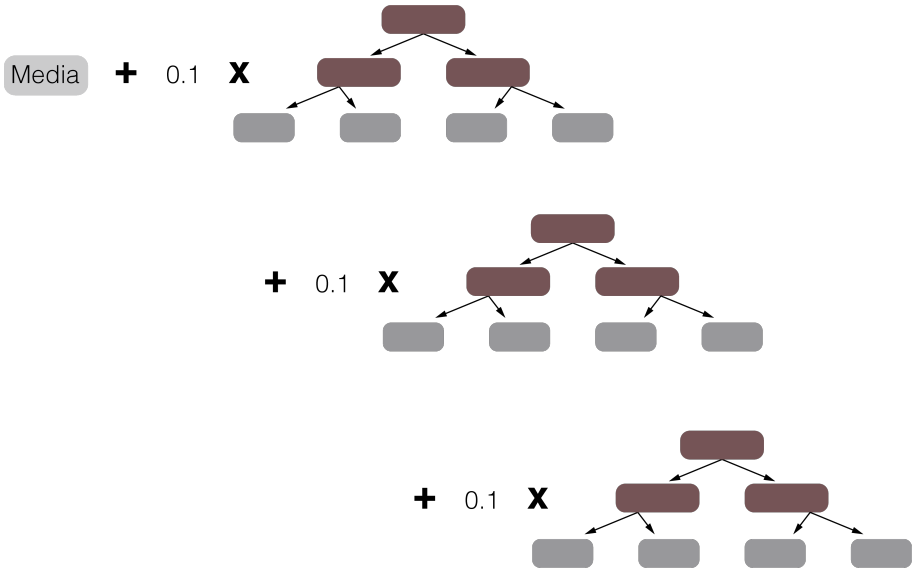


Figura 5.10. Gradient Boosting Regression Tree per la regressione

A livello matematico l'algoritmo GBRT segue determinati step per poter arrivare al valore predetto desiderato.
Gli step da seguire sono presentati nella figura (5.11) dove vengono mostrate anche le formulazioni utilizzate dall'algoritmo.

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:
   1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

   2. Fit a base learner (or weak learner, e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

Figura 5.11. Gradient Boosting Regression Tree steps matematici

Nell'algoritmo GBRT utilizzato per la predizione di inquinanti sono stati scelti i seguenti iperparametri da poter sottoporre a tuning per arrivare ad un valore massimizzato dell'accuratezza in quel caso specifico.
Questi iperparametri però nel suddetto codice sono stati inizialmente ottimizzati con due strategie diverse, GridSearch e BayesSearch, in modo da capire se questa differenza potesse portare a risultati diversi e quindi valori di r^2 considerevolmente diversi.
Nei prossimi capitoli di analisi dei risultati verranno dimostrate queste differenze, ma in questo capitolo ci si sofferma a presentare i procedimenti di analisi con i due diversi metodi per capirne le differenze.

Nel caso del **GridSearchCV** il procedimento è diretto ed intuitivo ed è composto dai seguenti passi:

1)Si sceglie il modello ML che si vuole usare: Gradient Boosting Regressor Tree

2) Si scelgono in modo arbitrario gli iperparametri del modello che si vogliono sottoporre al tuning, mostrati in figura (5.12). Questi iperparametri sono gli stessi che vengono utilizzato nei primi due casi studio, nel terzo, come vedremo, si andrà ad aggiungerne altri per rendere più solide le trasformazioni utilizzate.

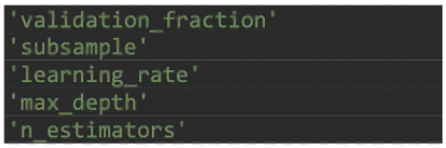


Figura 5.12. Iperparametri scelti per il GridSearchCV

3) Si assegna ad ogni iperparametro un set di valori casuali e nel range di attinenza[1] che si vogliono provare, figura (5.13)

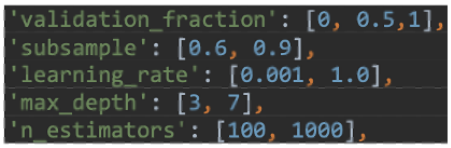


Figura 5.13. Valore degli iperparametri scelti per il GridSearchCV

4) Si allena il modello su tutte le combinazioni senza ripetizione possibili con la serie di parametri definiti. Nel nostro caso quindi si proveranno 3x2x2x2x2=48 combinazioni di iperparametri che rappresentano tutte le combinazioni dei valori da provare precedentemente assegnati al punto 3, il tutto moltiplicato per k-fold, nel nostro caso 5 per un totale di 240 casistiche.

5) Si sceglie il set di iperparametri che hanno prodotto il migliore risultato.

Si noti infatti che viene utilizzato il GridSearchCV ed il CV finale sta per Cross-Validation. Questo è un metodo di training che nasce per permettere di avere un'idea generalizzata di quanto il modello possa essere performante una volta allenato: il set di dati di allenamento vengono divisi in n parti chiamate "fold", nel nostro caso per esempio si è deciso di dividerlo in 5.
Si allena quindi il modello a ruota su 4/5 dei dati e si testa sul 1/5 e si calcola il valore medio della metrica di valutazione per tutti i fold.

L'altra metodologia utilizzata per l'ottimizzazione dei valori degli iperparametri scelti è la BayesSearchCV. Si precisa che in entrambi i casi sono stati scelti gli stessi iperparametri

[1] In questo caso validation fraction verrà provato con i valori 0, 0.5 ed 1 mentre subsample verrà provato con 0.6 ed 0.9

per poter confrontare i due metodi direttamente. Questo confronto avverrà nel capitolo successivo.
Usando il **BayesSearchCV** in pratica implementiamo l'ottimizzazione *Bayesiana* la quale sfrutta il teorema di Bayes per ricercare gli iperparametri ottimali.
Il procedimento in questo caso consta dei seguenti passi:

1) Si sceglie il modello ML che si vuole usare: Gradient Boosting Regressor Tree

2) Si scelgono in modo arbitrario gli iperparametri del modello che si vogliono sottoporre al tuning (fig. (5.14))

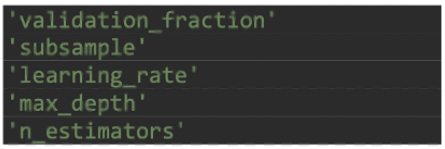


Figura 5.14. Iperparametri scelti per il BayesSearchCV

3) Si assegna ad ogni iperparametro un set di valori casuali e nel range di attinenza che si vogliono provare come nell'immagine (5.15).

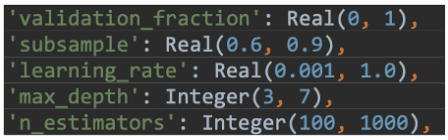


Figura 5.15. Valore degli iperparametri scelti per il BayesSearchCV

4) Si fissa il numero di iterazioni in modo arbitrario ed ad ogni iterazione ogni parametro viene scelto tramite il teorema di Bayes in modo da ottimizzare la funzione obiettivo.

5) Si sceglie il set di iperparametri che hanno prodotto il migliore risultato.
Per la scelta del numero di iterazioni risulta che essa è arbitraria. Tuttavia un numero troppo elevato causa overfitting mentre un numero troppo basso non permette di trovare il valore ricercato. Anche qui si procede per esperienza agendo sul range dei parametri e sul numero di iterazioni a seconda dei casi.
Di solito però se si ottiene che alla fine il r^2 sul test set è basso allora si prova ad alzare il

numero delle iterazioni e viceversa.

Il Teorema di Bayes o teorema della probabilità delle cause sul quale si basa questa metodologia deriva da due teoremi fondamentali delle probabilità: il Th. della probabilità composta e il Th. della probabilità assoluta.
In statistica e teoria delle probabilità, il teorema di Bayes descrive la probabilità di un evento, in base alla conoscenza preliminare delle condizioni che potrebbero essere correlate all'evento. Serve come un modo per capire la probabilità condizionata.
Essenzialmente, il teorema consente di aggiornare un'ipotesi ogni volta che viene introdotta una nuova evidenza.
L'equazione che la rappresenta è la seguente (5.1):

$$P(A|B) = \frac{P(B|A)\ P(A)}{P(B)} \tag{5.1}$$

Dove:
P è il simbolo per indicare la probabilità.
$P(A|B)$ è la probabilità dell'evento A (ipotesi) verificatasi dato che B (evidenza) si è verificata. Essa viene definita anche come probabilità posteriore.
$P(B|A)$ è la probabilità che l'evento B (evidenza) si verifichi dato che A (ipotesi) si è verificata.
$P(B)$ è la probabilità dell'evento B (ipotesi) che si verifichi.
$P(A)$ è la probabilità dell'evento A (evidenza) che si verifichi. Essa viene anche chiamata probabilità precedente perché è la conoscenza che abbiamo del valore di A prima di guardare le osservabili B.

5.3.1 Grafici utilizzati

Per capire pienamente gli studi successivi è opportuno identificare i grafici utilizzati per poterne capire in seguito il significato e giustificare i valori numerici predetti dal codice GBRT.
Questi grafici, come vedremo, sono implementati nel codice GBRT direttamente nell'ambiente di compilazione Python e corrispondono a precise righe di codice che verranno mostrate in questo capitolo.

Grafico - Prediction vs Test

Questo grafico, mostrato in figura (5.16), mette in x i valori *y_test* ed in y i valori *y_predicted* che sarebbero rispettivamente i valori originali e i valori predetti dal codice e poi traccia una linea rossa che congiunge i punti [0,0][1,1].

In sostanza mostra come si spargono i valori attorno ad una relazione perfettamente lineare, che coinciderebbe con l'accuratezza del 100% se contenesse tutti i punti predetti. Perciò se il risultato *y_predicted* fosse stato uguale a *y_test* (accuratezza 100%) i valori in blu sarebbero stati tutti sulla retta perché in ogni punto *y_test* sarebbe stato uguale ad *y_predicted*.

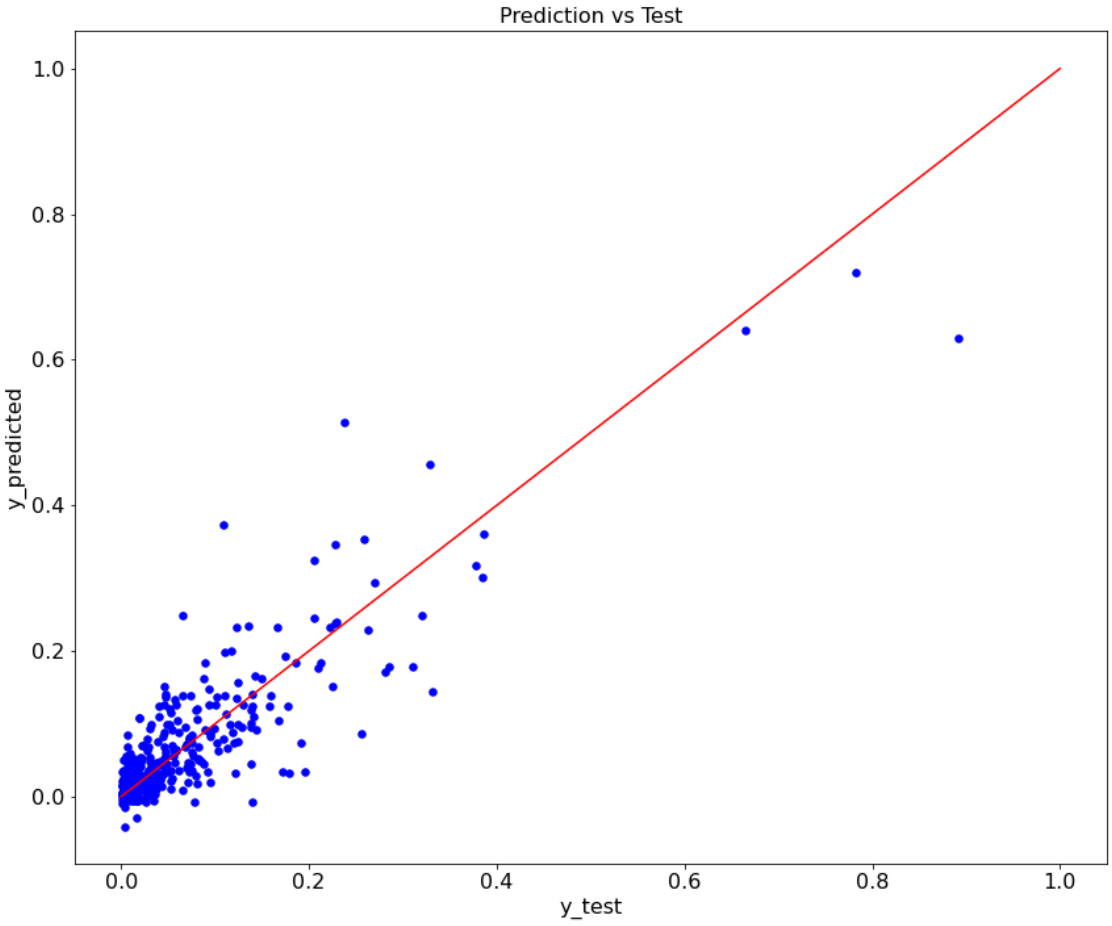


Figura 5.16. Esempio grafico Prediction vs Test

Quindi vorrei accuratezza dei punti blu sulla curva a 45 ed allo stesso tempo avere il min errore possibile della funzione che approssima i punti nello spazio senza però incorrere

in overfitting. Questo va assolutamente evitato perché il modello che ti ritrovi alla fine funziona molto bene solo per i dati sui quali lo si è allenato allenato.
Non appena gli si danno nuovi dati poco diversi da quelli di allenamento non è in grado di predire con accuratezza.

Il codice implementato in ambiente Python per il seguente grafico è quello mostrato nelle righe di codici presenti in figura (5.17):

```
plt.figure(figsize=(12,10))
col=['blue']
plt.scatter(y_test, y_predicted, s=30,
            c=col, edgecolors=col, marker='o')
plt.gcf()
plt.title('Prediction vs Test')
plt.xlabel('y_test')
plt.ylabel('y_predicted')
plt.gcf()
plt.plot([0,1],[0,1], c='r')
plt.savefig(folder_path+"/y_pred vs y_test.png")
plt.show()
```

Figura 5.17. Codice in ambiente Python per "Prediction vs Test"

Grafico - All points of prediction and Test
In questo caso sono riportati in uno stesso grafico (5.18) i valori delle predizioni e i corrispondenti valori originali.
Supponiamo che il target sia l'emissione di NO_X allora in arancio si hanno i valori di emissione predetti dal modello mentre in blu i valori veri del dataset.
Nel grafico sull'asse x ci sono i punti che sembra siano 250 e per ognuno di questi si ha sia il valore predetto (arancio) che quello originale.

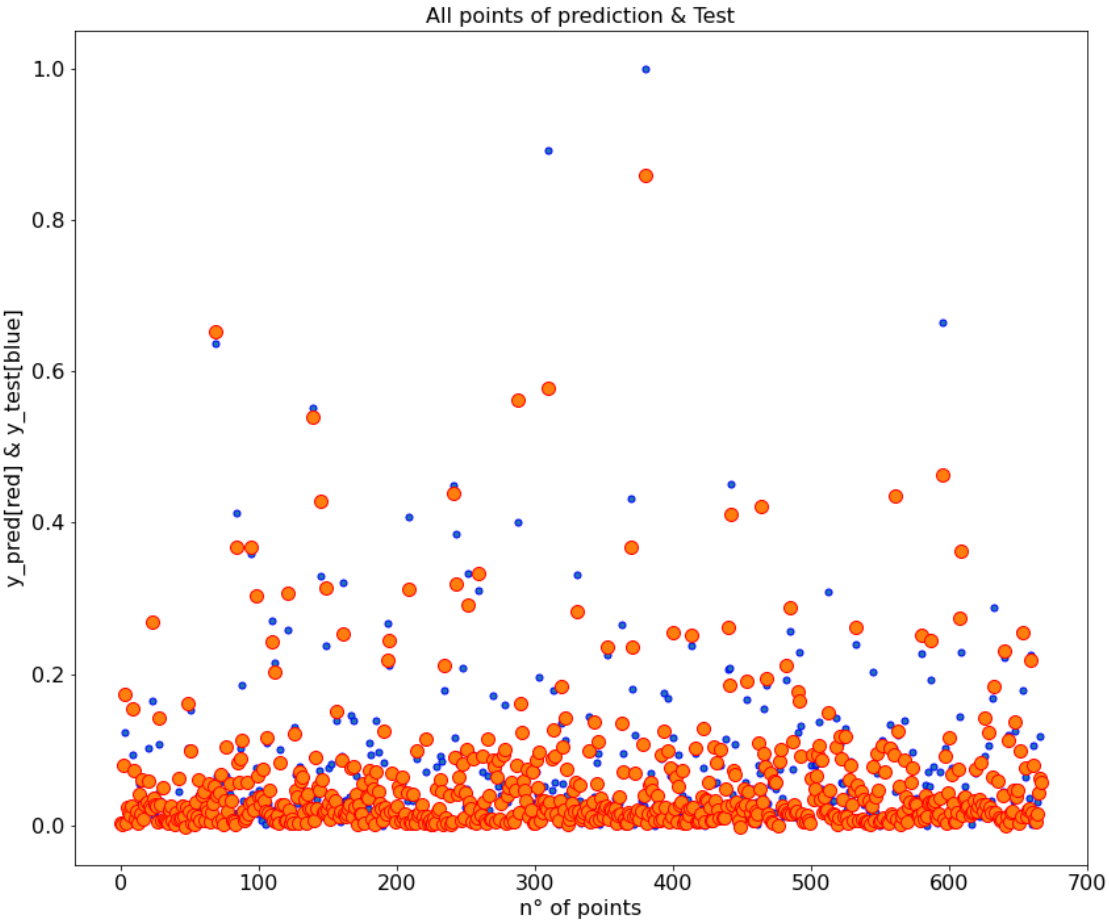


Figura 5.18. Esempio grafico All points of prediction and Test

Nel caso i valori predetti fossero esattamente uguali a quelli originali del test i punti per ogni grafico sarebbero completamente sovrapposti.
Viene mostrato anche in questo caso la parte di codice (5.19) che implementa in Python il seguente grafico.

```
def pred_vs_test(folder_path,y_test, y_predicted, norm, engine, pollutant):
    x_samp=range(len(y_test))
    fig=plt.figure(figsize=(12,10))
    ax1=fig.add_subplot(111)
    ax1.scatter(x_samp,y_test,s=100, c=None, edgecolors='b', marker='.')
    ax1.scatter(x_samp,y_predicted,s=100, c=None, edgecolors='r', marker='o')
    print(len(y_test),len(y_predicted))
    ax1.title.set_text('All points of prediction & Test')
    ax1.set_xlabel('n° of points')
    ax1.set_ylabel('y_pred[red] & y_test[blue]')
    plt.savefig(folder_path+"/y_pred[red] & y_test[blue].png")
    plt.show()
```

Figura 5.19. Codice in ambiente Python per "All points of prediction and Test"

Grafico - Cumulative Importance

Dal codice ottengo una classifica delle features più influenti. La funzione 'cum_imp_fig' prende in ingresso i valori cumulativi di importanza, la lista contenente i nomi delle features e l'ordine di importanza per costruire il grafico.

La figura (5.20) contiene in x le features; in y per ogni feature vengono segnati i valori della rispettiva importanza cumulata e questi vengono uniti dalla linea verde; la linea rossa rappresenta è segnata al punto dove l'importanza accumulata è pari al 90%.

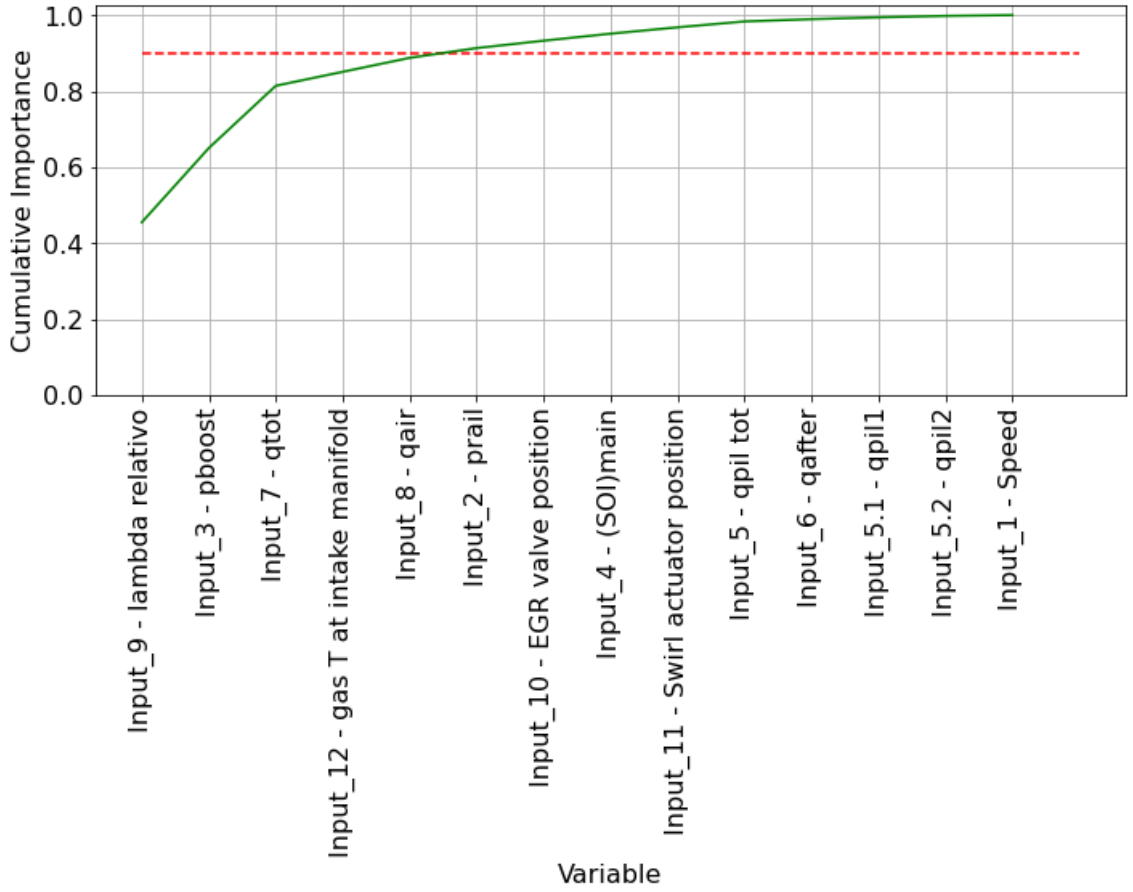


Figura 5.20. Esempio grafico Cumulative Importance

Note le features più importanti si proverà a riallenare il modello senza utilizzare le feature meno importanti. Per questo si è arbitrariamente deciso di prendere solo le features che danno un valore cumulato di importanza maggiore del 90%.

Si mostra anche in questo caso il codice (5.21) scritto in Python per l'implementazione del seguente grafico.

```python
def cum_imp_fig(feats_name, feat_imp_sorted_idx, cumulative_importances):
    # Plotting the cumulative feature importances and setting the threshold
    importances=list(feats_name[feat_imp_sorted_idx])
    x_values = list(range(len(importances)))
    plt.plot(x_values, cumulative_importances, 'g-')
    # Draw line at 90% of importance retained
    plt.hlines(y = 0.90, xmin=0, xmax=len(feat_imp_sorted_idx),
               color = 'r', linestyles = 'dashed')
    # Format x ticks and labels
    plt.yticks(np.arange(0, 1.1, 0.2))
    plt.xticks(x_values, importances, rotation = 'vertical')
    # Axis labels and title
    plt.xlabel('Variable'); plt.ylabel('Cumulative Importance'); plt.grid();
    fig=plt.gcf()
```

Figura 5.21. Codice in ambiente Python per "Cumulative Importance"

Grafico - Relative Importance

Dal codice ottengo una classifica delle features più influenti. La seguente funzione permette di costruire un horizontal bar plot con i valori di importanza delle features.

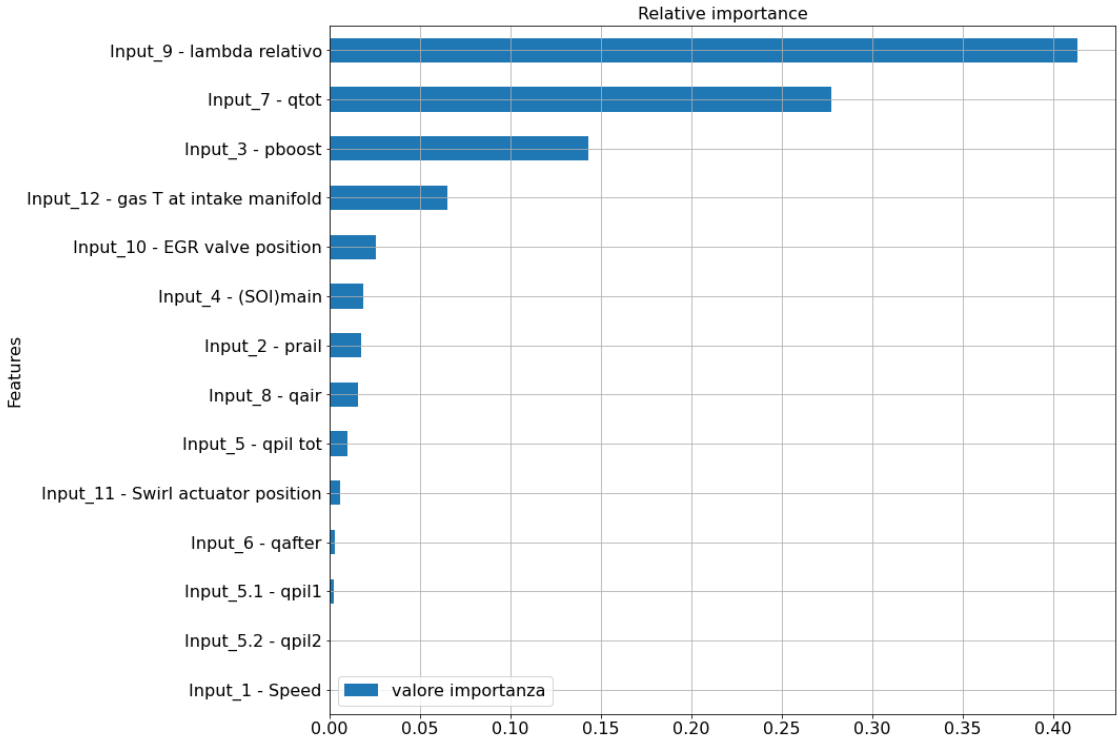


Figura 5.22. Esempio grafico Relative Importance

Nel grafico (5.22) vengono mostrati i parametri motoristici più influenti in ordine per la predizione dell'inquinante che andremo a studiare.
Il codice (5.23) scritto in ambiente Python si presenta così:

```
def save_bk_fig(folder_path,feat_imp_sorted, feats_name, feat_imp_sorted_idx):
    imp = pd.DataFrame({
        'valore importanza': feat_imp_sorted[::-1],
        'Features': feats_name[feat_imp_sorted_idx[::-1]],
    })
    imp.plot(kind='barh', x='Features', figsize=(15, 10),
             title='Relative importance', grid=True)
    #
    plt.savefig(folder_path+"/Features importance.png")
    plt.show()
```

Figura 5.23. Codice in ambiente Python per "Relative Importance"

Capitolo 6

Modello predittivo per motore 2.0L - inquinante “Soot”

Nel presente capitolo verrà analizzato il Dataset per il motore Diesel 2.0 litri di cilindrata con i rispettivi punti di funzionamento e successivamente si passerà ai risultati ottenuti con l’applicazione e la validazione dell’algoritmo di Machine Learning GBRT precedentemente affrontato.
Per la comprensione delle simulazioni è necessario effettuare uno studio fenomenologico sui parametri del motore che influiscono sull’emissione dell’inquinante. In questa casistica verrà analizzata l’emissione di inquinante di *particulate matter* cioè il particolato (soot).
Analizzando i punti di funzionamento e i parametri del motore noteremo l’influenza degli input del dataset in funzione dell’output.

Il motore Diesel a cui si fa riferimento in questo studio è un motore *GM-E11 Euro 5*. In tabella (6.1) vengono mostrate le specifiche di questo motore a combustione interna[2].
I punti motore analizzati sono presenti nella tabella (6.2) dove associamo ogni Key-points ad una strategia di iniezione ed il corrispondente numero di misurazioni effettuate.

Engine type	2.0 L "Twin-Stage" Euro 5
Displacement	1956 cm^3
Bore x stroke	83.0 mm x 90.4 mm
Connecting rod length	145 mm
Compression ratio	16.5
Valves per cylinder	4
Turbocharger	Twin-stage with valve actuators
Fuel injection system	Common Rail 2000 bar piezo
Specific power and torque	71 kW/l to 205 Nm/l

Figura 6.1. Specifiche motore GM-E11 Euro 5

Key-point	No. of tests
1500 x 2 PPM	147
1500 x 5 PPM	129
1500 x 5 PMA	131
2000 x 2 PPM	137
2000 x 5 PPM	134
2000 x 5 PMA	146
2500 x 8 PMA	130
2750 x 12 PMA	158

Figura 6.2. Key-points

I dati sperimentali sono stati registrati tramite un banco di test dinamico di ICEAL-PT (Internal Combustion Engine Advanced Laboratory at the Politecnico di Torino), in collaborazione di ricerca con GMPT-E (General Motors PowerTrain-Europe) per la calibrazione del motore[2].

Il Dataset per il motore Diesel da 2.0L di cindrata è quindi composto da:
•1112 punti di funzionamento motore: *rpm* (giri al minuto) x *pme* (pressione media effettiva);

•Due strategie di iniezione:
-PMA: pilot main after
-PPM: pilot pilot main
•Per ogni singolo punto di funzionamento si hanno 15 parametri misurate:
-14 parametri costituiscono gli input cioè le features
-1 parametro invece è l'output del problema nonchè il particolato

Nei prossimi sottocapitoli verrà testato e validato l'algoritmo GBRT dapprima nel caso in cui si usi il dataset completo nelle varie situazioni distinguendo le diverse configuazioni e in un secondo momento si passerà ad effettuare misurazioni con lo stesso algoritmo GBRT, ma operando con un dataset ridotto.
I dataset sui quali verranno effettuate misurazioni sono:
-Dataset completo comprendendo tutti i 1112 punti descritti precedentemente.
-Dataset PMA: sottoinsieme del completo con 565 punti di funzionamento con strategia di iniezione pilot main after
-Dataset PPM: sottoinsieme del completo 547 punti di funzionamento con strategia di iniezione pilot pilot main.
-Ulteriori prove verranno effettuate con Dataset ridotto mantenendo solo le features più importanti.

