Università degli studi di Milano–Bicocca
Scuola di Economia e Statistica

Corso di Laurea Magistrale in
Scienze Statistiche ed Economiche

Reinforcement learning per sistemi dinamici: studio e applicazione di metodi Fitted Q-Iteration

Relatore: Prof. Matteo Borrotti
Correlatrice: Dott.ssa Valentina Zangirolami
Correlatore: Prof. Antonio Candelieri

Tesi di laurea di:
Giorgia Faccanoni
Matricola N. 871869

Anno Accademico 2024/2025

Indice

Introduzione

Negli ultimi decenni si è consolidata l'idea che le macchine possano sfruttare le informazioni a loro disposizione per risolvere problemi e prendere decisioni, ispirandosi ai meccanismi dell'apprendimento umano. La scienza dell'apprendimento riveste oggi un ruolo centrale nei campi della statistica, del *machine learning* e dell'intelligenza artificiale, trovando applicazioni trasversali in ambiti quali la medicina, la finanzia e l'ingegneria. Gli esseri umani apprendono in modo naturale interagendo con l'ambiente circostante: da queste interazioni emerge una grande quantità di informazioni sulle cause, sugli effetti e sulle conseguenze delle azioni, che permettono di adattare il comportamento al fine di raggiungere determinati obiettivi. Seguendo questo modello, sono stati sviluppati diversi metodi che consentono alle macchine di affrontare problemi decisionali complessi in ambienti dinamici.

In questo ambito, il *Reinforcement Learning* (RL) si distingue come un approccio particolarmente efficace. In tale contesto, un agente di apprendimento—che può essere rappresentato da un *software*, un robot o un veicolo autonomo—interagisce con l'ambiente circostante, riceve ricompense (*reward*) sotto forma di valori numerici, che variano in base alle azioni eseguite, e, attraverso un processo di apprendimento iterativo, affina progressivamente le proprie decisioni con l'obiettivo di massimizzare la ricompensa totale cumulata (Sutton & Barto, 2018). Le origini di RL risalgono al campo della psicologia comportamentale, dove si è dimostrato che è possibile insegnare agli animali ad eseguire azioni, anche complesse, mediante semplici meccanismi, come la somministrazione di un premio dopo aver svolto l'azione richiesta. L'idea di fondo è che un agente—umano, animale o artificiale—possa imparare a ottimizzare il proprio comportamento basandosi sulle esperienze passate.

Tra i diversi algoritmi di RL proposti in letteratura, il *Q-learning* rappresenta uno dei metodi più noti e diffusi. Tuttavia, la sua applicazione diretta risulta limitata quando si hanno a disposizione spazi di stato e di azione ampi o continui, oppure quando non è possibile interagire direttamente con l'ambiente. In questi scenari trova applicazione il metodo *Fitted Q-Iteration* (FQI), un approccio *offline* in cui

l'agente apprende sulla base di un dataset preesistente, ricevendo un insieme di dati, invece di interagire direttamente con l'ambiente. A partire dai dati disponibili, l'agente è in grado di determinare la regola decisionale da utilizzare ad ogni istante temporale (*policy* o politica), cercando di avvicinarsi il più possibile a quella ottimale. In particolare, l'agente calcola un'approssimazione della funzione valore-azione Q, la quale fornisce una misura di bontà di trovarsi in un determinato stato seguendo una determinata politica. FQI sfrutta modelli di apprendimento supervisionato per approssimare la funzione valore-azione Q e determinare *policy* ottimali a partire dal dataset disponibile (Ernst et al., 2005). Questo metodo è impiegato per risolvere problemi decisionali sequenziali e consente di identificare le azioni che, nel tempo, massimizzano la ricompensa cumulata.

Nei modelli di previsione tradizionali, l'obiettivo è ottenere una previsione puntuale. Tuttavia, queste previsioni spesso non forniscono indicazioni sul loro grado di affidabilità. Di conseguenza, un aspetto cruciale nei problemi reali è la quantificazione dell'incertezza nelle stime prodotte dai modelli al fine di garantire decisioni più sicure e affidabili. In scenari applicativi, come il controllo di sistemi dinamici, non è sufficiente disporre di una previsione puntuale: è fondamentale conoscere anche il livello di affidabilità delle stime per supportare adeguatamente le decisioni operative. Per affrontare questo problema è stata sviluppata la tecnica di *Conformal Prediction* (CP), che offre uno strumento semplice e di facile comprensione per la costruzione di intervalli di previsione, i quali, per definizione, contengono il vero valore della variabile di interesse con elevata probabilità (Angelopoulos & Bates, 2021). CP è un approccio *distribution-free*, in grado di quantificare l'incertezza delle previsioni generate da algoritmi predittivi arbitrari. Inoltre, CP richiede che i dati soddisfino la proprietà di scambiabilità, ossia che la probabilità congiunta della sequenza di osservazioni non cambia al variare dell'ordine dei dati. Tuttavia, nella pratica questa ipotesi può essere violata a causa di fenomeni di *distribution shift*, quando la distribuzione dei dati cambia tra il *training set* e il *test set*. Sotto questa ipotesi costruire intervalli di previsione seguendo il metodo CP classico comporta a una distorsione nell'ampiezza degli intervalli. Per superare questo limite è stato sviluppato il metodo di *Weighted Conformal Prediction* (Tibshirani et al., 2019). Nel contesto dei problemi decisionali sequenziali, l'obiettivo dell'applicazione della tecnica di CP è costruire intervalli di previsione per le stime della funzione valore-azione Q, prodotte dall'algoritmo FQI. L'algoritmo FQI si sviluppa nell'ambito dei problemi decisionali a più stadi e per costruire intervalli di previsione ottimali è stato proposto

un algoritmo ad hoc, estendendo l'approccio *Conformal off-policy prediction* (COPP) proposto da Zhang et al. (2023).

Nell'ambito dei metodi offline di RL il dataset a disposizione deriva dall'applicazione di un sistema dinamico, che si modifica nel tempo seguendo una politica definita. Questa tesi propone lo studio e l'applicazione dell'algoritmo FQI con diversi metodi di regressione, svolgendo l'applicazione pratica sul dataset *House Heating* proposto in Candelieri et al. (2023). L'obiettivo dell'applicazione consiste nel controllare la temperatura interna di un edificio nell'arco di 60 giorni, selezionando azioni che garantiscano al contempo condizioni termiche confortevoli e la minimizzazione dei costi di gestione.

Questa tesi propone lo studio e l'applicazione dell'algoritmo FQI, integrato con metodi di CP.

L'obiettivo principale di questo lavoro è quello di valutare l'efficacia dell'approccio basato su FQI nel controllo di sistemi dinamici, mettendone in luce i vantaggi e le possibili criticità. In particolare, sono stati confrontati diversi modelli di regressione all'interno dell'algoritmo FQI, evidenziando il modello con cui si sono ottenuti i risultati migliori, in termini di reward cumulata. Un secondo obiettivo consiste nell'analizzare il contributo offerto dalla nuova metodologia di CP per algoritmi di tipo sequenziale, come FQI, attraverso i risultati ottenuti nell'applicazione, ponendo così le basi per possibili futuri sviluppi metodologici nell'ambito dell'incertezza nel *reinforcement learning*.

Nel primo capitolo viene introdotto il RL nelle sue componenti fondamentali, soffermandosi in particolare sui processi decisionali di *Markov*.

Nel secondo capitolo è dedicato ai principali modelli di RL, con un approfondimento sul Q-learning e sulle sue varianti *off-policy* e *offline*, per poi presentare in dettaglio l'algoritmo FQI e i metodi di regressione utilizzati al suo interno.

Nel terzo capitolo viene affrontato il tema di CP per la costruzione di intervalli di previsione, inizialmente nella sua formulazione classica per poi specializzarlo nell'ambito degli algoritmi *off-policy*, in particolare all'algoritmo FQI.

Nel quarto ed ultimo capitolo, i metodi discussi vengono applicati al dataset *House Heating*, relativo al controllo della temperatura interna di un edificio. Inizialmente viene applicato l'algoritmo FQI per l'individuazione di azioni ottimali permettendo il controllo della temperatura. Successivamente viene impiegato il metodo di CP per la stima dell'incertezza delle previsioni ottenute da FQI.

Capitolo 1

Reinforcement Learning

In questo capitolo viene introdotto il reinforcement learning in modo generale, definendo l'idea di base e gli aspetti chiave. In seguito, verranno spiegate più dettagliatamente: le caratteristiche principali, gli elementi principali del reinforcement learning e la differenza con i metodi supervisionati di machine learning. Viene illustrato come si procede per la selezione delle azioni migliori, la stima delle funzioni valore e i processi di Markov che sono la base teorica degli algoritmi di reinforcement learning. Le referenze principali utilizzate per questo capitolo sono Sutton & Barto (2018), Agarwal et al. (2019) e Munos (2005).

Negli ultimi decenni lo sviluppo di tecniche di apprendimento automatizzato nell'ambito del machine learning e dell'intelligenza artificiale ha avuto un enorme crescita. Il machine learning si concentra sullo sviluppo di algoritmi e metodi che sono in grado di prendere decisioni, individuare dei pattern e fare previsioni in modo autonomo utilizzando grandi quantità di dati. Il **reinforcement learning** (RL) è una branca dell'apprendimento automatizzato che si sviluppa nell'ambito di problemi sequenziali, in cui un agente interagisce con l'ambiente in cui si trova, riceve delle ricompense e, in base a quest'ultime, è in grado di migliorarsi e apprendere in modo autonomo senza che nessuno gli imponga una regola su come adattarsi. Si può fare un semplice confronto con la vita di tutti i giorni, in cui le persone apprendono in modo autonomo interagendo con l'ambiente circostante. Basti pensare ad un bambino che impara ad andare in bicicletta, impara a stare in equilibrio e a pedalare provando e osservando gli altri senza che nessuno gli dica esplicitamente come fare. La connessione tra persone e ambiente produce una grande quantità di informazioni sulle cause e sugli effetti che hanno le azioni che si compiono. Nel corso della vita le interazioni che si hanno con l'ambiente intorno a noi sono una fonte principale di conoscenza e di apprendimento. Inoltre, si è in grado di influenzare l'ambiente che

ci circonda adattando le nostre azioni e i nostri comportamenti. Si può dire quindi, che l'interazione tra ambiente e agente è fondamentale per apprendere ed essa sta alla base delle maggiori tecniche di apprendimento automatizzato, soprattutto del reinforcement learning. Il reinforcement learning si tratta, quindi, di un insieme di tecniche e di modelli adatti a problemi sequenziali, in cui un agente di apprendimento interagisce con l'ambiente che gli sta intorno e prende decisioni ottimali, in modo sequenziale, osservando ad ogni passo una ricompensa che riceve dall'ambiente con l'obiettivo di massimizzarla. Il reinforcement learning si basa su 3 caratteristiche fondamentali:

- si tratta di un problema a "ciclo chiuso", poiché le decisioni che vengono prese sono influenzate dallo stato al passo corrente e a loro volta influenzano lo stato al passo successivo;
- l'agente di apprendimento non ha indicazioni su quali azioni attuare, deve scoprire quali sono le migliori soltanto provandole;
- le conseguenze delle azioni attuate non influiscono soltanto la ricompensa immediata, ma anche quella dei periodi successivi.

Storicamente, la prima classe di metodi per la risoluzione di problemi decisionali a più stadi è stata la programmazione dinamica (DP). Il termine è stato inizialmente utilizzato negli anni quaranta dal matematico Richard Bellman per descrivere il processo di risoluzione di problemi in cui è necessario trovare le soluzioni migliori in modo sequenziale. Questi algoritmi classici hanno un'utilità pratica limitata in Reinforcement Learning a causa delle seguenti ragioni. In primo luogo, richiedono una conoscenza completa dell'ambiente di apprendimento; in termini statistici ciò significa una conoscenza della distribuzione multivariata dei dati. In molte applicazioni è spesso impraticabile assumere completa conoscenza del meccanismo di generazione dei dati. In secondo luogo, i metodi di programmazione dinamica sono molto costosi dal punto di vista computazionale e sono difficili da gestire in ambienti con elevata dimensionalità. Tuttavia, la programmazione dinamica costituisce un utile base teorica per anche i metodi recenti di reinforcement learning.

Il reinforcement learning si distingue in diverse categorie a seconda delle caratteristiche metodologiche (o di stima) dei criteri. Ad esempio, alcuni si basano su dataset pre-esistenti di dimensione finita, metodi **offline**; mentre altri ricevono i dati man mano sfruttando l'interazione tra agente e ambiente, i quali prendono il nome di metodi **online**. Per entrambi i metodi esiste un'ulteriore distinzione:

i metodi **model-based**, in cui si possono definire un modello dell'ambiente che stima la probabilità di transizione e la funzione di ricompensa oppure si assume che tali quantità siano note, lo si utilizza per simulare scenari futuri e pianificare le azioni da compiere; e metodi **model-free based** in cui l'agente impara direttamente dall'esperienza senza costruire un modello specifico dell'ambiente. Quest'ultimo approccio si suddivide a sua volta in due categorie principali:

1. modelli **value-based**, in cui l'agente stima una funzione di valore $V(s)$ oppure una funzione di valore-azione $Q(s, a)$, che rappresenta il valore atteso dell'esecuzione di un'azione in un determinato stato. Un esempio di questo approccio è l'algoritmo Q-learning;

2. modelli **policy-based** in cui l'agente apprende direttamente la policy π, la quale può essere deterministica o stocastica. In entrambi i casi, l'obiettivo è stimare una funzione che, sulla base dello stato corrente, determini le azioni cercando sempre di massimizzare la somma cumulata delle reward.

In questa tesi viene utilizzato e presentato in modo dettagliato l'algoritmo **Fitted Q-iteration (FQI)**. In poche parole esso si tratta di un algoritmo offline e batch che stima la funzione valore $Q(s, a)$ a partire da un dataset fisso di esperienze, composto da stato, azione, stato successivo e reward osservata (s, a, s', r), senza interagire attivamente con l'ambiente. FQI si basa sull'approssimazione supervisionata della Q-function usando metodi di regressione come random forest, modelli additivi generalizzati, reti neurali o altri modelli di machine learning.

1.1 Reinforcement learning vs supervised learning

Il reinforcement learning ha caratteristiche differenti rispetto al supervised learning. L'apprendimento supervisionato è il tipo di apprendimento più studiato e utilizzato in ricerca nell'ambito del machine learning. Si tratta di algoritmi in cui il modello impara a fare previsioni o classificazioni in base a una serie di dati in cui si osserva anche la variabile risposta. Il modello, osservando le covariate e la variabile target, è in grado di individuare pattern nei dati e formulare previsioni sui valori della variabile risposta, producendo valori continui nel caso della regressione e valori discreti nel caso della classificazione. L'apprendimento supervisionato consiste nell'apprendere da dati etichettati, forniti da un supervisore esterno esperto. Ogni dato rappresenta una possibile interazione tra la variabile risposta e le variabili indipendenti, l'obiettivo del modello è quello di imparare a generalizzare quell'interazione e quindi di prevedere

in modo corretto la variabile target su nuovi dati non presenti nel dataset di addestramento. In alcune situazioni è però difficile trovare una relazione già osservata e quindi è importante che l'algoritmo sia in grado di generalizzare. Di seguito vengono elencate le differenze principali tra il metodo supervised learning e il reinforcement learning.

- I metodi supervisionati osservano una variabile target, mentre il reinforcement learning osserva una ricompensa in base all'azione che sceglie, quindi l'obiettivo dei due metodi è molto diverso. I metodi supervisionati cercano di prevedere la variabile target nel modo più corretto possibile e quindi cercano di minimizzare l'errore di previsione, mentre i metodi di reinforcement learning scelgono la miglior azione da applicare ad ogni passo in modo da massimizzare la ricompensa che ottiene nel tempo.

- Nei metodi di apprendimento supervisionato, come già detto, i dati sono strutturati in un dataset composto da variabili indipendenti e da una variabile dipendente o variabile risposta. Mentre nel reinforcement learning i dati possono essere strutturati in una tabella in cui ogni riga è formata dallo stato corrente, dall'azione osservata, dallo stato successivo e dalla reward osservata. Nel caso di reinforcement offline la tabella ha dimensione finita mentre nel caso online i dati si continuano ad aggiungere ed aggiornare nel tempo, trattandosi quindi di dati illimitati.

- Un'ulteriore differenza sta nel fatto che i dati nel supervised learning sono indipendenti e identicamente distribuiti, ovvero ogni riga del dataset è indipendente dal resto; questa assunzione non vale nel caso del reinforcement learning in cui i dati sono di tipo sequenziale, ovvero ogni riga del dataset rappresenta un determinato tempo t e la riga successiva sarà indicizzata come $t + 1$.

- Nei metodi nei metodi supervisionati, visto che ogni riga del dataset è indipendente, la previsione che il modello esegue per una istanza non influenza le altre. Mentre nel caso di modelli RL le decisioni dell'agente influiscono sui dati successivi, poiché l'azione ottima scelta ad ogni passo influisce sullo stato successivo.

In poche parole, l'obiettivo di reinforcement learning è trovare la politica ottima, mentre nel caso supervisionato è quello di trovare la previsione migliore. Questo, però, non esclude che per alcuni metodi di reinforcement learning, i metodi supervisionati

entrano in gioco per trovare approssimazioni di quantità utili per derivarne il criterio ottimo.

1.2 Elementi del Reinforcement Learning

I metodi di reinforcement learning sono in grado di stabilire un'interazione attiva tra l'agente e l'ambiente che lo circonda, in cui l'agente cerca di raggiungere l'obiettivo, ovvero la massimizzazione delle reward cumulate, nonostante l'incertezza e la non perfetta conoscenza dell'ambiente. La formulazione più semplice del problema è rappresentata dai **Processi decisionali di Markov**. Per interpretare al meglio questo tipo di algoritmi bisogna definire i componenti principali del reinforcement learning.

- **Spazio degli stati** $\mathcal{S}$: può essere discreto o continuo, si tratta dell'insieme di tutti gli stati possibili. Lo stato corrente s rappresenta la condizione che assume l'ambiente in cui si trova l'agente di apprendimento al tempo t.

- **Spazio delle azioni** $\mathcal{A}$: può essere discreto o continuo. Rappresenta l'insieme delle azioni tra cui l'agente seleziona quella che è in grado di portare a livelli della reward più elevati. Si parla di azione discreta A quando, ad esempio nell'ambito medico, si deve scegliere tra differenti tipi di cure o farmaci da somministrare ad un paziente. Mentre si parla di azione A continua quando, ad esempio nell'ambito del controllo della guida di veicoli autonoma, si deve scegliere la manovra che deve compiere il veicolo. L'azione è continua anche nell'ambito di applicazione di questa tesi nel dataset *House heating* in cui l'obiettivo è quello di impostare una determinata temperatura all'interno di un edificio ogni 30 minuti nell'arco della giornata in modo da mantenere la temperatura costante per tutto il giorno.

- **Policy** π: si tratta del modo in cui l'agente agisce in un dato periodo. È il cuore del RL poiché essa definisce in modo sufficiente il comportamento dell'agente. La policy descrive la regola decisionale da utilizzare ad ogni istante temporale (o decisionale), si tratta quindi di una strategia tramite cui si decide la sequenza di azioni da applicare. La policy può essere **deterministica**, ovvero dipende da una funzione deterministica d_t, e $a = \pi(S)$ rappresenta l'azione presa allo stato S dettata dalla policy deterministica sottostante. Mentre $\pi(a|S)$ rappresenta la probabilità di attuare l'azione a dato stato S sotto policy **stocastica**.

- **Funzione di transizione** p: si tratta della probabilità di passare dallo stato corrente allo stato successivo sulla base dell'azione scelta dall'agente: $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$. La distribuzione dello stato iniziale è data da $p_0 : \mathcal{S} \to [0, 1]$.

- **Funzione Reward** R: si tratta dell'obiettivo principale di un problema di RL, è una funzione dell'ambiente, quindi dello stato e delle azioni, che mappa lo spazio stato e azione in un numero $R : \mathcal{S} \times \mathcal{A} \to [a, b]$. A volte, il codominio può essere rappresentato anche dai numeri reali $\mathbb{R}$. Dopo aver eseguito l'azione, l'ambiente restituisce all'agente un numero che rappresenta la reward r. L'obiettivo dell'agente è quello di massimizzare il totale delle reward che riceve nel lungo periodo. La reward r dipende dallo stato corrente s e dall'azione corrente a, quindi l'unico modo che l'agente ha per influenzare direttamente la reward è agire sull'azione.

- **Funzione Valore o Value function** $V(s)$: si tratta di una funzione che fornisce una misura di bontà dell'agente in un dato stato s che segue una determinata policy π. Rappresenta il totale delle reward che un agente si aspetta di accumulare nel tempo, partendo da un certo stato e avendo tenuto conto degli stati visitati e delle reward disponibili a quegli stati. L'agente cerca di individuare azioni che portino ad un valore alto della funzione valore. Mentre la reward viene fornita direttamente dall'ambiente ad ogni passo, stimare la funzione valore può essere complesso, poiché deve essere riformulata in modo ricorsivo per ogni stato visitato e azione considerata.

- L'ultimo elemento, per gli algoritmi *model-based* del reinforcement learning è il **modello dell'ambiente**. Per modello si intende un'entità in grado di simulare il comportamento dell'ambiente. Per esempio, dato uno stato e un'azione, il modello è il grado di predire il risultato del prossimo stato e della prossima reward.

Come mostrato in Figura 1.1, l'algoritmo RL funziona nel seguente modo: ad ogni tempo t l'agente riceve in input uno stato $S_t \in \mathcal{S}$ e la reward $R_t \in \mathcal{R}$ del passo precedente e in base a questi decide l'azione $A_t \in \mathcal{A}$ migliore da compiere che lo porterà allo stato successivo S_{t+1}. Questo processo avviene in modo ricorsivo per un numero di iterazioni che può essere pre-stabilito oppure fino ad arrivare a convergenza. L'agente ad ogni iterazione cerca di modificare le azioni in modo da raggiungere reward ad ogni passo più elevate oppure da avere una ricompensa maggiore nel lungo periodo, considerando le reward cumulate.

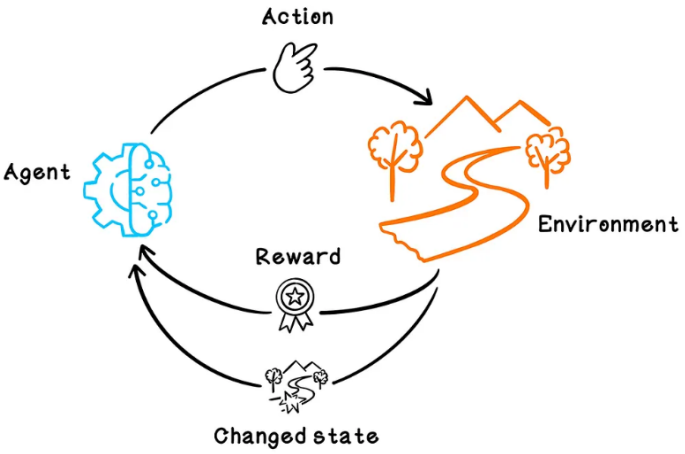

Figura 1.1: Algoritmo di Reinforcement learning (Rhurbans, 2023).

Quando l'algoritmo RL si definisce episodico, la formulazione prevede che vi siano un numero finito di **episodi**, ovvero una collezione di traiettorie composte da diversi stati e da diverse azioni che osserva e intraprende l'agente. Si osservano quindi N episodi/traiettorie del tipo: $\{(s_0^1, a_0^1, ..., s_{T-1}^1, a_{T-1}^1, s_T^1), ..., (s_0^N, a_0^N, ..., s_{T-1}^N, a_{T-1}^N, s_T^N)\}$. Se ognuna di queste traiettorie ha un tempo T finito allora di parla di **orizzonte finito**, invece si parla di **orizzonte infinito** quanto $T = \infty$.

1.3 Procedura di selezione delle azioni e Markov Decision Process

In questo contesto si considerano metodi di Reinforcement learning ad orizzonte finito, la notazione utilizzata deriva da Puterman (2014). Come già definito precedentemente, l'agente ad ogni istante temporale seleziona un'azione in base ad una regola decisionale che può essere una funzione deterministica o casuale, ovvero una funzione di probabilità. La sequenza delle regole decisionali caratterizza la strategia sottostante, la policy: $\pi = (d_1, ..., d_T)$. Una caratteristica fondamentale della policy è che solitamente si assume **stazionaria** (Shi, 2025). Questo implica che la regola decisionale non cambia nel tempo: $d_t = d$, $\forall t \in T$ e la policy viene definita come:$\pi = (d, ...d)$.

Un'altra importante proprietà di cui gode l'ambiente e gli stati nella maggior parte degli algoritmi di RL è la proprietà di **Markovianità** (Shi, 2025). In generale, lo stato S al passo $t + 1$ e la reward R che ne deriva vengono definiti secondo la seguente funzione di probabilità:

$$P(S_{t+1} = s', R_t = r|s_0, a_0, s_1, a_1, r_1, ..., s_t, a_t, r_t) \tag{1.1}$$

In questo caso lo stato al passo successivo $t+1$ dipende dall'intera storia passata, ovvero dipende da tutte le azioni, le reward e gli stati osservati partendo dal passo $t = 0$. La proprietà di Markovianità fa si che lo stato al passo successivo dipenda soltanto dal passo immediatamente precedente, ovvero:

$$P(S_{t+1} = s', R_t = r|S_t, A_t) \tag{1.2}$$

Essendo P una funzione di probabilità vale che:

$$\sum_{s' \in S} \sum_{r \in R} P(s', r|s, a) = 1, \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{1.3}$$

In altre parole vale la proprietà di Markov se l'equazione (1.1) equivale all'equazione (1.2) per ogni possibile sequenza di stati e azioni passate. Grazie a questa proprietà è possibile prevedere lo stato successivo e la reward associata conoscendo soltanto la coppia stato-azione all'istante precedente. Un algoritmo di reinforcement learning che soddisfa la proprietà di Markov è detto **Markov Decision Process (MDP)**, se l'insieme degli stati e delle azioni sono finiti allora è detto **finite Markov Decision Process**. Un finite MDP è definito tramite un set di stati e azioni ad ogni passo, e lo stato al passo successivo viene calcolato tramite la funzione di transizione $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$:

$$p(s'|s, a) = P(S_{t+1} = s'|S_t = s, A_t = a) = \sum_{r \in R} p(s', r|s, a) \tag{1.4}$$

Analogamente, data una coppia di stato-azione, la reward attesa è definita dalla funzione reward $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$r(s, a, s') = E[R_t|S_t = s, A_t = a, S_{t+1} = s'] \tag{1.5}$$

La struttura dei Markov Decision Process rappresenta un'astrazione dei modelli di reinforcemnt learning e potrebbe non essere sufficiente per rappresentare tutti i problemi di apprendimento decisionale, ma si è dimostrato essere ampiamente utile e applicabile. Naturalmente, gli stati e le azioni variano molto dal contesto in cui si applicano i modelli, e il modo in cui vengono rappresentati può influenzare fortemente le prestazioni.

1.4 Reward attesa e funzione valore

Come già descritto in precedenza, l'obiettivo dell'agente è quello di massimizzare la reward cumulata che riceve nel lungo periodo. In generale, si cerca di massimizzare la reward attesa, dove per reward attesa si intende una funzione della sequenza di reward che l'agente osserva ad ogni stato $r_1, ..., r_T$ per tutti gli step osservati fino al tempo T. La reward attesa più semplice da considerare è la somma delle reward ad ogni passo, se si considera partendo dal passo t si ottiene:

$$G_t = R_{t+1} + R_{t+2} + ... + R_T \tag{1.6}$$

Solitamente questo approccio viene utilizzato quando si ha una sequenza di passi finiti. Ogni episodio termina in uno stato terminale che rappresenta il passo al tempo T, quando si è nell'ambito di orizzonte finito. Lo stato iniziale di ogni episodio è indipendente da quello che è successo nell'episodio precedente, quindi è indipendente dallo stato terminale precedente. Questo permette di poter considerare episodi successivi come indipendenti tra di loro. Quando, invece, si è nel caso di orizzonte infinito, quindi quando $T = \infty$, la reward cumulata è espressa come la somma delle reward scontata, i.e. Equazione 1.7.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{i=0}^{\infty} \gamma^i R_{t+i+1} \tag{1.7}$$

dove γ è il tasso di sconto e $0 \leq \gamma \leq 1$. Il tasso di sconto determina l'incidenza delle ricompense future. Una ricompensa ricevuta tra $i+1$ passi avrà un valore pari a γ^i volte quello di una reward ricevuta immediatamente. Se $0 \leq \gamma \leq 1$ allora la sommatoria infinita converge a un valore finito, purché la sequenza di reward sia limitata, poiché $\sum_{i=0}^{\infty} \gamma^i = \frac{1}{1-\gamma}$. Quando γ si avvicina a 0 l'algoritmo da importanza soprattutto alle reward dei passi più vicini, mentre quando γ si avvicina a 1 tiene sempre di più in conto delle reward future. Il problema sta nel fatto che non è possibile conoscere il valore delle reward future, quindi negli algoritmi di reinforcement learning viene definito il concetto di **Value funcion**.

