Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Matematica

FOOTBALL DATA PREDICTION

Tesi di Laurea in Data Science

Relatore:
Prof. Luca De Angelis

Presentata da:
Matteo Agresti

Correlatore:
Prof.ssa Margherita Porcelli

Sessione Unica
Anno Accademico 2022-2023

Introduzione

Il calcio è uno degli sport più popolari e seguiti a livello mondiale. Ogni settimana, milioni di persone si appassionano alle partite, tifando per le proprie squadre del cuore e analizzando le prestazioni dei giocatori. Nel corso degli anni, il calcio si è evoluto non solo come forma di intrattenimento, ma anche come una scienza che può essere studiata e analizzata in modo approfondito. In particolare, ha guadagnato sempre più rilevanza negli ultimi anni la *Football Data Analysis*, che si concentra sull'elaborazione, la manipolazione e l'interpretazione dei dati specifici del calcio. Gli analisti di dati calcistici raccolgono e organizzano informazioni provenienti da diverse fonti, come registrazioni delle partite, dati statistici e dati sulle prestazioni dei giocatori. Questi dati vengono poi analizzati utilizzando metodi statistici e tecniche di *data analysis* al fine di estrarre informazioni rilevanti sulle prestazioni, le tendenze e le caratteristiche delle squadre e dei giocatori.

L'analisi dei dati nel calcio offre numerosi vantaggi e opportunità. Innanzitutto, consente di ottenere una visione più oggettiva delle prestazioni di una squadra o di un giocatore, superando le opinioni soggettive e basandosi su dati concreti e misurabili. Inoltre, la Football Data Analysis permette di identificare modelli e tendenze all'interno dei dati, consentendo di prendere decisioni più informate sulle tattiche di gioco, la formazione della squadra e le strategie da adottare.

Essere in grado di fare previsioni scientifiche in ambito calcistico fornisce informazioni chiave per le decisioni tattiche e strategiche delle squadre. La conoscenza delle probabilità di esito delle partite e delle performance individuali dei giocatori influenza infatti le scelte degli allenatori riguardo alla formazione, alle sostituzioni e alle strategie di gioco. Oltre all'impatto sul campo, le previsioni calcistiche hanno anche un ruolo

importante nell'analisi dei dati e nella ricerca scientifica. I *data scientist* e i *data analyst* utilizzano queste previsioni per studiare i modelli di gioco, le tendenze delle squadre e gli schemi tattici, fornendo spunti per migliorare le prestazioni delle squadre, sviluppare nuove strategie e scoprire nuovi talenti. Per queste ragioni, abbiamo deciso di studiare un metodo di previsione di dati calcistici.

Questo lavoro rappresenta il naturale proseguimento del mio tirocinio presso Soccerment, un'azienda italiana leader nel campo della Football Data Analysis. Durante il tirocinio, abbiamo sviluppato tecniche di *Machine Learning* per prevedere metriche avanzate utili all'analisi delle performance delle squadre di calcio. In particolare, ci siamo concentrati sulla previsione prepartita del *field tilt*[1] e sulla costruzione di intervalli di previsione che fornissero una misura dell'incertezza delle nostre stime puntuali. L'obiettivo principale di questo progetto è, quindi, quello di esplorare diversi modelli di Machine Learning per prevedere le statistiche di una partita tra due squadre, utilizzando le statistiche e le cosiddette "metriche avanzate" raccolte dalle partite precedenti delle due squadre durante la stagione. Sono state implementate diverse tecniche per la previsione puntuale e per la costruzione di intervalli di previsione, e gli algoritmi associati sono stati applicati a un dataset che include i dati delle partite delle top 5 leghe europee (Serie A italiana, Premier League inglese, Bundesliga tedesca, LaLiga spagnola, Ligue 1 francese) dalla stagione 2017-2018 alla stagione 2022-2023. I risultati ottenuti durante l'addestramento dei modelli mostrano che CatBoost, un algoritmo di *Gradient Boosting* basato sugli alberi decisionali, è il migliore per la previsione puntuale del field tilt, mentre il metodo di *conformal prediction* Jackknife+ after Bootstrap si è dimostrato il migliore per la costruzione di intervalli di previsione. Tuttavia, dall'applicazione di tali algoritmi a dati nuovi e non usati durante l'addestramento, è evidente che la strada per costruire modelli predittivi sulle statistiche di gara richiede ulteriori sforzi, poiché i risultati ottenuti sono ancora lontani dagli obiettivi prefissati.

Questa tesi è strutturata in sette capitoli.

[1] Il *field tilt* è una generalizzazione del possesso palla, usata per misurare il dominio territoriale di una squadra durante la partita. Per maggiori dettagli si rimanda al Paragrafo 1.1.1.

Il primo capitolo offre un quadro generale delle principali nozioni e concetti legati alla Football Data Analysis. Sono presentati i principali indicatori di prestazione utilizzati e le metriche avanzate, insieme a una breve rassegna dei principali lavori nel campo della Sport Analytics che hanno ispirato il presente studio. Inoltre, viene delineato l'obiettivo principale del lavoro.

Nel secondo capitolo vengono introdotti i concetti fondamentali del Machine Learning, focalizzandosi sul problema della regressione e sui passi principali da seguire nell'approcciarsi a un problema di apprendimento automatico.

Il terzo capitolo è dedicato alla descrizione dettagliata dei modelli di regressione utilizzati. Sono presentati i modelli di regressione lineare, Support Vector Regression e altri algoritmi basati su alberi decisionali come Decision Trees, Random Forest e Gradient Boosting. In particolare, vengono approfondite alcune varianti del Gradient Boosting come XGBoost, LightGBM e CatBoost.

Il quarto capitolo espone tre metodologie per la costruzione di intervalli di previsione: regressione quantile, conformal prediction e conformalized quantile regression. Viene fornita un'illustrazione delle idee di base di ciascun metodo e sono presentati risultati teorici che ne garantiscono l'efficacia.

Nel quinto capitolo è descritto il dataset utilizzato per allenare e testare i modelli di previsione sviluppati, le fasi iniziali di elaborazione dei dati e viene spiegata l'importanza relativa delle variabili per la regressione.

Il sesto capitolo presenta i risultati dell'addestramento dei modelli sui dati noti, mentre nell'ultimo capitolo sono presentati i risultati delle predizioni su nuovi dati mai visti durante la fase di addestramento.

Legenda

Per agevolare la lettura, indichiamo qui le principali notazioni utilizzate e convenzioni seguite:

$\mathbf{z}^T$:= vettore trasposto di $\mathbf{z}$, per $\mathbf{z} \in \mathbb{R}^m$, $m \in \mathbb{N}$

In riferimento ai valori di un dataset, indicheremo

$\mathbf{x}_i$:= vettore riga. Rappresenta i valori delle covariate associate all'i-esima osservazione

$\mathbf{x}^{(j)}$:= vettore colonna. Rappresenta le osservazioni associate alla j-esima variabile

$\mathbf{X}$:= matrice. Rappresenta le variabili indipendenti del dataset

$x_i^{(j)}$:= numero reale. Rappresenta il valore della j-esima variabile associato all'i-esima osservazione

Sia dunque $\mathbf{X} \in \mathbb{R}^{n \times d}$. Sono equivalenti

$$\mathbf{X} = \begin{pmatrix} x_1^{(1)} & \dots & x_1^{(d)} \\ \vdots & \ddots & \vdots \\ x_n^{(1)} & \dots & x_n^{(d)} \end{pmatrix} = \begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{pmatrix} = \left(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(d)} \right).$$

Sia Y una variabile aleatoria e A un insieme boreliano. Indichiamo:

$\mathbb{P}(Y \in A)$:= probabilità che Y assuma valori in A

$\mathbb{E}[Y]$:= valore atteso di Y

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Football Data Analysis

Il calcio odierno (e più in generale lo sport) non può più affidarsi esclusivamente all'istinto e al colpo d'occhio, anche se queste qualità rimangono fondamentali. È necessario adottare un approccio *data driven*, che implica un'analisi approfondita e una visione oggettiva dei problemi basandosi sui dati ed elaborandoli in maniera scientifica.

La pubblicazione nel 2003 di [Lewis (2003)] rappresenta l'introduzione della *Sport Analytics* al vasto pubblico. Il libro (e otto anni dopo il film *Moneyball*) si concentra sulla squadra di baseball degli Oakland Athletics e sul suo direttore generale, Billy Bane ed è stato determinante per divulgare l'analisi statistica del baseball e, più in generale, la Sport Analytics. La dirigenza degli Oakland Athletics creò un vero e proprio vantaggio competitivo utilizzando gli indicatori prestazionali più analitici e innovativi, ottimizzando lo scouting dei giocatori, cosa che permise loro di costruire una squadra che riuscisse a competere con i club più ricchi della Major League Baseball, tanto da riuscire a vincere 20 partite consecutive.

La *Football Analytics* è una branca della Sport Analytics e si occupa di analizzare i dati e le informazioni relative alle partite, alle squadre e ai giocatori al fine di identificare modelli, tendenze e relazioni che possano contribuire a una migliore comprensione del gioco del calcio. Gli analisti di calcio utilizzano le loro competenze per estrarre informazioni significative dai dati e fornire elementi utili per prendere decisioni strategiche e tattiche all'interno del calcio professionistico.

Uno delle principali componenti delle Football Analytics è la *Football Data Analysis*, ossia la scienza che si focalizza sull'elaborazione, la manipolazione e l'interpretazione dei dati specifici del calcio. I *Football Data Analyst* raccolgono e organizzano informazioni provenienti da diverse fonti, come registrazioni delle partite, dati statistici e dati sulle prestazioni dei giocatori. Questi dati vengono poi analizzati utilizzando metodi statistici e tecniche di analisi dei dati al fine di estrarre informazioni rilevanti sulle prestazioni, le tendenze e le caratteristiche delle squadre e dei giocatori.

Oltre all'analisi dei dati, la Football Analytics include anche altre metodologie e approcci, come l'analisi video e le analisi tattiche, che rientrano nel campo della *Match Analysis*, oltre ad altre tecniche di valutazione qualitativa.

Da quando i club dei principali campionati europei hanno iniziato ad abbracciare un approccio analitico e quantitativo, stanno guadagnando un notevole vantaggio competitivo grazie a una gestione più obiettiva delle risorse disponibili. Questa nuova mentalità ha dimostrato di offrire importanti risultati nel miglioramento delle performance sia delle squadre che dei singoli giocatori. I club più all'avanguardia sfruttano la Football Data Analysis principalmente per:

- elaborare strategie e tattiche di gioco;
- valutare le prestazioni dei giocatori e delle squadre;
- pianificare le strategie di calciomercato e la gestione finanziaria della rosa;
- prevenire infortuni e migliorare le prestazioni fisiche e atletiche dei calciatori.

Il calcio europeo offre esempi di società che registrano importanti risultati utilizzando un approccio data driven. Tra queste spiccano il Liverpool e il Manchester City, che si trovano da anni ai vertici del calcio inglese ed europeo, grazie anche alle consistenti risorse economiche a loro disposizione. Tuttavia, il corretto utilizzo dell'analisi dati può essere sufficiente per ottenere ottimi risultati anche senza avere un grande budget a disposizione. Esempi virtuosi sono rappresentati dai club inglesi del Brentford e del Brighton & Hove Albion, dai danesi del Midtjylland e dalla società olandese dell'AZ Alkmaar, che sono stati in grado di competere a livelli che altrimenti sarebbero al di sopra delle loro possibilità, massimizzando i rendimenti nel calciomercato sfruttando metriche avanzate

di analisi dei calciatori.

Soccerment è un'azienda italiana che opera nel campo della Football Analytics, dedicandosi all'elaborazione dei dati calcistici e allo sviluppo di modelli predittivi per fornire informazioni di valore alle squadre di calcio e agli appassionati del gioco. La missione di Soccerment è accelerare l'adozione della *Data Analytics* nel calcio, portando l'approccio basato sui dati dal calcio d'élite fino alla base della piramide calcistica. Questo si basa sull'idea che attraverso l'oggettività dei dati si possa creare maggiore meritocrazia, coinvolgimento e inclusività nel calcio. Ciò consente una visione più approfondita delle prestazioni di una squadra o di un giocatore e permette di prendere decisioni tattiche e strategiche più informate.

La durata fissa degli incontri di calcio (a differenza di sport come il tennis) e la sua natura a punteggio basso (raramente si segnano più di 4-5 gol in una partita) rendono fondamentale comprendere come effettuare l'analisi dei dati nel contesto calcistico. Guardando solo i gol, si capisce poco sulla performance di una squadra. I risultati sono importanti, ma ciò che conta è soprattutto comprendere come essi siano stati raggiunti e quanto siano duraturi nel tempo. Per queste ragioni, è fondamentale studiare le statistiche di gara e sviluppare metriche avanzate che consentano una migliore interpretazione dell'andamento di una partita. Riuscire a predire il comportamento di una squadra in termini di tali metriche può diventare estremamente importante nella preparazione delle partite da parte di uno staff tecnico, poiché consente di prendere decisioni tattiche e strategiche più informate.

L'obiettivo di questo progetto è utilizzare tecniche di Machine Learning per predire le statistiche di una partita prima che venga giocata, basandosi sull'andamento stagionale delle squadre misurato attraverso le metriche avanzate descritte nel Paragrafo 1.1. Oltre alla previsione della statistica, si cerca di fornire un'indicazione dell'incertezza associata a tale previsione, in termini di intervallo di previsione. In particolare, ci si concentra sulla previsione del *field tilt*, una metrica di stile che viene descritta nel dettaglio nel Paragrafo 1.1.1.

Nei prossimi capitoli verranno spiegate le tecniche utilizzate per la previsione puntuale e per la stima dell'intervallo di previsione, verrà presentato il dataset utilizzato per la costruzione dei modelli di Machine Learning e verranno forniti i risultati delle analisi effettuate su tale dataset.

1.1 Field tilt e metriche avanzate Soccerment

In questo paragrafo andiamo a descrivere brevemente la statistica principale su cui ci siamo soffermati nel corso del lavoro, ossia il *field tilt* e spieghiamo alcune metriche avanzate sviluppate da Soccerment che sono state utilizzate per affrontare il problema della previsione delle performance di gara. Queste metriche sono:

- gli *Expected Goals*, o **xG**;
- gli *Expected Points*, o **xP**;
- la *Gegenpressing Intensity* e la *Gegenpressing Efficiency*, o **GPI** e **GPE**;
- gli *one-twos*, o triangolazioni.

1.1.1 Field tilt

Il field tilt è una statistica che nasce come evoluzione del concetto di possesso palla, una statistica che tiene conto della percentuale di passaggi effettuati da una squadra rispetto al totale dei passaggi compiuti durante la partita. Tuttavia, questo dato può risultare fuorviante: se una squadra compie un grande numero di passaggi in zone del campo lontane dalla porta avversaria, come la propria area di rigore o il centrocampo, allora avere un alto possesso di palla non fornisce alcuna informazione riguardo alla pericolosità della squadra. Da questa considerazione, nasce il calcolo del field tilt.

Dividiamo il campo in tre bande verticali uguali, come nella Figura 1.1. Nell'immagine, consideriamo una squadra che attacca da sinistra verso destra. Con questa suddivisione, possiamo identificare per entrambe le squadre in campo tre settori: uno difensivo, uno centrale e uno offensivo. Chiaramente, il terzo difensivo della squadra A

coincide con il terzo offensivo della squadra B e viceversa. Consideriamo ora solo i passaggi riusciti dalle due squadre, in cui il ricevitore del pallone si trova nel terzo di campo offensivo (relativamente alla propria squadra), escludendo da tali passaggi le rimesse laterali, i rinvii dei portieri e i cross. Chiamiamo i passaggi rimanenti, per semplicità, "passaggi offensivi".

Figura 1.1: Suddivisione del campo per il calcolo del field tilt

Definizione 1.1.1. Siano p^A e p^B il numero di passaggi "offensivi" compiuti, rispettivamente, dalla squadra A e dalla squadra B. Definiamo il field tilt delle due squadre come le quantità

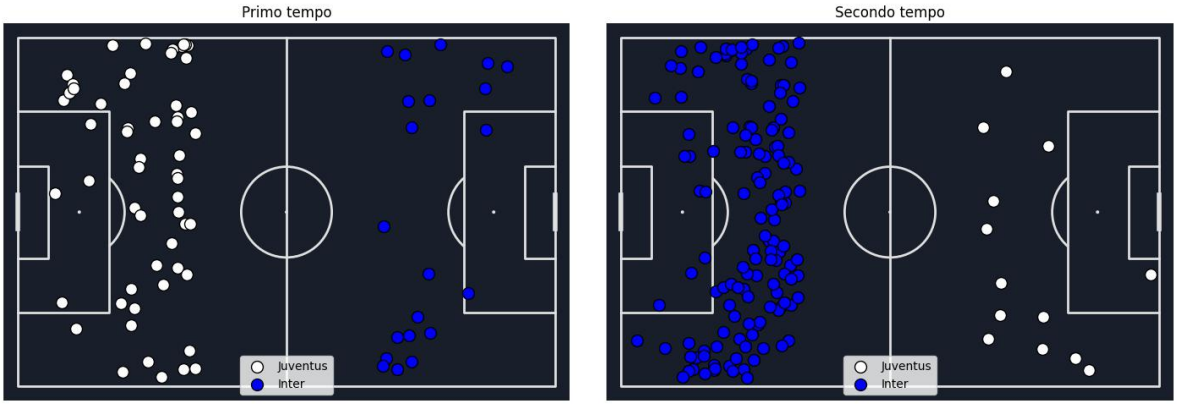
$$ft^A := \frac{p^A}{p^A + p^B} * 100; \qquad ft^B := \frac{p^B}{p^A + p^B} * 100.$$

Vale a dire, il field tilt misura la percentuale dei passaggi "offensivi" corrispondente a ciascuna squadra.

Esempio 1.1. Nella Figura 1.2 sono mostrati i passaggi "offensivi" effettuati nei due tempi della partita Juventus-Inter del 15/05/2021. I pallini bianchi corrispondono ai passaggi della Juventus, mentre i pallini blu a quelli dell'Inter.

Vediamo che nel primo tempo la Juventus ha effettuato più passaggi "offensivi" dell'Inter (56 contro 21). Al contrario, nel secondo tempo è stata l'Inter a dominare il gioco, producendo 138 passaggi "offensivi" contro i soli 13 della Juventus. Sommando i dati abbiamo che $p^{Juventus} = 69$, mentre $p^{Inter} = 159$. Con questi numeri, otteniamo che il field tilt delle due squadre è:

$$ft^{Juventus} = 30.3\%; \qquad ft^{Inter} = 69.7\%.$$



(a) Primo tempo (b) Secondo tempo

Figura 1.2: Passaggi svolti nel terzo di campo offensivo nella partita Juventus-Inter

Il field tilt fornisce dunque una misura del predominio tattico di una squadra rispetto all'avversario: avere un field tilt alto significa riuscire a effettuare molti più passaggi rispetto all'avversario nella zona più pericolosa del campo. Generalmente, valori alti di field tilt caratterizzano squadre che adottano uno stile di gioco molto offensivo e che possono contare su giocatori di qualità, capaci quindi di passarsi efficacemente il pallone in zone del campo dove la pressione della difesa avversaria è maggiore.

1.1.2 xG

Gli Expected Goals (**xG**) sono un concetto centrale della Football Analytics e sono la statistica avanzata più utilizzata e conosciuta nel mondo del calcio. Gli xG rappresentano una misura statistica che assegna a ogni tiro una probabilità di essere convertito in rete. Questi valori vengono calcolati utilizzando un modello che considera diversi fattori, come la posizione del tiro, l'angolo di tiro, la distanza dalla porta, il tipo di assist (passaggio filtrante, cross, ecc.), il punteggio, il piede (o parte del corpo) utilizzato per il tiro, la situazione di gioco (azione, calcio piazzato, ecc.), la presenza di difensori e molti altri parametri. Nel caso dei calci di rigore, che sono eventi semplici con condizioni prefissate, il valore di xG è una costante pari al rapporto tra i rigori segnati e i rigori calciati, che nel dataset di Soccerment è 0.78.

Gli xG forniscono una metrica oggettiva per valutare la qualità delle occasioni da gol create da una squadra o da un giocatore. A ciascuna occasione viene assegnato un valore di xG compreso tra 0 e 1, dove un valore più vicino a 1 indica una maggiore probabilità di segnare. Ad esempio, se a un tiro è associato un valore di xG pari a 0.2, significa che il modello stimato una probabilità del 20% che il tiro si concretizzi in gol.

È importante sottolineare, tuttavia, che gli xG non devono essere considerati come una probabilità esatta di segnare su singolo tiro. Solo analizzando un grande numero di dati, ad esempio l'insieme di tutti i tiri effettuati in un'intera stagione, si può ottenere una corrispondenza approssimata, quindi un significativo potere predittivo, nel confrontare gli xG e il numero effettivo di gol realizzati. Pertanto, a livello di singolo tiro, gli xG vanno interpretati come una misura della qualità dell'occasione da gol.

Gli xG possono anche essere utilizzati per analizzare le prestazioni delle squadre nel corso della stagione, poiché una serie di performance caratterizzate da un alto numero di xG ci dicono che nel lungo termine ci si può aspettare che la squadra ottenga buoni risultati.

Per quanto riguarda i singoli giocatori, il valore medio di xG per tiro permette di valutare l'efficienza di un attaccante, indicando quanto il giocatore è selettivo nel decidere se e quando tirare. Degli xG per tiro sotto la media ci possono dire di un giocatore, ad esempio, che tenta molti tiri dalla distanza. Viceversa, valori alti di xG per tiro sono tipici di un attaccante che tira soprattutto dall'interno dell'area di rigore.

Gli Expected Goals sono quindi uno strumento prezioso per gli addetti ai lavori nel processo decisionale, nel reclutamento dei giocatori e nella valutazione complessiva delle prestazioni di una squadra.

Esempio 1.2. La Figura 1.3 mostra i gol e tiri con xG maggiore di 0.25 dell'attaccante Polacco Robert Lewandowski nella stagione 2020-2021, esclusi i rigori. Una maggiore dimensione dei pallini indica un alto valore di xG del tiro. Il calciatore ha totalizzato 33 gol, a fronte di un'attesa, calcolata in termini di xG, di 24.64. Questo significa che l'attaccante è stato estremamente bravo a convertire in rete anche tiri che normalmente sono più difficili da segnare. La sigla "P90" indica la media di xG calcolata ogni 90 minuti di gioco, ossia il tempo regolamentare di una partita.

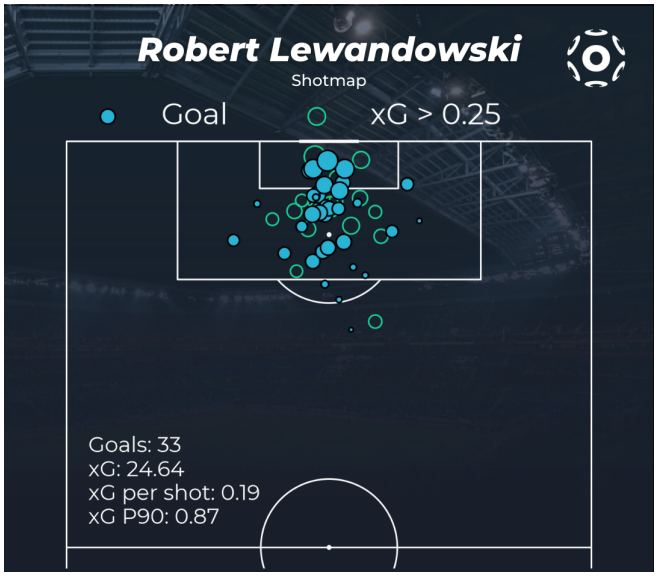


Figura 1.3: Shotmap Robert Lewandowski, stagione 2020-2021. Fonte: Soccerment Analytics

1.1.3 xP

Strettamente collegati agli xG ci sono gli Expected Points (**xP**). Come abbiamo visto, i modelli di Expected Goals ci permettono di quantificare la qualità delle singole occasioni di tiro. Questo ci offre la possibilità di valutare le probabilità che una squadra vinca, perda o pareggi una partita, basandoci sugli xG totali delle due squadre durante la partita. A loro volta, queste probabilità possono essere tradotte in un valore di stima predittiva per i punti ottenuti in ogni partita, a cui diamo il nome di Expected Points.

L'idea di base del modello di Expected Points è piuttosto semplice. Per prima cosa, facciamo attenzione a non considerare tutti i tiri effettuati durante una partita. Ad esempio, se più di un tiro avviene nella stessa azione, potrebbe accadere che la somma dei loro valori di xG sia maggiore di 1, il che rappresenterebbe una probabilità di segnare superiore al 100% durante l'azione, cosa che è ovviamente impossibile. D'altra parte, se uno qualsiasi dei tiri effettuati in rapida successione avesse trovato la rete, gli altri non sarebbero nemmeno esistiti. Pertanto, è necessario strutturare gli eventi della partita in *sequenze* e *possessi*.

Definiamo una sequenza come una serie di eventi durante i quali il possesso del pallone

appartiene sempre alla stessa squadra. Ad esempio, una serie di passaggi che inizia con un rinvio del portiere e termina con un tiro respinto in calcio d'angolo, senza che nel frattempo la squadra avversaria entri in possesso della palla, costituisce una sequenza.

Un possesso, a sua volta, è una serie di sequenze che non vengono interrotte da un cambio di possesso. Nell'esempio precedente, il calcio d'angolo successivo avvia una nuova sequenza, ma continua lo stesso possesso.

Questa strutturazione degli eventi della partita ci permette di calcolare la probabilità di segnare non basandoci sui valori di xG dei singoli tiri, ma su tutti gli xG di un possesso. È importante aggregare i dati per possessi, invece che per sequenze, poiché tiri successivi nello stesso possesso non avverrebbero se uno qualsiasi dei precedenti portasse a un gol. Ora, ripensiamo all'esempio del calcio d'angolo e supponiamo che il corner porti a un tiro in porta: questo tiro non avverrebbe se il tiro precedente, che ha originato il calcio d'angolo e concluso la propria sequenza di appartenenza, avesse trovato la rete.

Fatte queste considerazioni, calcolare le probabilità di segnare in un possesso è molto semplice. Per prima cosa, indichiamo con G il verificarsi di un gol nel corso del possesso, mentre G^c indica che nessuno dei tiri effettuati durante il possesso venga trasformato in rete. La probabilità che in un possesso ci sia un gol viene calcolata indirettamente come

$$\mathbb{P}(G) = 1 - \mathbb{P}(G^c).$$

Di conseguenza, ci riduciamo a calcolare la probabilità che nessuno dei tiri che avvengono durante un possesso sia gol. Questa probabilità viene calcolata sfruttando il valore di xG di ciascun tiro.

Supponiamo che in un possesso si verifichino T tiri totali. Si ha

$$\mathbb{P}(G^c) = \prod_{t=1}^{T}(1 - \mathrm{xG}_t),$$

dove xG_t indica il valore di xG del t-esimo tiro.

Questa formula riflette il fatto che ogni tiro è possibile solo se quelli precedenti non hanno portato a un gol.

A questo punto, la probabilità di segnare durante il possesso viene utilizzata per effettuare migliaia di simulazioni, seguendo una distribuzione di Poisson. A partire da

questa distribuzione e considerando tutti i possessi di entrambe le squadre all'interno della partita, si derivano le probabilità di vittoria, pareggio e sconfitta.

Siano quindi P_v e P_p le probabilità che la squadra A vinca o pareggi. Chiaramente, la probabilità P_s di una sconfitta si ottiene come $P_s = 1 - P_v - P_p$. Inoltre, la probabilità di vittoria della squadra A è uguale alla probabilità di sconfitta della squadra B, mentre le probabilità di pareggio per le due squadre sono uguali. Poiché nel calcio la vittoria vale 3 punti, il pareggio 1 e la sconfitta 0, gli xP della squadra A risultano essere

$$\text{xP}^A = 3 * P_v + 1 * P_p + 0 * P_s = 3 * P_v + P_p.$$

Per le considerazioni fatte precedentemente, abbiamo che

$$\text{xP}^B = 3 * (1 - P_v - P_p) + P_p = 3 - 3 * P_v - 2 * P_p.$$

Il calcolo degli Expected Points può essere utile per valutare in maniera oggettiva l'andamento di una squadra, soprattutto nel lungo periodo. Valori alti di xP ci dicono che la squadra ha "meritato" di ottenere risultati utili, indipendentemente dal punteggio finale.

Esempio 1.3. Con la Figura 1.4 possiamo vedere un confronto tra la media di punti a partita e di xP a partita delle squadre di Serie A nelle prime 15 giornate del campionato 2022-2023.

Notiamo immediatamente che il Napoli, oltre ad essere in testa alla classifica, è la squadra che ha totalizzato fino a quel momento il maggior numero di xP, confermando il dominio che la squadra partenopea ha esercitato fino alla fine di un campionato meritatamente vinto. Squadre come Milan, Juventus o Udinese, che si trovano al di sopra della linea azzurra, hanno totalizzato più punti di quanto previsto dal modello di xP. Questo potrebbe significare che hanno vissuto un periodo di forma eccezionale o fortunato, e probabilmente la flessione di risultati che ha caratterizzato queste squadre nella seconda parte di stagione può essere spiegata come una perdita dello stato di grazia in cui si trovavano.

Un'analisi interessante può essere effettuata per le tre squadre nell'angolo in basso a sinistra, ossia Cremonese, Sampdoria e Hellas Verona, le quali hanno ottenuto nella prima parte di stagione molto meno punti di quanto avrebbero meritato sul campo. Mentre gli *Scaligeri* nella seconda parte del campionato hanno cominciato a raccogliere,

in termini di punti, quanto seminato a livello di prestazioni (e quindi di xP), Cremonese e Sampdoria non sono mai riusciti a concretizzare le prestazioni in risultati, andando incontro a un'inevitabile retrocessione.

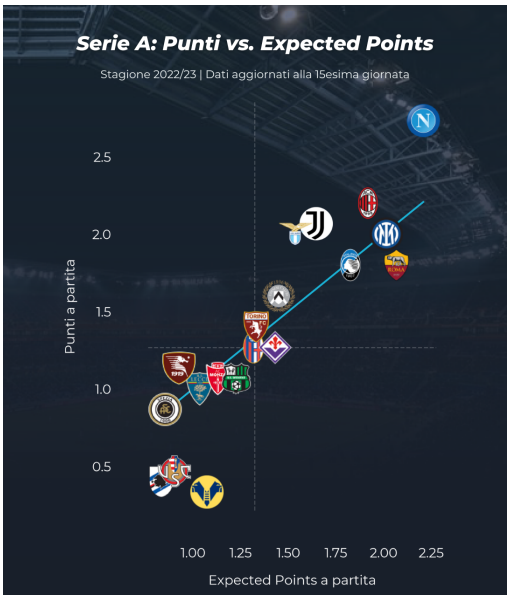


Figura 1.4: Punti vs xP, Serie A stagione 2022-2023. Fonte: Soccerment Analytics

1.1.4 GPI e GPE

Il pressing è una strategia chiave per molte squadre di successo nel calcio moderno. La misurazione di questa componente del gioco è sempre stata una sfida nell'ambito dell'analisi basata sui dati degli eventi. Una particolare tipologia di pressing è chiamata *gegenpressing*, o contro-pressing. Questi termini si riferiscono alla reazione di una squadra o di un giocatore alla perdita del possesso durante la fase di attacco: gegenpressing significa pressare immediatamente l'avversario per recuperare il pallone il più velocemente possibile, invece di tornare indietro e consentire all'avversario di consolidare il possesso e avviare una fase di costruzione controllata. Questa è una strategia ben consolidata in molte squadre basate sul possesso palla, perché, se viene effettuata con successo, consente di controllare il pallone nelle zone d'attacco contro una difesa disorganizzata, una situazione simile a un contropiede, ma che inizia già in zone avanzate.

Per definire le azioni di contro-pressing, si considerano le perdite di possesso (*turnover*) nell'ultimo 40% dell'area d'attacco del campo (in rosso nella Figura 1.5 a sinistra) e si tiene conto solo delle azioni difensive eseguite nella metà campo d'attacco nei successivi sei secondi.

Con azione difensiva si tengono in considerazione il maggior numero possibile di situazioni di contropressione, come falli, contrasti, intercetti, respinte e disimpegni, ma anche altri eventi che indicano che la squadra che ha appena perso il pallone sta cercando attivamente di recuperarlo: duelli aerei, duelli al 50-50, passaggi, dribbling e tiri.

Inoltre, vengono considerate come contro-pressing di successo quelle situazioni in cui c'è un passaggio errato che parte dalla metà campo della squadra sotto pressione (in verde nella Figura 1.5 a destra) e si verifica un recupero del possesso nella metà campo della squadra in pressing (ad esempio, dopo un lungo lancio) entro la soglia di tempo. Questo consente di registrare casi in cui l'azione di contropressione nella zona d'attacco non si traduce in un evento esplicito sul pallone e quindi non viene registrata nei dati degli eventi, ma costringe l'avversario a liberarsi del pallone con un passaggio lungo o impreciso.

(a) Turnover zone (b) Gegenpressing zone

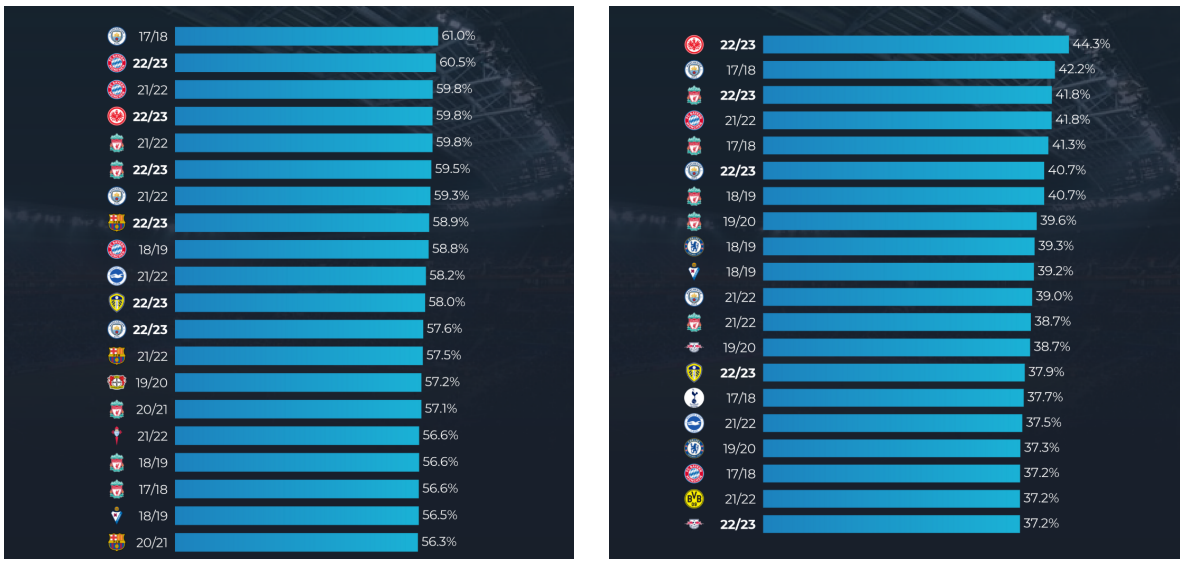
Figura 1.5: Zone del campo in cui si tengono in considerazione le contropressioni

La Gegenpressing Intensity (**GPI**) è una metrica di stile sviluppata da Soccerment che rappresenta la percentuale di volte in cui una squadra cerca immediatamente di ricon-

quistare il pallone tramite una contropressione dopo averlo perso nelle zone d'attacco, invece di ritirarsi indietro.

La GPI considera i tentativi di recupero nei primi sei secondi indipendentemente dall'esito di tale riaggressione, mentre la percentuale di contropressioni considerate riuscite, ossia quelle che portano con successo al recupero del pallone, consente di calcolare una metrica di efficienza del gegenpressing strettamente legata alla GPI, chiamata Gegenpressing Efficiency (**GPE**).

Nella Figura 1.6 mostriamo le prime 20 squadre nelle top 5 leghe europee (Italia, Inghilterra, Germania, Spagna, Francia) per GPI e GPE dalla stagione 2017-2018 alla stagione 2022-2023. Come prevedibile, troviamo in testa a entrambe le classifiche squadre note per il loro aggressivo contro-pressing, come il Manchester City, il Bayern Monaco e il Liverpool. I *Reds*, spesso considerati la prima squadra a trasformare il gegenpressing in una strategia vincente sotto la guida dell'allenatore tedesco Jürgen Klopp, compaiono più frequentemente, figurando tra le prime 20 squadre in entrambe le categorie in cinque delle ultime sei stagioni.