Prima di passare alla parte numerica è utile contestualizzare in breve le strategie di iniezione e il motivo per cui esse vengono utilizzate nel treno di iniezione.
In linea generale esse sono frazionamenti dell'iniezione che corrispondono a piccole percentuali di combustibile introdotto in camera prima o dopo l'iniezione principale per ovviare a delle problematiche.
Le diverse iniezioni, che la nuova generazione del sistema Common Rail consente di attuare, sono le seguenti:
Iniezione pilota: viene effettuata con elevato anticipo rispetto a quella principale, permette di abbassare drasticamente il livello del rumore di combustione, migliorando anche il funzionamento a freddo del motore e producendo un incremento della coppia ai bassi regimi. Consiste in una piccola percentuale di combustibile introdotta spesso il 5% del totale che consente di limitare la temperatura del ciclo motore andando a ridurre la formazione di NO_x.
Pre-Iniezione: effettuata con bassissimi valori di anticipo rispetto alla main e permette, assieme alla After, di modulare l'andamento della combustione contenendo le emissioni di inquinanti.

Main: è l'iniezione principale indispensabile per la produzione di potenza e produrre l'effetto utile del ciclo. La sua durata può essere gestita indipendentemente dalla pressione di iniezione;
After: Si indica un'iniezione effettuata subito dopo l'iniezione principale, che possiede lo stesso scopo della pre-iniezione. Presenta un breve ritardo e consente di raggiungere temperature più alte che sono favorevoli per continuare ad ossidare il soot.
Post-Iniezione: viene effettuata nelle ultime fasi della combustione con lo scopo di aumentare le temperature di scarico, queste temperature elevate dei gas di scarico permettendo (periodicamente) di riscaldare il DOC e poter far raggiungere alte temperature al DPF, trappola per il particolato, permettendone la rigenerazione. La post-iniezione permette inoltre di creare un ambiente riducente immancabile per la rigenerazione del catalizzatore DeNOx e ,perciò, per l'abbattimento degli ossidi di azoto.

Infine, tornando alla stuttura del dataset, nella figura (6.3) vengono presentati i parametri del motore 2.0L usati come features di input nell'algoritmo GBRT.

Input	Unità di misura	Descrizione
Speed	rmp	Velocità di rotazione del motore
p_rail	bar	Pressione di iniezione del combustibile (pressione nel rail)
p_boost	bar	Pressione dell'aria nell' intake manifold (collettore di aspirazione)
SOI_main	Deg before PMS	Inizio dell'iniezione della main
q_pil_1	mm^3/cyc/cyl	Quantità di combustibile iniettata durante la prima *pilot*
q_pil_2	mm^3/cyc/cyl	Quantità di combustibile iniettata durante la seconda *pilot*
q_pil_tot	mm^3/cyc/cyl	Quantità di combustibile totale della *pilot*
q_after	mm^3/cyc/cyl	Quantità di combustibile iniettata durante la *after*
q_tot	mm^3/cyc/cyl	Quantità totale di combustibile iniettato
q_air	mg/cyc/cyl	Quantità di aria iniettata in camera
Lambda relativo	-	Rapporto tra α e α_{stec}
EGR valve position	%	Posizione della valvola che controlla il flusso di gas esausti ricircolati (EGR)
Swirl actuator position	%	Indica il grado di *swirl*
T_gas_intake	°C	Temperatura del gas all'ingresso motore

Figura 6.3. Input di riferimento del modello

Mentre l'output usato come riferimento nel codice è mostrato nella figura (6.4).

Output	Unità di misura	Descrizione
PM experimental	mg/cyc	Quantità misurata di Particulate Matter

Figura 6.4. Output di riferimento del modello

6.1 Risultati dataset completo

Andiamo ora ad analizzare i risultati ottenuti tramite le simulazioni con il codice GBRT considerando la totalità del dataset fornito.
Verranno fatte delle considerazioni riguardo la scelta degli iperparametri e in seguito verranno valutate le performance del codice in termini di r^2 (coefficiente di determinazione) e $rmse$ (errore quadratico medio).
Dallo studio di questo caso verrano salvati anche i parametri più rilevanti in modo da sostenere nel sottocapitolo successivo anche delle ulteriori prove.
Relativamente agli studi saranno considerate anche casistiche con 3 diverse normalizzazioni dei dati elencate di sotto e mostrate nel codice Python (6.5)):
-Nessuna normalizzazione (Norm 0)
-Norma 1 (Norm 1)
-Norma 2 (Norm 2)

```
def norm(X_train, y_train, X_test, y_test, xmax, xmin, ymax, ymin):
    norm=int(input(
 'Inserire il tipo di normalizzazione che si vuole effettuare sul dataset:

    if norm==0:
        X_train=X_train
        y_train=y_train
        X_test=X_test
        y_test=y_test
    if norm==1:
        X_train=X_train/xmax
        y_train=y_train/ymax
        X_test=X_test/xmax
        y_test=y_test/ymax
    if norm==2:
        X_train=(X_train-xmin)/(xmax-xmin)
        y_train=(y_train-ymin)/(ymax-ymin)
        X_test=(X_test-xmin)/(xmax-xmin)
        y_test=(y_test-ymin)/(ymax-ymin)
```

Figura 6.5. Normalizzazioni implementate nel codice

Assieme ad esse saranno considerate anche delle divisioni per quanto riguarda il train test split considerando come "test size" il 20%, 40%, 60% e 80% dei dati a disposizione.
tts = 0.2 → 20% Train, 80% Test
tts = 0.4 → 40% Train, 60% Test
tts = 0.6 → 60% Train, 40% Test
tts = 0.8 → 80% Train, 20% Test

Si supponga di avere 100 dati, se scegliessi 0.2 si userebbero 80 dati per il training e 20 per il testing; Se invece scegliessi 0.4 userei 60 dati per il training e 40 per il testing, scegliendo 0.6 userei 40 dati per il training e 60 per il testing e così via.
Come ovvio aspettarsi le performance calano notevolmente al diminuire dei dati che si usano per allenare il modello e questo fa capire come nel Machine Learning la cosa più importante e fondamentale siano i dati che si hanno a disposizione.
Per comprendere meglio:
se supponessi di essere in una condizione dove per migliorare il modello si ha la possibilità di scegliere fra effettuare l'ottimizzazione di BayesSearchCV/GrideSearchCV oppure aggiungere altri dati, si deve assolutamente scegliere la seconda opzione senza pensarci due volte perchè le performance dei modelli migliorano a volte in modo esponenziale all'aumentare dei dati a disposizione molto di più che facendo il tuning dei parametri in maniere più o meno efficienti.

L'implementazione nel codice di queste taglie di train e test split vengono mostrate nell'immagine (6.6)

```
def tts(X, y):
    testsize=float(input(
  'Inserire il numero corrispondente al valore della dimensione del testset:

    if testsize==0.2:
        test_size=0.2

    elif testsize==0.4:
        test_size=0.4

    elif testsize==0.6:
        test_size=0.6

    elif testsize==0.8:
        test_size=0.8

  X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=test_size)
  return X_train, X_test, y_train, y_test
```

Figura 6.6. Train/Test split size

Questo viene fatto poichè una delle tecniche usate per la validazione del codice è quella di analizzare l'andamento dell'accuratezza in funzione della dimensione dei dati di training e testing forniti, così da poter constatare la robustezza del codice.

Poiché gli algoritmi ad albero non si basano su distanze euclidee non sono influenzati in grande maniera dal tipo di normalizzazione e per questo magari ottengo valori finali di r^2

che sono simili.
Il fatto che raramente abbiamo ottenuto valori anche maggiori nel caso senza normalizzazione è una motivazione stocastica in quanto i modelli stessi essendo basati su assunzioni statistiche non è detto che diano valori perfettamente simili di accuratezza o r^2 ad ogni training.
Il modello che è stato tratto è un modello statistico e quindi se si facesse n volte lo stesso training otterrei n valori che si scostano di poco l'uno rispetto all'altro e mai esattamente uguali se non in casi random.
Ogni *run* perciò è stato ripetuto tre volte e i risultati sono stati mediati al fine di evitare errori di tipo probabilistico e/o Overfitting o Underfitting dell'algoritmo.
Nelle tabelle sotto riportate sono mostrati i valori medi per ogni caso e vengono messi in evidenza il miglior valore, in verde, e il peggiore, in rosso, in modo da poter mostrare ed analizzare gli andamenti delle curve.

Per questo codice sono state scelte due differenti strade per l'ottimizzazione degli iperparametri, il BayesSearchCV e GridSearchCV, come spiegato nel capitolo precedente, in modo da dimostrare quale soluzione porta ad un miglior valore di r^2 che tende il più possibile ad 1.
Partamo anzitutto dalla considerazione che i due metodi sono finalizzati alla ricerca del set degli iperparametri che rendono il più performante possibile il nostro modello.
Gli iperparametri sono parametri del modello che fanno parte dell'algoritmo stesso sul quale si basa il nostro modello, per questo motivo non possono essere "imparati" dai dati durante la fase di training del modello.
Come ovvia conseguenza perciò l'unico modo per trovarli è quello di osservare come varia l'output del modello in funzione alla variazione degli iperparametri col fine di utilizzare infine quelli che corrispondono all'output migliore per il problema trattato.

GridSearchCV

Il procedimento è il più diretto ed intuitivo possibile e consta dei seguenti passi:

1)Si sceglie il modello ML che si vuole usare: Gradient Boosting Regressor Tree

2) Si scelgono in modo arbitrario gli iperparametri del modello che si vogliono sottoporre al tuning, mostrati in figura (6.7)

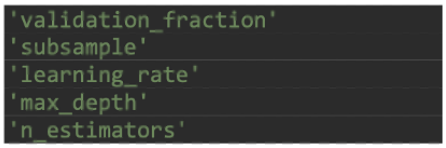


Figura 6.7. Iperparametri scelti per il GridSearchCV

3) Si assegna ad ogni iperparametro un set di valori casuali e nel range di attinenza[1] che si vogliono provare, figura (6.8)

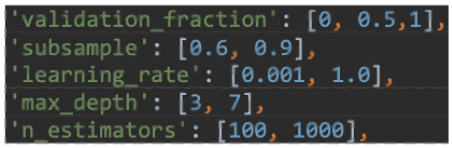


Figura 6.8. Valore degli iperparametri scelti per il GridSearchCV

4) Si allena il modello su tutte le combinazioni senza ripetizione possibili con la serie di parametri definiti. Nel nostro caso quindi si proveranno 3x2x2x2x2=48 combinazioni di iperparametri che rappresentano tutte le combinazioni dei valori da provare precedentemente assegnati al punto 3, il tutto moltiplicato per k-fold, nel nostro caso 5 per un totale di 240 casistiche.

5) Si sceglie il set di iperparametri che hanno prodotto il migliore risultato.

Si noti infatti che viene utilizzato il GridSearchCV ed il CV finale sta per Cross-Validation. Questo è un metodo di training che nasce per permettere di avere un'idea generalizzata di quanto il modello possa essere performante una volta allenato: il set di dati di allenamento vengono divisi in n parti chiamate "fold", nel nostro caso per esempio si è deciso di dividerlo in 5.
Si allena quindi il modello a ruota su 4/5 dei dati e si testa sul 1/5 e si calcola il valore medio della metrica di valutazione per tutti i fold.

La tabella (6.9) porta i risultati di "accuratezza" espressi con r^2 coefficiente di determinazione ottenuti per lo studio dell'inquinante soot nel motore Diesel da 2.0 litri di cilindrata

[1] In questo caso validation fraction verrà provato con i valori 0, 0.5 ed 1 mentre subsample verrà provato con 0.6 ed 0.9

in funzione del dataset di input ed output registrati tramite banco motore.
Nella tabella (6.10) invece possiamo osservare i corrispondenti valori di rmse per ogni caso analogo.

GBRT 2.0L Accuracy	Norm0	Norm1	Norm2
TTs=0.2	0,771	0,799	0,803
TTs=0.4	0,754	0,773	0,781
TTs=0.6	0,723	0,733	0,751
TTs=0.8	0,644	0,704	0,714

Figura 6.9. Risultati espressi in r^2 per l'algoritmo Gradient Boosting Regression Trees relativamente al Dataset completo del motore GM-E Euro 5 Diesel 2.0 L con ottimizzazione GridSearchCV

GBRT 2.0L rmse	Norm0	Norm1	Norm2
TTs=0.2	0,029	0,035	0,045
TTs=0.4	0,042	0,044	0,048
TTs=0.6	0,043	0,052	0,050
TTs=0.8	0,059	0,056	0,053

Figura 6.10. Risultati espressi in *rmse* per l'algoritmo Gradient Boosting Regression Trees relativamente al Dataset completo del motore GM-E Euro 5 Diesel 2.0 L con ottimizzazione GridSearchCV

Salta subito all'occhio il valore massimo raggiunto per $TTs = 0.2$ con $Norm_2$ di 0,803, questo valore però soddisfa minimante l'obiettivo prefissato, cioè quello di effettuare con elevata accuratezza il calcolo predittivo per l'inquinante in questione (PM).
Infatti soltanto un valore di r^2 superiore a circa 0.8 può essere considerato soddisfacente per l'obiettivo. Questo valore però si desideri non crolli vertiginosamente col crescere di TTs. In questo studio il valore arriva al minimo a 0,644 che potrebbe risultare accettabile considerando che sfruttiamo un TTs di 0.8 che significa che solo il 20% dei dati è usato

per l'addestramento. Seppur reputato accettabile si preferisce avere come valore minimo in questo studio un valore che non scenda sotto lo 0.7.

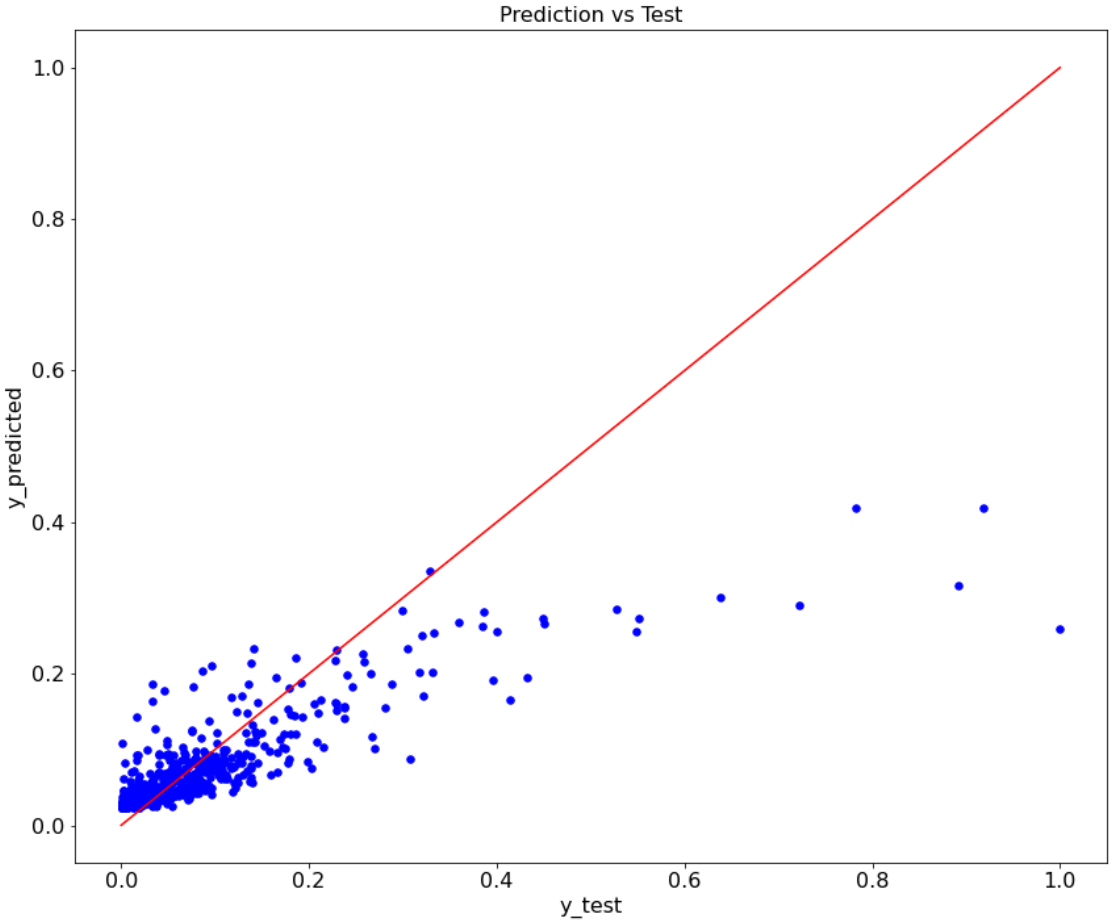


Figura 6.11. Grafico "Prediction vs Test" ottenuto per il peggior caso $TTs = 0.8$ con $Norm_0$ col metodo GridSearchCV

Nell'esempio successivo si avrà modo di approfondire gli andamenti dei punti nei grafici ottenuti, ma in quest'esempio, seppur dimostrato che non soddisfa gli ideali di Machine Learning, si vuole presentare il grafico "Prediction vs Test" ottenuto per mostrare l'enorme distanza che c'è tra i valori predetti e quelli calcolati a banco. All'occhio subito salta la distanza dei punti dalla curva di ottimo segnata in rosso che si traduce in un inesattezza del processo che non porta a dei valori predetti attendibili.

Per queste motivazioni perciò si scegli di ottimizzare il set di iperparametri con la BayesSearchCV, che è un secondo metodo finalizzato alla ricerca degli iperparametri che permette di rendere più performante possibile il nostro modello di Machine Learning.

BayesSearchCV

Usando il BayesSearchCV in pratica implementiamo l'ottimizzazione *Bayesiana* la quale sfrutta il teorema di Bayes per ricercare gli iperparametri ottimali.
Il procedimento in questo caso consta dei seguenti passi:

1) Si sceglie il modello ML che si vuole usare: Gradient Boosting Regressor Tree

2) Si scelgono in modo arbitrario gli iperparametri del modello che si vogliono sottoporre al tuning (fig. (6.12))

```
'validation_fraction'
'subsample'
'learning_rate'
'max_depth'
'n_estimators'
```

Figura 6.12. Iperparametri scelti per il BayesSearchCV

3) Si assegna ad ogni iperparametro un set di valori casuali e nel range di attinenza che si vogliono provare come nell'immagine (6.13).

```
'validation_fraction': Real(0, 1),
'subsample': Real(0.6, 0.9),
'learning_rate': Real(0.001, 1.0),
'max_depth': Integer(3, 7),
'n_estimators': Integer(100, 1000),
```

Figura 6.13. Valore degli iperparametri scelti per il BayesSearchCV

4) Si fissa il numero di iterazioni in modo arbitrario ed ad ogni iterazione ogni parametro viene scelto tramite il teorema di Bayes in modo da ottimizzare la funzione obiettivo.

5) Si sceglie il set di iperparametri che hanno prodotto il migliore risultato.
Per la scelta del numero di iterazioni risulta che essa è arbitraria. Tuttavia un numero troppo elevato causa overfitting mentre un numero troppo basso non permette di trovare

il valore ricercato. Anche qui si procede per esperienza agendo sul range dei parametri e sul numero di iterazioni a seconda dei casi.
Di solito però se si ottiene che alla fine il r^2 sul test set è basso allora si prova ad alzare il numero delle iterazioni e viceversa.

L'implementazione della funzione BayesSearchCV nell'algoritmo GBRT avviene secondo qeste poche righe di codice, figura (6.14).

```
bayes_cv_tuner = BayesSearchCV(estimator=GradientBoostingRegressor(),
                               search_spaces=params_bayes,
                               scoring='r2',
                               cv=5,
                               n_jobs=-1,
                               n_iter=n_iterazioni,
                               verbose=0,
                               refit=True,
                               random_state=42)
```

Figura 6.14. Pezzo di codice che implementa il BayesSearchCV nel modello GBRT

Il parametro $n\_iter = n\_iterazioni$ è un valore che non viene fissato all'interno della funzione per permettere una più facile modifica. In questo studio il parametro è stato fissato a 20 iterazione, poiché andando ad aumentarle non si notavano ulteriori miglioramenti.
Invece il parametro $cv = 5$ è il "k-fold" della cross validation, cioè suddivido il training set in k parti uguali e uso 1/k per la validation e k-1/k per il training set. Questa suddivisione random avverrà per k volte, cioè 5, considerando ogni volta un sottoinsieme diverso come validation set.

Con questa soluzione per l'ottimizzazione degli iperparametri notiamo sostanziali miglioramenti per quanto riguarda i valori di r^2 e $rmse$ come si evince dalla tabella (9.1) per il valore di accuratezza e dalla tabella (6.16) per il valore di errore quadratico.

GBRT 2.0L Accuracy	Norm0	Norm1	Norm2
TTs=0.2	0,933	0,942	0,941
TTs=0.4	0,857	0,870	0,896
TTs=0.6	0,895	0,853	0,863
TTs=0.8	0,712	0,751	0,827

Figura 6.15. Risultati espressi in r^2 per l'algoritmo Gradient Boosting Regression Trees relativamente al Dataset completo del motore GM-E Euro 5 Diesel 2.0 L con ottimizzazione BayesSearchCV

GBRT 2.0L rmse	Norm0	Norm1	Norm2
TTs=0.2	0,013	0,021	0,019
TTs=0.4	0,028	0,032	0,030
TTs=0.6	0,019	0,035	0,035
TTs=0.8	0,037	0,049	0,040

Figura 6.16. Risultati espressi in *rmse* per l'algoritmo Gradient Boosting Regression Trees relativamente al Dataset completo del motore GM-E Euro 5 Diesel 2.0 L con ottimizzazione BayesSearchCV

Si osserva come una riduzione della dimensione del training dataset, con contemporaneo aumento di quella del test dataset, porti ad una diminuzione delle performance, dovuto al fatto che il modello si allena su una quantità di dati minore. Tutto questo però in una maniera molto più alta del caso precedente.
Con l'ottimizzazione degli iperparametri col metodo di Bayes infatti arriviamo ad un valore medio massimo di 0,942, nettamente superiore al 0,803 trovato in precedenza col metodo Grid.
Anche per il valore medio minimo, che troviamo per il $TTs = 0.8$ si hanno valori superiori e perfettamente accettabili.

Il miglior valore medio che si ottiene per questo studio è naturalmente per un $TTs = 0.2$

e si è ottenuto grazie alla normalizzazione di tipo 1.
In questo miglior caso si mostrano e si analizzano gli andamenti delle curve che rispondono alla soluzione del problema riportate nei grafici sottostanti.

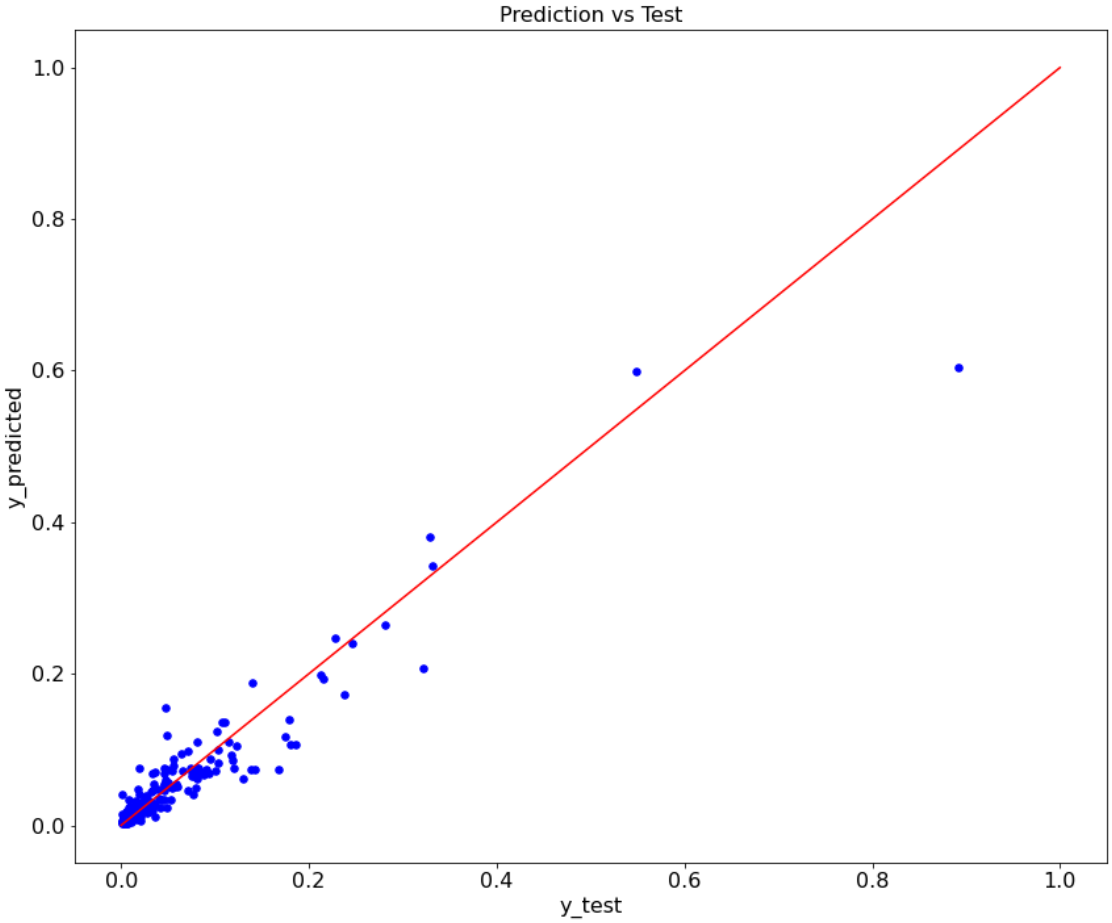


Figura 6.17. Grafico ”Prediction vs Test” ottenuto per il miglior caso (TTs=0.2 con Norm1)

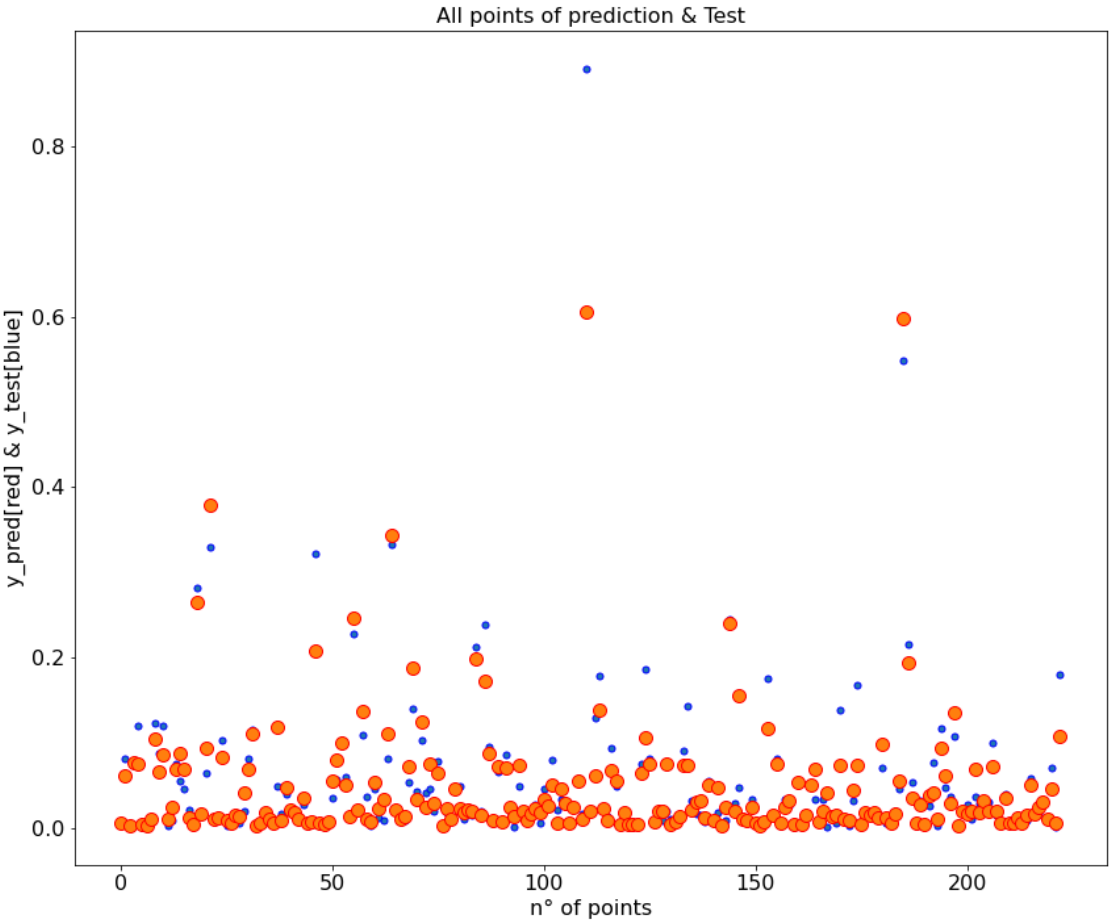


Figura 6.18. Grafico “All points of prediction & Test” ottenuto per il miglior caso (TTs=0.2 con Norm1) dove tutti i punti della predizione e del test vengono sovrapposti

Il grafico (6.17) mette in x i valori y_test ed in y i valori $y\_predicted$ che sarebbero rispettivamente i valori originali e i valori predetti dal codice e poi traccia una linea rossa che congiunge i punti [0,0][1,1].
In questo caso vediamo come si ottengono valori nettamente nell’intorno della curva rossa che rappresenterebbe la perfezione della predizione. Si presentano anche pochi ”outlier” cioè dei valori anomali che si discostano maggiormente dall’andamento che si desidera.
Nel grafico (6.18) invece vengono applicati i valori originali cioè il target dell’emissione di soot e quelli predetti cioè i risultati trovati dal codice GBRT. Anche in questo grafico ottenuto per $TTs = 0.2$ con $Norm_1$ si presenta un coincidenza rilevante che giustifica a pieno il 0,942 ottenuto, presentando nuovamente un solo outlier eclatante.
È interessante osservare ancora anche i valori che assumono gli iperparametri nella miglior

condizione, figura (6.19) *Best Hyperparameters*.

GBRT_Best_Parameters	Valori
learning_rate	0,01661583
max_depth	6
n_estimators	825
subsample	0,6
validation_fraction	1

Figura 6.19. Migliori valori ottenuti per gli iperparametri scelti nel modello GBRT per il caso $TTs = 0.2$ con $Norm_1$

Ora si procede col vedere la differenza tra i grafici precedenti con i grafici ottenuti per il caso peggiore in cui si arriva ad un valore di r^2 di 0,712.