Come già anticipato, un'altro importante componente dei metodi di RL è la funzione valore $V(s)$, ovvero una funzione degli stati che rappresenta una stima della bontà dell'agente di trovarsi in quel determinato stato, in altre parole definisce quanto è "accurato/buono" per l'agente trovarsi in quello stato in quel determinato momento. La value function viene definita in base alle ricompense future che riceverà

l'agente, ed esse sono determinate dal comportamento che attuerà l'agente che segue una precisa policy π. Una policy definisce la probabilità di selezionare ciascuna azione possibile in ogni stato: in particolare, $\pi(a \mid s)$ rappresenta la probabilità di scegliere l'azione $A_t = a$ trovandosi nello stato $S_t = s$. La funzione valore $v_\pi(s)$ esprime il valore atteso delle ricompense future a partire da uno stato s, assumendo che l'agente segua la policy π. In altre parole, $v_\pi(s)$ è la somma attesa delle ricompense che l'agente riceverà nel tempo, iniziando dallo stato s e seguendo la policy π.

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s\right] = \mathbb{E}_\pi \left[\sum_{t \geq 0} \gamma^t R_{t+1} | S_t = s\right] \forall s \in \mathcal{S} \tag{1.8}$$

Un'altra misura di bontà del comportamento dell'agente è definita dalla **funzione Valore-Azione** $Q(a, s)$, che rappresenta quanto è "buona" l'azione intrapresa dall'agente considerando che si trovi nello stato s seguendo una determinata policy π, ed è definita da:

$$q_\pi(a, s) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a\right] = \mathbb{E}_\pi \left[\sum_{t \geq 0} \gamma^t R_{t+1} | S_t = s, A_t = a\right] \tag{1.9}$$

Per ogni policy π ed ogni stato s queste funzioni possono essere riscritte ricorsivamente nel seguente modo:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi \left[\sum_{t \geq 0} \gamma^t R_{t+1} \mid S_t = s\right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \sum_{t \geq 1} \gamma^t R_{t+1} \mid S_t = s\right] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \mathbb{E}_\pi \left[\sum_{t \geq 1} \gamma^t R_{t+1}\right]\right] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma v_\pi(s')\right] \end{aligned} \tag{1.10}$$

dove $s', s \in \mathcal{S}$, $a \in \mathcal{A}$ e $r \in \mathbb{R}$, inoltre $\pi(a|s)$ rappresenta la probabilità di scegliere l'azione a dato lo stato s e $p(s', r \mid s, a)$ rappresenta la probabilità di transizione allo stato s'. Il terzo passaggio dell'equazione si ottiene per definizione di valore atteso. La **ricorsività** di queste funzioni è una proprietà fondamentale per gli algoritmi di reinforcement learning e per la programmazione dinamica, ed essa si lega strettamente alle equazioni di Bellman. L'Equazione 1.10 rappresenta a tutti gli effetti l'**equazione di Bellman** per $v_\pi(s)$; essa esprime la relazione tra il valore

di uno stato e il valore degli stati successivi, in pratica rappresenta una media tra tutte le possibilità che si possono avere come stati successivi, pesando ogni stato con la relativa probabilità. L'equazione di Bellman, che prende il nome da Richard Bellman matematico degli anni quaranta, è un concetto fondamentale nel campo del reinforcement learning. Fornisce una scomposizione ricorsiva per risolvere il problema di trovare una politica ottimale. L'equazione di Bellman è fondamentale per i differenti tipi di algoritmi RL, tra cui il temporal difference learning (TD) e il Q-learning, che verranno presentati nel Capitolo 2.3.

Quando il numero di stati e azioni è elevato le funzioni valore vengono approssimate, utilizzando delle funzioni i cui parametri vengono aggiornati in modo da avvicinarsi al valore corrispondente di reward di un particolare stato o di una coppia stato-azione. Tali funzioni possono essere trovate sfruttando metodi di supervised learning.

1.4.1 Funzione valore ottima

Trovare la politica ottimale che porta ai risultati migliori possibili significa trovare una politica $\pi(a|s)$ che sia in grado di ottenere ricompense alte per ogni tempo t e soprattutto nel lungo periodo. Per quantificare le reward future si stimano le funzioni valore $V(s)$, quindi le funzioni valore sono in grado di stabilire un ordinamento tra le diverse politiche poiché esse rappresentato la somma delle reward future sotto una determinata policy. Una policy π è considerata migliore o uguale a una policy π' se la sua reward attesa futura è maggiore o uguale a quella di π' per ogni stato.

$$\pi \geq \pi' \Leftrightarrow v_\pi(s) \geq v_{\pi'}(s) \tag{1.11}$$

Grazie a questa relazione tra policy e funzioni valore si può definire una politica ottimale che è migliore o uguale a tutte le altre e viene definita come π^*, anche se possono esserci più politiche ottimali. La value function associata alla policy ottimale v^* è definita come:

$$v^*(s) = \max_\pi v_\pi(s), \forall s \in \mathcal{S} \tag{1.12}$$

Inoltre si può definire anche la funzione valore-azione ottima come quella che segue la policy ottimale:

$$q^*(a, s) = \max_\pi q_\pi(a, s), \forall s \in \mathcal{S}, a \in \mathcal{A} \tag{1.13}$$

Essa può essere riscritta in termini di funzione valore come:

$$q_\pi^*(a, s) = \mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1})|S_t = s, A_t = a] \tag{1.14}$$

Dato che la funzione valore ottima v^* corrisponde a una funzione valore associata alla politica ottima allora vale la proprietà di ricorsività dimostrata in precedenza e quindi vale l'equazione di Bellman:

$$v^*(s) = \max_{a \in A} q_{\pi^*}(a, s) = \max_{a \in A} \sum_{s', r} p(s', r|s, a)[r + \gamma v^*(s')] \tag{1.15}$$

L'equazione di Bellman per v^* in 1.15 evidenzia che la funzione valore ottima (relativa quindi alla policy ottima) corrisponde al valore atteso delle reward per l'azione migliore in un dato stato.

In questa sezione sono state definite le politiche ottimali e le funzioni valore o valore-azione, in cui un agente apprende quale sia la policy che porta a risultati ottimali, questo vale a livello teorico, ma a livello pratico questo accade raramente. Anche avendo un modello completo e accurato della dinamica dell'ambiente, di solito non è possibile calcolare una politica ottimale risolvendo semplicemente l'equazione di Bellman. La policy ottimale può essere stimata solo a costo di una grande quantità di calcoli estremamente complessi, quindi un problema fondamentale è sicuramente la potenza computazione dell'agente. Anche la memoria disponibile rappresenta un vincolo importante. Una grande quantità di memoria è spesso necessaria per costruire approssimazioni delle funzioni valore, delle politiche e dei modelli. Quando il numero di stati e azioni non è elevato è possibile costruire delle tabelle in cui ogni entrata rappresenta ogni stato o ogni coppia di stato e azione, ma ciò accade raramente. La maggior parte delle volte il numero di stati e azioni è molto elevato e quindi impossibile da poter rappresentare in tabella. In questi casi, come già detto, le funzioni devono essere approssimate usando funzioni parametriche o non parametriche.

Capitolo 2

Modelli Q-learning e fitted Q-iteration

In questo capitolo viene presentata la teoria dei modelli di reinforcement learning che verranno poi utilizzati in ambito di applicazione, partendo da modelli meno avanzati, quindi dalla programmazione dinamica (DP), passando per i metodi Temporal-difference learning (TD), arrivando poi alla definizione di Q-learning. Si descrive, inoltre, la differenza tra i metodi online e offline, le loro caratteristiche e viene poi presentato l'algoritmo di **fitted Q-iteration** in ambito off-policy utilizzato nell'applicazione.

Infine, viene descritto il modello supervisionato di machine learning che si utilizza principalmente nell'applicazione di questa tesi all'interno dell'algoritmo di fitted Q-iteration. In particolare una volta calcolati i nuovi valori target tramite la massimizzazione dei valori Q rispetto all'azione, come verrà spiegato in seguito nell'algoritmo FQI, si allena un modello di regressione per trovare una relazione tra i nuovi valori target e lo stato e l'azione osservati nel dataset a disposizione. Il modello proposto è la regressione semi-parametrica tramite **spline**, in particolare un modello additivo generalizzato **GAM** (*Generalized additive model*) in cui viene inserita una spline su ogni covariata. Nell'ultima sezione di questo capitolo vengono presentati diversi tipi di spline, considerando quelli che sono stati poi utilizzati all'interno dell'applicazione. Le referenze principali utilizzate per la prima parte del capitolo sono state Sutton & Barto (2018) e Ernst et al. (2005). Mentre per la spiegazione del modello di regressione GAM con spline le referenze principali sono James et al. (2013), Hastie et al. (2009) e Wasserman (2006).

2.1 Programmazione dinamica

Storicamente, la prima classe di metodi per la risoluzione di problemi decisionali a più stadi è stata la programmazione dinamica (DP). Il termine è stato inizialmente

utilizzato negli anni quaranta dal matematico Richard Bellman per descrivere il processo di risoluzione di problemi in cui sia necessario trovare le soluzioni migliori in modo sequenziale. La famiglia di algoritmi di programmazione dinamica (*Dynamic programming, DP*) è infatti composta da metodi che sono in grado di risolvere una vasta gamma di problemi sequenziali. Si tratta di algoritmi che vengono utilizzati per calcolare politiche ottimali quando si dispone di un modello noto per l'ambiente che viene rappresentato tramite un processo decisionale di Markov. Questi metodi richiedono quindi la conoscenza di un modello per la probabilità di transizione $p(s', r \mid s, a)$ ovvero si parla di algoritmi **model based**. L'idea chiave della DP è l'uso delle funzioni valore o valore-azione per la ricerca di buone politiche, una volta trovate le funzioni v^* o q^* che soddisfano le equazioni di Bellman si può trovare la policy ottimale.

Gli algoritmi classici di programmazione dinamica hanno utilità limitata nel reinforcement learning, sia per l'assunzione di un modello noto dell'ambiente, che per il notevole costo computazionale. Tuttavia, restano importanti dal punto di vista teorico poiché forniscono una base per comprendere metodi più complessi come il Q-learning e il fitted Q-iteration in cui non viene fatta nessuna assunzione sul modello dell'ambiente, ovvero si parla di metodi **model-free**. Nella programmazione dinamica solitamente si assume che l'ambiente sia un processo decisionale di Markov finito, ovvero si assume che lo stazio degli stati S, delle azioni A e delle reward R sia finito, ma si può anche applicare nell'ambito di spazi continui; inoltre, si possono anche considerare problemi con diversi episodi. Sebbene i concetti della DP possano essere applicati anche a problemi con spazio degli stati e delle azione continui, soluzioni esatte sono possibili solo in casi speciali. Un modo comune per ottenere soluzioni approssimate in questi casi è discretizzare lo spazio degli stati e delle azioni, e quindi applicare metodi di DP per spazi finiti.

I metodi di programmazione dinamica risolvono i problemi dei processi decisionali di Markov mediante l'iterazione di due passi di svolgimento chiamati **policy evaluation** e **policy improvement** che fanno parte di un processo chiamato **Generalized Policy Iteration** (GPI).

2.2 Generalized Policy Iteration

Generalized policy iteration è un metodo che si utilizza nella programmazione dinamica e in generale negli algoritmi di reinforcement learning per poter calcolare la policy ottima. Esso coinvolge due passi principali.

- Il primo passo è quello di **policy evaluation** ovvero quello di calcolare la funzione valore v_π per un'arbitraria policy π. In questo caso si utilizzano le equazioni di Bellman come regola di aggiornamento per calcolare la value function. Partendo da un'approssimazione iniziale della funzione valore v_0 per ogni stato $s \in \mathcal{S}$, spazio degli stati finito, la value function viene calcolata iterativamente seguendo la seguente equazione:

$$v_{t+1} = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_t(s')] \tag{2.1}$$

  In altre parole questo passo dell'algoritmo calcola la funzione valore per ogni stato e itera più volte questo processo in modo da migliorare la funzione valore per tutti gli stati. L'algoritmo arriva a convergenza soltanto quando $t \to \infty$, ma solitamente si definisce una regola di convergenza, ovvero viene interrotto quando il nuovo valore v_{t+1} è molto simile al valore precedente v_t. In particolare quando $\max_{s \in S} |v_{t+1}(s) - v_t(s)| \leq \varepsilon$, dove $\varepsilon \geq 0$ è un valore scelto a priori piccolo.

- Il secondo passo è quello di **policy improvement** in cui sulla base del valore di v_{t+1}, calcolato al passo precedente, si calcola il valore della funzione valore-azione q_π che viene poi massimizzato rispetto all'azione per poi aggiornare la policy π.

$$q_\pi(a, s) = \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_{t+1}(s')] \tag{2.2}$$

  A questo punto se $q_\pi(a, s)$ è maggiore del valore v_{t+1} determinato secondo la policy π si aggiorna la politica secondo la seguente formula:

$$\pi'(s) = \arg\max_{a \in A} q_\pi(a, s) \tag{2.3}$$

  Questo secondo passaggio ha quindi il compito di modificare la policy, in modo *greedy*, ossia cerca di ottenere una soluzione ottima globale ovvero la policy ottimale π^*, attraverso scelte ottime locali.

Questi due passi dell'algoritmo di GPI interagiscono tra di loro, come rappresentato in Figura 2.1, nel senso che, nel primo passo si calcola una funzione valore che influenza il secondo passo nel calcolo del miglioramento della politica, che a sua volta influenza la stima della funzione valore all'iterazione successiva. I due passi si stabilizzano quando le iterazioni successive terminano con risultati equivalenti, o quasi, alle precedenti iterazioni e a questo punto la policy e la funzione valore

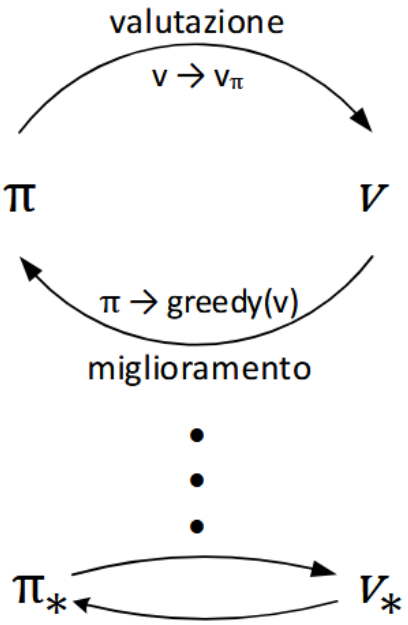

Figura 2.1: Algoritmo Generalized Policy Iteration (Sutton & Barto, 2018).

ottenute saranno quelle ottime. Questo perché la value function si stabilizzerà solo quando sarà coerente con la policy, e la policy si stabilizzerà solo quando avrà un comportamento che segue la value function corrente. Se la policy variasse il comportamento rispetto alla nuova value function, allora di conseguenza anche la value function sarebbe modificata nella iterazione seguente. Per questo motivo, se la value function e la policy si stabilizzano, saranno sia ottime, che coerenti l'una con l'altra.

Anche se la programmazione dinamica può non essere utilizzata per problemi con numerosità molto elevata, se confrontata con altri metodi per risolvere i processi decisionali di Markov, in realtà risulta piuttosto efficiente. Nel caso peggiore, il tempo di esecuzione per trovare una politica ottimale usando DP è una funzione polinomiale rispetto al numero di stati e azioni. Se si indica con n il numero di stati e con k il numero di azioni, con un metodo DP si può individuare una politica ottimale in un tempo ragionevole, anche se il numero totale di politiche possibili è k^n. In questo senso, DP è esponenzialmente più veloce rispetto a qualsiasi metodo che cerchi direttamente nello spazio delle politiche, perché una ricerca diretta dovrebbe esaminare tutte le politiche possibili per offrire la stessa garanzia.

2.3 Temporal-Difference learning

L'idea centrale del reinforcement learning è quella di apprendere in modo autonomo, ovvero l'agente deve essere in grado di scegliere l'azione ottimale ad ogni passo in base alla sua esperienza passata, senza che ci sia un modello già impostato che indichi quale sia l'azione corretta da compiere in quel preciso tempo e stato. Questa idea sta alla base del **Temporal-Difference learning (TD)**. I metodi TD apprendono direttamente dall'esperienza dell'algoritmo, senza bisogno di un modello noto dell'ambiente, come invece accade nei metodi di programmazione dinamica. Tra i due metodi presentati ci sono alcune similitudini e alcune differenze, ad esempio, come nei metodi DP anche negli algoritmi temporal-difference le stime delle funzioni valore vengono calcolate sulla base di altre stime appena apprese, senza aspettare l'esito finale (questo processo viene chiamato *bootstrap*). Anche i metodi TD seguono l'idea definita dalla Generalized policy iteration per il calcolo della policy ottimale. Infatti, data una certa esperienza acquisita seguendo una politica π, i metodi TD aggiornano la stima della funzione valore v_π per gli stati non terminali, ovvero per tutti gli stati che non si trovano alla fine di un episodio. Una caratteristica fondamentale dei metodi TD è che aggiornano gli *expected return*, ovvero la reward attesa, ad ogni step; infatti, allo step $t+1$ sono già in grado di aggiornare il valore nuovo target osservando la reward R_{t+1} e la stima della funzione valore $V(S_{t+1})$. Nel caso più semplice e generale, la logica di aggiornamento dei valori è definita come:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(s_t)\right] \tag{2.4}$$

dove il parametro α è compreso tra 0 e 1, rappresenta l'indice di aggiornamento del valore dello stato S_t rispetto alla nuova stima. Dalla formula si nota che il calcolo del valore di uno stato dipende dalla reward del passo successivo e dal valore dello stato sempre al passo successivo ma ridotto di un paramento di sconto γ.

I metodi temporal-difference sono molto simili ad altri metodi, chiamati metodi di Monte Carlo (MC), in cui non si assume nessun modello per l'ambiente, ma al posto di aggiornare i valori dello stato ad ogni passo, essi vengono aggiornati alla fine di ogni episodio poiché si aspetta quando le reward cumulate sono note, ovvero quando è noto G_t.

Questi algoritmi per scegliere l'azione ottimale e stimare la funzione valore sfruttano il trade-off tra **exploration** e **exploitation**. La fase di exploration consiste nell'esplorazione di nuove soluzioni nello spazio di ricerca, pur non avendo la garanzia

che siano soluzioni ottimali, quindi si tratta della ricerca di nuove azioni possibili da adottare. Mentre la fase di exploitation (o sfruttamento delle informazioni già apprese) consiste nello sfruttare le informazioni già esistenti, ovvero le azioni già utilizzate, e concentrarsi sulle soluzioni più promettenti. Se ci si concentra troppo sulla fase di esplorazione si rischia di finire in zone dello spazio di ricerca non ottimali, mentre se si sviluppa troppo la fase di exploitation si rischia di aumentare di molto i costi computazionali e di rallentare il raggiungimento della soluzione ottimale.

2.3.1 Vantaggi dei metodi temporal-difference

Come già spiegato precedentemente, i metodi TD aggiornano le loro stime basandosi, in parte, su altre stime, come avviene nei metodi di programmazione dinamica. A differenza di questi ultimi, però, non richiedono un modello noto per l'ambiente, né la distribuzione di probabilità degli stati successivi. Un altro vantaggio, rispetto ai metodi di Monte Carlo, è che i metodi TD sono in grado di calcolare la reward ad ogni passo temporale, mentre con i metodi MC bisogna aspettare la fine dell'episodio per conoscere le rewards. Questa differenza è molto importante poiché in alcune applicazioni gli episodi sono molto lunghi e quindi rimandare tutto l'apprendimento dell'algoritmo alla fine dell'episodio risulta troppo lento.

Un ulteriore vantaggio dei metodi TD è il fatto che garantiscono la convergenza, ovvero per qualsiasi policy fissata π, il metodo TD è dimostrato convergere a v_π.

2.3.2 Metodi TD on-policy e off-policy

I metodi TD si distinguono in due classi principali: i metodi **on-policy** in cui la policy che genera i dati equivale con la policy ottimizzata e quelli **off-policy** in cui la policy ottimizzata differisce dalla policy che genera i dati.

In seguito viene definito, in sintesi, l'algoritmo *SARSA* come metodo on-policy per dare un'idea generale di come funziona, di seguito verrà spiegato il metodo off-policy Q-learning che si tratta dell'algoritmo base su cui verrà poi sviluppata l'applicazione.

Nei metodi **on-policy**, ma come verrà specificato di seguito questo vale anche per i metodi off-policy, la funzione valore da stimare è $q(a, s)$, poiché il valore di uno stato $v(s)$ non è sufficiente in assenza di un modello dell'ambiente per permettere alla policy di determinare, dato uno stato, quale sia l'azione migliore da eseguire. Si utilizza la seguente equazione, che deriva da 2.4, per aggiornare i valori Q se lo stato

S_t non è uno stato terminale:

$$Q(A_t, S_t) = Q(A_t, S_t) + \alpha \left[R_{t+1} + \gamma Q(A_{t+1}, S_{t+1}) - Q(A_t, S_t)\right] \tag{2.5}$$

se, invece, lo stato S_{t+1} è terminale allora si impone $Q(A_{t+1}, S_{t+1}) = 0$. Essendo di natura on-policy, SARSA stima q_π sulla base dei comportamenti della politica π, e allo stesso tempo modifica il comportamento della policy rispetto alle stime aggiornate da q_π. La convergenza di SARSA, e più in generale di tutti i metodi TD, dipende dalla natura della politica. Nei metodi TD, vengono utilizzate politiche ε-greedy dove ε, ad ogni step viene decrementata.

Per quanto riguarda, invece, i metodi **off-policy** ci si concentra sull'algoritmo **Q-learning** da cui verrà poi sviluppato l'algoritmo utilizzato nell'applicazione di questa tesi: il **fitted Q-iteration** spiegato dettagliatamente nel prossimo paragrafo 2.4.

Il **Q-Learning** è un metodo di apprendimento Temporal Difference che permette ad un agente di apprendere il comportamento ottimale in un processo decisionale di Markov. Il Q-Learning stima la funzione valore $Q(s, a)$ in modo incrementale, aggiornando ad ogni passo temporale il valore della coppia stato-azione, seguendo la logica di aggiornamento della formula generale di stima dei valori per i metodi TD, ovvero seguendo l'equazione di Bellman. L'algoritmo Q-learning è un metodo off-policy, ovvero, a differenza dei metodi on-policy che utilizzano una policy secondaria per aggiornare le stime, l'azione che viene scelta come ottima è sempre quella che massimizza il valore $\max_a Q(s, a)$, la policy π viene utilizzata per determinare le coppie stato-azione da visitare e aggiornare, ma non interviene nella scelta dell'azione ottima. Ciò significa che l'agente impara la politica ottimale indipendentemente dalla policy comportamentale osservata. I valori di $Q(s, a)$ vengono stimati nel seguente modo:

$$Q(A_t, S_t) = Q(A_t, S_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(a, S_{t+1}) - Q(A_t, S_t)\right] \tag{2.6}$$

dove $\alpha \in (0, 1]$ rappresenta il tasso di apprendimento che controlla l'aggiornamento dei valori Q e $\gamma \in [0, 1]$ rappresenta il fattore di sconto, il quale permette di dare più o meno peso alle rewards ottenute in futuro, come già descritto nei metodi TD in generale. Si tratta di un algoritmo di tipo tabellare, ovvero richiede la memorizzazione esplicita di $Q(s, a)$ per ogni coppia stato-azione. Nella pratica si costruisce una tabella con tutte le possibili combinazioni tra gli stati e le azioni.

Per questo motivo il metodo Q-learning si utilizza soltanto quando lo spazio degli stati e quello delle azioni è discreto e non troppo ampio. Si tratta, inoltre, di un algoritmo **online**, ciò significa che aggiorna le Q-function passo dopo passo durante l'esplorazione ed è adatto per ambienti piccoli e discreti o quando si può interagire direttamente con l'ambiente.

Di seguito viene presentato uno *pseudo-codice* dell'algoritmo di Q-learning in modo da visualizzare passo per passo la procedura.

Algorithm 1 Q-Learning (tabellare)

Input : $\alpha \in (0, 1]$ (learning rate);
$\gamma \in [0, 1]$ (fattore di sconto);
Episodi (numero massimo di episodi)
Output : Tabella $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ (ottimale o approssimata)

Inizializzare $Q(s, a)$ arbitrariamente per tutti $s \in \mathcal{S}$, $a \in \mathcal{A}$, e $Q(s, a) = 0$ per gli stati terminali;
for *episodio* $\leftarrow 1$ **to** *Episodi* **do**
  Inizializzare lo stato s:
  while s *non è terminale* **do**
    Scegliere un'azione a da s usando una politica ϵ-greedy basata su Q.
    Eseguire l'azione a e si osserva la reward r e il nuovo stato s'.
    Aggiornare i valori Q con:
    $Q(s, a) \leftarrow Q(s, a) + \alpha\big[\, r + \gamma \max_a Q(s', a) - Q(s, a) \,\big]$
    Passare allo stato successivo: $s \leftarrow s'$
return Q

Per prima cosa si scelgono i valori dei parametri α e γ e si sceglie un numero massimo di episodi possibili. Si inizializzano i valori $Q(s, a)$ arbitrariamente per ogni stato e azione, in particolare quelli riferiti agli stati terminali (ovvero quelli a fine dell'episodio) si inizializzano pari a 0. Per ogni episodio si inizializza lo stato s e finché non ci si trova in uno stato terminale, si sceglie per ogni stato un'azione in base alla politica π, si esegue l'azione e si osserva lo stato successivo e la reward. A questo punto si aggiornano i valori $Q(s, a)$ e si passa allo stato successivo.

2.4 Fitted Q-iteration (FQI)

Il fitted q-iteration è un metodo di apprendimento automatico che fa parte del reinforcement learning, in particolare si può identificare come una estensione degli algoritmi Q-learning. Come già definito in precedenza, nei metodi di reinforcement learning un agente opera in un tempi discreti, osservando ad ogni passo temporale lo

stato S_t dell'ambiente, compiendo un'azione S_t, ricevendo una ricompensa istantanea dall'ambiente R_t e spostandosi nello stato successivo S_{t+1} o S'. Dopo un certo tempo finito, l'esperienza raccolta dall'agente può essere rappresentata attraverso una un'insieme di quattro elementi: (S_t, A_t, R_t, S_{t+1}).

Nei metodi online, esiste una politica esplorativa per collezionare i dati che permette di valutare azioni differenti per diminuire l'incertezza nella stima della funzione Q. Nei metodi online, ulteriormente, si hanno a disposizione potenzialmente infiniti dati che permettono quindi, con maggiore facilità, il raggiungimento di una politca ottimale. Nel metodo fitted Q-iteration si considera, invece, un apprendimento in modalità *batch*, ovvero si tratta di un **metodo offline**, in cui l'agente apprende sulla base di un dataset pre-esistente, ricevendo quindi un insieme di dati, invece di interagire direttamente in modo online con l'ambiente. Sulla base dei dati si è in grado di determinare una politica di controllo il più possibile vicino a quella ottimale. Partendo dal dataset ricevuto l'agente calcola un'approssimazione della Q-function definita sulla coppia stato-azione e deriva da essa la politica di controllo.

Quando gli spazi degli stati e delle azioni sono finiti e sufficientemente piccoli, la Q-function può essere rappresentata in forma tabellare, e la sua approssimazione (sia in modalità batch che on-line), così come la derivazione della politica di controllo, risultano semplici. Tuttavia, quando si ha a che fare con spazi di stati e azioni continui o molto grandi e discreti, la Q-function non può più essere rappresentata da una tabella con una voce per ogni coppia stato-azione.

