Università degli Studi di Catania

Dipartimento di Ingegneria Elettrica Elettronica e Informatica

Corso di laurea triennale in Ingegneria Informatica

Sviluppo e valutazione di modelli di machine learning per la localizzazione indoor

TESI DI LAUREA

Autore
Salvatore Emmanuel La Porta

Relatore
Chiar.mo Prof. D. Patti

Correlatore
Chiar.mo Prof. G. C. La Delfa

ANNO ACCADEMICO 2023/2024
26 luglio 2024

Salvatore Emmanuel La Porta

Sviluppo e valutazione di modelli di machine learning per la localizzazione indoor

Tesi di laurea triennale

UNIVERSITÀ DI CATANIA

26 luglio 2024

Indice

Elenco delle figure

Capitolo 1

Introduzione

Sebbene il GPS (*Global Positioning System*) sia una tecnologia affidabile per la localizzazione outdoor, la sua efficacia diminuisce significativamente all'interno degli edifici a causa della riflessione e dell'assorbimento del segnale dalle strutture circostanti. Questo fenomeno rende la localizzazione indoor una sfida ancora da affrontare, ma il suo superamento porterebbe a vantaggi significativi in settori come la sanità, la logistica e la comunicazione.

I casi di utilizzo sono molteplici: dall'individuazione della posizione dei pazienti e delle attrezzature mediche negli ospedali, alla localizzazione precisa degli oggetti da spedire in un magazzino per ordini specifici. Inoltre, è possibile inviare notifiche push ai visitatori dei musei quando si trovano vicino a un'opera d'arte, consentendo loro di accedere comodamente tramite smartphone a informazioni storiche e curiosità relative all'opera, nella lingua predefinita del dispositivo.

La presente tesi si propone di esaminare quale algoritmo di machine learning sia più adatto per migliorare la precisione della localizzazione indoor, utilizzando le informazioni disponibili. Dopodichè, verrà costruita una semplice dashboard di visualizzazione dei dati. Nel capitolo 2, verrà eseguita una revisione della letteratura scientifica, esaminando lo stato attuale delle tecnologie e degli approcci utilizzati generalmente nella localizzazione indoor. Nel capitolo 3, sarà presentata la metodologia utilizzata per confrontare gli algoritmi di posizionamento, includendo dettagli sul dataset utilizzato, l'ambiente di sviluppo e le tecniche di preparazione del dataset per l'addestramento. Nel capitolo 4, saranno sviluppati e analizzati modelli basati su algoritmi come *Random Forest*, *K-Nearest Neighbor*, *Support Vector Machines* e *XGBoost*. Nel capitolo 5, verranno confrontate e valutate le prestazioni dei vari modelli, sia singolarmente che in modo complessivo, al fine di identificare il modello migliore per la localizzazione indoor. Infine, nel capitolo 6, verrà effettuato il deploy dei modelli basati sull'algoritmo più performante ottenuto dall'analisi del capitolo precedente.

Capitolo 2

Revisione della letteratura

2.1 Caratterizzazione della posizione

Nel paper [1], sono presentate diverse tecniche per la caratterizzazione della posizione, le quali possono essere raggruppate nei seguenti tipi:

- *Basate sul tempo*: Questo approccio considera il tempo impiegato per la comunicazione tra il trasmettitore (il dispositivo utilizzato per tracciare il soggetto, denominato *beacon* o *tag*) e il ricevitore (lo strumento per registrare i valori, denominato *anchor node* o *reference node*). L'approccio ToA (*Time of Arrival*) [2] si basa su questo principio, richiedendo l'impiego di almeno 3 anchor nodes (come mostrato in Figura 2.1) per ottenere una precisa localizzazione bidimensionale in condizioni di LoS (*Line of Sight*). Tuttavia, l'implementazione di questo approccio richiede la sincronizzazione tra i nodi e si possono verificare problemi come il multipath e il rumore additivo, che possono essere affrontati attraverso l'utilizzo di TDoA (*Time Difference on Arrival*) [3].