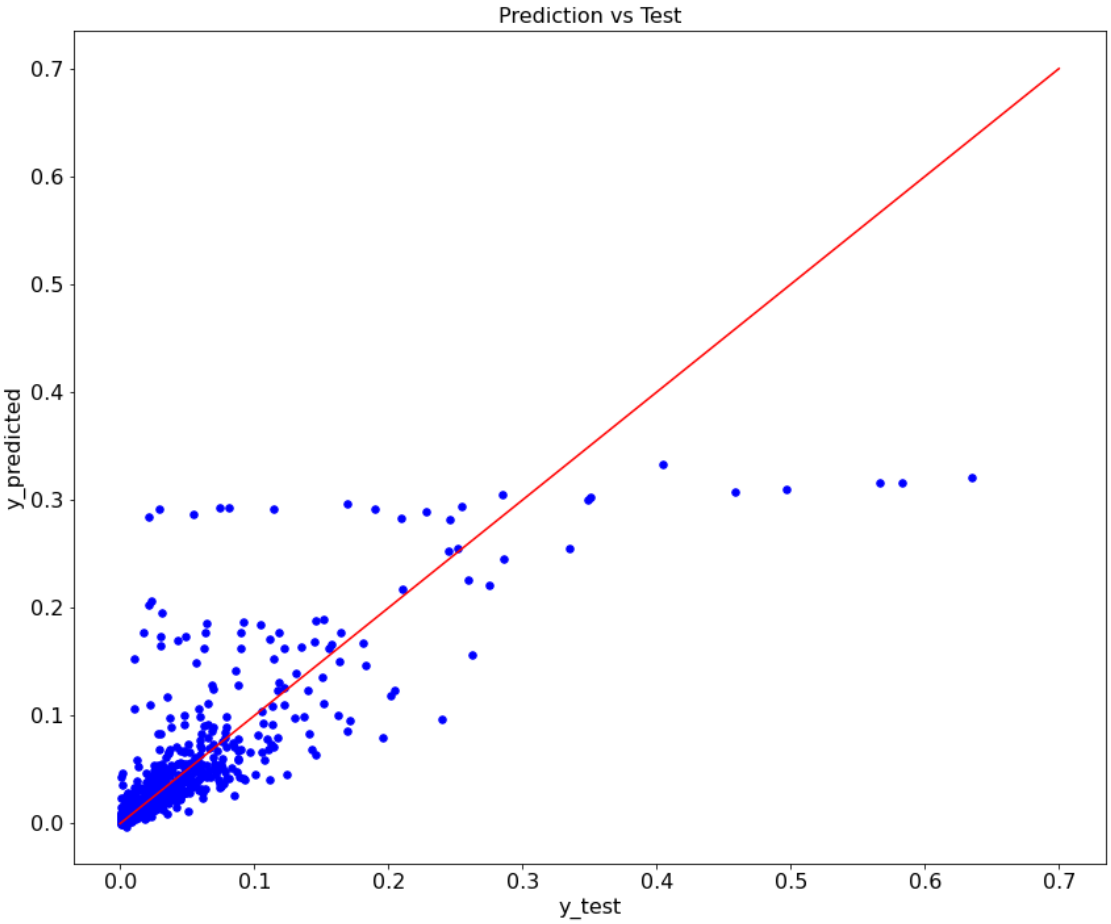


Figura 6.20. Grafico ”Prediction vs Test” ottenuto per il peggior caso (TTs=0.8 con Norm0)

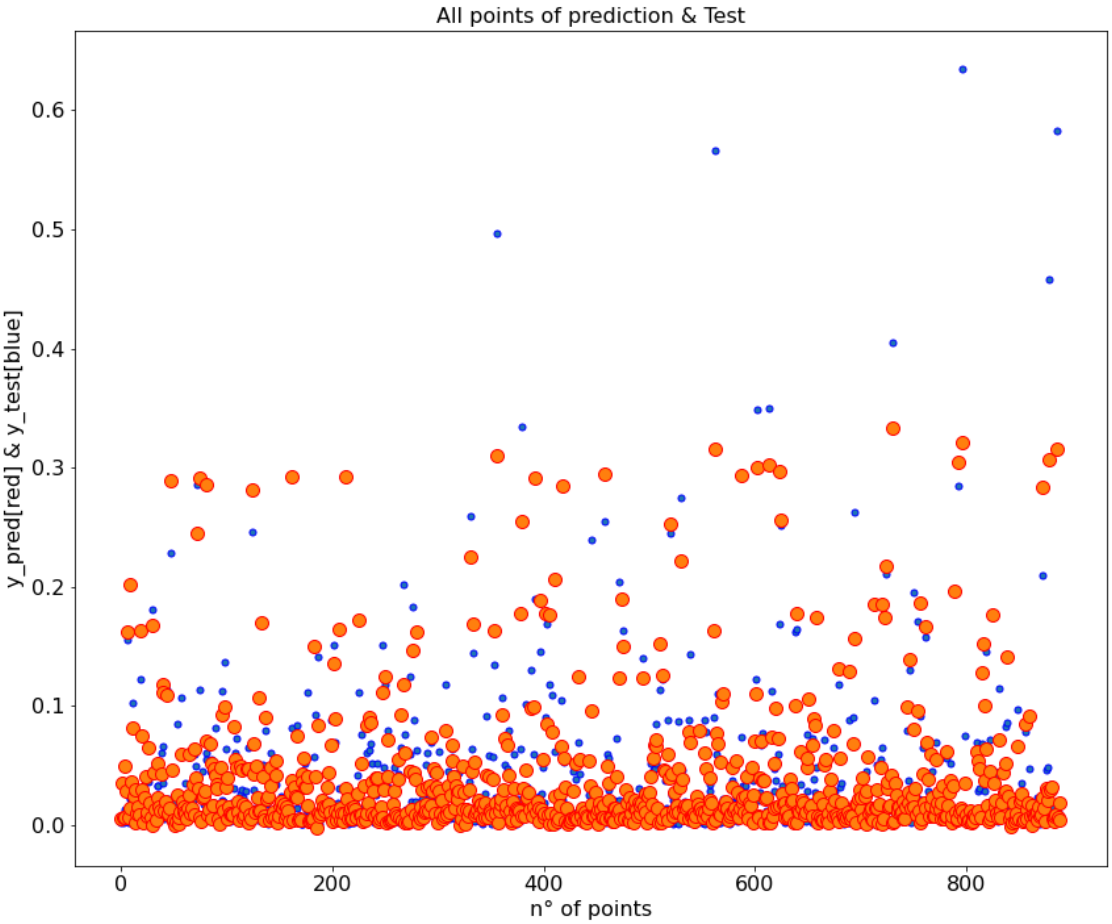


Figura 6.21. Grafico “All points of prediction & Test” ottenuto per il peggior caso (TTs=0.8 con Norm0) dove tutti i punti della predizione e del test vengono sovrapposti

Immediato all'occhio il fatto che nel peggior caso di predizione che si ha nel codice GBRT ci sia un distacco dalla curva rossa nel grafico (6.20) dove si vede chiaramente che i punti predetti sono lontani dalla linea a 45 gradi rossa che simula un algoritmo con accuratezza 100% cioè 1.
Sull'asse y abbiamo i valori predetti e sull'asse x i valori test predetti che se fossero coincisi troverebbero il punto sulla bisettrice a 45 gradi.
Questo grafico presente un gran numero di outliers che giustifica il basso valore di accuratezza ottenuto con la predizione.
Nell'immagine (6.21) si mostra un altro grafico che rende comprensibile la gran distanza, rispetto al miglior caso, fra i valori di inquinante predetti e quelli misurati sul banco prova.

In relazione alla tabella (9.1) per il seguente studio è stato fatto un grafico Excel (6.22) per valutare l'andamento delle accuratezze in funzione della normalizzazione e del TTs. Osservando gli andamenti delle tre curve di si può dimostrare che si ottiene un andamento decrescente in funzione dell'aumentare del $train\_test\_split$, cioè al diminuire dei dati utilizzati per l'addestramento c'è un ovvia riduzione dell'accuratezza ottenuta con il calcolo predittivo.
Inoltre grazie alle medesime curve si registra un andamento comune tra le normalizzazioni effettuate, che vedono una preferenza della normalizzazione 2 ($Norm_2$) ad esclusione del $TTs = 0.2$ dove, se pur di pochissimo, la normalizzazione 1 ($Norm_1$) registra il picco massimo del coefficiente di determinazione r^2.
Infine si prende nota di un eccezione nel caso $TTs = 0.6$ dove la $Norm_0$ presenta un valore di accuratezza superiore alle altre due normalizzazioni.

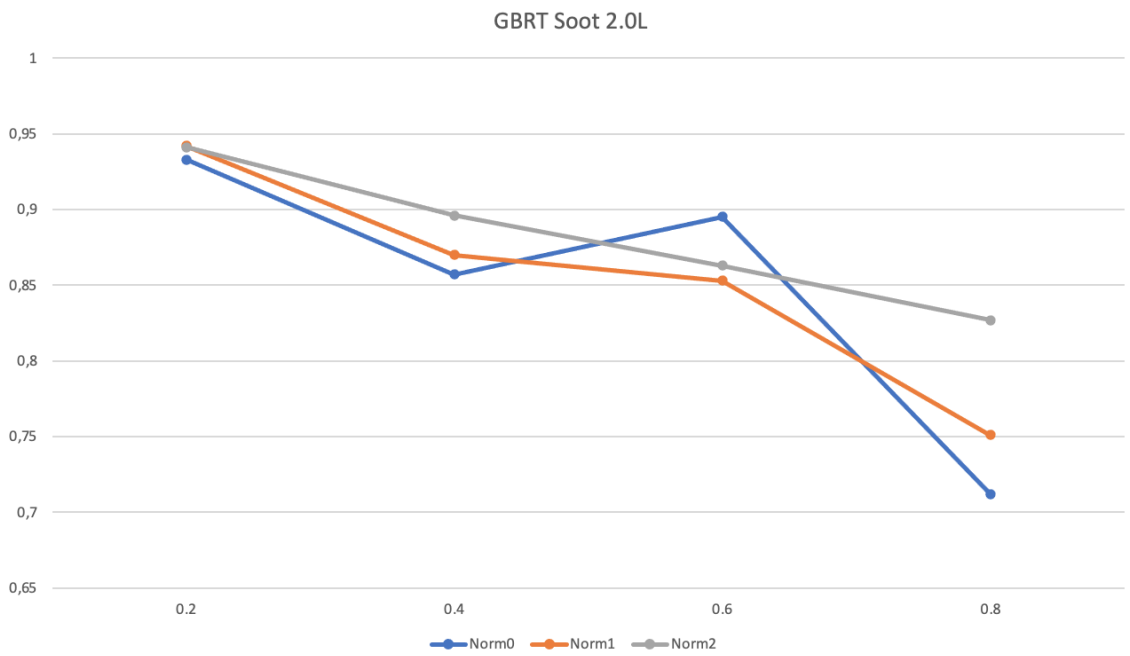


Figura 6.22. Andamento delle tre normalizzazioni in funzione del r^2 e dei 4 TTs per il motore 2.0L nel caso dell'inquinante *soot*

6.2 Risultati dataset ridotto

6.2.1 Dataset Features Importance

In questo capitolo vengono presentati ulteriori casi studio che analizzano maggiormente, tramite l'algoritmo in questione, il dataset di dati che si hanno per il calcolo predittivo dell'emissione inquinante di soot.

È interessante analizzare il TTs che porta risultati migliori ($TTs = 0.2$) e vedere la dipendenza di esso dalle features maggiormente importanti tramite *features importance*.
Solitamente si prova a sfruttare tutte le features che si hanno, ma non è sempre conveniente.
Quello che succede è che fino ad una certa soglia l'accuratezza che ottieni aumenta all'aumentare delle features (perché aumenta l'informazione che stai dando al modello) dopo questa soglia però l'accuratezza comincia a scendere perché si ottiene il cosiddetto "curse of dimensionality" cioè non si hanno abbastanza osservazioni rispetto alla dimensionalità del problema.
Nel caso in esame si sta usando il modello stesso per definire l'importanza di ciascuna

feature. Questa di solito è misurata sulla base dell'autovalore associato all'autovettore della matrice degli input.
Si metteno a schermo i grafici (6.23) e (6.24)[2] dove si mostrano in ordine le *features più importanti* quelle cioè che apportano maggiore informazione alla risoluzione del problema. Per constatare di non incorrere nel "curse of dimensionality" successivamente si guarda il valore di r^2 in funzione delle 7 features più influenti.
Di solito accade che si ha bisogno di scegliere le features perché ci si può trovare con dei datasets che contengono anche 1000 colonne, in questo caso ridurre le colonne mantenendo quanta più informazione possibile è fondamentale per far funzionare gli algoritmi molto più grossi.

Nel dettaglio il grafico (6.23) mostra una classifica delle features più influenti. La seguente funzione permette di costruire un "horizontal bar plot" con i valori di importanza delle features.
Questi valori vengono calcolati durante l'addestramento per ogni TTs, indipendentemente dal tipo di normalizzazione, in modo da mostrare le 7 features che descrivono meglio la situazione.
Sarà dunque il codice stesso ad evidenziare le features più importanti.

[2]I seguenti grafici sono stati valutati per la condizione di $TTs = 0.2$

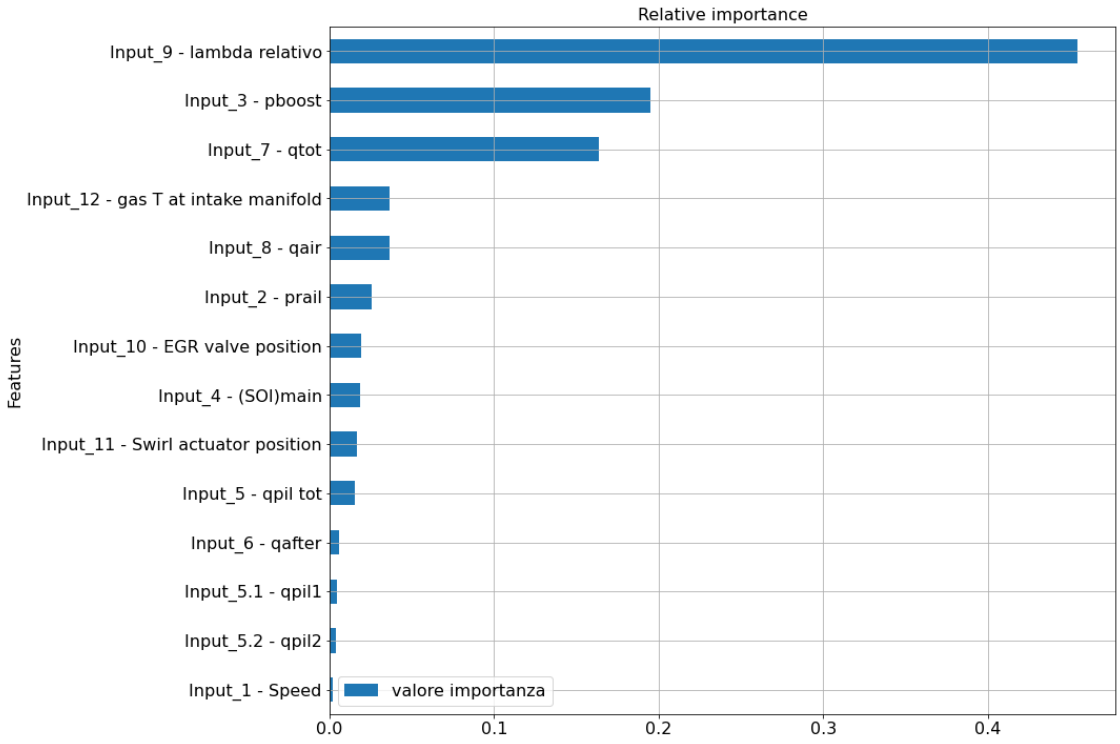


Figura 6.23. Istogramma che descrive l’importanza relativa di ogni singola feauteres

Nel secondo grafico che descrive la features importance (6.24) ottengo sempre tramite il codice principale una classifica delle features più influenti con un valore cumulativo.

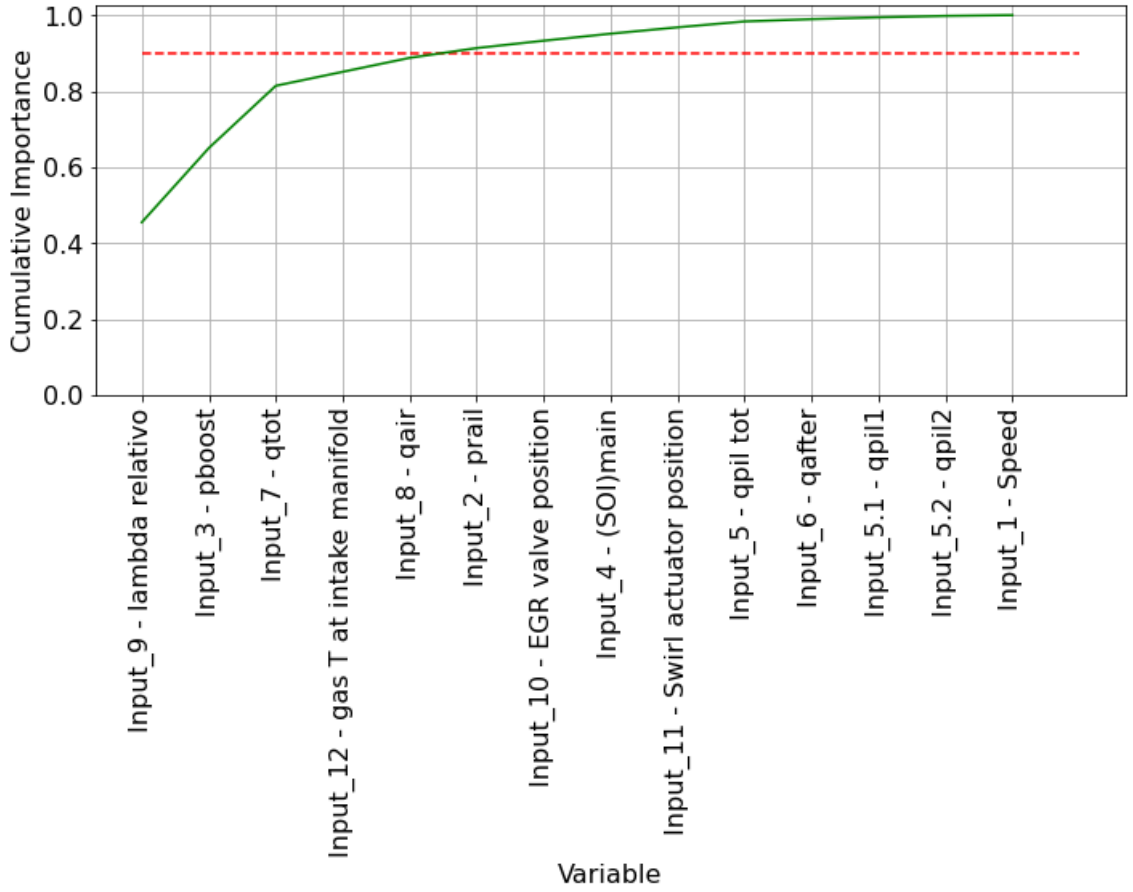


Figura 6.24. Importanza cumulativa delle features utilizzate

La funzione $'cum\_imp\_fig'$, come mostrato nel capitolo precedente tramite il pezzo di codice prende in ingresso i valori cumulativi di importanza, la lista contenente i nomi delle features e l’ordine di importanza per costruire il grafico.
La figura contiene in x le features; in y per ogni feature vengono segnati i valori della rispettiva importanza cumulata e questi vengono uniti dalla linea verde; la linea rossa rappresenta è segnata al punto dove l’importanza accumulata è pari al 90%.
Note le features più importanti si prova ora a riallenare il modello senza utilizzare le feature meno importanti. Per questo si è arbitrariamente deciso di prendere solo le features che danno un valore cumulato di importanza maggiore del 90% ottenendo così i valori presenti nella tabella (6.25)

(Dateset ridotto) TTs=0.2	Norm0	Norm1	Norm2
Accuracy	0,866	0,887	0,896
rmse	0,0187	0,029	0,026

Figura 6.25. Risultati espressi in r^2 e *rmse* con dataset ridotto considerando features importance

I valori numerici registrati in tabella rispondono al calcolo predittivo dell'inquinante soot nel caso in cui si usasse un $TTs = 0.2$, e dunque l'80% dei dati del dataset utilizzati per l'addestramento, con le tre diverse tipologie di normalizzazione.
In tal modo si è dimostrata una certa corrispondenza tra l'algoritmo (che si dissocia totalmente dall'aspetto fenomenologico del problema) e il modello utilizzato (GBRT) che individua degli input che hanno un peso maggiore di altri.

I valori numerici, mediati su 3 prove effettuate, soddisfano pienamente l'aspettativa riposta nell'algoritmo per il calcolo di questo inquinante. I risultati si presentano più bassi dei valori ottenuti col dataset completo perché si sta andando a togliere informazioni all'algoritmo che, anche se in minor parte influenti, possono contribuire all'innalzamento dell'accuratezza.
Questi risultati, inoltre sottolineano le sette features (input del problema) che maggiormente contribuiscono per la predizione.

1) *Lambda relativo*: rapporto aria carburante rispetto al valore stechiometrico
2) *p_boost*: pressione dell'aria all'aspirazione
3) *q_tot*: quantità di combustibile totale
4) *T_gas_intake*: temperatura del gas all'ingresso motore
5) *q_air*: quantità di aria iniettata in camera
6) *p_rail*: pressione di iniezione del combustibile
7) *EGR valve position*: posizione valvola che controlla il flusso gas esausti ricircolati

Questi input sono strettamente legati al combustibile e all'aria, variabili che, per il modello semi- empirico di partenza, sono connesse alla formazione e all'ossidazione del soot.

6.2.2 Dataset PMA

Il codice e dunque lo studio condotto sul dataset dei punti di funzionamento con strategia di iniezione Pilot Main After è identico alla casistica precedente, con la differenza di una quantità di dati inferiore.
Questo sottocapitolo ed il prossimo non mirano ad uno studio fatto per massimizzare il rendimento dell'accuratezza del codice, ma per verificarne la robustezza avendo a disposizione una minor quantità di dati.
Come vedremo nella tabella (6.26) le prestazioni saranno inferiori, anche se soddisfacenti, poiché andremo a sottoporre al codice un minor numero di dati sul quale allenarsi e costruire la funzione predittiva per l'inquinante in questione.
Un'analisi di questo tipo, può fornire un'indicazione sulla taglia del dataset utilizzato per lo studio, notando l'andamento dell'r^2 al diminuire dei dati di training.

PMA Accuracy	Norm0	Norm1	Norm2
TTs=0.2	0,844	0,861	0,856
TTs=0.4	0,808	0,803	0,825
TTs=0.6	0,768	0,746	0,781
TTs=0.8	0,675	0,714	0,716

Figura 6.26. Risultati GBRT sul dataset PMA

I parametri motoristici più rilevanti restano pressoché gli stessi fino a che però il dataset sia costituito da un numero sufficiente di dati. I dati a disposizione nei casi PMA e PPM comunque sono sufficienti per uno studio di robustezza anche se osserveremo nel caso in cui il TTs sia 0.8 si vanno ad ottenere dei valori veramente molto bassi 0,675 per il quale variano anche i parametri motori che descrivono il problema. Queste nuove variabili però non possono essere ritenute attendibili per descrivere dal punto di vista fenomenologico lo studio.

6.2.3 Dataset PPM

Analogamente a quanto fatto nel sottocapitolo precedente, si riportano i risultati per il dataset contenente i dati riguardanti la strategia di iniezione Pilot Pilot Main nella tabella (6.27).
In questo caso rispetto al precedente però notiamo dei valori leggermente migliori e del tutto considerabili anche se ottenuti con un minor numero di dati.

PPM Accuracy	Norm0	Norm1	Norm2
TTs=0.2	0,872	0,895	0,887
TTs=0.4	0,831	0,863	0,858
TTs=0.6	0,801	0,829	0,827
TTs=0.8	0,740	0,756	0,762

Figura 6.27. Risultati GBRT sul dataset PPM

Anche in questo caso, come giusto che sia, si ottengono valori minori rispetto al caso col dataset completo che però consentono di constatare la robustezza del codice GBRT.

Capitolo 7

Modello predittivo per motore 11.0L - inquinante “NOx”

Nel presente capitolo si andrà ad adottare il codice di Machine Learning scritto in ambiente Python per un motore Diesel differente. Tratteremo un motore Diesel Heavy Duty Cursor 11.0 Litri a 6 cilindri immagine (7.1), conforme agli standard europei Euro VI, per il calcolo predittivo dell'inquinante NO_X.

Figura 7.1. Motore 11.0 Litri HD

Le misurazioni dei punti di funzionamento e dei parametri motoristici sono state effettuate in condizioni *steady state* cioè in condizioni stazionarie, con un numero totale di punti motore pari a 4879, quantità nettamente superiore al caso precedente del motore 2.0L di cilindrata.
In un primo momento il dataset completo contenente 4879 punti con 206 variabili rilevate, ha subito una prima scrematura a causa di errori di misura e variabili non significative, modificando la sua forma diventando così un nuovo dataset composto da 4711 punti per 27 parametri[2].
Il dataset finale è suddiviso in sei parti, dipendentemente dalla configurazione delle misurazioni:
-Prove su engine map senza EGR: totale 152 punti;
-Prove con diversi valori di SOI e pressione nel rail senza EGR: totale 949 punti;
-Prove su engine map con EGR: totale 149 punti;
-Prove con diversi valori di SOI e pressione nel rail con EGR: totale 959 punti;

-Trade off EGR/VGT[1] totale 421 punti; -DoE ASCMO: totale 2081 punti.
L'ultimo caso non viene considerato in questo studio. Molte di queste casistiche sono invece considerate due volte poichè misurate sia su banco prova e sia stimate in centralina ECU (engine control unit).
Dunque vengono creati due Dataset in funzione della metodologia con cui sono stati ottenuti i dati:

-**Bench Dataset**: 2630 x 15 variabili (13 input e 2 tipologie di output);

-**ECU Dataset**: 2630 x 15 variabili (13 input e 2 tipologie di output).

Come si nota, rispetto al caso affrontato nel capitolo precedente, si possiede una dimensione del dataset molto più grande, più del doppio, sia in termini di variabili che in quantità di dati per ognuna.
Avendo perciò un allungamento dei dati si sceglierà un range di scelta degli iperparametri più piccoli per compensare i tempi di computazione del modello.
I parametri motoristici valutati, come nel caso precedente, costituiscono le features dell'algoritmo nonché gli input e gli output, e sono mostrati nelle tabelle (7.2) e (7.3).

[1]VGT = Variazione della Geometria della Turbina

Input	Unità di misura	Descrizione
Speed	rmp	Velocità di rotazione del motore
p_rail	bar	Pressione di iniezione del combustibile (pressione nel rail)
SOI_pil	Deg before PMS	SOI dell'iniezione pilota
SOI_main	Deg before PMS	Inizio dell'iniezione della main
q_pil_1	mm^3/cyc/cyl	Quantità di combustibile iniettata durante la prima *pilot*
q_main	mm^3/cyc/cyl	Quantità di combustibile iniettata durante l'iniezione main
IMAT	K	Temperatura nel collettore di aspirazione
IMAP	bar	Pressione nel collettore di aspirazione
O_2	%	Concentrazione di Ossigeno
q_tot	mm^3/cyc/cyl	Quantità totale di combustibile iniettato
Air_mass	kg/cyc/cyl	Quantità di aria aspirata in camera
Inj.RAF	-	Rapporto tra quantità di aria e combustibile
EGR .mass	kg/cyc/cyl	Massa del flusso di gas esausti ricircolati (EGR)

Figura 7.2. Input di riferimento del modello

In questo caso studio avremo a disposizione due output e vedremo come l'unità di misura dell'output usato come riferimento nel codice non varierà i risultati poiché si passerà per alcune normalizzazioni.

Output	Unità di misura	Descrizione
Emi.NOx	ppm	Quantità misurata di NOx
Emi.NOx.mol	mol/mol	Quantità misurata in moli di NOx

Figura 7.3. Output di riferimento del modello

7.1 Risultati dataset completo

Come spiegato in precedenza, per questo motore Diesel da 11.0L di cilindrata si andrà a testare e validare il codice per due tipologie di Dataset *Bench strategy* (dati misurati a banco) o *ECU strategy* (dati stimati in centralina).
Perciò in questo capitolo useremo sia per l'addestramento che per il test la stessa tipologia di dati, o entrambi Banco-Banco o entrambi ECU-ECU, nel capitolo successivo invece si vedrà il caso studio dove si andrà a stimare il numero di inquinanti NO_X considerando i dati Banco-ECU.

```
TrainTest = int(input(
    'indicare dove si vuole fare training e testing rispettivamente:\n1 - Banco-
Banco\n2 - Banco-ECU\n3 - ECU-ECU\n'))
if TrainTest == 1:
    X = X[0:4711, :]
    y = y[0:4711]
    X_train, X_test, y_train, y_test = tts(X, y)
if TrainTest == 2:
    X_train = X[0:4711, :]
    y_train = y[0:4711]
    X_test = X[4711:, :]
    y_test = y[4711:]
if TrainTest == 3:
    X = X[4711:, :]
    y = y[4711:]
    X_train, X_test, y_train, y_test = tts(X, y)
```

Figura 7.4. Il codice di riferimento per la suddivisione dei dati

Quindi si ha che per configurazione:
Banco-Banco: righe che vanno dalla 0 alla 4711 per entrambi test e train.
Banco-ECU: righe che vanno dalla 0 alla 4711 per il training e dalla 4711 fino a fine file per test.
ECU-ECU: righe che vanno dalla 4711 a fine file per entrambi test e train.

Anche qui valuteremo i risultati in funzione delle tre tipologie di normalizzazione ed assieme ad esse considereremo anche le divisioni per quanto riguarda il train test split considerando come "test size" il 20%, 40%, 60% e 80% dei dati a disposizione, nella stessa maniera del caso precedente.
tts = 0.2 → 20% Train, 80% Test
tts = 0.4 → 40% Train, 60% Test
tts = 0.6 → 60% Train, 40% Test

tts = 0.8 → 80% Train, 20% Test

Prima di passare all'efficienza della predizione del calcolo del numero di inquinanti NO_X nel motore 11.0L, è opportuno precisare che lo studio dei vari casi tramite l'algoritmo GBRT è stato effettuato utilizzato esclusivamente il metodo di ottimizzazione degli iperparametri BayesSearchCV poiché già nel capitolo precedente si è dimostrato come il codice "*Gradient Boosting Regression Tree*" si sposa meglio con una scelta degli iperparametri tramite il teorema di Bayes.