Per superare questo problema è stato costruito l'algoritmo di fitted Q-iteration che è in grado di gestire spazi continui e grandi, ovvero con molti stati e azioni possibili, grazie all'uso di approssimatori parametrici o non (come la regressione lineare nel caso più semplice, il random forest o le reti neurali quando la relazione tra le variabili input e la target non è lineare). L'algoritmo di fitted Q-iteration è, quindi, definito come un algoritmo di reinforcement learning in modalità batch che fornisce un'approssimazione della Q-function utilizzando un approccio iterativo per migliorarne la stima, risolvendo ad ogni passo problemi di regressione sequenziali. Il fitted Q-iteration funziona nel seguente modo:

- Alla prima iterazione calcola un'approssimazione della Q-function con un passo di ottimizzazione. Ovvero la stima Q al primo passo è data dal valore atteso condizionato della reward istantanea data la coppia stato-azione: $q_1(s, a) = \mathbb{E}[R_t \mid S_t = s, A_t = a]$. Questa approssimazione può essere calcolata applicando un metodo di regressione al dataset disponibile i cui input sono

l'insieme di coppie stato-azione e gli output sono le reward immediate r_t, ovvero alla prima iterazione si ottene che $q_{1,t} = r_t$.

- All'iterazione n-esima l'approssimazione della Q-function corrisponde a n passi di ottimizzazione, in cui ad ogni passo l'algoritmo di regressione utilizzato stima i nuovi valori di output e li aggiorna ad ogni passo utilizzando la *value iteration*, ovvero l'equazione di Bellman: $q_{n,t} = r_t + \gamma \max_a Q_{n-1}(s', a)$ dove γ è il fattore di sconto.

In Figura 2 viene spiegato e presentato in dettaglio l'algoritmo di fitted Q-iteration, utilizzato anche nell'applicazione di questa tesi. L'algoritmo FQI illustrato si basa su un dataset composto da diversi episodi e, ogni episodio può avere un diverso numero di istanti temporali T_e. Questi episodi vengono considerati indipendenti gli uni dagli altri.

Algorithm 2 Fitted Q-Iteration (FQI)

Input : $\mathcal{D} = \{(s_{e,t}, a_{e,t}, r_{e,t}, s'_{e,t})\}_{e=1}^{E},\ t = 1, \ldots, T_e$ dataset con E episodi e T_e passi per episodio
$\gamma \in [0, 1]$ fattore di sconto
K numero di iterazioni
$\mathcal{F}$ classe di funzioni di regressione (es. regressione lineare, rete neurale,...)

Output : Funzione approssimata $Q_K(s, a)$

Inizializzare $Q_0(s, a) \leftarrow 0$ (oppure un altro valore iniziale)
for $k \leftarrow 1$ **to** K **do**

Costruire il dataset di training $\mathcal{T}_k = \left\{\left((s_{e,t}, a_{e,t}),\ y^k_{e,t}\right)\right\}_{e=1}^{E},\ t = 1, \ldots, T_e$, dove:

$$y^k_{e,t} = r_{e,t} + \gamma \max_a Q_{k-1}(s'_{e,t}, a)$$

Allenare un modello di regressione minimizzando l'errore quadratico:

$$Q_k \leftarrow \arg\min_{f \in \mathcal{F}} \sum_{e=1}^{E} \sum_{t=1}^{T_e} \left(f(s_{e,t}, a_{e,t}) - y^k_{e,t}\right)^2$$

return $\hat{Q}_K(s, a)$

L'algoritmo FQI richiede 4 elementi di input, il valore del fattore di sconto γ, il numero di iterazioni K, una classe di modelli di regressione $\mathcal{F}$ ed infine, poiché l'algoritmo lavora in modalità batch offline, il dataset osservato, composto dallo stato s, dall'azione a, dallo stato successivo s' e dalla reward osservata r. Queste 4 componenti vengono osservate per ogni passo temporale, ovvero per $t = 1, ..., T_e$ e per

ogni episodio considerato $e = 1, ..., E$. L'algoritmo viene inizializzato impostando il valore Q pari a 0 per ogni coppia di stato-azione. Si crea successivamente un ciclo che si ripete per K iterazioni, in cui ad ogni passo si calcola la nuova variabile risposta indicata con $y_{e,t}^k$ che è data dalla somma della reward osservata (dato un instante temporale e un episodio) e dal valore Q che viene massimizzato secondo l'azione ottima. L'azione ottima selezionata ad ogni passo tra tutte le azioni disponibili nello spazio delle azioni è, appunto, quella che massimizza il valore Q. Una volta calcolato il nuovo valore target si utilizza un modello di regressione per la previsione dei valori target che minimizza l'errore quadratico.

In questo caso come regola di *stopping* dell'algoritmo è stato impostato il numero massimo di iterazioni, ma esistono diverse regole che permettono all'algoritmo di fermarsi senza definire a priori il numero di iterazioni, poiché in questo caso può esserci il rischio di non raggiungere la convergenza.

2.4.1 Condizioni di arresto e convergenza dell'algoritmo FQI

L'algoritmo FQI ha bisogno di condizioni impostate a priori per decidere a quale iterazione fermarsi, ovvero per quale valore di K il processo può essere interrotto. Un metodo semplice per interrompere l'algoritmo è scegliere direttamente il numero massimo di iterazioni. Per scegliere a priori il numero di iterazioni massimo si può utilizzare la seguente disuguaglianza e risolverla rispetto a K:

$$\|Q_{\pi_K^*} - Q_{\pi_\infty^*}\|_\infty \leq \frac{2\gamma^K B_r}{(1-\gamma)^2} \tag{2.7}$$

La parte sul lato sinistro della disuguaglianza, $|Q_{\pi_N^*} - Q_{\pi_\infty^*}\|_\infty$, rappresenta l'errore di sub-ottimalità, ovvero rappresenta quanto la politica dopo K iterazioni si discosta dalla politica ottimale. $Q_{\pi_K^*}$ è valore della funzione Q seguendo la politica ottimale ottenuta dopo K iterazioni, mentre il valore $Q_{\pi_\infty^*}$ rappresenta il valore della funzione Q ottenuto utilizzando la politica ottimale vera, ottenuta dopo infinite iterazioni. La norma infinito è il massimo errore su tutti gli stati. Solitamente questa differenza non è possibile calcolarla, per cui si sostituisce con un valore ϵ piccolo, ad esempio $\epsilon = 0.001$. La parte sul lato destro, invece, rappresenta il limite superiore dell'errore. $\gamma \in [0, 1]$ è il fattore di sconto che informa quanto valgono le rewards future rispetto a quelle immediate, B_r è il massimo valore assoluto che può avere una reward e K, come già detto, è il numero di iterazioni da impostare. Al crescere di K, γ^K tende a 0, ovvero l'errore diminuisce esponenzialmente. Se si imposta, quindi, un valore

di ϵ piccolo, si può definire il numero minimo di iterazioni K che garantisce che la politica ottenuta abbia un errore inferiore o uguale a ϵ.

Un'ulteriore possibilità è quella di interrompere il processo iterativo quando la distanza tra Q_K e Q_{K-1} si trova al di sotto di una certa soglia. Il problema sta nel fatto che per alcuni algoritmi di apprendimento supervisionato non vi è alcuna garanzia che la sequenza delle funzioni Q_K converga effettivamente, e quindi, questo tipo di criterio di stop potrebbe non essere utile nella pratica. L'obiettivo dell'algoritmo è, quindi, quello di individuare una policy ottimale, ovvero la target policy che si avvicini maggiormente alla vera policy ottimale. In altre parole l'output di questo algoritmo è il valore della funzione Q_K approssimata che deve convergere al valore della funzione ottima vera Q^*.

$$\begin{aligned} Q_k &: S \times A \to \mathbb{R} \quad \text{(funzione Q stimata con } k = 1, \ldots, K) \\ \|Q_K - Q^*\|_\infty &\to 0 \quad \text{(convergenza in norma } \infty) \end{aligned} \tag{2.8}$$

La convergenza (rappresentata in 2.8) del metodo FQI non è garantita in generale, ma dipende da alcune ipotesi chiave. Il dataset deve coprire adeguatamente lo spazio degli stati e delle azioni, ovvero il campionamento deve essere eseguito in modo da osservare un range di dati che permette poi ai modelli di essere in grado di generalizzare. Gli algoritmi di supervised learning utilizzati (ad esempio reti neurali, random forest o modelli additivi) devono essere in grado di rappresentare e stimare adeguatamente la vera funzione Q^*.

Dato un dataset D composto da input $i = (s, a, s', r)$ e output $o =$ *target values calcolati ad ogni iterazione*, il metodo di apprendimento supervisionato deve creare un modello del tipo:

$$f(i) = \sum_{l=1}^{\#D} k(i_l, i) \cdot o_l \tag{2.9}$$

dove $\#D$ rappresenta la cardinalità del dataset D, ovvero il numero di osservazioni, e $k(i_l, i)$ rappresenta il kernel di transizione che deve soddisfare il fatto di cambiare poco tra un'iterazione e l'altra. Inoltre, il kernel definito deve soddisfare l'Equazione 2.10 in modo da assicurare che la funzione Q venga aggiornata in modo "controllato", come una media pesata dei valori target.

$$\sum_{l=1}^{\#D} |k(i_l, i)| = 1, \quad \forall i \tag{2.10}$$

2.5 Modello di regressione per l'algoritmo FQI

In questa sezione viene presentato il modello additivo con regressione spline che viene utilizzato nell'applicazione all'interno dell'algoritmo fitted Q-iteration.

Quando ci si trova in un problema di regressione l'obiettivo è quello di prevedere nel modo migliore possibile la variabile risposta, allenando un modello che si basa su informazioni contenute nelle variabili indipendenti. Un modello di regressione può essere di molti tipi; in questa sezione ci si concentra su un modello additivo, in cui a ciascuna covariata disponibile viene applicata una spline. Si parla quindi di **regressione spline**.

Prima di introdurre il concetto di spline, è utile ricordare che, nella maggior parte dei contesti applicativi, la relazione tra le variabili indipendenti e quella dipendente non è lineare. In questi casi si ricorre alla regressione polinomiale, che consiste nell'applicare una trasformazione polinomiale alla covariata che presenta una relazione non lineare con la risposta. Nella regressione polinomiale si può considerare anche la regressione lineare poiché essa approssima l'andamento della variabile indipendente tramite una retta, ovvero un polinomio di primo grado. Tuttavia, nella maggior parte dei casi, una retta non è in grado di catturare a sufficienza l'andamento della variabile indipendente per prevedere in modo soddisfacente la risposta; per questo motivo si ricorre spesso a polinomi di grado superiore.

La regressione polinomiale, però, presenta diversi svantaggi. Quando la relazione tra le covariate e la variabile dipendente non è lineare si ricorre talvolta alla regressione non parametrica dove la previsione in un punto x_i dipende principalmente dalle osservazioni vicine al punto x_i. Tuttavia, i polinomi non sono stimatori non parametrici: essi hanno natura globale e non locale. Questo significa che anche punti molto lontani da x_i possono influenzare la stima di $f(x_i)$, introducendo oscillazioni spurie e instabilità nelle previsioni. Inoltre, aumentando il grado del polinomio si rischia l'overfitting, ovvero il modello si adatta troppo bene ai dati del training ma non è in grado di generalizzare; fuori dall'intervallo di valori del training il polinomio diverge molto velocemente verso $\pm\infty$. In aggiunta la regressione polinomiale è molto sensibile agli outlier e quando il grado del polinomio d è elevato si creano problemi di instabilità numerica. In Figura 2.2 vengono rappresentati con diversi colori diversi gradi di un polinomio applicato ai dati che sono rappresentati dai puntini neri, in rosso è rappresentato un polinomio di primo grado, in verde di secondo grado, in arancione di terzo grado e in blu di quarto grado. Si nota come il polinomio di quarto grado è quello che è in grado maggiormente di interpolare i dati ma, come già detto,

questo non significa che sia il migliore nel rappresentare i dati, poiché non si conosce l'intera distribuzione di essi. Un modello, quindi, si deve adattare bene ai dati che si conoscono, ma deve essere anche in grado di generalizzare, ovvero adattarsi anche ai dati che non si conoscono.

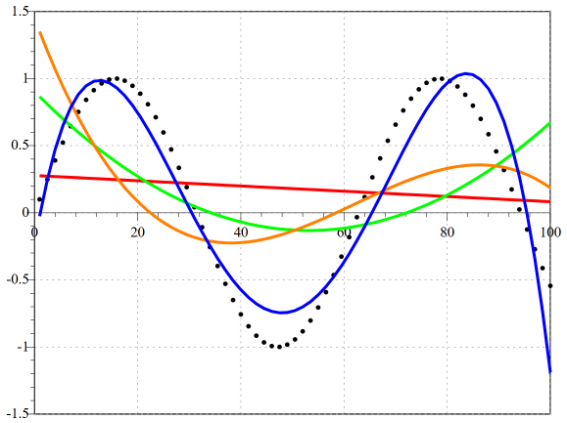

Figura 2.2: Regressione polinomiale con diversi gradi del polinomio (Wikipedia contributors, 2024a).

Nel tempo sono stati sviluppati ulteriori modelli che mantengono i vantaggi dei polinomi e cercano di eliminarne i problemi. Si parla di una generalizzazione della regressione polinomiale, detta *basis expansion*. Date $h_1(x), ..., h_p(x)$ funzioni prespecificate che trasformano il predittore x in una funzione non lineare e considerando di avere soltanto una covariata, si può definire il modello come:

$$y = f(x, \beta) + \varepsilon \tag{2.11}$$

in cui la funzione $f(x, \beta)$ viene esplicitata nel seguente modo:

$$f(x, \beta) = \sum_{j=1}^{p} h_j(x)\beta_j \tag{2.12}$$

Questo tipo di modello rimane lineare nei parametri β, come una regressione lineare classica, ma la variabile indipendente x entra nel modello in modo non lineare. Le funzioni $h_j(x)$ sono dette *piecewise polynomial function*, si tratta di polinomi solitamente di secondo o terzo grado che vengono applicati in ogni intervallo in cui è stato suddiviso il range di variazione della covariata.

Come mostrato in Figura 2.3 l'intervallo dei valori della covariata viene suddiviso in sotto-intervalli, che possono essere anche di ampiezza diversa, delimitati da nodi ed in ogni intervallo viene stimato un polinomio, in questo caso un polinomio

cubico. Le linee verticali tratteggiate indicano le posizioni dei due nodi ξ_1 e ξ_2, la curva blu rappresenta la funzione reale, dalla quale i dati sono stati generati e le linee verdi rappresentano le funzioni polinomiali cubiche stimate all'interno di ogni intervallo. Nel quadrante in alto a sinistra si presenta il problema della discontinuità in corrispondenza di ciascun nodo, ovvero le $h_j(x)$ funzioni non sono continue nei nodi ξ_j. Nel quadrante in alto a destra viene inserito il vincolo di continuità delle funzioni cubiche considerate, nel quadrante in basso a destra si impone il vincolo di continuità anche per la derivata prima delle funzioni $h_j(x)$ e infine nell'ultimo quadrante viene inserito anche il vincolo di continuità per la derivata seconda, ottenendo quindi una curva continua in ogni punto. Quest'ultima curva rappresenta proprio la **regressione spline** che aggiunge, appunto, vincoli di continuità dalla piecewise regression.

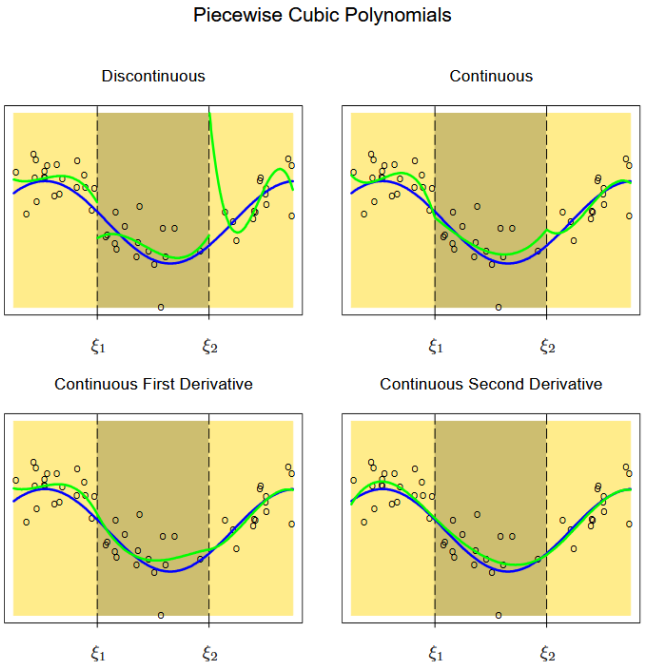

Figura 2.3: Esempio di Piecewise polynomial regression cubica, con ordini crescenti di continuità (Hastie et al., 2009).

2.5.1 Regressione Spline

La regressione tramite le spline rappresenta un metodo di regressione **semi-parametrico** che utilizza polinomi stimati in sotto-intervalli della covariata, come nel caso delle funzioni piecewise, ma impone vincoli di continuità in ciascun nodo e permette di controllare la complessità del modello, introducendo un effetto di regolarizzazione e garantendo maggiore stabilità nella stima. Una spline è definita come una piecewise polynomial function con lisciamento e parametri di continuità.

Definizione 1. *Dati $\xi_1, \ldots, \xi_k$ un insieme di nodi appartenenti all'intervallo (a, b) dei valori della variabile x, una **spline** è una funzione $f(x, \beta) : (a, b) \to \mathbb{R}$ tale che in ciascun sotto-intervallo definito dai nodi, $f(x, \beta)$ è un polinomio di grado d e ha derivate continue fino all'ordine $d - 1$ su tutto l'intervallo (a, b).*

In altre parole, una spline di grado d è una funzione piecewise polynomial $f(x, \beta)$ di ordine d, tale che:

$$f(\xi_j^+) = f(\xi_j^-), ..., f^{(d-1)}(\xi_j^+) = f^{(d-1)}(\xi_j^-) \tag{2.13}$$

dove ξ_j^+ e ξ_j^- rappresentano il limite sinistro e destro dei nodi. Se $d = 1$ si parla di funzioni piecewise costanti, se $d = 2$ di funzioni piecewise quadratiche. Solitamente, però, si utilizzano spline cubiche, di grado $d = 3$, in cui in ogni nodo $\xi_j \in (a, b)$ si ha un punto di continuità delle funzioni, ovvero tale che $f(\xi_j^+) = f(\xi_j^-)$, inoltre anche la derivata prima e seconda delle funzioni devono soddisfare i punti di continuità, cioè: $f'(\xi_j^+) = f'(\xi_j^-)$ e $f''(\xi_j^+) = f''(\xi_j^-) \quad \forall j = 1, ..., k$.

Il concetto di spline ha origini pratiche, ben prima della sua formalizzazione matematica. Il termine deriva da una tecnica tradizionale utilizzata in ingegneria navale e nel disegno tecnico prima della sua utilità in ambito matematico statistico. Servivano per tracciare curve lisce che passassero attraverso un insieme di punti predefiniti, come mostrato in Figura 2.4. Per disegnare queste curve, gli artigiani utilizzavano una sottile striscia flessibile di legno o metallo, chiamata appunto spline, che veniva fissata in corrispondenza di punti predefiniti tramite dei pesi metallici, detti nodi. La naturale elasticità del materiale permetteva alla barra di piegarsi formando una curva continua e regolare. Negli anni successivi, con l'evoluzione della matematica applicata e dell'informatica, il comportamento fisico di queste aste flessibili venne descritto in termini matematici. A partire dagli anni '40 e '50, le spline cominciarono a essere studiate formalmente come funzioni polinomiali a tratti, con condizioni di continuità e derivabilità nei nodi. Un contributo fondamentale in questa direzione fu dato dal matematico Isaac Schoenberg, che nel 1946 introdusse le spline polinomiali, in particolare le spline cubiche, che divennero rapidamente uno strumento standard nell'analisi matematica e statistica.

La definizione di spline, come descritta precedentemente, è abbastanza astratta e non applicabile in un modello di regressione, viene quindi definito un teorema fondamentale nella definizione di **regression spline**.

Teorema 2.1 (Truncated power basis)**.** *Siano $\xi_1 < \xi_2 < \cdots < \xi_k$ un insieme di punti ordinati, detti nodi, appartenenti all'intervallo aperto (a, b). Si definiscono le*

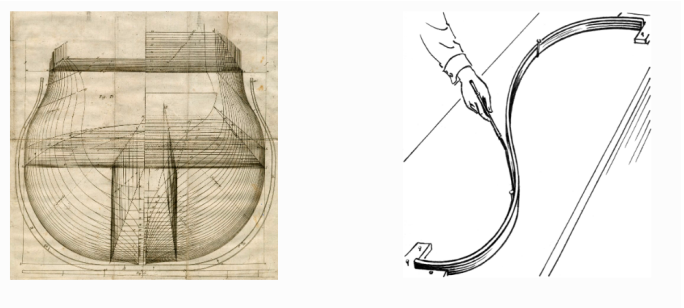

Figura 2.4: Invenzione delle spline in ambito navale (Wikipedia contributors, 2024b).

seguenti funzioni:

$$h_j(x) = x^{j-1}, \quad \textit{per } j = 1, \ldots, d+1,$$

ovvero le prime $d+1$ *basi sono polinomi, mentre dopo* $d+1$ *si utilizzano funzioni cubiche, che garantiscono la continuità nei nodi per definizione:*

$$h_{j+d+1}(x) = (x - \xi_j)_+^d, \quad \textit{per } j = 1, \ldots, k,$$

dove $(x - \xi_j)_+^d$ *denota la truncated basis, cioè:*

$$(x - \xi_j)_+^d = \begin{cases} (x - \xi_j)^d & \textit{se } x > \xi_j, \\ 0 & \textit{altrimenti.} \end{cases}$$

Allora, l'insieme di funzioni $\{h_1, h_2, \ldots, h_{k+d+1}\}$ *costituisce una base per lo spazio delle spline di grado* d *con nodi* $\xi_1, \ldots, \xi_k$*, nota come truncated power basis.*

Di conseguenza, ogni spline di grado d *con tali nodi può essere scritta come una combinazione lineare delle funzioni di base:*

$$f(x; \boldsymbol{\beta}) = \sum_{j=1}^{k+d+1} h_j(x)\, \beta_j,$$

dove $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{k+d+1})^\top$ *è il vettore dei coefficienti, con* $k + d + 1 = p$*.*

Il teorema appena enunciato impone automaticamente i vincoli di continuità delle funzioni spline e in questo modo si può semplicemente stimare la regressione tramite i minimi quadrati. Si definisce $\boldsymbol{B}$ la matrice $n \times p$ del disegno i cui elementi sono ottenuti tramite le truncated basis: $[\boldsymbol{B}]_{i,j} = h_j(x_i)$ con $j = 1, ..., p$ e $i = 1, ..., n$. Quindi, data la funzione spline $f(x, \beta) = \sum_{j=1}^{p} h_j(x)\beta_j$ si applicano i minimi quadrati

per calcolare il valore di $\hat{\beta}$ stimato:

$$\hat{\beta} = (B^T B)^{-1} B^T y \Rightarrow \ \hat{f}(x) = \sum_{j=1}^{p} h_j(x)\, \hat{\beta}_j = \sum_{i=1}^{n} s_i(x)\, y_i \tag{2.14}$$

Pertanto, le spline sono un esempio di *linear smoother* (in realtà del *modello lineare*). In questo caso, la **matrice di smoothing** è $S = B(B^T B)^{-1} B^T$,, la cui traccia è $\mathrm{tr}(S) = p$, che rappresenta il numero di gradi di libertà.

Solitamente un parametro da scegliere e ottimizzare nella scelta delle spline è il numero di nodi k in cui dividere l'intervallo (a, b) e la loro posizione; si tratta di parametri di complessità che solitamente vengono scelti tramite *cross-validation*. Per la posizione dei nodi una scelta comune è quella di posizionarli in modo da essere separati equamente nell'intervallo (a, b), oppure posizionarli sui quantili della distribuzione di x. Il numero di nodi influenza il numero di gradi di libertà data la relazione $p = k + d + 1$. La spline presentata finora è definita come spline classica, ma esistono diversi tipi: come le *Natural spline*, le *Smoothing spline* e le *Thin-plate spline*.

2.5.2 Natural spline

Una **Natural cubic spline** è una spline classica cubica $f(x, \beta)$ che assume un andamento lineare sui lati. Ovvero, prima del primo nodo tra a e ξ_1 e dopo l'ultimo nodo tra ξ_k e b, non è più un polinomio di grado $d = 3$, ma è una retta. In questo caso si avrà quindi che la derivata seconda della spline in corrispondenza del primo e dell'ultimo nodo sarà pari a 0: $f(\xi_1'') = f(\xi_k'') = 0$. Questo tipo di spline è utile poiché molte volte capita che vicino ai margini dell'intervallo (a, b) ci siano pochi dati e stimare con un polinomio questa parte sarebbe troppo complesso e flessibile, si utilizza una retta in modo da ridurre la varianza.

Proposizione 2.1. *Un insieme di $n \geq 2$ punti distinti (x_i, y_i) può essere interpolato utilizzando una spline cubica naturale, prendendo i punti $x_1 < \cdots < x_n$ come nodi. La spline cubica naturale interpolante è* ***unica****.*

Nella pratica, la *truncated power basis* può essere facilmente modificata per ottenere la seguente funzione:

$$N_1(x) = 1, \quad N_2(x) = x, \tag{2.15}$$
$$N_{j+2}(x) = \frac{(x-\xi_j)_+^3 - (x-\xi_k)_+^3}{\xi_k - \xi_j} - \frac{(x-\xi_{k-1})_+^3 - (x-\xi_k)_+^3}{\xi_k - \xi_{k-1}}, \quad j = 1, \ldots, k-2.$$

Questa formula rappresenta una *versione scalata* della *truncated power basis* per ogni $x \leq \xi_{k-1}$, ovvero:

$$N_{j+2}(x) = \frac{(x-\xi_j)_+^3}{\xi_k - \xi_j}, \quad x \leq \xi_{k-1}, \quad j = 1, \ldots, k-2. \tag{2.16}$$

Nelle *spline cubiche naturali* si ha $k = p$ e la funzione può essere espressa come:

$$f(x;\beta) = \sum_{j=1}^{k} N_j(x)\,\beta_j. \tag{2.17}$$

2.5.3 B-spline per l'implementazione

Sia per la versione classica che per quella naturale delle spline, l'implementazione in termini computazionali ha un problema fondamentale, ovvero la matrice $B^T B$ è *ill-conditioned*, cioè porta a instabilità numerica. La soluzione al problema sono le **B-spline** in cui si considerano $B_1(x), ..., B_p(x)$ funzioni che riparametrizzano le funzioni $h_j(x)$ e quindi le colonne della matrice del disegno B.

$$B_j(x) = \sum_{l=1}^{p} \gamma_{l,j} h_l(x), \quad j = 1, ..., p \tag{2.18}$$

dove $\gamma_{l,j}$ sono dei pesi. Si tratta di una trasformazione lineare, quindi le previsioni non cambiano.

2.5.4 Smoothing spline

Le **smoothing spline** sono un altro tipo di spline in cui il numero di nodi k non rappresenta più il parametro di complessità, poiché i nodi sono individuati tramite le osservazioni stesse. Nelle smoothing spline si considera la seguente funzione di perdita:

$$L(f;\lambda) = \sum_{i=1}^{n} \{y_i - f(x_i)\}^2 + \lambda \underbrace{\int_a^b \{f''(t)\}^2\, dt}_{\text{penalità}} \tag{2.19}$$

dove (a, b) è l'intervallo contenente i dati e $\lambda > 0$ è il parametro di *smoothing*. Si considera lo stimatore come quello che minimizza la funzione di perdita: $\hat{f}(x) = \arg\min_{f \in \mathcal{F}} L(f, \lambda)$. Il parametro λ di complessità, tramite la penalità, quantifica quanto "oscilla" la curva spline. Ci sono due estremi:

- con $\lambda = 0$, non si ha la penalità;
- con $\lambda = \infty$, necessariamente si ha che $f''(x) = 0$, allora la soluzione è il modello lineare.

Teorema 2.2 (Smoothing spline)**.** *Dati $n_0 \leq n$ distinti punti tra $x_1, ..., x_n$ con $x_i \in (a, b)$. Suppongo $n_0 \geq 3$, allora per ogni $\lambda > 0$ lo stimatore che minimizza $L(f, \lambda)$ è unico ed è una natural cubic spline con n_0 nodi in corrispondenza di ogni osservazione. Il numero dei nodi è uguale al numero di osservazioni.*

La smoothing spline è definita come una natural cubic spline ma con il numero e la posizione dei nodi già impostata:

$$f(x, \beta) = \sum_{j=1}^{n_0} N_j(x)\beta_j \tag{2.20}$$

Nelle smoothing spline la complessità non è controllata tramite il numero dei nodi k che è discreto, ma viene controllata tramite il parametro λ che è continui e pesa la penalizzazione.