(a) GPI (b) GPE

Figura 1.6: Ranking GPI e GPE delle ultime 6 stagioni nelle top 5 leghe europee. Fonte: Soccerment Analytics

1.1.5 One-twos

La progressione della palla e la creazione di opportunità sono fasi chiave nel gioco del calcio e possono essere raggiunte in molti modi diversi, che sia attraverso passaggi corti e pazienti o lanci diretti lunghi. I passaggi in uno-due, o più semplicemente triangolazioni, sono combinazioni tra due giocatori, in cui il primo (che apre il passaggio in uno-due) scambia la palla con un compagno di squadra (che chiude il passaggio in uno-due), guadagnando terreno con movimenti senza palla tra i due passaggi. Questo tipo di azione è caratterizzata da un elevato livello di coordinazione tra i due giocatori e un'ottima capacità di interpretare e reagire alle intenzioni e ai movimenti reciproci.

Soccerment ha creato una metrica in grado di rilevare le triangolazioni a partire dai dati degli eventi. L'idea di base è abbastanza semplice: vengono inizialmente selezionati tutti i passaggi completati in *open play* (cioè escludendo palle inattive) che sono seguiti da un altro passaggio completato della stessa squadra e ricevuto dallo stesso giocatore che ha effettuato il primo passaggio. Successivamente, vengono applicate una soglia di progressione e una soglia temporale. Vengono considerati solo quegli scambi in cui la progressione tra le coordinate di partenza del primo passaggio e le coordinate d'arrivo del secondo passaggio avvicinano il giocatore iniziale al centro della porta o alla linea di porta di almeno il 25% della distanza iniziale. Ad esempio, se il giocatore che avvia la triangolazione si trova a 40 metri dalla porta, affinché lo scambio sia considerato un uno-due, lo stesso giocatore deve ricevere indietro il pallone a non meno di 30 metri dalla porta. Allo stesso tempo, per etichettare l'evento come uno-due, viene richiesto che tra il primo e il secondo passaggio non passino più di quattro secondi. Infine, per isolare ulteriormente le triangolazioni come eventi in cui il secondo giocatore è principalmente statico e semplicemente favorisce la progressione del primo giocatore, vengono scartati tutti i passaggi in cui il secondo giocatore porta il pallone per più di cinque metri prima di restituirlo al compagno. I passaggi rimanenti vengono conteggiati nella metrica degli **one-twos**.

I passaggi in uno-due sono un'azione di valore in termini di creazione di opportunità principali e secondarie, poiché portano a passaggi chiave con una percentuale significativamente più alta rispetto ai passaggi generici.

1.2 Lavori precedenti

In questo paragrafo presentiamo brevemente i principali articoli di argomento Football Analytics da cui abbiamo preso spunto per procedere con il nostro lavoro.

[Dixon, Coles (1997)] è un punto di riferimento dell'applicazione della Football Analytics per previsioni. L'articolo si concentra sulla modellazione dei punteggi nel calcio e sull'identificazione delle inefficienze nel mercato delle scommesse sul calcio. Gli autori utilizzano metodi di modellazione statistica per analizzare i punteggi delle partite e valutare le opportunità di scommessa che possono presentarsi a causa di inefficienze o discrepanze tra i punteggi attesi e quelli offerti dai bookmaker.

Gli autori di [Hughes, Bartlett (2002)] esaminano e discutono l'importanza di identificare correttamente gli *indicatori di performance* e forniscono raccomandazioni generali sull'uso e l'applicazione di tali indicatori al fine di valutare le prestazioni individuali e di squadra in numerosi sport. Con "indicatori di performance" si intende una selezione, o una combinazione, di caratteristiche che definiscono alcuni o tutti gli aspetti di una prestazione; possono essere indicatori biomeccanici, tattici, tecnici o relativi all'incontro. L'articolo sottolinea che il successo o il fallimento di una prestazione devono essere contestualizzati sia rispetto all'avversario che alle prestazioni precedenti della squadra o dell'individuo. Per ottenere un'interpretazione completa e obiettiva dei dati derivanti dall'analisi di una prestazione, è necessario confrontare i dati raccolti con i dati aggregati di un gruppo di squadre o individui che competono a uno standard appropriato. Inoltre, qualsiasi analisi della distribuzione delle azioni sul campo deve essere normalizzata rispetto alla distribuzione totale delle azioni nell'area. Se espressi come rapporti adimensionali, gli indicatori di performance hanno il vantaggio di essere indipendenti dalle unità di misura utilizzate; inoltre, sono implicitamente indipendenti da ogni altra caratteristica. Infine, gli autori forniscono alcuni suggerimenti relativi alle migliori metodologie di lavoro con le varie categorie di indicatori:

- **Classificazione della partita**: confrontare sempre gli indicatori con i dati degli avversari e, quando possibile, con i dati aggregati delle prestazioni degli omologhi;
- **Biomeccanica**: confrontare i dati con le prestazioni precedenti e con quelle dei membri della propria squadra e degli avversari. Oltre alla presentazione dell'analisi

dei dati originali, considerare di presentare i dati normalizzati quando esiste un valore massimo o globale che è rilevante o quando sono necessari confronti tra singoli giocatori o tra le prestazioni effettuate nel tempo da uno stesso giocatore;

- **Aspetti tecnico tattici**: normalizzare sempre le caratteristiche rispetto alla frequenza totale.

In [Baboota, Kaur (2019)], viene presentato un approccio basato sul Machine Learning per la previsione e la modellazione degli esiti (vittoria, pareggio, sconfitta) delle partite del campionato di calcio inglese Premier League. L'articolo si focalizza sulla selezione delle variabili più significative (*feature engineering*) e sull'utilizzo di algoritmi di Machine Learning per analizzare i dati storici delle partite e sviluppare modelli predittivi per prevedere i risultati futuri. Viene sottolineata l'importanza cruciale di tenere in considerazione il "fattore campo", ovvero se una squadra gioca in casa o in trasferta. Nel nostro lavoro abbiamo quindi deciso di analizzare le statistiche che generano le nostre variabili distinguendo i dati raccolti per le partite giocate in casa e quelle in trasferta. Un altro aspetto fondamentale rilevato dagli autori è la convenienza di calcolare le statistiche da utilizzare in modo indipendente per ogni stagione, senza ereditare i valori dalle stagioni precedenti. La ragione di ciò è attribuibile alla discontinuità che spesso caratterizza la fine di una stagione e l'inizio della successiva, a causa della lunga sosta estiva (generalmente da 2 a 3 mesi), dei cambiamenti nelle rose delle squadre (calciomercato) e degli allenatori, che portano a modifiche negli stili di gioco e nella forza delle squadre.

Una strategia innovativa viene proposta in [Berrar et al. (2019)], dove viene esplorato il modo in cui il sapere specifico in ambito calcistico può essere integrato nell'Machine Learning per migliorare le previsioni degli esiti delle partite di calcio. Gli autori sottolineano l'importanza di combinare l'esperienza umana nel calcio con le tecniche di Machine Learning per ottenere previsioni più accurate sugli esiti delle partite. Pertanto, una strategia efficace potrebbe essere quella di unire la scienza all'esperienza degli analisti. Gli autori suggeriscono che sia opportuno considerare, oltre al fattore campo e alla forza dell'avversario (come già suggerito in [Baboota, Kaur (2019)] e [Hughes, Bartlett (2002)]), caratteristiche che diano un'indicazione riguardo a:

- **Performance offensiva**, descrivendo la capacità di una squadra di segnare gol;

- **Performance difensiva**, descrivendo la capacità di una squadra di prevenire gol avversari;
- **Performance recente**, caratterizzando la condizione attuale di una squadra in termini di prestazioni aggregate nelle partite recenti.

Nel tenere in considerazione queste caratteristiche, gli autori mettono in luce alcune problematiche nel processo di modellazione. Ad esempio, non è banale decidere quante partite tenere in considerazione nel valutare la condizione di una squadra nel tempo. Inoltre, all'inizio di una stagione le squadre iniziano con zero punti e senza uno storico; quindi è necessario aspettare che ogni squadra giochi un certo numero di partite prima di avere un registro significativo delle prestazioni passate. Questo porta a perdere un considerevole numero di dati riferiti alle prime partite della stagione, poiché risultano poco indicativi sul reale andamento della squadra. Considerazioni di questo tipo ci hanno portato a tenere conto, come variabili predittive, dell'andamento stagionale delle squadre, ma solo a partire dalla quarta giornata di ciascun campionato. Gli autori sollevano un ultimo problema, legato al finale di stagione: alcune partite potrebbero non essere pienamente competitive poiché alcune squadre non hanno più obiettivi da raggiungere. Pertanto, le caratteristiche predittive basate su queste partite potrebbero essere meno affidabili.

La combinazione tra scienza ed esperienza umana per prevedere gli esiti delle partite di calcio viene riproposta in [Beal et al. (2021)]. Gli autori sottolineano che la performance delle squadre dipende solitamente non solo dalle loro abilità, ma anche dall'ambiente in cui operano. Consideriamo il caso di una squadra di alto livello che si ritrova a giocare contro una delle peggiori squadre della stessa lega. Il fatto che quest'ultima stia affrontando la lotta per non retrocedere in una lega inferiore potrebbe fornirle una motivazione extra per vincere la partita. Data questa situazione, in molti casi la performance della squadra più forte potrebbe non essere facilmente prevedibile. Ulteriore attenzione bisogna porre nel considerare lo storico delle performance di una squadra per valutarne il momento di forma. Infatti, il rendimento storico potrebbe non essere utile quando la prestazione della squadra dipende da fattori dinamici come la condizione umana (morale, infortuni, strategie) o variabili ambientali (meteo, contesto della competizione, umore pubblico) ed è molto improbabile che i modelli di Machine Learning siano in grado di

identificare tali fattori e tenerne conto durante l'analisi. Per questa ragione, viene proposto un approccio originale: utilizzare i prodotti dei media (giornali, televisioni, social network) nel tentativo di migliorare l'accuratezza dei metodi scientifici per la previsione dell'esito delle partite. Alcuni esempi di ciò potrebbero derivare da articoli che discutono la possibile formazione della squadra e se un allenatore sia intenzionato a fare ampie rotazioni (*turnover*) o se una squadra ha ingaggiato un nuovo allenatore/giocatore. Seguendo questa strategia, nell'articolo viene dimostrato che incorporando i fattori umani nel modello, anziché utilizzare solo le statistiche di base sulla performance, è possibile migliorare l'accuratezza.

Capitolo 2

Machine Learning

In questo capitolo andremo a presentare i concetti fondamentali del Machine Learning. Vedremo i principali problemi di Machine Learning e ci concentreremo in particolare sui problemi di regressione.

Il Machine Learning (o apprendimento automatico) è una branca dell'Intelligenza Artificiale che si concentra sullo sviluppo di sistemi capaci di assimilare in maniera autonoma informazioni, a partire dai dati ricevuti in input, tramite la costruzione di algoritmi che permettono di elaborare nuove informazioni alla luce di quelle apprese in precedenza. Una delle più recenti definizioni di Machine Learning è quella di T. M. Mitchell, professore dell'Università di Carnegie Mellon:

> *Si dice che un programma impara da una certa esperienza E rispetto a una classe di compiti T ottenendo una performance P, se la sua performance nel realizzare i compiti T, misurata da P, migliora con l'esperienza E.*

Dunque, un programma è in grado di apprendere dall'esperienza se, nel realizzare un compito, le sue prestazioni migliorano durante il periodo di tempo in cui si ripete la stessa attività (fase di *training*).

Le prime sperimentazioni per la realizzazione di macchine in grado di apprendere in modo automatico risalgono all'inizio degli anni '50 del Novecento, quando gli studiosi cominciarono a ipotizzare metodi statistici per le macchine per renderle in grado di prendere decisioni tenendo conto delle probabilità di accadimento di un evento.

Il primo a parlare di Intelligenza Artificiale fu Alan Turnig, il quale nell'ottobre 1950 in [Turing (1950)] sollevò la questione se le macchine potessero pensare.

Da quando i computer sono diventati sufficientemente potenti, la ricerca ha preso slancio. Tra le creazioni più note ci sono **Deep Blue** di IBM, che ha battuto il campione mondiale di scacchi Garry Kasparov nel 1997 e **AlphaGo** di Google, che ha battuto il maestro di *Go* Lee Sedol nel 2016.

La Figura 2.1 mostra la differenza tra la programmazione tradizionale e il Machine Learning:

- Nella **programmazione tradizionale** tutta la conoscenza sul mondo è codificata all'interno del programma; specificando tutti i casi, il processo di elaborazione risulta semplificato. Nel caso in cui una regola non sia stata descritta, tuttavia, la macchina non sarà in grado di svolgere il compito;
- Nel **Machine Learning** alla macchina vengono passati esempi, a partire dai quali costruisce le regole che li descrivono e capisce autonomamente se un nuovo caso risponde o meno alla regola che ha ricavato. Quindi 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 risulta poco conveniente.

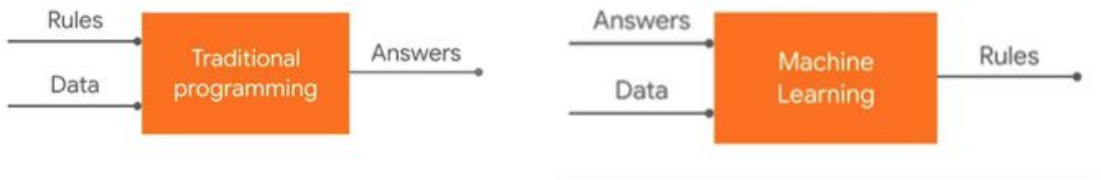


Figura 2.1: Programmazione tradizionale VS Machine Learning

2.1 Panoramica sui metodi di Machine Learning

I metodi di apprendimento automatico possono essere suddivisi in tre categorie principali:

- **Supervised Learning**: durante la fase di addestramento vengono forniti al modello una serie di dati con il relativo risultato (*labeled*), l'obiettivo è quello di apprendere la relazione che lega i valori di input con quelli di output;
- **Unsupervised Learning**: nella fase di addestramento vengono forniti al modello una serie di dati senza alcuna indicazione riguardo al risultato associato (*unlabeled*) e il sistema dovrà risalire a schemi o pattern nascosti, cercando di identificare negli input una struttura logica senza che questi siano stati in precedenza etichettati;
- **Reinforcement Learning**: il sistema apprende attraverso l'interazione con un ambiente dinamico, ricevendo feedback in forma di ricompense o punizioni per le sue azioni. L'obiettivo dell'agente è massimizzare le ricompense totali nel lungo termine.

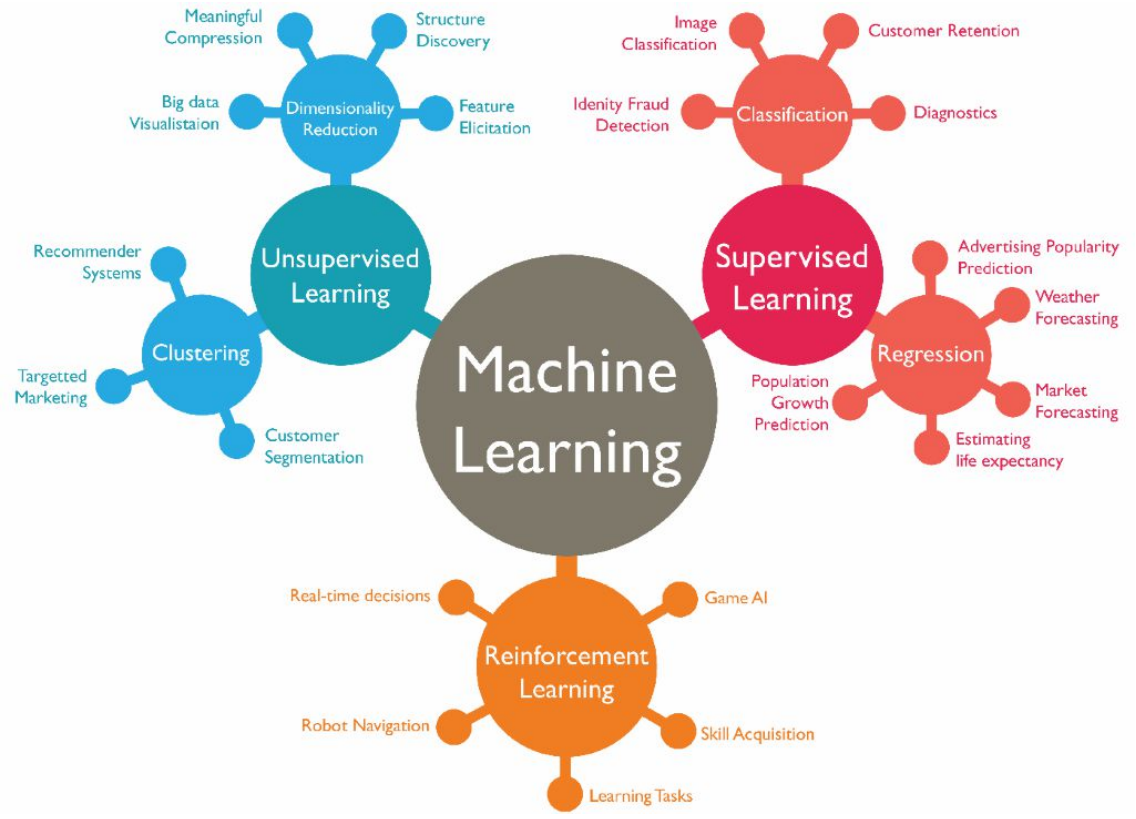


Figura 2.2: Modelli di Machine Learning

Supervised Learning

Vengono forniti all'algoritmo coppie di input (dati raccolti) e di output (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.

Gli algoritmi di apprendimento supervisionato possono essere usati per compiti di classificazione e di regressione, a seconda che i valori da prevedere siano, rispettivamente, discreti o continui:

- Classificazione: l'algoritmo è allenato con dati appartenenti ciascuno a una classe. Analizzandoli ricava una regola generale grazie alla quale classificherà un nuovo caso non etichettato;
- Regressione: l'algoritmo prevede un certo valore numerico dopo essersi allenato su dati input-output chiamati predittori e target.

Un esempio molto noto di classificazione riguarda i passeggeri del Titanic: il dataset è costituito da dati riguardanti ciascun passeggero, quali età, sesso, prezzo del biglietto, sito di imbarco e molti altri; queste caratteristiche sono le variabili predittive. È inoltre presente una variabile che indica se il passeggero è sopravvissuto oppure no. L'algoritmo di Machine Learning applicato a questo problema dovrà cercare di prevedere la classe di appartenenza di ciascun passeggero, cioè se sia sopravvissuto o meno al disastro, a partire dai valori delle variabili predittive.

Unsupervised Learning

Il dataset di addestramento non è etichettato, cioè non si hanno output per gli input forniti. 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*).

Le tecniche di apprendimento non supervisionato lavorano confrontando i dati e cercando similarità o differenze fra essi, senza averne una conoscenza preliminare.

Tra le applicazioni più importanti in questa categoria troviamo il *Clustering* e la riduzione di dimensionalità:

- Clustering: l'algoritmo individua dei gruppi (*clusters*) fra i dati accorpando quelli con caratteristiche simili e trovando, così, delle classi per un problema che inizialmente non le fornisce. Spesso il numero di clusters non è noto a priori;

- Riduzione di dimensionalità: l'algoritmo riduce il numero dei pattern in input (i.e. il numero di variabili) senza perdere le informazioni più importanti per il problema. Risulta molto utile per problemi con dimensioni molto elevate, scartando quelle ridondanti o instabili.

Soccerment ha sviluppato un modello di Clustering applicato al calcio. Utilizzando statistiche dettagliate dalla stagione 2017/18 alla 2020/21, è stata applicata un'analisi di Machine Learning per classificare i giocatori in base alla loro funzione in campo. Sulla base di metriche avanzate l'algoritmo ha suddiviso i giocatori in tredici gruppi che permettono di analizzare le caratteristiche dei giocatori andando oltre al classico ruolo che identifica solamente la posizione in campo (terzino, mezzala, centravanti, ecc.).

Reinforcement Learning

Il Reinforcement Learning ha come obiettivo l'apprendimento di un comportamento ottimale a partire dalle esperienze passate. Man mano che la macchina esplora il dominio del problema gli vengono restituiti dei feedback in modo da poterla indirizzare verso la soluzione migliore: nel caso in cui il sistema riesca a raggiungere un obiettivo, vengono fornite ricompense, se invece vengono commessi degli errori, allora si restituiscono punizioni.

Immaginiamo un agente virtuale che impara a giocare a un videogioco di corsa automobilistica. L'obiettivo dell'agente è quello di imparare a guidare l'auto nel modo più veloce possibile e completare il percorso senza incidenti.

Inizialmente, l'agente non sa nulla sul gioco e su come guidare l'auto. Utilizzando l'algoritmo di Reinforcement Learning, l'agente esplora l'ambiente di gioco e impara dalle sue azioni. L'ambiente di gioco restituisce all'agente uno stato corrente (ad esempio, la posizione dell'auto sulla pista), e l'agente prende una decisione su quale azione intraprendere (ad esempio, accelerare, frenare, sterzare a sinistra o a destra).

All'inizio, l'agente può prendere decisioni casuali e sperimentare diverse azioni. Tuttavia, l'ambiente di gioco fornisce un feedback sotto forma di ricompense. Ad esempio, se l'auto rimane sulla pista senza incidenti, l'agente riceve una ricompensa positiva. Al contrario, se l'auto esce di strada o collide con gli ostacoli, l'agente riceve una ricompensa negativa.

L'agente utilizza queste ricompense per apprendere quali azioni portano a risultati migliori. Utilizzando un algoritmo di apprendimento, l'agente aggiorna le sue conoscenze sulle azioni migliori da intraprendere in determinati stati. Ad esempio, se sterzare a sinistra in un determinato punto della pista ha portato a una ricompensa positiva, l'agente associa uno stato di "sterzare a sinistra in quel punto" a una ricompensa elevata.

Con il tempo, l'agente accumula conoscenze sull'ambiente di gioco e impara una strategia di guida ottimale che massimizza le ricompense. Ad esempio, impara a prendere le curve nel modo migliore, a evitare gli ostacoli e a mantenere una velocità appropriata. Man mano che l'agente continua a interagire con l'ambiente di gioco, può migliorare la sua strategia di guida e diventare sempre più abile nel completare il percorso nel minor tempo possibile.

2.2 Modelli di regressione nel Machine Learning

Nei prossimi paragrafi andremo a vedere le caratteristiche fondamentali di un modello di regressione nel contesto del Supervised Learning.

2.2.1 Formulazione matematica

Consideriamo un dataset costituito da dati in input $\mathbf{X}$ e output cercato $\mathbf{y}$. Solitamente $\mathbf{X}$ ha una rappresentazione matriciale in cui ogni riga corrisponde a un'osservazione e ogni colonna corrisponde una variabile.

Siano dunque $\mathbf{X} = (\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(d)})$, $\mathbf{x}^{(j)} \in \mathbb{R}^n$ per ogni $j = 1, \dots, d$, con $n, d \in \mathbb{N}$ e $\mathbf{y} \in \mathbb{R}^n$, e supponiamo che $\mathbf{y}$ sia legato a $\mathbf{X}$ attraverso il modello esatto

$$\mathbf{y} = f(\mathbf{X}) + \varepsilon, \tag{2.1}$$

dove ε è un rumore che spesso si assume segua una distribuzione normale con media nulla.

L'obiettivo del Machine Learning è quello di sviluppare un modello $\hat{f}$ che approssimi il modello reale per ottenere una stima $\hat{\mathbf{y}}$ del valore vero. Poiché persino un'approssimazione perfetta lascia un errore ε, ci si riferisce a quest'ultimo come errore irriducibile. Per sviluppare $\hat{f}$, il modello si allena riducendo una funzione obiettivo L, detta *loss function*, che rappresenta l'errore nella stima $\hat{\mathbf{y}}$ di $\mathbf{y}$.

Esempi di loss function sono il *mean absolute error*, o MAE, che misura la media tra tutte le osservazioni delle differenze in valore assoluto tra valori reali e valori predetti, o il *mean squared error*, o MSE, che calcola la media dei quadrati delle differenze tra i risultati della regressione e i valori effettivi della variabile output.

Un modello f può essere descritto dai suoi parametri, che variano a seconda del metodo utilizzato (cfr. Capitolo 3). Questi parametri possono essere, ad esempio, i coefficienti di una funzione associata al modello, come nel caso della regressione lineare (cfr. Paragrafo 3.1) o le modalità con la quale avviene la divisione nel caso degli alberi decisionali, dei quali parleremo nel Paragrafo 3.3. L'obiettivo del processo di addestramento è trovare i parametri ottimali tali per cui la loss function L sia minima.

2.2.2 Valutazione del modello di regressione

Il dataset che viene fornito all'algoritmo (input e output) viene suddiviso in due parti: training e testing. A sua volta il primo si divide in *training set* e *validation set*, come mostrato nella Figura 2.3. Vediamoli nel dettaglio.

Training

- Training set: sono noti i valori di output oltre a quelli di input. Qui l'algoritmo si allena trovando i valori ottimali dei parametri minimizzando la loss function;

- Validation set: i valori di output vengono "nascosti" e l'algoritmo fornisce le sue previsioni che verranno successivamente valutate attraverso la loss function insieme ai valori reali.

Testing

Il modello finale viene applicato sui valori di input del *test set* e valutato definitivamente confrontando i risultati ottenuti con i valori di reali.

Figura 2.3: Divisione del dataset in training set, validation set e test set

Per determinare la performance del modello nella fase di training, viene calcolato l'errore sulla predizione. Esso può essere scomposto come somma di tre contributi:

- Var $\left(\hat{f}(x)\right)$: varianza del modello. Ci dà un'indicazione su quanto il modello che stiamo valutando varierebbe se applicato a differenti training set. Un'alta varianza significa che il modello tiene in considerazione il rumore del dataset;

- Bias: errore prodotto semplificando il dataset. Se il bias è alto il modello fallisce nel generalizzare il dataset.

I primi due addendi costituiscono le componenti riducibili dell'errore.

- Var(ε): errore irriducibile. È l'errore intrinseco dei dati e non può essere ridotto.

La Figura 2.4 mostra come tali componenti varino con la complessità del modello. Idealmente, un modello ha bassa varianza e basso bias; tuttavia, si nota che aumentando la complessità, la varianza aumenta e si abbassa il bias. Perciò, spesso si parla di *compromesso bias-varianza.*

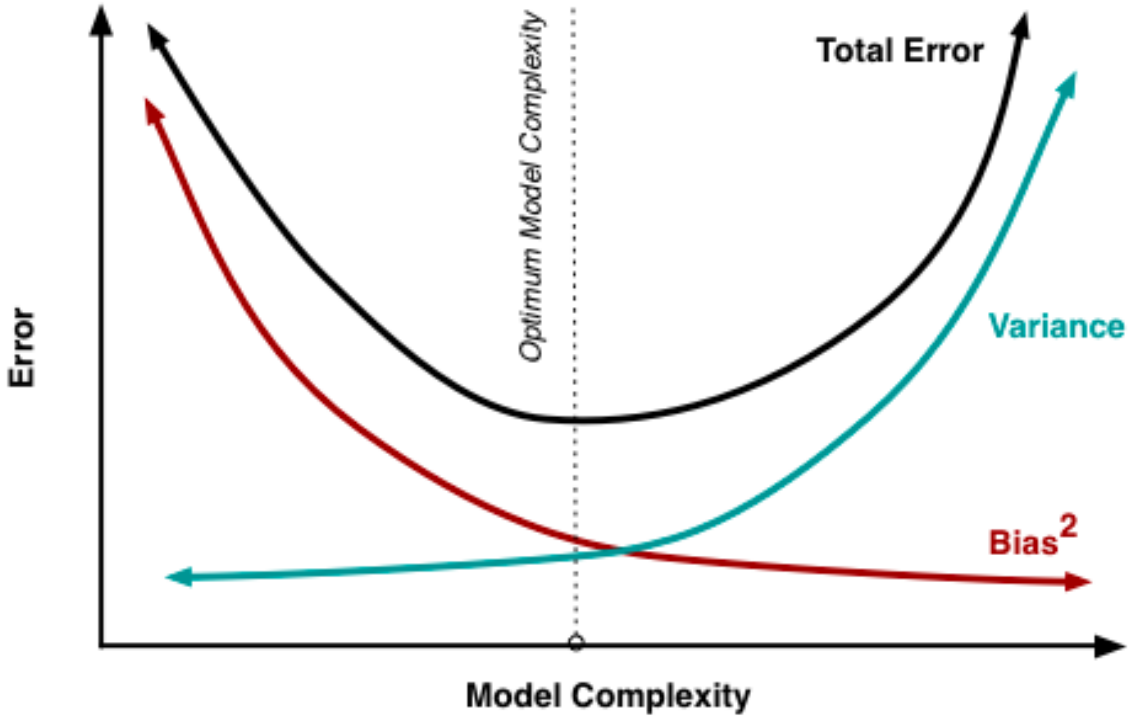


Figura 2.4: Aumentando la flessibilità del modello, il bias si abbassa e cresce la varianza. Il valore ottimale della complessità sta nel mezzo

2.2.3 Ottimizzazione degli iperparametri

Molti modelli hanno parametri addizionali, detti *iperparametri*, che non sono allenati ma che vanno impostati allo scopo di aggiustare bias e varianza.

Il processo di ottimizzazione degli iperparametri consiste nell'allenare il training set, calcolare l'errore sui dati "sconosciuti" del validation set e cambiare la combinazione degli gli iperparametri ottenuti per abbassare tale errore.

Questo procedimento può tuttavia risentire del modo in cui i dati noti sono stati divisi in training e validation set, ecco perché spesso si ricorre a un procedimento noto come *cross validation*.

2.2.4 K-fold cross validation

La cross validation comincia con la suddivisione dell'insieme dei dati di training in sottoinsiemi complementari, dopodiché il modello è allenato su una parte di questi e validato sulla rimanente. Tale operazione è ripetuta per diverse combinazioni dei sottoinsiemi.

Il training set viene dunque suddiviso in k parti (*fold*) di uguale dimensione, in genere si sceglie $k = 5$ o $k = 10$. Quindi si seleziona una parte da usare come validation set, mentre le restanti parti $k - 1$ continuano a comporre il training set vero e proprio.

Si avvia il processo di addestramento e si ripete la procedura per k volte, selezionando ogni volta un fold diverso come validation set.

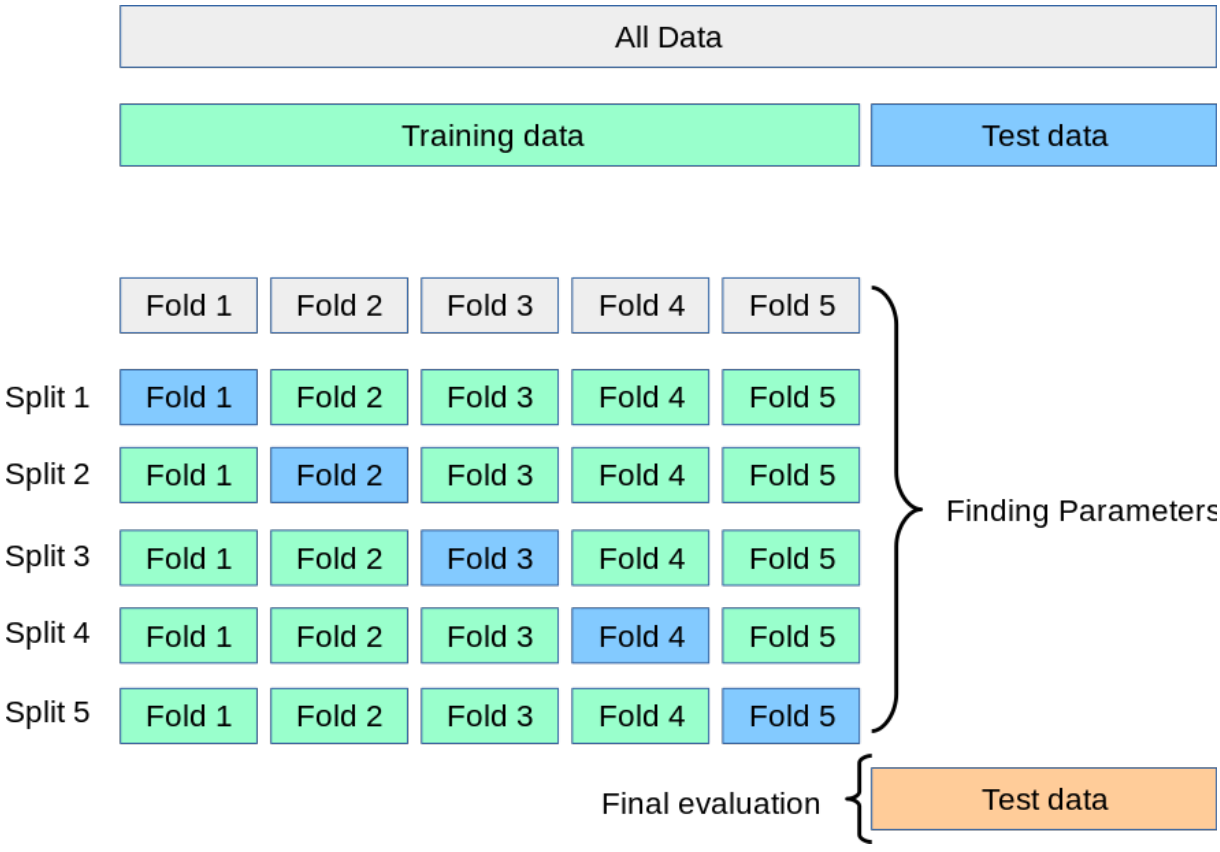


Figura 2.5: Ad ogni split viene scelto un fold da usare come validation set (in blu) e gli altri sono tenuti per l'addestramento (in verde)

L'errore associato alla combinazione corrente di iperparametri è la media dei k errori calcolati sui singoli fold scelti come validation set. Una volta definita la combinazione che restituisce le prestazioni migliori, il modello finale viene allenato sul training set completo e successivamente applicato al test set per fornire l'errore. Il processo di cross validation è schematizzato nella Figura 2.5.

2.2.5 Underfitting e Overfitting

Dalle precedenti considerazioni riguardo il compromesso tra bias e varianza è possibile introdurre due concetti fondamentali: *underfitting* e *overfitting*, schematizzati dalla Figura 2.6.

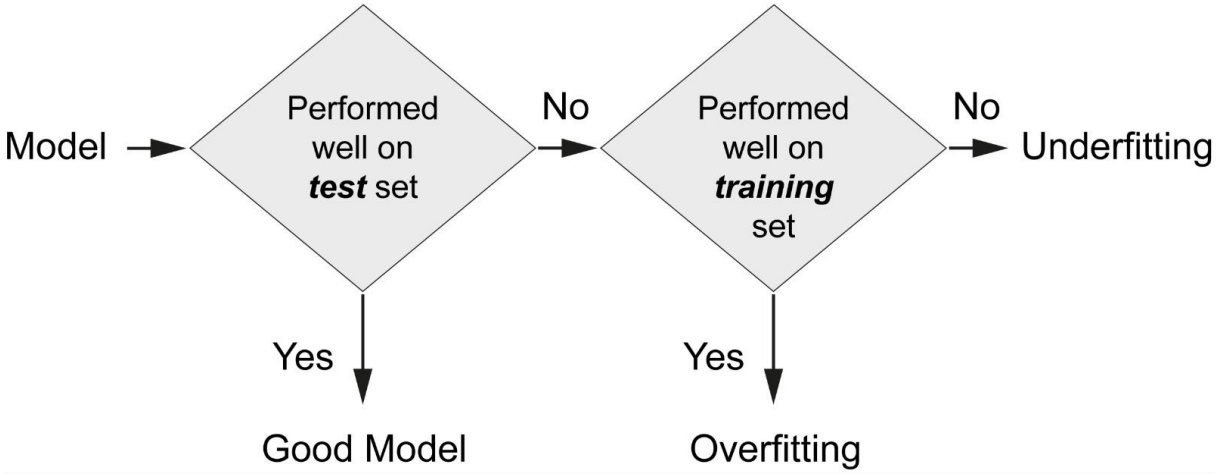


Figura 2.6: Schema underfitting e overfitting

Underfitting

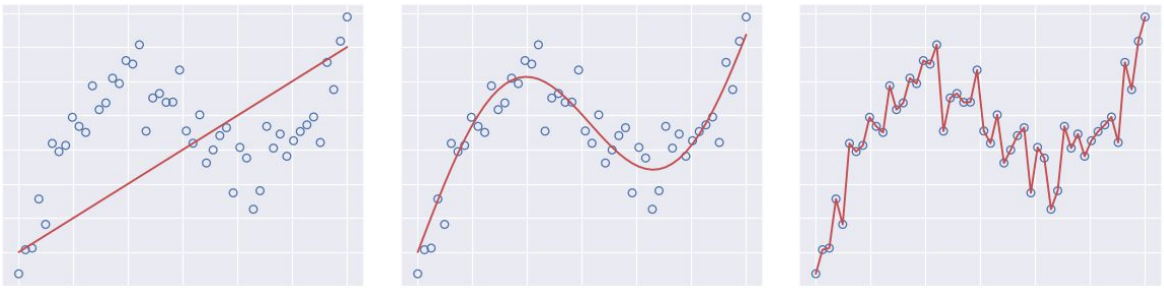
Il modello non performa bene né con i dati di training, né con quelli di test. Ciò è più probabile che accada a modelli troppo semplici oppure inadatti a risolvere il problema.