7.1.1 Bench Strategy

Bench Strategy

La tabella (7.5) mostra i valori di r^2, coefficiente di determinazione, per il caso in cui sia per i valori di addestramento che quelli di test vengano usati dati derivanti dalle misurazioni effettuate su banco per il motore 11.0L a Diesel.
La tabella successiva (7.6) invece esprime i risultati sotto forma di rmse cioè l'errore quadratico medio.

Bench Strategy 11.0L Accuracy	Norm0	Norm1	Norm2
TTs=0.2	0,969	0,979	0,975
TTs=0.4	0,962	0,976	0,975
TTs=0.6	0,957	0,963	0,959
TTs=0.8	0,921	0,921	0,928

Figura 7.5. Risultati espressi in r^2 per l'algoritmo GBRT relativamente al dataset Banco–Banco con ottimizzazione BayesSearchCV

Bench Strategy 11.0L rmse	Norm0	Norm1	Norm2
TTs=0.2	0,019	0,018	0,021
TTs=0.4	0,028	0,020	0,021
TTs=0.6	0,027	0,025	0,027
TTs=0.8	0,036	0,038	0,038

Figura 7.6. Risultati espressi in *rmse* per l'algoritmo GBRT relativamente al dataset Banco-Banco con ottimizzazione BayesSearchCV

Immediatamente saltano all'occhio gli ottimi valori di predizione ottenuti in questo caso che garantiscono un efficienza molto elevata per la predizione delle emissioni di inquinante NO_X in funzione delle features, parametri motoristici, presentati nella tabella (7.2).
Come si osserva dalla tabella riportante il valore di accuratezza r^2 il valore massimo di affidabilità nella predizione è di 0,979, il quale è un valore supera notevolmente la soglia minima di accettazione. Questo valore sarà giustificabile in seguito degli annessi grafici che presentano i risultati predetti.
Prima di mostrare tali grafici però, è interessante osservare come per un $TTs = 0.8$ si arrivi a valori molto molto alti il che convalida l'elevata robustezza dell'algoritmo che potrà permettere un r^2 elevato pur andando a ridurre dal 80% al 20% i dati a disposizione per l'addestramento del codice GBRT.

L'andamento del grafico (7.7) conferma infatti gli elevati valori raggiunti poiché la concentrazione dei punti predetti si muove molto intorno alla retta rossa posta a 45 gradi che descrive la precisione unitaria del codice.
Interessante, ai fini dello studio, osservare come in questo grafico non si presenti nessun outlier che si allontana sostanzialmente dall'andamento lineare della regressione.

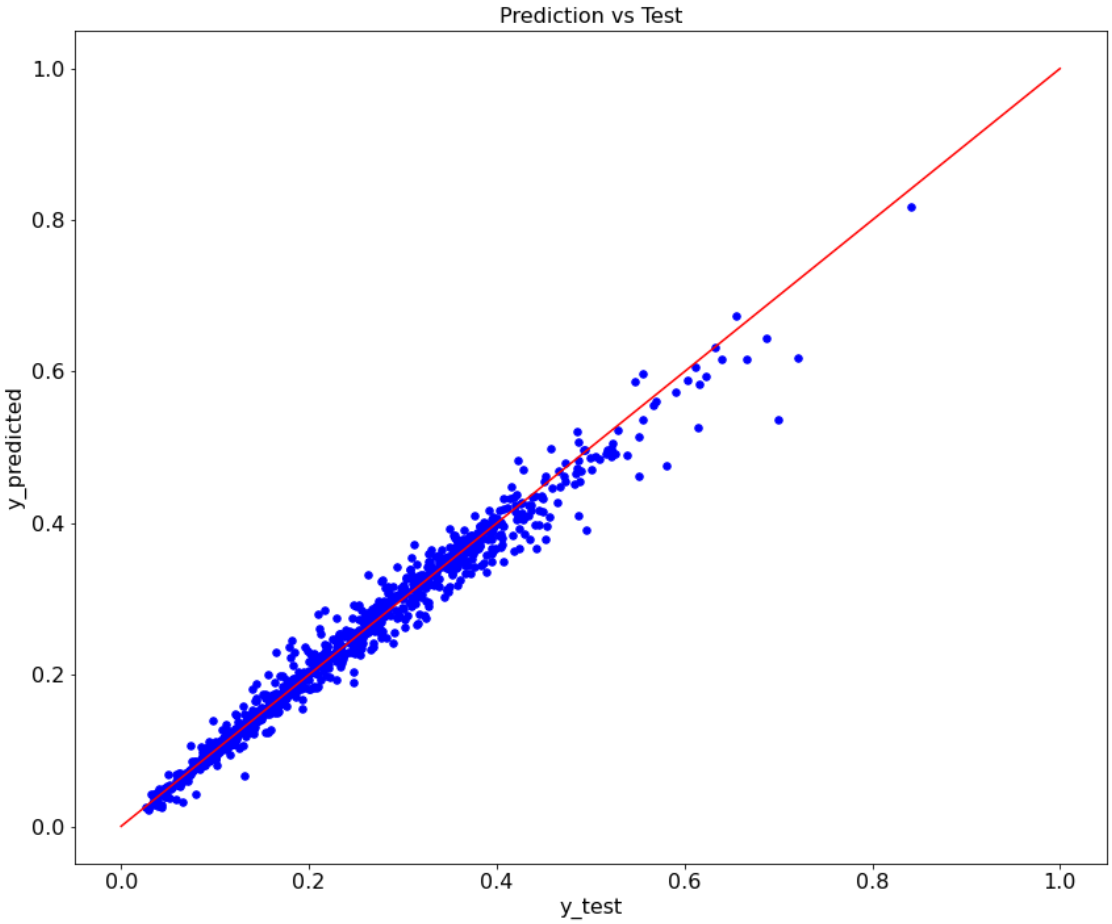


Figura 7.7. Grafico ”Prediction vs Test” ottenuto per motore 11.0L per il caso migliore ($TTs = 0.2$ con $Norm_1$)

Nel grafico (7.8) invece si osserva come i pallini arancioni, valori predetti, coincidano quasi perfettamente con i pallini blu, valori test. Questo a dimostrazione che nel caso $TTs = 0.2$ si toccano valori pienamente soddisfacenti.

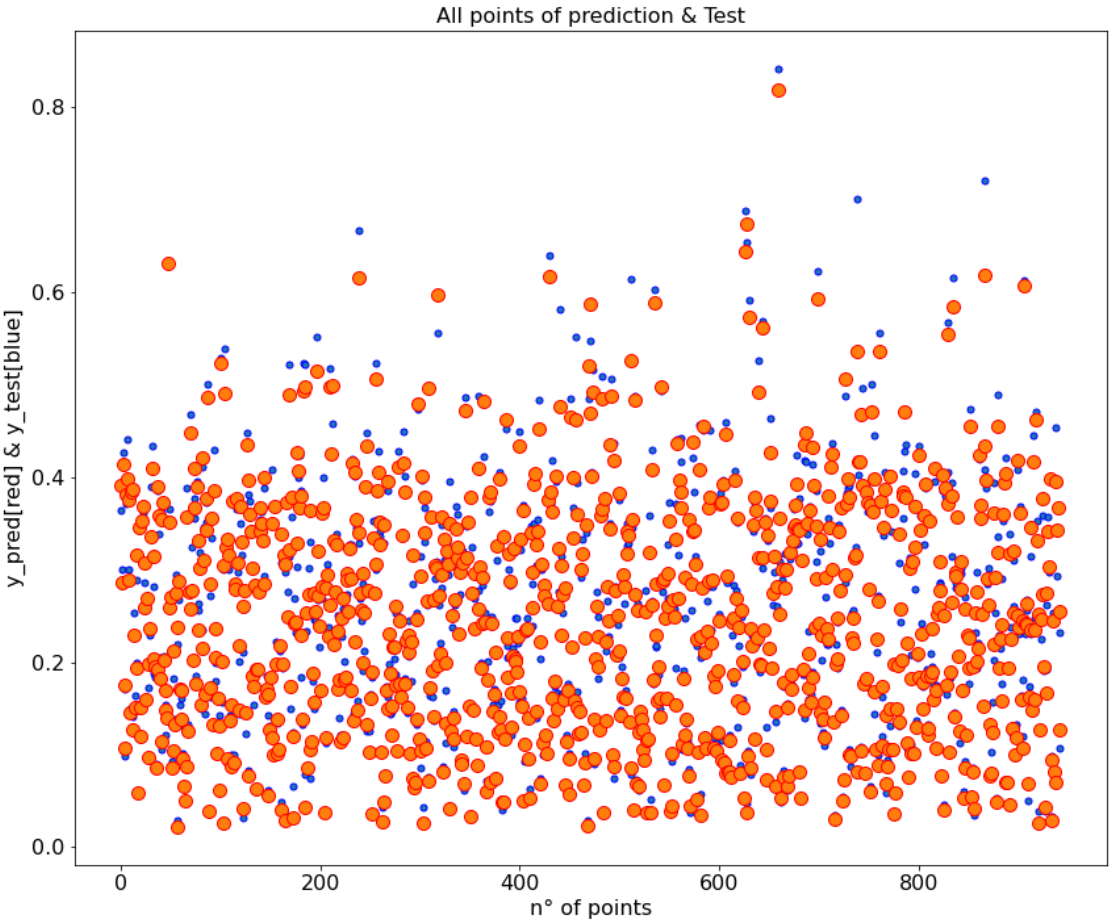


Figura 7.8. Grafico ”All points of prediction & Test” ottenuto per motore 11.0L per il caso migliore ($TTs = 0.2$ con $Norm_1$) dove tutti i punti della predizione vengono sovrapposti a quelli del test

Seppur interessante osservare i valori massimi ottenuti è indispensabile sottolineare l'accuratezza di precisione del codice quando gli sottoponiamo un quantitativo decisamente inferiore di dati per poter svolgere l'addestramento. Questo testimonia l'elevata robustezza dell'algoritmo GBRT nel calcolo di questa casistica.
Questi valori si riescono ad ottenere poiché alla radice il dataset utilizzato, rispetto il caso del motore 2.0L, è molto più numeroso e ricco di dati. Perciò quando vado ad abbassare la percentuale di TTs, sottoporrò lo stesso al codice un numero sufficiente di situazioni per addestrarsi.

Nel grafico (7.9) si presenta la peggior situazione di predizione in cui ci imbattiamo nello

studio ($TTs = 0.8$ *con* $Norm_0$) che risulta molto performante ed adatta per lo scopo.
Se andassi a confrontare questo grafico con il grafico (6.20), che rappresenta il peggior caso del motore precedente, vedremmo immediatamente come la dispersione dei punti di predizione non si allontanano dalla retta, ma si presentano nell'intorno della bisettrice con $m = 1$.

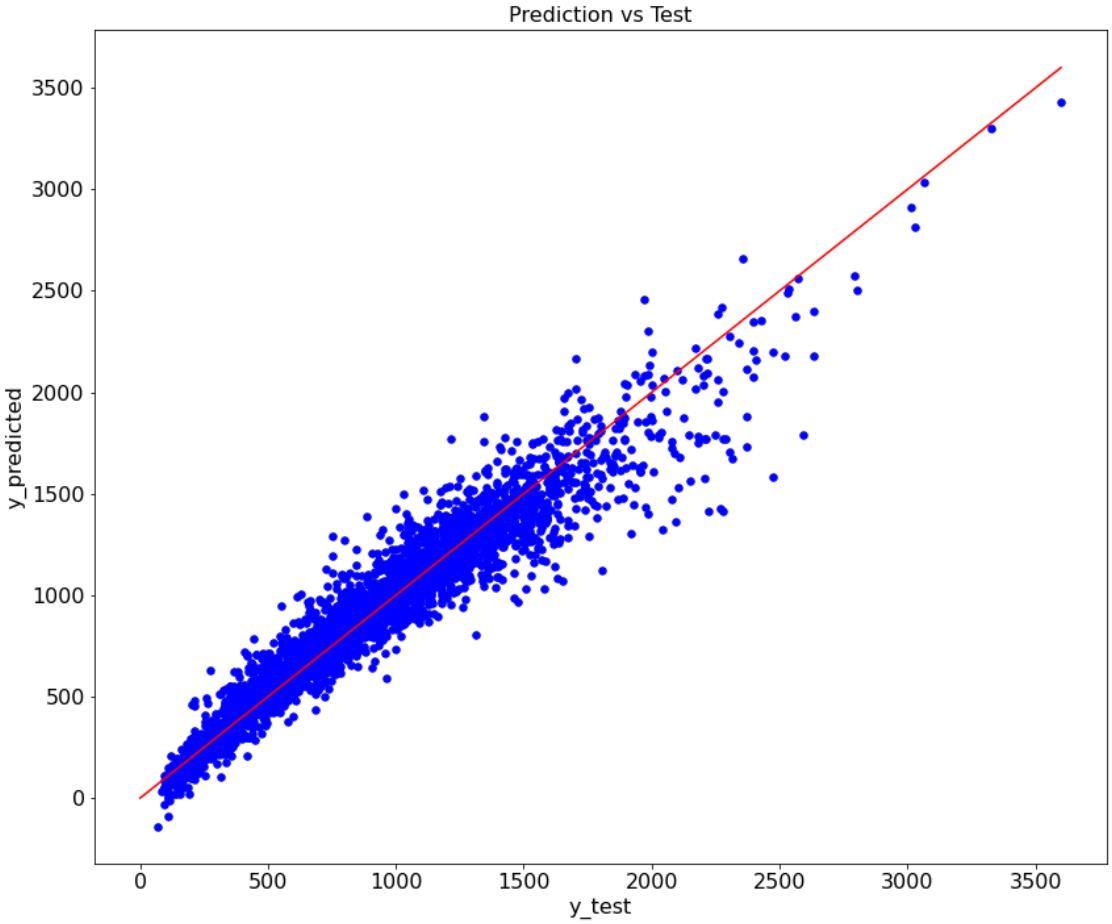


Figura 7.9. Grafico "Prediction vs Test" ottenuto per il peggior caso ($TTs = 0.8$ con $Norm_0$)

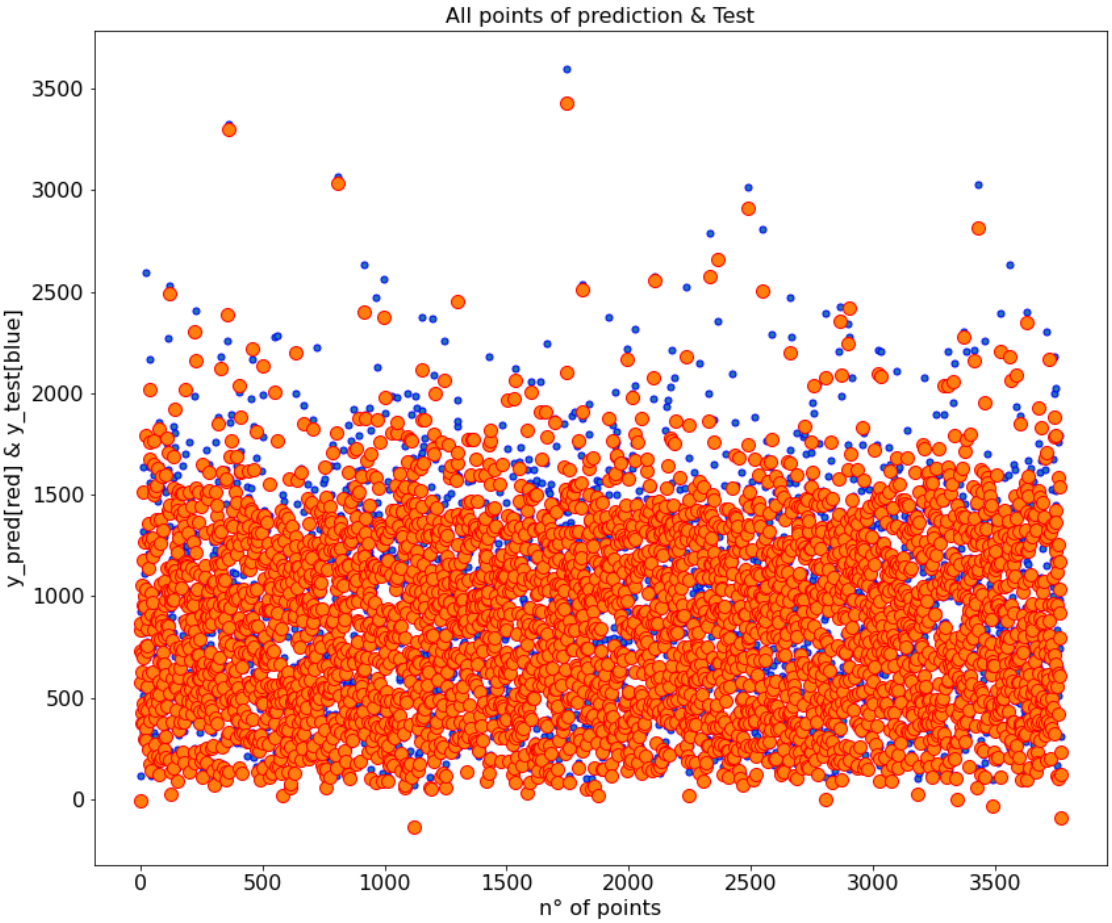


Figura 7.10. Grafico "All points of prediction & Test" ottenuto per il peggior caso ($TTs = 0.8$ con $Norm_0$) dove tutti i punti della predizione e del test vengono sovrapposti

Certamente si osserva una differenza tra il migliore ed il peggiore caso di predizione poiché la dispersione dei punti col crescere del TTs va ad aumentare il suo volume, ma per concludere si può stabilire con certezza che un numero elevato di dati, e dunque un alta size del dataset, risultano essere quindi necessari per eseguire un apprendimento dell'algoritmo in maniera adeguata per poter ottenere alti valori di r^2 sia con tanti e sia con pochi dati a disposizione per l'addestramento.

I valori degli iperparametri scelti per il codice GBRT sono stati ottenuti con l'ottimizzazione BayesSearch dopo Cross-Validation e sono presentati nella seguente tabella (7.11) per il miglior caso di predizione $TTs = 0.2$ con $Norm_1$:

GBRT_Best_Parameters	Valori
learning_rate	0,129978528
max_depth	3
n_estimators	1000
subsample	0,6
validation_fraction	1

Figura 7.11. *Best Hyperparameters*

Infine, in funzione dei valori di r^2 e del train test split, si va ad analizzare e valutare l'andamento per ogni tipo di normalizzazione utilizzata nel codice di Machine Learning. Rispetto al grafico Excel registrato per il motore Diesel da 2.0L di cilindrata vediamo l'asse y del grafico molto più zoomato poiché i valori numeri delle accuratezze sono molto più elevati; l'asse x invece descriverà la suddivisione del TTs che sarà la medesima.
L'andamento mostrato nel grafico (7.12) in questo caso è molto meno inclinato verso il basso (bisognerebbe osservarlo a parità di zoom), ma andrà giustamente comunque a diminuire poiché fornisco man mano minor numero di dati per l'addestramento.

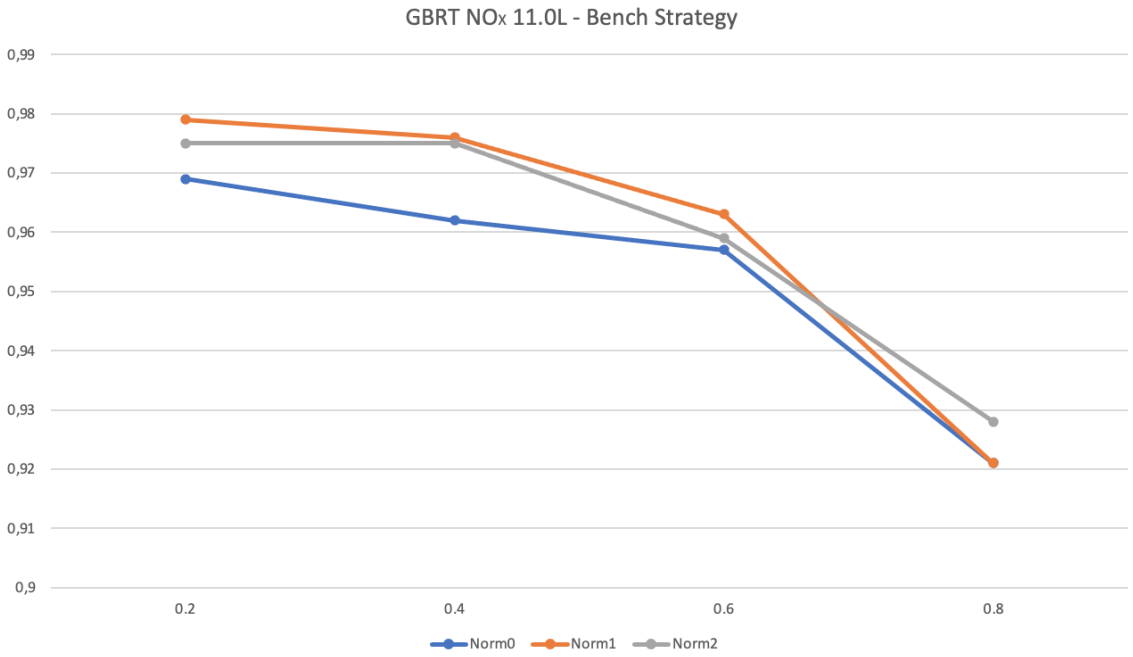


Figura 7.12. Andamento delle tre normalizzazioni in funzione del r^2 e dei 4 TTs per il motore 11.0L nel caso dell'inquinante NO_X con configurazione Banco-Banco

7.1.2 ECU Strategy

ECU Strategy

Analogamente a quanto svolto per il dataset derivante dalle misurazioni a banco, con il dataset ottenuto dalle valutazioni in ECU si riportano tutti i risultati delle simulazioni nelle tabelle (7.13) e (7.14), con le stesse dimensioni dei dataset di training e test del caso precedente.
Come spiegato precedentemente, considereremo esclusivamente l'ottimizzazione BayesSearchCV per individuare gli iperparametri migliori.

ECU Strategy 11.0L Accuracy	Norm0	Norm1	Norm2
TTs=0.2	0,949	0,960	0,962
TTs=0.4	0,939	0,942	0,947
TTs=0.6	0,917	0,919	0,925
TTs=0.8	0,892	0,897	0,917

Figura 7.13. Risultati espressi in r^2 per l'algoritmo GBRT relativamente al dataset ECU-ECU per il motore 11.0L

ECU Strategy 11.0L rmse	Norm0	Norm1	Norm2
TTs=0.2	0,026	0,025	0,025
TTs=0.4	0,032	0,031	0,030
TTs=0.6	0,038	0,341	0,340
TTs=0.8	0,044	0,042	0,039

Figura 7.14. Risultati espressi in *rmse* per l'algoritmo GBRT relativamente al dataset ECU-ECU per il motore 11.0L

I valori degli iperparametri scelti dal codice GBRT ottenuti trameite l'ottimizzazione BayesSearch dopo Cross-Validation sono presentati nella seguente tabella (7.15) per il miglior caso di predizione che questa volta coincide con $TTs = 0.2$ con $Norm_2$:

GBRT_Best_Parameters	Valori
learning_rate	0,210124394
max_depth	3
n_estimators	1000
subsample	0,9
validation_fraction	0,800203475

Figura 7.15. *Best Hyperparameters*

Confrontando le 2 tabelle che riportano le accuratezze per il caso Banco-Banco e per il caso ECU-ECU vediamo che in entrambi i casi si ottengono alti valori di r^2 e bassi di $rmse$ che indicano una perfetta convergenza sul train set.
L'addestramento prevede un processo iterativo che si finalizza e si ottiene convergenza quando:
-L'output della loss function ha andamento decrescente rispetto al numero di iterazioni, si minimizza l'errore $rmse$.
-L'accuratezza ha andamento crescente rispetto al numero di iterazioni, si massimizza il valore del coefficiente di determinazione r^2 che indica l'accuratezza in un problema di regressione.
Anche da un rapido confronto fra i valori che assumo gli iperparametri nei due differenti casi studio non si notano sostanziali differenze.

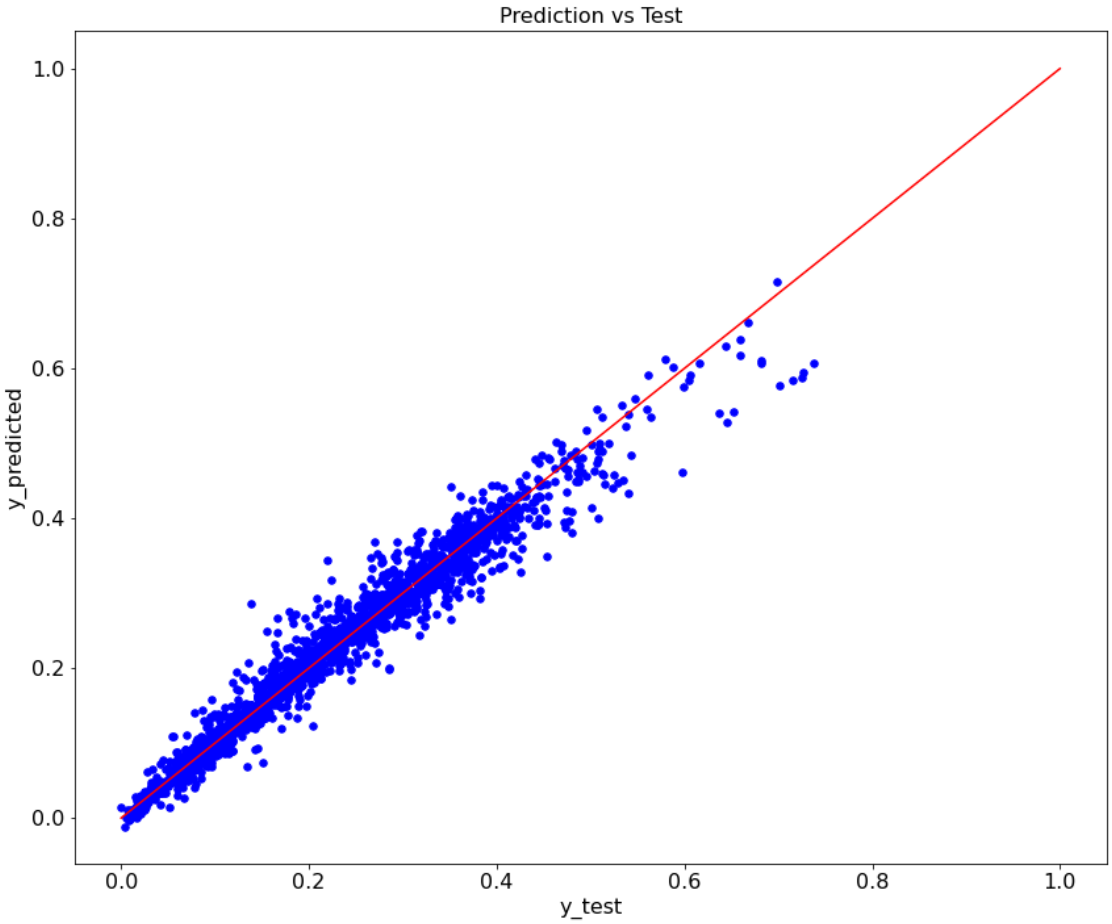


Figura 7.16. Grafico ”Prediction vs Test” ottenuto per il miglior caso ($TTs = 0.2$ con $Norm_2$)

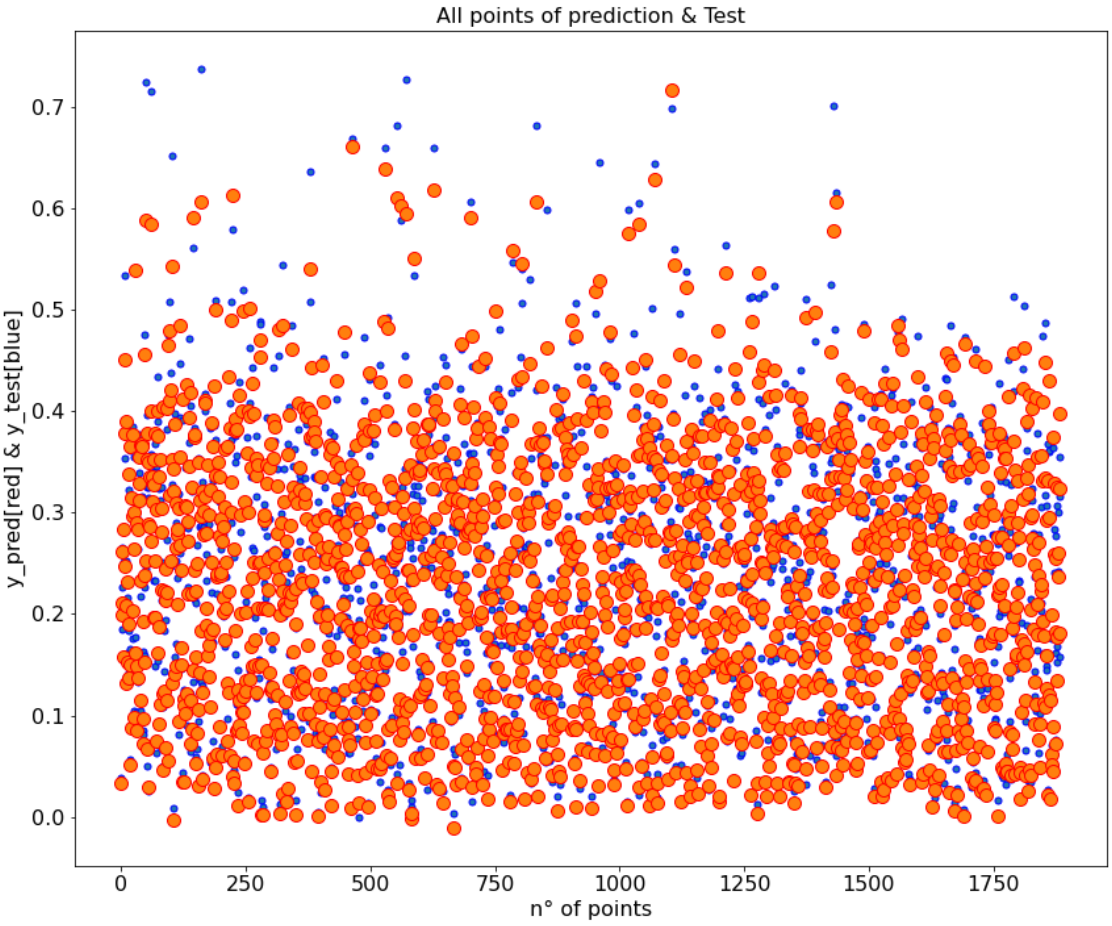


Figura 7.17. Grafico “All points of prediction & Test” ottenuto per il miglior caso ($TTs = 0.2$ con $Norm_2$) dove tutti i punti della predizione e del test vengono sovrapposti

Infine anche in questo caso osserviamo gli andamenti (figura (7.18)) delle tre normalizzazioni in funzione del r^2 e dei 4 TTs per il motore 11.0L nel caso dell'inquinante NO_X con configurazione ECU-ECU.
Si osserva anche qui un notevole zoom sull'asse y dovuto all'alto valore delle accuratezze ottenute anche in questo caso.
Gli andamenti sono molto lineari perché anche con $TTs = 0.8$ che dovrebbe generare risultati bassi si riesce ad ottenere accuratezze molto elevate su valori di circa 0,9.