Come negli altri tipi di spline, anche in questo caso si risolve come una regressione lineare, in particolare come la regressione *ridge*, ovvero i coefficienti β_j vengono stimati nel seguente modo:

$$\hat{\beta} = (N^T N + \lambda\Omega)^{-1} N^T y, \quad con \quad [\Omega]_{ij} = \int_a^b N_j''(t) N_k''(t) dt \tag{2.21}$$

dove $N_j''(t)$ e $N_k''(t)$ sono la derivata seconda della natural cubic spline. Anche le smoothing spline a livello operativo vengono implementate tramite le B-spline spiegate precedentemente.

2.5.5 Thin plate spline

Le **thin plate spline** (TPS) sono un tipo di smoothing spline utilizzato per la visualizzazione di relazioni complesse tra le variabili indipendenti continue e la variabile risposta. Le thin plate spline sono ideali per esaminare l'effetto combinato di due predittori continui su un singolo outcome, grazie al loro aspetto multidimensionale.

Invece di una singola curva, le thin plate splines sono rappresentate come una superficie flessibile; infatti, come si può intuire al nome, è come piegare una lamina di metallo fissandola in certi punti (nodi), la forma che assume minimizza la curvatura, quindi è "lisciata". TPS cerca proprio la funzione con curvatura minima che rispetta i vincoli dati. Come mostrato in Figura 2.5 ogni variabile continua è tracciata su un asse x distinto, creando una superficie bidimensionale. La risposta, invece, viene tracciata sull'asse y, in modo continuo, su quella superficie bivariata in tre dimensioni.

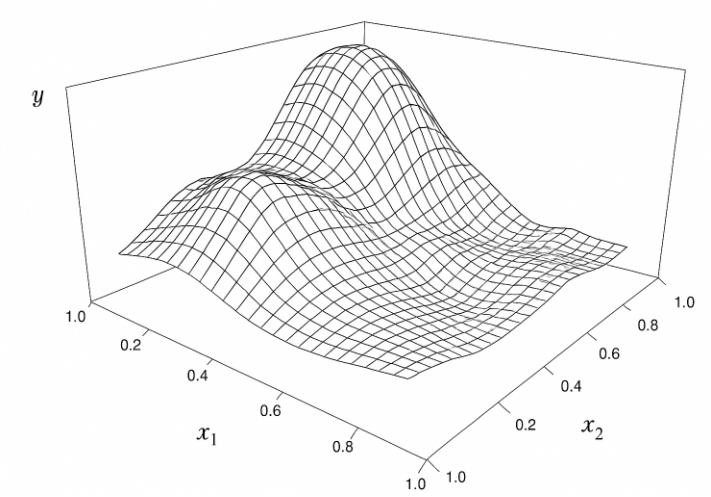

Figura 2.5: Thin plate spline (of Public Health, 2011).

Trattandosi di un tipo particolare di smoothing spline, esse si definiscono in modo simile. Infatti, come nelle smoothing spline, la funzione di perdita da minimizzare è caratterizzata da una componente di penalizzazione, controllata dal parametro λ, che ha la funzione di controllare anche la complessità della spline. La funzione di perdita è definita nel seguente modo:

$$L(f,\lambda) \;=\; \sum_{i=1}^{K} \|y_i - f(x_i)\|^2 + \lambda \iint \left[\left(\frac{\partial^2 f}{\partial x_1^2} \right)^2 + 2 \left(\frac{\partial^2 f}{\partial x_1 \, \partial x_2} \right)^2 + \left(\frac{\partial^2 f}{\partial x_2^2} \right)^2 \right] \mathrm{d}x_1 \, \mathrm{d}x_2 \tag{2.22}$$

dove la funzione spline è $f(x) = \sum_{i=1}^{N} w_i \, U(\|x - x_i\|)$ in cui w_i sono pesi da determinare e $U(r) \;=\; r^2 \log r$ è il kernel radiale della TPS. Il vantaggio principale delle thin plate spline, come per altre smoothing spline è che non richiedono conoscenza a priori della forma funzionale dei dati o della relazione da modellare. Inoltre, non c'è la necessità di specificare il numero e la posizione dei nodi, che può essere un problema con le spline cubiche, poiché ogni nodo è caratterizzato da ogni osservazione disponibile.

2.5.6 Modelli generalizzati additivi

Tutti i tipi di spline descritte presentano il problema della dimensionalità, nel senso che quando si passa a un problema con molte covariate e di conseguenza a più dimensioni, stimare con modelli semi o non parametrici diventa difficile. Il problema risiede nel fatto che si tratta di stime locali, mentre lo spazio multidimensionale è molto ampio; di conseguenza, la stima locale, che interpola solo un numero limitato di dati, finisce per assumere un carattere globale coprendo l'intero spazio, poiché i dati non risultano più sufficientemente vicini.

Un **modello additivo generalizzato** (GAM) è definibile come un'estensione del modello lineare generalizzato (GLM), dove la relazione tra i predittori e la risposta non è lineare, ma può essere espressa come somma di funzioni:

$$f(x_i) \;=\; \beta_0 + f_1(x_{i1}) + \cdots + f_p(x_{ip}) \;=\; \beta_0 + \sum_{j=1}^{p} f_j(x_{ij}), \qquad i = 1, \ldots, n \quad (2.23)$$

$f_1, ..., f_p$ sono funzioni non lineari univariate, esse possono essere delle spline di qualsiasi tipo. Per essere identificabile un modello additivo è necessario che ogni f_j sia centrata in 0: $\sum_{i=1}^{n} f_j(x_{ij}) = 0, \qquad j = 1, \ldots, p.$

Esistono diverse strategie per stimare le funzioni f_j, una di esse è l'algoritmo di **Back-fitting**. Si suppone che ogni f_j sia una spline del tipo $f_j(x) = \sum_{m=1}^{M_j} \beta_{mj} h_{mj}(x)$, l'algoritmo è il seguente:

Algorithm 3 Backfitting per Modelli Additivi

Input : Dati $\{(x_{i1}, \ldots, x_{ip}, y_i)\}_{i=1}^n$
Output : Stime $\hat{\beta}^{(0)}, \hat{f}_1, \ldots, \hat{f}_p$ delle componenti additive

Inizializzare: $\hat{\beta}^{(0)} = \bar{y}$, $\hat{f}_j(x_j) = 0$, per $j = 1, \ldots, p$

repeat
  for $j = 1$ **to** p **do**
    Calcolare i residui parziali:

$$r_{ij} = y_i - \hat{\beta}^{(0)} - \sum_{k \neq j} \hat{f}_k(x_{ik})$$

    Aggiornare la funzione f_j usando la smoothing spline S_j sui residui:

$$\hat{f}_j(x) \leftarrow S_j(\{x_{ij}, r_{ij}\}_{i=1}^n)$$

    Centrare la funzione:

$$\hat{f}_j(x) \leftarrow \hat{f}_j(x) - \frac{1}{n} \sum_{i=1}^{n} \hat{f}_j(x_{ij})$$

until *convergenza*
return $\hat{\beta}^{(0)}, \hat{f}_1, \ldots, \hat{f}_p$

In altre parole un modello additivo utilizzando degli stimatori come le spline, permette di utilizzare una stima semi o non parametrica anche in caso di elevata dimensionalità. Infatti nell'applicazione viene utilizzato un modello additivo su cui su ogni covariata viene applicata una regressione spline.

Nel Capitolo 4 viene descritta l'applicazione di questa tesi, in particolare l'applicazione dell'algoritmo FQI in cui è stato utilizzato un modello additivo (GAM) applicando diversi tipi di spline. In particolare è stato individuato come modello migliore, quello ottenuto utilizzando le thin plate spline.

Capitolo 3

Conformal Prediction

In questo capitolo viene introdotto un nuovo metodo statistico per la costruzione di intervalli di previsione nel contesto degli algoritmi di apprendimento *off-policy*. In particolare, viene analizzato un approccio che consente di quantificare e valutare l'incertezza delle previsioni generate dall'algoritmo di *Fitted Q-Iteration* (FQI). L'obiettivo è fornire non solo una stima puntuale della funzione valore-azione Q, ma anche una misura dell'affidabilità associata a tali stime, rendendo così l'algoritmo più robusto e interpretabile in scenari decisionali. La quantificazione dell'incertezza rappresenta un aspetto centrale in ambito statistico, specialmente in contesti ad alta criticità (come la medicina, la finanza, la guida autonoma), dove è importante fornire anche informazioni sulla qualità delle stime prodotte, così da poter prendere decisioni tenendo conto non solo del valore stimato in modo puntuale, ma anche della sua precisione.

Nei modelli statistici classici, come la regressione lineare, assumendo una specifica distribuzione per la variabile risposta, risulta possibile costruire intervalli di previsione ad hoc, che consentono di valutare le stime in modo più completo rispetto alla sola previsione puntuale, in modo da poter quantificare la precisione e la bontà di tali stime. Tuttavia, le assunzioni riguardati la distribuzione della variabile risposta possono portare a problemi di misspecificazione che impattano sulla bontà delle previsioni e questo risulta essere uno dei problemi più comuni nelle applicazioni reali. Di conseguenza, gli intervalli di previsione costruiti sulla base di tali assunzioni possono risultare poco affidabili, con una copertura teorica che non corrisponde a quella reale. Per questo motivo nell'ambito del *machine learning*, l'approccio più studiato e sviluppato è il metodo di **Conformal Prediction** (CP) che permette di quantificare l'incertezza nelle previsioni generate da algoritmi predittivi arbitrari. In particolare, la CP permette di costruire intervalli di valori (in regressione) o insiemi

di classi (in classificazione) che, con elevata probabilità, contengono il vero valore della variabile di interesse.

In questo capitolo viene introdotta CP nel contesto di problemi di regressione. In particolare, tra la classe di metodi di interesse emerge **Split Conformal Prediction**, il quale corrisponde ad un metodo computazionalmente efficiente per la costruzione di *prediction set*. Tale metodo, infatti, risulta essere centrale in questo capitolo. Successivamente, si affrontano i contesti in cui l'assunzione di scambiabilità non sia soddisfatta per l'utilizzo di CP in modelli *off-policy*. Infatti, quest'ultimi sono caratterizzati da *shift* nella distribuzione delle azioni che rappresenta una delle covariate nei problemi RL. Per tale ragione, il secondo metodo di principale interesse è rappresentato da **Weighted Conformal prediction**. Le referenze principali in questo capitolo sono: Angelopoulos & Bates (2021), Zhang et al. (2023) e Barber et al. (2023).

3.1 Conformal prediction standard

Conformal prediction (CP) è una tecnica innovativa di machine learning, introdotta da Vovk et al. (2005) e sviluppata significativamente negli ultimi anni. Il vantaggio principale (ed il motivo del suo successo) riguarda lo sviluppo di intervalli di previsione validi a fronte di assunzioni rilassate rispetto agli altri metodi presenti in letteratura. Si tratta di uno strumento di facile interpretazione e applicazione, utilizzabile in diversi ambiti, come il machine learning e il reinforcement learning. CP consente di costruire insiemi o intervalli di incertezza statisticamente significativi per le previsioni dei modelli, risultando particolarmente utile nei contesti in cui le decisioni devono essere prese sulla base delle stime prodotte. Un aspetto fondamentale è costituito dall'assenza di assunzioni riguardante la distribuzione delle variabili risposta, difatti definendo un metodo *distribution free*, rendendo CP un metodo generale e applicabile ad ogni modello. Tale proprietà permette di superare le problematiche di misspeficazione del modello, rendendo più robusta l'incertezza e, pertanto, l'intervallo di previsione. Infatti, CP può essere applicata a qualunque modello già addestrato, come una rete neurale o un random forest, per produrre insiemi che garantiscono di contenere il vero valore con una probabilità prestabilita dall'utente.

Al fine di descrivere le caratteristiche di base legate alla costruzione degli intervalli di previsione, di seguito vengono enunciate le quantità necessarie. Sia $\mathcal{D}$ un dataset composto dalle coppie (x_i, y_i), *con* $i = 1, \dots n$, dove $x_i \in \mathcal{X}$ che è il vettore delle

covariate, quindi delle variabili indipendenti, e $y_i \in \mathcal{Y}$ rappresenta la variabile risposta. La coppia (x_{n+1}, y_{n+1}) indica la nuova osservazione su cui fare previsione tramite un modello selezionato. Si assume che le osservazioni siano *exchangeable*, ciò implica che la probabilità congiunta della sequenza di osservazioni non cambia se si permuta l'ordine delle osservazioni. A volte questa l'ipotesi di scambiabilità viene rafforzata restringendosi al campo delle osservazioni *i.i.d.*, le quali oltre a richiedere la stessa distribuzione, devono essere indipendenti tra di loro.

In questa fase si considera un modello predittivo qualunque, dove la previsione sulla nuova osservazione è data da $\hat{f}(x_{n+1})$. Tuttavia, questo valore rappresenta soltanto una previsione puntuale che non esprime la confidenza o l'incertezza del modello di addestramento, e non comunica se la previsione stessa possa essere considerata affidabile. Per risolvere a questo problema è utile fornire un margine di errore attorno alla previsione, oppure più semplicemente, un insieme di previsione $\mathcal{C}(x_{n+1})$, tale che:

$$P(y_{n+1} \in \mathcal{C}(x_{n+1})) \geq 1 - \alpha \tag{3.1}$$

dove $\alpha \in (0, 1)$ è il livello d'errore specificato dall'utente (es. $\alpha = 0.1$).

A livello intuitivo, calcolando un intervallo o un insieme che accompagni la previsione puntuale del modello, $\hat{f}(x_{n+1})$, si è in grado di stabilire se con una certa probabilità il vero valore osservato risiede in tale intervallo, migliorando quindi la completezza di un risultato con scopo previsivo. Se la dimensione dell'insieme $\mathcal{C}(x_{n+1})$ è ampia, ciò indica elevata incertezza nella previsione. L'insieme di incertezza definito è l'obiettivo principale di CP che mira a quantificare il grado di affidabilità della previsione senza fare assunzioni sulla correttezza del modello. In particolare, se il modello predittivo si adatta male ai dati allora l'intervallo di previsione sarà grande, mentre se un modello si adatta in modo ottimale ai dati, l'intervallo associato sarà piccolo e preciso.

3.1.1 Split conformal prediction

Un semplice metodo per implementare e calcolare il *prediction set*, ovvero l'intervallo di previsione, è tramite il metodo **split conformal prediction**, in cui il dataset a disposizione viene suddiviso in due parti:

- **Training set** $\mathcal{D}_{\text{train}}$: formato da $(x_i, y_i), \quad i = 1, \dots n_1$, su cui viene stimato il modello predittivo $\hat{f} : \mathcal{X} \to \mathcal{Y}$. Qui l'obiettivo è ottenere una funzione che fornisca previsioni puntuali $\hat{y} = \hat{f}(x)$;

- **Calibration set** $\mathcal{D}_{\text{cal}}$: è un sottoinsieme del datset iniziale, formato da $(x_i, y_i), \quad i = 1, \dots n_2$, non utilizzato per addestrare il modello $\hat{f}$. Questa parte di dataset è impiegata per calcolare metriche utili nella costruzione dell'intervallo di previsione.

La numerosità del dataset iniziale n è data dalla somma tra la numerosità del calibration set n_2 e quella del training set n_1. L'assunzione chiave è quella di **scambiabilità** dei dati, ovvero deve valere che la distribuzione del training set sia uguale a quella del calibration set. Grazie a quest'ipotesi, si garantisce il risultato in Equazione 3.1. Una definizione più formale di scambiabilità (da Angelopoulos & Bates (2021)) è specificata nella Definizione 2.

Definizione 2. *Siano* $X_1, \dots, X_n \in \mathcal{X}$ *variabili aleatorie con una distribuzione congiunta. Il vettore casuale* $(X_1, \dots, X_n)$ *si definisce* ***scambiabile*** *se, per ogni permutazione* $\sigma \in S_n$ *vale che:*

$$(X_1, \dots, X_n) \overset{d}{=} (X_{\sigma(1)}, \dots, X_{\sigma(n)}) \tag{3.2}$$

dove $\overset{d}{=}$ *indica l'uguaglianza in distribuzione e* S_n *è l'insieme delle possibili permutazioni* σ*.*

Considerando come test set $(\mathcal{D}_{\text{test}})$ una sola osservazione (x_{n+1}, y_{n+1}) gli obiettivi principali del metodo di *split conformal prediction* sono elencati di seguito.

- **Coverage**, rappresenta la probabilità effettiva che il vero valore di Y_{n+1} sia contenuto all'interno del *prediction set*.
- L'**ampiezza dell'intervallo**: gli intervalli di previsione devono avere un'ampiezza adeguata. Un intervallo eccessivamente ampio, infatti, risulta poco informativo poiché tende a includere un numero troppo elevato di valori, segnalando che il modello considerato non fornisce stime sufficientemente precise.
- L'**adattività dell'intervallo**: gli intervalli devono esser in grado di adattarsi alla complessità del modello, diminuendo o aumentando la dimensione in base alla difficoltà del modello nella previsione.

Per giudicare la qualità di un intervallo di previsione è fondamentale valutare congiuntamente l'ampiezza dell'intervallo e la coverage: valori elevati di coverage

possono essere ottenuti anche con intervalli molto larghi, ma in tal caso l'informazione fornita sull'incertezza delle previsioni risulterebbe limitata.

In un problema di regressione, quando la variabile risposta $Y \in \mathbb{R}$ assume valori continui, l'idea del metodo di *split conformal prediction* è quella di costruire dei residui in modo tale da trattare tutti i dati (incluso il dato di test) in maniera simmetrica, così da garantire che i residui rispettino la condizione di scambiabilità necessaria per ottenere la copertura. L'intervallo di previsione viene creato seguendo la seguente procedura:

1. Si allena il modello di previsione $\hat{f}$ sul training set, che rimane fisso per tutte le fasi successive.

2. Per ogni coppia di valori all'interno del calibration set $(x_i, y_i) \in \mathcal{D}_{\text{cal}}$ si calcola la **score function** s, data da i residui assoluti:

$$s(x_i, y_i) = R_i = |y_i - \hat{f}(x_i)|, \quad i \in 1, ..., n_2 \tag{3.3}$$

3. I residui ottenuti vengono poi ordinati in modo crescente: $R_{(1)} \leq \cdots \leq R_{(n_2)}$ e utilizzati per determinare la soglia $\hat{q}$, calcolata come il quantile empirico di livello $\lceil (1-\alpha)(n_2+1) \rceil$:

$$\hat{q} = quantile(R_{(1)}, \ldots, R_{(n_2)}; \lceil (1-\alpha)(n_2+1) \rceil) \tag{3.4}$$

4. Il *prediction set* per un nuovo punto x_{n+1} è definito come l'insieme di tutti i possibili valori y il cui la *score function* non supera la soglia $\hat{q}$:

$$\mathcal{C}(x_{n+1}) = \{y \in \mathcal{Y} : \ s(x_{n+1}, y) \leq \hat{q}\} \tag{3.5}$$

   Per un problema di regressione, questo si traduce nel seguente intervallo di previsione:

$$\mathcal{C}(x_{n+1}) = [\hat{f}(x_{n+1}) - \hat{q}, \ \ \hat{f}(x_{n+1}) + \hat{q}] \tag{3.6}$$

Sotto l'assunzione di scambiabilità dei dati, la *split conformal prediction* garantisce che l'insieme di predizione $\mathcal{C}(x_{n+1})$ contenga il vero valore y_{n+1} con una probabilità di almeno $1 - \alpha$.

3.2 Weigthed conformal prediction

CP è una tecnica molto diffusa per fornire intervalli di previsione validi attraverso modelli predittivi arbitrari. Essa si basa sull'assunzione di scambiabilità, secondo cui la distribuzione congiunta del calibration set e del test set deve essere invariante rispetto all'ordine delle osservazioni, e sulla simmetria dell'algoritmo di stima del modello rispetto ai dati. Tuttavia, l'ipotesi di scambiabilità viene spesso violata quando i modelli predittivi vengono applicati nella pratica. Si potrebbero avere dati che non sono scambiabili, ad esempio potrebbero essere indipendenti ma non identicamente distribuiti, oppure potrebbero presentare dipendenze che generano non scambiabilità, ad esempio dipendenze temporali. Inoltre, anche l'ipotesi di simmetria dell'algoritmo può essere violata; ad esempio, nel caso di dati raccolti nel tempo, si potrebbero preferire algoritmi che attribuiscono un peso maggiore alle le osservazioni più recenti. In questa sezione, il contesto principale è rappresentato da dati caratterizzati da **distribution shift**, ovvero quando la distribuzione dei dati cambia tra il training set e il test set. Sotto questa ipotesi costruire intervalli di previsione seguendo il metodo CP classico comporta a una distorsione nell'ampiezza degli intervalli, poiché alcuni punti del training set possono più o meno essere rappresentativi del test set. Attraverso il metodo di **Weighted Conformal Prediction** ogni osservazione viene pesata in modo tale da dare maggior importanza alle istanze del training set più simili a quelle del test set, migliorando così l'efficienza e la robustezza degli intervalli di previsione.

Per affrontare la problematica di scambiabilità dei dati, viene introdotto da Tibshirani et al. (2019) il concetto di *weighted exchangeability* che rilassa l'assunzione classica di *i.i.d.* e permette di costruire intervalli di previsione tramite *weighted CP*.

Definizione 3. *Siano $x_1, \ldots, x_n$ un campione di dati, con il test set formato da x_{n+1}. Si dice che essi sono* ***weighted exchangeable*** *se la densità f della loro distribuzione congiunta può essere fattorizzata come:*

$$f(x_1, \ldots, x_{n+1}) = \prod_{i=1}^{n+1} w_i(x_i) g(x_i, \ldots, x_{n+1}) \tag{3.7}$$

per certe funzioni di peso $\{w_i\}_{i=1}^{n+1}$ e una funzione g invariante a permutazioni, tale che:

$$g(x_{\sigma(1)}, \ldots, x_{\sigma(n+1)}) = g(x_1, \ldots, x_{n+1}) \tag{3.8}$$

per ogni permutazione σ di $\{1, \ldots, n+1\}$.

Secondo questa definizione, dati indipendenti sono sempre *weighted exchangeable*, con i pesi corrispondenti al *likelihood ratio* tra la distribuzione del dataset di training e la distribuzione del dataset di test.

Data la funzione di score $s(x_i, y_i)$ sul calibration set, calcolata a partire da un determinato algoritmo allenato sul training set, l'intervallo di previsione non si basa sui quantili empirici della funzione di score, ma l'algoritmo *weighted CP* considera una versione pesata della score. Data un'oservazione del test x_{n+1} si costruisce l'intervallo predittivo per il valore di risposta y come:

$$\mathcal{C}(x_{n+1}) = \{y \in \mathcal{Y} : s(x_{n+1}, y) \leq \ quantile\ (1-\alpha)\ di\ \sum_{i=1}^{n} w_i \delta_{s_i} + w_{n+1}\delta_{\infty}\} \quad (3.9)$$

dove $(1-\alpha)$ indica il livello di copertura desiderato, δ_α denota la distribuzione di *Dirac* che concentra tutta la massa sul valore α. Inoltre, è importante ricordare che il metodo *weigthed CP* possiede diverse proprietà statistiche: non dipende da alcuna assunzione sulla distribuzione della risposta date le covariate e garantisce che l'insieme di previsione $\mathcal{C}(x_{n+1})$ contiene una probabilità $\geq (1-\alpha)$ di contenere il vero valore della risposta.

3.3 Conformal off-policy prediction

Nel contesto degli algoritmi di apprendimento *off-policy* è fondamentale valutare l'efficacia di una policy prima della sua implementazione, per stimarne l'impatto in termini di stati visitati che di rewards ottenute. A questo scopo Zhang et al. (2023) definisce il problema della *off-policy evaluation*, che si basa su un dataset osservato generato da una policy totalmente diversa da quella che poi viene selezionata come migliore. Nei metodi *off-policy*, infatti, la distribuzione del training set differisce dalla distribuzione del test set. Il training set è generato secondo una policy osservata (detta *behavior policy* π_b), mentre il test set segue una policy sviluppata tramite algoritmo FQI (detta *target policy* π). A causa di questo i dati non possono essere considerati scambiabili e non è possibile applicare CP nella sua formulazione classica. Nel reinforcement learning questo problema è ulteriormente accentuato: i dati sono raccolti il modo sequenziale, e ogni stato dipende dal precedente. Di conseguenza, l'assunzione di scambiabilità viene violata, poiché la distribuzione congiunta del training set e del test set varia in funzione dell'ordine delle osservazioni.

Per affrontare questo tipo di problemi, Zhang et al. (2023) propone una nuova procedura per la costruzione di intervalli predittivi in contesti *off-policy*, utilizzando

la reward ottenuta tramite una *target policy*. Inizialmente si concentra su un processo decisionale a un singolo stadio, in cui l'obiettivo è costruire un intervallo di previsione per la variabile risposta Y_{n+1}^{π}, ovvero il valore della reward generata seguendo la *target policy*. Successivamente, il problema viene esteso a un contesto sequenziale, in cui la reward è osservata solo all'ultimo stadio. In questa sezione ci si focalizza sul primo caso: la variabile risposta Y corrisponde alla reward, mentre le covariate $\mathbf{X}$ sono costituite dallo stato S e dall'azione A.

Un primo approccio per la costruzione dell'intervallo di previsione in ambito *off-policy* è l'utilizzo di **weighted conformal prediction** con pesi definiti in modo appropriato per soddisfare l'assunzione di *weighted exchangeability*. I pesi sono definiti di seguito:

- $w_i(x) = 1 \quad \forall\, i \in \mathcal{D}_{\text{cal}}$;
- $w_i(x, y) = \dfrac{dP_{Y^{\pi}|X}(y|x)}{dP_{Y|X}(y|x)} \;\; \forall\, i \in \mathcal{D}_{\text{test}}$.

dove $P_{Y^{\pi}|X}$ indica la distribuzione condizionata di Y dato X nel test set, mentre $P_{Y|X}$ rappresenta la distribuzione di Y dato X nel calibration set. Per applicare questo metodo è necessario stimare il peso delle osservazioni del test set, un compito estremamente complesso in sistemi non lineari ad alta dimensionalità. Inoltre, questo approccio risulta inefficace se la il modello della densità condizionata $Y^{\pi}|X$ è specificato in modo errato.

Un altro approccio per gestire il problema della scambiabilità dei dati è il **subsampling-based method**. Il quale consiste nel selezionare un sottoinsieme di dati la cui distribuzione π_b sia simile alla distribuzione target π e applicare il metodo CP standard a questo sotto-campione. In particolare, per ogni osservazione del calibration set, si campiona una **pseudo-azione** E_i secondo la policy target π, si seleziona il sotto-campione per il quale la *pseudo-azione* coincide con l'azione osservata e si applica CP a questi dati. Tuttavia, questo approccio non è molto valido e, in generale, tende a produrre intervalli di previsione che non coprono adeguatamente la risposta $Y^{\pi}|X$. Ciò avviene perché la distribuzione dei sotto-campioni selezionati nel calibration set è diversa da quella del test set; infatti, due distribuzioni coincidono solo in casi particolari.

Per migliorare i due metodi presentati precedentemente Zhang et al. (2023) propone un nuovo approccio denominato **Conformal off-policy prediction** (COPP). Invece di campionare secondo la policy target π, si costruisce una nuova **pseudo-policy/policy ausiliaria** π_a la cui distribuzione dipende sia da π che da

π_b, in modo tale che la distribuzione dei sotto-campioni coincida con quella di $Y^{\pi}|X$. La distribuzione di π_a è:

$$\pi_a(\cdot|x) \propto \frac{\pi(\cdot|x)}{\pi_b(\cdot|x)} \quad \forall\, x \tag{3.10}$$

Sia A^* la *pseudo-azione* generata secondo π_a, allora i sotto-campioni selezionati in modo tale che l'azione selezionata A^* sia uguale a quella osservata A, seguono una distribuzione condizionata coincidente con quella delle risposte target $P_{Y^{\pi}|X}$. Tuttavia, rimane il problema che le covariate X dei sotto-campioni seguono una distribuzione diversa rispetto a quella dei valori target. Questo problema può essere gestito dal metodo di *weighted CP*, costruendo pesi adeguati. I pesi in questo contesto sono definiti come:

- $w_i(x) = 1 \quad \forall\, i : A_i^* = A_i$;
- $w_i(x, y) \propto \dfrac{1}{\mathbb{P}(A = A^*|X = x)}$.

Questa specificazione è robusta rispetto alla specificazione errata della distribuzione condizionata $P_{Y^{\pi}|X}$. L'algoritmo COPP viene presentato in modo dettagliato in Zhang et al. (2023) e consente di superare i limiti principali degli approcci di *weighted conformal prediction* e di *subsampling* discussi in precedenza. In altre parole, l'algoritmo COPP utilizza una *policy ausiliaria* π_a per selezionare i sotto-campioni, applicando successivamente il metodo *weighted CP* con pesi specifici.