Questa condizione è caratterizzata da un valore di bias elevato e una bassa varianza.

Overfitting

Il modello tiene in considerazione in maniera eccessiva il rumore tra i dati e non riesce a identificare relazioni che valgono in generale nell'insieme di addestramento. Questa situazione può verificarsi nel caso in cui l'insieme di training sia molto piccolo, oppure quando la complessità del modello è troppo elevata, ed è solitamente identificata nel momento in cui il modello performa bene sui dati di training ma male in quelli di test. In questi casi si è in presenza di un basso bias e alta varianza.

La Figura 2.7 mostra un esempio di interpolazione in cui sono utilizzati tre modelli. Nel primo caso si verifica underfitting, essendo il modello troppo semplice. Nell'ultimo caso siamo in presenza di overfitting, in quanto il modello è eccessivamente complesso. Infine, nel caso in mezzo, l'equilibrio tra bias e varianza dà luogo al modello ottimale.



(a) Underfit (b) Buon fit (c) Overfit

Figura 2.7: Esempio di underfitting (sinistra), buon fit (centro) e overfitting (destra)

Verrebbe da chiedersi se esista un modello migliore in senso assoluto, ossia un modello in grado di risolvere al meglio qualsiasi problema. In effetti tale modello non può esistere, come garantisce il noto Teorema *No free lunch*, per la cui dimostrazione si rimanda a [Wolpert, Macready (1997)].

Teorema 2.2.1 (No free lunch)**.** *Dato l'insieme di tutte le funzioni $\mathcal{F}$ e un insieme di funzioni di riferimento $\mathcal{F}_1$, se l'algoritmo $\mathcal{A}_1$ è migliore in media dell'algoritmo $\mathcal{A}_2$ su $\mathcal{F}_1$, allora l'algoritmo $\mathcal{A}_2$ deve essere migliore dell'algoritmo $\mathcal{A}_1$ su $\mathcal{F} \setminus \mathcal{F}_1$.*

Capitolo 3

Modelli di regressione

Come anticipato nel Capitolo 2, nel corso dell'elaborato ci occuperemo di un problema di regressione, ossia una particolare tecnica di apprendimento supervisionato che cerca di stabilire una relazione tra due o più variabili. Quando forniamo a un modello di regressione i valori di una o più variabili $\mathbf{X}$, questo restituisce il corrispondente valore (continuo) della variabile target $\mathbf{y} = (y_1, \ldots, y_n)^T$ generato dall'elaborazione di $\mathbf{X}$. In presenza di più variabili dipendenti si parla di regressione multivariata.

In pratica, per ognuna delle osservazioni, dati i predittori $\mathbf{x}_i = (x_i^{(1)}, \ldots, x_i^{(d)})$, l'obiettivo della regressione è predire y_i attraverso un modello $\hat{\mu}$ della funzione μ che rappresenta l'attesa condizionata, vale a dire:

$$\mu(\mathbf{x}_i) = \mathbb{E}[y \mid \mathbf{X} = \mathbf{x}_i].$$

Nel seguito considereremo il caso in cui abbiamo a disposizione n osservazioni per ognuna delle d variabili predittive $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(d)}$. Il nostro obiettivo è calcolare una previsione $\hat{\mathbf{y}}$ della variabile target $\mathbf{y}$ per ciascuna di queste osservazioni.

Nei prossimi paragrafi andremo a presentare i modelli di regressione che sono stati utilizzati per la predizione del field tilt. Questi sono la regressione lineare, SVR, Decision Trees, Random Forest e alcune varianti di Gradient Boosting.

3.1 Regressione lineare

Il primo modello utilizzato è la regressione lineare. Questa tecnica prevede di calcolare una funzione lineare che descriva al meglio la relazione fra le variabili predittive e la variabile target. Dato il set di variabili predittive $\mathbf{X} = (\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(d)})$, la regressione lineare comporta il calcolo dei coefficienti $\mathbf{q} = (q_1, \dots, q_n)^T$ e $\beta = (\beta_1, \dots, \beta_d)^T$ della funzione

$$\hat{\mathbf{y}} = f(\mathbf{X}) = \mathbf{q} + \beta_1 \mathbf{x}^{(1)} + \cdots + \beta_d \mathbf{x}^{(d)}$$

che minimizza la distanza fra la funzione stessa valutata nei dati e i valori reali di $\mathbf{y}$, misurata con un'opportuna loss function.

In termini vettoriali, possiamo scrivere

$$\hat{\mathbf{y}} = \mathbf{q} + \mathbf{X}\beta. \tag{3.1}$$

Si noti che nelle equazioni è stato omesso l'errore irriducibile ε in quanto, per definizione, non può essere minimizzato.

Poiché i coefficienti potrebbero risentire del diverso ordine di grandezza delle feature, spesso occorre *standardizzare* i dati: invece di considerare $\mathbf{X}$, cerchiamo i coefficienti $\mathbf{q}$ e β relativi alle quantità $\tilde{\mathbf{X}} = (\tilde{\mathbf{x}}^{(1)}, \dots, \tilde{\mathbf{x}}^{(d)})$ definiti come

$$\tilde{x}_i^{(j)} := \frac{x_i^{(j)} - m^{(j)}}{\sigma^{(j)}} \qquad \forall i = 1, \dots, n, \quad \forall j = 1, \dots, d,$$

dove $m^{(j)}$ e $\sigma^{(j)}$ sono, rispettivamente, la media e la deviazione standard di $\mathbf{x}^{(j)}$. Un'operazione analoga viene applicata a $\mathbf{y}$. Per non appesantire le notazioni, nel seguito ci riferiremo a $\tilde{\mathbf{X}}$ e $\tilde{\mathbf{y}}$ semplicemente come $\mathbf{X}$ e $\mathbf{y}$, specificando il caso in cui stiamo lavorando con i valori standardizzati. È importante sottolineare che non siamo interessati a conoscere i valori specifici di $\mathbf{q}$ e β: l'obiettivo del Machine Learning è proprio quello di trovarli autonomamente addestrandosi sul training set e fornirci il punteggio ottenuto tramite la loss function sul validation set.

3.2 SVR

L'algoritmo Support Vector Regression (SVR) è un tipo specifico di Support Vector Machine (SVM) utilizzato per risolvere problemi di regressione. SVM si basa sull'idea di

trovare un iperpiano che separi al meglio un set di dati in due o più classi. In particolare, SVR raggiunge tale scopo utilizzando una mappatura non lineare f (detta *kernel function*) per trasportare le variabili in input in uno spazio di una dimensione superiore. Successivamente, cerca l'iperpiano che massimizza il margine (distanza) tra l'iperpiano e i punti dati più vicini, minimizzando contemporaneamente l'errore di previsione.

Per cominciare, consideriamo una retta di regressione come in (3.1). SVR stabilisce una soglia di errore ϵ intorno alla retta, con l'obiettivo di trovare una funzione $f(\mathbf{X})$ che abbia al massimo una deviazione $\epsilon > 0$ dai valori effettivi di $\mathbf{y}$ per tutti i dati di training e che sia il più possibile "piatta". In altre parole, non siamo interessati all'errore se è inferiore di ϵ, ma non accetteremo uno scostamento maggiore di questa soglia.

Consideriamo dunque una funzione lineare

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + q,$$

con $\mathbf{x}, \mathbf{w} \in \mathbb{R}^d$, $q \in \mathbb{R}$, con la quale vogliamo predire il valore di $y \in \mathbb{R}$. In questo caso volere f "piatta" equivale a richiedere che $\mathbf{w}$ sia piccolo in norma. Il problema risolto da SVR non è altro che un problema di ottimizzazione convessa della forma

$$\begin{aligned} &\text{minimizzare} \quad \frac{1}{2}\|\mathbf{w}\|^2 \\ &\text{vincolato a} \quad \begin{cases} y_i - \mathbf{w}^T\mathbf{x}_i - q \leq \epsilon \\ \mathbf{w}^T\mathbf{x}_i + q - y_i \leq \epsilon \end{cases} \qquad i = 1, \dots, n. \end{aligned} \tag{3.2}$$

Viene poi definita, per ogni i, una variabile di *slack* ξ_i che rappresenta la distanza di y_i dagli iperpiani $f(\mathbf{x}) \pm \epsilon$. Se y_i si discosta dalla retta di regressione di meno di ϵ, allora $\xi_i = 0$. Invece, se y_i si trova al di sopra della retta, allora ξ_i rappresenta la distanza verticale tra y_i e $f(\mathbf{x}) + \epsilon$. Infine, se y_i si trova al di sotto della retta, ξ_i rappresenta la distanza verticale tra y_i e $f(\mathbf{x}) - \epsilon$. In entrambi gli ultimi casi, ξ_i misura l'errore di previsione rispetto alla soglia ϵ. A questo punto il problema (3.2) diventa

$$\begin{aligned} &\text{minimizzare} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ &\text{vincolato a} \quad \begin{cases} y_i - \mathbf{w}^T\mathbf{x}_i - q \leq \epsilon + \xi_i \\ \mathbf{w}^T\mathbf{x}_i + q - y_i \leq \epsilon + \xi_i \\ \xi_i > 0 \end{cases} \qquad i = 1, \dots, n, \end{aligned}$$

dove $C > 0$ è chiamato *parametro di regolarizzazione* e determina il compromesso tra la "piattezza" di f e la tolleranza delle deviazioni maggiori di ϵ. Per i dettagli teorici sulla soluzione del problema, che vanno oltre gli obiettivi di questo elaborato, si guardi [Smola, Schölkopf (2004)].

Lo step successivo è quello di selezionare la kernel function f: in genere vengono usati modelli di regressione lineari, polinomiali o radiali detti RBF (*Radial Basis Function*).

In ogni caso, C e ϵ sono i principali iperparametri ottimizzati. Come nel caso della regressione lineare, è necessario standardizzare i dati prima di applicare l'algoritmo.

3.3 Decision Trees

L'espressione *Decision Trees* si riferisce a uno strumento statistico non parametrico basato sulla partizione ricorsiva. La locuzione "non parametrica" è spesso usata come descrizione generale per i metodi che non si basano su ipotesi relative a una distribuzione di probabilità da cui sono tratti i dati.

Un albero decisionale è composto da tre parti fondamentali:

- Nodi, che rappresentano condizioni o domande che vengono poste sulle variabili predittive. Ogni nodo ha un insieme di regole o criteri che determinano la divisione del dataset in sottoinsiemi più piccoli;
- Rami, che partono da un nodo padre a un nodo figlio e rappresentano i possibili esiti delle condizioni poste nel nodo padre. Ogni ramo corrisponde a un possibile valore o risultato della condizione.
- Foglie, che contengono l'output della previsione della variabile obiettivo.

In corrispondenza di ogni nodo avviene un'operazione di split dell'albero in più rami, che terminano in nuovi nodi. Le foglie sono nodi derivanti da questa operazione di split che non vengono ulteriormente divisi. È importante notare che, a causa della struttura dell'albero, il modello riesce ad assegnare diversi gradi di importanza alle variabili prese in considerazione: quanto più un nodo è posizionato in alto nell'albero, tanto maggiore è l'importanza della variabile ad esso associata.

Il partizionamento ricorsivo segue una procedura *divide et impera*: consideriamo un'osservazione y della variabile obiettivo i relativi valori $x^{(1)}, \ldots, x^{(d)}$ delle feature. Viene selezionato un valore ξ, in corrispondenza del quale viene eseguita una suddivisione binaria di una variabile $x^{(j)}$ a seconda che si verifichi $x^{(j)} > \xi$ o $x^{(j)} \leq \xi$. Una volta raggiunta una foglia, la predizione di y è basata sulla media o la mediana dei valori ottenuti su quella foglia durante la fase di training.

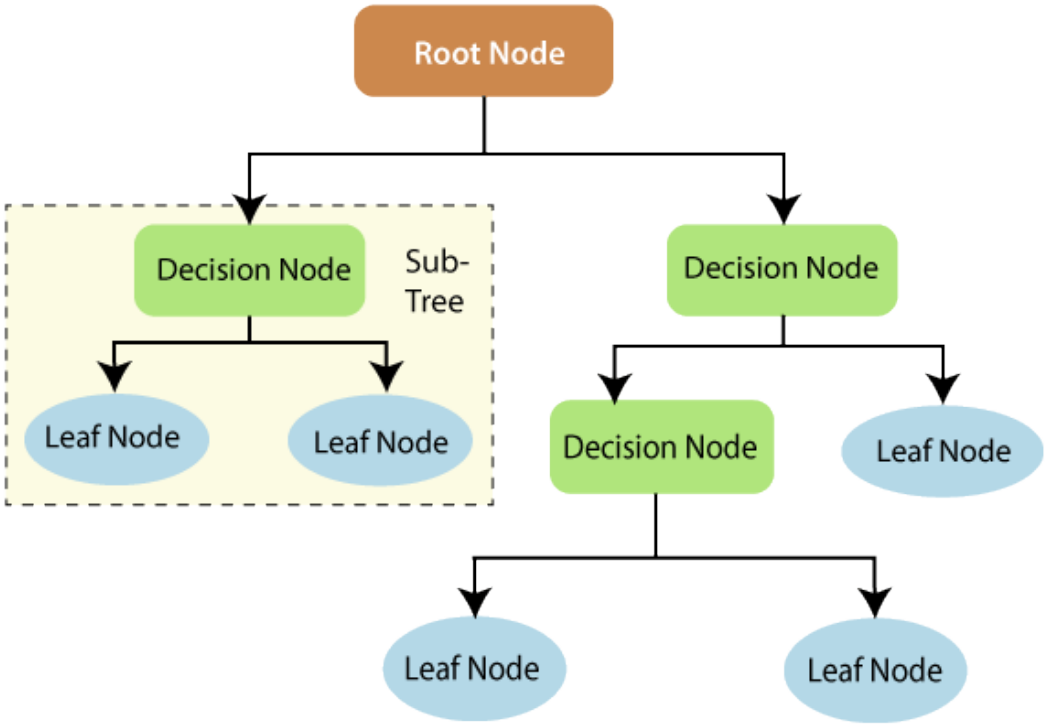


Figura 3.1: Schema albero decisionale

Dato un nodo A, associamo un'*impurità* $I(A)$ definita come

$$I(A) := \sum_i (y_i - \bar{y}(A))^2,$$

dove $\bar{y}$ è il valore medio delle osservazioni y_i presenti in A.

Il miglioramento ottenuto da una suddivisione s in due nodi A_D e A_S è definita da

$$\Delta I(s, A) := I(A) - I(A_D) - I(A_S).$$

L'algoritmo considera poi tutti i possibili split di tutte le variabili e seleziona la suddivisione s associata al maggior miglioramento $\Delta I(s, A)$.

La costruzione degli alberi decisionali richiede metodologie specifiche per decidere su quali variabili e sotto quali condizioni eseguire il processo di suddivisione, ma necessita anche di stabilire opportune condizioni di arresto che consentano di ridurre il rischio di overfitting del modello. Per prevenire tale situazione, l'algoritmo può avviare una fase di potatura (*pruning*) in cui individua le feature che non hanno contribuito in modo significativo ad una suddivisione delle istanze ed elimina i nodi corrispondenti, riunendo le istanze al livello superiore.

Un'altra strategia comunemente utilizzata è quella di definire una profondità massima per l'albero: l'algoritmo esegue il processo di suddivisione ad ogni nodo fino a quando si raggiunge la profondità prefissata o quando non ci sono più variabili disponibili per distinguere ulteriormente i dati.

I principali vantaggi dei Decision Trees includono:

- Interpretabilità: gli alberi decisionali producono risultati facilmente interpretabili, specialmente quando hanno una struttura semplice o una piccola dimensione. Le regole decisionali prese durante il processo di suddivisione sono chiaramente visibili nell'albero, consentendo una comprensione intuitiva del processo di decisione;
- Bassi costi computazionali: il processo di suddivisione dell'albero è relativamente veloce e, una volta che l'albero è costruito, la previsione per nuovi dati richiede solo la navigazione lungo il percorso dell'albero, senza la necessità di calcoli complessi;
- Spiegabilità del modello: le regole prese durante il partizionamento consentono di identificare le variabili più importanti per la previsione del target.

I maggiori svantaggi associati a questo metodo sono invece:

- Alta possibilità di overfitting; gli alberi decisionali tendono a adattarsi troppo ai dati di training, creando modelli che memorizzano il rumore e le caratteristiche specifiche dei dati di addestramento. Ciò può portare a una scarsa capacità di generalizzazione su nuovi dati e a un alto livello di errore di previsione;

- Sensibilità alle piccole modifiche nel training set: piccole modifiche nel training set possono portare a grandi differenze nei risultati di testing. Ciò significa che gli alberi decisionali possono essere instabili e sensibili alle variazioni nei dati di input.

In fase di cross validation, i principali iperparametri presi in considerazione per la regolarizzazione dei Decision Trees sono:

- max_depth, indica la massima profondità consentita dell'albero;
- $min\_samples\_leaf$, ossia il numero minimo di istanze che un nodo foglia deve avere. Un punto di suddivisione a qualsiasi profondità viene considerato solo se lascia almeno $min\_samples\_leaf$ campioni di training samples in entrambi i rami destro e sinistro;
- $max\_features$, il massimo numero di variabili da considerare per dividere ogni nodo.

3.4 Random Forest

Per comprendere le Random Forest è necessario introdurre tre concetti fondamentali: *ensemble*, *Bootstrap* e *Bagging*.

Ensemble

I metodi ensemble seguono l'idea che anche con semplici tecniche di previsione si può ottenere un potente strumento di predizione, se queste vengono aggregate in maniera utile, in un modo tale da massimizzare le prestazioni usando i punti di forza di ognuna di esse e limitandone le relative debolezze.

L'idea generale è di combinare diversi metodi di predizione, detti *base learner*, in uno strumento predittivo più accurato. Per una nuova osservazione, si utilizza un predittore aggregato che tiene conto delle previsioni di tutti i base learner, ad esempio calcolando la media dei risultati. I metodi ensemble riflettono questa idea generale e tra di loro differiscono per due caratteristiche:

- base learner;

- il modo di generare sottocampioni o ripesare i dati in ogni iterazione.

Bootstrap

Gli alberi di decisione descritti nel Paragrafo 3.3 possono produrre risultati con elevata varianza, questo significa che effettuando una diversa suddivisione in training e validation set del dataset di training, potremmo ottenere risultati anche molto diversi.

Il Bootstrap è una tecnica statistica di ricampionamento con reimmissione utilizzata per approssimare la distribuzione campionaria di una statistica: il metodo parte da un dataset iniziale e tramite vari ricampionamenti crea nuovi dataset.

La tecnica Bootstrap presenta diversi vantaggi rispetto ai metodi tradizionali. Innanzitutto, non richiede assunzioni iniziali sulla distribuzione dei dati, il che la rende molto flessibile. Inoltre, può essere utilizzata anche con campioni di bassa dimensione, che è un vantaggio importante quando si dispone di un numero limitato di dati. Tuttavia, il principale svantaggio del metodo è la sua alta complessità computazionale, poiché richiede la generazione di numerosi dataset di Bootstrap e il calcolo delle statistiche corrispondenti.

Bagging

Bagging (unione delle parole *Bootstrap aggregation*) è una tecnica ensemble per generare versioni multiple di un base learner e combinarle per ottenere un predittore aggregato. Gli alberi di regressione sono utilizzati come base learner e l'aggregazione avviene tramite media, mentre le versioni multiple del predittore sono costruite generando campioni Bootstrap dei dati.

Il guadagno di accuratezza di questo metodo deriva dall'instabilità dei base learner. Poiché gli alberi includono sempre suddivisioni binarie, per la loro costruzione sono piuttosto dipendenti dall'insieme di dati.

La procedura generale di Bagging e la seguente:

1. si costruiscono S campioni Bootstrap dall'training set originario;

2. per ognuno dei campioni di Bootstrap si fa crescere un grande albero;

3. per predire il valore della variabile obiettivo di una nuova osservazione si fanno passare le relative covariate attraverso tutti gli S alberi e si calcola la media delle predizioni ottenute.

In ognuno degli alberi si ottiene una previsione $f^k(\mathbf{x}_i)$, dunque la previsione finale è

$$f_{\text{Bootstrap}}(\mathbf{x}_i) = \frac{1}{S}\sum_{k=1}^{S} f^k(\mathbf{x}_i).$$

La peculiarità del Bagging è che risulta più performante quando i base learner non sono particolarmente stabili e sono quindi molto sensibili alle variazioni del training set, come nel caso degli alberi decisionali. Invece, in presenza di learner stabili le previsioni potrebbero peggiorare. Per le garanzie di funzionamento della procedura di Bagging, si rimanda a [Breiman (1996)].

Random Forest è un ulteriore metodo ensemble basato su alberi decisionali e presenta molte somiglianze con il Bagging. In effetti, il Bagging può essere visto come un caso speciale di Random Forest.

Come nel Bagging, gli alberi di regressione sono utilizzati come base learner. Per la modifica dei dataset, in ogni iterazione vengono estratti campioni Bootstrap delle osservazioni e i singoli predittori di questi sottoinsiemi vengono aggregati mediante una media. A differenza del Bagging, invece, non tutte le covariate entrano in ogni base learner. Infatti, per le Random Forest la casualità non è inclusa solo nella selezione delle osservazioni, ma anche nella selezione delle variabili per la crescita degli alberi: in ogni iterazione viene incluso solo un certo sottoinsieme di covariate. Di conseguenza, gli alberi dei singoli base learner differiscono maggiormente rispetto al metodo Bagging, poiché vi è una maggiore variazione nell'insieme delle variabili selezionate. Le variabili che hanno un effetto relativamente basso sulla risposta hanno maggiori possibilità di entrare nello schema di previsione finale nelle Random Forest rispetto a quanto accade con il metodo Bagging.

Il numero di variabili candidate che sono selezionate ad ogni step è specificato dall'iperparametro $max\_features$: se impostato su `'sqrt'` o `'log'` vengono selezionate,

rispettivamente, la radice quadrate o il logaritmo naturale del numero totale di covariate; se impostato su `None` si selezionano tutte le variabili, ottenendo il Bagging.

A partire dal training set vengono costruiti un elevato numero di alberi decisionali, ognuno dei quali genererà una previsione; tale numero è specificato dall'iperparametro $n\_estimators$.

Come nel caso dei Decision Trees, l'iperparametro max_depth rappresenta la massima profondità dell'albero. Per maggiori dettagli si guardi [Brieman (2001)].

Random Forest eredita i vantaggi di Bootstrap e Bagging, che consistono nel ridurre l'overfitting e la dipendenza dei risultati dal dataset, rispetto ai semplici Decision Trees. L'introduzione di elementi randomici e l'utilizzo della tecnica ensemble contribuiscono a migliorare le prestazioni del modello. Tuttavia, bisogna considerare che ciò comporta un aumento significativo dei costi computazionali e del consumo di memoria. Inoltre, a causa della complessità del modello e dell'aggregazione dei risultati, l'interpretazione dei risultati e l'importanza delle variabili possono risultare meno intuitivi rispetto ai Decision Trees tradizionali.

3.5 Gradient Boosting

Un altro metodo ensemble molto usato nel Machine Learning è il Gradient Boosting, che deve il suo nome a due nozioni chiave: *Gradient Descent* e *Boosting*. Esaminiamo questi concetti da una prospettiva più ravvicinata.

Boosting

Il Boosting è una tecnica utilizzata per migliorare le prestazioni di un modello predittivo combinando diversi modelli più semplici, noti come *weak learner*. L'obiettivo è creare un modello più complesso e accurato, in grado di generalizzare meglio rispetto ai singoli modelli base.

La principale idea del Boosting è l'addestramento iterativo di una serie di weak learner. Ad ogni iterazione, viene aggiunto un weak learner che si basa sulle informazioni elaborate dai modelli precedenti. Il risultato finale del modello, detto *strong learner*

è ottenuto dalla combinazione dei risultati dei weak learner. È importante notare che ogni weak learner non deve necessariamente essere una versione migliorata dei modelli costruiti nelle iterazioni precedenti. Piuttosto, ogni weak learner è progettato per prevedere e correggere le variazioni rispetto al valore precedentemente stimato, con l'obiettivo di ridurre la loss function.

In sintesi, il processo di Boosting inizia creando un primo modello utilizzando i dati del training set. Successivamente, viene creato un secondo modello che cerca di correggere gli errori del primo modello. Questo processo viene ripetuto iterativamente, con l'obiettivo di modellare al meglio gli errori ottenuti durante le iterazioni precedenti.

Il Gradient Boosting, presentato in [Friedman (2001)], è un modello di Boosting in cui i weak learner sono alberi decisionali e ad ogni step viene risolto un problema di discesa del gradiente.

Gradient Descent

La discesa del gradiente, o Gradient Descent, è un algoritmo iterativo utilizzato nell'ambito dell'ottimizzazione numerica per trovare i minimi locali (o globali) di una funzione g. Il principio di base consiste nel generare una successione di approssimazioni muovendosi lungo la direzione opposta al gradiente di g, al fine di avvicinarsi al minimo della funzione. Ad ogni iterazione, vengono calcolate le derivate parziali rispetto ai parametri del modello e il valore dei parametri viene aggiornato in modo proporzionale al gradiente.

Il procedimento seguito è il seguente: si sceglie un punto di partenza arbitrario z_0 e ad ogni passo k ci si muove in una direzione $\Delta z_k = -\nabla g(z_k)$ con un passo t_k fino al prossimo punto z_{k+1}. La regola ricorsiva seguita è

$$z_{k+1} = z_k - t_k \nabla g(z_k), \qquad k \in \mathbb{N}.$$

Nel caso del Gradient Boosting, la scelta del passo t_k viene presa in modo da minimizzare il valore della funzione nel prossimo punto z_{k+1}. Dunque, si ha

$$t_k = \arg\min_{t \geq 0} g(z_k - t \nabla g(z_k)).$$

I metodi di Gradient Boosting sono comunemente caratterizzati da tre iperparametri principali:

- il numero di alberi da utilizzare M, spesso chiamato $n\_estimators$ o $iterations$. Aumentandolo, cresce anche il rischio di overfitting, a differenza di varianti di Bagging come Random Forest;
- un parametro ν detto $learning\_rate$, generalmente molto piccolo, che determina il peso da attribuire al risultato ottenuto dall'aggiunta di un nuovo albero. Tale parametro previene l'overfitting;
- l'ampiezza dei weak learner, alla quale ci si riferisce come max_depth.

Proviamo ora a spiegare in modo sintetico il metodo:

1. date le osservazioni target y_i e le relative covariate $\mathbf{x}_i$, $i = 1, \dots, n$, definiamo la loss function $L(y, \cdot)$;
2. inizializziamo una successione di modelli $F_m(\mathbf{X})$, avente come valore iniziale la costante
$$F_0(\mathbf{x}) = \arg\min_{\rho} \sum_{i=1}^{n} L(y_i, \rho);$$
3. scegliamo il numero M di alberi da generare e per $m = 1, \dots, M$
   - calcoliamo i cosiddetti *pseudo-residui*
   $$r_{i,m} = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]\Bigg|_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} \qquad \forall i = 1, \dots, n;$$
   - alleniamo un albero $h_m(\mathbf{x})$ sul dataset $\{\mathbf{x}_i, r_{i,m}\}_{i=1}^{n}$;
   - calcoliamo
   $$\rho_m = \arg\min_{\rho} \sum_{i=1}^{n} L(y_i, F_{m-1}(\mathbf{x}_i + \rho h_m(\mathbf{x}_i));$$
   - aggiorniamo il modello
   $$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu\rho_m h_m(\mathbf{x}).$$

È importante notare come la predizione $F_m(\mathbf{X})$ sia influenzata dalla predizione $F_{m-1}(\mathbf{X})$ calcolata nello step precedente, a cui poi va ad aggiungersi il risultato dell'albero h_m

appena creato, tenendo conto del learning rate.

I punti di forza del Gradient Boosting, rispetto ad altri metodi ensemble, sono molteplici: l'abilità di gestire dati eterogenei, la capacità di rilevare relazioni non lineari tra le variabili e la gestione automaticamente le feature mancanti.

Nei prossimi paragrafi presentiamo alcune recenti varianti di Gradient Boosting: XGBoost, LightGBM e CatBoost.

3.5.1 XGBoost

XGBoost (eXtreme Gradient Boosting) è un algoritmo di Gradient Boosting sviluppato nel 2016 da Tianqui Chen ([Chen, Guestrin (2016)]). Questo metodo è noto per la sua velocità e capacità di gestire grandi set di dati, oltre a offrire una buona accuratezza predittiva. Tra le principali caratteristiche di XGBoost ci sono la capacità di gestire informazioni mancanti nei dataset e la regolarizzazione per prevenire l'overfitting, penalizzando i modelli più complessi per migliorarne la generalizzazione. Inoltre, XGBoost utilizza un algoritmo migliorato per la ricerca dello split ottimale in fase di Boosting.

Diamo ora una panoramica di ciò che c'è alla base di XGBoost.

Supponiamo di voler costruire ognuno degli M alberi con un massimo di Z foglie. Chiamiamo $q : \mathbb{R}^d \to \{1, \dots, Z\}$ la funzione che assegna ad ogni osservazione i un numero intero che indica la foglia finale dell'albero di decisione in cui l'osservazione ricade, sulla base dei valori dei predittori corrispondenti $\mathbf{x}_i$. Sia inoltre $O^{(m)} : \mathbb{R}^d \to \mathbb{R}$ una funzione che restituisce la predizione associata alle variabili indipendenti in input, ottenuta dall'albero m. Vale a dire

$$O^{(m)}(\mathbf{x}_i) = w^{(m)}_{q(\mathbf{x}_i)}.$$

Il vettore $\mathbf{w}^{(m)} = (w^{(m)}_1, \dots, w^{(m)}_Z)^T \in \mathbb{R}^Z$ ha come componenti i valori output associati alle Z foglie dell'albero m.

Durante il Boosting, XGBoost genera l'albero m cercando di minimizzare la quantità

$$\mathcal{L}^{(m)} := \sum_{i=1}^{n} L(y_i, \hat{y}_i^{(m-1)} + O^{(m)}(\mathbf{x}_i)) + \frac{1}{2}\lambda\|\mathbf{w}^{(m)}\|^2,$$

dove $\hat{y}_i^{(m-1)}$ è la previsione di y_i ottenuta con l'albero $m-1$ e λ è un parametro di penalità. Tale funzione viene minimizzata al variare dei valori di $\mathbf{w}^{(m)}$. Svolgendo i conti, per i quali si rimanda a [Chen, Guestrin (2016)], si ottiene che per ogni foglia j il valore ottimale di $w_j^{(m)}$ è

$$w_j^{(m)} = \frac{\sum_{i \in I_j} g_i^{(m)}}{\sum_{i \in I_j} h_i^{(m)} + \lambda}.$$

$I_j = \{i \mid q(\mathbf{X}_i) = j\}$ è insieme di osservazioni che ricadono sulla foglia j, mentre $g_i^{(m)} = \dfrac{\partial L(y_i, \hat{y}_i^{(m-1)})}{\partial \hat{y}_i^{(m-1)}}$ e $h_i^{(m)} = \dfrac{\partial g_i^{(m)}}{\partial \hat{y}_i^{(m-1)}}$ sono, rispettivamente, gradiente e hessiana della loss function.

3.5.2 LightGBM

Nel 2017 ([Ke et al. (2017)]) Microsoft presenta Light Gradient Boosting Machine, o LightGBM, un algoritmo di Gradient Boosting che supera i suoi predecessori per velocità ed efficienza. LightGBM è ottimale nella gestione di variabili sia numeriche che *categoriche*, ossia variabili che possono assumere un insieme discreto di valori che non possono essere confrontati tra loro, ad esempio colori. Inoltre, è in grado di gestire valori mancanti e ridurre il rischio di overfitting mediante l'applicazione di regolarizzazione.

Una delle principali ragioni che ha spinto alla creazione di questo metodo è la gestione di grandi dataset: gli algoritmi classici risultano spesso inefficienti quando si lavora con un grande volume di dati, poiché è necessario esaminare tutte le osservazioni per ciascuna variabile al fine di generare i migliori split negli alberi. Ciò fa sì che, applicando Gradient Boosting su un dataset di n osservazioni e d covariate, il costo computazionale sia $\mathcal{O}(n * d)$. Per affrontare questo problema, LightGBM sfrutta due nuove tecniche: GOSS (*Gradient-based One-Side Sampling*) e EFB (*Exclusive Feature Bundling*). Queste tecniche permettono di diminuire notevolmente i costi computazionali, riducendo il numero di osservazioni e variabili da considerare durante la fase di Boosting.

GOSS

Con GOSS una proporzione significativa di osservazioni che danno origine a piccoli gradienti (ossia che determinano una variazione poco significativa della loss function) viene esclusa, permettendo di usare solo il resto in fase di divisione in nodi degli alberi. Le osservazioni con grandi gradienti (dette *sottoaddestrate*) vengono tutte conservate, mentre di quelle con piccoli gradienti viene scelto in maniera aleatoria un campione da tenere in considerazione.

Si può dimostrare ([Ke et al. (2017)]) che GOSS, utilizzando una strategia di campionamento basata sul gradiente, riesce a ottenere stime accurate del guadagno di informazione, nonostante l'utilizzo di un set di dati di dimensioni ridotte, poiché le osservazioni che contribuiscono maggiormente alla variazione della loss function giocano un ruolo più importante nel calcolo del guadagno di informazione.

EFB

Con EFB le variabili reciprocamente esclusive, cioè quelle che raramente assumono valori diversi da zero contemporaneamente, vengono raggruppate in *bundle*, riducendo così il numero di predittori. Così facendo, EFB trasforma il problema di raggruppamento ottimale delle variabili in un problema di colorazione del grafo. In questo contesto, ogni variabile rappresenta un vertice del grafo e vengono aggiunti lati tra le coppie di variabili che non sono mutualmente esclusive. L'algoritmo EFB risolve questo problema di colorazione del grafo utilizzando un approccio *greedy*, cercando di assegnare colori diversi alle variabili che condividono un lato nel grafo. Questo permette di ridurre l'interazione tra le variabili e semplifica il processo di creazione degli alberi durante il Boosting. Poiché il numero di bundle creati è rilevantemente inferiore al numero totale di variabili indipendenti, i costi computazionali diminuiscono considerevolmente. I dettagli sull'algoritmo di risoluzione del problema di colorazione del grafo sono esposti in [Ke et al. (2017)].

Due iperparametri spesso considerati quando si usa LightGBM sono num_leaves e $min\_child\_samples$. Essi sono strettamente collegati tra loro: il primo indica il numero massimo Z di foglie che possono avere i base learner, il secondo specifica la quantità

minima di osservazioni che devono ricadere in ogni foglia. Aumentando Z o abbassando le osservazioni per ogni foglia, cresce il rischio di overfitting.

3.5.3 CatBoost

CatBoost è un algoritmo basato su Gradient Boosting sviluppato da Yandex e rilasciato nel 2017. È progettato per gestire dati di grandi dimensioni, specialmente quelli con feature categoriche.