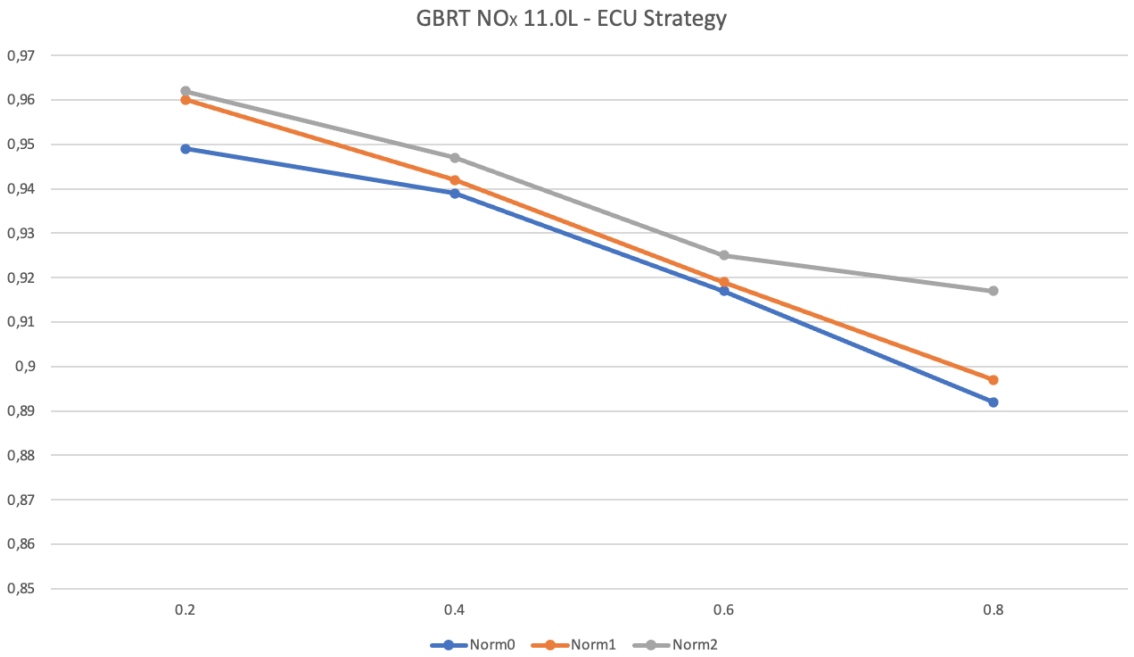


Figura 7.18. Andamento delle tre normalizzazioni in funzione del r^2 e dei 4 TTs per il motore 11.0L nel caso dell'inquinante NO_X con configurazione ECU-ECU

Dal confronto finale tra la tabella Banco-Banco (7.5) (dove sia per il train che per il test si usano dati misurati su banco) e la tabella ECU-ECU (7.13) (dove sia per il train che per il test si usano dati stimati in centralina) e dal confronto tra i corrispettivi grafici indicanti l'accuratezza della predizione si osserva che l'accuratezza stimata tramite dati derivanti dalla centralina (ECU) perde alcuni punti percentuali facendo sì che il codice di Machine Learning usante l'algoritmo GBRT performi leggermente meno con questa tipologia di dati.

7.2 Risultati dataset ridotto

7.2.1 Dataset Feature Importance

In questo capitolo presentiamo un ulteriori caso studio, svolto sia per la configurazione Banco-Banco e sia per la configurazione ECU-ECU, che analizza maggiormente, tramite l'algoritmo in questione, il dataset di dati che si hanno per il calcolo predittivo dell'emissione inquinante di NO_X.
È interessante analizzare il TTs che porta risultati migliori cioè il $TTs = 0.2$ e vedere la dipendenza di esso dalle features maggiormente importanti tramite *features importance*.
Generalmente in un algoritmo si pensa di sfruttare tutte le features che si hanno a disposizione, ma non è sempre conveniente.
Nel caso in esame si usa il modello GBRT stesso per definire l'importanza di ciascuna feature e tramite due grafici capire quali parametri motoristici contribuiscono maggiormente a generare l'emissione inquinante.
Si metteno a schermo i grafici (7.19) e (7.20)[2] dove si mostrano in ordine le *features più importanti* quelle cioè che apportano maggiore informazione alla risoluzione del problema.

Si osserverà da prima la condizione Banco-Banco e successivamente ECU-ECU.

Bench Strategy

Nella prima configurazione abbiamo il grafico (7.19) mostra una classifica delle features più influenti. La seguente funzione permette di costruire un "horizontal bar plot" con i valori di importanza delle features.
Questi valori vengono calcolati durante l'addestramento per ogni TTs, indipendentemente dal tipo di normalizzazione, in modo da mostrare le 7 features che descrivono meglio la situazione.
Sarà dunque il codice stesso ad evidenziare le features più importanti.

[2]La seguente tipologia di grafici è stata valutata per la condizione di $TTs = 0.2$ sia per configurazione B-B che per E-E

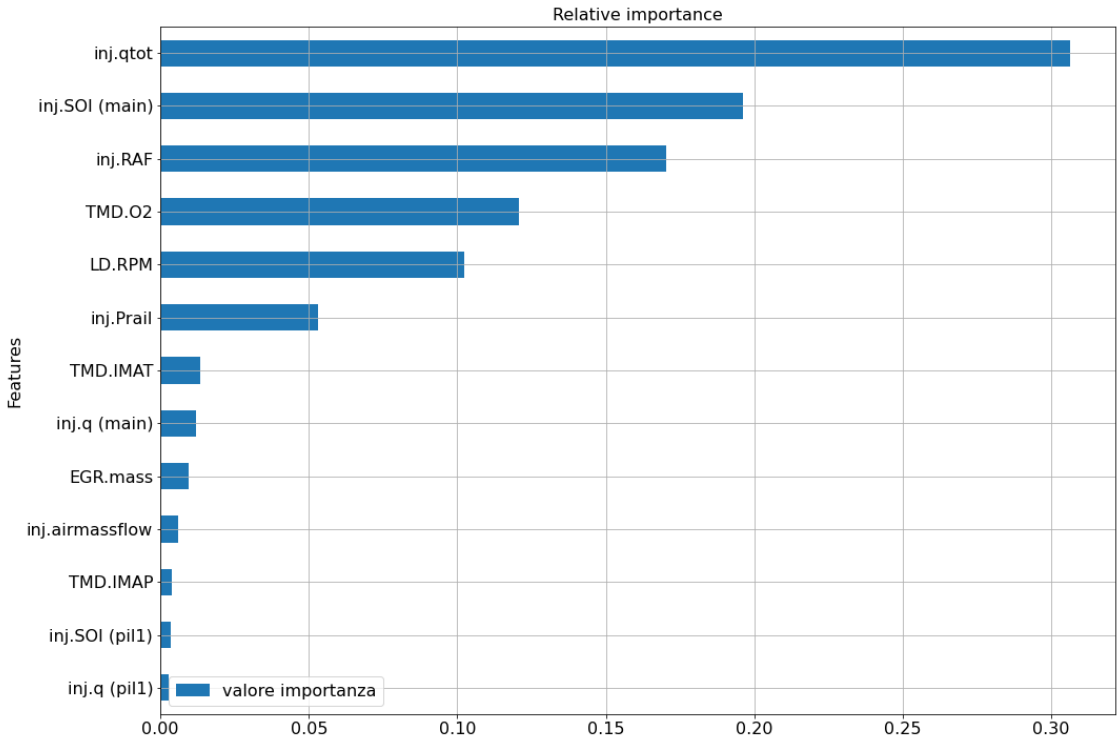


Figura 7.19. Istogramma che descrive l’importanza relativa di ogni singola feauteres per la configurazione Banco-Banco

Nel secondo grafico che descrive la features importance (7.20) ottengo sempre tramite il codice principale una classifica delle features più influenti con un valore cumulativo.

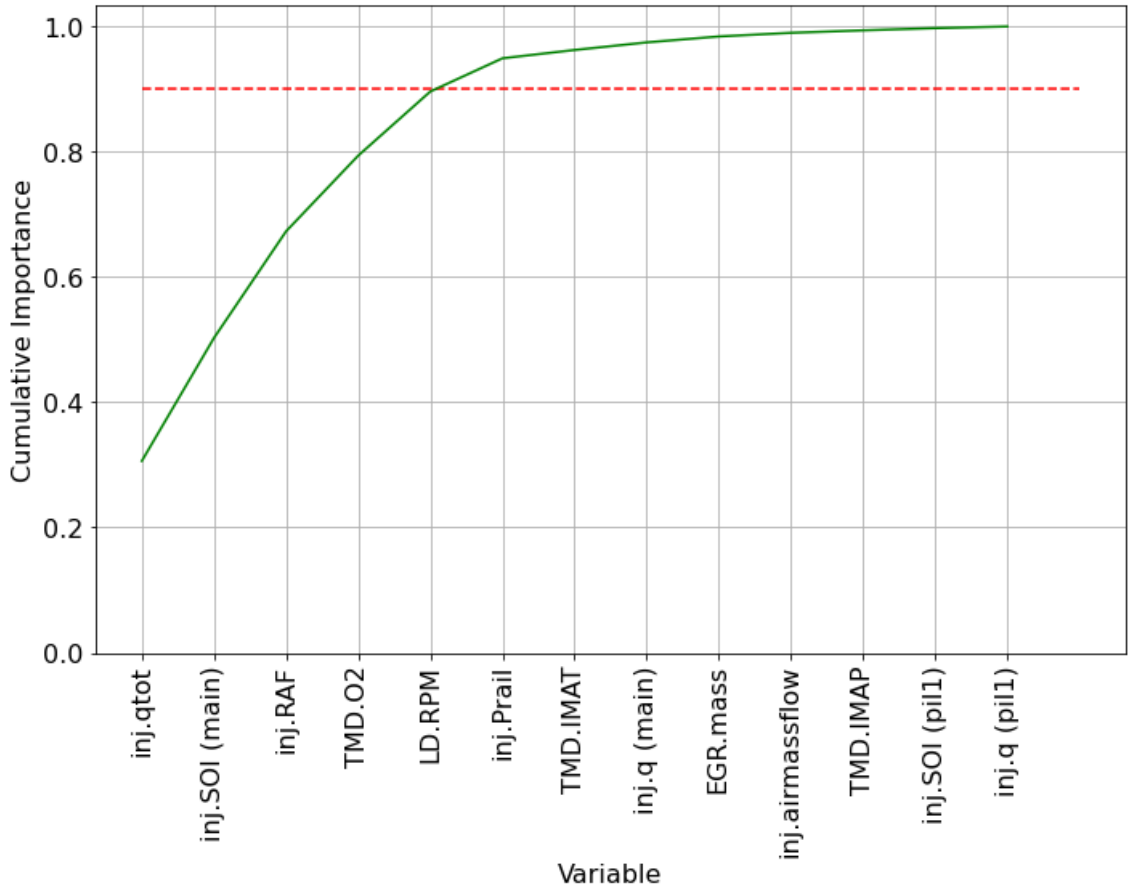


Figura 7.20. Importanza cumulativa delle features utilizzate per la configurazione Banco-Banco

La funzione $'cum\_imp\_fig'$, come mostrato nel capitolo 3 tramite il pezzo di codice prende in ingresso i valori cumulativi di importanza, la lista contenente i nomi delle features e l'ordine di importanza per costruire il grafico.
La figura contiene in x le features; in y per ogni feature vengono segnati i valori della rispettiva importanza cumulata e questi vengono uniti dalla linea verde; la linea rossa rappresenta è segnata al punto dove l'importanza accumulata è pari al 90%.
Note le features più importanti si prova ora a riallenare il modello senza utilizzare le feature meno importanti. Per questo si è arbitrariamente deciso di prendere solo le features che danno un valore cumulato di importanza maggiore del 90% ottenendo così i valori del dataset ridotto alle 7 features più importanti presenti nella tabella (7.21).

Dateset ridotto B-B TTs=0.2	Norm0	Norm1	Norm2
Accuracy	0,965	0,975	0,971
rmse	0,026	0,020	0,022

Figura 7.21. Risultati espressi in r^2 e $rmse$ con dataset ridotto considerando features importance per la configurazione Banco-Banco

I valori numerici registrati in tabella rispondono al calcolo predittivo dell'inquinante NO_X nel caso in cui si usasse un $TTs = 0.2$, e dunque l'80% dei dati del dataset utilizzati per l'addestramento, con le tre diverse tipologie di normalizzazione.
In tal modo si è dimostrata una certa corrispondenza tra l'algoritmo (che si dissocia totalmente dall'aspetto fenomenologico del problema) e il modello utilizzato (GBRT) che individua degli input che hanno un peso maggiore di altri.
I valori numerici, mediati su 3 prove effettuate, restano molto alti proprio come si aveva nel caso studio in cui disponevamo di tutte le features del dataset.
Questi risultati, inoltre sottolineano le sette features (input del problema), in ordine decrescente, che maggiormente contribuiscono per la predizione.

1) q_tot: quantità di combustibile totale
2) SOI_main: inizio dell'iniezione della main
3) inj_RAF: rapporto tra quantità di aria e combustibile
4) O_2: concentrazione di Ossigeno
5) $Speed$: velocità di rotazione del motore
6) p_rail: pressione di iniezione del combustibile
7) $IMAT$: temperatura nel collettore di aspirazione

ECU Strategy

La seconda configurazione che verrà studiata col dataset ridotto è la configurazione ECU-ECU. In questo caso di studio visto in precedenza si andrà a considerare le 7 feauteres più influenti nel caso di studio $TTs = 0.2$ il quale offre valori di r^2 maggiori.
Le 7 features importance non varieranno dal tipo di normalizzazione utilizzata infatti andremo a determinare i valori di accuratezza usando le stesse 7 features per i 3 tipi di

normalizzazione.
Sia il grafico (7.22) che (7.23) hanno gli stessi significati e scopi di quelli visti nel caso precedente per la configurazione Banco-Banco (Bench Strategy).

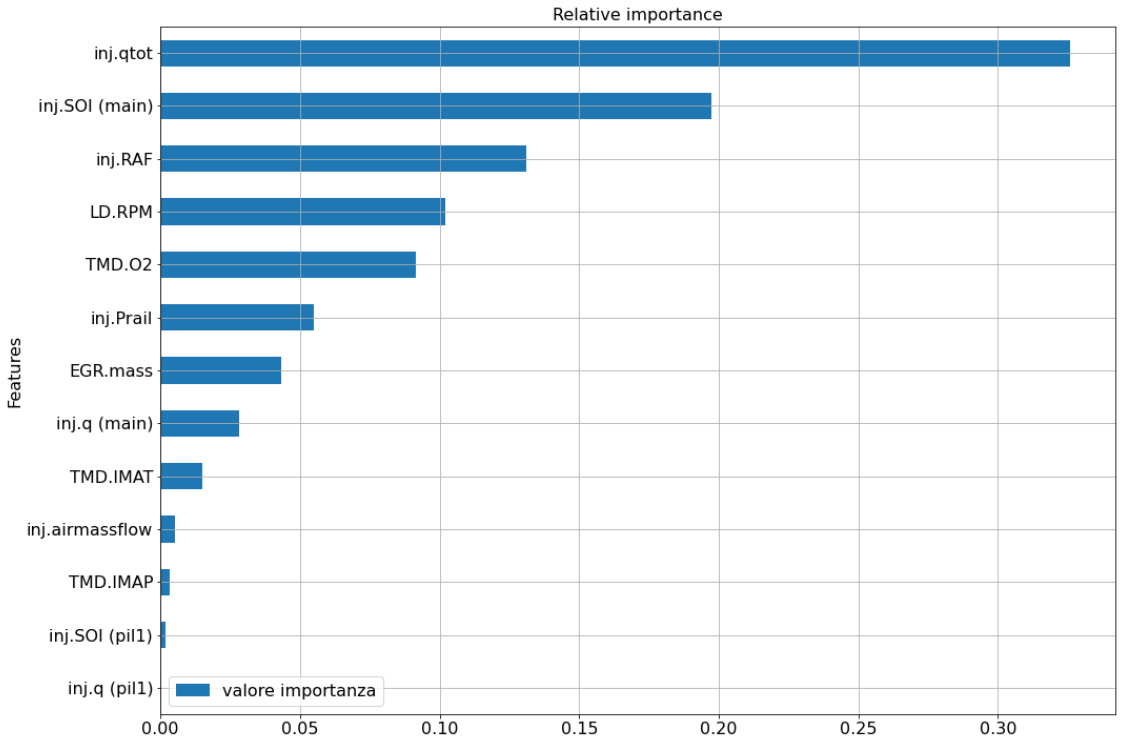


Figura 7.22. Istogramma che descrive l'importanza relativa di ogni singola feauteres per la configurazione ECU-ECU

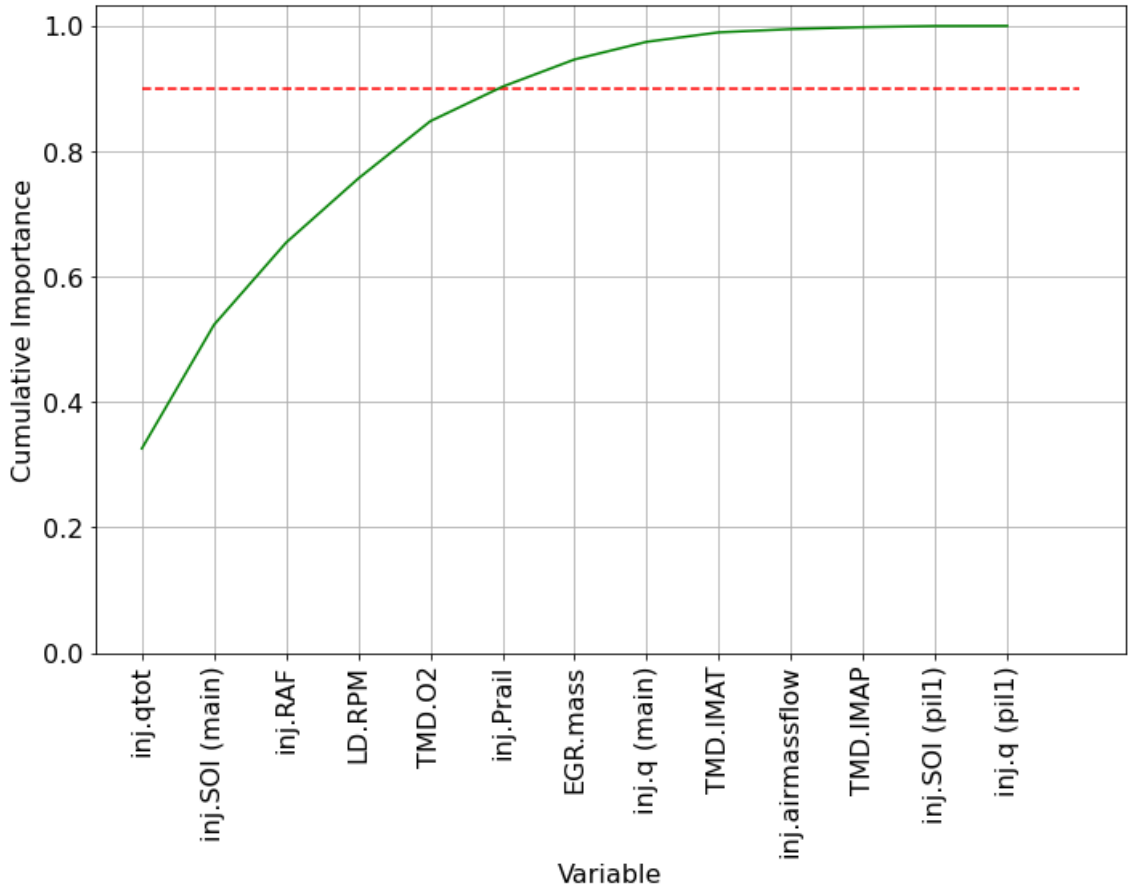


Figura 7.23. Importanza cumulativa delle features utilizzate per la configurazione ECU-ECU

Questi andamenti hanno selezionato parametri motoristici differenti dal caso precedente che hanno portato a valori di r^2 e $rmse$ leggermente minori rispetto a prima.
Come visto nella configurazione del dataset completo, però, la strategia ECU portava ad una situazione leggermente minore a quella del Banco e lo stesso rapporto si ritrova anche utilizzando una dataset ridotto.

Dateset ridotto E-E TTs=0.2	Norm0	Norm1	Norm2
Accuracy	0,945	0,955	0,960
rmse	0,028	0,027	0,026

Figura 7.24. Risultati espressi in r^2 e *rmse* con dataset ridotto considerando features importance per la configurazione ECU-ECU

Tutto questo sottolineano altre sette features (input del problema), in ordine decrescente, non precisamente identiche al caso precedente mostrando una dipendenza diversa dei parametri del motore dall'inquinante NO_X trovato con la configurazione ECU-ECU (*ECU Strategy*).
I 7 parametri più influenti evidenziati questa volta sono:

1) q_tot: quantità di combustibile totale
2) SOI_main: inizio dell'iniezione della main
3) inj_RAF: rapporto tra quantità di aria e combustibile
4) *Speed*: velocità di rotazione del motore
5) O_2: concentrazione di Ossigeno
6) p_rail: pressione di iniezione del combustibile
7) EGR_mass: massa del flusso di gas esausti ricircolati

Per concludere si osserva che i parametri più influenti sia nella Bench strategy che in quella ECU restano pressoché gli stessi ad eccezione del settimo che nella configurazione ECU-ECU diventa la massa del flusso di gas esausti ricircolati. Si può anche annotare una variazione di posizione d'importanza tra il parametro *Speed* e quello O_2 che nella configurazione ridotta Banco-Banco occupavano posizioni inverse.

7.3 Risultati Banco-ECU

Sono stati effettuati ulteriori studi per il motore 11.0 litri di cilindrata nel caso dell'inquinante NO_X.
Per queste prove si attinge da dati provenienti da quelli misurati su banco e da dati stimati

dalla centralina del motore.
Lo studio è stato composto in questa maniera:
- Training su dati da banco
- Testing sui dati ricavati da centralina (ECU)
Per questa casistica non si è ritenuto necessario testare l'algoritmo su diversi valori di TTs.

GBRT NOx Bench-ECU	Norm0	Norm1	Norm2
Accuracy	0,961	0,965	0,966
rmse	0,026	0,024	0,025

Figura 7.25. Risultati espressi in r^2 e $rmse$ per la configurazione Bench-ECU

Le prove banco-ECU sono molto utili perché rappresentano l'idea più vicina allo scopo finale degli algoritmi di Machine Learning nel campo automotive, ossia l'implementazione del calcolo predittivo in centralina.
Infatti effettuando il training su banco il modello si adatta in maniera più efficace alla fenomenologia del caso, in quanto i dati da banco sono generalmente più precisi di quelli da centralina.
Successivamente si prosegue con il testing direttamente sui dati ECU ricavati con opportuni sensori.

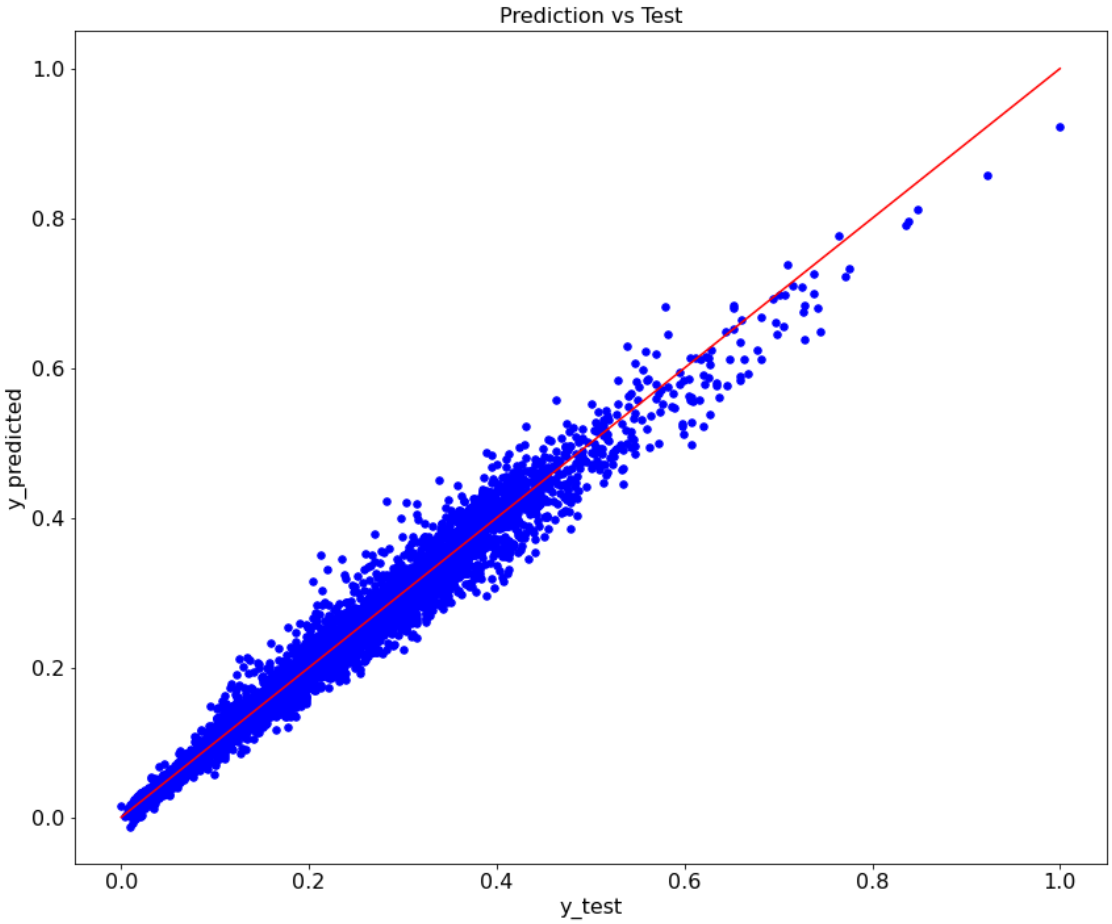


Figura 7.26. Grafico ”Prediction vs Test” nel caso migliore per la configurazione Bench-ECU

Si arriva a notare come i valori di accuratezza presenti in tabella (7.25) e rappresentati nel grafico (7.26) sono più bassi di quelli ottenuti in precedenza, ma non di troppo, in quanto prima si allena tenendo in considerazione la fisica del problema e poi si testa sui dati che potrebbero essere soggetti a imprecisione dovuta ai sensori presenti sul motore Cursor da 11.0L.
L’elevato quantitativo di dati a disposizione però consente un ottimo utilizzo dell’algoritmo GBRT per il calcolo predittivo dell’inquinante NO_X.

Capitolo 8

Modello predittivo per motore 11.0L - inquinante “Soot”

In questo capito si analizzerà la configurazione per il motore Diesel da 11.0 litri di cilindrata per lo studio predittivo dell'inquinante Soot nel caso Banco-Banco ed ECU-ECU. Solo in seguito affronteremo altri casi studio trattando dataset ridotti alle features più importanti oppure considerando per l'addestramento i dati derivanti da banco e per il test i dati stimati in ECU.
Si parte con l'analizzare i dati a disposizione al fine di trovare la soluzione al nostro problema utilizzando gli stessi parametri e le stesse scelte per il codice. Di seguito si considera il caso motore 11.0L, inquinante soot e configurazione banco-banco.

Da subito adattando lo stesso codice Gradient Boosting Regression trees di Machine Learning si notano sostanziali differenze rispetto i risultati ottenuti nella casistica di predizione dell'inquinante NO_X per lo stesso motore.
Si riproducono le stesse prove con le identiche combinazioni di normalizzazioni e di train test split (TTs).
-Normalizzazione dei dati
1. Nessuna normalizzazione (Norm 0)
2. Norma 1 (Norm 1)
3. Norma 2 (Norm 2)
-“Train/Test split size”
1. 20% Train, 80% Test ($TTs = 0.2$)
2. 40% Train, 60% Test ($TTs = 0.4$)
3. 60% Train, 40% Test ($TTs = 0.6$)

4. 80% Train, 20% Test ($TTs = 0.8$)
Così facendo sono state ottenute un totale di 12 valori per l'inquinante *soot* per il motore 11.0L.
Ogni prova è stata ripetuta tre volte e i risultati sono stati mediati al fine di evitare errori di tipo probabilistico e/o Overfitting/Underfitting dell'algoritmo.

Il dataset in questione si compone di parametri motoristici derivanti dalle prove di banco e stimati dalla centralina (ECU) installata sul motore e si divide in due parti, input, tabella (8.1) ed output, tabella (8.2). Per questo studio, come vedremo, infatti si è valutato lo studio su entrambi gli output per determinare quale possa portare un vantaggio all'intero processo di predizione.
Anche in questo caso analizzeremo la predizione con la Bench Strategy e con l'ECU Stategy poiché il dataset dispone di misure effettuate a banco e misure stimate in centralina, come già anticipato.