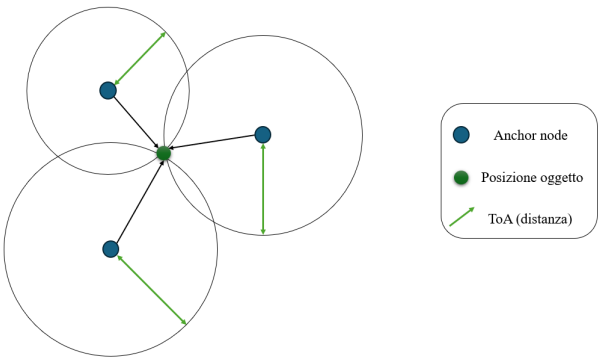

Figura 2.1: Approccio Time of Arrival (ToA)

- *Basate sulla direzione:* Questo approccio sfrutta la direzione del segnale dal tag al reference node, calcolata sfruttando la differenza di fase tra di essi. AoA (*Angle of Arrival*) è una tecnica che si basa su questo principio [4]. Per implementare AoA sono necessari almeno 2 anchor nodes (come illustrato in Figura 2.2) e una condizione di buona Line of Sight (LoS).

FIGURA 2.2: Approccio Angle of Arrival (AoA)

Nella Figura 2.3, si può osservare come ToA e AoA possano essere combinati per ridurre ulteriormente il numero di anchor nodes necessari a uno solo [5]. Tuttavia, sia AoA che la sua versione ibrida con ToA sono soggette al multipath, e i disturbi causati dalla mancanza di LoS in AoA sono ancora più accentuati rispetto a TDoA [4], [6].

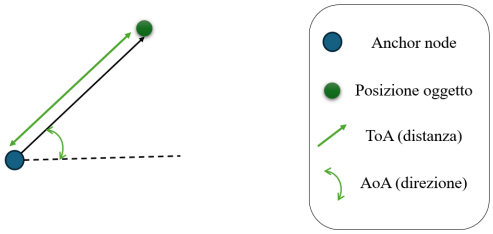

FIGURA 2.3: Approccio ibrido ToA/AoA

- *Basate sul segnale:* Questo approccio sfrutta la potenza del segnale ricevuto tra il beacon e l'anchor node per determinare la distanza tra di essi. Utilizzando un solo anchor node, è possibile associare un valore specifico di potenza del segnale a ogni posizione all'interno della stanza. Questo principio è alla base di RSSI (*Received Signal Strength Indicator*). RSSI può essere classificato come range-based o range-free. Nel primo caso, viene calcolata la distanza euclidea tra il beacon e l'anchor node, e questo valore viene utilizzato per la localizzazione attraverso algoritmi come la trilaterazione, min-max e maximum-likelihood [1]. Nel secondo caso, viene creata una mappa radio dell'ambiente indoor, rappresentata da un database di RSSI per la localizzazione, da utilizzare con algoritmi di posizionamento come il fingerprinting [7] (Figura 2.4). Una descrizione dettagliata degli algoritmi è fornita nella sezione 2.4.

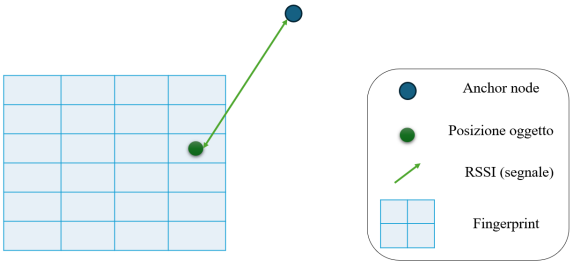

FIGURA 2.4: Approccio RSSI e fingerprinting

2.2 Panoramica delle tecnologie

Definiti i metodi per caratterizzare la localizzazione in contesti indoor, esamineremo ora le tecnologie adoperate dalla comunità scientifica per tale scopo.