CatBoost introduce due importanti avanzamenti algoritmici: l'implementazione del *Boosting ordinato*, un'alternativa basata sulla permutazione all'algoritmo classico di Boosting, e un innovativo algoritmo per il trattamento delle variabili categoriche. Entrambe le tecniche sono state create per contrastare un *prediction shift*, uno spostamento delle previsioni causato da un particolare tipo di perdita di informazioni nella variabile target presente in tutte le implementazioni esistenti degli algoritmi di Gradient Boosting.

Variabili categoriche

Una tecnica popolare per gestire le variabili categoriche negli algoritmi di Gradient Boosting è l'*one-hot encoding*: per ogni categoria c della variabile categorica x si aggiunge una nuova variabile *dummy*, ossia una variabile binaria che assume il valore 1 solo per le osservazioni il cui valore di x è proprio c. Tuttavia, nel caso di variabili con alta cardinalità, questa tecnica porta a un numero eccessivamente elevato di nuove variabili dummy. Per affrontare questo problema, è possibile raggruppare le categorie in un numero limitato di gruppi e successivamente applicare l'one-hot encoding. Un metodo popolare è quello di raggruppare le categorie utilizzando le *statistiche di target* (TS) che stimano il valore atteso della variabile obiettivo per ogni categoria. Nello specifico, per ogni categoria, vengono calcolate le statistiche di target come la media, la mediana o la deviazione standard della variabile obiettivo per quella categoria. Queste statistiche rappresentano una stima del comportamento della variabile target per ciascuna categoria. Successivamente, le categorie vengono raggruppate in base alle loro similitudini statistiche. Ad esempio, le categorie con valori di target simili possono essere raggruppate insieme. Si noti che le variabili TS richiedono il calcolo e la memorizzazione di un solo numero per

ogni raggruppamento, diminuendo i costi sia in termini di tempo, che di memoria. Tuttavia, è importante tenere presente che l'applicazione delle statistiche di target per il raggruppamento delle categorie può comportare una perdita di informazione, in quanto si riduce la specificità delle categorie individuali.

I dettagli per la gestione di CatBoost delle variabili categoriche, comprensivi dell'implementazione dell'algoritmo associato, sono riportati in [Prokhorenkova et al. (2018)].

Boosting ordinato

La tecnica del Boosting ordinato nasce dalla necessità di risolvere il problema del *prediction shift*, tipico dei modelli di Gradient Boosting: le previsioni del modello sui dati di training sono diverse da quelle sui dati di test, anche quando i dati provengono dalla stessa distribuzione. Ciò dipende dal fatto che i gradienti utilizzati ad ogni passo m vengono stimati utilizzando i valori target delle stesse osservazioni su cui è stato costruito il modello corrente F_{m-1}. Il prediction shift può influire negativamente sulla capacità del modello di generalizzare correttamente i nuovi dati e può portare a risultati meno accurati. Per risolvere questo problema, in [Prokhorenkova et al. (2018)] viene dettagliatamente spiegato il Boosting ordinato. Vediamone i concetti principali.

Ad ogni passo del Boosting, viene campionato un nuovo dataset $\mathcal{D}_m$ in modo indipendente. Supponiamo di addestrare un modello con M alberi. Per rendere il residuo $r_{i,m-1}$ non distorto, è necessario che F_{M-1} sia addestrato senza i valori di $\mathbf{X}_i$. Poiché abbiamo bisogno di residui non distorti per tutte le osservazioni, nessuna di esse può essere utilizzato per addestrare F_{M-1}, il che a prima vista rende il processo di training impossibile. Tuttavia, è possibile mantenere un insieme di modelli che differiscono per le osservazioni utilizzate per il loro addestramento. Pertanto, per calcolare il residuo su un'osservazione, viene utilizzato un modello addestrato senza di essa.

Per illustrare l'idea, supponiamo di prendere una permutazione casuale σ delle osservazioni di training e mantenere p modelli di supporto distinti $\mathcal{M}_1, \ldots, \mathcal{M}_p$ tali che il modello $\mathcal{M}_k$ sia allenato utilizzando solo le prime k osservazioni nella permutazione. Ad ogni passo, per ottenere il residuo per l'i-esima osservazione, viene utilizzato il modello $\mathcal{M}_{i-1}$.

Capitolo 4

Previsione di intervalli di confidenza

Poiché il processo di addestramento e previsione del modello è di natura statistica, è importante considerare l'aleatorietà dei risultati. Pertanto, per valutare in modo affidabile le prestazioni del modello, è necessario tener conto di questa caratteristica intrinseca.

I metodi tradizionali di previsione si concentrano sulla stima di un valore puntuale, mentre la predizione di intervalli fornisce un intervallo di valori all'interno del quale ci si aspetta che cada il valore effettivo. Questo intervallo è chiamato *intervallo di confidenza*.

Invece di fornire una sola previsione, che potrebbe sottostimare o sovrastimare l'incertezza, la predizione di intervalli offre un modo più completo per comprendere la variabilità dei risultati possibili. Gli intervalli di confidenza vengono costruiti in modo tale da contenere il vero valore di un'osservazione y con una certa probabilità predefinita, chiamata *livello di confidenza*.

Il metodo standard per generare intervalli di previsione consiste nel calcolare un modello di regressione per la media condizionata e assumere che i dati seguano una specifica distribuzione. A partire da questa distribuzione, è possibile calcolare un intervallo di previsione simmetrico rispetto alla stima puntuale. Questo approccio funziona bene quando l'ipotesi sulla distribuzione è corretta e abbiamo informazioni valide sulla varianza del metodo di stima. Tuttavia, ciò non è sempre garantito, specialmente quando si tratta di dati del mondo reale: non c'è alcuna garanzia che le osservazioni seguano una legge precisa, motivo per cui è necessario trovare altri metodi per la predizione di intervalli di

confidenza.

Nel seguito, data l'osservazione $\mathbf{x}_i = (x_i^{(1)}, \ldots, x_i^{(d)})$, ci riferiremo all'intervallo di previsione del corrispondente output y_i come

$$PI(\mathbf{x}_i) = [\hat{y}_i^l, \hat{y}_i^u],$$

dove $\hat{y}_i^l$ e $\hat{y}_i^u$ sono gli estremi inferiore e superiore dell'intervallo costruito, all'interno del quale vogliamo ricada y.

Proponiamo nei prossimi paragrafi tre differenti metodi di costruzione di intervalli di confidenza: regressione quantile, *conformal prediction* e *conformalized quantile regression*.

4.1 Regressione quantile

La regressione quantile è una tecnica che consente di stimare i quantili condizionali di una variabile dipendente, dati i valori delle variabili predittive. Prima di entrare nei dettagli, occorre dare la definizione di quantile.

Definizione 4.1.1 (Quantile)**.** Siano F una funzione di ripartizione e $\tau \in]0,1[$. Si chiama τ-*quantile*, o equivalentemente $100 * \tau$ *percentile*, di F la quantità

$$Q_\tau = \inf\{y \in \mathbb{R} \mid F(y) \geq \tau\}.$$

In particolare, se Y è una variabile aleatoria con funzione di ripartizione F_Y, chiamando $Q_\tau(Y)$ il suo τ-*quantile*, allora si ha che

$$\mathbb{P}(Y \leq Q_\tau(Y)) = F_Y(Q_\tau(Y)) = \tau.$$

Osservazione 4.1. Se F_Y è invertibile, con inversa F_Y^{-1}, si ha

$$Q_\tau(Y) = F_Y^{-1}(\tau).$$

Abbiamo visto che l'obiettivo della regressione (puntuale) è stimare l'attesa condizionata di y tramite una funzione μ tale per cui

$$\mu(\mathbf{x}_i) = \mathbb{E}[y \mid \mathbf{x} = \mathbf{x}_i].$$

Il ragionamento alla base della regressione quantile è il medesimo. Tuttavia, invece che cercare la media condizionata, siamo interessati a calcolare il τ-quantile condizionato $Q_\tau(y \mid \mathbf{x})$. Applicandolo a un'osservazione $\mathbf{x}_i$ si ha

$$Q_\tau(y \mid \mathbf{x} = \mathbf{x}_i) := \inf\{z \in \mathbb{R} \mid \mathbb{P}(y \leq z \mid \mathbf{x} = \mathbf{x}_i) \geq \tau\}.$$

Per alleggerire la notazione, indichiamo

$$q_\tau(\mathbf{x}_i) := Q_\tau(y \mid \mathbf{x} = \mathbf{x}_i)$$
$$F_{y|\mathbf{x}}(y_i \mid \mathbf{x}_i) := \mathbb{P}(y \leq y_i \mid \mathbf{x} = \mathbf{x}_i).$$

Ne segue che

$$q_\tau(\mathbf{x}_i) = \inf\{z \in \mathbb{R} \mid F_{y|\mathbf{x}}(z \mid \mathbf{x}_i) \geq \tau\}.$$

Siano ora $0 < \eta < \zeta < 1$ e consideriamo i quantili condizionati $q_\eta(\mathbf{x})$ e $q_\zeta(\mathbf{x})$. Posti

$$y_i^l := q_\eta(\mathbf{x}_i)$$
$$y_i^u := q_\zeta(\mathbf{x}_i),$$

si ha

$$\mathbb{P}(y_i^l < y < y_i^u \mid \mathbf{x} = \mathbf{x}_i) = F_{y|\mathbf{x}}(y_i^u \mid \mathbf{x}_i) - F_{y|\mathbf{x}}(y_i^l \mid \mathbf{x}_i) = \zeta - \eta.$$

Abbiamo ottenuto dunque un intervallo

$$PI_{\zeta-\eta}(\mathbf{x}_i) = [y_i^l, y_i^u],$$

al quale la variabile dipendente y_i appartiene con probabilità condizionata $\zeta - \eta$.

Se vogliamo un livello di confidenza di $1-\alpha$, $\alpha \in]0, 1[$, è sufficiente scegliere i quantili di ordine ζ e η affinché

$$\zeta - \eta = 1 - \alpha.$$

Generalmente, si sceglie di avere quantili simmetrici, ossi tali per cui valga

$$1 - \zeta = \eta.$$

Per ottenere ciò basta scegliere $\eta = \frac{\alpha}{2}$ e $\zeta = 1 - \frac{\alpha}{2}$.

L'utilizzo della regressione quantile nella predizione di intervalli è dunque quello di stimare due funzioni $q_{\frac{\alpha}{2}}$ e $q_{1-\frac{\alpha}{2}}$ tali per cui, dato $0 < \alpha < 1$, l'output y_i corrispondente all'osservazione $\mathbf{x}_i$ appartenga all'intervallo $[q_{\frac{\alpha}{2}}(\mathbf{x}_i), q_{1-\frac{\alpha}{2}}(\mathbf{x}_i)]$ con probabilità $1 - \alpha$.

Tutto ciò ci fornisce una strategia per costruire un intervallo di previsione con livello di confidenza $1 - \alpha$ per un dataset di osservazioni $\mathbf{X} = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \mathbb{R}\}_{i=1}^n$, per ogni $i = 1, \ldots, n$:

1. sviluppare un modello $\hat{q}_\tau$ che approssimi le funzioni q_τ per $\tau = \frac{\alpha}{2}, 1 - \frac{\alpha}{2}$, attraverso la minimizzazione di un'opportuna loss function;

2. calcolare $\hat{\mathrm{PI}}_{1-\alpha}(\mathbf{x}_i) = [\hat{y}_i^l, \hat{y}_i^u]$ come stima di $PI_{1-\alpha}(\mathbf{x}_i)$, dove $\hat{y}_i^l = \hat{q}_{\frac{\alpha}{2}}(\mathbf{x}_i)$ e $\hat{y}_i^u = \hat{q}_{1-\frac{\alpha}{2}}(\mathbf{x}_i)$.

Cerchiamo ora di dare una corretta interpretazione dell'intervallo di previsione $PI_{1-\alpha}(\mathbf{x})$. Assumiamo per semplicità di avere una sola variabile dipendente $\mathbf{x}$. Per una nuova osservazione $(x_{\text{new}}, y_{\text{new}})$, $PI_{1-\alpha}(x_{\text{new}})$ dovrebbe ricoprire y_{new} con una probabilità di $1-\alpha$. Ci sono fondamentalmente due modi per interpretare la copertura di un intervallo di previsione per le future osservazioni:

- **Interpretazione condizionale**: fissato un valore x_{new} della variabile x e considerando un campione di osservazioni $\mathbf{y} = (y_1, \ldots, y_N)^T$ che hanno tutte lo stesso valore x_{new} della corrispondente variabile predittiva x, circa l'$(1 - \alpha) * 100\%$ delle osservazioni rientreranno nell'intervallo di previsione $PI_{1-\alpha}(x_{\text{new}})$. La copertura si riferisce quindi alle osservazioni associate a x_{new};

- **Interpretazione euristica**: per qualsiasi campione di S coppie $(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^S \times \mathbb{R}^S$, circa l'$(1 - \alpha) * 100\%$ delle istanze di $\mathbf{y}$ rientrerà negli intervalli di previsione $PI_{1-\alpha}(x_1), \ldots, PI_{1-\alpha}(x_S)$. La copertura si riferisce quindi all'intero campione.

Nel testare gli intervalli di previsione in problemi del mondo reale, solo l'interpretazione euristica è ammissibile: nella maggior parte dei casi sarà impossibile ottenere un numero

sufficiente di osservazioni x_{new} per dimostrare in modo rigoroso la copertura dell'intervallo. Si noti inoltre che un intervallo che funziona per l'interpretazione condizionale, dovrebbe mantenere la copertura anche per l'interpretazione euristica. Di conseguenza, utilizzare l'interpretazione euristica può fornire un'indicazione se la copertura degli intervalli è corretta. In molte situazioni pratiche, potrebbe essere l'unica soluzione praticabile.

4.2 Conformal prediction

La *conformal prediction* è un framework per il Machine Learning introdotto per la prima volta in [Vovk et al. (2005)] nel 2005. L'idea principale è quella di adattare un modello di regressione sul training set, quindi utilizzare i residui ottenuti sul validation set per quantificare l'incertezza delle future previsioni.

Il primo passo consiste nel dividere il dataset di training in due sottoinsiemi disgiunti:

- un training set vero e proprio $\{(\mathbf{x}_i, \mathbf{y}_i) \mid i \in \mathcal{I}_1\}$;
- un *calibration* set $\{(\mathbf{x}_i, \mathbf{y}_i) \mid i \in \mathcal{I}_2\}$.

Dopodiché, dato un qualsiasi algoritmo di regressione $\mathcal{M}$, un modello $\hat{\mu}$ viene addestrato sul training set.

In seguito, vengono calcolati gli errori assoluti sul calibration set come segue

$$R_i := |y_i - \hat{\mu}(\mathbf{x}_i)|, \qquad i \in \mathcal{I}_2. \tag{4.1}$$

Si sceglie ora il livello di confidenza $1 - \alpha$ e si calcola

$$Q_{1-\alpha}(R, \mathcal{I}_2) := (1-\alpha)(1 + \frac{1}{|\mathcal{I}_2|}) - \text{quantile dell'insieme } \{R_i \mid i \in \mathcal{I}_2\}.$$

A questo punto, per una nuova osservazione $\mathbf{x}_{\text{new}}$ l'intervallo di confidenza risulta essere

$$PI_{1-\alpha}(\mathbf{x}_{\text{new}}) := [\hat{\mu}(\mathbf{x}_{\text{new}}) - Q_{1-\alpha}(R, \mathcal{I}_2), \hat{\mu}(\mathbf{x}_{\text{new}}) + Q_{1-\alpha}(R, \mathcal{I}_2)]. \tag{4.2}$$

Questo intervallo garantisce il raggiungimento di un livello di confidenza maggiore o uguale a $1 - \alpha$, come mostrato in [Vovk et al. (2005)]. Tuttavia, in generale, gli intervalli risultanti da questo metodo tendono ad essere eccessivamente conservativi, con

un'ampiezza che rende la stima poco significativa. Inoltre, la definizione stessa degli intervalli proposta nell'equazione (4.2) rivela un ulteriore limite della conformal prediction: l'ampiezza di $PI_{1-\alpha}(\mathbf{x}_{\text{new}})$ è fissa e uguale a $2Q_{1-\alpha}(R, \mathcal{I}_2)$, indipendentemente dal valore delle covariate $\mathbf{x}_{\text{new}}$. Ciò significa che il metodo non tiene conto della variabilità dei nuovi dati e non è in grado di fornire un intervallo specifico per ogni possibile nuova osservazione, a differenza di quanto avviene con la regressione quantile.

Presentiamo ora due metodi di conformal prediction, che differiscono nel modo in cui vengono costruiti gli intervalli.

4.2.1 Jackknife+

Jacckinfe+ è un metodo presentato nel 2021 in [Barber et al. (2021)] che rappresenta un'evoluzione del più semplice metodo Jackknife.

Il metodo Jackknife originale prevede la rimozione sequenziale di una singola osservazione dal dataset di training e il ricalcolo del modello di regressione utilizzando il dataset ridotto. Ripetendo questo processo per ogni istanza, vengono ottenute diverse stime dei parametri di regressione. L'idea alla base è che queste stime ripetute catturino la variabilità dei dati e consentano di stimare l'errore di campionamento associato alle stime dei parametri.

Inizialmente, viene allenato il modello $\hat{\mu}$ su tutto il dataset di training, successivamente, viene costruito l'intervallo seguendo una procedura *leave-one-out*:

1. si considera per ogni osservazione $i = 1, \dots, n$ il dataset senza la coppia $(\mathbf{x}_i, y_i)$ e si addestra su di esso il modello di regressione $\hat{\mu}_{-i}$;

2. si calcolano gli errori assoluti, usando al posto dei residui in (4.1) le quantità

$$R_i^{\text{LOO}} := |y_i - \hat{\mu}_{-i}(\mathbf{x}_i)| \qquad i = 1, \dots, n;$$

3. si calcola

$$\tilde{Q}_{1-\alpha}(R^{\text{LOO}}) := (1-\alpha) - \text{quantile dell'insieme } \{R_i^{\text{LOO}} \mid i = 1, \dots, n\}.$$

Per una nuova istanza $\mathbf{x}_{\text{new}}$ l'intervallo di previsione è

$$PI_{1-\alpha}^{\text{Jackknife}}(\mathbf{x}_{\text{new}}) := [\hat{\mu}(\mathbf{x}_{\text{new}}) - \tilde{Q}_{1-\alpha}(R^{\text{LOO}}), \hat{\mu}(\mathbf{x}_{\text{new}}) + \tilde{Q}_{1-\alpha}(R^{\text{LOO}})].$$

Osservazione 4.2. Dal momento che $\hat{\mu}(\mathbf{x}_{\text{new}})$ non dipende dall'indice i, possiamo riscrivere in modo equivalente l'intervallo costruito come

$$PI_{1-\alpha}^{\text{Jackknife}}(\mathbf{x}_{\text{new}}) := [\tilde{Q}_{1-\alpha}^{-}(R^{\text{LOO}}), \tilde{Q}_{1-\alpha}^{+}(R^{\text{LOO}})],$$

con

$$\tilde{Q}_{1-\alpha}^{\pm}(R^{\text{LOO}}) := (1-\alpha) - \text{quantile dell'insieme } \{\hat{\mu}(\mathbf{x}_{\text{new}}) \pm R_i^{\text{LOO}} \mid i = 1, \dots, n\}. \quad (4.3)$$

La procedura leave-one-out previene il rischio di overfitting poiché i residui R_i^{LOO} sono calcolati su punti inutilizzati nell'addestramento dei modelli $\hat{\mu}_{-1}$. Tuttavia, non esistono garanzie teoriche che tale metodo soddisfi il livello di confidenza cercato, specialmente quando l'algoritmo di regressione μ risulta instabile. Questo è evidenziato in [Barber et al. (2021)] per dataset in cui il numero di variabili si avvicina al numero di osservazioni.

Jackknife+ propone un approccio nuovo: dopo aver allenato i modelli $\hat{\mu}_{-i}$ e calcolato i residui R_i^{LOO}, $i = 1, \dots, n$, l'intervallo costruito è

$$PI_{1-\alpha}^{\text{Jackknife+}}(\mathbf{x}_{\text{new}}) := [Q_{1-\alpha}^{-}(R^{\text{LOO}}), Q_{1-\alpha}^{+}(R^{\text{LOO}})], \quad (4.4)$$

dove

$$Q_{1-\alpha}^{\pm}(R^{\text{LOO}}) := (1-\alpha) - \text{quantile dell'insieme } \{\hat{\mu}_{-i}(\mathbf{x}_{\text{new}}) \pm R_i^{\text{LOO}} \mid i = 1, \dots, n\}. \quad (4.5)$$

Osservando le definizioni (4.3) e (4.5) possiamo notare la differenza tra Jackknife e Jackknife+: quest'ultimo utilizza come estremi degli intervalli di previsioni i quantili dell'insieme i cui elementi sono

$$a_i + (\hat{\mu}_{-i}(\mathbf{x}_{\text{new}}) - \hat{\mu}(\mathbf{x}_{\text{new}})) \qquad i = 1, \dots, n,$$

dove A $= \{a_i \mid i = 1, \dots, n\}$ è l'insieme su cui Jackknife classico calcola i quantili. Pertanto, mentre gli intervalli costruiti da Jaccknife sono simmetrici rispetto al valore

$\hat{\mu}(\mathbf{x}_{\text{new}})$, gli intervalli di Jackknife+ sono centrati nella mediana (ossia lo 0.5-quantile) dell'insieme $\{\hat{\mu}_{-i}(\mathbf{x}_{\text{new}}) \mid i = 1, \dots, n\}$. Dunque, Jackknife+ applica agli elementi di A n traslazioni

$$\hat{\mu}_{-i}(\mathbf{x}_{\text{new}}) - \hat{\mu}(\mathbf{x}_{\text{new}})$$

che permettono all'algoritmo di tenere conto di eventuali instabilità del modello $\hat{\mu}$ dovute alla natura dei dati. Queste considerazioni garantiscono l'efficacia del metodo Jackknife+, come afferma il prossimo teorema, la cui dimostrazione si trova in [Barber et al. (2021)].

Teorema 4.2.1. *L'intervallo di previsione* $PI_{1-\alpha}^{Jackknife+}(\mathbf{x}_{new})$ *definito come in* (4.4) *soddisfa*

$$\mathbb{P}(y_{new} \in PI_{1-\alpha}^{Jackknife+}(\mathbf{x}_{new})) \geq 1 - 2\alpha.$$

Nonostante il Teorema 4.2.1 garantisca solo un livello di confidenza di $1 - 2\alpha$, in [Barber et al. (2021)] si dimostra che, a meno di casi patologici, la copertura ottenuta da Jackknife+ si avvicina molto al valore cercato $1 - \alpha$.

CV+

Per dataset molto grandi, Jackknife+ può risultare estremamente dispendioso in termini di computazione, poiché richiede di addestrare n modelli di regressione $\hat{\mu}_{-i}$ per calcolare i quantili $Q_{1-\alpha}^{\pm}(R^{\text{LOO}})$. Per questo motivo, in [Barber et al. (2021)] viene proposta una variante chiamata CV+.

Per cominciare, si divide il dataset in K fold disgiunti $S_1, \dots, S_K$ della stessa dimensione.

Per ogni fold, si allena il modello $\hat{\mu}_{-j}$ utilizzando il dataset costituito dalle coppie

$$(\mathbf{x}_i, y_i) \qquad \text{tali che } i \in \{1, \dots, n\} \setminus S_j,$$

A questo punto si calcolano i residui

$$R_i^{\text{CV}} := \left| y_i - \hat{\mu}_{-j(i)}(\mathbf{x}_i) \right| \qquad i = 1, \dots, n,$$

dove $j(i) \in 1, \dots, K$ identifica il sottoinsieme di osservazioni che contiene i.

Possiamo infine calcolare

$$Q^{\pm}_{1-\alpha}(R^{\mathrm{CV}}) := (1-\alpha) - \text{quantile dell'insieme } \{\hat{\mu}_{-j(i)}(\mathbf{x}_{\mathrm{new}}) \pm R^{\mathrm{CV}}_i \mid i = 1, \dots, n\}$$

con i quali costruiamo gli intervalli

$$PI^{\mathrm{CV}+}_{1-\alpha}(\mathbf{x}_{\mathrm{new}}) := [Q^{-}_{1-\alpha}(R^{\mathrm{CV}}), Q^{+}_{1-\alpha}(R^{\mathrm{CV}})]. \tag{4.6}$$

Osservazione 4.3. Jackknife+ può essere considerato come un caso speciale di CV+ nel quale si sceglie $K = n$.

Il vantaggio del metodo CV+ è che, quando si sceglie $K << n$, è necessario addestrare un numero minore di modelli per ottenere gli estremi degli intervalli. Tuttavia, ciò comporta spesso che gli intervalli ottenuti siano leggermente più ampi, poiché i modelli $\hat{\mu}_{-j(i)}$ vengono addestrati utilizzando un campione di dati di dimensione inferiore, il che tende, in genere, ad aumentare leggermente i residui.

Vale il seguente Teorema.

Teorema 4.2.2. *L'intervallo di previsione ottenuto con K-fold CV+* $PI^{CV+}_{1-\alpha}(\mathbf{x}_{new})$ *definito come in* (4.6) *soddisfa*

$$\begin{aligned}\mathbb{P}(y_{new} \in PI^{CV+}_{1-\alpha}(\mathbf{x}_{new})) &\geq 1 - 2\alpha - \min\left\{\frac{2(K-1)}{n+K}, \frac{n-K}{n(K+1)}\right\} \\ &\geq 1 - 2\alpha - \sqrt{\frac{2}{n}}.\end{aligned}$$

Dimostrazione. Si veda [Barber et al. (2021)]. □

Per valori molto grandi di n, il termine $\sqrt{\frac{2}{n}}$ risulta essere trascurabile, il che significa che le garanzie di copertura del metodo CV+ non differiscono molto da quelle di Jackknife+, ma con una notevole risparmio computazionale.

4.2.2 Jackknife+ after Bootstrap

Jackknife+ after Bootstrap rappresenta un'ulteriore evoluzione del metodo Jackknife+ che sfrutta la tecnica ensemble Bootstrap vista nel Paragrafo 3.4.

Inizialmente vengono creati B Bootstrap $S_1, \ldots, S_B$ del dataset iniziale $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ e per ogni $b = 1, \ldots, B$ viene addestrato un modello $\hat{\mu}_b$ utilizzando il sottoinsieme S_b. Si sceglie poi una funzione statistica φ, ad esempio la media o la mediana.

Successivamente, per ogni $i = 1, \ldots, n$, viene seguito un procedimento analogo a quello utilizzato per il metodo CV+:

1. si definisce il modello aggregato
$$\hat{\mu}_{\phi-i} := \varphi(\{\hat{\mu}_b \mid b = 1, \ldots, B, i \in S_b\});$$
2. si definiscono i residui
$$R_i^{\mathrm{B}} := |y_i - \hat{\mu}_{\varphi-i}(\mathbf{x}_i)|\,;$$
3. si calcola per una nuova osservazione
$$Q_{1-\alpha}^{\pm}(R^{\mathrm{B}}) := (1-\alpha) - \text{quantile dell'insieme } \{\hat{\mu}_{\varphi-i}(\mathbf{x}_{\mathrm{new}}) \pm R_i^{\mathrm{B}} \mid i = 1, \ldots, n\};$$
4. si costruisce l'intervallo
$$PI_{1-\alpha}^{\text{Jackknife+ aB}}(\mathbf{x}_{\mathrm{new}}) := [Q_{1-\alpha}^{-}(R^{\mathrm{B}}), Q_{1-\alpha}^{+}(R^{\mathrm{B}})]. \tag{4.7}$$

Notiamo che, a differenza di quanto avviene nel caso della cross validation, un'osservazione i può appartenere a più di un Bootstrap, poiché viene consentita la reimmissione. Pertanto, i modelli $\hat{\mu}_{\varphi-i}$ vengono allenati sugli insiemi S_b che non contengono i.

Il Teorema 4.2.3 fornisce la garanzia di copertura per l'intervallo di previsione ottenuto tramite Jackknife+ after Bootstrap, con un'interessante variazione: il numero totale di modelli di base B deve essere estratto casualmente anziché scelto a priori.

Teorema 4.2.3. *Siano $\tilde{B} \geq 1$, $m \geq 1$ e sia φ una funzione di aggregazione. Supponiamo che il numero di Bootstrap B sia generato seguendo una distribuzione binomiale*
$$B \sim \mathrm{Bin}\left(\tilde{B}, \left(1 - \frac{1}{n+1}\right)^m\right).$$
Allora l'intervallo di previsione $PI_{1-\alpha}^{Jackknife+\ aB}(\mathbf{x}_{new})$ definito come in (4.7) soddisfa
$$\mathbb{P}(y_{new} \in PI_{1-\alpha}^{Jackknife+\ aB}(\mathbf{x}_{new})) \geq 1 - 2\alpha.$$

Dimostrazione. Si veda [Kim et al. (2020)] □

4.3 Conformalized quantile regression

La *conformalized quantile regression* (CQR) è un metodo statistico utilizzato per l'analisi dei dati che combina la regressione quantile con la conformal prediction. Invece di ottenere un singolo valore stimato per i quantili, la CQR fornisce un intervallo di confidenza per ciascun quantile desiderato. Questo intervallo di confidenza rappresenta l'incertezza associata alla stima del quantile, tenendo conto della variabilità dei dati e delle fluttuazioni dei modelli. In questo modo la CQR eredità sia la validità e l'indipendenza da ipotesi sulla distribuzione dei dati della conformal prediction che l'efficienza statistica della regressione quantile.

Vediamo ora l'algoritmo alla base dei metodi di CQR.

Si inizia, come nel caso della conformal prediction, con la suddivisione del dataset di training in un training set effettivo, indicizzato da $\mathcal{I}_1$ e un calibration set, indicizzato da $\mathcal{I}_2$.

Dato un algoritmo di regressione quantile $\mathcal{Q}$ e un livello di confidenza $1-\alpha$ e posti $\eta = \dfrac{\alpha}{2}$ e $\zeta = 1 - \dfrac{\alpha}{2}$, si addestrano due quantili condizionali $\hat{q}_\eta$ e $\hat{q}_\zeta$ sul training set.

Il prossimo step è il più importante: si calcolano i *conformity score* che quantificano l'errore commesso dall'intervallo $\hat{PI}(\mathbf{x}_i) = [\hat{q}_\eta(\mathbf{x}_i), \hat{q}_\zeta(\mathbf{x}_i)]$. Per ogni osservazione del calibration set i conformity score sono definiti come

$$E_i := \max\left(\hat{q}_\eta(\mathbf{x}_i) - y_i, y_i - \hat{q}_\zeta(\mathbf{x}_i)\right) \qquad \forall i \in \mathcal{I}_2.$$

Cerchiamo di capire come interpretare il significato degli E_i:

- Se $y_i < \hat{q}_\eta(\mathbf{x}_i)$, ovvero se l'output dell'osservazione è minore dell'estremo inferiore dell'intervallo di previsione, allora $E_i = |y_i - \hat{q}_\eta(\mathbf{x}_i)|$ misura l'entità dell'errore commesso dalla previsione dell'intervallo;

- Allo stesso modo, se l'output osservato è maggiore dell'estremo superiore predetto, i.e. $y_i > \hat{q}_\zeta(\mathbf{x}_i)$, allora $E_i = |y_i - \hat{q}_\zeta(\mathbf{x}_i)|$;

- Infine, se y_i è compreso nell'intervallo predetto, vale a dire $\hat{q}_\eta(\mathbf{x}_i) < y_i < \hat{q}_\zeta(\mathbf{x}_i)$, allora E_i è il massimo tra due numeri non positivi, dunque è anch'esso non positivo.

Il conformity score tiene quindi conto sia del difetto nella copertura che dell'eccesso.

Come ultimo step, si definisce l'intervallo di previsione relativo a una nuova osservazione con variabili input $\mathbf{x}_{\text{new}}$ come

$$PI_{1-\alpha}(\mathbf{x}_{\text{new}}) = [\hat{q}_\eta(\mathbf{x}_{\text{new}}) - Q_{1-\alpha}(E, \mathcal{I}_2), \hat{q}_\zeta(\mathbf{x}_{\text{new}}) + Q_{1-\alpha}(E, \mathcal{I}_2)], \tag{4.8}$$

dove

$$Q_{1-\alpha}(E, \mathcal{I}_2) := (1-\alpha)(1 + \frac{1}{|\mathcal{I}_2|}) - \text{quantile dell'insieme } \{E_i \mid i \in \mathcal{I}_2\}.$$

Il Teorema 4.3.1, la cui dimostrazione è riportata in [Romano et al. (2019)], garantisce il livello di confidenza cercato.

Teorema 4.3.1. *Se le coppie* $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n+1$ *sono interscambiabili, allora l'intervallo di previsione* $PI_{1-\alpha}(\mathbf{x}_{n+1})$ *definito come in* (4.8) *soddisfa*

$$\mathbb{P}(y_{n+1} \in PI_{1-\alpha}(\mathbf{x}_{n+1})) \geq 1 - \alpha.$$

Si osservi che la CQR costruisce sempre intervalli di previsione di ampiezza differente per ogni osservazione, seguendo quindi la variabilità dei dati.

Capitolo 5

Dataset

In questo capitolo introduciamo il dataset utilizzato per la previsione del field tilt, esponiamo la procedura di pulizia e preparazione dei dati (*preprocessing)* e presentiamo uno studio sull'importanza delle variabili (*feature importance).* L'elaborazione dei dati e l'applicazione degli algoritmi di regressione descritti nei precedenti capitoli sono stati svolti in ambiente Python.

I dati sono stati forniti da Soccerment, che ha elaborato i dati grezzi provenienti dall'azienda inglese Opta Sports, una delle principali società nel settore delle statistiche sportive e dell'analisi dei dati. Soccerment calcola, sulla base dei dati Opta, le metriche stagionali e relative alle partite.

5.1 Caratteristiche del dataset

Il dataset è composto da un totale di 10554 valori relativi a 91 variabili. Ogni istanza corrisponde a una delle partite dei top 5 campionati europei (Serie A italiana, Premier League inglese, LaLiga spagnola, Ligue 1 francese e Bundesliga tedesca) che si sono svolte nelle stagioni calcistiche dalla 2017-2018 alla 2022-2023. Ad ogni partita sono associati alcuni *metadati*, ossia variabili che forniscono informazioni contestuali riferite agli incontri.

I 9 metadati sono:

- `game_id`, un codice di identificazione univoco dell'incontro;

- `Date`, la data della partita;
- `home_opta_id`, un identificatore univoco della squadra in casa;
- `away_opta_id`, un identificatore univoco della squadra in trasferta;
- `home_score`, il punteggio della squadra in casa;
- `away_score`, il punteggio della squadra in trasferta;
- `season`, l'anno d'inizio del campionato in cui è stata giocata la partita. Assume valori da 2017 a 2022;
- `MatchDay`, la giornata del campionato a cui corrisponde l'incontro;
- `competition_id`, un codice di identificazione univoco del campionato.

Oltre ai metadati sono presenti 82 variabili che verranno utilizzate per la regressione. Alcune di esse si riferiscono ai dati medi stagionali delle squadre che si affrontano, altre alla prestazione ottenuta in partita dalle due squadre. Per ognuna di queste variabili, sono presenti i valori sia riferiti all'home team (precedute dalla stringa `home_`) che all'away team (precedute da `away_`). Pertanto, abbiamo 41 differenti feature per ogni squadra.

Alcune delle variabili che tengono conto delle medie stagionali terminano con la stringa `_season` e sono:

- `xG`;
- `npxG`, gli xG che non derivano da calci di rigore;
- `xGA`, la media degli xG registrati dagli avversari incontrati nell'arco della stagione;
- `npxGA`, analogo degli xGA per quanto riguarda npxG;
- `field_tilt`;
- `gpi`;
- `gpe`;

- `aerial_won_perc`, la percentuale di possessi vinti dai giocatori della squadra in seguito a un duello aereo;
- `dribbling`;
- `dribbling_against`, numero medio di dribbling subiti a partita;
- `passing_accuracy`, percentuale di passaggi riusciti;
- `passing_accuracy_against`, percentuale di passaggi riusciti dalle squadre avversarie;
- `fouls`, media falli commessi dalla squadra;
- `fouls_against`, media falli subiti dalla squadra;
- `long_pass_perc`, la percentuale di lanci lunghi sul totale di passaggi effettuati;
- `op_cross_perc`, la percentuale di cross, esclusi calci da fermo, sul totale di passaggi effettuati;
- `cross_accuracy`, la percentuale di cross riusciti;
- `one_twos`, la media di triangolazioni a partita;
- `one_twos_against`, la media di triangolazioni subite a partita;
- `poss_won_att_3rd`, la media a partita di possessi guadagnati nel terzo di campo offensivo;
- `lost_ball_def_3rd`, la media a partita di possessi persi nel terzo di campo difensivo.