3.4 Algoritmo di conformal prediction per FQI

In questa sezione viene descritto l'algoritmo utilizzato nell'applicazione attraverso il quale si costruiscono i *prediction set*, le cui referenze sono Barber et al. (2023), Oliveira et al. (2024) e Zheng & Proutiere (2024). L'algoritmo in considerazione è stato sviluppato partendo dall'approccio descritto nel Paragrafo 3.3, apportando alcune modifiche. In particolare, come variabile risposta Y è stata considerata la reward cumulata, partendo dal passo corrente fino all'ultimo istante temporale di ogni episodio e come valori previsti dal modello $\hat{f}$ vengono considerati i valori $\hat{Q}$ stimati della funzione valore-azione. Il dataset a disposizione in cui si osservano gli stati, le azioni e le reward secondo una *behavior policy* viene suddiviso in training set $\mathcal{D}_{\text{train}}$, calibration set $\mathcal{D}_{\text{cal}}$ e test set $\mathcal{D}_{\text{test}}$, gli intervalli di previsione vengono creati seguendo la procedura elencata di seguito.

1. Si allena l'algoritmo FQI sul training set, utilizzando al suo interno un modello di previsione $\hat{f}$.

2. Si costruisce un dataset destinato alla determinazione dei pesi w_i nell'applicazione del metodo di *weighted CP*. Per la sua creazione, si campiona un sotto-insieme delle osservazioni del training set; successivamente, tramite il modello ottenuto dall'algoritmo FQI, si calcolano le azioni secondo la nuova policy. A queste osservazioni viene assegnato $Y = 0$, mentre per le azioni effettivamente osservate nel sotto-insieme si assegna $Y = 1$.

3. Per la determinazione dei pesi si allena sul nuovo dataset, creato al punto precedente, un modello di regressione logistica.

4. Si calcolano le azioni e gli stati del calibration set aggiornandoli secondo la nuova policy individuata nell'algoritmo FQI. Per ogni azione selezionata si prevede il peso associato w_i tramite il modello di regressione logistica. I pesi ottenuti vengono successivamente normalizzati.

5. Si definisce la funzione di score $s(x_i, y_i)$ come la somma delle reward cumulate fino alla fine dell'episodio meno il valore $\hat{Q}_i$ stimato moltiplicato per il peso associato all'azione w_i:

$$s(x_i, y_i) = s_i = \left(\gamma \sum_{i=t}^{T} reward_i - \hat{Q}_i \right) w_i, \quad \forall\, i \in \mathcal{D}_{\text{cal}} \tag{3.11}$$

   dove γ è il fattore di sconto nell'algoritmo FQI.

6. Si ordinano in modo crescente i valori della funzione di score $s_{(1)}, \ldots, s_{(n)}$, dove n è la numerosità di $\mathcal{D}_{\text{cal}}$, e si calcola il quantile empirico di ordine $\lceil (1-\alpha)(n+1) \rceil$:

$$\hat{q} = quantile(s_{(1)}, \ldots, s_{(n)}; \lceil (1-\alpha)(n+1) \rceil) \tag{3.12}$$

7. Infine, sul test set si calcolano le azioni ottime e gli stati visitati seguendo la policy determinata dall'algoritmo FQI e per ogni valore $\hat{Q}_i$ si costruiscono gli intervalli di previsione:

$$\mathcal{C}(x_i) = [\hat{Q}_i - \hat{q},\ \hat{Q}_i + \hat{q}], \quad \forall\, i \in \mathcal{D}_{\text{test}} \tag{3.13}$$

L'algoritmo descritto consente di calcolare i *prediction set* nell'ambito FQI. Una possibile estensione consiste nell'inserire la variabile tempo come covariata, di modo da poter utilizzare algoritmi non simmetrici. Tale generalizzazione può condurre a

intervalli di previsione più accurati e verrà inserita nell'applicazione i cui risultati sono presentati in Sezione 4.3. I dettagli tecnici dell'algoritmo sono riportati in Barber et al. (2023).

Capitolo 4

Applicazione su dataset *House Heating*

In quest'ultimo capitolo viene presentata la parte applicativa di questa tesi. In particolare, si concentra sull'utilizzo dell'algoritmo *Fitted Q-Iteration* (FQI) in cui vengono utilizzati diversi tipi di modelli *supervised* individuando quello che porta a performance migliori sulla base della somma delle reward ottenute. Poiché l'algoritmo FQI viene definito come metodo offline, esso richiede l'esistenza di un dataset che comprende un numero limitato di osservazioni derivanti dall'utilizzo di una policy pre-esistente e sub-optimale. Nella prima sezione viene, quindi, descritto il dataset utilizzato, il quale è stato proposto da Candelieri et al. (2023). Nella seconda sezione, invece, vengono descritte le peculiarità applicative dell'algoritmo di reinforcement learning in esame, la sua implementazione, i sui vantaggi e svantaggi, presentando i risultati ottenuti e focalizzandosi sul modello statistico migliore. Infine, viene implementato l'algoritmo di *Conformal Prediction*, illustrato in 3.4, evidenziando i risultati ottenuti sul dataset.

4.1 Dataset *House Heating*

Nell'ambito dei metodi offline di reinforcement learning, tra cui FQI, è molto importante la scelta del dataset da impiegare poiché esso deve essere formattato in un modo specifico. Per applicare questo tipo di algoritmi, i dati sono il risultato dell'applicazione di un sistema dinamico, che si modifica nel tempo seguendo un criterio (o *policy*). Con sistema dinamico si intende qualsiasi sistema - naturale, tecnico o sociale - in cui lo stato cambia nel tempo in base a regole interne che possono essere influenzate da stimoli esterni. In altre parole, è un sistema che evolve continuamente, e il cui comportamento è influenzato dall'ambiente che lo circonda e dalla situazione in cui si trova in ogni istante temporale. I sistemi dinamici si possono riferire a diversi ambiti di applicazione reale, possono essere sistemi naturali (come

un ecosistema o il clima) oppure artificiali (come un impianto industriale). Possono essere semplici, come il controllo della temperatura interna di un edificio tramite un termostato oppure molto complessi, come la gestione di un'intera rete elettrica nazionale. I sistemi dinamici coprono molti ambiti della vita reale, ad esempio in ambito ingegneristico, medico e biologico, economico e finanziario ed energetico.

Il controllo di sistemi dinamici reali si basa spesso su politiche orientate alla sicurezza, a scapito di prestazioni sub-ottimali. Generalmente, queste politiche sono progettate da esperti, sono fondate sulla loro conoscenza del sistema di riferimento e permettono al sistema di ritrovarsi in stati in cui la sicurezza dell'ambiente circostante è garantita. Questo tipo di politiche sono definite *safe-by-design*, ovvero per definizione si tratta di politiche che permettono di scegliere azioni che mantengono livelli di stato all'interno dei vincoli di sicurezza (*safe constrains*), non garantendo però l'efficienza della scelta in termini di reward. Il concetto di sicurezza è da intendersi in senso ampio: riguarda sia la prevenzione di interruzioni del sistema stesso, ad esempio impianti di produzione o di riscaldamento che si bloccano, sia nell'evitare di fornire servizi di scarsa qualità, come la distribuzione inefficiente di acqua o energia.

Il dataset utilizzato nell'applicazione di questa tesi è stato creato seguendo una politica *safe-by-design*, quindi le azioni intraprese derivano da una politica che viene definita per cercare di massimizzare la sicurezza del sistema. L'obiettivo è quello di applicare l'algoritmo FQI in modo da individuare una politica ottimale e, di conseguenza le azioni che permettano sia di non violare le condizioni di sicurezza che di massimizzare le rewards ottenute (nel caso in esame si tratta di massimizzare il negativo dei costi, in modo da minimizzarli). I dati considerati rappresentano il controllo di un sistema di riscaldamento all'interno di un'abitazione, rappresentato schematicamente in Figura 4.1. La politica *safe-by-design* π riceve come input la temperatura interna corrente s_t ed in base ad essa imposta la temperatura sul termosifone a_t (azione), per quell'istante temporale. Essendo una politica di controllo, la scelta dell'azione non viene influenzata dalla reward r_t osservata, mentre la politica stimata tramite l'algoritmo FQI tiene conto anche di essa. Sul sistema agisce anche una componente incerta data dalla temperatura esterna ξ_t, che varia durante la giornata: più alta è la differenza tra la temperatura interna e quella esterna, maggiore è la dissipazione di calore (a seconda delle proprietà fisiche delle pareti della casa).

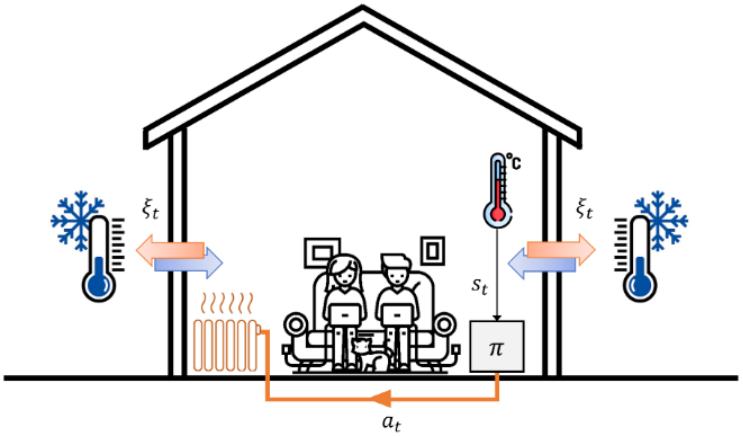

Figura 4.1: Sistema dinamico di riscaldamento dell'abitazione (Candelieri et al., 2023).

Tale sistema dinamico rappresenta un problema di controllo della temperatura interna che mira a individuare una policy che permetta di raggiungere una temperatura interna obiettivo entro un certo intervallo di tempo dopo l'accensione del riscaldamento, evitando sia un'eccesiva sovraelongazione, sia cercando di mantenere una temperatura interna il più stabile possibile durante tutta la giornata. I vincoli di sicurezza (*safe constrains*) in questo tipo di applicazione si riferiscono al fatto di evitare grandi disagi agli abitanti della casa. Nel dataset utilizzato la temperatura interna, quindi lo stato s_t, segue una regola di transizione definita tramite la seguente equazione:

$$s_{t+\Delta t} = s_t + \frac{\Delta t}{\kappa}\left[M\kappa(a_t - s_t) + \frac{s_t - \xi_t}{R}\right] \tag{4.1}$$

dove:

- $M\kappa(a_t - s_t) = \frac{dQ_G}{dt}$ rappresenta il *tasso di guadagno di calore*;
- $\frac{s_t - \xi_t}{R}$ rappresenta il *tasso di perdita di calore*;
- M è il tasso di aria emessa dal riscaldatore considerata costante nell'intervallo di tempo ($M = 180\,\text{kg·h}$);
- κ è la capacità del riscaldatore ($\kappa = 1500.4$ Joule/°C $\cdot$ kg);
- m è la massa d'aria nella casa ($m = 1470\,\text{kg}$);
- R è la resistenza termica della casa ($R = 4.329 \cdot 10^{-5}\,°\text{C} \cdot \text{h/Joule}$).

Infine, per la temperatura esterna, ξ_t è stato utilizzato il dataset delle temperature del parco Yosemite (*Yosemite temperature dataset*) contenente le misurazioni

giornaliere della temperatura, campionate ogni minuto, relative a 60 giorni differenti. Questo dataset è stato però adattato in modo da poterlo utilizzare nell'applicazione campionando le temperature ogni 30 minuti al posto che ogni singolo minuto. Tutti i valori della temperatura, originariamente espressi in gradi Fahrenheit, sono stati convertiti in gradi Celsius.

I **vincoli di sicurezza (safe constrains)** sono:

- portare la temperatura interna il più vicino possibile alla temperatura obiettivo di 18° entro 30 minuti dall'accensione del termosifone: $s^* = 18°$ *temperatura target*;
- non superare una temperatura di 19° nei primi 30 minuti: $s_t \leq 19°$ *per* $t \leq 30$ *min*;
- mantenere una temperatura interna vicina alla temperatura target, $s_t \in [s^* - 0.5, s^* + 0.5]$ *per* $t > 30$ *min*, ovvero la temperatura deve restare tra i $17,5°$ e i $18.5°$.

Il **vincolo operativo**, invece, determina le temperature che si possono impostare nel termosifone, ovvero le azioni a_t possono variare in un range che va da 0° a 70°: $a_t \in [0°, 70°]$.

La politica *safe-by-design* descrive la procedura di scelta delle azioni nel seguente modo:

$$a_t = \min \left\{ K_p\, \varepsilon_t + K_i \sum_{j=0}^{t} \varepsilon_j + K_d \frac{\varepsilon_t - \varepsilon_{t-1}}{\Delta t},\ a_{\max} \right\} \tag{4.2}$$

dove: $\varepsilon = s^* - s_t$ rappresenta l'errore di temperatura rispetto al valore target, $a_{\max}$ è pari a 70° e il resto dei parametri sono $K_p = 17.5$, $K_i = 5.9$ e $K_d = 1.4$.

In Figura 4.2, viene mostrata l'implementazione della politica *safe-by-design* in un determinato giorno, in cui viene riportata la temperatura dell'abitazione insieme ai vincoli di sicurezza.

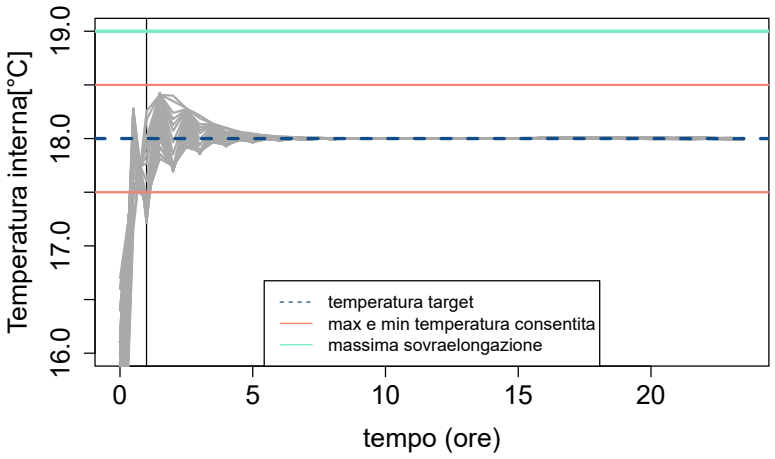

Figura 4.2: Implementazione della policy *safe-by-design* in un determinato giorno.

La politica che viene utilizzata nel dataset a disposizione, però, non tiene conto dei costi che si sostengono ad ogni istante temporale, le azioni vengono scelte solo in modo da rispettare i vincoli. I costi relativi al riscaldamento dipendono dalle seguenti tariffe energetiche orarie:

$$\text{prezzo}_t = \begin{cases} 1\,€/°\text{C}, & \text{se } t \in [0,7] \cup [21,24] \\ 10\,€/°\text{C}, & \text{se } t \in (7,10] \cup (17,21] \\ 5\,€/°\text{C}, & \text{se } t \in (10,17] \end{cases}$$

Il dataset impiegato è stato simulato attraverso un generatore di dati, ed è stato costruito in modo da ottenere osservazioni ogni 30 minuti per 60 giorni. Ad ogni mezz'ora si registra lo stato s_t, ossia la temperatura interna della casa, l'azione compiuta a_t secondo la politica *safe-by-design*, e i costi sostenuti per l'utilizzo del riscaldamento nell'intervallo considerato, calcolati moltiplicando l'azione compiuta in una determinata ora per il prezzo orario. L'obiettivo è quello di minimizzare i costi totali:

$$\min_{a_t \in [0°, 70°]} \sum_{t=0}^{T} a_t \times \text{prezzo}_t \tag{4.3}$$

4.2 Implementazione e risultati

Per utilizzare i dati a disposizione è stata costruita una tabella di 2880 righe, in cui ciascuna riga contiene lo stato, lo stato al passo successivo, l'azione, la ricompensa, tutti riferiti a intervalli di 30 minuti e l'indicazione (*done*) se lo stato risulta essere lo

stato finale della giornata. Si considerano 60 giorni e quindi 60 episodi indipendenti, in cui lo stato nella prima mezz'ora di ogni giorno è indipendente dall'ultimo stato del giorno precedente. Ogni episodio è formato da 48 righe. Per iniziare l'analisi si è suddiviso il dataset in *training set*, formato da 40 giorni (1920 righe), e *test set*, formato da 20 giorni (960 righe), è stato applicato l'algoritmo solo alla parte di training e valutandolo poi tramite la parte di test.

4.2.1 Definizione della reward

Nel dataset considerato sono presenti i costi sostenuti per ogni intervallo di 30 minuti, secondo l'azione svolta e il prezzo orario. Considerare questo dato come reward non è molto vantaggioso, poiché l'algoritmo è predisposto in modo da massimizzare le rewards che ottiene ad ogni passo e quindi c'è bisogno di definirla in modo corretto. Una prima ipotesi consiste nel considerare esclusivamente i costi, impostando la reward pari al negativo dei costi. La selezione di una reward di questo tipo porta a non tener conto dei vincoli di sicurezza e si concentra solo sulla minimizzazione dei costi, ricordando che nell'algoritmo FQI si procede con la massimizzazione della reward. Questo conduce alla selezione di azioni molto piccole, ovvero temperature molto basse vicino al valore di 0°, in tutto l'arco della giornata; senza quindi riuscire a mantenere una temperatura stabile e vicino a quella target all'interno della casa. Per questo motivo è stata introdotta una nuova reward che contenesse una penalizzazione. La reward utilizzata nel dataset è stata calcolata nel seguente modo:

$$r_t = -\text{costo}_t - \lambda|18 - s_t| \tag{4.4}$$

In questo modo, la minimizzazione dei costi risulta penalizzata rispetto alla necessità che la temperatura nello stato corrente sia sufficientemente vicina alla temperatura obiettivo. Tramite la penalizzazione l'algoritmo è in grado di selezionare azioni che permettono di minimizzare i costi mantenendo una temperatura interna adeguata. Il parametro λ rappresenta il peso della penalizzazione e, nel caso in esame, dopo diverse prove, è stato fissato al valore ottimale di 600. Si tratta di una misura piuttosto elevata: analizzando tutti i valori degli stati $s_t,\ t = 1, \ldots, 2880$ presenti nel dataset, si osserva che essi risultano essere molto vicini alla temperatura target di 18°, di conseguenza, la parte di penalizzazione assume valori decimali prossimi a 0. Quindi, per far si che la penalizzazione avesse un impatto significativo sul costo, è stato necessario inserire un valore elevato del parametro λ, anche in considerazione del fatto che il costo varia in un intervallo compreso tra 0 e 189.

Di seguito, in Figura 4.3 e 4.4 vengono rappresentati i grafici della reward in funzione rispettivamente dello stato e dell'azione. Si nota come la relazione tra le variabili non assume un andamento lineare. Nel grafico della relazione tra stato e reward sembra che ci sia un andamento lineare crescente per i valori al di sotto dei 18°, mentre un andamento decrescente per quelli con valori al di sopra dei 18°. Tuttavia, la maggior parte delle osservazioni assume, per la variabile stato, un valore vicino a 18° a cui sono associati valori della reward abbastanza elevati, vicini a 0. Nel grafico della relazione tra azione e reward si nota che la maggior parte delle osservazioni si concentrano in alto a sinistra, che rappresenta valori dell'azione al di sotto dei 30° a cui sono associati valori della reward più elevati, vicini a 0.

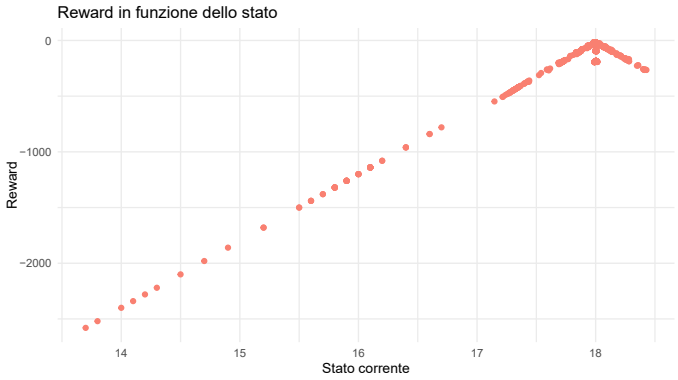

Figura 4.3: Stato vs Reward.

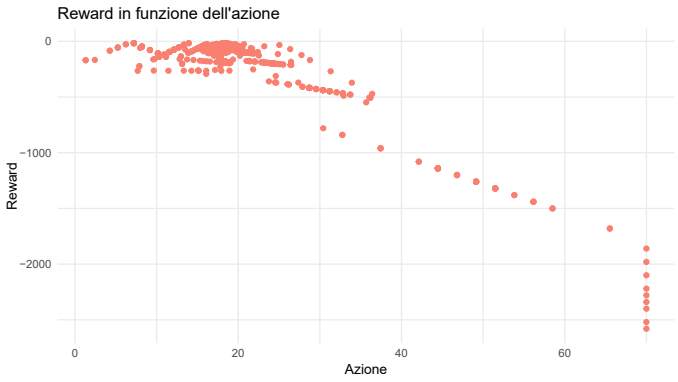

Figura 4.4: Azione vs Reward.

4.2.2 Modelli utilizzati

Nell'applicazione dell'algoritmo fitted Q-iteration il passaggio principale è l'aggiornamento, ad ogni iterazione, del valore Q trovando l'azione tra le possibili che lo

massimizzi, come mostrato nell'equazione che segue 4.5.

$$y_t = r_t + \gamma \max_{a \in \mathcal{A}} Q_{(k-1)}(s'_t, a) \tag{4.5}$$

Le azioni osservate all'interno del dataset a disposizione variano nell'intervallo compreso tra 1.323° e 70°, come mostrato in Figura 4.4. Per implementare l'algoritmo è stato necessario discretizzare le azioni possibili, in quanto il metodo FQI nella sua definizione originale si applica a spazi di azione discreti. Inoltre, un aumento della granularità della discretizzazione comporta ad un notevole aumento dei tempi computazionali. Le azioni possibili che sono state tenute in considerazione, infatti, variano nell'intervallo tra 0° e 70°, con passo di 0.5°, trattandosi di azioni discrete.

Un altro passo fondamentale nello sviluppo dell'algoritmo è la stima dei nuovi valori target tramite un modello di regressione, utilizzando come covariate le azioni e gli stati osservati. Nell'applicazione sono stati impiegati diversi modelli di regressione, addestrati sul dataset di training e successivamente valutati sul test set al fine di individuare il modello con le prestazioni migliori. La valutazione è stata effettuata considerando la riduzione dei costi, sia su base giornaliera sia complessiva, in funzione delle azioni selezionate dall'algoritmo (e quindi generate dalla nuova policy individuata), rispetto ai costi sostenuti adottando la policy safe-by-design. Ulteriormente, si è tenuto in considerazione se le azioni nuove selezionate portassero a temperature interne che rispettassero i vincoli citati precedentemente. Il principale vincolo da rispettare riguarda il mantenimento della temperatura interna prossima a 18°, garantendo in particolare che rimanesse compresa nell'intervallo $[17.5°, 18.5°]$, durante l'intera giornata.

L'algoritmo fitted Q-iteration è stato implementato applicando diversi modelli di regressione: la *regressione lineare*, il modello additivo con *regressione spline*, il *Random forest* e le *reti neurali*. Il fattore di sconto γ in 4.5 è stato impostato per tutti i modelli pari a 0.99.

Regressione lineare

Come prima prova è stato implementato l'algoritmo utilizzando una semplice **regressione lineare**, considerando sia le covariate stato e azione in modo singolare, sia l'interazione tra di esse, come mostrato nel modello nell'Equazione 4.6.

$$y_t = \beta_0 + \beta_1 * \text{stato}_t + \beta_2 * \text{azione}_t + \beta_3 * \text{stato}_t * \text{azione}_t + \varepsilon_t \tag{4.6}$$

Questo modello non ha portato a nessun risultato soddisfacente. Utilizzando il modello ottenuto all'ultima iterazione dell'algoritmo nella fase di aggiornamento degli stati nel test set, esso porta alla selezione dell'azione 0° per tutti i periodi di tempo considerato. Per questo motivo si è deciso di utilizzare modelli più complessi che fossero in grado di gestire la relazione non lineare tra i valori target modificati ad ogni iterazione e le covariate. Nel grafico seguente, in Figura 4.5, viene rappresentata la media, su tutti i giorni, della reward cumulata di ogni giorno, per ogni iterazione e l'intervallo di confidenza associato. Questo grafico è molto utile per la scelta del numero di iterazioni da inserire nell'algoritmo, si controlla quando la media delle rewards cumulate si stabilizza intorno a un valore. In questo caso si considerano come valore ottimale 50 iterazioni.

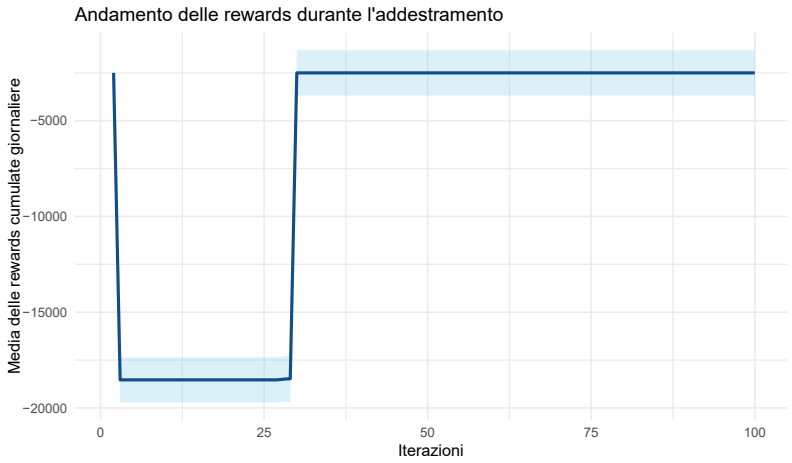

Figura 4.5: Media delle rewards cumulate giornaliere per ogni iterazione, tramite regressione lineare.

Regressione spline

Come modello additivo generalizzato (GAM) è stata utilizzata una regressione spline, in particolare *thin plate spline*, applicata sia alla variabile stato, che alla variabile azione, aggiungendo come predittore anche l'interazione tra le due covariate. Il modello ottenuto è il seguente:

$$y_t = \beta_0 + f(\text{stato}_t) + f(\text{azione}_t) + \beta_1 * \text{stato} * \text{azione} + \varepsilon_t \tag{4.7}$$

dove con $f(\text{stato}_t)$ e $f(\text{azione}_t)$ si intende una spline di tipo *thin plate*. Per l'implementazione è stato utilizzato il comando "gam" della libreria *mgcv* col software R, utilizzando il metodo "REML", che, per stimare i parametri di varianza, utilizza un metodo diverso rispetto alla verosimiglianza classica, portando a stime meno

distorte. Questo comando serve per stimare un modello additivo, in cui sia sulla covariata stato che sulla covariata azione è stata inserita una spline con il comando "s". In questo caso sono stati utilizzati gli iperparametri di default all'interno del comando per l'implementazione delle spline. In particolare, il parametro "bs" che rappresenta il tipo di spline da utilizzare, è stato impostato come "tp", ovvero *thin plate spline*. Un altro iperparametro è "k" che rappresenta il limite superiore dei gradi di libertà associati alla spline. Poiché i gradi di libertà sono associati ai nodi della spline, nel caso di *smoothing spline*, e quindi di *thin plate*, non c'è la necessità di impostarlo a priori (ogni nodo è rappresentato da ogni osservazione). In Figura 4.6, viene rappresentato il grafico della media delle rewards cumulate giornaliere per ogni iterazione. Si nota che, dopo poche iterazioni, i valori si stabilizzano. Per tale ragione sono state utilizzate 50 iterazioni per ottenere i risultati che vengono mostrati nella Sezione 4.2.3.

Figura 4.6: Media delle rewards cumulate giornaliere per ogni iterazione, tramite regressione thin plate spline.