Input	Unità di misura	Descrizione
Speed	rmp	Velocità di rotazione del motore
p_rail	bar	Pressione di iniezione del combustibile (pressione nel rail)
SOI_pil	Deg before PMS	SOI dell'iniezione pilota
SOI_main	Deg before PMS	Inizio dell'iniezione della main
q_pil_1	mm^3/cyc/cyl	Quantità di combustibile iniettata durante la prima *pilot*
q_main	mm^3/cyc/cyl	Quantità di combustibile iniettata durante l'iniezione main
IMAT	K	Temperatura nel collettore di aspirazione
IMAP	bar	Pressione nel collettore di aspirazione
EMAP	bar	Pressione nel condotto di scarico
O_2	%	Concentrazione di Ossigeno
q_tot	mm^3/cyc/cyl	Quantità totale di combustibile iniettato
Air_mass	kg/cyc/cyl	Quantità di aria aspirata in camera
Inj.RAF	-	Rapporto tra quantità di aria e combustibile
EGR .mass	kg/cyc/cyl	Massa del flusso di gas esausti ricircolati (EGR)

Figura 8.1. Input - parametri motoristici di riferimento

Output	Unità di misura	Descrizione
Emi.soot	mg/cyc/cyl	Quantità misurata di soot
Emi.soot_g_h	g/h	Quantità misurata in g/h di soot

Figura 8.2. Output - quantità dell'inquinante PM

Analizziamo ora i risultati conseguenti alle simulazioni effettuate per il dataset sopra elencato con il medesimo codice GBRT, senza apportare alcuna modifica, valutando le performance in termini di accuratezza (r^2) ed errore quadratico medio ($rmse$) relativamente a diversi valori di train test split adottati.

Procedendo l'analisi, con il codice visto finora, si ottengono valori di r^2 estremamente bassi ed inaccettabili rispetto ai valori ottenuti per le altre casistiche sia adottando un ottimizzazione degli iperparametri col metodo BayesSearchCV (immagine (8.3)) e sia con il metodo GridSearchCV (immagine (8.4)). Nelle figure vengono mostrati gli output del codice che restituisce valori inaccettabile per il caso $TTs = 0.2$ con $Norm_1$ che dovrebbe coincidere con il caso migliore poiché disponiamo di più dati possibile per l'addestramento.

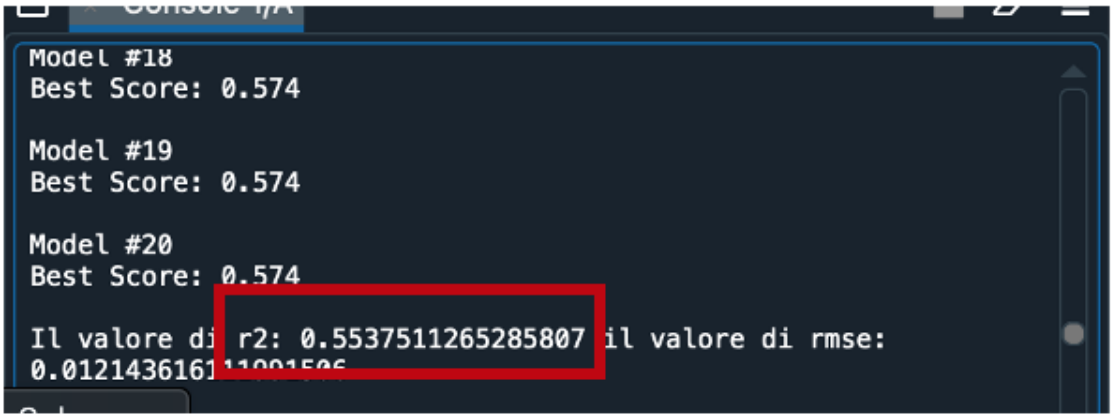


Figura 8.3. Predizione motore 11.0L per inquinante soot col metodo BayesSearchCV

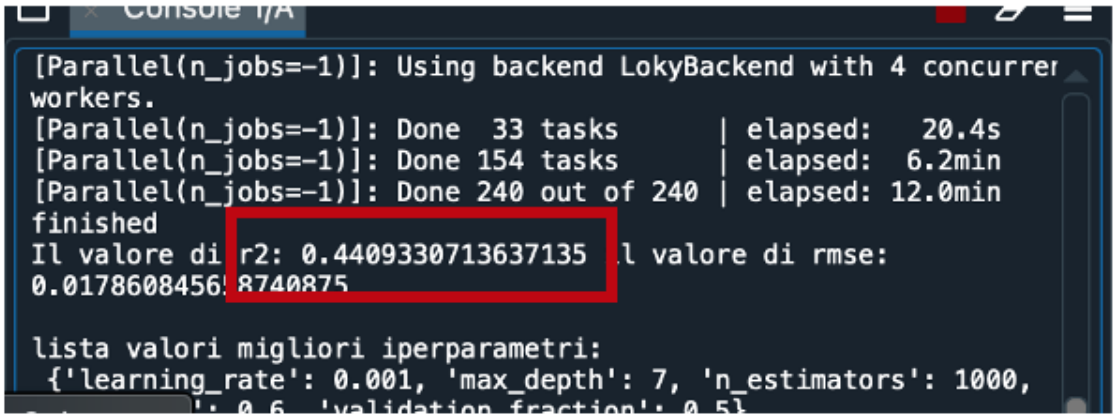


Figura 8.4. Predizione motore 11.0L per inquinante soot col metodo GridSearchCV

Il perimetro del rettangolo evidenziato in rosso racchiude per entrambi i grafici i valori di accuratezza espressa in r^2 per lo stesso caso studio, cioè l'inquinante soot nel motore

11.0L.
Nonostante i valori numeri siano inadatti allo scopo e quindi non considerabili a tal fine, sottolineano nuovamente un dettaglio non trascurabile, cioè che la ricerca dei valori degli iperparametri nell'algoritmo GBRT col metodo BayesSearchCV ottimizza il fine ultimo di questo studio. Infatti a conferma di questa tesi possiamo vedere osservare come i valori mediati si aggirano intorno al 0.55 per il BayesSearchCV e solo 0.44 per il GridSearchCV. Ricordo però che questi valori corrispondono al caso più ricco di dati per l'addestramento e che da questo valori in poi si andrebbe a diminuire l'accuratezza rendendo ancor più inadatta questa soluzione per lo studio del calcolo di inquinanti mediante la predizione con il codice GBRT.

Si sceglie a questo punto di approfondire i motivi di questa variazione eclatante di risultati pur utilizzando lo stesso codice che precedentemente aveva portato a valori ottimi sia per il motore 2.0L che per quello 11.0L.
Viene calcolato a tal proposito la cross-validation (analisi (8.5)) usando il GBRT base e quello che risulta è che ad ogni iterazione r^2 sul training è tipo 0.92/0.93 e sul validation molto basso attorno allo 0.3/0.4, come evidenziato in figura.
Quest'analisi è stata condotta sia sul parametro *emi.soot* e sia sul parametro *emi.soot_g_h*.

```
Colonna output: emi.soot
[ 1.52484859  0.70095829  1.53386308 -1.51770084 -1.45420289 -1.51509612
  0.72225338  0.95312323  0.14678636  0.73496903  2.02763176  1.15217417
 -0.51457367 -0.98031827]
CV test scores: [ 0.3403346  -1.62119409 -0.22663654  0.31630909]
CV train scores: [0.92049296 0.93334162 0.93586649 0.89159598]
Test-set r2 0.6869485503099013
```

Figura 8.5. Predizione motore 11.0L per inquinante soot col metodo GridSearchCV

Siccome per ogni iterazione del cross-validation r^2 sul training è alto sembra un problema legato principalmente all'overfitting motivo per il quale vengono provate diverse opzioni come ad esempio la regolarizzazione sul GBRT per semplificare l'algoritmo o un diverso uso di iperparametri, ma entrambe non funzionano portando risultati di accuratenza addirittura più bassi di quelli ottenuti.
Risulta perciò necessario fare un'analisi sui dati componenti il dataset del caso in questione (soot).

Analizziamo anzitutto le **variabili target**:

```
sns.histplot(data['emiE.soot_g_h'])
plt.show()
sns.histplot(data['emi.soot'])
plt.show()
```

Figura 8.6. Codice per ottenere le distribuzione dei dati

Tramite il codice (8.6) possiamo valutare la distribuzione dei dati all'interno del dataset soot per il motore 11.0L.

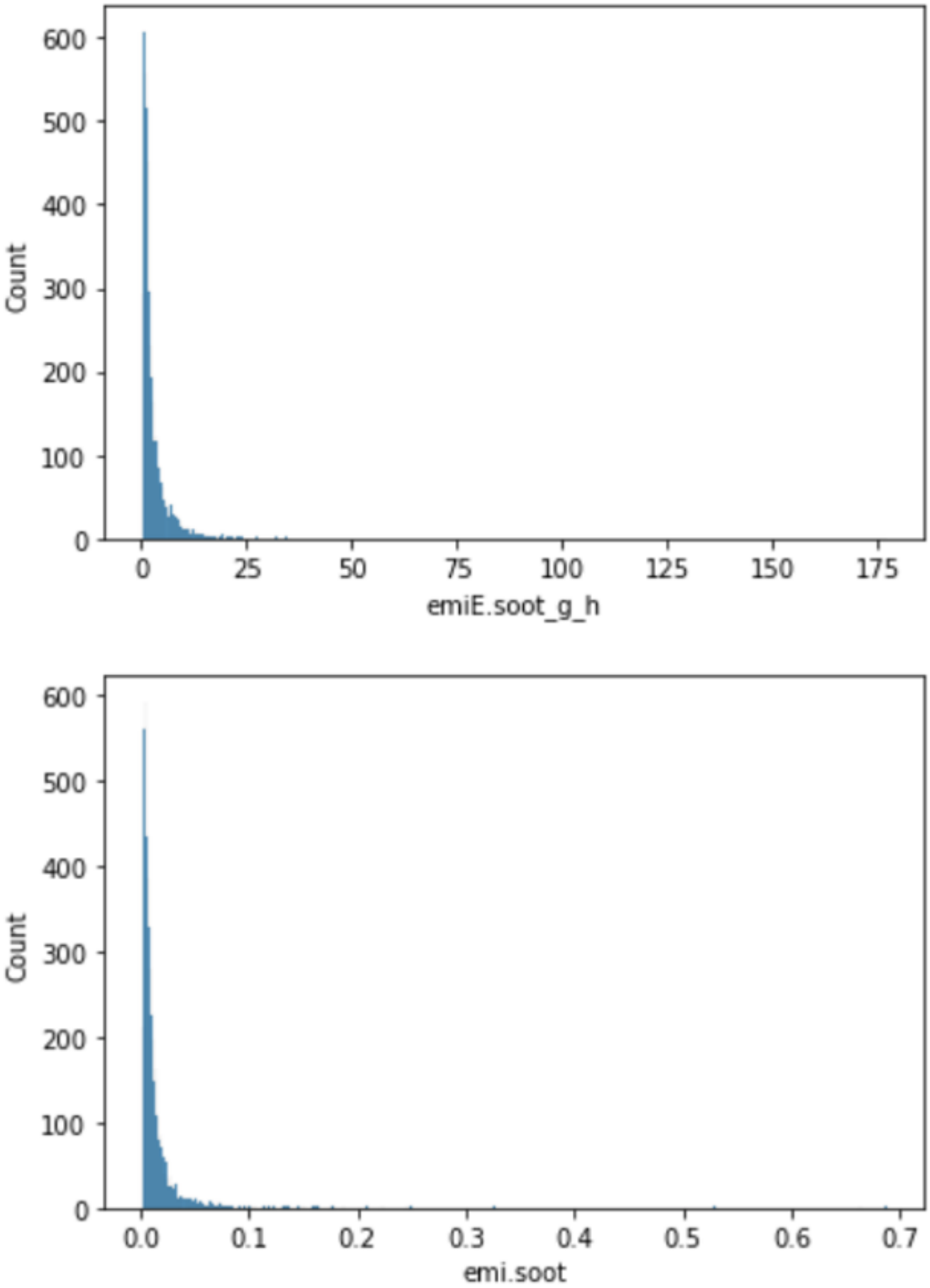


Figura 8.7. Distribuzione delle variabili

Come si può notare dall'immagine (8.7) il problema risulta chiaro: le variabili che vogliamo predire in entrambi i casi hanno una distribuzione asimmetrica e distorta.
A questo punto diventa lecito chiedersi il perché ci interessi se le distribuzioni dei dati sono distorte?
Se la variabile di risposta è distorta come in questo caso, il modello verrà addestrato su un numero molto maggiore di valori di emissione "moderati" (vicino allo zero nel nostro caso) e sarà meno probabile che preveda con successo i valori di emissione più elevati.
Se i valori di una determinata variabile indipendente sono distorti, a seconda del modello, l'asimmetria può violare le ipotesi sulle quali si basa il modello stesso o può compromettere l'interpretazione dell'importanza delle features. Ed è proprio quello che avviene in questo caso studio, poiché l'algoritmo GBRT nello specifico non performa se associato a delle distribuzioni asimmetriche.

A tal proposito possiamo oggettivamente quantificare se la variabile è distorta calcolando i valori di *curtosi* e il *coefficiente di asimmetria di Fisher-Pearson*.
La **Curtosi** è un allontanamento dalla normalità distributiva, rispetto alla quale si verifica un maggiore appiattimento (distribuzione platicurtica) o un maggiore allungamento (distribuzione leptocurtica).
La sua misura più nota è l'**indice di Pearson**, rapporto tra il momento centrato di ordine 4 e il quadrato della varianza.
Il valore dell'indice corrispondente alla distribuzione normale (gaussiana) è 0. Un valore minore di 0 indica una distribuzione platicurtica, mentre un valore maggiore di 0 indica una distribuzione leptocurtica.
Il coefficiente di Curtosi è dato dalla formula:

$$\gamma_2 = \beta_2 - 3 \tag{8.1}$$

Dove:

$$\beta_2 = \frac{m_4}{m_2^2} \tag{8.2}$$

m_4 e m_2 sono rispettivamente il momento centrale di ordine 4 e 2.
Si eseguono perciò le valutazioni:

```
In [122]: from scipy.stats import kurtosis,skew
          kurtosis(data['emiE.soot_g_h']),skew(data['emiE.soot_g_h'])
Out[122]: (279.2064019245791, 12.396694790752406)

In [123]: kurtosis(data['emi.soot']),skew(data['emi.soot'])
Out[123]: (311.8100451277306, 13.706849917213106)
```

Figura 8.8. Valori degli indici della distribuzione

Come ci si aspettava i valori di curtosi sono maggiori di zero che indica una distribuzione leptocurtica (più allungata rispetto alla distribuzione normale) e i valori del coefficiente di asimmetria sono maggiori di zero che indica la presenza di coda destra[13]. Tutto questo avviene sia per l'output *emi.soot_g_h* e sia per *emi.soot*.
I grafici (8.7) nelle pagine precedenti rispecchiano a pieno questi valori.

Tutti questi fattori contribuiscono all'ottenimento dei bassi valori di output presenti per questa predizione.
Un modo per migliorare il potere predittivo del modello di Machine Learning risiede proprio nell'applicare una *trasformazione*, ove sia possibile, ai dati in modo da poterli rendere più adatti al training. Queste trasformazioni si traducono nella "Features Engineering"[14] che dovrà essere fatta sui dati del dataset riguardante l'inquinante soot.

8.1 Features engineering sui dati

Fatta l'analisi delle colonne si è trovato che la nostra colonna target delle emissioni ("*emi.soot*" e "*emiE.soot_g_h*") presenta asimmetria e curtosi positiva ed anche molto alta.
Questo come detto è un problema per qualsiasi algoritmo di Machine Learning che voglia un buon margine di generalizzazione perché il training risulta molto sbilanciato e la sensibilità alla variazione della distribuzione statistica di un dato campione sul quale fare le predizioni è molto alta.
Per risolvere il problema allora si applicano delle trasformate[12] alle variabili in gioco con l'obiettivo di ridurre curtosi e asimmetria nel target e sparpagliare le features per renderle più normalmente distribuite per il training.

La soluzione migliore trovata consiste nell'applicare la trasformata **Box-Cox**[14] alla variabile target e la trasformata di **Yeo-Johnson**[15] alle features.
Ottenere il risultato però non è così diretto in questo caso, c'è un procedimento da seguire in ordine:

8.1.1 Trasformate di Box-Cox e Yeo-Johnson

Prima di passare al procedimento utilizzato all'interno del codice GBRT per l'applicazione delle trasformate, si ritiene necessario ed opportuno presentare le due trasformate descrivendo le parti che li compongono.

In statistica, una *Power Trasform*[12] è una famiglia di funzioni che vengono utilizzate per creare una trasformazione monotona di dati utilizzando funzioni di potenza.
Questa è un'utile tecnica di trasformazione dei dati utilizzata per stabilizzare la varianza, rendere i dati più simili alla distribuzione normale, migliorare la validità delle misure di associazione come la correlazione di Pearson tra variabili e per altre procedure di stabilizzazione dei dati.

1) La trasformata di **Box-Cox** è una particolare trasformazione, proposta da Box e Cox nel 1964 per l'analisi della varianza allo scopo di rendere additivi, indipendenti, lineari gli effetti sulle variabili in esame.
Oggi, questa trasformazione si applica in tutti i settori statistici perché, generalmente, accentua simmetria e quindi tendenza alla variabile casuale Normale di una distribuzione empirica.
In molte tecniche statistiche, assumiamo che gli errori siano normalmente distribuiti. Questa ipotesi ci consente di costruire intervalli di confidenza e condurre test di ipotesi. Trasformando la variabile di destinazione, possiamo normalizzare i nostri errori.
Inoltre, la trasformazione delle nostre variabili può migliorare il potere predittivo dei nostri modelli perché le trasformazioni possono eliminare il cosiddetto "rumore bianco".
Se w è la nostra variabile trasformata e y è la nostra variabile obiettivo, allora:

$$w_t = \begin{cases} log(y_t), & \text{se } \lambda{=}0 \\ (y_t^{\lambda} - 1)/\lambda), & \text{altrimenti} \end{cases} \tag{8.3}$$

dove t è il periodo di tempo e λ è il parametro che scegliamo (puoi eseguire la trasformazione Box-Cox anche su dati non di serie temporali).

Quando λ è uguale a 1 i nostri dati si spostano verso il basso, ma la forma di essi non cambia.
Pertanto, se il valore ottimale per λ è 1, i dati sono già normalmente distribuiti e la trasformazione Box-Cox non è necessaria. Scipy ha una funzione boxcox che sceglierà per noi il valore ottimale di lambda.
Tuttavia prima di procedere con le previsioni sarà necessario ritrasformare i dati nella scala originale e solo allora si potrà continuare il lavoro.

2)La trasformazione **Yeo–Johnson** consente anche valori zero e negativi di λ che può essere qualsiasi numero reale, dove produce la trasformazione dell'identità.

$$w_t = \begin{cases} log(y_t + 1), & se\ \lambda = 0, y \equiv 0 \\ ((y_t + 1)^{\lambda} - 1)/\lambda), & se\ \lambda \neq 0, y \equiv 0 \\ -[(-y_t + 1)^{2-\lambda} - 1]/(2 - \lambda), & se\ \lambda \neq 2, y < 0 \\ -log(-y_t + 1), & se\ \lambda = 2, y < 0 \end{cases} \tag{8.4}$$

A questo punto si passa all'applicazione delle trasformazioni Box-Cox e Yeo-Johnson.
Il risultato però non è così diretto e immediato, c'è un preciso procedimento che bisogna seguire per applicare con efficacia le power trasform.
Il processo consiste nei seguenti punti:
1) Train-test split sui dati per avere X_train, X_test,y_train, y_test.
2) **Feature Engineering** sui dati applicando la trasformata **Box-Cox** ad y_train ed **Yeo-Johnson** ad X_train, X_test.
3) Training del modello usando X_train, y_train.
4) Predizione su X_test: $y\_pred = f(X\_test)$.
5) Trasformata inversa sulla predizione appena fatta su X_test e cioè $f^{-1}(y\_pred)$, in modo che sia confrontabile con y_test.
6) Calcolo del r^2.
Il procedimento riportato è legato al fatto che non possiamo in alcun modo applicare alcuna trasformazione ai valori y_test in quanto se lo facessimo avremmo un modello completamente diverso: l'obiettivo finale è quello di avere un modello che sia in grado di predire quanto meglio possibile i valori y_test originali come presenti nel dataset, per questo fine abbiamo piena libertà di poter svolgere feature engineering, tuning sui modelli etc usando X_train,X_test ed y_train ma MAI toccando y_test che una volta definito all'inizio non va toccato in alcun modo.

Una volta allenato il modello sui dati trasformati, alle predizioni va applicata la trasformata inversa a quella applicata ai dati di allenamento in modo che sia direttamente confrontabile con i dati originali *y_test* che verranno usati per calcolare il valore r^2 ed *rmse*.
Prima di procedere con lo studio delle due diverse casistiche è stato provato ad applicare le due trasformate ed immediatamente dopo applicare la funzione inverse per essere sicuri che si arrivasse allo stesso punto di partenza senza che esse portassero variazioni non desiderate ai dati.

Caso 1 variabile target: *Emi.soot*

Si riporta di seguito il caso del modello che si ottiene utilizzando come variabile target la colonna "*Emi.soot*":
Come prova delle scarse capacità predittive del modello utilizzando i dati originali calcoliamo r^2 dalla cross validation proprio come già mostrato all'inizio del capitolo.

```
from sklearn.model_selection import cross_validate

reg=GradientBoostingRegressor()
reg.fit(X_train,y_train)
y_predictions=reg.predict(X_test)
scores = cross_validate(reg, X_train, y_train, cv=5,scoring='r2',return_train_score=True,n_jobs=-1)
print('CV test scores:',scores['test_score'],'\nCV train scores:',scores['train_score'])
test_score = r2_score(y_test, y_predictions)# calcolo il valore di r2

rmse = mean_squared_error(y_test, y_predictions)# calcolo l'errore ai minimi quadrati

print('Test-set r2',test_score,'\nRmse',rmse)
```

```
CV test scores: [0.20822845 0.60476852 0.43076435 0.67767018 0.4886629 ]
CV train scores: [0.94086776 0.93424079 0.92058115 0.94312235 0.92432967]
Test-set r2 0.5718428655281398
Rmse 0.0002377516140403328
```

Figura 8.9. Codice utilizzato per il calcolo della r^2 sia nel train che nel test

Risulta è che ad ogni iterazione r^2 sul training è superiore al 0.9 e sul validation mai oltre il 0.67 (nel migliore dei casi).
Siccome per ogni iterazione del cross-validation r^2 sul training è alto sembra un problema

legato principalmente all'overfitting motivo per il quale si sono provate diverse opzioni come algoritmi diversi, regolarizzazione sul GBRT per semplificare l'algoritmo ma non funzionano.
Si decide perciò di procedere con l'applicazione della trasformata Box-Cox ad y_train ed Yeo-Johnson ad X_train,X_test.
Nel frammento di codice si può osservare l'implementazione delle trasformate all'interno dell'ambiente Python. Queste funzioni sono state importate dalla libreria sklearn.preprocessing.

```
from sklearn.preprocessing import PowerTransformer

bc_Ytrain=None
bc_Xtrain=None
pretrained=False
params=None

# TRASFORMATA DI BOX-COX E DI YEO-JOHNSON
if soot_cases==True:
    bc_Xtrain = PowerTransformer()
    bc_Xtrain.fit(X_train)
    bc_Ytrain = PowerTransformer(method='box-cox')
    bc_Ytrain.fit(y_train.reshape(-1, 1))
    X_train = bc_Xtrain.transform(X_train)
    X_test = bc_Xtrain.transform(X_test)
    y_train = bc_Ytrain.transform(y_train.reshape(-1, 1))
    y_train = y_train.reshape(-1)
    pretrained = True
```

Figura 8.10. Codice in ambiente Python per l'implementazione nell'algoritmo delle power trasform di Box-Cox e Yeo-Johnson

La trasformazione di Yeo-Johnson è implementata di default nelle Power Trasformer, ma quella di Box-Cox deve essere specificato come metodo utilizzato, il tutto è visibile nella parte di codice mostrato in figura (8.10).
Si riportano ora i grafici della distribuzione della variabile target prima e dopo l'applicazione della trasformata Box-Cox e Yeo-Johnson, figura (8.11).

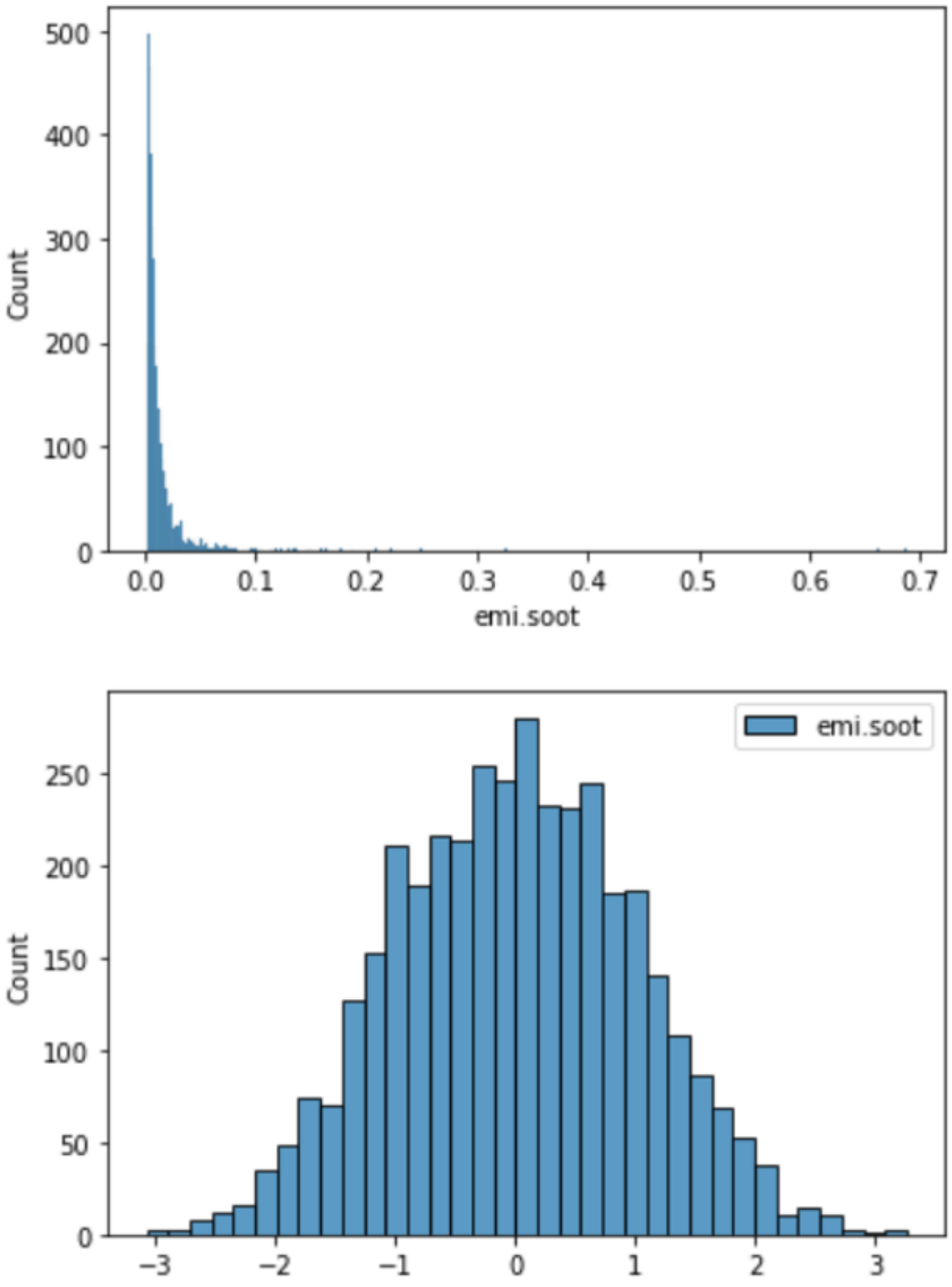


Figura 8.11. Distribuzione della variabile target prima e dopo l’applicazione delle trasformate

Come si può notare l’effetto è quello della normalizzazione della distribuzione.

Si procede quindi ad all’analisi dei valori di r^2 che si ottengono dalla cross-validation e dal test, (stesso test fatto in precedenza):

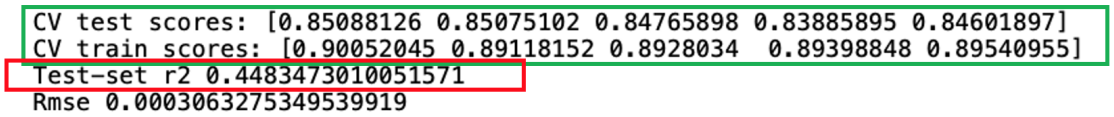


Figura 8.12. Valori CV step intermedio

Come si può notare nella figura (8.12) i valori di r^2 dalla cross-validation sono molto alti rispetto a quelli ottenuti utilizzando i dati originali all’inizio, il problema che si pensava fosse legato all’overfitting era invece legato proprio allo sbilanciamento dei dati.
Risolto questo si procede a questo punto con il *tuning degli iperparametri* al fine di aumentare il valore r^2 sul test set.
Si precisa che cv-train e cv-test sono calcolati con il metodo della cross-validation con i dati trasformati.
I valori alti di r^2 sono relativi al validation set e non test set.
Facendo il tuning degli iperparametri il risultato migliora perché i parametri si adattano meglio al problema trattato.
In questo caso studio si decide di utilizzare più iperparametri per permettere un ottimizzazione migliore.
Diversamente dagli altri casi, qui abbiamo i seguenti iperparametri:

-*loss*: Funzione di perdita da ottimizzare. Il valore ”lad” (la deviazione minima assoluta) assegnato è una funzione di perdita altamente robusta basata esclusivamente sulle informazioni sull’ordine delle variabili di input.

-max_depth: Profondità massima dei singoli stimatori di regressione. La profondità massima limita il numero di nodi nell’albero. Si regola questo parametro per ottenere le migliori prestazioni; il valore migliore dipende dall’interazione delle variabili di input.

-$max\_features$: Il numero di features da considerare quando si cerca la divisione migliore. Se è un int, come nel nostro caso, bisogna considerare le features $max\_features$ in ogni

divisione.

-$min\_samples\_leaf$: Il numero minimo di campioni richiesto da trovarsi in un nodo foglia. Un punto di divisione a qualsiasi profondità verrà considerato solo se lascia almeno $min\_samples\_leaf$ campioni di addestramento in ciascuno dei rami sinistro e destro. Ciò potrebbe avere l'effetto di smussare il modello, soprattutto nella regressione.

-$min\_samples\_split$: Il numero minimo di campioni richiesti per dividere un nodo interno.

-$n\_estimators$: Il numero di fasi di potenziamento da eseguire. L'aumento del gradiente è abbastanza robusto per l'adattamento eccessivo, quindi un numero elevato di solito produce prestazioni migliori.

-*subsample*: Indica la frazione di campioni da utilizzare per l'adattamento dei singoli studenti di base. Se inferiore a 1.0, si ottiene un aumento del gradiente stocastico. il sottocampione interagisce con il parametro $n\_estimators$. La scelta del sottocampione < 1.0 porta a una riduzione della varianza e ad un aumento del bias.