Altre variabili che registrano le medie a partita dalla squadra durante la stagione sono le seguenti:

- `regular_npxG` (`regular_npxGA`), i npxG (npxGA) registrati su azione manovrata;
- `fast_npxG` (`fast_npxGA`), i npxG (npxGA) registrati su azioni di contropiede;

- `set_piece_npxG` (`set_piece_npxGA`), i npxG (npxGA) registrati su azioni di palla inattiva (calci d'angolo e calci di punizione);
- `point_per_game`, i punti totalizzati;
- `xP_per_game`, gli xP calcolati.

Le medie stagionali sono riferite alle sole partite giocate nelle medesime condizioni casa-trasferta. Ciò significa, ad esempio, che la variabile `home_xG_season` contiene la media degli xG a partita ottenuta dall'home team solo nelle partite giocate in casa. Analogamente, `away_dribbling_against` rappresenta la media a partita di dribbling che l'away team subisce quando gioca in trasferta. Questa distinzione ci permette di analizzare se ci sono diversi comportamenti nei dati delle squadre dovuti al giocare in casa o in trasferta.

Sono infine presenti variabili che registrano alcune statistiche ottenute durante l'incontro dalle due squadre. Queste variabili, che terminano con il suffisso `_game`, sono:

- `xG`;
- `npxG`;
- `field_tilt`;
- `aeriel_won_perc`;
- `dribbling`;
- `passing_accuracy`;
- `fouls`;
- `long_pass_perc`;
- `op_cross_perc`;
- `cross_accuracy`;
- `one_twos`;
- `poss_won_att_3rd`.

5.2 Preprocessing

Prima di applicare gli algoritmi di regressione è necessario studiare le caratteristiche del dataset, selezionando prima le variabili a cui siamo più interessati.

Il primo step consiste nel separare il dataset di training da quello di testing. Si è scelto di valutare le performance finali degli algoritmi su un dataset che ha come osservazioni le partite della Premier League della stagione 2022-2023. Tutte le altre partite fanno parte del dataset di training.

Per entrambi i dataset ottenuti vengono mantenute solo le osservazioni che corrispondono a partite con `MatchDay` > 4. Il motivo di questo filtraggio è che le medie stagionali calcolate su un numero di partite inferiore (tenendo anche conto dell'alternanza casa-trasferta) sono poco significative. Una volta filtrato il dataset, è necessario rimuovere le osservazioni che contengono valori `NaN` (*Not a Number*) o *outliers*, cioè valori che si discostano in modo significativo dal resto dei dati e la cui presenza può essere dovuta a errori di misurazione o anomalie statistiche. Per individuare gli outliers da eliminare si è sfruttata la funzione `LocalOutlierFactor` del pacchetto `neighbors` della libreria `sklearn`. Dopo aver applicato l'algoritmo, rimangono 8533 osservazioni per il dataset di training e 264 per quello di testing.

Si è poi proceduto con la scelta delle variabili indipendenti. Dal momento che siamo interessati a predire i valori della variabile `home_field_tilt_game` e della variabile `away_field_tilt_game`, non possiamo sfruttare altre variabili che rappresentano statistiche della partita stessa. Abbiamo inoltre deciso di usare come predittori per il field tilt solo alcune metriche offensive della squadra in esame e alcune metriche difensive della squadra avversaria.

Elenchiamo qui sotto le covariate utilizzate. Le variabili sono da intendersi con la stringa `away_` al posto di `opp_`, per l'home team, mentre `home_` sostituisce `opp_` per l'away team. Tutti gli altri regressori includono anche la stringa `home_` (`away_`) per la squadra in casa (in trasferta).

```
[regular_npxG, fast_npxG, set_piece_npxG,\
opp_regular_npxGA, opp_fast_npxGA, opp_set_piece_npxGA,\
field_tilt_season, field_tilt_season,\
```

```
xP_per_game, opp_xP_per_game, gpi_season, gpe_season,\
aerial_won_perc_season, opp_aerial_won_perc_season,\
dribbling_season, opp_dribbling_against_season,\
passing_accuracy_season, opp_passing_accuracy_against_season,\
long_pass_perc_season, op_cross_perc_season, cross_accuracy_season,\
one_twos_season, opp_one_twos_against_season,\
poss_won_att_3rd_season, opp_lost_ball_def_3rd_season]
```

5.3 Feature importance

SHAP (*Shapley Additive Explanations*) è un approccio innovativo per l'interpretazione dei modelli di Machine Learning. Al suo centro si trova il concetto di *valori Shapley*, derivato dalla teoria dei giochi, che rappresenta una misura dell'importanza relativa di ciascun giocatore all'interno di un gioco cooperativo. I valori Shapley misurano l'incremento di utilità che ogni giocatore apporta quando si unisce al gruppo, tenendo conto dell'interazione tra i giocatori e assegnando loro un peso adeguato. Nel contesto del Machine Learning, i valori Shapley vengono utilizzati per assegnare l'importanza delle variabili coinvolte nella predizione di un modello. Vengono calcolati considerando tutte le possibili combinazioni di variabili e misurando il contributo marginale di ciascuna di esse nel processo decisionale del modello. Questo approccio permette di identificare i fattori che influenzano maggiormente le predizioni, fornendo una stima dell'importanza di ogni variabile e consentendo una comprensione più approfondita del loro effetto sulle predizioni del modello. Ciò contribuisce a rendere le decisioni dei modelli più trasparenti, interpretabili e giustificabili. La libreria `shap` permette di calcolare i valori SHAP ottenuti a partire da un algoritmo di regressione addestrato.

Presentiamo l'analisi dei valori SHAP sul training set in seguito all'applicazione del miglior algoritmo di regressione trovato. I dettagli sulla scelta dell'algoritmo sono riportati nel Capitolo 6.

Home Team

La Figura 5.3 mostra la media dei valori assoluti SHAP tra tutte le osservazioni. La tendenza delle squadre in termini di field tilt portata avanti durante l'arco della stagione risulta essere il fattore più determinante per la previsione del field tilt in partita della squadra di casa. In particolare, il field tilt dell'avversario è l'elemento più impattante per l'algoritmo di regressione. Subito dopo le variabili `_field_tilt_season`, vediamo gli xP medi della squadra in trasferta e quelli della squadra in casa. Ciò significa che la forza delle due squadre, in particolare quella dell'avversario, ha un'importanza estremamente rilevante nel predire il comportamento in termini di stile di gioco e di predominio territoriale.

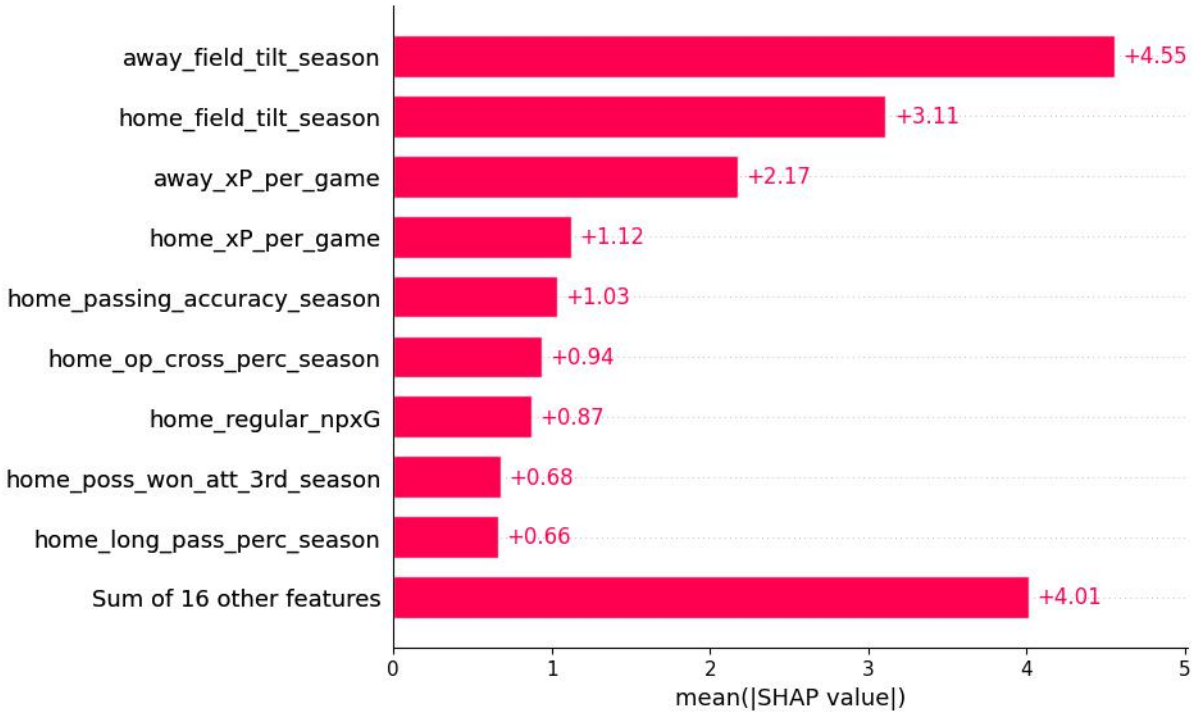


Figura 5.1: Valori assoluti medi SHAP, home team

Il grafico rappresentato nella Figura 5.3, detto *Beeswarm plot*, ci permette di visualizzare la distribuzione dei valori SHAP per le covariate più determinanti. Sull'ascissa sono rappresentati i valori SHAP, mentre i colori sono in relazione con il valore della variabile

associata: all'aumentare del valore della covariata, i colori passano dal blu al rosso. Possiamo dunque comprendere che, come era prevedibile, alti valori del field tilt stagionale dell'avversario quando gioca in trasferta, così come una media di xP maggiore, portano a una diminuzione della previsione della variabile `home_field_tilt_game`. Al contrario, all'aumentare della media stagionale del proprio field tilt o del numero di npxG su azione manovrata, cresce anche la previsione del field tilt per la partita.

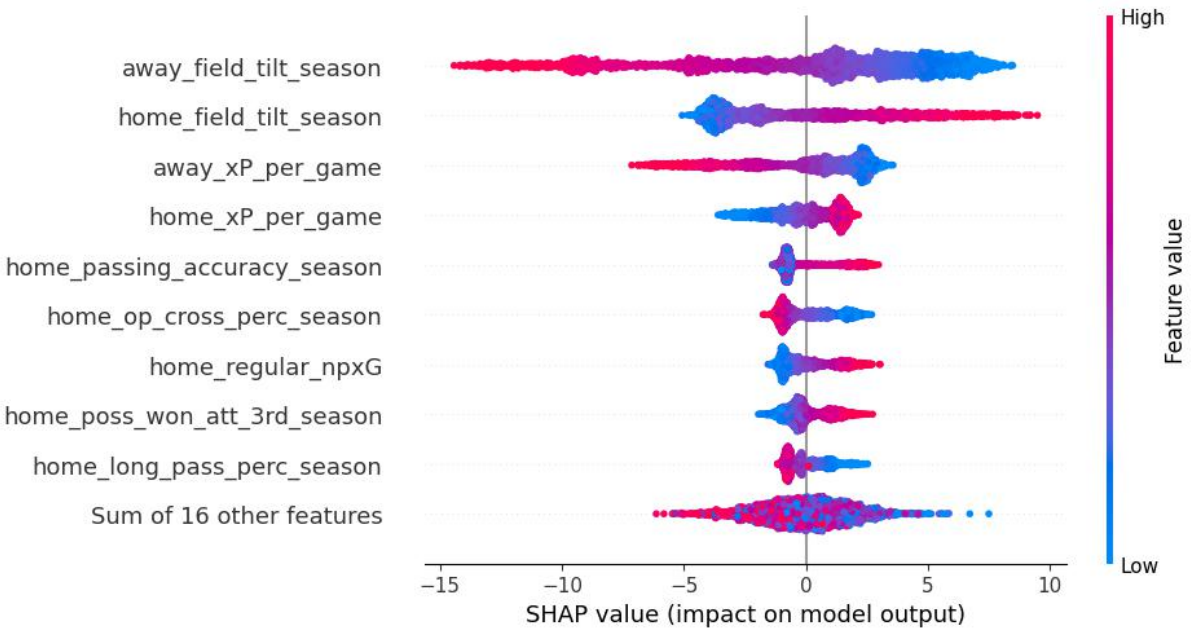


Figura 5.2: Beeswarm plot, home team

Away Team

Analizzando la media dei valori assoluti dei valori SHAP delle covariate, mostrati nella Figura 5.3, notiamo che, come nel caso dell'home team, l'impatto del field tilt stagionale e degli xP medi risulta prevalente per la previsione della variabile `away_field_tilt_game`. Tuttavia, i valori tra casa e trasferta risultano scambiati rispetto a quanto visto per la squadra in casa: sono le statistiche dell'avversario (dell'home team in questo caso) a dare un contributo maggiore.

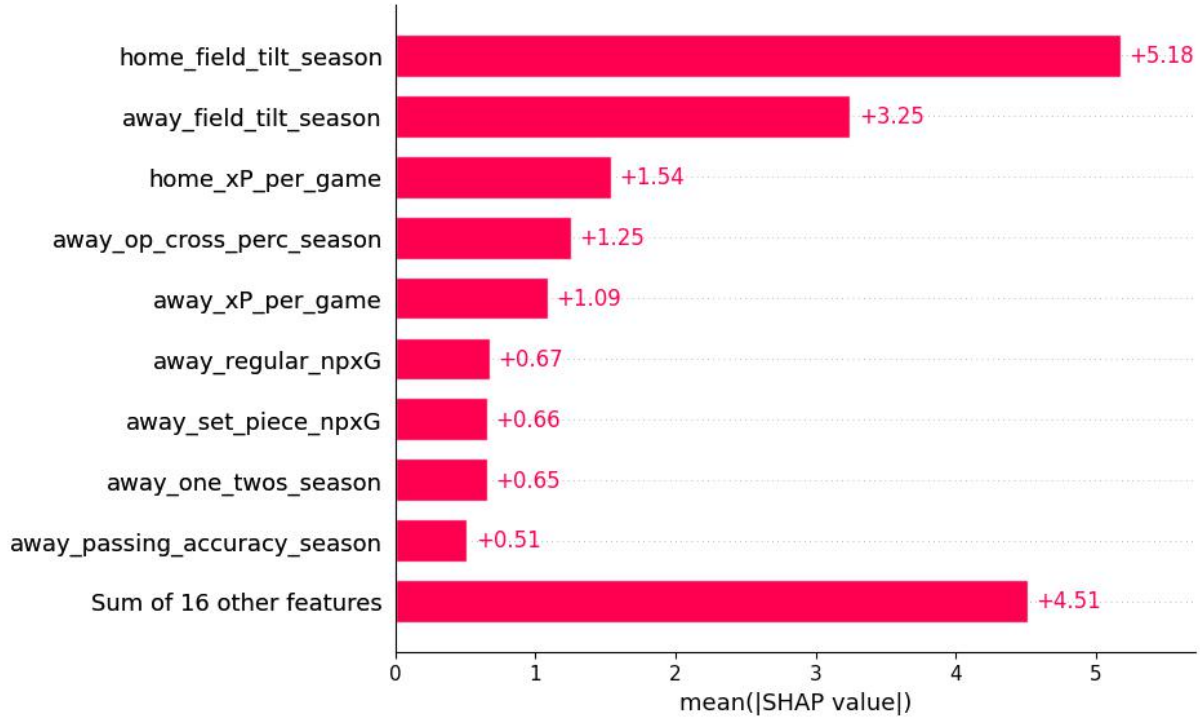


Figura 5.3: Valori assoluti medi SHAP, away team

Il Beeswarm plot riportato nella Figura 5.3 mostra che, come ci aspettavamo, bassi valori di field tilt stagionale dell'avversario tendono ad aumentare la previsione del field tilt dell'away team per la partita. Lo stesso accade quando consideriamo la percentuale di cross in open play effettuati sul totale dei passaggi da parte dell'away team: quanto più numerosi sono i palloni giocati dalla squadra tramite cross su azione manovrata, tanto più basso è il valore atteso di field tilt. Il comportamento è opposto quando prendiamo in considerazione statistiche come gli xP medi dell'away team o il numero di gol arrivati da palla inattiva.

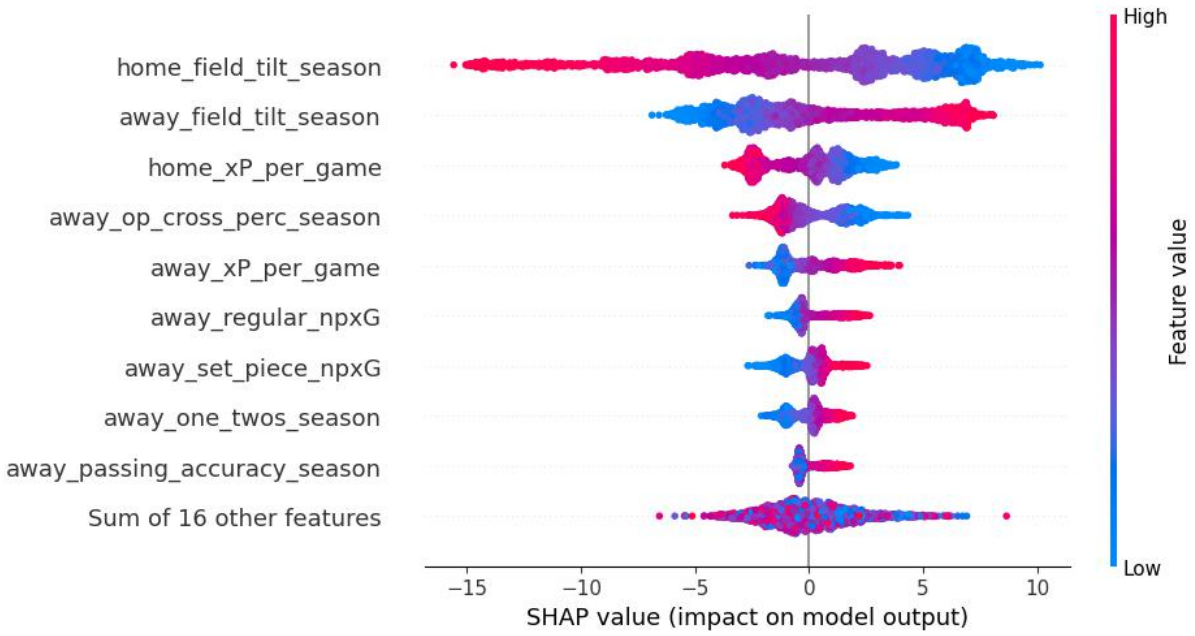


Figura 5.4: Beeswarm plot, away team

Capitolo 6

Risultati training

In questo capitolo presentiamo i risultati raggiunti in seguito all'applicazione degli algoritmi descritti nei Capitoli 3 e 4 sul dataset di training.

6.1 Previsione puntuale

Vediamo i risultati ottenuti in fase di allenamento dai modelli di regressione per ottenere una stima puntuale del field tilt.

Per ognuno dei metodi studiati, il modello è stato allenato riducendo una loss function L. La scelta principale per la loss function è stata il *coefficiente di determinazione* R^2, definito come segue.

Definizione 6.1.1 (R^2)**.** Siano $\mathbf{y} = (y_1, \dots, y_n)^T$ e $\hat{\mathbf{y}} = (\hat{y}_1, \dots, \hat{y}_n)^T$, rispettivamente, la variabile obiettivo e la sua predizione ottenuta tramite la stima $\hat{f}$ di un modello f definito come in (2.1), con n numero di osservazioni. Si definisce *coefficiente di determinazione* tra $\mathbf{y}$ e $\hat{\mathbf{y}}$ la quantità

$$R^2 := 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2},$$

dove

$$\bar{y} := \frac{1}{n}\sum_{i=1}^{n} y_i.$$

L'R^2 rappresenta la proporzione della varianza di $\mathbf{y}$ spiegabile dalle variabili indipendenti del modello. Fornisce un'indicazione della bontà di adattamento del modello ai dati osservati e quindi misura la probabilità che campioni non osservati siano previsti dal modello, attraverso la proporzione di varianza spiegata.

Il miglior punteggio possibile è 1, nel caso in cui il modello sia in grado di predire esattamente tutti i valori, e può essere negativo, qualora il modello predica valori significativamente lontani da quelli esatti. Un modello costante che predice sempre il valore medio di $\mathbf{y}$, senza tener conto delle variabili in ingresso, otterrebbe come punteggio 0.

Nel caso del coefficiente di determinazione, minimizzare la loss function significa cercare di ottenere il valore R^2 sul validation set più alto possibile.

Si è anche tenuto conto di un'altra funzione di loss, ossia la radice quadrata dell'*errore quadratico medio*, o RMSE (dall'inglese *root-mean squared error*).

Definizione 6.1.2 (MSE)**.** Siano $\mathbf{y}$ e $\hat{\mathbf{y}}$ come nella Definizione 6.1.1. Chiamiamo *errore quadratico medio* la quantità

$$\text{MSE} := \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

Estraendo la radice quadrata dal MSE, otteniamo il RMSE. Chiaramente, quanto minore risulta il RMSE, tanto migliori sono le nostre predizioni.

Vediamo ora i risultati ottenuti da ciascun metodo in seguito alla cross validation. I valori finali dell'R^2 e del RMSE sono riassunti nelle Tabelle 6.3 e 6.4.

6.1.1 Regressione lineare

La regressione lineare è stata implementata attraverso la funzione `LinearRegression` del pacchetto `linear_model` di `sklearn`. Non ci sono iperparametri da ottimizzare. Vediamone i risultati applicati sul dataset standardizzato, relativamente alle variabili dipendenti `home_field_tilt_game` e `away_field_tilt_game`.

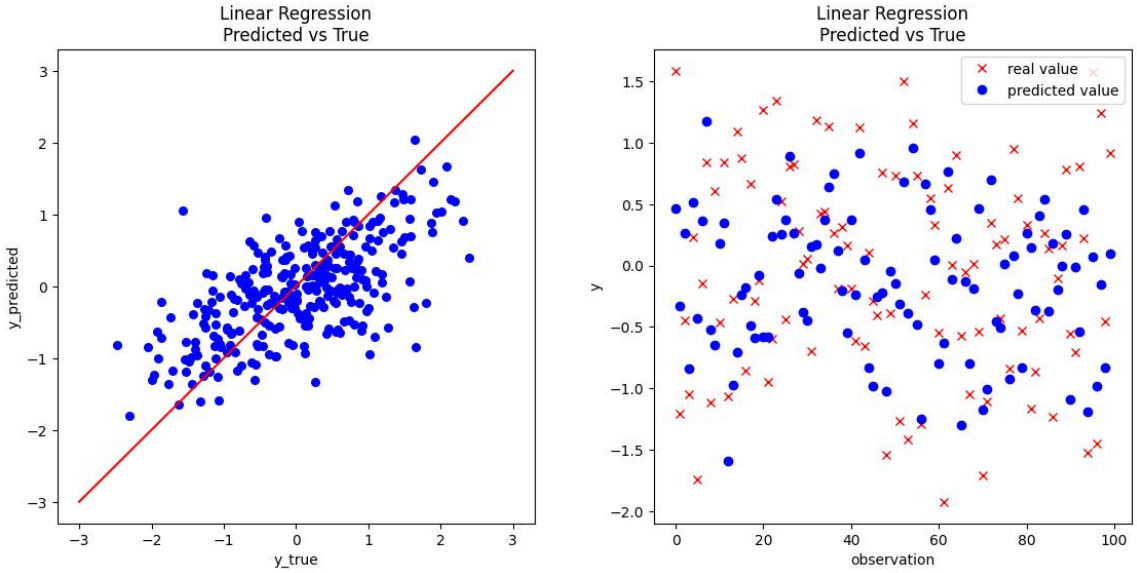
Home Team

I valori riportati sotto corrispondono a media ± deviazione standard del coefficiente di

determinazione e della radice quadrata dell'errore quadratico medio calcolati nei 5 step della cross validation.

$$R^2 = (0.411 \pm 0.011);$$
$$\text{RMSE} = (0.767 \pm 0.008).$$

La prossima figura mostra due differenti scatterplot. Nel primo, sull'asse delle ascisse sono riportati i valori reali delle prime 300 osservazioni del validation set, mentre sull'asse delle ordinate sono rappresentati i corrispondenti valori predetti. La diagonale rossa rappresenta la linea ideale in cui i punti dovrebbero allinearsi per una predizione accurata. Quanto più un punto si avvicina a questa diagonale rossa, tanto più accurata è la predizione relativa a quell'osservazione. Nel secondo grafico, sull'asse delle ascisse sono invece rappresentati i numeri delle osservazioni, mentre sull'asse delle ordinate sono mostrati i valori reali delle osservazioni (rappresentati da croci) e i valori predetti dall'algoritmo (rappresentati da pallini). Se una croce e un pallino si sovrappongono sulla stessa ascissa, allora la predizione è considerata perfetta. Al contrario, se sono distanti verticalmente, significa che la previsione si è discostata dal valore osservato.



(a) Scatterplot delle prime 300 osservazioni (b) Grafico delle prime 100 osservazioni

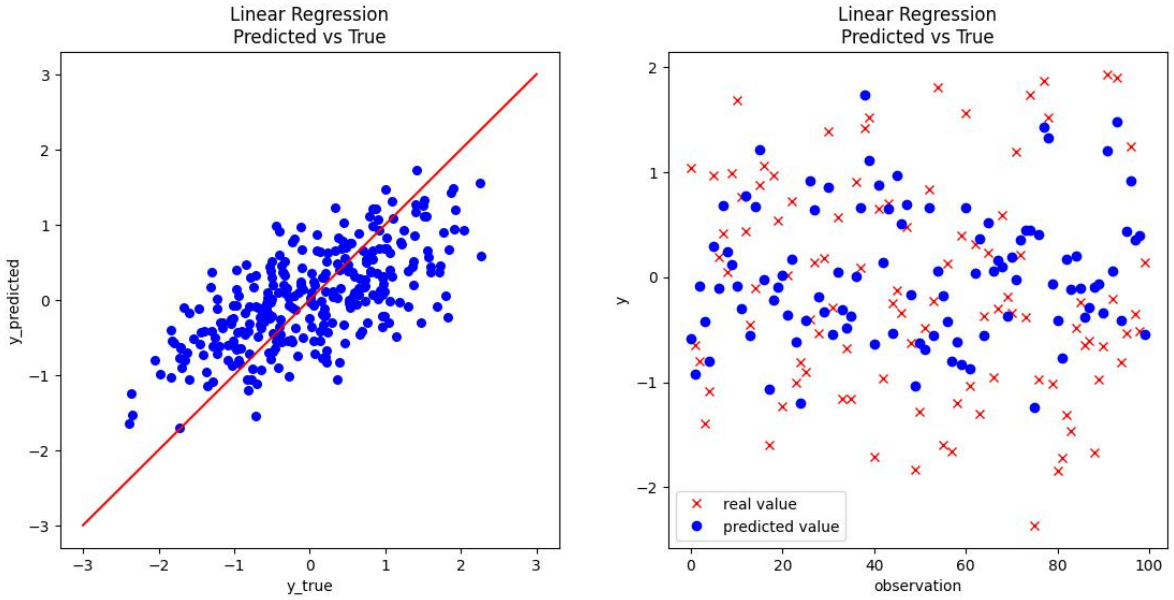
Figura 6.1: Previsione vs valore reale, home team, regressione lineare

Away Team

Per quanto riguarda le predizioni della squadra in trasferta, i punteggi ottenuti sono:

$$R^2 = (0.406 \pm 0.018);$$
$$\text{RMSE} = (0.770 \pm 0.011).$$

La Figura 6.2 è l'analogo della Figura 6.1 per quanto riguarda l'away team.



(a) Scatterplot delle prime 300 osservazioni

(b) Grafico delle prime 100 osservazioni

Figura 6.2: Previsione vs valore reale, away team, regressione lineare

6.1.2 SVR

La SVR è stata implementata tramite la funzione di `sklearn SVR`, usando `'rbf'` come kernel function e ottimizzando come iperparametri[1] ϵ e il parametro di regolarizzazione C, scegliendo fra:

- *epsilon* : $[0.2, 0.3, 0.4, 0.5]$;

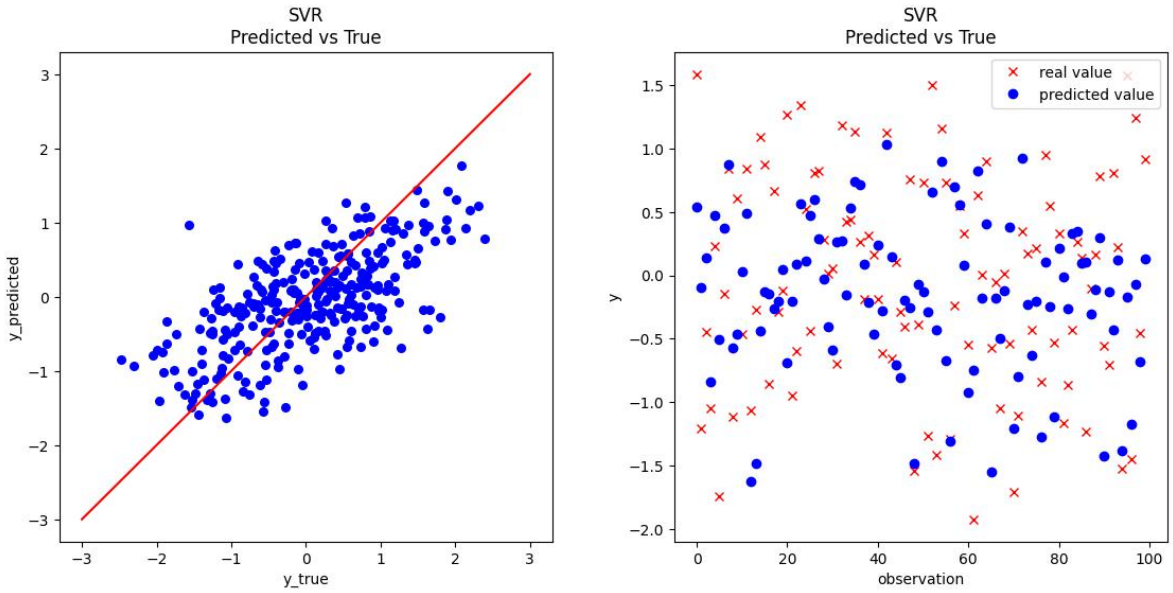
[1]Le ricerche degli iperparametri di tutti i metodi sono state effettuate combinando l'uso della funzione `GridSearchCV` del modulo `model_selection` di `sklearn` e verifiche empiriche dei risultati.

- $C : [0.2, 0.3, 0.4, 0.5, 0.6]$.

Riportiamo ora le migliori combinazioni di iperparametri per la squadra di casa e di trasferta e gli score ottenuti, mentre le Figure 6.3 e 6.4 mostrano i relativi scatterplot.

Home Team $epsilon = 0.3$, $C = 0.2$.
I punteggi registrati sono:

$$R^2 = (0.410 \pm 0.004);$$
$$\text{RMSE} = (0.768 \pm 0.010).$$



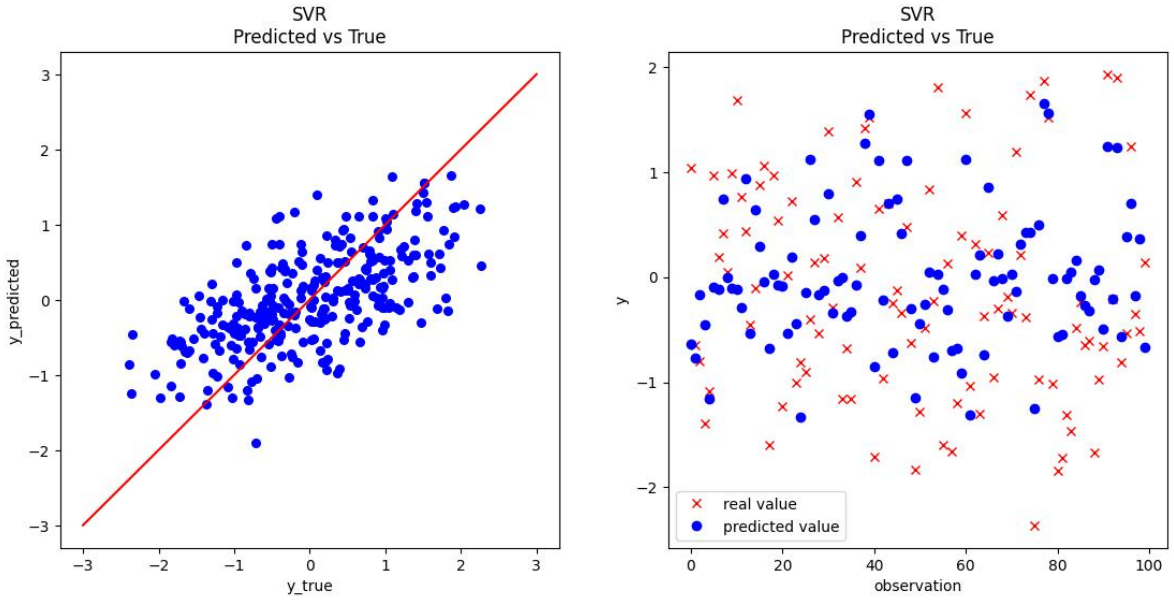
(a) Scatterplot delle prime 300 osservazioni

(b) Grafico delle prime 100 osservazioni

Figura 6.3: Previsione vs valore reale, home team, SVR

Away Team $epsilon = 0.3$, $C = 0.3$.
I corrispettivi score sono:

$$R^2 = (0.396 \pm 0.018);$$
$$\text{RMSE} = (0.777 \pm 0.010).$$



(a) Scatterplot delle prime 300 osservazioni

(b) Grafico delle prime 100 osservazioni

Figura 6.4: Previsione vs valore reale, away team, SVR

6.1.3 Decision Trees

L'implementazione del modello di alberi decisionali è stata effettuata sfruttando l'algoritmo `DecisionTreeRegressor`, presente nel metodo `tree` di `sklearn`.

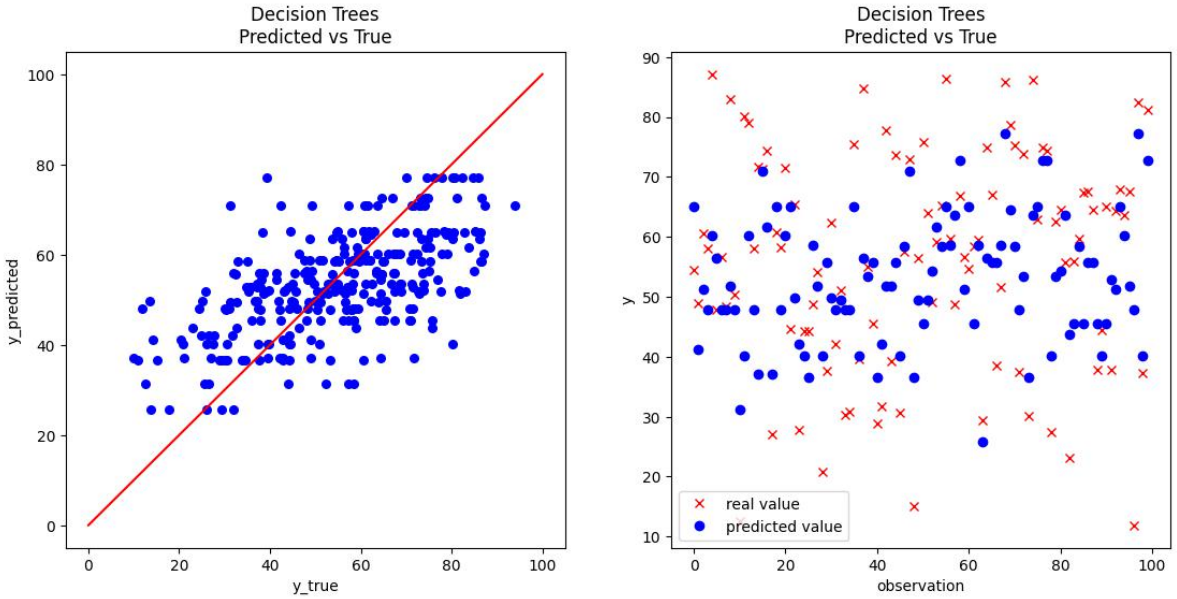
La miglior combinazione di iperparametri è stata ottenuta scegliendo tra:

- max_depth : $[4, 5, 6, 7, 8, 9]$;
- $max\_features$: [`'sqrt'`, `None`];
- $min\_samples\_leaf$: $[100, 115, 125, 130, 135, 140, 150, 165]$.