Random Forest

Il **random forest** è un metodo supervisionato di machine learning che, partendo da un dataset formato da covariate $(x_1, ...x_p)$ e una variabile risposta y, permette di prevedere i valori della variabile y (sia per classificazione, che per regressione) tramite una collezione di alberi decisionali non correlati. Si creano B campioni bootstrap dal dataset di training su cui si crea un albero T_b. Ad ogni nodo dell'albero, si campionano m covariate delle p disponibili e si utilizzano quelle per definire la regola decisionale che divide il nodo in due sotto-nodi. Si ripete questa operazione fino al raggiungimento dei nodi terminali, che vengono definiti così, quando un nodo contiene un minimo di osservazioni prefissato. Si ottiene, quindi, una collezione di

alberi $T_1, ...T_B$ meno correlati tra loro, in modo da ridurre la varianza ed evitare l'overfitting. Nel caso del dataset *house heating* le covariate considerate sono l'azione a_t e lo stato s_t, l'obiettivo del random forest è quello di prevedere i valori y_t in 4.5; per l'implementazione è stato utilizzato il comando "randomForest" in R. Gli iperparametri da scegliere sono:

- B: numero di campioni bootstrap, che è stato lasciato come quello preimpostato, ovvero i campioni sono pari al numero delle osservazioni;
- m: numero di variabili da campionare in ogni nodo dell'albero. Nel caso della regressione il valore di default è pari a $m = \frac{p}{3}$, nel nostro caso è pari a 1, una sola variabile campionata ad ogni split;
- *ntree*: numero di alberi, che è stato impostato pari a 200, diminuendo il valore di default che è pari a 500, per motivi computazionali;
- *nodesize*: numero di osservazioni minimo che contengono i nodi terminali, è stato lasciato il valore di default pari a 5.

Anche per il modello random forest è stato costruito il grafico della media della reward cumultata giornaliera per scegliere il numero di iterazioni, in Figura 4.7. Osservando che, dopo pochi iterazioni, i valori tendono a stabilizzarsi e per limitare il carico computazionale, il numero di iterazioni è stato impostato a 50.

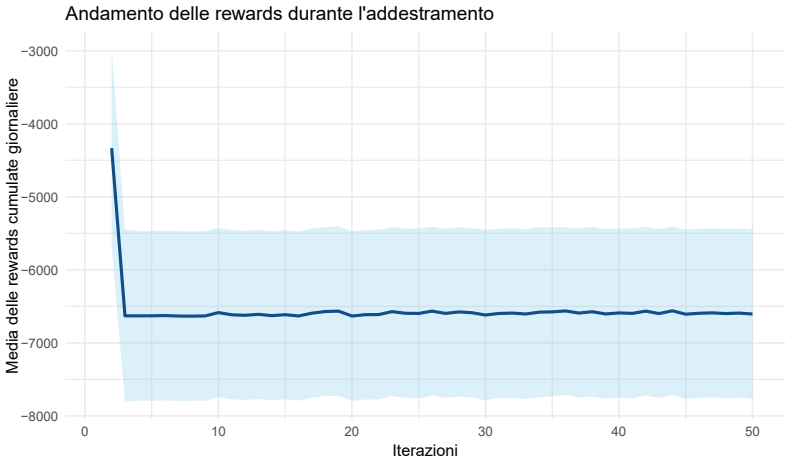

Figura 4.7: Media delle rewards cumulate giornaliere per ogni iterazione, tramite random forest.

Reti neurali

Un'ulteriore modello utilizzato nell'applicazione per prevedere i valori Q è stata una **rete neurale**. Applicando le reti neurali all'algoritmo di fitted Q-iteration, si parla

di **Neural fitted Q-iteration** (NFQ). In NFQ, l'algoritmo con cui vengono stimati i valori della funzione valore-azione Q è una rete neurale. Generalmente, il numero di layer è superiore a 1, portando quindi a considerare reti neurali *multi-layer*.

Una rete neurale è un modello di machine learning che prende decisioni in modo simile al cervello umano, utilizzando processi che imitano il modo in cui i neuroni lavorano insieme e si attivano per identificare fenomeni, valutare opzioni e trarre conclusioni. In questa applicazione, viene uilizzata una rete neurale classica, una *feed forward neural network* costituita da più livelli di nodi o neuroni artificiali: un layer di input, due hidden layer e un layer di output. In una rete neurale, gli input sono i dati a disposizione che tramite i neuroni vengono pesati in base alla loro importanza e sommati all'interno di ogni neurone. Successivamente, questa somma viene elaborata in base alla funzione di attivazione scelta, ottenendo il valore di output specifico al problema trattato. L'architettura di una rete è rappresentata dal numero di layer e di nodi in ogni layer determinando la complessità della rete stessa. All'aumentare del numero di layer e nodi, la complessità della rete neurale e, dunque, del modello aumenta. L'aumento della complessità favorisce l'insorgenza di potenziali problemi, come l'*overfitting*. L'elaborazione dei dati avviene all'interno degli hidden layer, in cui ogni nodo combina linearmente i segnali ricevuti dai nodi del layer precedente e, successivamente, applica una funzione di attivazione, in modo da introdurre non linearità nel modello. L'informazione all'interno di una rete si propaga in modo *forward*, ovvero dall'input all'output. Una volta arrivati all'output si passa alla fase di *back-propagation* in cui si calcola l'errore tra i valori previsti e quelli osservati. Successivamente, l'errore si usa per valutare le derivate rispetto ai pesi che, verranno poi utilizzate nella fase di aggiornamento dei pesi (*gradient descent).*

La rete sviluppata nel caso dell'applicazione del dataset house heating è costituita da:

- un layer di input formato da 2 nodi che rappresentano le covariate, stato e azione, presenti nel dataset;
- 2 hidden layer densi: il primo formato da 64 nodi, mentre il secondo da 32 nodi. In questi layer, la funzione di attivazione corrisponde alla funzione *ReLu*;
- un layer di output che, essendo un problema di regressione, è composto da un singolo nodo in cui si utilizza, come funzione di attivazione, la funzione identità.

Come funzione di perdita è stato utilizzato il *mean squared error* e come optimizer *ADAM*.

Anche in questo caso il numero di iterazioni da selezionare nel modello è stato fissato a 100, sulla base di quanto evidenziato dal grafico in Figura 4.8. Nel grafico è stata rappresenta la media della reward cumulata giornaliera per ogni iterazione e l'intervallo di confidenza associato. Nel caso delle reti, l'intervallo di confidenza assume valori molto piccoli che non vengono evidenziati nel grafico.

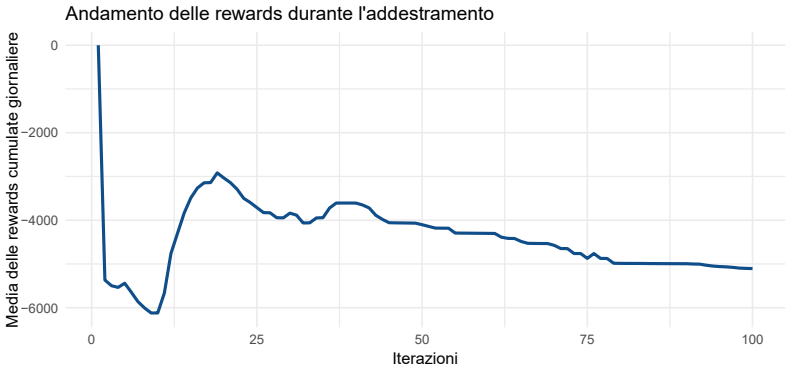

Figura 4.8: Media delle rewards cumulate giornaliere per ogni iterazione, tramite rete neurale.

4.2.3 Risultati ottenuti

In questa sezione, vengono illustrati i risultati ottenuti tramite l'implementazione dell'algoritmo di *fitted Q-iteration* utilizzando i metodi di regressione descritti precedentemente, focalizzandosi sul metodo che ha fornito risultati più soddisfacenti. Per la valutazione dei diversi approcci considerati, vengono calcolati i costi giornalieri e totali derivanti dalle azioni selezionate. Tali azioni sono determinate applicando il modello di apprendimento finale ottenuto all'ultima iterazione dell'algoritmo. Per calcolare i costi e aggiornare gli stati secondo la nuova policy individuata, il modello selezionato al termine dell'algoritmo FQI è stato applicato al dataset di test. Il test set è composto dagli ultimo 20 giorni dei 60 presenti nel dataset completo, ogni giorno è considerato un episodio indipendente e ciascun episodio è formato da 48 osservazioni, una per ogni mezz'ora del giorno. L'aggiornamento degli stati e la selezione delle azioni sono stati effettuati considerando, per ogni episodio, come stato iniziale lo stato osservato nel dataset al tempo $t = 0$. In questo modo, ogni episodio parte dallo stato registrato nel dataset, senza tener conto dello stato ottenuto nell'ultima mezz'ora del giorno precedente: ciò consente di mantenere gli episodi indipendenti tra di loro. Per generare gli stati successivi, nell'arco della giornata è stata, invece,

determinata l'azione ottimale in base allo stato corrente, la quale è stata poi applicata per calcolare lo stato successivo. Lo stato successivo è stato determinato in base allo stato precedente, all'azione ottima selezionata dal modello e da diversi parametri che permettono di calcolare la perdita e il mantenimento del calore nell'abitazione, oltre alla potenza erogata dal sistema di riscaldamento. In questo modo si è ottenuto l'aggiornamento degli stati seguendo la nuova policy individuata dall'algoritmo con la possibilità di calcolare i costi giornalieri e totali, secondo l'Equazione 4.1. Il prezzo del riscaldamento varia in base alla temperatura impostata all'azione selezionata e all'orario di emissione del riscaldamento. L'implementazione dell'algoritmo è disponibile in Appendice 4.3.

Nella tabella seguente vengono confrontati i risultati ottenuti con i modelli utilizzati, in particolare si mostrano i quantili delle temperature ottenute secondo la nuova policy all'interno della casa e il numero di volte che si viola il vincolo di mantenimento della temperatura all'interno dell'intervallo $[17.5°, 18.5°]$. Con "quantili temperatura" si intende il valore minimo, il quantile (0.25), la mediana (0.5), il quantile (0.75) e il valore massimo osservati. Il parametro λ inserito della tabella è il coefficiente che controlla la penalizzazione della reward in 4.4.

λ	**Modello**	**Quantili temperatura**	$< \mathbf{17.5°}$	$> \mathbf{18.5°}$
600	Regressione lineare	$2.12°, 2.68°, 4.16°, 7.77°, 16.40°$	960	0
	Thin plate spline	$14.20°, 17.68°, 17.70°, 17.75°, 18.56°$	21	1
	Random forest	$14.20°, 17.90°, 17.92°, 17.95°, 18.02°$	131	0
	Rete neurale	$14.20°, 17.87°, 17.88°, 17.89°, 18.00°$	20	0

Tabella 4.1: Risultati ottenuti con i diversi modelli per $\lambda = 600$.

Come già detto in precedenza e, come si nota dalla Tabella 4.1, la regressione lineare non è un buon metodo per questo tipo di problema, infatti, non riesce a riportare la temperatura interna ad un valore ottimale, restando sempre inferiore ai 18°. Lo stesso risultato si può osservare nel grafico in Figura 4.9, in cui viene rappresentato l'andamento della temperatura interna nel tempo, in cui in nessun istante temporale la temperatura è compresa nei vincoli di sicurezza considerati.

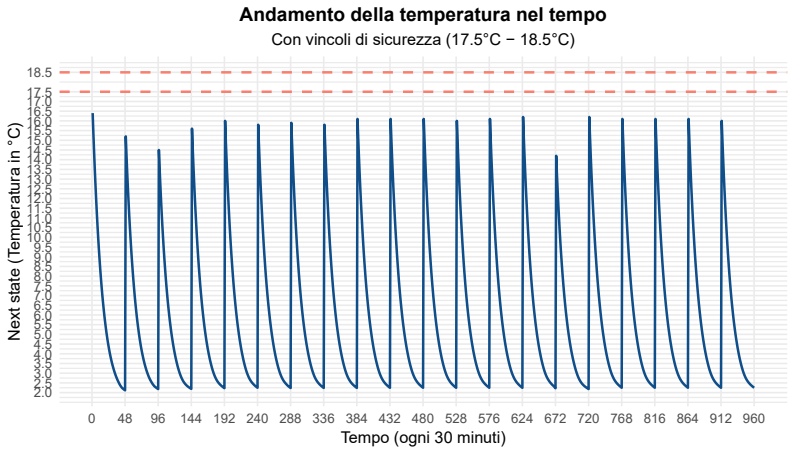

Figura 4.9: Andamento della temperatura interna tramite regressione lineare.

Per quanto riguarda gli altri modelli considerati, si osserva che i valori dello stato, riportati in Tabella 4.1, partono da una temperatura minima di 14.20°. Tale valore compare in tutti i modelli poiché corrisponde alla temperatura iniziale di un dato episodio. In particolare questo valore dello stato caratterizza l'inizio del cinquantacinquesimo giorno nel dataset completo, che rappresenta il quindicesimo giorno nel test set. Come già detto, per mantenere l'indipendenza tra gli episodi, si è scelto di far partire ciascun giorno dallo stato effettivamente osservato nel dataset, che in tutti casi risulta inferiore a 17.5°. Di conseguenza, nei risultati si osservano almeno 20 sforamenti al di sotto di questa soglia.

Il grafico in Figura 4.10 mostra l'andamento della temperatura interna (dello stato) per il modello thin plate spline. I picchi verso il basso corrispondono allo stato all'inizio di ogni giorno e sono pari a 20. Dal secondo stato di ciascun giorno, il modello riesce a riportare la temperatura all'interno dei vincoli di sicurezza ed a mantenerla abbastanza stabile per tutta la giornata. Si osserva un unico sforamento al di sopra dei 18.5°, in corrispondenza dello stato immediatamente successivo al primo nel secondo giorno di test. Attraverso questo modello la temperatura, nell'arco della giornata, oscilla molto di più rispetto alla temperatura osservata con gli altri modelli, grazie a questo si riescono a ottenere costi più bassi.

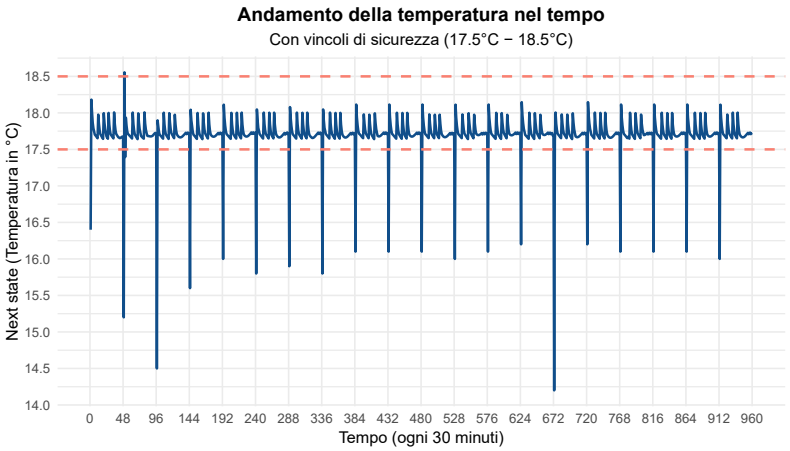

Figura 4.10: Andamento della temperatura interna con il modello thin plate spline.

Il grafico in Figura 4.11 riporta l'andamento delle temperature nell'arco dei 20 giorni per il modello random forest. In questo caso, si osservano numerosi sforamenti al di sotto della soglia di 17.5° (131 in totale), suggerendo che la nuova policy incontri più difficoltà nell'aumentare la temperatura durante la giornata. Inoltre, la temperatura massima raggiunta si mantiene intorno ai 18° e non supera mai il limite superiore consentito.

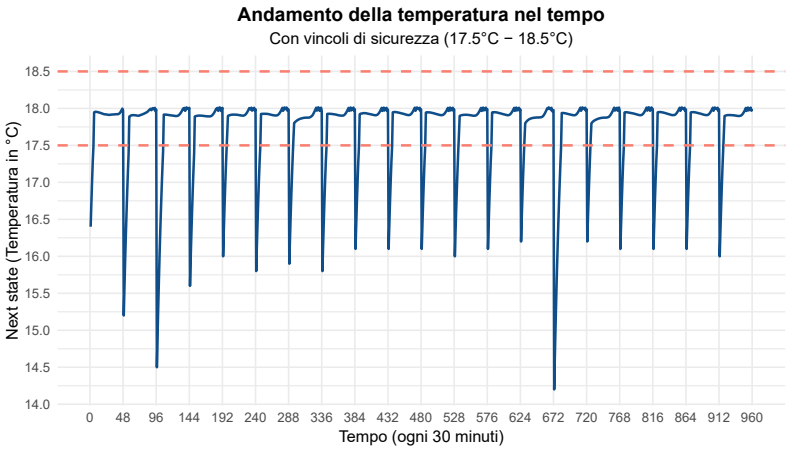

Figura 4.11: Andamento della temperatura interna con il modello random forest.

Per quanto riguarda il modello con la rete neurale il grafico dell'andamento della temperatura nel tempo è quello rappresentato in Figura 4.12. Si nota che, subito dopo il valore di temperatura registrato all'inizio della giornata, questa tende a stabilizzarsi intorno ai 18°, senza ulteriori variazioni significative. Analizzando la distribuzione dei quantili si osserva infatti che già a partire dal primo quantile (0.25) i valori si avvicinano molto alla temperatura di riferimento, senza mai violare i vincoli, né inferiori né superiori.

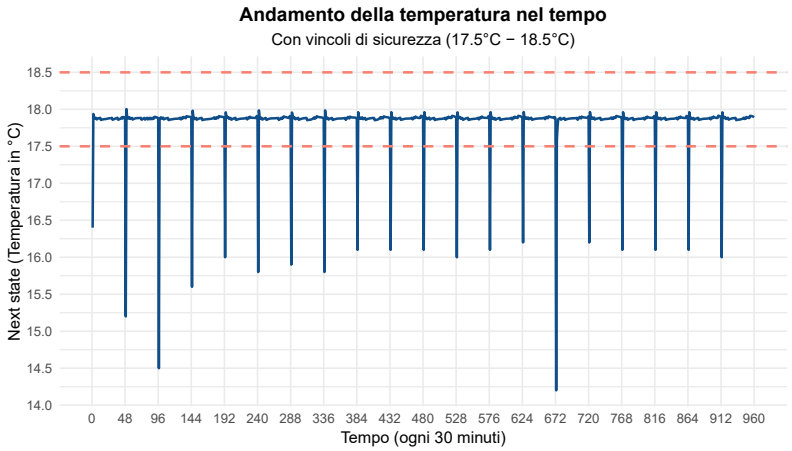

Figura 4.12: Andamento della temperatura interna con il modello rete neurale.

Nella Tabella 4.2 sono riportate le azioni selezionate applicando l'algoritmo FQI con i diversi modelli considerati. Un modello può essere ritenuto efficace se è in grado di scegliere l'azione più appropriata al momento opportuno. La tabella mostra le azioni effettivamente adottate nel test set e il numero di volte in cui ciascuna di esse è stata selezionata. Le azioni possibili variano da 0° a 70°, con incrementi di 0.5°.

Modello	**Azioni selezionate**	**Numerosità**
Regressione lineare	0°	960
Thin plate spline	0°, 15.5°, 16°, 16.5°, 17°, 17.5°, 18°, 18.5°, 23.5°, 24°, 24.5°, 26.5°, 45.5°, 48°, 49°, 50.5°, 51.5°, 52.5°, 55.5°, 70°	1, 13, 42, 41, 81, 142, 556, 1, 15, 46, 1, 1, 1, 2, 7, 3, 1, 2, 1, 3
Random forest	17.5°, 18°, 18.5°, 21°, 21.5°, 22°, 25°, 25.5°, 26°	45, 43, 735, 10, 110, 3, 3, 8, 3
Rete neurale	16.5°, 17°, 17.5°, 18°, 18.5°, 20°, 21°, 41.5°, 45°, 46.5°, 48°, 49.5°, 51.5°, 54.5°, 61°, 69.5°	3, 14, 1, 325, 595, 1, 1, 1, 2, 7, 3, 1, 2, 1, 1, 2

Tabella 4.2: Azioni selezionate e la numerosità associata.

Come si può osservare dalla tabella, la regressione lineare non rappresenta un modello ottimale, poiché non è in grado di selezionare alcuna azione diversa da 0°. I modelli che, invece, mostrano una maggiore varietà di scelte sono il thin plate spline e la rete neurale, che selezionano azioni distribuite lungo tutto intervallo disponibile, con una concentrazione più elevata in prossimità dei valori di 18°. Il random forest,

al contrario, restringe le proprie scelte all'intervallo $[17.5°, 26°]$, concentrandosi in particolare sulle temperature pari a 18.5° e 21.5°.

Per la valutazione dei modelli impiegati nell'algoritmo FQI, oltre all'analisi del valore degli stati, sono stati considerati i costi sostenuti applicando le azioni selezionate dal modello in ciascuno stato, valutandoli sui 20 giorni inclusi nel test set. La Tabella 4.3 mette in evidenza la differenza nei costi complessivi ottenuti adottando la nuova policy rispetto a quelli derivanti dall'applicazione della policy *safe-by-design*.

Modello	**Riduzione complessiva dei costi**
Regressione lineare	−84582.88 €
Thin plate spline	−1570.38 €
Random forest	+308.62 €
Rete neurale	−685.38 €

Tabella 4.3: Riduzione costi.

Per la regressione lineare si ottiene, ovviamente, una notevole riduzione del costo complessivo, poiché l'unica azione ottima osservata è pari a 0 e questo produce inevitabilmente costi nulli, ma ciò equivale ad interfacciarsi con una distruzione del sistema. Sia per il modello thin plate spline, che per la rete neurale si ottiene una riduzione di costi utilizzando la nuova policy, in particolare con le smoothing spline si ottengono i risultati migliori. Mentre per il random forest si ottiene, anche se non di molto, un aumento dei costi sostenuti.

Di seguito sono riportati i grafici della densità dei costi giornalieri ottenuti con la nuova policy e quelli ottenuti con la policy *safe-by-design*. Per il modello thin plate spline, mostrato in Figura 4.13, si osserva come i costi giornalieri derivanti dall'applicazione della nuova policy risultino sempre inferiori a quelli ottenuti tramite la policy di riferimento. La linea verticale che interseca le due densità rappresenta la mediana dei costi giornalieri: nel caso della nuova policy la mediana è di 4162.25€, mentre per la policy originale risulta pari a 4227.53€.

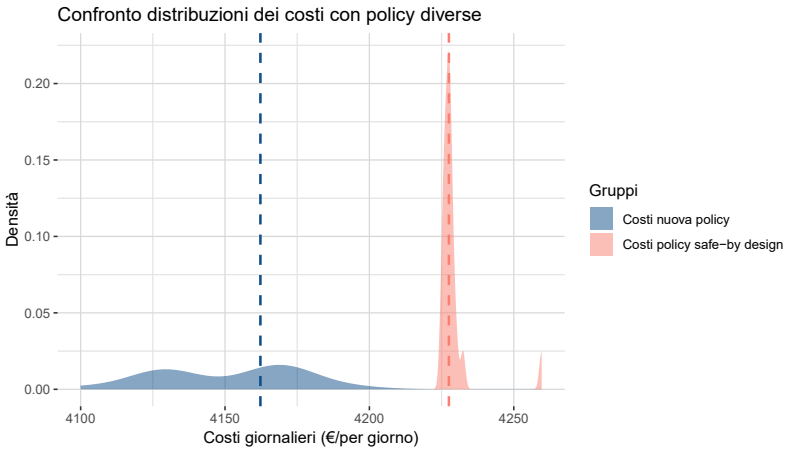

Figura 4.13: Densità dei costi giornalieri per la nuova policy e quella *safe-by-design* con il modello thin plate spline.

Per il modello random forest, invece, in Figura 4.14, si rileva il fenomeno opposto: i costi con la nuova policy risultano superiori rispetto a quelli ottenuti con la policy originale, ad eccezione di una singola giornata, in cui si registrano costi più elevati con la policy *safe-by-design*. Analizzando i valori giornalieri, emerge che si tratta comunque di cifre molto simili tra loro, con differenze limitate a poche decine di euro. La mediana dei i costi giornalieri tra le due policy, infatti, si discosta solamente di circa 15€.

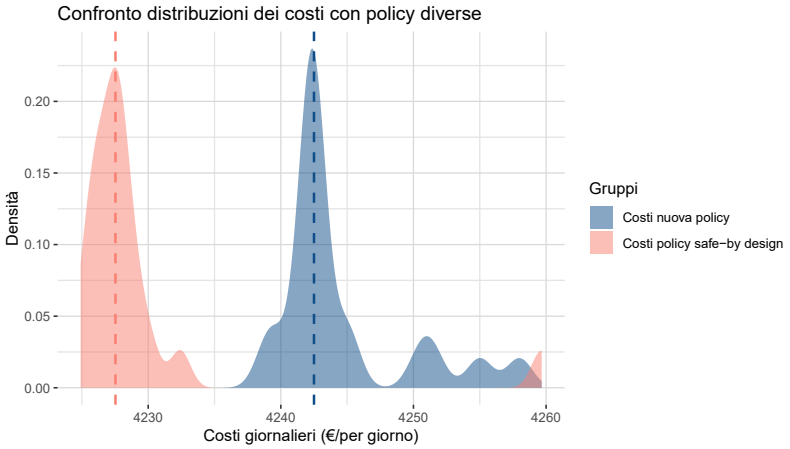

Figura 4.14: Densità dei costi giornalieri per la nuova policy e quella *safe-by-design* con il modello random forest.

Per il modello con la rete neurale si osservano costi giornalieri inferiori con la nuova policy, come mostrato in Figura 4.15. Solo per una giornata si osservano costi molto simili a quelli che si ottengono con la policy originale. Le mediane dei costi per le due differenti policy si discostano solamente di circa 33€.

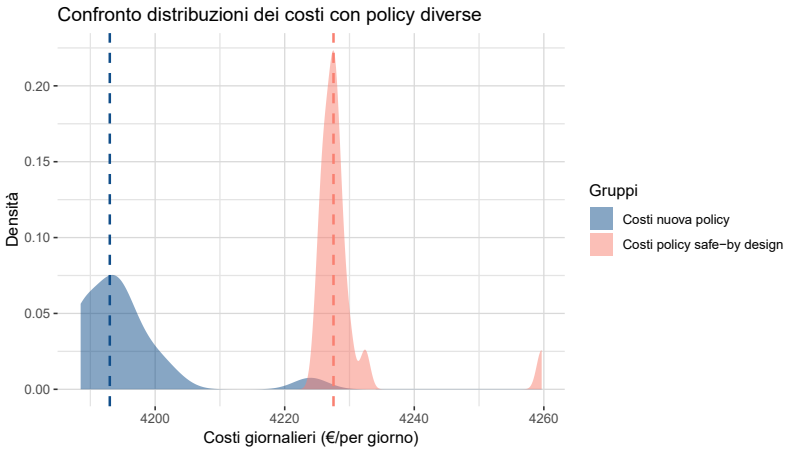

Figura 4.15: Densità dei costi giornalieri per la nuova policy e quella *safe-by-design* con il modello rete neurale.

Riepilogo e risultati sul dataset completo

In questo paragrafo sono stati confrontati quattro modelli di apprendimento supervisionato impiegati all'interno dell'algoritmo FQI, con l'obiettivo di stimare in maniera ottimale la reward cumulata e, di conseguenza, selezionare azioni efficaci che conducano a stati dell'ambiente adeguati al contesto e a una riduzione dei costi. Il confronto tra i modelli è stato effettuato considerando la diminuzione dei costi ottenuta, la capacità di individuare azioni promettenti e livelli di temperatura adeguati.

La regressione lineare, come già evidenziato, è il modello meno efficace in quanto non è in grado di cogliere la relazione tra le covariate considerate e la reward, selezionando unicamente azioni pari a 0°. Il modello con cui si ottiene il miglior compromesso tra il rispetto dei vincoli e la riduzione dei costi è il modello additivo con regressione **thin plate spline**; il quale supera i limiti della temperatura solo in due casi (di pochi decimi di grado) e permette di ottenere costi più bassi rispetto a tutti gli altri modelli. Anche con il modello basato su rete neurale produce risultati soddisfacenti: in questo caso i vincoli della temperatura non vengono mai violati sebbene i costi risultino superiori rispetto a quelli del modello con spline e più vicini a quelli osservati con la policy *safe-by-design*. Infine, il modello random forest è quello che produce risultati meno soddisfacenti, escludendo la regressione lineare, sia in termini di violazione dei vincoli, che in termini di costi. Questo modello, infatti, comporta costi superiori rispetto a quelli osservati e mostra una limitata capacità di mantenere la temperatura target.

Considerando come modello ottimale quello ottenuto tramite la regressione *thin plate spline*, esso è stato successivamente impiegato per valutare le sue prestazioni sull'intero dataset, composto da 60 giorni. Tenendo conto dell'aggiornamento degli stati e della selezione delle azioni secondo la nuova policy, si sono ottenuti i seguenti risultati.