-$validation\_fraction$: Indica la percentuale di dati di addestramento da mettere da parte come set di convalida per l'interruzione anticipata. Deve essere compreso tra 0 e 1.

Il tuning di questi iperparametri è stato fatto col fine di aumentare il valore r^2 sul test set, perciò, diversamente dagli altri casi, qui sono stati trovati con cura e con molte iterazioni dei valori degli iperparametri che portavano ad un aumento sostanzioso di accuratezza nella predizione di soot nel motore 11.0L.
Dopo circa 300 iterazioni con il metodo BayesSearchCV sono stati ottenuti i seguenti valore degli iperparametri ottimali, mostrati in figura (8.13).

```
import collections
params=collections.OrderedDict([('learning_rate', 0.12236942089288666),
             ('loss', 'lad'),
             ('max_depth', 7),
             ('max_features', 13),
             ('min_samples_leaf', 1),
             ('min_samples_split', 98),
             ('n_estimators', 966),
             ('subsample', 0.40000794181874977),
             ('validation_fraction', 1.0)])
params
```

Figura 8.13. Valore degli iperparametri ottimali

Siccome il training e la predizione da parte del modello sono fenomeni statistici risulta chiaro che per avere un valore realmente attendibile delle effettive performance del modello non si può trascurare questa caratteristica intrinseca della stocasticità dei risultati: si procede perciò alla costruzione di un campione di valori di r^2 con numerosità n, al fine di poter definire un valore medio di r^2 e un opportuno intervallo di confidenza per quest'ultimo.
A tal fine si ripete la predizione 1000 volte ottenendo 1000 valori di r^2 (figura (8.14)) sulla base dei quali si può costruire la distribuzione di densità. In fine si calcola l'intervallo di confidenza al 95% tramite **bootstrapping**.
Si riporta di seguito la distribuzione di densità dei valori di r^2:

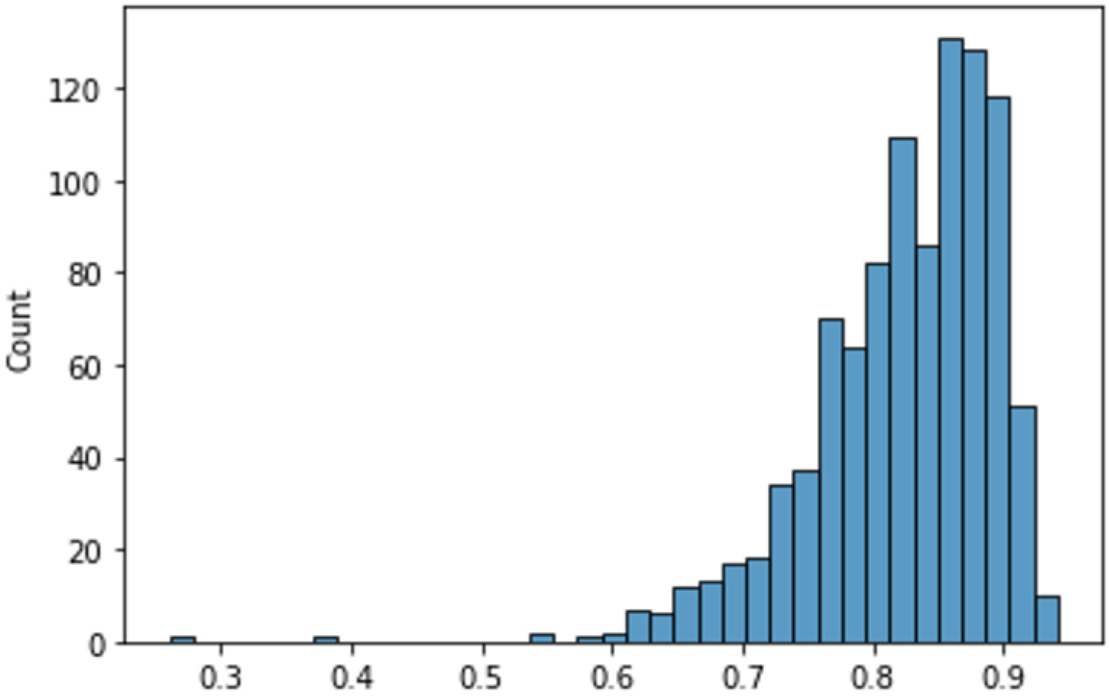


Figura 8.14. Distribuzione di densità dei valori di accuratezza in uscita dal codice per la stessa predizione ripetuta 1000 volte

Il bootstrap è una tecnica statistica di ricampionamento con reimmissione per approssimare la distribuzione campionaria di una statistica. Permette perciò di approssimare media e varianza di uno stimatore, costruire intervalli di confidenza etc quando non si conosce la distribuzione della statistica di interesse. Si tratta di un metodo statistico per stimare la distribuzione di campionamento di uno stimatore nel nostro caso il valor medio di r^2.
Si usa il 95% e poi si considerano estremi al 2,5% ed 97,5% semplicemente perché il 95% centrato rispetto al 100% lascia fuori il 2,5% a sinistra e il 2,5% a destra visto che $100 - 95 = 5$ e $5/2 = 2{,}5$. Partendo quindi da 2,5% a predire il 95% arrivi al punto che sta al 97,5%. $N = 100000$ sono i campionamenti, tutto questo però lo osserveremo nel seguente calcolo dell’intervallo di confidenza.

Notiamo dalla figura (8.14) usando gli iperparametri ottimali trovati, il valore mediano di r^2 sul test risulta superiore all’ 80% ed i valori tendono ad addensarsi fra l’80% e il 90%. Siccome la distribuzione presenta degli outliers attorno allo 0.3,0.4, risulta evidente la maggiore correttezza nell’utilizzare la mediana (tabella (8.15)) anziché la media come opportuno estimatore delle performance del modello.

Prima di calcolare la mediana e decretare perciò il valore ottenuto dallo studio e i notevoli vantaggi ottenuti, è opportuno osservare come già dal grafico si siano ottenuti ottenuti valori sempre molto prossimi allo 0.90 che quindi sposano perfettamente lo scopo della predizione con elevata accuratezza dell'inquinante soot anche in questo caso oltre che per i precedenti.

```
np.median(test_score)
```

```
0.8355851201441837
```

Figura 8.15. Valore della mediana della popolazione

Si procede ora con il calcolo dell'*intervallo di confidenza* tramite bootstrapping.
La procedura per trovare l'intervallo è la seguente:
1) Estrazione (campionamento con sostituzione) di N campioni dal campione originale ciascuno dei quali contenente M elementi, con M di grandezza esattamente uguale a quella del campione originale (nel nostro caso M=1000).
2) Per ciascuno dei campioni estratti, calcolo valore mediano di r^2.
3) Disposizione in ordine di grandezza di questi valori mediani trovati al punto 2.
4) Per ottenere, diciamo, un intervallo di confidenza del 95%, calcoleremo gli estremi che delimitano il 95% dei valori mediani di r^2. Per questo, si trovano i valori ai percentili del 2,5% e del 97,5%. Il 2,5esimo percentile sarà nella posizione (0,025) (N + 1) e il 97,5esimo percentile sarà nella posizione (0,975) (N + 1). I valori di queste posizioni sono i limiti inferiore e superiore dell'intervallo di bootstrap del 95% per la media reale. Nel nostro caso scegliamo N=100000.

```
ts_emi=test_score
ts_emi=np.array(ts_emi)
N = 100000

mean_estimates = []
for _ in range(N):
    re_sample_idx = np.random.randint(0, len(ts_emi),ts_emi.shape)
    mean_estimates.append(np.median(ts_emi[re_sample_idx]))
sorted_estimates = np.sort(np.array(mean_estimates))
conf_interval = [sorted_estimates[int(0.025 * N)], sorted_estimates[int(0.975 * N)]]
conf_interval
```

Figura 8.16. Codice per l'intervallo di confidenza

Dunque l'intervallo ottenuto per queste predizioni si trova in ogni misurazione in mezzo a questi due limiti mostrati nella figura (4.2):

```
Out[134]: [0.8303953903830659, 0.8417717069091135]
```

Figura 8.17. Limiti dell'intervallo di confidenza

Risulta in definitiva che il modello costruito è in grado di realizzare un valore mediano r^2 pari ad **0.835** con intervallo di confidenza al 95% pari ad [0.8304, 0.8418].
Come vedremo a breve questo ragionamento è stato fatto per entrambi gli output in modo da capire quale considerare come output della predizione e che miglioramenti in termini di accuratezza porta.

Caso 2 variabile target: $Emi.soot\_g\_h$

Applichiamo ora la stessa metodologia e gli stessi passi usando come variabile target, cioè output del problema la colonna "$emiE.soot\_g\_h$" anzichè la "$emi.soot$".

```
from sklearn.model_selection import cross_validate
reg=GradientBoostingRegressor()
reg.fit(X_train,y_train)
y_pred=reg.predict(X_test)
scores = cross_validate(reg, X_train, y_train, cv=5,scoring='r2',return_train_score=Tru
print('CV test scores:',scores['test_score'],'\nCV train scores:',scores['train_score']
test_score = r2_score(y_test, y_pred)# calcolo il valore di r2
rmse = mean_squared_error(y_test, y_pred)# calcolo l'errore ai minimi quadrati
print('Test-set r2',test_score,'\nRmse',rmse)
```

```
CV test scores: [0.4346811  0.64245151 0.42526487 0.59740504 0.52501078]
CV train scores: [0.91844936 0.92078872 0.91023162 0.92538631 0.89818506]
Test-set r2 0.564819429917663
```

Figura 8.18. Calcolo r^2 per test e per train

Come nel caso precedente i valori di r^2 dal cross validation per il training sono superiori allo 0.9 mentre per il validation si aggirano attorno al 0.5. Tutto questo lo si può osservare nella parte di codice (8.18).
Applichiamo perciò le trasformate ai nostri dati, con le stesse ipotesi del caso precedente:

Si procede ora con l'applicazione della trasformata Box-Cox ad y_train ed Yeo-Johnson ad X_train,X_test, codice (8.19).

```
from sklearn.preprocessing import PowerTransformer


bc_Xtrain = PowerTransformer()      Yeo-Johnson di default
bc_Xtrain.fit(X_train)

bc_Ytrain = PowerTransformer(method='box-cox') Box-Cox
bc_Ytrain.fit(y_train.values.reshape(-1,1))


X_train = bc_Xtrain.transform(X_train)
X_train=pd.DataFrame(X_train,columns=X_test.columns)

X_test = bc_Xtrain.transform(X_test)
X_test=pd.DataFrame(X_test,columns=X_train.columns)

y_train_old=y_train
y_train = bc_Ytrain.transform(y_train.values.reshape(-1,1))
y_train=pd.DataFrame(y_train,columns=[target])
```

Figura 8.19. Codice implementazione PowerTransformers

Riportiamo nelle seguenti figure (immagine (8.20)) come varia la distribuzione dei dati a seguito dell'applicazione delle trasformate, passando da una distribuzione tutta concentrata nell'intorno dello zero ad una distribuzione più simile ad una Normale:

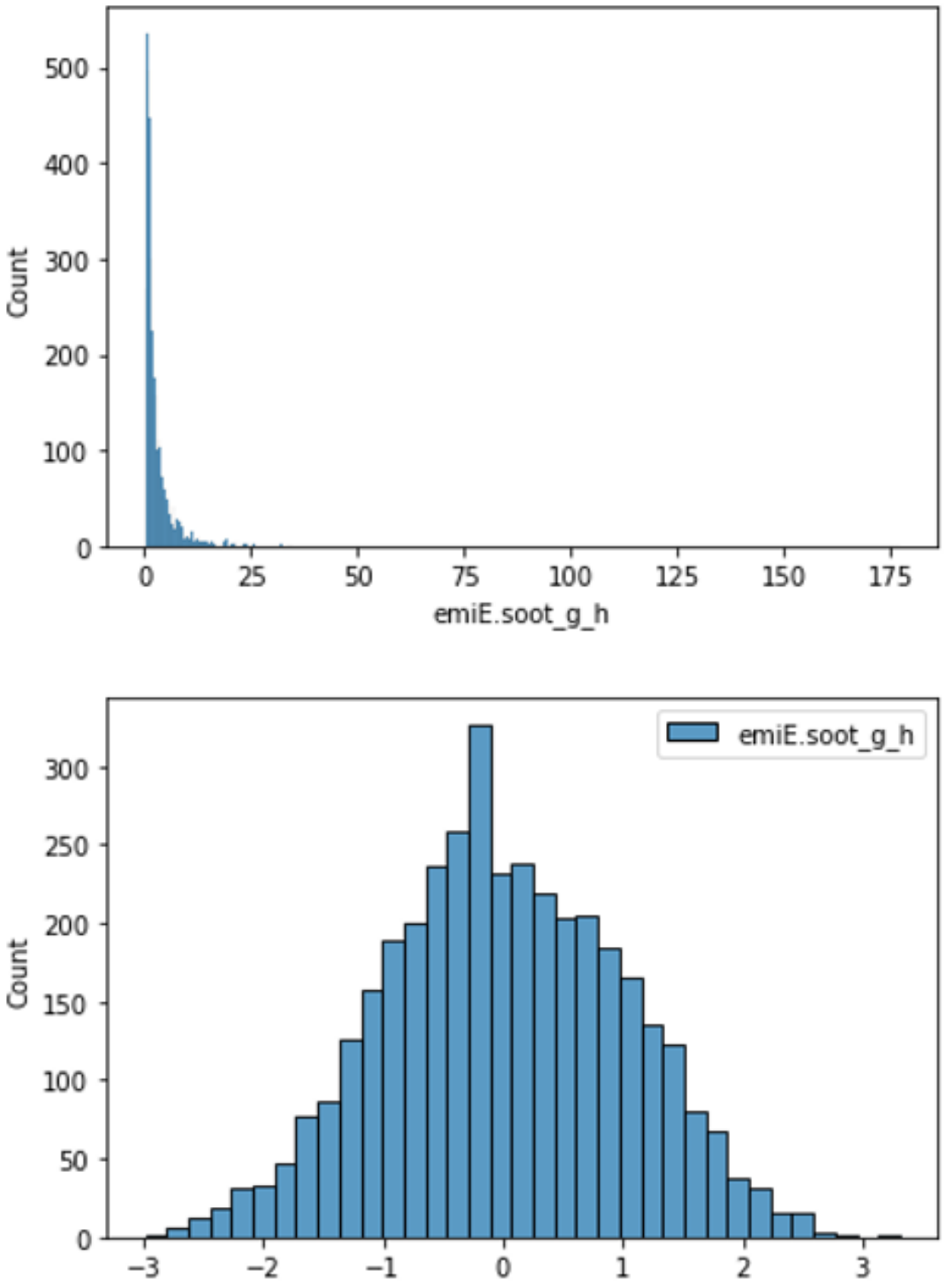


Figura 8.20. Variazione della distribuzione della variabile target prima e dopo l’applicazione delle trasformate di Box-Cox e Yeo-Johnson

Come ormai ci si aspettava la distribuzione risulta molto più normalizzata e assomigliante ad una gaussiana. Questa tipologia di distribuzione come già detto in precedenza consente all'algoritmo GBRT di ottimizzare la sua predizione del numero di inquinanti.
Si torna perciò ad analizzare i valori di r^2 sia per il train che per il test andando a vedere se la situazione è migliorata rispetto i valori precedenti.

```
CV test scores: [0.85745455 0.85010566 0.85635673 0.84475442 0.84637602]
CV train scores: [0.90275644 0.90192494 0.90115552 0.90061038 0.90108869]
Test-set r2 0.5123276189221089
Rmse 16.80716660828172
```

Figura 8.21. Calcolo r^2 per test e per train dopo l'applicazione delle PowerTransform

Risulta evidente perciò, come riquadrato in verde nella figura (8.21), che i valori di accuratezza per il train e per il test sono valori elevati che coincidono con lo scopo preposto.
Resta ancora basso, come in precedenza, il valore sul test-set perciò si decide di operare sul tuning degli iperparametri aggiungendone alcuni rispetto il caso del motore 2.0L e 11.0L (inquinante NO_X).
Come si può notare i valori di r^2 dal cross-validation sono molto alti rispetto a quelli ottenuti utilizzando i dati originali all'inizio del capitolo, il problema che si pensava fosse legato all'overfitting era invece legato proprio allo sbilanciamento dei dati.
Risolto questo si procede a questo punto con il tuning degli iperparametri (figura (8.22)) al fine di aumentare il valore r^2 sul test set.
Nuovamente si decide di pre-itinerare con 300 iterazioni Bayes gli iperparametri in modo da individuarne i valori precisi e ottimali che permettono all'algoritmo GBRT di riuscire a predire con esattezza il numero di inquinante soot allo scarico del motore Diesel da 11.0L di cilindrata.

```
import collections
params=collections.OrderedDict([('learning_rate', 0.127305028859358),
             ('loss', 'ls'),
             ('max_depth', 7),
             ('max_features', 14),
             ('min_samples_leaf', 22),
             ('min_samples_split', 100),
             ('n_estimators', 500),
             ('subsample', 0.4302653844866947),
             ('validation_fraction', 0.0)])
params
```

Figura 8.22. Collezione del valore degli iperparametri ottimizzati

Come fatto precedentemente ricostruiamo la distribuzione di densità dei valori r^2 al fine di definire il valor medio che il modello sarà in grado di realizzare e l'intervallo di confidenza per quest'ultimo al 95%. Si riporta di seguito, in figura (8.23), la distribuzione di densità dei valori di r^2:

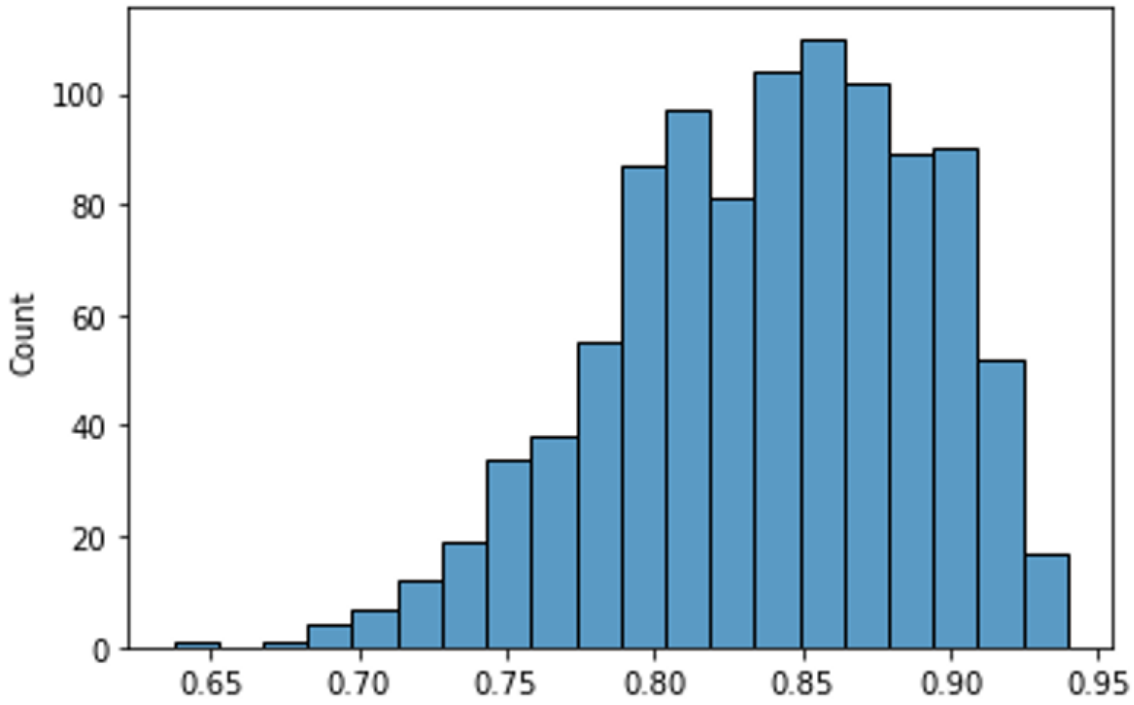


Figura 8.23. Collezione del valore degli iperparametri ottimizzati

Come si può notare, anche in questo secondo caso, usando gli iperparametri ottimali trovati il valore mediano di r^2 sul test risulta superiore all'80% ed i valori tendono ad addensarsi

fra l'80% e il 90%.
Siccome la distribuzione presenta degli outliers attorno allo 0.3,0.4, risulta evidente la maggiore correttezza nell'utilizzare la mediana anziché la media come opportuno estimatore delle performance del modello, ottenendo così:

```
np.median(test_score_e)
0.8424221886942653
```

Figura 8.24. Mediana del test score

Si procede anche in questo secondo caso con il calcolo dell'*intervallo di confidenza* tramite bootstrapping.
La procedura per trovare l'intervallo resta la medesima:
1) Estrazione (campionamento con sostituzione) di N campioni dal campione originale ciascuno dei quali contenente M elementi, con M di grandezza esattamente uguale a quella del campione originale (nel nostro caso M=1000).
2) Per ciascuno dei campioni estratti, calcolo valore mediano di r^2.
3) Disposizione in ordine di grandezza di questi valori mediani trovati al punto 2.
4) Per ottenere, diciamo, un intervallo di confidenza del 95%, calcoleremo gli estremi che delimitano il 95% dei valori mediani di r^2. Per questo, si trovano i valori ai percentili del 2,5% e del 97,5%. Il 2,5esimo percentile sarà nella posizione (0,025) (N + 1) e il 97,5esimo percentile sarà nella posizione (0,975) (N + 1). I valori di queste posizioni sono i limiti inferiore e superiore dell'intervallo di bootstrap del 95% per la media reale. Nel nostro caso scegliamo N=100000.
Dunque l'intervallo ottenuto per le predizioni in questo secondo caso è mostrato in figura (8.25) ed ogni valore di accuratezza compreso nel 95% della confidenza ricadrà in questo intervallo.

```
Out[178]: [0.8379492237047887, 0.8472105671180736]
```

Figura 8.25. Limiti dell'intervallo di confidenza nel secondo caso studiato

Risulta in questo caso che il modello costruito è in grado di realizzare un valore mediano r^2 pari ad **0.842** con intervallo di confidenza al 95% pari ad [0.838, 0.847].
Una volta terminati entrambi gli studi delle power transform sui due diversi output possiamo giungere a delle importanti conclusioni riguardanti l'accuratezza della predizione di inquinante soot nel motore 11.0L.

Conclusioni:
Si nota che usando la colonna chiamata **"emiE.soot_g_h"** che riporta le emissioni in altre unità di misura, l'r^2 è poco più alto rispetto all'utilizzare la colonna **"emi.soot"**.
Questo benché strano risulta attendibile in quanto nonostante la colonna "$emiE.soot\_g\_h$" dovrebbe riportare lo stesso valore delle emissioni in diverse unità di misura sembra che si discosti di pochissimo rispetto alla colonna "$emi.soot$": il coefficiente di correlazione fra le due è pari ad 0.94. Dallo studio della matrice di correlazione del dataset si vede come i due output non siano perfettamente uguali, ma si correlano a vicenda con un valore nei pressi dell'1, ma non 1, perciò è attendibile che per i due casi si ottengano valori diversi.
Quindi si può vedere i valori di r^2 considerando come target '**emiE.soot_g_h**' di addensano sul **0.842** con varianza ragionevolmente bassa: l'intervallo di confidenza al 95% sta fra [0.838, 0.847]. Per target '**emi.soot**' stiamo attorno a valore mediano **0.835** con intervallo di confidenza al 95% pari a [0.8304, 0.8418] quindi poco più basso dell'altro caso.
Per concludere questo studio sulle power transformation, oltre a dimostrare l'utilità di esse ottenendo valori di accuratezza molto più alti rispetto i casi iniziali, si arriva a determinare che il miglior target, cioè output, da utilizzare per l'algoritmo GBRT nello studio dell'inquinante soot è la colonna "$emiE.soot\_g\_h$".

8.1.2 Risultati dataset completo

Grazie le trasformate di Box-Cox e Yeo-Johnson è stato possibile ottenere valori di accuratezza elevati anche per la predizione di soot nel motore 11.0L.
In questo sottocapitolo si presenteranno i valori tabulati il dataset completo sia affrontando una strategia con valori calcolati a banco e sia con una strategia che usa valori stimati in centralina ECU.

Bench Strategy

In primo luogo si tratta la strategia Banco-Banco in cui sia per l'addestramento che per il test sono stati utilizzati esclusivamente i valori uscenti dal banco prova.

I valori risultati per cui possiamo valutare l'andamento della predizione sono sempre l'r^2 e l'$rmse$ come vediamo nelle tabelle.

Bench Strategy 11.0L Accuracy	Norm0	Norm1
TTs=0.2	0,842	0,815
TTs=0.4	0,768	0,810
TTs=0.6	0,728	0,697
TTs=0.8	0,624	0,586

Figura 8.26. Risultati della predizione espressi in r^2 per l'algoritmo GBRT che sfrutta le trasformate B-C e Y-J

Bench Strategy 11.0L rmse	Norm0	Norm1
TTs=0.2	0,007	0,010
TTs=0.4	0,010	0,013
TTs=0.6	0,012	0,019
TTs=0.8	0,013	0,020

Figura 8.27. Risultati della predizione espressi in $rmse$ per l'algoritmo GBRT che sfrutta le trasformate B-C e Y-J

Dalla tabella (8.26) si nota subito il valore rappresentato in verde che coincide con il valore uscente dallo studio fatto nel capitolo precedente cioè il valor mediano della distribuzione di r^2 ottenuti dopo l'applicazione delle trasformate di Box-Cox e Yeo-Johnson.
È notevole il vantaggio che si è ottenuto dall'applicazione di esse, perché tornando alla figura (8.3) si può notare come senza di esse il valore predetto toccava un massimo di 0.553. Si ricorda che sia questa tabella che quella iniziale che presentava 0.553 come picco sono valutate col metodo di Bayes perché usando il GridSearchCV per l'ottimizzazione degli iperparametri si arrivava ad un massimo di 0.440 come mostrato in figura (8.4).

Perciò si può affermare che l'utilizzo delle trasformate ha portato da un picco di 0.553 ad uno di **0.842**, che rappresenta un grossissimo risultato per la predizione tramite algoritmo GBRT.
In questo studio, come nell'ECU Strategy, inoltre si è optato per soltanto due tipologie di normalizzazione poiché non si sono registrate sostanziali differenze.

Del caso migliore si mostrano in figura (8.28) e (8.29) gli andamenti dei punti predetti confrontati con quelli registrati nel dataset iniziale.

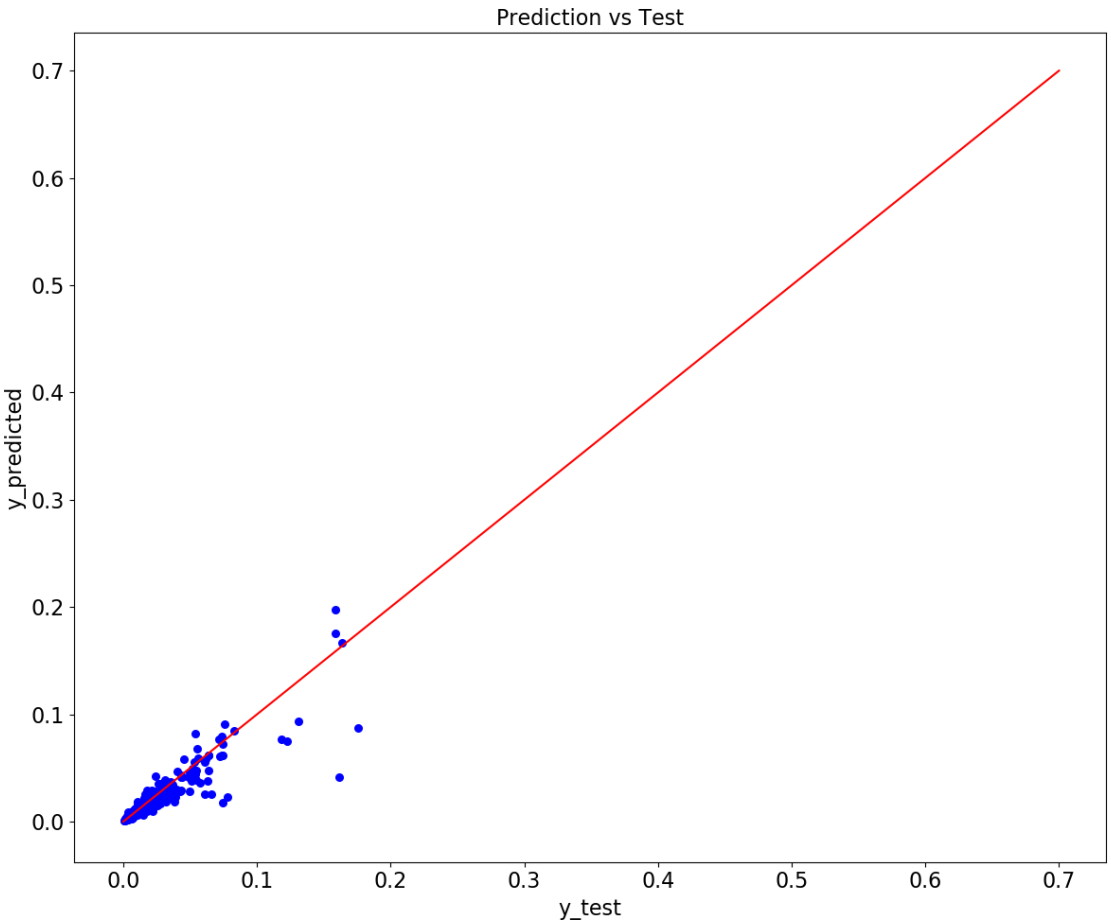


Figura 8.28. Grafico "Prediction vs Test" ottenuto per motore 11.0L per l'inquinante soot nel caso migliore $TTs = 0.2$ e $Norm_0$

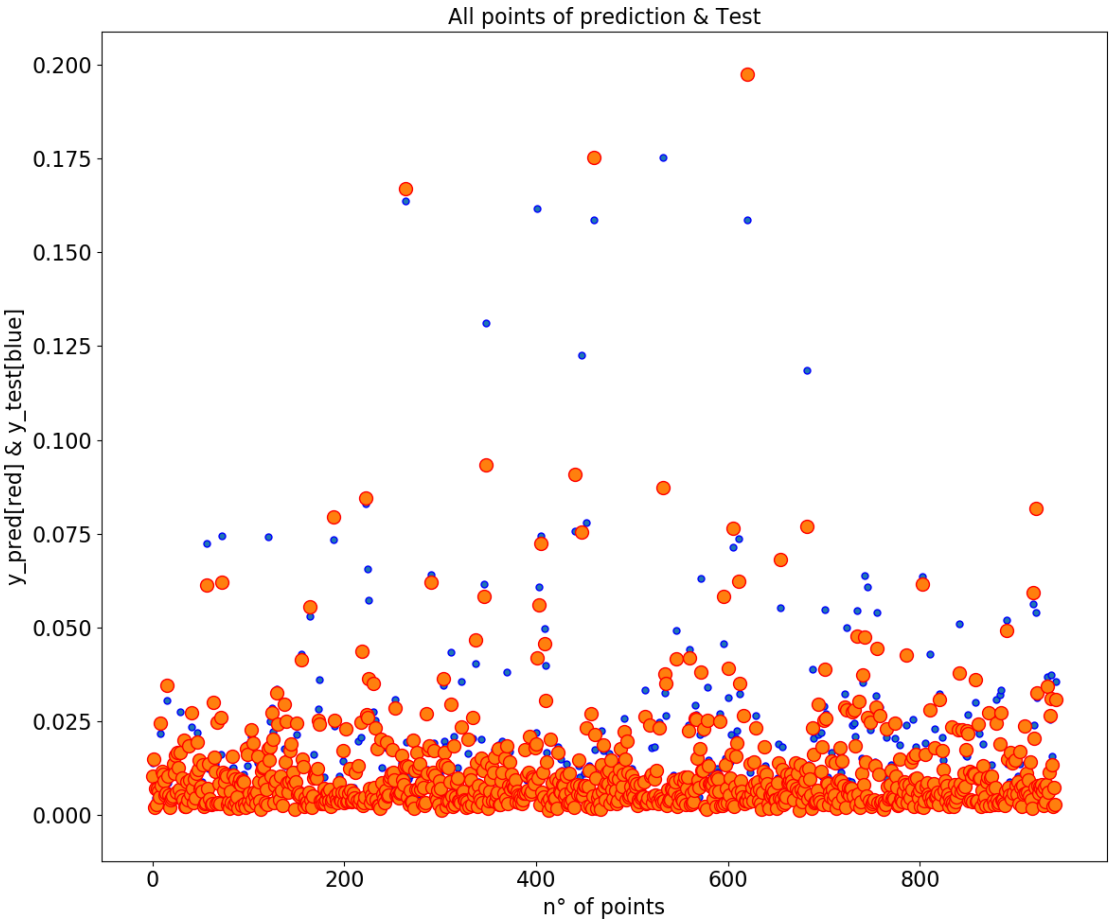


Figura 8.29. Grafico "All points of prediction & Test" ottenuto per motore 11.0L per l'inquinante soot nel caso migliore $TTs = 0.2$ e $Norm_0$

Il valore di accuratezza massimo non registra grandissime differenze da tutti gli altri casi essendo un valore pienamente accettabile rispetto quello iniziale. È però interessante osservare lo stesso grafico ottenuto però per il caso peggiore evidenziato in rosso nella tabella (8.26).