Home Team $max\_depth = 7$, $max\_features =$ `None`, $min\_samples\_leaf = 125$.
L'utilizzo di tali iperparametri ha permesso di raggiungere questi punteggi di R^2 e RMSE:

$$R^2 = (0.373 \pm 0.015);$$
$$\text{RMSE} = (13.867 \pm 0.151).$$

Si noti che il considerevole aumento del RMSE è dovuto al fatto che il dataset non è stato standardizzato.



(a) Scatterplot delle prime 300 osservazioni

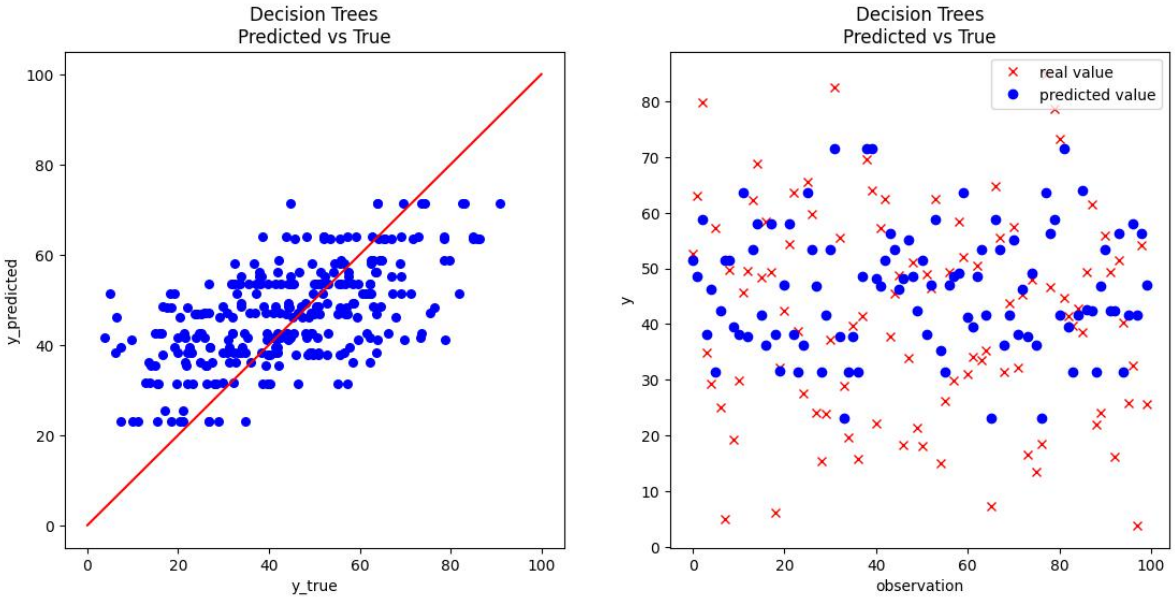
(b) Grafico delle prime 100 osservazioni

Figura 6.5: Previsione vs valore reale, home team, Decision Trees

Osservando le Figure 6.5 e 6.6 vediamo che l'algoritmo di Decision Trees presenta difficoltà nel predire correttamente i valori delle variabili dipendenti. Analizzando le figure di sinistra, si nota che i punti non si distribuiscono vicino alla diagonale come ci si aspetterebbe, ma si dispongono su righe orizzontali. Questo suggerisce che l'algoritmo riesce ad attribuire solo un numero limitato di valori alla variabile dipendente. Di conseguenza, risulta necessario provare algoritmi basati su alberi decisionali più complessi, come Random Forest o Gradient Boosting.

Away Team $max\_depth = 6$, $max\_features =$ `None`, $min\_samples\_leaf = 130$.
I risultati della previsione della variabile `away_field_tilt_game` sono:

$$R^2 = (0.363 \pm 0.012);$$
$$\text{RMSE} = (13.969 \pm 0.096).$$



(a) Scatterplot delle prime 300 osservazioni

(b) Grafico delle prime 100 osservazioni

Figura 6.6: Previsione vs valore reale, away team, Decision Trees

6.1.4 Random Forest

Aumentiamo la complessità del modello, utilizzando l'algoritmo `RandomForestRegressor` di `sklearn`, appartenente al metodo `ensemble`.

Gli iperparametri ottimizzati sono:

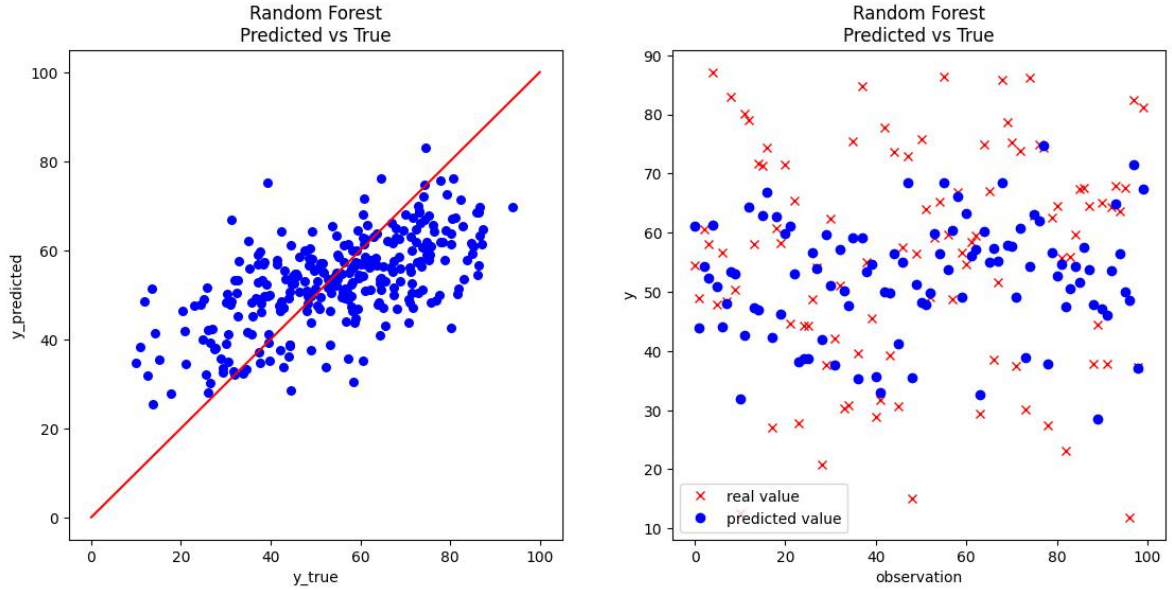
- $n\_estimators$: $[700, 750, 800, 850, 900, 950, 1000]$;
- $max\_features$: $[$`'sqrt'`, `None`$]$;
- max_depth : $[18, 19, 20, 21, 22]$.

Vediamo i risultati ottenuti per la squadra di casa e di trasferta, seguiti dai consueti scatterplot.

Home Team $n\_estimators = 900$, $max\_features =$ `'sqrt'`, $max\_depth = 18$. Gli score associati sono:

$$R^2 = (0.418 \pm 0.016);$$
$$\text{RMSE} = (13.363 \pm 0.172).$$

Dai punteggi delle nostre loss function si evince che le previsioni ottenute con il nuovo metodo sono molto più accurate rispetto a quelle del metodo precedente. Ciò è evidenziato anche nei grafici della Figura 6.7.



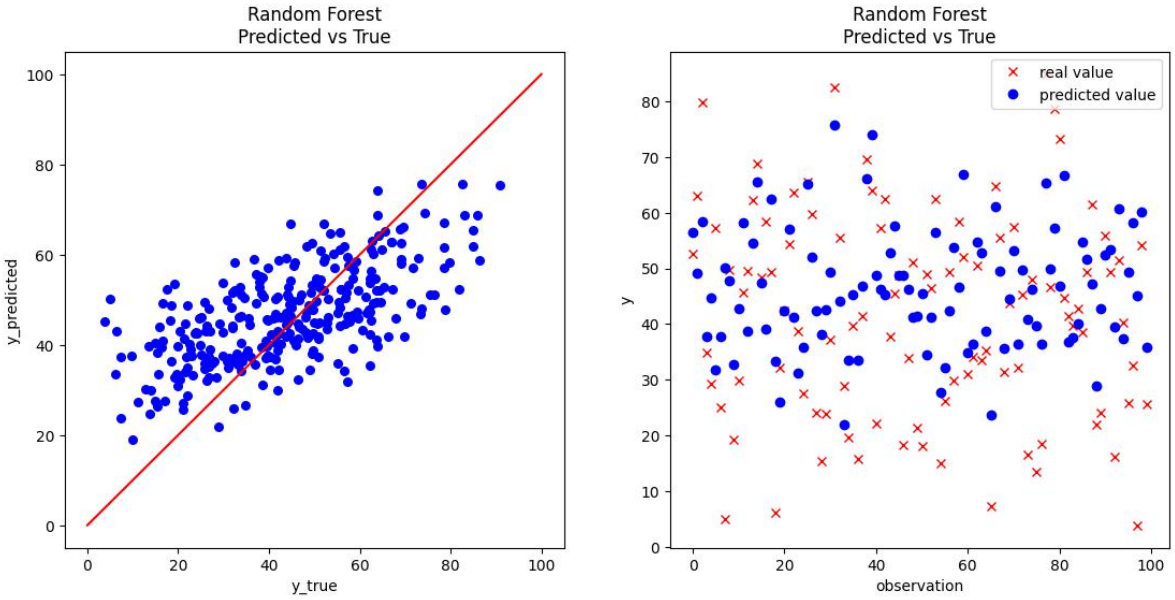
(a) Scatterplot delle prime 300 osservazioni

(b) Grafico delle prime 100 osservazioni

Figura 6.7: Previsione vs valore reale, home team, Random Forest

Away Team $n\_estimators = 1000$, $max\_features =$ `'sqrt'`, $max\_depth = 21$. Come nel caso della variabile `home_field_tilt_game`, anche qui tanto gli score quanto i grafici della Figura 6.8 mostrano un miglioramento delle prestazioni nel passare da Decision Trees a Random Forest.

$$R^2 = (0.416 \pm 0.012);$$
$$\text{RMSE} = (13.375 \pm 0.093).$$



(a) Scatterplot delle prime 300 osservazioni

(b) Grafico delle prime 100 osservazioni

Figura 6.8: Previsione vs valore reale, away team, Random Forest

6.1.5 Gradient Boosting

Per selezionare il metodo di Gradient Boosting con le migliori prestazioni, abbiamo condotto una cross-validation.

La funzione `GradientBoostingRegressor` del modulo `ensemble` all'interno della libreria `sklearn` è stata utilizzata per testare il modello base di Gradient Boosting; l'algoritmo XGBoost è stato applicato tramite l'implementazione `XGBRegressor` del pacchetto `xgboost`, mentre la funzione utilizzata per il LightGBM è `LGBMRegressor` del pacchetto `lightgbm`. Infine, per effettuare la regressione tramite CatBoost è stata utilizzata la funzione `CatBoostRegressor`, propria del pacchetto `catboost`. Gli iperparametri sono stati mantenuti al valore di default.

I valori delle variabili `home_field_tilt_game` e `away_field_tilt_game` ottenuti tramite cross validation sono riportati, rispettivamente, nelle Tabelle 6.1 e 6.2 nel formato media $\pm$ deviazione standard.

Notiamo immediatamente che XGBoost performa nettamente peggio rispetto agli al-

tri regressori, i quali presentano valori di R^2 che appartengono al range $[0.39, 0.43]$ e di RMSE all'intervallo $[13.2, 13.7]$, per quanto riguarda `home_field_tilt_game` e negli intervalli $[0.39, 0.44]$ e $[13.2, 13.7]$, rispettivamente di R^2 e RMSE, relativamente a `away_field_tilt_game`.

	R^2	RMSE
`GradientBoostingRegressor`	0.415 ± 0.014	13.40 ± 0.16
`XGBRegressor`	0.333 ± 0.023	14.30 ± 0.23
`LGBMRegressor`	0.406 ± 0.013	13.50 ± 0.13
`CatBoostRegressor`	0.410 ± 0.013	13.45 ± 0.15

Tabella 6.1: Selezione metodo di Gradient Boosting, home team

	R^2	RMSE
`GradientBoostingRegressor`	0.416 ± 0.015	13.37 ± 0.12
`XGBRegressor`	0.334 ± 0.015	14.28 ± 0.13
`LGBMRegressor`	0.406 ± 0.014	13.49 ± 0.13
`CatBoostRegressor`	0.410 ± 0.015	13.44 ± 0.14

Tabella 6.2: Selezione metodo di Gradient Boosting, away team

Poiché i valori ottenuti sono molto simili tra loro, abbiamo deciso di procedere con l'ottimizzazione degli iperparametri per tutti i metodi di Gradient Boosting, ad eccezione di XGBoost. I risultati di queste ottimizzazioni sono presentati nei prossimi paragrafi.

Metodo base

Gli iperparametri ottimizzati per questo metodo sono stati i seguenti:

- $learning\_rate$: $[0.005, 0.01, 0.015, 0.02, 0.025, 0.03]$;
- max_depth : $[1, 2, 3, 4, 5]$;

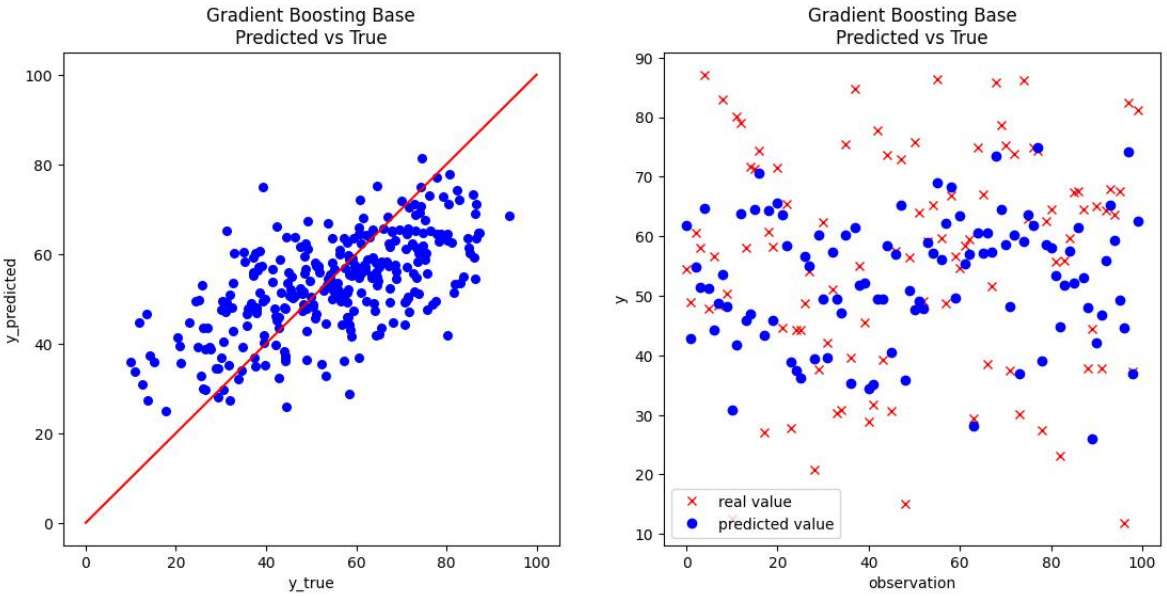
- $n\_estimators$: $[500, 550, 600, 650, 700, 750, 800, 850, 900]$.

Ecco le migliori combinazioni, con i relativi risultati.

Home Team $learning\_rate = 0.025$, $max\_depth = 2$, $n\_estimators = 650$.
Gli score ottenuti sono:

$$R^2 = (0.418 \pm 0.012);$$
$$\text{RMSE} = (13.363 \pm 0.132).$$

Applicando il modello al validation set otteniamo i grafici mostrati dalla Figura 6.9.



(a) Scatterplot delle prime 300 osservazioni

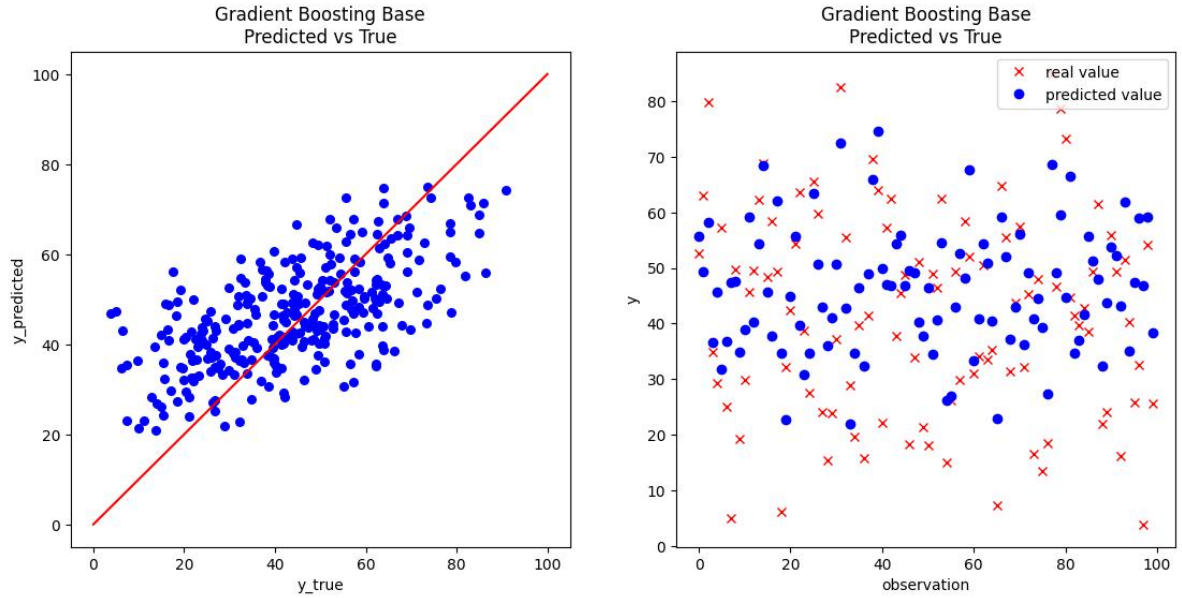
(b) Grafico delle prime 100 osservazioni

Figura 6.9: Previsione vs valore reale, home team, Gradient Boosting base

Away Team $learning\_rate = 0.02$, $max\_depth = 2$, $n\_estimators = 800$.
Con tali iperparametri, otteniamo come score:

$$R^2 = (0.417 \pm 0.013);$$
$$\text{RMSE} = (13.336 \pm 0.123).$$

Come sempre, nella Figura 6.10 vediamo un confronto tra le previsioni e i valori reali del validation set.



(a) Scatterplot delle prime 300 osservazioni

(b) Grafico delle prime 100 osservazioni

Figura 6.10: Previsione vs valore reale, away team, Gradient Boosting base

LightGBM

Per questo metodo di Gradient Boosting si è deciso di ottimizzare gli iperparametri scegliendo la combinazione migliore tra:

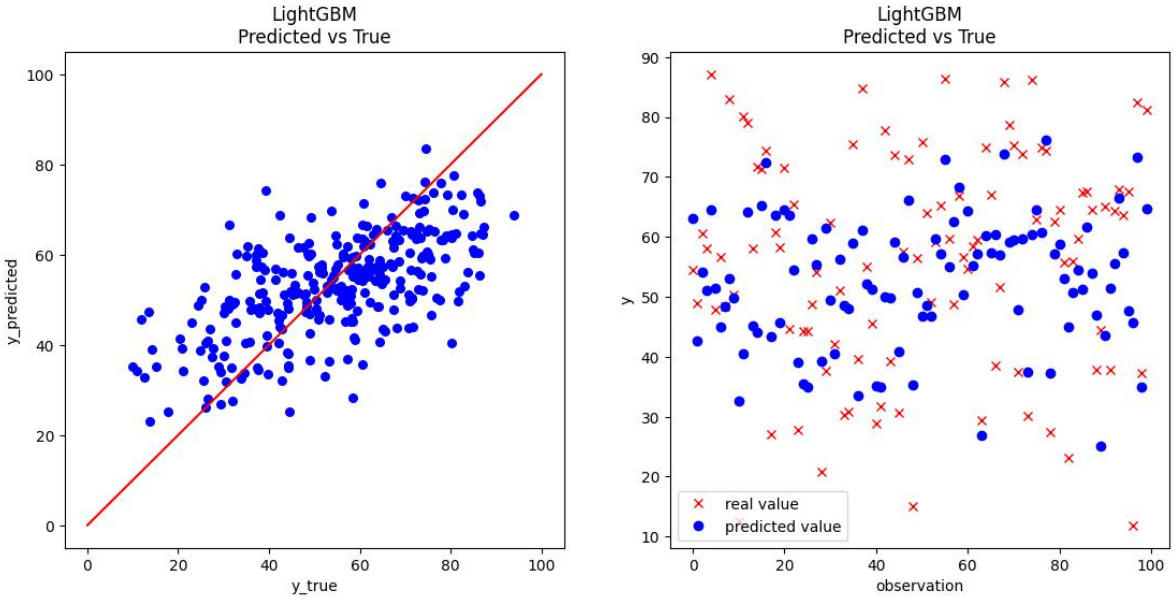
- $learning\_rate$: $[0.01, 0.015, 0.02, 0.025, 0.03]$;
- max_depth : $[3, 4, 5, 6, 7, 8]$;
- $min\_child\_samples$: $[50, 70, 90, 110, 130]$;
- $n\_estimators$: $[300, 350, 400, 450, 500, 550, 600]$;
- num_leaves : $[10, 12, 14, 15, 16, 20, 30, 32, 50, 64, 80, 100, 120, 128]$.

Vediamo i risultati ottenuti con le migliori combinazioni.

Home Team $learning\_rate = 0.015$, $max\_depth = 4$, $min\_child\_samples = 110$, $n\_estimators = 500$, $num\_leaves = 14$.
I valori relativi di R^2 e RMSE sono:

$$R^2 = (0.420 \pm 0.011);$$
$$\text{RMSE} = (13.363 \pm 0.121).$$

La Figura 6.11 mostra, al solito, il confronto tra i primi valori predetti e i rispettivi valori reali.



(a) Scatterplot delle prime 300 osservazioni

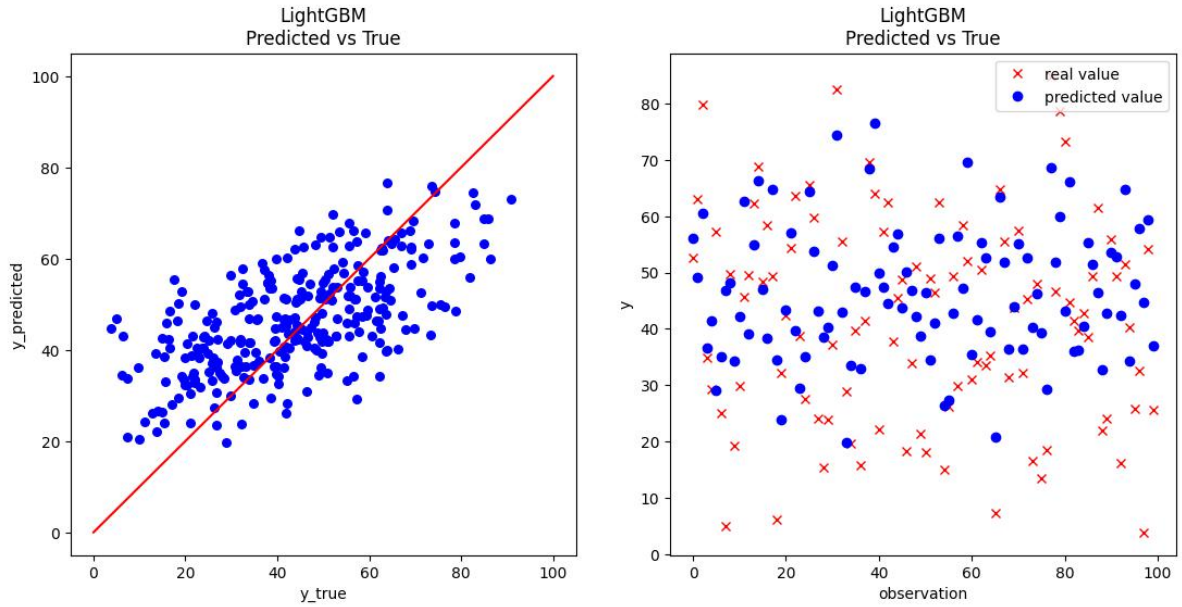
(b) Grafico delle prime 100 osservazioni

Figura 6.11: Previsione vs valore reale, home team, LightGBM

Away Team $learning\_rate = 0.02$, $max\_depth = 7$, $min\_child\_samples = 70$, $n\_estimators = 350$, $num\_leaves = 15$.
I punteggi ottenuti dalle loss function sono:

$$R^2 = (0.419 \pm 0.013);$$
$$\text{RMSE} = (13.336 \pm 0.112).$$

L'usuale confronto tra le previsioni e i valori reali è presentato nella Figura 6.12.



(a) Scatterplot delle prime 300 osservazioni

(b) Grafico delle prime 100 osservazioni

Figura 6.12: Previsione vs valore reale, home team, LightGBM

CatBoost

I possibili valori degli iperparametri erano:

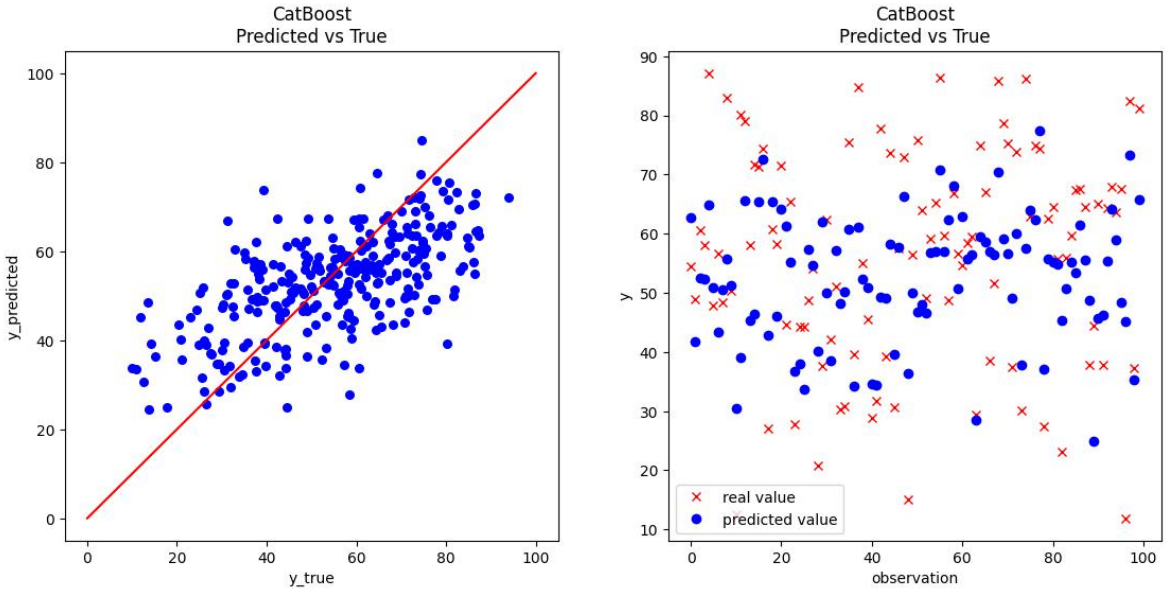
- $depth$: $[5, 6, 7, 8]$;
- $iterations$: $[800, 850, 900, 950, 1000]$;
- $learning\_rate$: $[0.01, 0.015, 0.02, 0.025]$.

Proponiamo ora i risultati ottenuti per home team e away team.

Home Team $depth = 7$, $iterations = 900$, $learning\_rate = 0.015$.
Ottenendo:

$$R^2 = (0.425 \pm 0.013);$$
$$\text{RMSE} = (13.281 \pm 0.145).$$

Confrontiamo valori predetti e reali con la Figura 6.13.



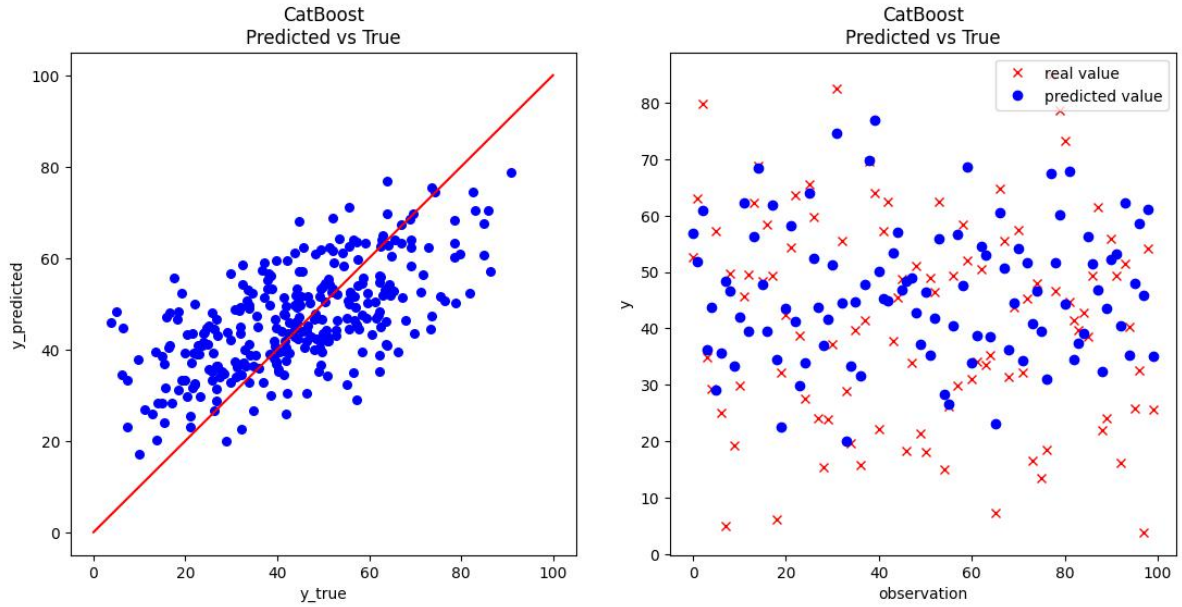
(a) Scatterplot delle prime 300 osservazioni (b) Grafico delle prime 100 osservazioni

Figura 6.13: Previsione vs valore reale, home team, CatBoost

Away Team $depth = 7$, $iterations = 1000$, $learning\_rate = 0.015$.
Con tali iperparametri, otteniamo come score:

$$R^2 = (0.425 \pm 0.014);$$
$$\text{RMSE} = (13.269 \pm 0.113).$$

Concludiamo la presentazione dei risultati di training con gli scatterplot delle previsioni, mostrati nella Figura 6.14.



(a) Scatterplot delle prime 300 osservazioni

(b) Grafico delle prime 100 osservazioni

Figura 6.14: Previsione vs valore reale, away team, CatBoost

6.1.6 Scelta del metodo di previsione puntuale

Riassumiamo quanto visto nei precedenti paragrafi.

Nelle Tabelle 6.3 (home team) e 6.4 (away team) è riportata la media dei valori di R^2 e RMSE ottenuti durante la cross validation di tutti i metodi utilizzati in seguito all'ottimizzazione degli iperparametri (dove effettuata).

Oltre ai punteggi delle loss function, sono presenti i valori dell'errore medio di previsione e del *mean absolute percentage error* (MAPE).

Per la previsione $\hat{\mathbf{y}} = (\hat{y}_1, \dots, \hat{y}_n)^T$ della variabile $\mathbf{y} = (y_1, \dots, y_n)^T$, l'errore medio è la quantità

$$\text{Errore medio} := \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)$$

e ci permette di riscontrare se ci sono dei bias nel modello: un errore medio significativamente maggiore di zero ci dice che l'algoritmo tende ad effettuare sovrastime della varia-

bile output. Viceversa, in caso di errore medio negativo e distante dallo zero il modello tende a fornire previsioni in difetto, ossia a sottostimare la variabile indipendente.

Il mean absolute percentage error è invece definito come segue.

Definizione 6.1.3 (MAPE)**.** Siano $\mathbf{y}$ e $\hat{\mathbf{y}}$ come nella Definizione 6.1.1. Chiamiamo *errore assoluto medio percentuale* la quantità

$$\text{MAPE} := \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{|y_i|} * 100.$$

Tutte le previsioni sono state effettuate sul dataset standardizzato, per consentire un confronto appropriato dei risultati.

Home Team

Si nota che l'algoritmo Decision Trees è il meno performante nella predizione dei risultati, come già osservato nell'analisi dello scatterplot. Aumentando la complessità dei metodi basati su alberi decisionali, migliorano sia il punteggio del coefficiente di determinazione sia quello relativo all'errore quadratico medio. In particolare, CatBoost risulta essere il metodo che ottiene, in media, i risultati migliori tra tutti i metodi valutati.

	R^2	RMSE	Errore medio	MAPE
`LinearRegression`	0.421	0.760	-4.92e-04	23.4%
`SVR`	0.416	0.763	9.77e-03	25.8%
`DecisionTreeRegressor`	0.370	0.793	-5.26e-04	24.3%
`RandomForestRegressor`	0.422	0.760	-3.36e-03	23.7%
`GradientBoostingRegressor`	0.423	0.759	-1.39e-03	23.5%
`LGBMRegressor`	0.424	0.758	-5.28e-04	23.4%
`CatBoostRegressor`	0.429	0.755	-1.58e-03	23.3%

Tabella 6.3: Risultati previsione puntuale home team, validation set

Tutti i metodi, ad eccezione di SVR, presentano un errore medio dell'ordine di grandezza non superiore a 10^{-3}. Questi risultati indicano l'assenza di un bias significativo

nei modelli, suggerendo che non vi sia una tendenza evidente né a sovrastimare né a sottostimare le previsioni. Piuttosto, gli errori di previsione in eccesso e quelli in difetto sembrano compensarsi reciprocamente.

Il confronto basato sul MAPE non è molto significativo: solo SVR registra un errore assoluto percentuale medio superiore al 25%, mentre per tutti gli altri metodi utilizzati il valore di MAPE si attesta tra il 23.3% e il 24.3%. In ogni caso, i valori sono piuttosto elevati, evidenziando così la complessità intrinseca del problema di previsione dei dati calcistici.

La scelta dunque ricade su CatBoost come modello di riferimento per le previsioni del field tilt della squadra di casa. Il nostro algoritmo sarà dunque:
`CatBoostRegressor(depth = 7, iterations = 900, learning_rate = 0.015)`.

Away Team

Anche in questo caso, CatBoost continua a distinguersi come il modello migliore, con il punteggio medio più elevato del coefficiente R^2 e il valore più basso di RMSE. Anche i comportamenti dell'errore medio e del MAPE sono molto simili a quanto visto per l'home team: l'errore medio non evidenzia bias particolari nelle previsioni e tutti i modelli registrano un errore percentuale medio tra il 27% e il 29%, ad eccezione di SVR.

	R^2	RMSE	Errore medio	MAPE
`LinearRegression`	0.423	0.760	3.49e-04	27.6%
`SVR`	0.416	0.764	-7.80e-03	29.5%
`DecisionTreeRegressor`	0.370	0.793	-4.92e-04	28.9%
`RandomForestRegressor`	0.421	0.761	3.11e-03	28.1%
`GradientBoostingRegressor`	0.425	0.758	-2.55e-04	27.7%
`LGBMRegressor`	0.424	0.759	-4.67e-04	27.6%
`CatBoostRegressor`	0.429	0.756	1.63e-05	27.4%

Tabella 6.4: Risultati previsione puntuale away team, validation set

Viene scelto come algoritmo di regressione per la predizione del field tilt della squadra di trasferta:
`CatBoostRegressor(depth = 7, iterations = 1000, learning_rate = 0.015)`.