- Il valore minimo osservato dello stato è 13.70°, i quantili rispettivamente di livello (0.25, 0.5 e 0.75) sono: 17.68°, 17.71° e 17.76° e il valore massimo è di 18.56°.
- 65 volte si supera il livello di temperatura minimo consentito (di 17.5°), delle quali 60 volte sono gli stati iniziali di ogni giorno osservati nel dataset originale. Solo 5 volte si osservano quindi violazioni del limite inferiore e si osservano sempre in corrispondenza della seconda mezz'ora della giornata.
- 2 volte si supera il limite di temperatura massimo preimpostato (di 18.5°) ma in modo relativo, per pochi centesimi di grado.
- Le azioni selezionate con relativa numerosità sono le seguenti: 0° (2), 14.5° (1), 15° (3), 15.5° (23), 16° (147), 16.5° (124), 17° (254), 17.5° (435), 18° (1623), 18.5° (5), 23.5° (49), 24° (147), 24.5° (1), 25.5° (1), 26.5° (3), 28° (1), 29° (1), 41° (1), 42.5° (2), 45.5° (3), 48° (2), 49° (13), 50.5° (5), 51.5° (8), 52.5° (8), 54° (2), 55.5° (3), 57.5° (2), 70° (11).

I livelli di stato raggiunti per tutti i 60 giorni sono molto soddisfacenti, nonostante i piccoli sforamenti dai vincoli considerati non si osservano ulteriori anomalie. Inoltre, si osserva un importante riduzione di costi totali rispetto ai costi sostenuti tramite la policy *safe-by-design*, pari a -4547.75€. In Figura 4.16, viene presentato il grafico delle densità dei costi giornalieri, confrontando le due policy. Si nota come l'applicazione dell'approccio proposto porta a una netta riduzione dei costi energetici rispetto alla politica safe-by-design, anche se si distribuiscono in un intervallo più ampio che va da circa 4100€ a 4210€ giornalieri.

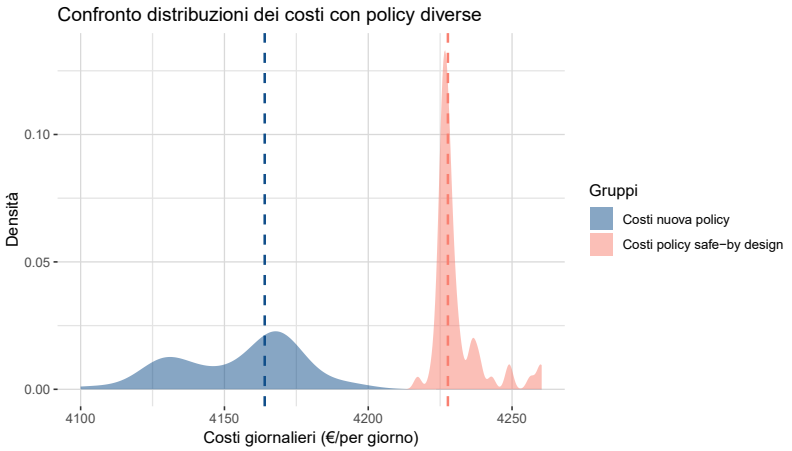

Figura 4.16: Densità dei costi giornalieri per le due differenti policy.

4.2.4 Analisi e modifica dei parametri del modello *thin plate spline*

Per approfondire i risultati ottenuti tramite il miglior modello selezionato, ovvero tramite thin plate spline, si è deciso di modificare alcuni parametri nell'implementazione, per verificare la stabilità dei risultati ottenuti e verificare la bontà del modello.

Variazione del parametro λ

Come primo intervento, si è scelto di modificare il valore del parametro λ all'interno della definizione della reward, in modo da controllare se ulteriori valori di questo parametro potessero consentire l'ottenimento di risultati migliori. Nella tabella seguente, in Tabella 4.4, si riportano i risultati dell'algoritmo considerando il valore di λ migliore (pari a 600) e altri due valori estremi del parametro (300 e 1000). Ulteriormente, si riportano la riduzione dei costi sui 20 giorni del test set rispetto a quelli ottenuti con la policy *safe-by-design*, i valori assunti dallo stato (i.e. le temperature osservate) e il numero di volte in cui vengono violati i vincoli di sicurezza.

λ	**Costi**	**Quantili temperatura**	$<$ **17.5°**	$>$ **18.5°**
300	-689.38 €	14.20°, 17.61°, 17.73°, 17.87°, 18.56°	20	1
600	-1570.38 €	14.20°, 17.68°, 17.70°, 17.75°, 18.56°	21	1
1000	-1288.88 €	14.20°, 17.74°, 17.75°, 17.78°, 18.15°	20	1

Tabella 4.4: Risultati ottenuti variando il valore di λ.

Dalla Tabella 4.4 si nota che considerando i nuovi valori del parametro λ si ottengono comunque risultati soddisfacenti, con una buona diminuzione dei costi e nessuna violazione dei limiti di sicurezza.

Nella Tabella 4.5, vengono mostrate le azioni selezionate dall'algoritmo con la relativa numerosità.

λ	**Azioni**	**Numerosità**
300	13.5°, 15°, 15.5°, 16°, 16.5°, 17°, 25°, 25.5°, 45.5°, 48°, 49.5°, 50.5°, 52°, 53.5°, 57°, 70°	1, 18, 213, 209, 212, 84, 168, 35, 1, 2, 7, 3, 1, 2, 1, 3
600	0°, 15.5°, 16°, 16.5°, 17°, 17.5°, 18°, 18.5°, 23.5°, 24°, 24.5°, 26.5°, 45.5°, 48°, 49°, 50.5°, 51.5°, 52.5°, 55.5°, 70°	1, 13, 42, 41, 81, 142, 556, 1, 15, 46, 1, 1, 1, 2, 7, 3, 1, 2, 1, 3
1000	15.5°, 16°, 16.5°, 17°, 17.5°, 18°, 18.5°, 23.5°, 45°, 47°, 48.5°, 49.5°, 50.5°, 51.5°, 54°, 62.5°, 70°	1, 10, 4, 18, 26, 560, 320, 1, 1, 2, 7, 3, 1, 2, 1, 1, 2

Tabella 4.5: Azioni ottenute variando il valore di λ.

Le azioni selezionate dai modelli considerando i diversi valori di λ sono abbastanza simili, variano in tutto il range disponibile, concentrandosi sui valori vicini alla temperatura target.

Nei grafici in Figura 4.17 e 4.18 è illustrato l'andamento dello stato nell'arco dei 20 giorni, rispettivamente per $\lambda = 300$ e $\lambda = 1000$. Si osserva che i due grafici presentano comportamenti differenti: con $\lambda = 300$ la temperatura oscilla sensibilmente tra i 17.5° e poco oltre i 18°, mentre con $\lambda = 1000$ essa si mantiene costante intorno ai 17.8° per l'intero intervallo temporale.

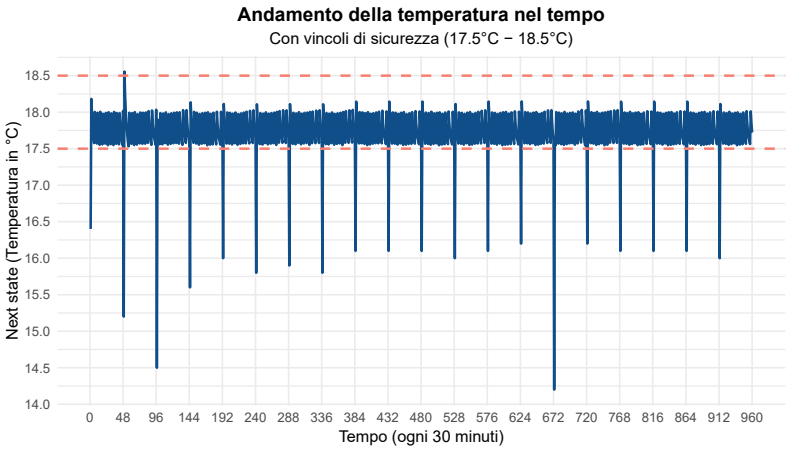

Figura 4.17: Andamento della temperatura nel tempo, con $\lambda = 300$.

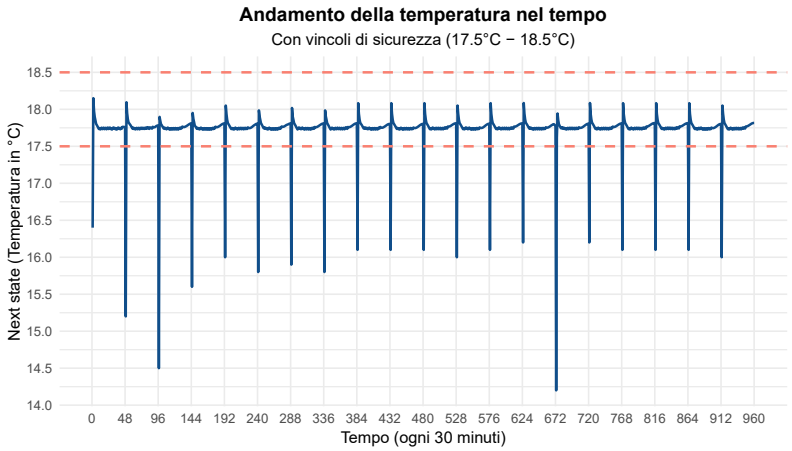

Figura 4.18: Andamento della temperatura nel tempo, con $\lambda = 100$.

Variazione della tipologia di regressione spline

Come ulteriore estensione, sono stati implementati diversi modelli additivi, basati su differenti tipologie di spline, al fine di analizzare i risultati e valutare il comportamento dell'algoritmo. In particolare, sono state considerate **spline cubiche**, **natural spline** e **B-spline**, mantenendo il valore del parametro λ pari a 600. Per tutte le tre nuove tipologie di spline sono stati fissati 3 nodi, posizionati sui quantili della distribuzione della variabile stato e di quella di azione. Il modello additivo impiegato per tutte le tipologie di spline include, come covariata, anche il termine di interazione tra stato e azione senza nessuna trasformata. In Tabella 4.6, sono riportati i risultati ottenuti, evidenziando il modello selezionato come migliore — quello con **thin plate spline** — in modo da consentire un confronto diretto con le altre tipologie.

Tipologia spline	**Costi**	**Quantili temperatura**	$< \mathbf{17.5}°$	$> \mathbf{18.5}°$
Spline cubiche	-7530.38 €	14.20°, 16.49°, 16.57°, 16.61°, 19.21°	911	12
Natural spline	-11325.88 €	14.20°, 15.77°, 15.84°, 15.93°, 17.90°	957	0
B-spline	-967.38 €	14.20°, 17.84°, 17.85°, 17.86°, 19.06°	20	18
Thin plate spline	-1570.38 €	14.20°, 17.68°, 17.70°, 17.75°, 18.56°	21	1

Tabella 4.6: Risultati utilizzando differenti tipi di spline.

Si osserva che tutte le tipologie di spline consentono di ottenere costi inferiori rispetto a quelli derivanti dalla policy *safe-by-design*.

Le **spline cubiche** mostrano risultati soddisfacenti solo in termini di riduzione di costi. Per quanto riguarda il rispetto dei vincoli, non sono in grado di raggiungere e individuare la temperatura target. L'algoritmo, nella maggior parte dei giorni, non è in grado di raggiungere un livello di temperatura appropriato; in altri casi, invece, si registrano valori di temperatura eccessivamente elevati che, però, non vengono mantenuti nel tempo. In Figura 4.19, viene rappresentato il grafico dell'andamento della temperatura nell'arco dei 20 giorni compresi nel test set. Per 7 giorni si osserva un picco della temperatura subito dopo il primo rilevamento della giornata e, successivamente, un forte abbassamento della temperatura.

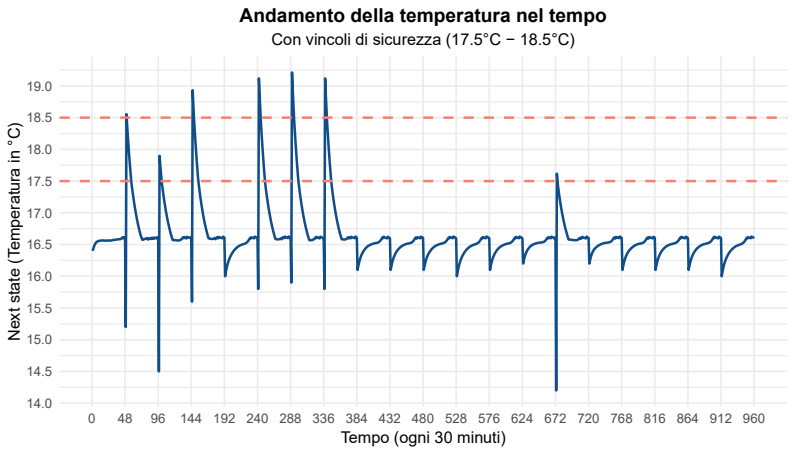

Figura 4.19: Andamento della temperatura interna con il modello *cubic spline*.

Problemi simili si riscontrano con le **natural spline**. Infatti, nella maggior parte degli istanti temporali, viene violato il vincolo inferiore di temperatura e non si raggiunge mai il valore ideale di 18°. Come mostrato in Figura 4.20, la temperatura, a partire dal valore iniziale di ogni giorno, tende a diminuire di qualche decimo di grado anziché aumentare. Si è osservato che la temperatura raggiunge i livelli target (al secondo istante temporale) solo quando lo stato iniziale è inferiore a 15°. Al secondo istante temporale la temperatura raggiunge i livelli target, e successivamente si stabilizza verso i 15°/16°. Tali valori risultano essere anche i gradi di temperatura visitati più frequentemente durante tutto il periodo di tempo considerato.

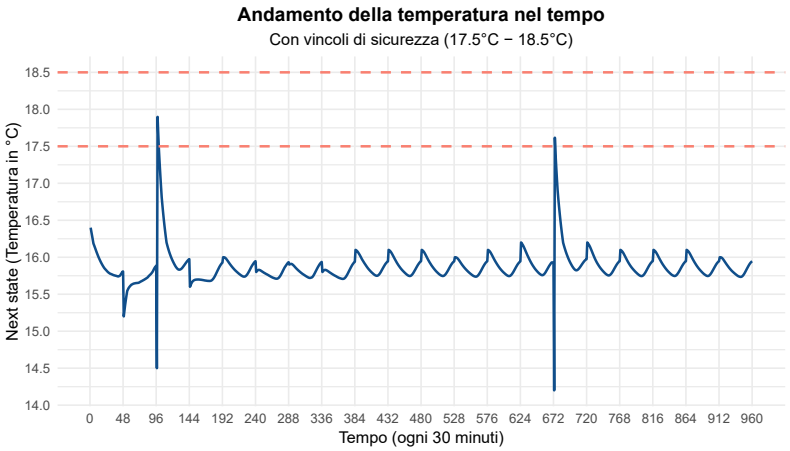

Figura 4.20: Andamento della temperatura interna con il modello *natural spline*.

Le **B-spline** presentano, invece, diverse violazioni del limite superiore di 18.5°, seppur di poco, poiché il valore massimo ottenuto è pari a 19.06°. Come si può notare dal grafico in Figura 4.21, per la maggior parte del tempo, la temperatura rimane costante ad un valore intorno ai 17.8°. Tuttavia, si osserva che, tranne per i due giorni in cui lo stato parte da una temperatura sotto ai 15°, nella prima parte della giornata si ha un picco della temperatura che supera quella limite di 18.5°.

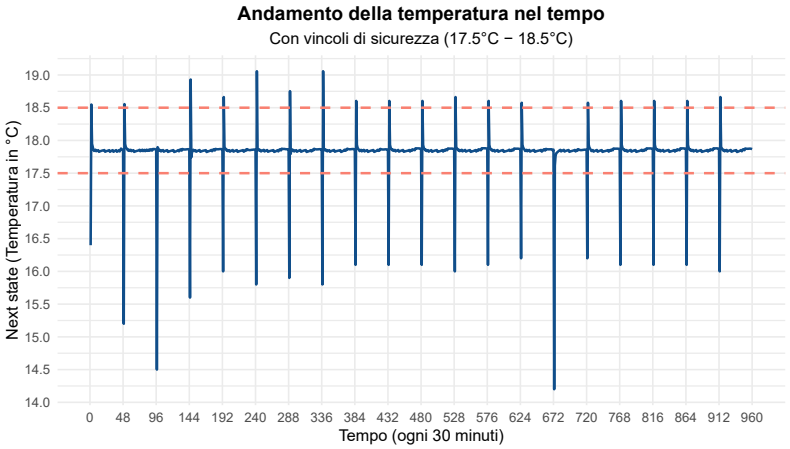

Figura 4.21: Andamento della temperatura interna con il modello *B-spline*.

In Tabella 4.7, vengono illustrate le azioni considerate nelle diverse tipologie di spline e la loro relativa numerosità.

Tipologia spline	**Azioni**	**Numerosità**
Spline cubiche	15.5°, 16°, 16.5°, 17°,70°	11, 35, 299, 608, 7
Natural spline	14.5°, 15°, 15.5°, 16°, 16.5°, 70°	3, 7, 19, 924, 5, 2
B-spline	0°, 3.5°, 6.5°, 8°, 8.5°, 9°, 17.5°, 18°, 18.5°, 19°, 19.5°, 20.5°, 51.5°, 55°, 57°, 59.5°, 62.5°, 69°, 70°	3, 1, 3, 7, 2, 2, 12, 392, 514, 2, 1, 1, 1, 2, 7, 3, 1, 2, 4
Thin plate spline	0°, 15.5°, 16°, 16.5°, 17°, 17.5°, 18°, 18.5°, 23.5°, 24°, 24.5°, 26.5°, 45.5°, 48°, 49°, 50.5°, 51.5°, 52.5°, 55.5°, 70°	1, 13, 42, 41, 81, 142, 556, 1, 15, 46, 1, 1, 1, 2, 7, 3, 1, 2, 1, 3

Tabella 4.7: Azioni ottenute con differenti tipi di spline.

In particolare, si nota che le *spline cubiche* e le *natural spline* selezionano poche azioni rispetto alle altre tipologie, focalizzandosi su temperature piuttosto basse comprese tra i 14.5° e i 16.5°. Il modello *B-spline* seleziona azioni simili a quelle ottenute tramite il modello migliore.

Come mostrato dai risultati, l'unica modifica al modello *thin plate spline* che porta a risultati soddisfacenti sia in termini di rispetto dei vincoli, sia in termini di riduzione costi, è il modello tramite *B-spline*. Il modello ottimale rimane quello già analizzato, tramite *thin plate spline*, soprattutto per la notevole riduzione di costi ottenuta.

4.3 Conformal prediction

In questa sezione vengono presentati i risultati ottenuti applicando l'algoritmo descritto in Sezione 3.4, volto alla costruzione di intervalli di previsione sui valori Q stimati tramite l'algoritmo FQI. Il dataset è stato suddiviso in: training set di 30 giorni (1440 osservazioni), calibration set di 10 giorni (480 osservazioni) e test set di 20 giorni (960 osservazioni).

Sono riportati i valori di *coverage*, la lunghezza media degli intervalli e il quantile $\hat{q}$ stimato della distribuzione della funzione *score* calcolata sul calibration set. Per *coverage* si intende la copertura media ottenuta su tutte le osservazioni del test set. Per ciascuna di esse è stato costruito un intervallo di previsione e si è calcolata la frequenza in cui tale intervallo contiene il vero valore della variabile di interesse; ossia la reward cumulata a partire dall'osservazione corrente fino all'ultimo istante

dell'episodio. Nel test set i valori della reward cumulata variano nell'intervallo $[-7603.61, -18.55]$. La *coverage* teorica è fissata al 90%, mentre nelle Tabelle 4.8 e 4.9 è riportata quella empirica stimata. La lunghezza dell'intervallo rappresenta un indicatore importante per stabilire la qualità dell'intervallo di previsione: valori ridotti corrispondono a una minore incertezza, mentre valori elevati indicano previsioni caratterizzate da maggiore variabilità.

In Tabella 4.8 sono riportati i risultati ottenuti con due approcci distinti. Il modello definito come *classico* è stato calcolato applicando l'algoritmo descritto in Sezione 3.4, mentre il modello denominato *sub-sample* è stato ottenuto campionando un sottoinsieme del calibration set, costituito da 7 giorni su 10.

Modello	**Coverage**	**Ampiezza intervallo**	**Quantile**
Classico	0.872	6545.339	3272.669
Sub-sample	0.625	4285.981	2142.991

Tabella 4.8: Statistiche sull'intervallo di previsione tramite *conformal prediction*.

Si osserva che l'utilizzo del modello *sub-sample* comporta una riduzione della coverage, accompagnata tuttavia da una diminuzione significativa dell'ampiezza media degli intervalli. Nel modello *classico* l'ampiezza degli intervalli è invece piuttosto elevata, e ciò spiega il valore di coverage relativamente alto: un intervallo che copre gran parte del range di variazione della reward cumulata tende infatti a includere quasi sempre il valore reale, producendo livelli di copertura prossimi a 1.

In Tabella 4.9 sono riportati i risultati ottenuti con due varianti dell'algoritmo. Nel modello *classico con tempo* è stata introdotta la variabile tempo come covariata all'interno del metodo di CP, definita come un indicatore che cresce progressivamente nel corso dell'episodio e si azzera al termine dello stesso. Nel modello *sub-sample*, invece, i risultati sono stati ottenuti campionando un sottoinsieme del calibration set, costituito da 7 giorni su 10, includendo la covariata tempo.

Modello	**Coverage**	**Ampiezza intervallo**	**Quantile**
Classico con tempo	0.908	5178.188	2589.094
Sub-sample con tempo	0.604	3554.346	1777.173

Tabella 4.9: Statistiche sull'intervallo di previsione tramite *conformal prediction* considerando come covariata il tempo.

In questo caso, i risultati del modello *classico con tempo* migliorano rispetto a quelli riportati in Tabella 4.8, sia in termini di coverage (0.908 vs 0.872) sia in termini di lunghezza media dell'intervallo (5178 vs 6545), che risulta più contenuta. Per il modello *sub-sample con tempo*, invece, si osserva una riduzione della coverage (0.604 vs 0.625), accompagnata però da intervalli più stretti (3554 vs 4285). In sintesi, l'introduzione della variabile tempo come covariata porta benefici evidenti nel modello *classico*, mentre per l'approccio *sub-sample* introduce un trade-off tra minore copertura e maggiore compattezza degli intervalli.

Conclusioni

Il lavoro svolto in questa tesi ha permesso di analizzare e applicare metodi di *Reinforcement Learning*, focalizzandosi su algoritmi di tipo *offline* e *off-policy*, con un particolare studio e sviluppo dell'algoritmo *Fitted Q-Iteration* combinato con diversi modelli di regressione per la stima della funzione valore-azione. Lo studio ha considerato differenti approcci di regressione, tra cui: regressione lineare, *random forest*, modello additivo (GAM) con regressione *spline* e reti neurali. Tra i modelli proposti, il più performante è risultato essere il modello additivo generalizzato, in cui è stata applicata una trasformazione di tipo *thin plate spline* alle covariate stato e azione, includendo anche l'interazione tra le due variabili. Questo modello si è rivelato capace di catturare relazioni non lineari tra le variabili indipendenti e la variabile risposta *reward*, mostrando buone performance sia in termini di rispetto dei vincoli del sistema dinamico analizzato, sia nella riduzione dei costi di gestione.

L'applicazione al dataset *House Heating* ha evidenziato l'importanza di individuare una *policy* nuova in grado di migliorare il comportamento del sistema dinamico, sia dal punto di vista dell'efficienza dei costi, sia nel mantenimento di una temperatura *target* predefinita all'interno dell'edificio, rispetto alle azioni selezionate dalla *policy safe-by design*. Con ogni modello di regressione applicato si sono individuati dei vantaggi e degli svantaggi. La regressione lineare non ha riportato nessun vantaggio non essendo in grado di cogliere la relazione tra le variabili indipendenti e la variabile risposta, portando a livelli di temperatura non adeguati. Con il modello *random forest* (RF) si sono osservati risultati soddisfacenti solo in ambito di rispetto dei vincoli preimpostati: le azioni selezionate da RF consentono infatti di mantenere la temperatura all'interno dell'intervallo $[17.5°, 18.5°]$. Tuttavia, i costi risultano più elevati rispetto a quelli selezionati tramite la *policy safe-by-design*. Le reti neurali permettono una gestione ottimale della temperatura, ma comportano costi più elevati rispetto al miglior modello individuato, ossia la regressione *thin plate spline* che si è dimostrato, attraverso ulteriori analisi sui suoi parametri, un modello stabile e robusto.

Successivamente, è stato elaborato un metodo che si basa sull'applicazione di CP ad un processo di RL per quantificare l'incertezza dei valori Q della funzione valore-azione, permettendo la costruzione di intervalli di previsione ottimali e fornendo una misura di incertezza affidabile. I risultati hanno mostrato che l'introduzione della variabile tempo come covariata all'interno dell'algoritmo FQI e nella procedura di CP, può migliorare la copertura predittiva e ridurre la variabilità degli intervalli, pur introducendo talvolta un *trade-off* tra i livelli di coverage e l'ampiezza dell'intervallo.

In sintesi, il lavoro conferma l'utilità di combinare approcci di *reinforcement learning offline* con strumenti di regressione avanzata per il controllo ottimale di sistemi dinamici complessi e come il loro utilizzo porti a una selezione di politiche più efficienti e ottimali rispetto agli approcci tradizionali. Inoltre, i metodi applicati per la quantificazione dell'incertezza delineano una strategia promettente al fine di supportare le decisioni da prendere, anche se si conferma la necessità di sviluppare ulteriori metodologie in modo da creare intervalli di previsioni più affidabili.

In generale, i risultati ottenuti sono soddisfacenti, anche se emergono alcune criticità. In primo luogo, l'applicazione si basa su un dataset simulato, i cui dati sono stati generati tramite un simulatore. Un possibile sviluppo futuro sarebbe quello di applicare i metodi analizzati a dataset reali, preferibilmente di dimensione e complessità maggiori.

A differenza del contesto simulato, in cui la formulazione degli stati a ogni passo è nota grazie al simulatore che li ha generati, nei dati reali la costruzione degli stati può essere influenzata da una variabile esogena non osservata e non stazionaria. In tale scenario, trascurare la variabile tempo come covariata all'interno dell'algoritmo FQI potrebbe condurre a stime distorte e, di conseguenza, ad una *policy* sub-ottimale. Per questo motivo, in questo lavoro, nella parte di analisi dell'incertezza tramite CP è stata introdotta la variabile tempo come covariata, sia nell'algoritmo di FQI durante la fase di training, sia nel calcolo della *score function* attraverso i valori Q della funzione valore-azione, così da valutare empiricamente l'efficacia di questa scelta. Un possibile sviluppo futuro potrebbe consistere nell'implementazione di metodi in grado di tenere conto della variabile esogena non osservata e di adattare l'algoritmo di conseguenza.

Un ulteriore limite individuato riguarda la discretizzazione delle azioni possibili considerate nell'algoritmo FQI, in modo da ridurle ad un numero finito. Un'evoluzione naturale potrebbe essere l'impiego di azioni continue, che possono essere gestite mediante algoritmi di RL *online*, come il *Proximal Policy Optimization* (PPO). Quest'ultimo mira a massimizzare la ricompensa cumulata prevista stimando una

distribuzione per la *policy* e risulta particolarmente adatto alla gestione di spazi delle azioni continui.

Infine, il metodo di CP elaborato in questa tesi per la quantificazione dell'incertezza dei valori della funzione Q mette in evidenza alcune criticità. La formulazione proposta dei valori della *score function* introduce dipendenza temporale, determinando la violazione della principale assunzione in *split conformal prediction*: la scambiabilità. Ciò accade perché ciascun *return* allo stato t include non solo la *reward* corrente, ma anche tutte quelle future, sovrapponendosi quasi completamente con il *return* successivo. Per affrontare questa limitazione, studi futuri potrebbero adottare schemi di *block conformal prediction*, sfruttando il fatto che, nello scenario classico di MDP, si può assumere l'indipendenza tra episodi. Inoltre, si potrebbero considerare definizioni alternative di *return* empirico.

Bibliografia

Agarwal, A., Jiang, N., Kakade, S. M. & Sun, W. (2019). Reinforcement learning: Theory and algorithms. Technical Report TR-2019-32, Department of Computer Science, University of Washington, Seattle, WA, USA.

Angelopoulos, A. N. & Bates, S. (2021). A gentle introduction to conformal prediction and distribution-free uncertainty quantification. *arXiv preprint arXiv:2107.07511* .

Barber, R. F., Candes, E. J., Ramdas, A. & Tibshirani, R. J. (2023). Conformal prediction beyond exchangeability. *The Annals of Statistics* **51**, 816–845.