Come precedentemente menzionato, l'utilizzo del GPS risulta impraticabile all'interno di edifici a causa della sua limitata precisione. Infatti, come riportato in [8], tale limitazione è principalmente attribuibile alla presenza di ostacoli che compromettono la Line-of-Sight (NLoS). Pertanto, risulta imperativo impiegare un *Indoor Positioning System* (IPS), costituito da una rete di dispositivi, al fine di localizzare oggetti e individui all'interno di ambienti chiusi.

Le tecnologie impiegate per la costruzione dei sistemi IPS, utilizzate comunemente per la localizzazione indoor, possono essere classificate come segue:

- *Campo magnetico:* Questo approccio sfrutta i campi magnetici per la localizzazione indoor, sfruttando la presenza di sensori magnetici già presenti negli smartphone. Tuttavia, la distribuzione pratica di una tecnologia basata esclusivamente su questo principio è ostacolata da varie sfide, come l'eterogeneità dei sensori, l'altezza dell'utente e le interferenze dei campi magnetici [9], [10].

- *Sensori inerziali:* Questo tipo di tecnologia integra i sensori IMU (*Inertial Measurement Unit*), come accelerometri e giroscopi, per migliorare la precisione di altre tecnologie. In [11] è stata effettuata una combinazione tra campo magnetico e sensori inerziali per risolvere un problema di localization indoor.

- *Radiofrequenza:* Tecnologie come Wi-Fi, Bluetooth e UWB (*Ultra Wideband*) sono comunemente impiegate nei sistemi IPS per la localizzazione indoor [1]. Queste tecnologie vengono approfondite nella sezione 2.3 e sono rappresentate in modo generico nella Figura 2.5.

- *Onde sonore:* Il suono è prevalentemente impiegato per il tracciamento subacqueo. Tale tecnologia può essere suddivisa in suono udibile e ultrasuono. Le onde sonore, essendo più lente delle onde elettromagnetiche, presentano un problema di sincronizzazione. Sebbene il metodo basato sul suono udibile risulti meno costoso, non è comunemente adottato nella pratica quotidiana a causa dell'inquinamento acustico [12]. D'altra parte, l'ultrasuono, con frequenze superiori ai 20 kHz, offre il vantaggio

di operare in tempo reale, con una frequenza di aggiornamento fino a 3 secondi, tuttavia la sua propagazione attraverso i muri è molto limitata [13].

- *Ottica:* Le sue applicazioni comprendono infrarossi e comunicazioni tramite luce visibile (VLC). Gli infrarossi sono ampiamente impiegati per il loro basso costo, ad esempio nel controllo remoto e nella trasmissione dati [14]. Le comunicazioni VLC, basate sui LED, sono meno influenzate da interferenze e fenomeni di multipath, garantendo una maggiore precisione rispetto alle onde radio e risultando ideali per ambienti sensibili come gli ospedali [15].

Come evidenziato da studi precedenti [11], queste tecniche possono essere combinate per migliorare ulteriormente la precisione della localizzazione indoor [16].

2.3 Focus sulle tecnologie radio-based

Un'analisi dettagliata delle tecnologie di questo genere è essenziale, considerando la loro ampia disponibilità attuale nella maggior parte dei dispositivi mobili e negli ambienti interni. In [1], [8], vengono presentate e discusse tali tecnologie, che possono essere riassunte come segue:

- *Wireless Fidelity (Wi-Fi):* Il tipo più diffuso di WLAN (Wireless Local Area Network), conforme allo standard IEEE 802.11, opera comunemente su frequenze di trasmissione di 2.4 GHz e 5 GHz al fine di garantire una maggiore velocità di trasferimento dei dati. Questa tecnologia è ampiamente integrata in una vasta gamma di dispositivi quali PC, tablet e smartphone, agevolando l'implementazione di infrastrutture per la localizzazione indoor a costi contenuti. In questo contesto, i dispositivi suddetti possono fungere da tag, mentre i router o i punti di accesso assumono il ruolo di nodi di riferimento. È possibile impiegare la tecnologia Wi-Fi con tutte le tecniche di caratterizzazione della posizione illustrate nella sezione 2.1. Il throughput medio rilevato in ambienti indoor è approssimativamente di 2.643 Mb/s [17], un valore significativo, tuttavia è importante considerare il consumo energetico associato, il quale risulta superiore rispetto ad altre tecnologie menzionate in questa trattazione [18].