6.2 Intervallo di confidenza

Una volta scelti i migliori algoritmi per la predizione puntuale, ci concentriamo ora sulla ricerca del miglior metodo per la predizione degli intervalli di confidenza. Nei paragrafi seguenti presenteremo i risultati dell'implementazione dei metodi illustrati nel Capitolo 4.

Dopo aver applicato un algoritmo, ne valutiamo la prestazione misurando due caratteristiche dell'intervallo ottenuto:

- copertura, cioè la percentuale di osservazioni i cui valori di output rientrano negli intervalli predetti;
- ampiezza, vale a dire la media delle differenze tra estremo superiore e estremo inferiore degli intervalli predetti.

Gli intervalli sono creati con l'obiettivo di raggiungere una copertura del 68%, che corrisponde all'ampiezza di una deviazione standard dalla media di una distribuzione normale.

La principale caratteristica a cui abbiamo prestato attenzione è l'ampiezza media degli intervalli. Una volta raggiunta la copertura del 68%, il miglior metodo è quello che garantisce una minore ampiezza media dell'intervallo. Ad esempio, se un metodo ottiene una copertura del 70% con un'ampiezza media di 28 punti percentuali di field tilt, mentre un secondo metodo raggiunge il 68% di copertura con un'ampiezza di 27.5, quest'ultimo viene scelto.

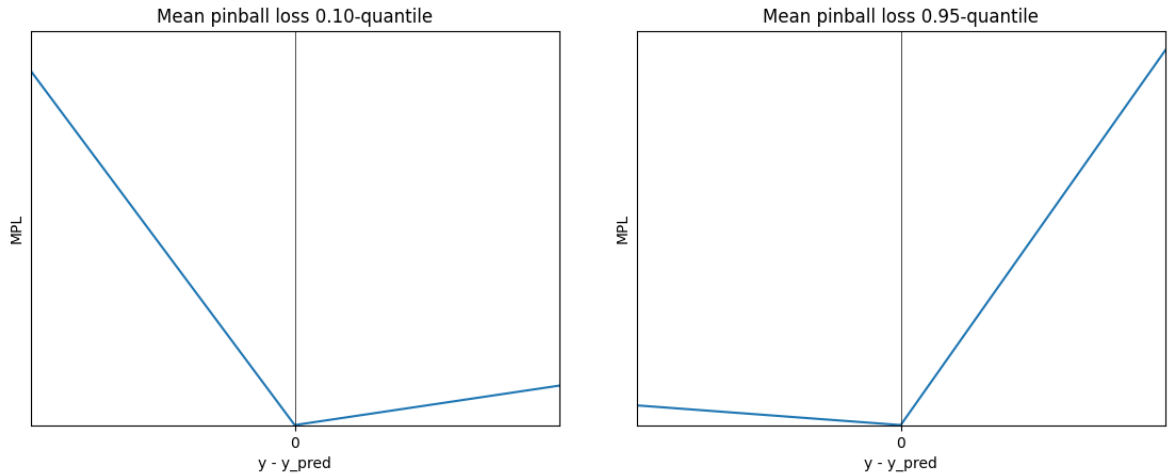
Ciascun algoritmo ha bisogno di un regressore in grado di essere addestrato sul training set e di effettuare previsioni sul validation set. Nel seguito, a meno che non sia specificato diversamente, si assume implicitamente che il metodo di base utilizzato sia CatBoost, con i parametri ottimizzati come descritto nei paragrafi precedenti.

6.2.1 Regressione quantile

Esattamente come nel caso della previsione puntuale, i modelli utilizzati allenano i dati minimizzando una loss function. Nel caso della regressione quantile tale funzione è chiamata *mean pinball loss* e si definisce come segue.

Definizione 6.2.1 (MPL)**.** Siano $\tau \in]0,1[$ e siano $\mathbf{y} = (y_1, \ldots, y_n)^T$ e $\hat{\mathbf{y}} = (\hat{y}_1, \ldots, \hat{y}_n)^T$, rispettivamente, la variabile obiettivo e la predizione del $\tau-$quantile. Si definisce *mean pinball loss* tra $\mathbf{y}$ e $\hat{\mathbf{y}}$ la quantità

$$\mathrm{MPL}_\tau := \frac{1}{n}\sum_{i=1}^{n} \tau \max(y_i - \hat{y}_i, 0) + (1-\tau)\max(\hat{y}_i - y_i, 0). \tag{6.1}$$



(a) 0.10-quantile (b) 0.95-quantile

Figura 6.15: Curva MPL

La parola *pinball* all'interno del nome è dovuta alla curva associata alla MPL (cfr. Figura 6.15), che ricorda traiettoria di una pallina di un flipper (in inglese pinball).

Si può dimostrare che una funzione che minimizza la MPL garantisce l'ottimalità dei quantili trovati. Per la prova di tale fatto si rimanda a [Koenker, Basset (1978)].

Per comprendere il comportamento della funzione (6.1) che definisce la mean pinball loss, vediamo un esempio. Supponiamo di addestrare un modello con l'obiettivo di predire

lo 0.95-quantile per una variabile y, con $n = 1$ osservazione. Consideriamo il caso in cui il valore reale della variabile target sia 30 e che il modello predica come quantile 50, ossia una sovrastima di 20 del valore esatto. La MPL associata è dunque

$$\mathrm{MPL}_{0.95}(30, 50) = 0.95 * (0) + 0.05 * (20) = 1.$$

Ora, con lo stesso valore reale di 30, supponiamo che il modello predica il quantile con valore 10, i.e. una sottostima del target sempre di 20. Il valore della loss function associato, tuttavia, è molto diverso:

$$\mathrm{MPL}_{0.95}(30, 10) = 0.95 * (20) + 0.05 * (0) = 19.$$

Dunque, nonostante la predizione del modello differisca in entrambi i casi di 20, la loss function penalizza le sottostime molto più delle sovrastime. Infatti, affinché lo 0.95-quantile sia predetto, la loss function penalizza ogni stima inferiore a valore reale di un fattore 0.95, mentre le stime maggiori del target sono penalizzate solo di un fattore 0.05. Di conseguenza, il modello è "obbligato" a favorire sovrastime rispetto a sottostime per predire lo 0.95-quantile. Lo stesso comportamento si verifica per la previsione di qualsiasi quantile sopra la mediana (0.5-quantile), mentre si ha il comportamento opposto quando si cercano quantili inferiori a 0.5.

Multiquantile

L'algoritmo `CatBoostRegressor` permette di calcolare simultaneamente diversi quantili semplicemente impostando come parametro della loss function `'Multiquantile:alpha='` e indicando la lista di quantili da ricercare. Nel nostro caso, i quantili di interesse sono il 16-percentile e l'84-percentile in modo da ottenere un intervallo che corrisponde al 68% di copertura.

Home Team

La Tabella 6.5 mostra i punteggi ottenuti dall'algoritmo: le prime due righe esprimono la MPL relativa ai quantili calcolati, la terza riga corrisponde alla copertura, mentre nell'ultima riga è riportata l'ampiezza dell'intervallo. Per evitare confusione, il valore dell'ampiezza è riportato senza il simbolo di percentuale, anche se rappresenta un'incertezza del field tilt, che è espresso come valore percentuale.

MPL 0.16	3.4168
MPL 0.84	3.2665
Coverage	56.9%
Width	22.1

Tabella 6.5: Risultati intervallo di previsione home team, multiquantile

Notiamo subito che la copertura ottenuta è ben lontana dal 68% cercato.

Vediamo ora i risultati relativi alla predizione dell'intervallo della variabile `away_field_tilt_game`.

Away Team

Dalla Tabella 6.6 si evince che neanche la previsione del field tilt della squadra in trasferta permette di ottenere una copertura sufficiente, dunque risultati sono distanti dall'obiettivo.

MPL 0.16	3.3322
MPL 0.84	3.4438
Coverage	56.2%
Width	21.5

Tabella 6.6: Risultati intervallo di previsione away team, multiquantile

I risultati ottenuti sono scadenti, proviamo perciò a calcolare singolarmente i quantili.

Monoquantile

Con `CatBoostRegressor` è possibile calcolare anche un singolo quantile, selezionando come parametro relativo alla loss function `'Quantile:alpha='` e il relativo quantile. Occorre dunque applicare due volte l'algoritmo. Focalizzandoci sui singoli valori dei percentili, possiamo eseguire una ricerca degli iperparametri per minimizzare la loss function. I parametri da ottimizzare sono stati scelti tra:

- $depth$: $[2, 3, 4, 5]$;
- $iterations$: $[400, 500, 600, 700, 800, 900]$;
- $learning\_rate$: $[0.01, 0.015, 0.02, 0.025, 0.03, 0.035, 0.04]$.

Vediamo i risultati ottenuti per le variabili di interesse.

Home Team

Per lo 0.16-quantile abbiamo come iperparametri:

$$depth = 3, \quad iterations = 500, \quad learning\_rate = 0.035.$$

Per lo 0.84-quantile invece abbiamo:

$$depth = 4, \quad iterations = 600, \quad learning\_rate = 0.02.$$

Con tali scelte, i risultati sono riassunti nella Tabella 6.7. Calcolando i quantili singolarmente miglioriamo sensibilmente la nostra copertura.

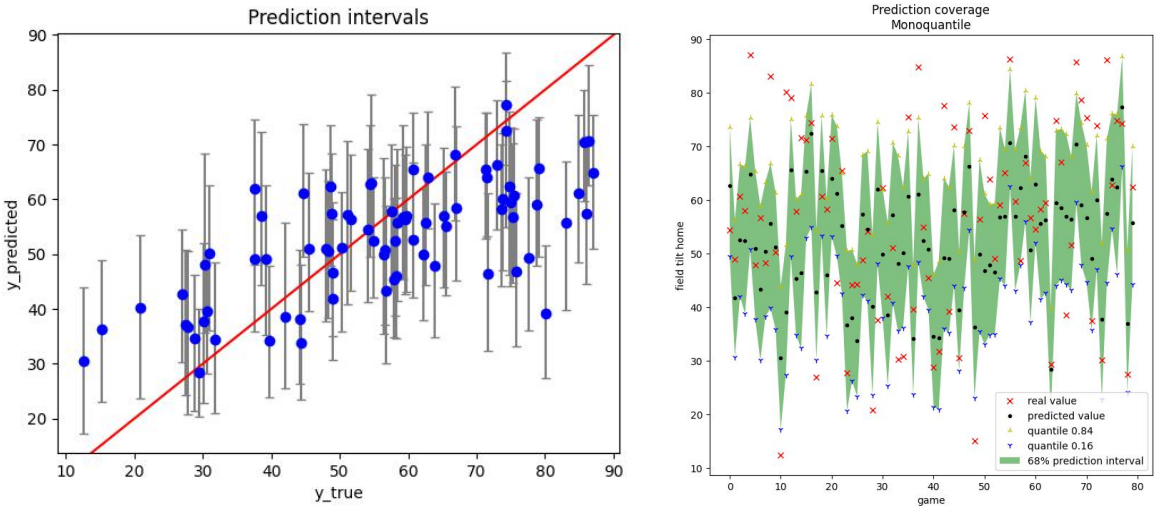
MPL 0.16	3.3327
MPL 0.84	3.1497
Coverage	65.4%
Width	25.8

Tabella 6.7: Risultati intervallo di previsione home team, monoquantile

La Figura 6.16 mostra gli intervalli calcolati sulle prime 80 osservazioni del training set.

Nel grafico a sinistra il valore dell'ascissa corrisponde al valore osservato della variabile `home_field_tilt_game`, mentre sull'ordinata sono presenti il valore predetto (pallino blu) e l'intervallo di previsione ottenuto con la regressione quantile (in grigio), avente come estremi i quantili calcolati. Se per un'osservazione il segmento grigio interseca la diagonale rossa, allora l'intervallo di previsione copre il valore reale, raggiungendo l'obiettivo della previsione per quella specifica osservazione.

Nell'immagine a destra, invece, l'intervallo è tanto migliore quante più croci rosse entrano nell'area verde.



(a) Intervalli di previsione delle prime 80 osservazioni (b) Copertura prime 80 osservazioni

Figura 6.16: Intervalli di previsione home team, monoquantile

Away Team

La combinazione ottimale di iperparametri per lo 0.16-quantile è la seguente:

$$depth = 4, \quad iterations = 800, \quad learning\_rate = 0.015.$$

Mentre per lo 0.84-quantile si ha:

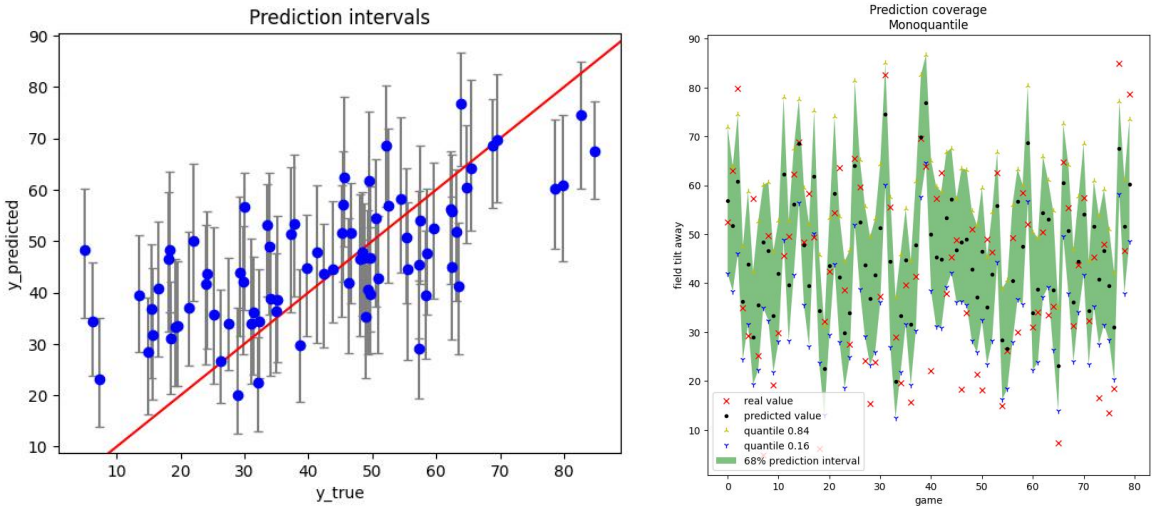
$$depth = 3, \quad iterations = 800, \quad learning\_rate = 0.02.$$

Presentiamo i relativi risultati nella Tabella 6.8.

MPL 0.16	3.1732
MPL 0.84	3.3715
Coverage	65.9%
Width	25.8

Tabella 6.8: Risultati intervallo di previsione away team, monoquantile

Anche qui i risultati sono significativamente migliori rispetto al caso multiquantile. Vediamo gli intervalli nella Figura 6.17.



(a) Intervalli di previsione delle prime 80 osservazioni (b) Copertura prime 80 osservazioni

Figura 6.17: Intervalli di previsione away team, monoquantile

6.2.2 Conformal prediction

L'applicazione della regressione quantile sul dataset di training non ci ha permesso di raggiungere l'obiettivo di copertura del 68%. Ciò significa che, tanto nel caso multi-quantile quanto in quello monoquantile, gli algoritmi utilizzati non forniscono risultati sufficienti; è dunque necessario cambiare approccio. Mostriamo nei prossimi paragrafi i risultati ottenuti applicando gli algoritmi di conformal prediction spiegati nel Capitolo 4.2. I codici delle funzioni utilizzate sono riportati nell'Appendice A.

Jackknife+

I primi metodi di conformal prediction utilizzati sono Jackknife+ e la sua variante CV+.

Tramite la funzione `jaccknife` il training set viene suddiviso casualmente *ntrial* volte in training set vero e proprio, di lunghezza N, e calibration set. I parametri *ntrial* e N vengono forniti in input alla funzione.

Per ciascuna suddivisione, vengono calcolati gli intervalli di confidenza tramite la funzione `compute_PIs`, cercando di raggiungere la copertura desiderata $1 - \alpha$ (nel nostro caso, $\alpha = 0.32$). L'algoritmo `compute_PIs` sfrutta un regressore dato in input per

calcolare gli estremi dell'intervallo di conformal prediction ottenuti sia attraverso la leave-one-out cross validation (Jackknife+), sia attraverso una 10-fold cross validation (CV+).

Le Tabelle 6.9 e 6.10 presentano i valori medi ± deviazione standard della copertura e dell'ampiezza media degli intervalli di confidenza calcolati, rispettivamente, sulle variabili `home_field_tilt_game` e `away_field_tilt_game`. Sono state effettuate $ntrial = 5$ iterazioni, con $N = 200$ osservazioni usate come training set. I valori di copertura sono intesi come punteggi percentuali.

Home Team

Nessuna delle due varianti fornisce i risultati desiderati, tuttavia CV+ si avvicina maggiormente al livello di confidenza cercato, con un aumento piccolo in termini di ampiezza. Lo scegliamo dunque come metodo di Jackknife+.

	Jackknife+	CV+
Coverage	67.3 ± 0.9	67.7 ± 1.2
Width	29.1 ± 0.5	29.3 ± 0.8

Tabella 6.9: Risultati intervallo di previsione home team, Jackknife+

Applicando l'algoritmo sul validation set otteniamo come valori di copertura e ampiezza:

$$\textbf{Coverage} = 69.8\%;$$
$$\textbf{Width} = 29.1.$$

Away Team

Anche nel caso dell'away team non riusciamo a ottenere la copertura cercata.

	Jackknife+	CV+
Coverage	67.3 ± 1.3	67.9 ± 1.2
Width	28.8 ± 0.7	29.2 ± 0.6

Tabella 6.10: Risultati intervallo di previsione away team, Jackknife+

Per ragioni analoghe al caso dell'home team, preferiamo al metodo Jackknife+ la variante CV+, che ci permette di ottenere sul validation set i seguenti punteggi:

$$\textbf{Coverage} = 71.6\%;$$
$$\textbf{Width} = 31.6.$$

Sia per la squadra di casa che per quella di trasferta, l'applicazione di Jackknife+ sul validation set ci permette di raggiungere il 68% di copertura. Tuttavia, l'ampiezza degli intervalli creata è eccessiva. Proviamo quindi un altro metodo di conformal prediction: Jackknife+ after Bootstrap.

Jackknife+ after Bootstrap

Per applicare il metodo si è sfruttata la funzione `jackknife_aB`, la quale calcola tot_trial volte gli intervalli di previsione per differenti dimensioni del Bootstrap, seguendo le ipotesi del Teorema 4.2.3. Tali dimensioni dipendono da un parametro $\tilde{B}$ e dalla quantità $m = p * N$, con N lunghezza del training set e p valore compreso tra 0 e 1. Per i dettagli dell'implementazione si vedano l'Appendice A e [Kim et al. (2020)].

Per calcolare gli intervalli sono state utilizzate come funzioni di aggregazione la media, la mediana e la media troncata al 25%. Confrontiamo i risultati ottenuti in seguito a $tot\_trial = 5$ iterazioni per ciascuna delle funzioni di aggregazione, usando i seguenti parametri:

- $\tilde{B} = 20$;
- $p = [0.2, 0.4, 0.6, 0.8, 1]$.

Home Team

Le Tabelle 6.11, 6.12 e 6.13 mostrano media ± deviazione standard sulle 5 iterazioni dei risultati ottenuti per le tre diverse funzioni di aggregazione, al variare dei valori di p.

I valori sono simili a quelli ottenuti con Jackknife+, ma richiedono un'ampiezza degli intervalli minore. Poiché la copertura del 68% non viene mai raggiunta, scegliamo come modello di riferimento per la predizione di intervalli di confidenza quello che più si avvicina al livello cercato. I valori più alti vengono ottenuti usando la mediana o la

media troncata, quando $p = 0.6$. Scegliamo di utilizzare come funzione di aggregazione la mediana.

p	**Coverage**	**Width**
0.2	67.6 ± 0.1	27.2 ± 0.0
0.4	67.6 ± 0.1	26.9 ± 0.0
0.6	67.6 ± 0.1	26.9 ± 0.0
0.8	67.6 ± 0.1	26.8 ± 0.0
1	67.5 ± 0.1	26.8 ± 0.0

Tabella 6.11: Risultati intervallo di previsione home team, Jackknife+ aB, media

p	**Coverage**	**Width**
0.2	67.4 ± 0.1	27.2 ± 0.0
0.4	67.5 ± 0.1	27.0 ± 0.0
0.6	67.7 ± 0.1	26.9 ± 0.0
0.8	67.4 ± 0.1	26.8 ± 0.0
1	67.5 ± 0.1	26.8 ± 0.0

Tabella 6.12: Risultati intervallo di previsione home team, Jackknife+ aB, mediana

p	**Coverage**	**Width**
0.2	67.4 ± 0.1	27.2 ± 0.0
0.4	67.6 ± 0.1	26.9 ± 0.0
0.6	67.7 ± 0.1	26.9 ± 0.0
0.8	67.5 ± 0.1	26.8 ± 0.0
1	67.5 ± 0.1	26.8 ± 0.0

Tabella 6.13: Risultati intervallo di previsione home team, Jackknife+ aB, media troncata

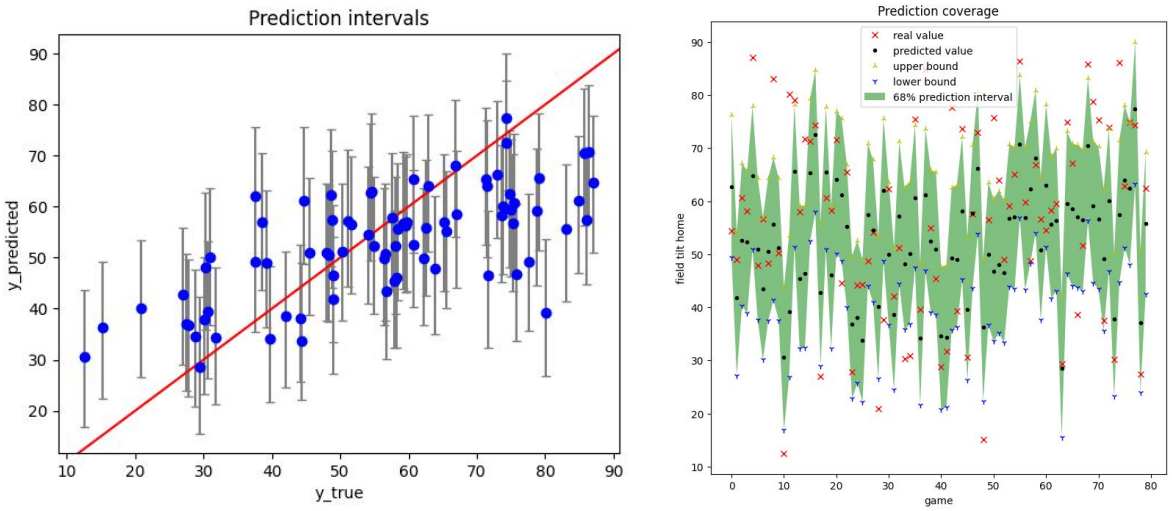
Addestrando l'algoritmo sul training set e successivamente predicendo i valori del validation set, otteniamo come copertura e ampiezza:

$$\textbf{Coverage} = 67.4\%;$$

$$\textbf{Width} = 26.9.$$

L'algoritmo MAPIE, implementato tramite la funzione `MapieRegressor` del pacchetto `regression`, appartenente alla libreria `mapie`, permette di calcolare sia la predizione puntuale, che gli intervalli di confidenza, prendendo come argomento l'algoritmo di regressione. Selezionando `subsample.Subsample` come parametro `cv`, viene utilizzato come metodo Jackknife+ after Bootstrap per la predizione di intervalli.

Nella Figura 6.18 vediamo gli intervalli associati alle prime 80 osservazioni del validation set.



(a) Intervalli di previsione delle prime 80 osservazioni

(b) Copertura prime 80 osservazioni

Figura 6.18: Intervalli di previsione home team, Jackknife+ after Bootstrap

Away Team

I risultati ottenuti sull'away team utilizzando Jackknife+ after Bootstrap sono riassunti nelle Tabelle 6.14, 6.15 e 6.16.

p	**Coverage**	**Width**
0.2	67.4 ± 0.1	26.9 ± 0.0
0.4	67.1 ± 0.0	26.7 ± 0.0
0.6	67.0 ± 0.0	26.6 ± 0.0
0.8	67.2 ± 0.0	26.5 ± 0.0
1	67.1 ± 0.1	26.5 ± 0.0

Tabella 6.14: Risultati intervallo di previsione away team, Jackknife+ aB, media

p	**Coverage**	**Width**
0.2	67.4 ± 0.1	26.9 ± 0.0
0.4	67.1 ± 0.1	26.7 ± 0.0
0.6	67.1 ± 0.1	26.6 ± 0.0
0.8	67.1 ± 0.1	26.5 ± 0.0
1	67.0 ± 0.1	26.5 ± 0.0

Tabella 6.15: Risultati intervallo di previsione away team, Jackknife+ aB, mediana

p	**Coverage**	**Width**
0.2	67.4 ± 0.1	26.9 ± 0.0
0.4	67.1 ± 0.1	26.7 ± 0.0
0.6	67.1 ± 0.1	26.6 ± 0.0
0.8	67.1 ± 0.1	26.5 ± 0.0
1	67.1 ± 0.0	26.5 ± 0.0

Tabella 6.16: Risultati intervallo di previsione away team, Jackknife+ aB, media troncata

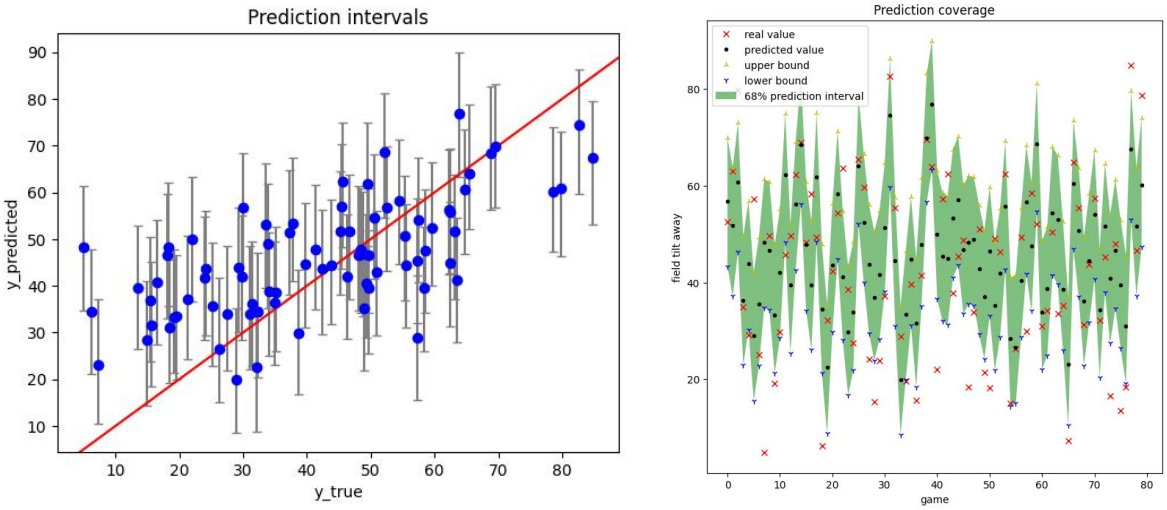
Nemmeno in questo caso i metodi raggiungono il 68% di copertura. I valori che più si avvicinano al livello di confidenza cercato si ottengono con $p = 0.2$ e funzione di aggregazione la media o la mediana. Per l'away team scegliamo di utilizzare la media,

che ci fa raggiungere come risultati sul validation set:

$$\textbf{Coverage} = 67.4\%;$$

$$\textbf{Width} = 27.0.$$

Vediamo con la Figura 6.19 gli intervalli associati alle prime 80 osservazioni del validation set.



(a) Intervalli di previsione delle prime 80 osservazioni (b) Copertura prime 80 osservazioni

Figura 6.19: Intervalli di previsione away team, Jackknife+ after Bootstrap

Sia per la squadra di casa che per quella di trasferta, Jackknife+ after Bootstrap risulta essere un compromesso tra la regressione quantile, che ottiene coperture troppo scarse, e Jackknife+, che, invece, è troppo conservativo nell'ampiezza degli intervalli. Scegliamo dunque Jackknife+ after Bootstrap come metodo per la previsione di intervalli di confidenza delle variabili `home_field_tilt_game` e `away_field_tilt_game`.

6.3 Dataset ristretto

Le analisi svolte fino ad ora hanno richiesto un considerevole sforzo computazionale. In particolare, la predizione degli intervalli di confidenza ha richiesto il calcolo di un grande numero di modelli di addestramento. Questo è dovuto principalmente al fatto che per

effettuare le nostre previsioni abbiamo utilizzato un elevato numero di variabili, ben 25, in relazione al numero totale di osservazioni .

Abbiamo quindi deciso di condurre una nuova analisi considerando un numero limitato di variabili che abbiamo ritenuto più significative nell'influenzare una statistica di stile come il field tilt. Tale selezione è stata operata utilizzando una combinazione di valutazioni quantitative, basate su risultati empirici, e valutazioni qualitative, fondate su conoscenze pregresse nel campo della Football Analytics. La rimozione delle altre variabili potrebbe eliminare componenti di rumore che potrebbero compromettere i risultati del modello.

Le variabili scelte sono le seguenti:

- `field_tilt_season`;
- `opp_field_tilt_season`;
- `xP_per_game`;
- `opp_xP_per_game`;
- `npxG_season`;
- `opp_npxGA_season`;
- `poss_won_att_3rd_season`;
- `opp_lost_ball_def_3rd_season`.

Le metriche sul field tilt stagionale e gli xP sono considerate inevitabili, in quanto sono anche le più rilevanti dal punto di vista di valori SHAP, come mostrato nel Capitolo 5. Si sono scelti i npxG stagionali, così come quelli subiti dall'avversario, come metriche di performance. Inoltre, i recuperi nel terzo di campo offensivo e i possessi persi nel terzo difensivo della squadra avversaria sono stati considerati come statistiche di stile. L'idea di base è che segnare molti gol e recuperare palloni nel terzo offensivo possano essere fortemente correlati a un maggior possesso palla in quella zona del campo, mentre concedere gol e perdere la palla nel proprio terzo difensivo possono portare a concedere

un maggior possesso palla agli avversari in quella zona.

Vediamo i risultati associati al nuovo dataset.

6.3.1 Previsione puntuale

Per predire i valori puntuali si è utilizzato direttamente CatBoost, in quanto tale algoritmo ha fornito le migliori prestazioni nel predire le stesse osservazioni, utilizzando più variabili. Gli iperparametri sono stati ottimizzati tramite la cross validation, scegliendo fra:

- $depth$: $[5, 6, 7, 8]$;
- $iterations$: $[800, 850, 900, 950, 1000]$;
- $learning\_rate$: $[0.005, 0.01, 0.015, 0.02]$.

Home Team $depth = 6$, $iterations = 850$, $learning\_rate = 0.01$.
Gli score associati sono i seguenti:

$$R^2 = (0.415 \pm 0.014);$$
$$\text{RMSE} = (13.397 \pm 0.141).$$

Calcolando MAPE ed errore medio, si hanno:

$$\text{MAPE} = 23.5\%;$$
$$\text{Errore medio} = \text{-8.83e-05}.$$

Away Team $depth = 5$, $iterations = 1000$, $learning\_rate = 0.01$.
Con tali parametri i punteggi ottenuti sono:

$$R^2 = (0.412 \pm 0.009);$$
$$\text{RMSE} = (13.414 \pm 0.101).$$

Per quanto riguarda MAPE ed errore medio abbiamo invece:

$$\text{MAPE} = 27.7\%;$$
$$\text{Errore medio} = \text{-2.54e-06}.$$

Notiamo immediatamente che i risultati della predizione puntuale sono peggiori. Non si evidenziano nemmeno in questo caso bias predittivi, in quanto l'errore medio è estremamente piccolo sia per l'home team che per l'away team. È possibile che rimuovere variabili abbia portato a una perdita di informazioni che rende più difficile la previsione puntuale.

Vediamo se lo stesso comportamento si verifica per quanto riguarda gli intervalli di confidenza.

6.3.2 Intervallo di confidenza

Per la stima degli intervalli si è proceduto direttamente ad applicare il metodo Jackknife+ after Bootstrap, utilizzando gli stessi parametri scelti per il dataset originario. Pertanto, sono stati selezionati $p = 0.6$ e come funzione di aggregazione la mediana per l'home team, mentre la media come funzione di aggregazione e 0.2 come valore di p per l'away team.

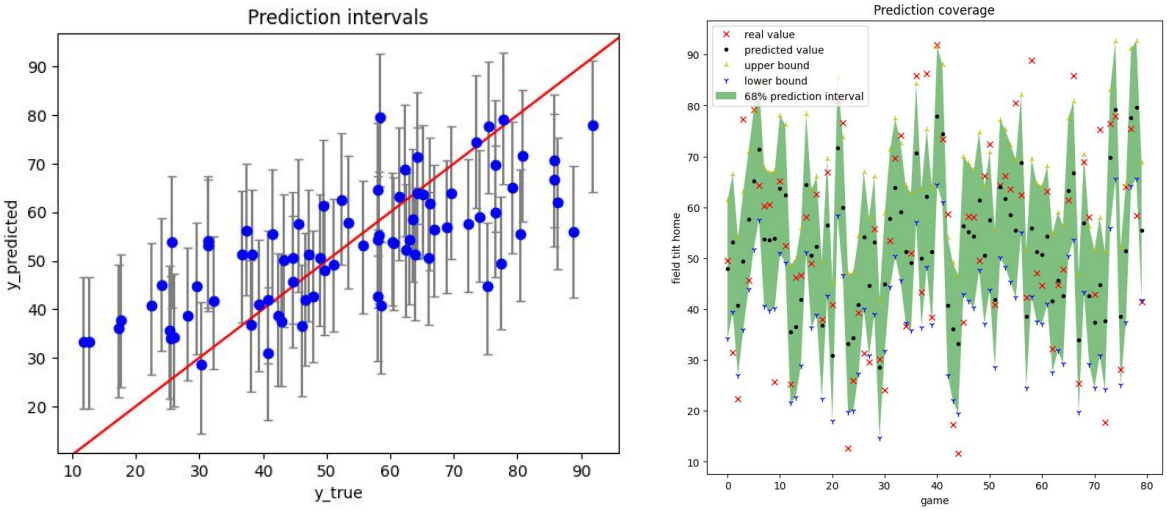
Home Team

Applicando l'algoritmo al validation set otteniamo come valori di copertura e ampiezza:

$$\textbf{Coverage} = 68.0\%;$$
$$\textbf{Width} = 27.2.$$

Raggiungiamo finalmente il punteggio di copertura cercato, al costo di aumentare l'ampiezza media degli intervalli.

Nella Figura 6.20 vediamo gli intervalli calcolati da MAPIE per le prime 80 osservazioni del validation set.



(a) Intervalli di previsione delle prime 80 osservazioni (b) Copertura prime 80 osservazioni

Figura 6.20: Intervalli di previsione home team, Jackknife+ after Bootstrap

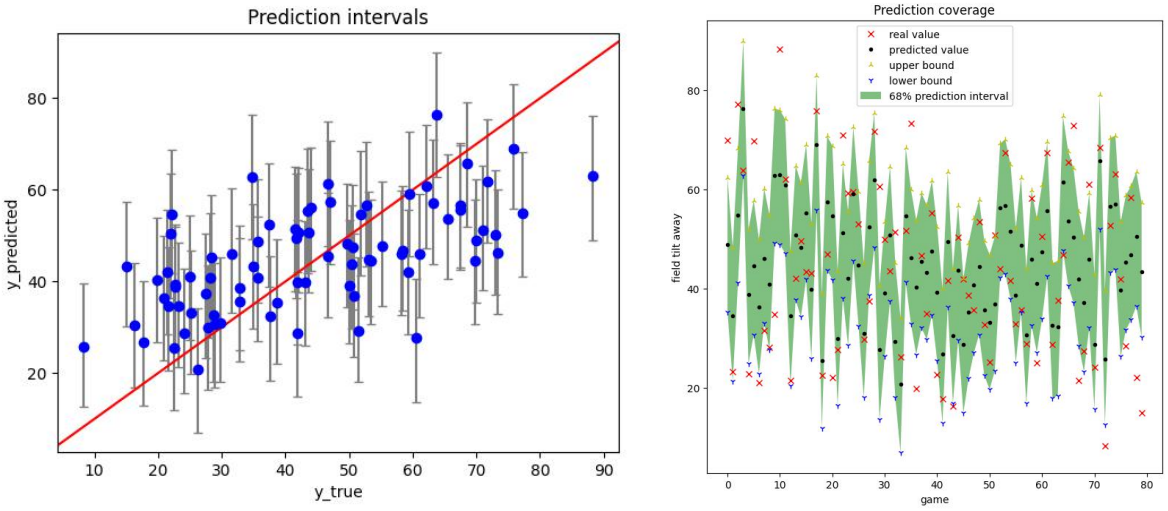
Away Team

I risultati relativi a copertura e ampiezza degli intervalli di previsioni dei valori del validation set sono:

$$\textbf{Coverage} = 66.9\%;$$
$$\textbf{Width} = 27.0.$$

In questo caso, il metodo applicato al dataset ristretto non riesce ad ottenere la copertura cercata, nonostante l'ampiezza media degli intervalli sia maggiore di quella ottenuta con il dataset originario.