Candelieri, A., Ponti, A., Fersini, E., Messina, E. & Archetti, F. (2023). Safe optimal control of dynamic systems: Learning from experts and safely exploring new policies. *Mathematics* **11**, 4347.

Ernst, D., Geurts, P. & Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research* **6**, 503–556.

Hastie, T., Tibshirani, R. & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. New York: Springer, 2nd ed.

James, G., Witten, D., Hastie, T. & Tibshirani, R. (2013). *An introduction to statistical learning: with applications in R*, vol. 103. Springer.

Munos, R. (2005). Error bounds for approximate value iteration. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, vol. 2. AAAI Press.

of Public Health, C. U. M. S. (2011). Thin plate spline regression. https://www.publichealth.columbia.edu/research/population-health-methods/thin-plate-spline-regression.

Oliveira, R. I., Orenstein, P., Ramos, T. & Romano, J. V. (2024). Split conformal prediction and non-exchangeable data. *Journal of Machine Learning Research* **25**, 1–38.

Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Rhurbans (2023). Reinforcement learning with q-learning. https://rhurbans.com/reinforcement-learning-with-q-learning/.

Shi, C. (2025). Statistical inference in reinforcement learning: A selective survey. *arXiv preprint arXiv:2502.16195* .

Sutton, R. S. & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press, 2nd ed.

Tibshirani, R. J., Foygel Barber, R., Candes, E. & Ramdas, A. (2019). Conformal prediction under covariate shift. *Advances in neural information processing systems* **32**.

Vovk, V., Gammerman, A. & Shafer, G. (2005). *Algorithmic learning in a random world*. Springer.

Wasserman, L. (2006). *All of nonparametric statistics*. Springer.

Wikipedia contributors (2024a). Curve fitting — Wikipedia, l'enciclopedia libera. https://it.wikipedia.org/wiki/Curve_fitting.

Wikipedia contributors (2024b). Funzione spline – Wikipedia, l'enciclopedia libera. https://it.wikipedia.org/wiki/Funzione_spline.

Zhang, Y., Shi, C. & Luo, S. (2023). Conformal off-policy prediction. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*. PMLR.

Zheng, F. & Proutiere, A. (2024). Conformal predictions under markovian data. *arXiv preprint arXiv:2407.15277* .

Appendice

Codice R

```
#librerie
library(splines)
library(dplyr)
library(tidyverse)
library(ggplot2)
library(e1071)
library(mgcv)
library(gridExtra)

#caricamento dati, creazione del dataset in un formato tabella
House<-readRDS("RECORD_30.RDS")  #dati ogni 30 minuti per 60
   giorni
House[["Hs"]]     #stato al tempo t, la temperatura al tempo t
House[["Xs"]]     #azione al tempo t
House[["CSIs"]]   #temperature esterne al tempo t

s<-seq(from=0.5, to=24, by=0.5)
min_30<-rep(s, 60)
giorno<-c()
for (i in 1:60) {giorno<-c(giorno, rep(i, 48))}

house_heat<-data.frame(
  stato<-as.vector(t(House[["Hs"]][,-49])),
  stato_succ<-as.vector(t(House[["Hs"]][,-1])),
  azione<-as.vector(t(House[["Xs"]])),
  giorno<-giorno,
  min_30<-min_30
)
```

```
28
29 colnames(house_heat)<-c("stato"  ,
30                         "stato_succ",
31                         "azione"  ,
32                         "giorno", "min_30"  )
33 #calcolo dei costi
34 tariffa_ora<-list(time=c(0,14,20, 34, 42),
35                   cost=c(1,10,5,10,1))
36
37 compute.cost <- function( xs, tariff ) {
38   COST<-c()
39   COST[1] <- 0
40   for( i in 2:48) {
41     min_30 <- i
42     ix <- which( tariffa_ora$time<=min_30)
43     ix <- rev(ix)[1]
44     COST[i] <-tariffa_ora$cost[ix] * xs[i]
45   }
46   return(COST)
47 }
48
49 hour.cost <- matrix(0,nrow=60, ncol=48)
50 for( i in 1:nrow(hour.cost)){
51   hour.cost[i,] <- compute.cost(House$Xs[i,], tariffa_ora)
52 }
53 rowSums(hour.cost)  #costi orari della policy safe
54 costo<- -as.vector(t(hour.cost))
55 done<-ifelse(house_heat$min_30==24, 1, 0)
56 house_heat<-cbind(house_heat,costo, done)
57 colnames(house_heat)<-c("stato"  ,
58                         "stato_succ",
59                         "azione"  ,
60                         "giorno"   ,
61                         "min_30"   ,
62                         "costo" ,
63                         "done")
64
65 write.csv(house_heat, "house_heat_30.csv")
```

```
66
67 #Analisi esplorativa del dataset "house heating"
68 dati<-read.csv("house_heat_30.csv")[,-1]
69 House<- RECORD_30 #dataset senza pre processing
70 dati$CSIs<-as.vector(t(House[["CSIs"]]))
71 #definizione nuova reward
72 lambda<-600
73 dati$reward_new<-dati$costo - lambda*abs(dati$stato-18)
74
75 ggplot(dati, aes(x=stato, y=reward_new))+
76   geom_point(col="salmon", pch=19)+
77   theme_minimal()+
78   xlab("Stato corrente") +
79   ylab("Reward") +
80   labs(title="Reward in funzione dello stato")
81
82 ggplot(dati, aes(x=azione, y=reward_new))+
83   geom_point(col="salmon", pch=19)+
84   theme_minimal()+
85   xlab("Azione") +
86   ylab("Reward") +
87   labs(title="Reward in funzione dell'azione")
88
89 #divisione del dataset in training e test
90 index<-1:1920
91 training<-dati[index, ] #training formato dai primi 40 giorni
92 test<- dati[-index, ] #test formato dagli ultimi 20 giorni
93
94 #inizializzazione per algoritmo FQI
95 stato_succ<-matrix(training$stato_succ, byrow=T, ncol=48, nrow
      =nrow(training)/48)
96 stato<- matrix(training$stato, byrow=T, ncol=48, nrow=nrow(
      training)/48)
97 gamma <- 0.99
98 azioni_possibili <- seq(0, 70, by = 0.5)
99 n_iter <-50
100 q_mod <- NULL
101 media_reward<-c()
```

```
102 std_reward<-c()
103 gruppi <- rep(1:(nrow(training)/48), each = 48)
104 costo_temp<-matrix(0, ncol=48, nrow=nrow(training)/48)
105 reward_temp<-matrix(0, ncol=48, nrow=nrow(training)/48)
106 mse<-c()
107 q_next_mat<-matrix(0, nrow=n_iter, ncol=length(unique(training
      $stato_succ)))
108
109 #Algoritmo Fitted Q-iteration
110 set.seed(123)
111 for (k in 1:n_iter) {
112   print(k)
113
114   target_q <- training$reward_new
115
116   if (!is.null(q_mod)) {
117     # Pre-calcolo tutti i Q-values per stati unici
118     stati_unici <- unique(training$stato_succ)
119     q_next <-  sapply(stati_unici, function(s) {
120       new_data <- data.frame(stato = s, azione = azioni_
            possibili)
121       p<-predict(q_mod, new_data)
122       out<-c(max(p),azioni_possibili[which.max(p)])
123       return (out)
124     })
125
126     idx <- match(training$stato_succ, stati_unici)
127     temp<-data.frame(stato=training$stato_succ, azione=q_next
          [2,][idx])
128
129     azione_mat<-matrix(temp$azione, ncol=48, nrow=nrow(
          training)/48, byrow=T)
130     for( i in 1:nrow(azione_mat)){
131       costo_temp[i,] <- compute.cost(azione_mat[i,], tariffa_
            ora)
132     }
133
134     for ( i in 1:nrow(azione_mat)){
```

```
135       reward_temp[i,]<- -costo_temp[i,] - lambda*abs(18 -
             stato[i,])}
136     reward_daily <- rowSums(reward_temp)
137     media_reward[k]<-mean(reward_daily)
138     std_reward[k]<-sd(reward_daily)
139
140     target_q[training$done != 1] <- target_q[training$done !=
           1] + gamma * q_next[1,][idx][training$done != 1]
141     q_next_mat[k,]<-q_next[1,]
142     }
143
144   #Modelli utilizzati:
145
146   #LM
147   #q_mod<-lm(target_q~stato + azione + stato*azione, data=
         training)
148
149   #SPLINES CUBICHE
150   #q_mod<-mgcv::gam(target_q~s(stato, bs="cr", k=7)+ s(azione,
          bs="cr", k=7)+ azione*stato,  data=training)
151
152   #SPLINES SMOOTHING (thin plate)
153   q_mod<-mgcv::gam(target_q~s(stato)+ s(azione)+ azione*stato,
          data=training, method="REML")
154
155   #B-SPLINE
156   #q_mod<-mgcv::gam(target_q~s(stato, bs="bs", k=7)+ s(azione,
          bs="bs", k=7)+ azione*stato,  data=training)
157
158   #NATURAL CUBIC SPLINE
159   #q_mod<-lm(target_q~ns(stato,  df=3,  knots=quantile(
         training$stato, c(0.25,0.5,0.75)))+ ns(azione, df=3,
         knots=quantile(training$azione, c(0.25,0.5,0.75)))+
         azione*stato,  data=training)
160
161   #RANDOM FOREST
162   #q_mod <- randomForest::randomForest(target_q ~ stato +
         azione, data = training, ntree = 200, nodesize = 5)
```

```
163
164   mse[k] <- mean((predict(q_mod) - target_q)^2)
165 }
166
167 # Funzione per calcolo dell'azione ottimale
168 optimal_action <- function(state) {
169   q_vals <- predict(q_mod, data.frame(stato = state, azione =
        azioni_possibili))
170   azioni_possibili[which.max(q_vals)]
171 }
172
173 #Aggiornamento del nuovo stato, azioni ottime sul test set
174 #parametri pre-impostati
175 m = 1470
176 c = 1005.4
177 M = 3600/20
178 R = 4.329 * 10^-5
179
180 h.ref <- 18
181 h0 <- 16
182 dt <- 1/2
183 Tf <- 24
184 ts <- seq(0,Tf-dt,by=dt)
185
186 Kp=17.5; Ki=5.9; Kd=1.4; max.heat=70
187
188 stato_nuovo<-numeric(nrow(test))
189 stato_nuovo[1]<- test$stato[1]
190 azione_ottima<-c()
191 stato_oss<- test$stato
192
193 for (i in 1:nrow(test)){
194   if (test$done[i]==0){
195     azione_ottima[i]<-optimal_action(stato_nuovo[i])
196     dQg = M * c * (azione_ottima[i]-stato_nuovo[i])
197     dQl = (stato_nuovo[i]-test$CSIs[i]) / R
198     stato_nuovo[i+1] = stato_nuovo[i] + (dt/(m*c)) * (dQg-dQl)
        }
```

```
199   else {
200     azione_ottima[i]<-optimal_action(stato_nuovo[i])
201     stato_nuovo[i+1]<-stato_oss[i+1]
202   }
203 }
204 
205 stato_nuovo<-stato_nuovo[-961]
206 quantile(stato_nuovo,c(0.01, 0.25,0.5,0.75, 1)) #stato
207 table(azione_ottima)  #azioni ottime
208 sum(stato_nuovo<17.5) #limite inferiore
209 sum(stato_nuovo>18.5) #limite superiore
210 
211 #Calcolo dei costi con la nuova policy
212 azione_mat<-matrix(azione_ottima, ncol=48, nrow=20, byrow=T)
213 costo<-matrix(0, ncol=48, nrow=20)
214 
215 for( i in 1:nrow(azione_mat)){
216   costo[i,] <- compute.cost( azione_mat[i,], tariffa_ora)
217 }
218 costo<-as.vector(t(costo))
219 sum(costo)      #costi toali con nuova policy
220 sum(test$costo) #costi totali con policy safe-by-design
221 sum(costo--test$costo) #differenza tra i costi
222 
223 #-------------------------------------------------
224 #Aggiornamento stato e azioni nuove sul dataset completo
225 stato_nuovo<-numeric(nrow(dati))
226 stato_nuovo[1]<- dati$stato[1]
227 azione_ottima<-c()
228 stato_oss<- dati$stato
229 for (i in 1:nrow(dati)){
230   if (dati$done[i]==0){
231     azione_ottima[i]<-optimal_action(stato_nuovo[i])
232     dQg = M * c * (azione_ottima[i]-stato_nuovo[i])
233     dQl = (stato_nuovo[i]-dati$CSIs[i]) / R
234     stato_nuovo[i+1] = stato_nuovo[i] + (dt/(m*c)) * (dQg-dQl)
        }
235   else {
```

```
236     azione_ottima[i]<-optimal_action(stato_nuovo[i])
237     stato_nuovo[i+1]<-stato_oss[i+1]
238   }
239 }
240 stato_nuovo<-stato_nuovo[-2881]
241 round(quantile(stato_nuovo,c(0.01, 0.25,0.5,0.75, 1)),2)
242 table(azione_ottima)
243 sum(stato_nuovo<17.5)
244 sum(stato_nuovo>18.5)
245 #Costi
246 azione_mat<-matrix(azione_ottima, ncol=48, nrow=60, byrow=T)
247 costo<-matrix(0, ncol=48, nrow=60)
248 for( i in 1:nrow(azione_mat)){
249   costo[i,] <- compute.cost( azione_mat[i,], tariffa_ora)
250 }
251 costo<-as.vector(t(costo))
252 sum(costo)
253 sum(dati$costo)
254 sum(costo--dati$costo)
255
256 #-----------------------------------------------------
257 #Conformal prediction
258 #Divisione del dataset in training, calibration e test
259 index<-1:1440
260 training<- dati[index, ] #30 giorni
261 cal<- dati[1441:1920, ] #10 giorni
262 test<- dati[1921:2880, ] #20 giorni
263
264 #Creazione dataset per calcolo pesi
265 oss<- seq(1:480) #sottoinsieme del training
266 azioni<-training$azione[oss]
267 stati<- training$stato[oss]
268 done<- training$done[oss]
269 csis<- training$CSIs[oss]
270 #Aggiornamento
271 m = 1470
272 c = 1005.4
273 M = 3600/20
```

```
R = 4.329 * 10^-5
h.ref <- 18
h0 <- 16
dt <- 1/2
Tf <- 24
ts <- seq(0,Tf-dt,by=dt)
Kp=17.5; Ki=5.9; Kd=1.4; max.heat=70
stati_nuovo<-numeric(length(stati))
stati_nuovo[1]<- stati[1]
azioni_ottime<-c()
stati_oss<- stati
for (i in 1:length(stati)){
  if (done[i]==0){
    azioni_ottime[i]<-optimal_action(stati_nuovo[i])
    dQg = M * c * (azioni_ottime[i]-stati_nuovo[i])
    dQl = (stati_nuovo[i]-csis[i]) / R
    stati_nuovo[i+1] = stati_nuovo[i] + (dt/(m*c)) * (dQg-dQl)
       }
  else {
    azioni_ottime[i]<-optimal_action(stati_nuovo[i])
    stati_nuovo[i+1]<-stati_oss[i+1]
  }
}
azioni_tot<- c(azioni, azioni_ottime)
y<- c(rep(1, times=480),rep(0, times=480))
dd<-data.frame(azioni=azioni_tot,  y=y)
dd$azioni<- round(dd$azioni*2)/2
dd$azioni
#Fit del modello di regressione logistica
logistica<- glm(y~azioni, data=dd, family = binomial)

#Calibration: aggiornamento azioni e q-values ottimi
optimal_action_cal <- function(state) {
  q_vals <- predict(q_mod, data.frame(stato = state, azione =
     azioni_possibili))
  out<- c(max(q_vals), azioni_possibili[which.max(q_vals)])
  out
}
```

```
stati_cal<-numeric(length(cal$stato))
stati_cal[1]<- cal$stato[1]
azioni_cal<-c()
stati_oss_cal<- cal$stato
q_val_cal<- c()
for (i in 1:length(cal$stato)){
  if (cal$done[i]==0){
    azioni_cal[i]<-optimal_action_cal(stati_cal[i])[2]
    q_val_cal[i]<-optimal_action_cal(stati_cal[i])[1]
    dQg = M * c * (azioni_cal[i]-stati_cal[i])     # gain
    dQl = (stati_cal[i]-cal$CSIs[i]) / R   # loss
    stati_cal[i+1] = stati_cal[i] + (dt/(m*c)) * (dQg-dQl)}
  else {
    azioni_cal[i]<-optimal_action_cal(stati_cal[i])[2]
    q_val_cal[i]<-optimal_action_cal(stati_cal[i])[1]
    stati_cal[i+1]<-stati_oss_cal[i+1]
  }
}

#calcolo pesi per ogni azione del calibration
previsti<-predict.glm(logistica, newdata=data.frame(azioni=
   azioni_cal), type="response")
pesi<- previsti/(1-previsti)
#calcolo della score function
giorni<- c(31,32, 33, 34, 35, 36, 37, 38, 39, 40)
score<-c()
score1<-c()
for(j in giorni ){
  reward_cal<- cal$reward_new[cal$giorno==j]
  for(i in 1:length(reward_cal)){
    score[i]<-abs(gamma*sum(reward_cal[i:length(reward_cal)])-
       q_val_cal[i])
  }
  score1<-c(score1, score)
  score<-c()
}
score1<- score1*pesi #score function pesata
n<-nrow(cal)
```

```
346 #calcolo il quantile con alpha=0.1
347 q<- ceiling((n+1)*(1-0.1))
348 score_ord<-sort(score1)
349 quantile<-score_ord[q]
350
351 #test: aggiornamento azioni e i values ottimi
352 stati_test<-numeric(length(test$stato))
353 stati_test[1]<- test$stato[1]
354 azioni_test<-c()
355 stati_oss_test<- test$stato
356 q_val_test<- c()
357 for(i in 1:length(test$stato)){
358   azioni_test[i]<-optimal_action_cal(stati_test[i])[2]
359   q_val_test[i]<-optimal_action_cal(stati_test[i])[1]
360   stati_test[i+1]<-stati_oss_test[i+1]
361 }
362
363 #intervallo
364 PI<- data.frame(lo=q_val_test-quantile, up=q_val_test+quantile
      )
365 quant<- rep(quantile, 960)
366 intervalli<- data.frame(q_values_test=q_val_test, quantile=
      quant, PI_lo=PI$lo, PI_up=PI$up )
367 #calcolo coverage
368 giorni<- unique(test$giorno)
369 reward_cum<-c()
370 reward_cum1<-c()
371 for(j in giorni ){
372   reward_test<- test$reward_new[test$giorno==j]
373   for(i in 1:length(reward_test)){
374     reward_cum[i]<- sum(reward_test[i:length(reward_test)])
375   }
376   reward_cum1<-c(reward_cum1, reward_cum)
377   reward_cum<-c()
378 }
379 coverage <- mean((reward_cum1 > intervalli$PI_lo) & (reward_
      cum1 < intervalli$PI_up))
380 #lunghezza intervallo
```

```
int_len <- round(mean( intervalli$PI_up - intervalli$PI_lo),
   5)

#-----------------------------------------------------
#Grafici
#Grafico media della reward cumulata giornaliera
ggplot_with_ci <- function(media, se,
            xlab = "Iterazioni",
            ylab = "Valore",
            title = "Andamento con intervalli di confidenza",
            level = 0.95) {
  n <- length(media)
  z_value <- qnorm(1 - (1 - level)/2)
  df <- data.frame(
    iterazione = 1:n,
    media = media,
    lower = media - z_value * se,
    upper = media + z_value * se
  )

  ggplot(df, aes(x = iterazione, y = media)) +
    geom_ribbon(aes(ymin = lower, ymax = upper),
                fill = "skyblue", alpha = 0.3) +  # IC
    geom_line(color = "dodgerblue4", size = 1) +  # media
    labs(x = xlab, y = ylab, title = title) +
    theme_minimal()
}

ggplot_with_ci(media_reward, std_reward,
    ylab = "Media delle rewards cumulate giornaliere",
    title = "Andamento delle rewards durante l'addestramento")


#grafico dell'andamento della temperatura nel tempo
dati_grafico <- data.frame(
tempo = 1:length(stato_nuovo),
stato = stato_nuovo
)
```

```
418
419 ggplot(dati_grafico, aes(x = tempo, y = stato)) +
420   geom_line(color = "dodgerblue4", linewidth = 0.8) +
421   geom_hline(yintercept = 17.5, color = "salmon", linetype = "
        dashed", linewidth = 0.8) +
422   geom_hline(yintercept = 18.5, color = "salmon", linetype = "
        dashed", linewidth = 0.8) +
423   labs(
424     title = "Andamento della temperatura nel tempo",
425     subtitle = "Con vincoli di sicurezza (17.5C - 18.5C)",
426     x = "Tempo (ogni 30 minuti)",
427     y = "Next state (Temperatura in C)"
428   ) +
429   theme_minimal() +
430   theme(
431     plot.title = element_text(hjust = 0.5, face = "bold"),
432     plot.subtitle = element_text(hjust = 0.5)
433   ) +
434   scale_x_continuous(
435     breaks = seq(0, max(dati_grafico$tempo), by = 48),
436     minor_breaks = NULL
437   ) +
438   scale_y_continuous(
439     breaks = sort(unique(c(
440       seq(floor(min(dati_grafico$stato)),
441           ceiling(max(dati_grafico$stato)),
442           by = 0.5),
443       17.5, 18.5
444     )))
445   )
446
447 #grafico della densita dei costi giornalieri
448 gruppi <- rep(1:(length(test$costo)/48), each = 48)
449 costi_oss <- tapply(-test$costo, gruppi, sum) #con policy safe
     -by-design
450 costi_new <- tapply(costo, gruppi, sum) #con nuova policy
451
452 df <- data.frame(
```

```
453   valore = c(costi_oss, costi_new),
454   gruppo = factor(rep(c("Costi policy safe-by design", "Costi
         nuova policy"),
455                       times = c(length(costi_oss), length(
                             costi_new))))
456 )
457
458 p <- ggplot(df, aes(x = valore, fill = gruppo)) +
459   geom_density(alpha = 0.5, linewidth = 0.8, color = NA) +
460   scale_fill_manual(values =
461                     c("Costi policy safe-by design" = "salmon",
462                     "Costi nuova policy" = "dodgerblue4"),
463                     name = "Gruppi") +
464   labs(title = "Confronto distribuzioni dei costi con policy
         diverse",
465        x = "Costi giornalieri (per giorno)",
466        y = "Densita") +
467   theme_bw() +
468   theme(panel.border = element_blank(),
469    panel.grid.major = element_line(color = "grey85", size =
         0.3),
470    panel.grid.minor = element_line(color = "grey90", size =
         0.2))
471 # Aggiunta mediane
472 medie <- aggregate(valore ~ gruppo, df, median)
473 p <- p + geom_vline(data = medie,
474       aes(xintercept = valore, color = gruppo),
475       linetype = "dashed", linewidth = 0.8, show.legend =
            FALSE) +
476   scale_color_manual(values =
477               c("Costi policy safe-by design" = "salmon",
478               "Costi nuova policy" = "dodgerblue4"))
479 print(p)
```

Codice Python

Il codice illustrato in questa sezione è stato utilizzato per l'implementazione dell'algoritmo NFQ, ossia una variante di FQI in cui i valori della funzione Q vengono stimati tramite una rete neurale.

```
#Librerie
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.preprocessing import StandardScaler
from torch.utils.data import DataLoader, TensorDataset
import matplotlib.pyplot as plt
!pip install --upgrade sympy

#Random seeds
np.random.seed(123)
torch.manual_seed(123)
#Caricamento e preparazione dati
dati = pd.read_csv("house_heat_30.csv").iloc[:, 1:]
lambda_val = 600
dati['reward_new'] = dati['costo'] - lambda_val * abs(dati['
   stato'] - 18)
dati["CSIs"]= pd.read_csv("CSIs_30.csv").iloc[:, 1]

index = range(1920)
training = dati.iloc[index].copy()
test = dati.drop(index).copy()

# Inizializzazione algoritmo FQI
gamma = 0.99
azioni_possibili = np.arange(0, 70, 0.5)
n_iter = 100
mse = np.zeros(n_iter)
media_reward = np.zeros(n_iter)
std_reward = np.zeros(n_iter)
gruppi = np.repeat(np.arange(len(training)//48), 48)
```

```
33
34 # Definizione della rete neurale neural network
35 class QNetwork(nn.Module):
36     def __init__(self, input_size):
37         super(QNetwork, self).__init__()
38         self.fc1 = nn.Linear(input_size, 64)
39         self.fc2 = nn.Linear(64, 32)
40         self.fc3 = nn.Linear(32, 1)
41         self.relu = nn.ReLU()
42
43     def forward(self, x):
44         x = self.relu(self.fc1(x))
45         x = self.relu(self.fc2(x))
46         x = self.fc3(x)
47         return x
48
49 scaler = StandardScaler()
50 X_train = training[['stato', 'azione']].values
51 X_scaled = scaler.fit_transform(X_train)
52 y_train = training['reward_new'].values.reshape(-1, 1)
53
54 X_tensor = torch.FloatTensor(X_scaled)
55 y_tensor = torch.FloatTensor(y_train)
56 dataset = TensorDataset(X_tensor, y_tensor)
57 dataloader = DataLoader(dataset, batch_size=256, shuffle=True)
58
59 input_size = X_train.shape[1]
60 q_mod = QNetwork(input_size)
61 criterion = nn.MSELoss()
62 optimizer = optim.Adam(q_mod.parameters(), lr=0.001)
63
64 #Definizione funzione costi
65 tariffa_ora = {'time': [0, 14,20, 34, 42], 'cost':
      [1,10,5,10,1]}
66 def compute_cost(xs, tariff):
67     cost = np.zeros(48)
68     for i in range(1, 48):
69         min_30 = i
```

```
70          ix = max([j for j, t in enumerate(tariff['time']) if t
                <= min_30])
71          cost[i] = tariff['cost'][ix] * xs[i]
72      return cost
73
74  all_unique_states = pd.concat([training['stato'], training['
    stato_succ'], test['stato'], test['stato_succ']]).unique()
75  state_to_index = {state: i for i, state in enumerate(all_
    unique_states)}
76  q_next_mat=np.zeros((n_iter, len(all_unique_states)))
77  # Fitted Q-Iteration
78  for k in range(n_iter):
79      print(f"Iteration FQI: {k+1}/{n_iter}")
80      target_q = training['reward_new'].values.copy()
81      if k > 0:
82          batch_size = 2048
83          q_next = np.zeros((2, len(all_unique_states)))
84          for i in range(0, len(all_unique_states), batch_size):
85              batch_states = all_unique_states[i:i+batch_size]
86              state_action_pairs = np.array([[s, a] for s in
                    batch_states for a in azioni_possibili])
87
88              state_action_scaled = scaler.transform(state_
                    action_pairs)
89              with torch.no_grad():
90                  q_preds = q_mod(torch.FloatTensor(state_action
                        _scaled)).numpy().flatten()
91                  q_preds = q_preds.reshape(len(batch_states),
                        len(azioni_possibili))
92
93                  q_next[0, i:i+len(batch_states)] = np.max(q_
                        preds, axis=1)
94                  q_next[1, i:i+len(batch_states)] = azioni_
                        possibili[np.argmax(q_preds, axis=1)]
95
96          # Aggiornamento target Q, stati non terminali
97          not_done = training['done'] != 1
```

```
98          idx = [state_to_index[s] for s in training['stato_succ
               ']]
99          target_q[not_done] += gamma * q_next[0, idx][not_done]
100         q_next_mat[k, :] = q_next[0, :]
101
102         #Calcolo reward giornaliere
103         temp_actions = q_next[1, idx]
104         action_mat = temp_actions.reshape(-1, 48)
105         reward_daily = []
106
107         for day_actions in action_mat:
108             day_cost = compute_cost(day_actions, tariffa_ora)
109             day_states = training.loc[training.index.
                   intersection(gruppi[np.where(action_mat == day_
                   actions)[0]]), 'stato']
110             day_reward = -np.sum(day_cost) - lambda_val * np.
                   sum(abs(18 - day_states))
111             reward_daily.append(day_reward)
112
113         media_reward[k] = np.mean(reward_daily)
114         std_reward[k] = np.std(reward_daily)
115
116     y_tensor = torch.FloatTensor(target_q.reshape(-1, 1))
117     dataset = TensorDataset(X_tensor, y_tensor)
118     dataloader = DataLoader(dataset, batch_size=256, shuffle=
           True)
119
120     # Alleno il modello
121     q_mod.train()
122     for epoch in range(5):
123         for X_batch, y_batch in dataloader:
124             optimizer.zero_grad()
125             outputs = q_mod(X_batch)
126             loss = criterion(outputs, y_batch)
127             loss.backward()
128             optimizer.step()
```