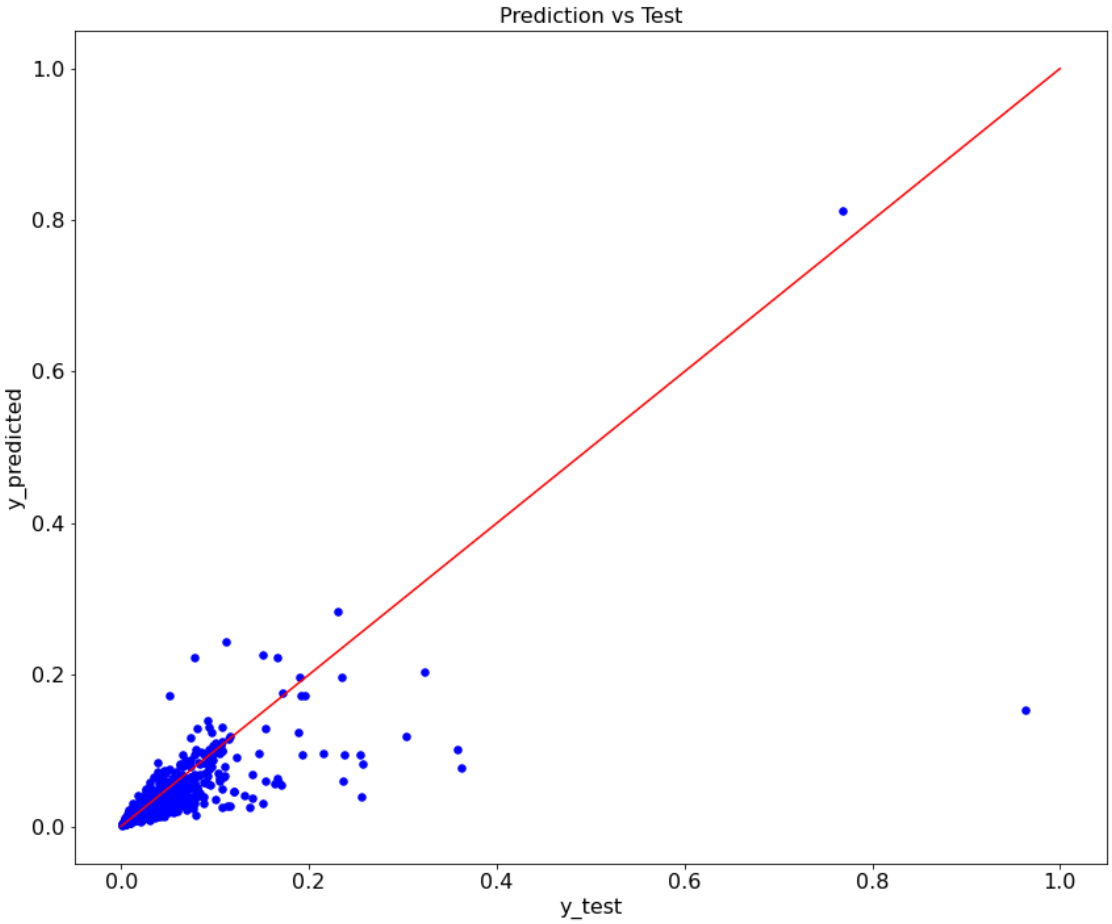


Figura 8.30. Grafico "Prediction vs Test" ottenuto per motore 11.0L per l'inquinante soot nel caso migliore $TTs = 0.8$ e $Norm_1$

Nel grafico (8.30) questa volta si vede un sostanziale allontanamento dalla linea rossa perchè si arriva a dei valori mediati di circa 0.600 che non sono esaustivi come quelli trovati per il caso dell'inquinante NO_X nello stesso motore.

Per questa configurazione Banco-Banco si mostra il valore per gli iperparametri dell'algoritmo. Come detto il precedenza in questo studio si è scelto di applicare iperparametri aggiuntivi per rendere più robusto l'utilizzo delle trasformazioni, tabella (8.31).

GBRT_Best_Parameters	Valore
learning_rate	0,122369421
loss	lad
max_depth	7
max_features	13
min_samples_leaf	1
min_samples_split	98
n_estimators	966
subsample	0,400007942
validation_fraction	1

Figura 8.31. Best Hyperparameters

Infine si va ad analizzare l'andamento delle due normalizzazioni in funzione dei diversi TTs. Rispetto agli andamenti trovati per l'inquinante NO_X notiamo una scala notevolmente diversa sulle y poichè i valori vanno da un massimo di circa 0.85 ad un min di circa 0.60. Queste due curve mostrate in figura (8.32) dimostrano come al ridurre dei dati a disposizione per l'addestramento il valore della predizione va criticamente diminuendo.

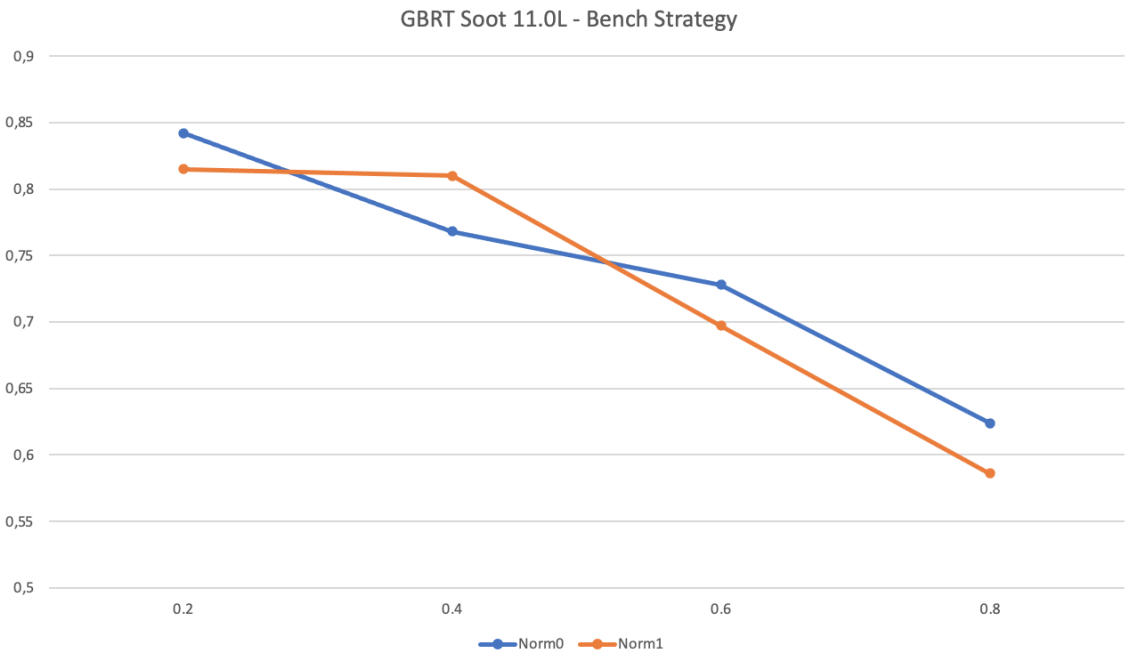


Figura 8.32. Andamento delle due normalizzazioni in funzione di r^2 e dei 4 TTs per il motore 11.0L nel caso dell'inquinante soot con configurazione Banco-Banco

Giustamente gli andamenti sono strattamente decrescenti e diminuiscono all'aumentare del TTs nonché al diminuire dei dati a disposizione per l'addestramento.

ECU Strategy

Fondamentalmente si ripercorrono gli stessi passi che portano a valori simili, ma leggermente minori perché tramite i dati stimati in centralina c'è una piccola e giustificata riduzione di efficienza perché le valutazioni sono meno precise.
I valori ottenuti per la strategia ECU cioè usando i dati derivanti dalla centralina sia per l'addestramento che per la validazione sono raccolti nelle tabelle (8.33) e (8.34):

ECU Strategy 11.0L Accuracy	Norm0	Norm1
TTs=0.2	0,800	0,817
TTs=0.4	0,662	0,714
TTs=0.6	0,638	0,589
TTs=0.8	0,515	0,547

Figura 8.33. Risultati della predizione espressi in r^2 per l'algoritmo GBRT che sfrutta le trasformate B-C e Y-J

ECU Strategy 11.0L rmse	Norm0	Norm1
TTs=0.2	0,009	0,017
TTs=0.4	0,012	0,016
TTs=0.6	0,013	0,018
TTs=0.8	0,016	0,024

Figura 8.34. Risultati della predizione espressi in $rmse$ per l'algoritmo GBRT che sfrutta le trasformate B-C e Y-J

Le stesse identiche considerazioni fatte per il caso precedente, Bench Strategy, possono essere adottate per questo caso poiché i valori sono molto simili e presentano gli stessi andamenti.
Per questa configurazione si mostra perciò direttamente l'andamento del valore di r^2 in funzione delle normalizzazioni e dei TTs (figura (8.35)) e subito dopo si presentano i valori che hanno assunto gli iperparametri per lo studio (tabella (8.36)).

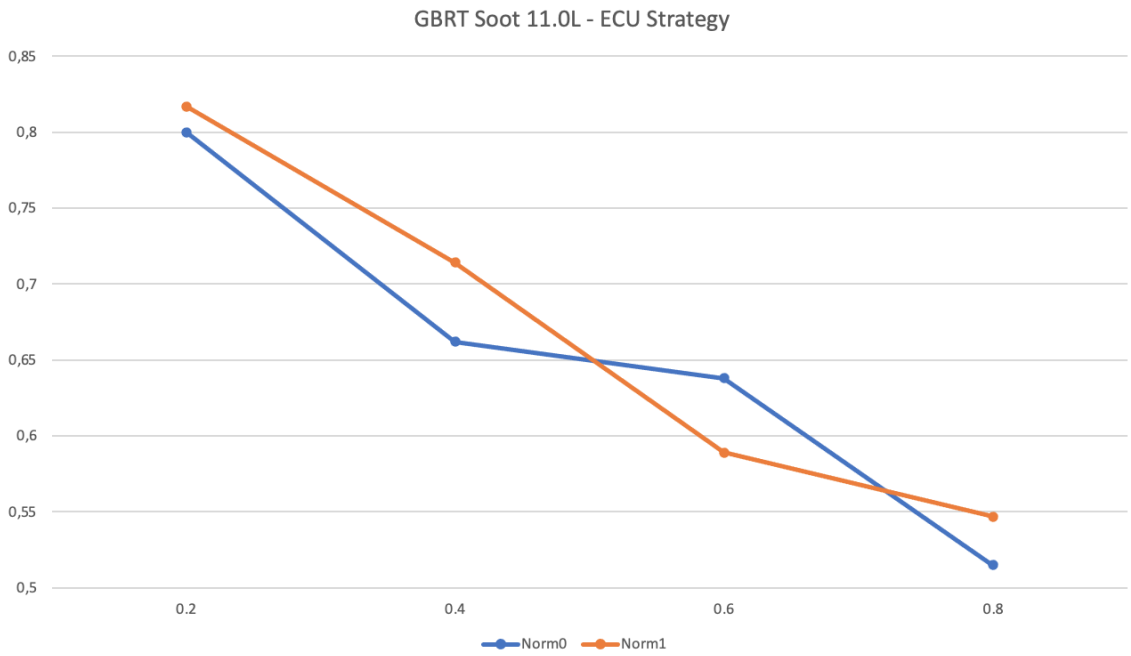


Figura 8.35. Andamento delle due normalizzazioni in funzione di r^2 e dei 4 TTs per il motore 11.0L nel caso dell'inquinante soot con configurazione ECU-ECU

GBRT_Best_Parameters	Valore
learning_rate	0,122369421
loss	lad
max_depth	7
max_features	13
min_samples_leaf	1
min_samples_split	98
n_estimators	966
subsample	0,400007942
validation_fraction	1

Figura 8.36. Best Hyperparameters

Si osserva che giustamente il valore degli iperparametri coincide in questo caso studio poiché in precedenza, dopo l'applicazione delle trasformate Box-Cox e Yeo-Johnson, sono stati pre-itinerati per 300 iterazioni col metodo di Bayes gli iperparametri scelti in modo da poter ottimizzarne il valore ed ottenere un alto valore di accuratezza per il calcolo predittivo dell'inquinante soot con l'algoritmo GBRT.

8.1.3 Risultati Banco-ECU

Un ulteriore caso studio per il motore da 11.0 litri di cilindrata nel caso dell'inquinante *soot* è stato quello di ottenere la predizione utilizzando per il training i dati misurati sul banco prova, mentre per il testing si utilizzano i banchi derivanti da centralina.
Non è stato ritenuto necessario provare la configurazione per diversi valori di TTs.

11L Soot Banco-ECU	Norm0	Norm1
Accuracy	0,791	0,794
rmse	0,010	0,015

Figura 8.37. Risultati espressi in r^2 e $rmse$ per la configurazione Banco-ECU

Questa prova è stata utile poiché rappresenta più a fondo lo scopo finale di questa idea della predizione degli inquinanti, cioè quella di implementare in futuro la predizione direttamente a bordo del veicolo.
Dalla tabella (8.37) si nota come per le due normalizzazioni si ottengono valori mediati nell'intorno di 0.8 .
Nella figura seguente (8.38) si nota la posizione dei punti predetti confrontati con quelli del dataset e si può vedere che sono nell'intorno della retta rossa posta a 45 gradi anche se non precisamente a cavallo e perciò giustifica il valore medio-alto di predizione.

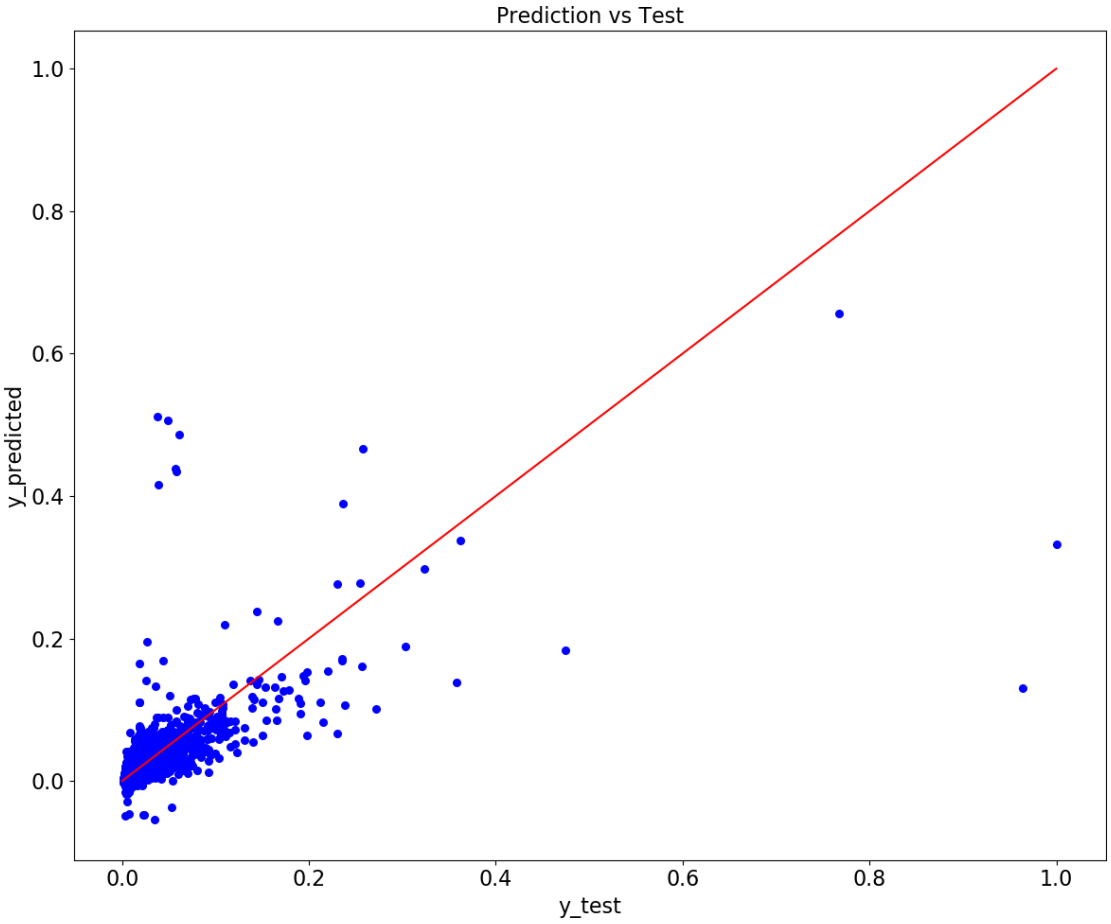


Figura 8.38. Grafico ”Prediction vs Test” nel caso di configurazione Banco-ECU per l’inquinante soot

Capitolo 9

Conclusione

L'elaborato di tesi si è occupato dell'applicazione di un modello di Machine Learning che sfrutta l'algoritmo Gradient Boosting Regression Tree. Questo codice, scritto in ambiente Python, è stato creato per poter stimare e predire sia il numero di particolato sia il numero di ossidi di azoto in condizioni stazionarie in due motori Diesel, e, più precisamente, in due motori di taglia differente.
Nella prima parte del lavoro è stato validato il codice applicandolo al motore 2.0 litri di cilindrata per la predizione del soot analizzando sia l'intero dataset che le prestazioni determinate dalle features importance, nonché un dataset ridotto.
Analogamente si è effettuato uno studio anche sul motore 11.0L per la predizione dell'emissione dell'inquinate NO_X.
Sono stati determinati, quindi, i parametri motoristici più influenti per l'analisi e la predizione del tipo di inquinante in punti di funzionamento stazionari.
L'algoritmo GBRT ha evidenziato la potenzialità di associare la produzione dell'inquinante a dei parametri motoristici ben precisi: dal punto di vista chimico-fisico, dosatura, pressione, inizio di iniezione del combustibile, quantità di combustibile, quantità di aria e concentrazione di ossigeno, percentuale di gas combusti ricircolati sono parametri in prima linea nella formazione e ossidazione di soot e NO_X, proprio come specificato dall'algoritmo che li classifica come parametri più importanti.
L'algoritmo è stato costruito per portare robustezza all'analisi fatta mediante lo studio sul numero di dati necessari per arrivare ad una soluzione soddisfacente. A tal proposito, si sono studiati i vari casi di TTs (train-test-size) dove al ridurre dei dati a disposizione per l'allenamento si va a perdere alcuni punti percentuale sul valore di accuratezza nella predizione dell'inquinante. Si è passati dall'utilizzare l'80% dei dati a disposizione per il training per l'allenamento dell'algoritmo, fino ad utilizzarne soltanto il 20%, osservando una progressiva diminuzione del valore di r^2. Come mostrato nel caso del motore 11.0L

una delle soluzioni vincenti per ovviare a questa riduzione potrebbe essere il disporre di un numero maggiore di dati per poter arrivare ad un'accuratezza migliore.
Tutto ciò ha evidenziato caratteristiche importanti del Machine Learning e della realizzazione di calcoli di regressione.
Lo studio, proposto anche per l'analisi e la predizione di inquinante NO_X in un motore 11.0L, ha portato all'ottenimento di valori dell'ordine dei 0,97 che garantiscono un'eccezionale accuratezza per l'obiettivo preposto.
Nella seconda parte della tesi sono state applicate delle trasformazioni sul dataset a disposizione per l'inquinante soot nel motore 11.0L. Le trasformate di Box-Cox e di Yeo-Johnson hanno portato la distribuzione della variabile target da un grafico concentrato tutto nei pressi del valore 0 ad una distribuzione con andamento Normale. Questo, a livello fisico, ha permesso di non ottenere casi di predizione di valori totalmente locati nei pressi del valore 0 (dati normalizzati), ma di distribuire questi valori nell'intero intervallo e rendere possibile un allenamento su tutto il campo a disposizione in modo tale da rendere capace il codice di Machine Learning di riconoscere e predire casistiche anche molto lontane dallo 0.
Questo studio ha evidenziato i parametri motoristici più influenti, che nel caso della predizione dell'inquinante soot nel motore 11.0L sono stati: la pressione di iniezione del combustibile nel rail, il numero di giri del motore, il rapporto tra quantità di aria e combustibile e il SOI dell'iniezione main.

9.1 Confronto con modello Random Forest (RF)

Il modello GBRT è stato confrontato con un modello realizzato in precedenza con l'algoritmo Random Forest[2] per constatarne l'affidabilità e l'efficacia.
Entrambi gli algoritmi sono utilizzati per stimare la formazione degli agenti inquinanti nel caso dell'inquinante soot nel motore 2.0L e l'inquinante NO_X nel motore 11.0L.
La casistica del soot nel motore 11.0L non sarà confrontata poiché, per l'ottenimento della predizione nell'algoritmo GBRT, sono state utilizzate delle Power Trasformer che ne hanno alterato la forma.
Entrambi gli algoritmi sono stati testati più volte e riportano dei valori mediati che attestano l'attendibilità dei risultati e ne confermano la validazione tramite le variazioni del Train Test size, presenti in entrambi i casi.
Come giusto che sia, per entrambi gli algoritmi si è arrivato a valori maggiori nel caso del $TTs = 0.2$ dove l'80% dei dati a disposizione è stato utilizzato per il training.

Nelle seguenti figure si pongono a confronto i valori ottenuti nel GBRT (9.1) e nel RF (9.2) per la predizione dell'inquinante soot nel motore 2.0L di cilindrata.

GBRT 2.0L Accuracy	Norm0	Norm1	Norm2
TTs=0.2	0,933	0,942	0,941
TTs=0.4	0,857	0,870	0,896
TTs=0.6	0,895	0,853	0,863
TTs=0.8	0,712	0,751	0,827

Figura 9.1. Risultati espressi in r^2 per il calcolo predittivo di soot nel motore 2.0L con l'algoritmo **GBRT**

RF 2.0L Accuracy	Norm0	Norm1	Norm2
TTs=0.2	0,899	0,907	0,933
TTs=0.4	0,874	0,876	0,892
TTs=0.6	0,812	0,818	0,857
TTs=0.8	0,703	0,706	0,766

Figura 9.2. Risultati espressi in r^2 per il calcolo predittivo di soot nel motore 2.0L con l'algoritmo **RF**

Infine, confrontando i due algoritmi, oltre che per il solo lato numerico, il codice GBRT presenta anche alcune migliorie rispetto il codice RF:
-Presenta una migliore precisione e accuratezza per la predizione degli inquinanti, stesso risultato si è ottenuto per il confronto nel caso NO_X.
-Maggiore versatilità poiché GBRT non ha riscontrato la necessità di modificare gli iper-parametri per poter effettuare studi con dataset ridotto.
-Tempi computazionali ridotti dovuti anche al fatto che nel GBRT si è preferito un ottimizzazione BayesSearchCV, mentre il RF ha utilizzato esclusivamente il GridSearchCV.

Il miglioramento dei risultati del modello in ML non è scienza esatta e dipende da molti fattori.
Per avere un'idea più chiara di cosa consiste l'intero processo che permette di arrivare a definire il modello migliore per un dato fenomeno, risulta la seguente successione di processi:
1) Collezione dei dati: si cercano più fonti possibili dalle quali estrarre i dati attinenti al problema (Banco e ECU).
2) Analisi, pulizia e studio dei dati per riuscire ad estrarne quante più informazioni possibili in termini statistici al fine di avere dati coerenti con il problema studiato.
3) Feature engineering: trasformazione dei dati al fine di mettere in risalto le particolari relazioni trovate; questo prevede anche la creazione di nuove features da quelle esistenti tramite applicazione di trasformate e combinazioni.
4) Scelta del modello: ricerca del modello che permette di ottenere i migliori risultati. Qui le possibilità sono molto ampie in quanto oltre la moltitudine di modelli base, si possono implementare lo stacking ensemble, che consiste in pratica nelle combinazioni dei vari algoritmi.
5) Tuning degli iperparametri del modello.
6) Feedback post messa in produzione.

Ognuna di queste fasi prevede inoltre una moltitudine di scelte e possibilità, quindi il fatto di trovare qualcosa di migliore non è cosi immediato ma richiede tempo e calcolo.
Di solito quando un modello è già attorno al 85%-90% di accuratezza, o che presenta un valore molto basso di $rmse$, non si tende a cercare di migliorarlo ulteriormente a meno che non si stia partecipando a qualche competizione, dove anche lo 0.1 di accuratezza in più può contare moltissimo ai fini della posizione in classifica.
Un modo diretto per aumentare le performance in maniera banale è sempre quello di trovare più dati da dare in pasto al modello.

9.2 Considerazioni finali

In conclusione si osserva come questo lavoro svolto costituisce solo uno dei primi passi per la realizzazione di un modello virtuale affidabile e accurato, pronto a sostituire o affiancare le mappe che accompagnano i sensori reali a valle del DPF per poter stimare con massima precisione i valori degli inquinanti prodotti dal motore Diesel allo scarico.
Negli step successivi, infatti, sarà interessante lo studio della formazione di altri inquinanti con lo scopo di ultimare il quadro e creare un codice che possa gestire la presenza di ogni inquinante completandone le informazioni necessarie all'after-treatment.
Il seguente lavoro di laurea magistrale si vuole muovere nella direzione dell'implementazione di questi algoritmi nella centralina del motore per rendere possibile un controllo on-board in tempo reale e quindi consentire l'ottimizzazione dell'after-tratment.
Risulta essenziale notare, infine, che il Machine Learning può costituire una grande arma a sostegno del mondo automotive per poter effettuare un level up nel campo delle emissioni inquinanti permettendo un minore inquinamento e gestendo al meglio i diversi *trade-off*.

Bibliografia

[1] Carbon Tracker: Electric vehicles - The catalyst to further decarbonisation

[2] Falai, A., Applicazione e validazione di un modello Random Forest per la stima della massa di particolato nei motori Diesel. Tesi di Laurea Magistrale, 2019 Politecnico di Torino, Torino.

[3] Millo, F., Propulsori termici, Laurea Magistrale Ingengeria meccanica, Propulsione dei veicoli terrestri, (Appunti del corso, 2019).

[4] Motori Diesel ad iniezione diretta: Alimentazione del combustibile e moto della carica - Università di Roma "Tor Vergata" – Corso di Motori a Combustione Interna

[5] Bella, G., Rocco, V. e Ubertini, S. (2002): "Combustion and Spray Simulation of a DI Turbocharged Diesel Engine", SAE Powertrain & Fluid Systems Conference & Exhibition 2002, paper No. 2002-01-2776.

[6] E. Spessa, Controllo delle Emissioni Inquinanti, Laurea Magistrale Ingengeria meccanica, Propulsione dei veicoli terrestri, (Appunti del corso, 2019).

[7] https://indigo.ai/it/machine-learning-definizione-e-campi-applicazione/

[8] https://www.cwi.it/tecnologie-emergenti/intelligenza-artificiale/machine-learning

[9] https:/scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html

[10] Manuale di Statistica per la Ricerca e la Professione, Lamberto Soliani - Università di Parma - 2007.

[11] A. Gèron, Hands-on Machine Learning with Scikit-Learn & TensorFlow, O'Reilly, 2017.

[12] https://en.wikipedia.org/wiki/Power_transform Box–Cox_transformation

[13] Zwillenger,D and Kokoska,S. (2000) CRC Standard Probability and Statistics Tables and Formulae. Chapman e Hall:New York.2000.Section 2.2.24.1

[14] I.K Yeo and R.A.Johnson, "A new family of power trasformations to improve normality or symmetry" Biometrica, 87(4), pp 954-959 (2000)

[15] G.E.P.Box and D.R.Cox, "An Analysis of Transformations", Journal of the Royal Statical Society B, 26, 211-252

[16] Micele, G. Natali, A. Sassu, N. , Utilizzo di strumenti di Machine Learning per la previsione di emissioni inquinanti da motori Diesel. Tesi di Laurea Magistrale, 2020 Politecnico di Torino, Torino.