Concludiamo la presentazione dei risultati di addestramento dei modelli sul dataset ristretto con la Figura 6.21, che mostra l'applicazione di MAPIE a training e validation set.



(a) Intervalli di previsione delle prime 80 osservazioni (b) Copertura prime 80 osservazioni

Figura 6.21: Intervalli di previsione away team, Jackknife+ after Bootstrap

Capitolo 7

Risultati testing

Nel Capitolo 6 abbiamo confrontato i risultati di training. Una volta ottenuti i modelli migliori, li abbiamo allenati sull'intero dataset di training e poi applicati sul dataset di testing (ossia le partite di Premier League della stagione 2022-2023). Questo procedimento è stato applicato sia sul dataset originario, che su quello ristretto. Presentiamo nel seguito i risultati relativi alla predizione puntuale e all'intervallo di confidenza.

7.1 Previsione puntuale

Home Team

Gli score ottenuti applicando CatBoost al dataset originario per predire la variabile `home_field_tilt_game` sono i seguenti:

$$R^2 = 0.452;$$
$$\text{RMSE} = 14.380;$$
$$\text{MAPE} = 31.1\%;$$
$$\text{Errore medio} = 4.32\text{e-}02.$$

I risultati sul dataset ristretto sono invece:

$$R^2 = 0.436;$$
$$\text{RMSE} = 14.210;$$
$$\text{MAPE} = 29.5\%;$$
$$\text{Errore medio} = 2.36\text{e-}01.$$

Away Team

I punteggi sul dataset originario relativi alla variabile `away_field_tilt_game` sono:

$$R^2 = 0.461;$$
$$\text{RMSE} = 14.284;$$
$$\text{MAPE} = 32.2\%;$$
$$\text{Errore medio} = \text{-}7.38\text{e-}01.$$

Con la selezione ristretta di variabili si ha:

$$R^2 = 0.430;$$
$$\text{RMSE} = 14.266;$$
$$\text{MAPE} = 31.8\%;$$
$$\text{Errore medio} = \text{-}9.11\text{e-}02.$$

Si osserva che il coefficiente di determinazione raggiunge valori più alti sfruttando tutte le variabili a disposizione, sia per l'home team, che per l'away team. Il minor numero di predittori invece potrebbe essere la ragione per cui l'errore quadratico medio sulle previsioni del field tilt della squadra di casa sia inferiore per il dataset ristretto rispetto al dataset originario. D'altra parte, l'errore relativo misurato tramite MAPE risulta leggermente più basso per le previsioni ottenute con il dataset ristretto rispetto a quelle effettuate con il dataset completo.

È importante notare che, a differenza di quanto visto durante il training, l'errore medio sul test set è stato calcolato su dati non standardizzati. Pertanto, è normale

osservare un aumento dell'ordine di grandezza nell'errore, che solo nel caso delle previsioni per la squadra ospite con il dataset completo supera 10^{-1}. Tuttavia, tenendo presente che il field tilt assume valori compresi tra 0 e 100, gli errori misurati rimangono abbastanza piccoli da non suggerire la presenza di un bias di sovrastima o sottostima del modello.

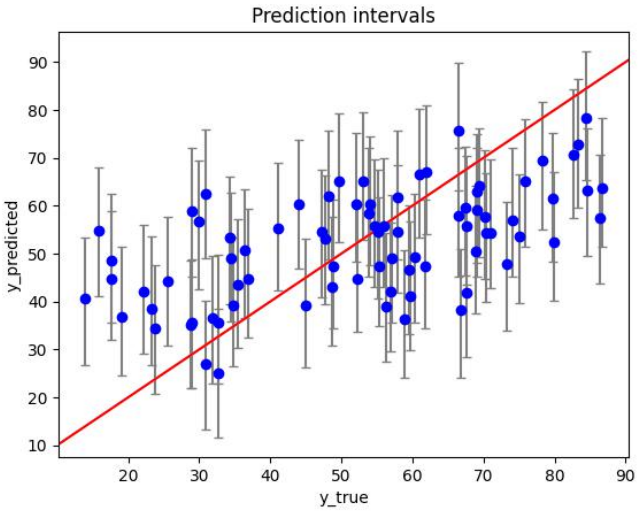
7.2 Intervallo di confidenza

Home Team

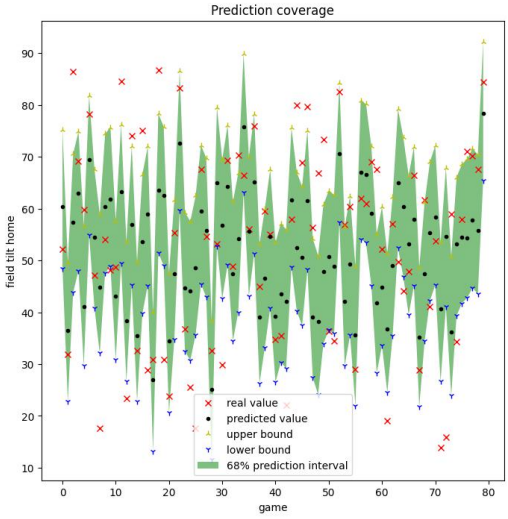
Jackknife+ after Bootstrap applicato al dataset originale ci permette di raggiungere i seguenti valori di copertura e ampiezza:

$$\textbf{Coverage} = 63.3\%;$$
$$\textbf{Width} = 26.7.$$

La Figura 7.1 mostra i primi 80 intervalli predetti in seguito all'applicazione dell'algoritmo MAPIE.



(a) Intervalli di previsione delle prime 80 osservazioni



(b) Copertura prime 80 osservazioni

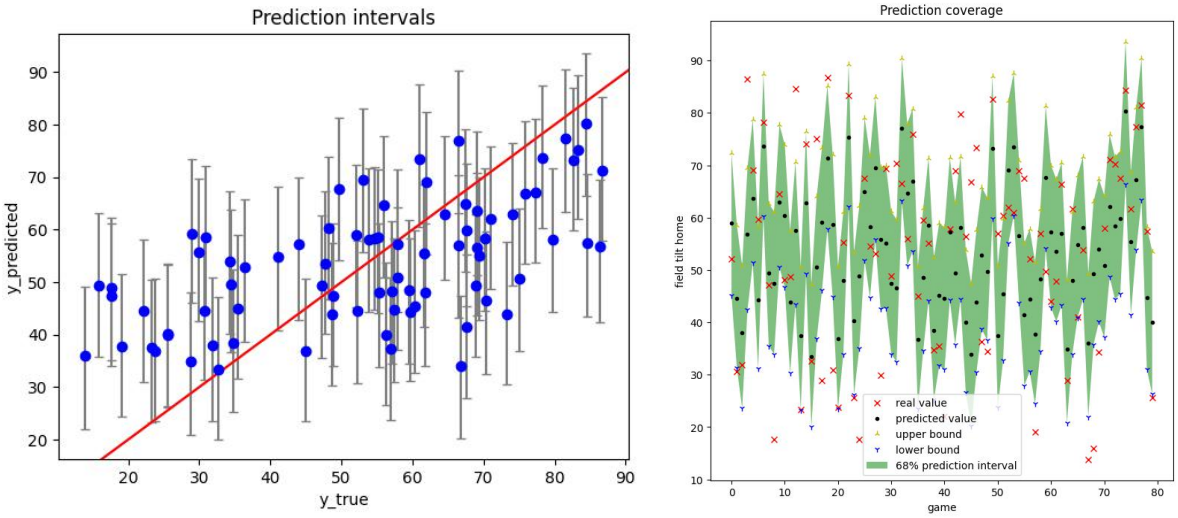
Figura 7.1: Intervalli di previsione home team, test set

Usando solo la selezione ristretta di variabili abbiamo:

$$\textbf{Coverage} = 63.6\%;$$

$$\textbf{Width} = 27.2.$$

Gli intervalli ottenuti tramite MAPIE sono riportati nella Figura 7.2.



(a) Intervalli di previsione delle prime 80 osservazioni (b) Copertura prime 80 osservazioni

Figura 7.2: Intervalli di previsione home team, dataset ristretto, test set

.

Away Team

Spostandoci sull'away team e sfruttando tutte le variabili, copertura e ampiezza sono:

$$\textbf{Coverage} = 62.6\%;$$

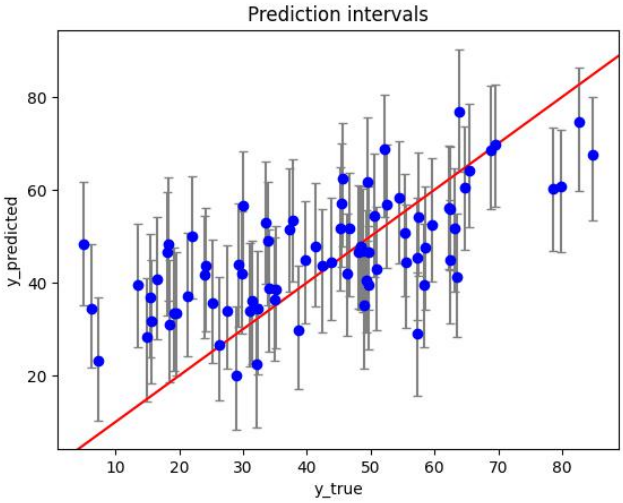
$$\textbf{Width} = 26.5.$$

Procedendo alla stessa analisi, ma con meno variabili, si ha:

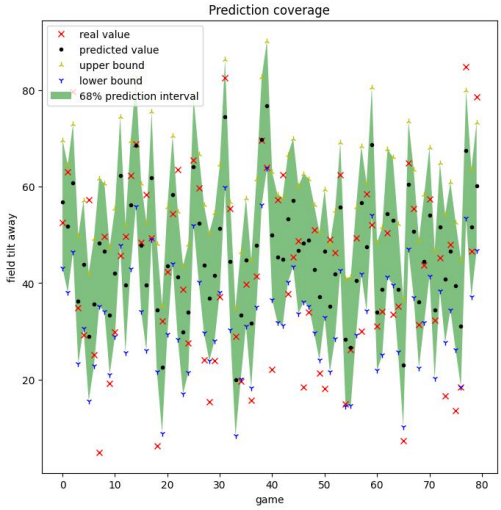
$$\textbf{Coverage} = 64.1\%;$$

$$\textbf{Width} = 27.1.$$

Concludiamo mostrando i primi 80 intervalli predetti da MAPIE sfruttando il dataset completo e quello ristretto con le Figure 7.3 e 7.4.

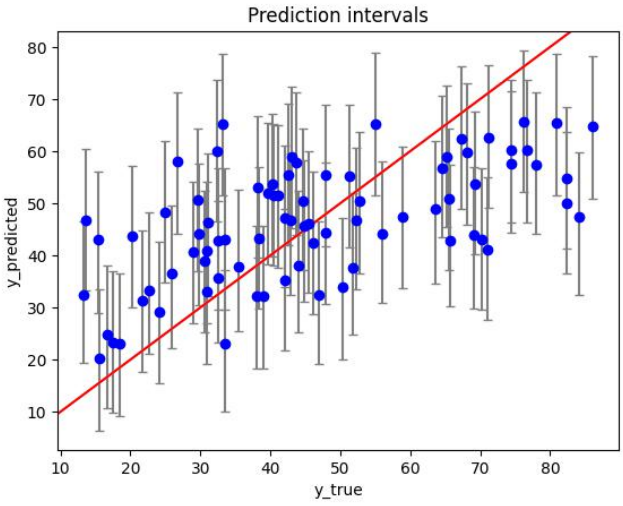


(a) Intervalli di previsione delle prime 80 osservazioni

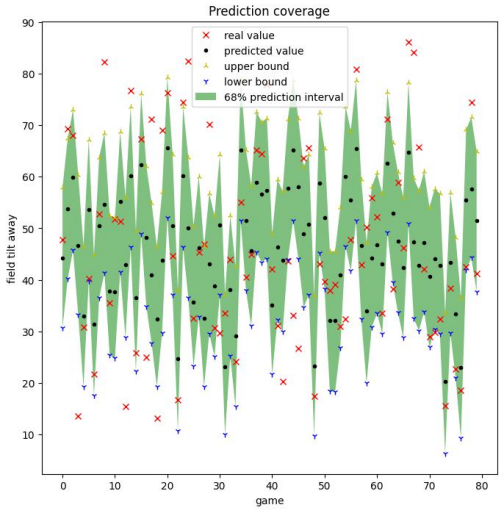


(b) Copertura prime 80 osservazioni

Figura 7.3: Intervalli di previsione away team, test set



(a) Intervalli di previsione delle prime 80 osservazioni



(b) Copertura prime 80 osservazioni

Figura 7.4: Intervalli di previsione away team, dataset ristretto, test set

Purtroppo, né utilizzando tutte le variabili predittrici, né sfruttandone solo un numero ridotto, i nostri miglior algoritmi sono in grado di garantire il livello di confidenza desiderato per nessuna delle due variabili che vogliamo predire. Tuttavia, è interessante notare che utilizzare meno variabili predittrici permette di raggiungere coperture più vicine all'obiettivo del 68%. Questo miglioramento è stato raggiunto a discapito di un leggero aumento dell'ampiezza media degli intervalli.

Questi risultati indicano che c'è un trade-off tra la precisione nella previsione puntuale e la capacità di ottenere intervalli di confidenza con la copertura desiderata. È necessario valutare attentamente tale compromesso e considerare il contesto specifico prima di decidere quale approccio utilizzare nella scelta del numero di covariate.

Conclusioni

Il progetto di tesi aveva l'obiettivo di trovare i migliori algoritmi di regressione e di costruzione di intervalli di confidenza al fine di predire il field tilt di due squadre che si affrontano in un incontro, a partire dalla media stagionale di statistiche di gara e metriche avanzate.

I risultati di ciascuno degli algoritmi di regressione sono stati valutati secondo i punteggi del coefficiente R^2 e RMSE in seguito alla cross validation. Dove possibile, è stata effettuata l'ottimizzazione degli iperparametri. CatBoost si è rivelato l'algoritmo migliore ed è stato dunque scelto come modello base per la costruzione degli intervalli di confidenza.

Gli intervalli sono stati costruiti seguendo due strategie: regressione quantile e conformal prediction, a sua volta implementata utilizzando i metodi Jackknife+ e Jackknife+ after Bootstrap. Gli algoritmi sono stati valutati considerando la copertura raggiunta e l'ampiezza media degli intervalli ottenuti. Jackknife+ after Bootstrap è risultato un compromesso tra la regressione quantile, che costruisce intervalli con una copertura lontana dal livello di confidenza cercato del 68%, e Jackknife+, che genera intervalli di confidenza eccessivamente ampi. Per queste ragioni, Jackknife+ after Bootstrap è stato selezionato come algoritmo migliore per la previsione degli intervalli.

Gli algoritmi sono stati successivamente testati sulle osservazioni nuove utilizzando prima tutte le variabili a disposizione, poi con una ristretta selezione di esse. Mentre per la predizione puntuale i risultati migliori vengono raggiunti sfruttando tutte le metriche a disposizione, è utilizzando solo alcune delle statistiche che gli intervalli costruiti hanno ottenuto migliori coperture. Tuttavia, non è stato raggiunto un livello di confidenza ritenuto accettabile. Le ragioni di tale insuccesso possono essere diverse. Innanzitut-

to, la previsione delle statistiche sportive è un ambito poco esplorato, sia dal punto di vista teorico che empirico. Pertanto, non è affatto scontato trovare immediatamente le strategie migliori. Inoltre, la complessità stessa del fenomeno suggerisce che sia estremamente difficile ottenere risultati soddisfacenti, sia a livello di previsioni puntuali che di intervalli di confidenza. Infine, va considerata la quantità limitata di dati disponibili: durante la fase di addestramento, sono state utilizzate solo circa 10^4 osservazioni, un numero relativamente esiguo per un problema di regressione di questa complessità. Se avessimo a disposizione più stagioni e campionati, quindi un maggior numero di partite e, di conseguenza, di osservazioni, l'algoritmo potrebbe offrire prestazioni più soddisfacenti.

I risultati ottenuti possono comunque essere migliorati percorrendo varie strade.

La prima riguarda la selezione delle variabili predittive. Seguendo la strategia proposta in [Berrar et al. (2019)] (cfr. Paragrafo 1.2), si potrebbe considerare l'aggiunta di variabili che considerino la conoscenza dell'ambito calcistico. Un ulteriore contributo significativo potrebbe derivare dalla creazione di cluster basati sullo stile di gioco delle squadre. L'introduzione di una variabile che rappresenti l'appartenenza delle due squadre a un determinato cluster potrebbe migliorare le capacità predittive dell'algoritmo riguardo all'atteggiamento tattico.

Miglioramenti potrebbero essere raggiunti anche aumentando la complessità dei metodi di costruzione degli intervalli di confidenza, ad esempio implementando algoritmi di CQR.

Recenti pubblicazioni hanno affrontato la costruzione di intervalli utilizzando algoritmi basati su alberi decisionali.

In [Sprangers et al. (2021)] è presentato *Probabilistic Gradient Boosting Machines* (PGBM), un metodo basato su Gradient Boosting che modella le distribuzioni di probabilità mediante ensemble di alberi decisionali. PGBM considera valori delle foglie come variabili aleatorie e approssima la media e la varianza di ogni campione del dataset attraverso un approccio stocastico. La media e la varianza predette vengono poi utilizzate come parametri in una distribuzione specificata per generare una previsione probabilistica. Questo permette di ottenere stime probabilistiche utilizzando un singolo modello, riducendo così i costi computazionali. Il modello di regressione stima la probabilità

condizionata ai valori delle covariate, consentendo di calcolare contemporaneamente la previsione puntuale (come valore atteso condizionato) e l'incertezza associata, che può essere interpretata come un intervallo di confidenza.

Un altro approccio innovativo è esposto in [Brophy, Lowd (2022)], dove viene presentato *Instance-Based Uncertainty estimation for Gradient-boosted regression trees*, o più semplicemente IBUG, un metodo per estendere qualsiasi predittore puntuale basato su Gradient Boosting e alberi decisionali (ad esempio LightGBM o CatBoost) al fine di produrre previsioni probabilistiche. IBUG calcola una distribuzione non parametrica intorno a una previsione utilizzando le osservazioni di training più "affini", ossia quelle che più spesso si trovano nella stessa foglia nell'addestramento degli alberi decisionali.

Infine, vale la pena considerare l'adozione di un approccio basato sulle serie temporali per la regressione. L'analisi delle serie temporali permetterebbe di identificare modelli di tendenza e fluttuazioni che si verificano durante la stagione. Ciò consentirebbe di rilevare se una squadra sta migliorando nel corso del tempo, se sta affrontando difficoltà in un particolare periodo o se ha avuto un picco di forma in una determinata fase. Queste informazioni possono risultare cruciali per prevedere il rendimento di una squadra in una specifica partita.

Inoltre, l'approccio delle serie temporali ci permette di individuare l'intervallo di partite più significativo da considerare per la previsione. Le variabili utilizzate per la previsione delle statistiche di gioco tengono conto delle medie stagionali per ciascuna voce. Concentrarci su un numero ristretto di partite precedenti a quella sulla quale siamo interessati a effettuare le previsioni potrebbe permetterci di ottenere una previsione più accurata delle statistiche.

Un'altra considerazione importante è che, mediante un approccio basato sulle serie temporali, saremmo in grado di catturare anche l'effetto dell'evoluzione delle condizioni esterne nel tempo. Ad esempio, potremmo osservare come le prestazioni di una squadra siano influenzate dal cambio di allenatore, dagli infortuni dei giocatori chiave o dalla concomitanza di altre competizioni. Spesso, le squadre impegnate nelle gare ad eliminazione diretta delle competizioni europee tendono a "sacrificare" il campionato, schierando riserve o adottando un atteggiamento più conservativo per evitare eccessivi dispendi di energie o infortuni. Questi fattori temporali possono avere un impatto significativo sulle

prestazioni delle squadre, e l'approccio delle serie temporali ci consentirebbe di tenerne conto in modo più accurato.

In definitiva, l'approccio delle serie temporali fornirebbe una visione più dettagliata delle variazioni delle prestazioni nel tempo, consentendo di individuare l'intervallo di partite più rilevante da considerare e tenendo conto dell'evoluzione delle condizioni esterne. Integrando tali aspetti, potremmo ottenere previsioni più precise e informate sulle prestazioni di una squadra in una determinata partita.

Appendice A

Codici Python

La funzione `jackknife` permette di calcolare copertura e ampiezza media degli intervalli di previsione che sono stati costruiti tramite la funzione `compute_PIs` utilizzando i metodi Jackknife+ e CV+.

Nel seguito utilizzeremo le librerie `pandas`, `numpy` e `scipy.stats` chiamandole, rispettivamente, `pd`, `np` e `st`.

```
def jackknife(X,y,fun,method_names,results,N,alpha,ntrial):

# input:  X, y          --> dataset di training
#         fun           --> algoritmo di regressione
#         method_names --> metodi di conformal prediction
#         results       --> dizionario contenente i risultati finora
#                           ottenuti
#         N             --> dimensione training set
#         alpha         --> valore per il livello di confidenza
#         ntrial        --> iterazioni

# output: dataframe con copertura e ampiezza degli intervalli per ogni
#         metodo e ogni iterazione

  for itrial in range(ntrial):
```

```
    # creazione training e calibration set

    train_inds = np.random.choice(len(y),N,replace=False)
    test_inds = np.setdiff1d(np.arange(len(y)),train_inds)
    X_train = X.iloc[train_inds]
    y_train = y[train_inds]
    X_val = X.iloc[test_inds]
    y_val = y[test_inds]

    # intervalli di previsione

    PIs = compute_PIs(X_train,y_train,X_val,alpha,fun)

    # calcolo copertura e ampiezza media

    for method in method_names:
      coverage = ((PIs[method]['lower'] <= y_val)&
                  (PIs[method]['upper'] >= y_val)).mean()
      width = (PIs[method]['upper'] - PIs[method]['lower']).mean()
      results.loc[len(results)]=[itrial,method,coverage,width]

results = results.set_index('method')
return(results)
```

```
def compute_PIs(X_train,y_train,X_val,alpha,fun):

# input :  X_train, y_train --> training set
#          X_val, y_val     --> calibration set

# output : dizionario con gli estremi degli intervalli di previsione

  X_train, X_val = X_train.values, X_val.values
  N = len(y_train)
  n1 = X_val.shape[0]

  ################################
  # jackknife+
  ################################

  # calcolo residui con metodo leave-one-out

  resids_LOO = np.zeros(n)
  muh_LOO = np.zeros((n,n1))
  for i in range(N):
    muh_vals_LOO = fun(np.delete(X_train,i,0),np.delete(y_train,i),\
                               np.r_[X_train[i].reshape((1,-1)),X_val])
    resids_LOO[i] = np.abs(y_train[i] - muh_vals_LOO[0])
    muh_LOO[i] = muh_vals_LOO[1:]
  ind_q = (np.ceil((1-alpha)*(N+1))).astype(int)

  ################################
  # CV+
  ################################

  # creazione fold
```

```
K = 10
n_K = np.floor(N/K).astype(int)
base_inds_to_delete = np.arange(n_K).astype(int)

# calcolo residui con metodo cross validation

resids_LKO = np.zeros(N)
muh_LKO = np.zeros((N,n1))
for i in range(K):
  inds = (base_inds_to_delete + n_K*i).astype(int)
  muh_vals_LKO = fun(np.delete(X_train,inds,0),\
                     np.delete(y_train,inds),\
                     np.r_[X_train[inds],X_val])
  resids_LKO[inds_to_delete] = np.abs(y_train[inds] -
                                muh_vals_LKO[:n_K])
  for inner_K in range(n_K):
    muh_LKO[inds[inner_K]] = muh_vals_LKO[n_K:]
ind_Kq = (np.ceil((1-alpha)*(N+1))).astype(int)

# costruzione intervalli di previsione

PIs =  {'jackknife+' :
        pd.DataFrame(np.c_[np.sort(muh_LOO.T -
                                   resids_LOO,axis=1).T[-ind_q],\
                           np.sort(muh_LOO.T +
                                   resids_LOO,axis=1).T[ind_q - 1]],\
                     columns = ['lower','upper']),\
        'CV+' :
        pd.DataFrame(np.c_[np.sort(muh_LKO.T -
                                   resids_LKO,axis=1).T[-ind_Kq],\
```

```
                                    np.sort(muh_LKO.T +
                                            resids_LKO,axis=1).T[ind_Kq - 1]],\
                            columns = ['lower','upper'])}
  return pd.concat(PIs_dict.values(), axis=1, keys=PIs_dict.keys())}
```

La funzione `jackknife_aB` permette di calcolare copertura e ampiezza media degli intervalli di previsione con il metodo Jackknife+ after Bootstrap, al variare di valori di p e delle funzioni di aggregazione scelte. I Bootstrap $S_1, \ldots, S_B$ vengono generati tramite la funzione `generate_bootstrap_samples`, mentre i modelli $\hat{\mu}_b$ sono allenati sui sottoinsiemi creati sfruttando la funzione `fit_bootstrap`. Infine, la costruzione degli intervalli si ottiene applicando la funzione `compute_PIs_jp_aB`.

```
def jackknife_aB(X_train,y_train,X_val,y_val,fun,results,aggre,B_,\
                 tot_trial,alpha):

# input:  X_train, y_train --> training set
#         X_val, y_val     --> calibration set
#         fun              --> algoritmo di regressione
#         results          --> dizionario contenente i risultati finora
#                              ottenuti
#         agg              --> funzioni di aggregazione
#         B_               --> parametro B_tilde del Teorema 5.2.3
#         tot_trial        --> iterazioni
#         alpha            --> valore per il livello di confidenza

# output: dataframe con copertura e ampiezza degli intervalli per ogni
#         funzione di aggregazione, valore di p = m/n e iterazione

  train_size = len(X_train)
  m_vals = np.round(train_size*np.linspace(0.2,1,num=5)).astype(int)
  for agg in aggre:
    for itrial in range(tot_trial):
```

```
    np.random.seed(98765+itrial)
    for m in m_vals:

      # numero sottinsiemi Bootstrap

      B = int(np.random.binomial(int(B_/((1-1./(1+train_size))**m*\
                                         (1-1./train_size)**m)),\
                                    (1-1./(1+train_size))**m,size=1))

      # intervalli di previsione

      PIs = compute_PIs_jp_aB(X_train,y_train,X_val,alpha,fun,B,m,agg)

      # calcolo copertura e ampiezza media

      coverage = ((PIs['lower']<=y_val)&(PIs['upper']>=y_val)).mean()
      width = (PIs['upper'] - PIs['lower']).mean()
      ratio = round(m/train_size,1)
      results.loc[len(results)]=[agg,itrial,coverage,width,ratio]
return results
```

```
def compute_PIs_jp_aB(X_train,y_train,X_val,alpha,fun,B,agg,m):

# input:  B   --> numero sottoinsiemi Bootstrap
#         agg --> funzione di aggregazione
#         m   --> parametro m del Teorema 5.2.3

# output: dataframe con gli estremi degli intervalli di previsione

  N = len(X_train)
  n1 = len(X_val)

  # addestramento modelli mu_hat sui sottoinsiemi creati tramite
  # Bootstrap

  [boot_idx,boot_pred] = fit_bootstrap(X_train,y_train,\
                                       np.vstack((X_train,X_val)),fun,\
                                       N,m,B)
  in_boot_sample = np.zeros((B,n),dtype=bool)
  for b in range(len(in_boot_sample)):
    in_boot_sample[b,boot_idx[b]] = True

  # calcolo residui

  resids_LOO = np.zeros(N)
  muh_LOO_vals = np.zeros((N,n1))
  for i in range(N):
    b_keep = np.argwhere(~(in_boot_sample[:,i])).reshape(-1)
    if(len(b_keep)>0):
      if agg == 'mean':
        resids_LOO[i] = np.abs(y_train[i] - boot_pred[b_keep,i].mean())
        muh_LOO_vals[i] = boot_pred[b_keep,N:].mean(0)
```

```
    elif agg == 'median':
      resids_LOO[i] = np.abs(y_train[i] -\
                             np.median(boot_pred[b_keep,i]))
      muh_LOO_vals[i] = np.median(boot_pred[b_keep,N:],axis=0)
    elif agg == 'tmean':
      resids_LOO[i] = np.abs(y_train[i] -\
                             st.trim_mean(boot_pred[b_keep,i],0.25))
      muh_LOO_vals[i] = st.trim_mean(boot_pred[b_keep,N:],0.25,axis=0)
ind_q = (np.ceil((1-alpha)*(n+1))).astype(int)

# costruzione intervalli di previsione

return pd.DataFrame(np.c_[np.sort(muh_LOO_vals.T -\
                                  resids_LOO,axis=1).T[-ind_q],\
                          np.sort(muh_LOO_vals.T +\
                                  resids_LOO,axis=1).T[ind_q-1]],\
                                  columns=['lower','upper'])
```

```
def fit_bootstrap(X_train,y_train,X_val,fun,N,m,B):

# input:  N   --> numero osservazioni training set

# output: lista di matrici [samples_idx,predictions]
#         samples_idx --> matrice B x m,  riga b = indici delle
#                         osservazioni appartenenti al b-esimo
#                         sottoinsieme Bootstrap
#         predictions --> matrice B x n1, riga b = predizioni delle
#                         predizioni del calibration set ottenute con
#                         il modello mu_b, n1 = numero osservazioni
#                         calibration set

  if type(X_train) == pd.core.frame.DataFrame:
    X_train = X_train.values
  if type(X_val) == pd.core.frame.DataFrame:
    X_val = X_val.values

  # Bootstrap

  samples_idx = generate_bootstrap_samples(N, m, B)
  n1 = len(X_val)

  # calcolo predizioni sul calibration set

  predictions = np.zeros((B, n1), dtype=float)
  for b in range(B):
    predictions[b] = fun(X_train[samples_idx[b],:],\
                         y_train[samples_idx[b],],X_val)
  return([samples_idx, predictions])
```

```
def generate_bootstrap_samples(N, m, B):

# output: matrice B x m, riga b = indici del b-esimo sottoinsieme
#         Bootstrap

  samples_idx = np.zeros((B, m),dtype=int)
  for b in range(B):
    sample_idx = np.random.choice(n, m)
    samples_idx[b, :] = sample_idx
  return(samples_idx)
```

Bibliografia

[Lewis (2003)] M. Lewis, *Moneyball: The Art of Winning an Unfair Game*, W. W. Norton & Co, 2003.

[Baboota, Kaur (2019)] R. Baboota, H. Kaur, *Predictive analysis and modelling football results using machine learning approach for English Premier League*, International Journal of Forecasting, 35(2), pp. 741–755, 2019.

[Dixon, Coles (1997)] M. J. Dixon, S. G. Coles, *Modelling Association Football scores and Inefficiencies in the Football Betting Market*, Journal of the Royal Statistical Society. Series C (Applied Statistics), 46(2), pp. 265-280, 1997.

[Hughes, Bartlett (2002)] M. D. Hughes, R. M. Bartlett, *The use of performance indicators in performance analysis*, Journal of Sports Sciences, 20(10), pp. 739-754, 2002.

[Hubacek et al. (2019)] O. Hubacek, G. Sourek, F. Zelezny, *Learning to Predict Soccer Results from Relational Data with Gradient Boosted Trees*, Machine Learning, 108(1), pp. 29-47, 2019.

[Berrar et al. (2019)] D. Berrar, P. Lopes, W. Dubitzky, *Incorporating Domain Knowledge in Machine Learning for Soccer Outcome Prediction*, Machine Learning 108 (1), pp. 97-126, 2019.

[Beal et al. (2021)] R. Beal, S. E. Middleton, T. J. Norman, S. D. Ramchurn, *Combining Machine Learning and Human Experts to Predict Match Outcomes in Football: A Baseline Model*, Proceedings of the AAAI Conference on Artificial Intelligence 35(17), pp. 15447-15451, 2021.

[Silver (2013)] N. Silver, *The signal and the noise: the art and science of prediction*, Pengunin UK, 2013.

[Turing (1950)] A. M. Turing, *I.-Computing machinery and intelligence*, Mind, LIX.236, pp. 433–460, 1950.

[Wolpert, Macready (1997)] D. H. Wolpert, W. G. Macready, *No Free Lunch Theorems for Optimization*, IEEE Transactions on Evolutionary Computation, 1(1), pp. 67-82, 1997.

[Smola, Schölkopf (2004)] A. J. Smola, B. Schölkopf, *A Tutorial on Support Vector Regression*, Statistics and Computing, 14(3), pp. 199-222, 2004.

[Quinlan (1986)] J. R. Quinlan, *Induction of decision trees*, Machine learning, 1(1), pp. 81–106, 1986.

[Breiman (1996)] L. Breiman, *Bagging Predictors*, Machine Learning, 24, pp. 123-140, 1996.

[Brieman (2001)] L. Breiman, *Random Forest*, Machine Learning, 45, pp. 5-32, 2001.

[Friedman (2001)] J. H. Friedman, *Greedy function approximation: a gradient boosting machine*, The Annals of Statistic. 29(5), pp. 1189-1232, 2001.

[Chen, Guestrin (2016)] T. Chen, C. Guestrin. *XGBoost: A scalable tree boosting system*, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 13-17, pp. 785-94, 2016.

[Ke et al. (2017)] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Y. Liu, *LightGBM: A Highly Efficient Gradient Boosting Decision Tree*, Advances in Neural Information Processing Systems, 30, pp. 3145-3157, 2017

[Prokhorenkova et al. (2018)] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, A. Gulin, *CatBoost: Unbiased boosting with categorical features*, Proceedings of the 32nd International Conference on Neural Information Processing Systems, 2018.

[Koenker, Basset (1978)] R. Koenker, G. Bassett, *Regression Quantile*, Econometrica, 46(1), pp. 33-50, 1978

[Vovk et al. (2005)] V. Vovk, A. Gammerman, G. Shafer, *Algorithmic learning in a random world*, Sprignger, 2005.

[Barber et al. (2021)] R. F. Barber, E. J. Candès, A. Ramdas, R. J. Tibshirani, *Predictive inference with the jackknife+*, The Annals of Statistic, 49(1), pp. 486-507, 2021.

[Kim et al. (2020)] B. Kim, C. Xu, R. F. Barber, *Predictive Inference Is Free with the Jackknife+-after-Bootstrap*, Proceedings of 34th International Conference on Neural Information Processing Systems, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, H. Lin, 348, pp. 4138-4149, 2020.

[Romano et al. (2019)] Y. Romano, E. Patterson, E. J. Candès, *Conformalized Quantile Regression*, Advances in neural information processing systems, 32, 2019.

[Winter (2002)] E. Winter, *The shapley value*, Handbook of game theory with economic applications, 3, pp. 2025–2054, 2002.

[Sprangers et al. (2021)] O. Sprangers, S. Schelter, M. de Rijke, *Probabilistic Gradient Boosting Machines for Large-Scale Probabilistic Regression*, Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pp. 1510-1520, 2021.

[Brophy, Lowd (2022)] J. Brophy, D. Lowd, *Instance-Based Uncertainty Estimation for Gradient-Boosted Regression Trees*, Proceedings of the 36th Conference on Neural Information Processing Systems, 2022.

[Soccerment] *Soccerment Analytics* https://soccerment.com/football-analytics