- *Bluetooth Low Energy (BLE):* Basandosi sullo standard IEEE 802.15 e operando nel range di frequenze compreso tra 2.4 GHz e 2.4835 GHz, questa tecnologia è da tenere in considerazione soprattutto se si intende integrarla con il Wi-Fi, considerando il rischio di interferenze segnalato nella letteratura [18]. Tuttavia, essa riveste un ruolo significativo nell'ambito del posizionamento indoor per diversi motivi: è particolarmente efficiente, infatti i suoi beacon alimentati a batteria possono trasmettere messaggi per lunghi periodi, fino a diversi anni [19]. Pertanto, si consiglia di considerare l'implementazione di tali beacon, data anche la loro diffusione e il loro costo contenuto. Inoltre, questa tecnologia può essere utilizzata in combinazione con tecniche come RSSI e trilaterazione, offrendo un throughput medio di circa 341 kbps [20].

- *ZigBee:* Basato sullo standard IEEE 802.15.4, questo tipo di tecnologia può trasmettere su diverse bande di frequenza: 868 MHz (in Europa), 915 MHz (in Nord America e Australia) e 2.45 GHz (a livello mondiale), offrendo rispettivamente throughput di 20 kbps, 40 kbps e 245 kbps [21]. Le prime due frequenze possono contribuire a mitigare il problema delle interferenze reciproche nel caso si voglia combinare questa tecnologia con quelle precedentemente menzionate. Tuttavia, è consigliabile utilizzare ZigBee per la trasmissione di pacchetti di dati di piccole dimensioni a causa del suo limitato throughput (rispetto a BLE e soprattutto Wi-Fi). Pur garantendo un basso tasso di trasmissione dati, ZigBee offre il vantaggio di un basso consumo energetico. È stato progettato per comunicazioni a corto raggio, pertanto potrebbe non essere l'opzione ideale se si desidera utilizzare tecniche come RSSI, dato il numero significativo di nodi di riferimento necessari. Questa limitazione deriva principalmente dalla scarsa diffusione attuale di questa tecnologia nei dispositivi IoT [1].

- *Radio-Frequency Identification (RFID):* Questo sistema impiega i propri nodi di riferimento denominati lettori RFID e i relativi tag RFID, i quali vengono identificati singolarmente e possono operare a bassa frequenza (LF), alta frequenza (HF) e altissima frequenza (UHF). La tecnologia RFID si suddivide in modalità attiva e passiva. A causa del suo basso consumo energetico e del costo significativamente inferiore rispetto alla modalità attiva, il passivo è preferito per gli approcci di localizzazione interna. Qui, i tag LF e HF operano in un range inferiore a 2 metri, mentre quelli UHF possono raggiungere fino a 6 metri, rendendoli preferibili in quest'ambito [22]. Questa tecnologia si adatta adeguatamente a tutti gli esempi illustrati nella sezione 2.1, ma è consigliabile utilizzare una parametrizzazione basata su RSSI. Tuttavia, essa è soggetta alle stesse problematiche di NLoS e multipath riscontrate nelle tecnologie precedenti, oltre a essere attualmente meno diffusa nei dispositivi IoT [1].

- *Ultra-Wideband (UWB):* Risulta essere la più precisa all'interno di questa lista, con un'accuratezza fino al centimetro [23] rispetto a Wi-Fi e BLE, che offrono rispettivamente un'accuratezza di circa 10 metri [18] e 3 metri [19] (Figura 2.5). UWB si basa sullo standard IEEE 802.15.4a e opera in un range di frequenze compreso tra 3.1 e 10.6 GHz. È in grado di supportare parametri basati sia sulla distanza che sulla direzione, tuttavia è raramente utilizzato con RSSI a causa della sua minore accuratezza rispetto ad altre opzioni disponibili [1]. Tuttavia, va considerato che questa precisione elevata comporta un costo maggiore.

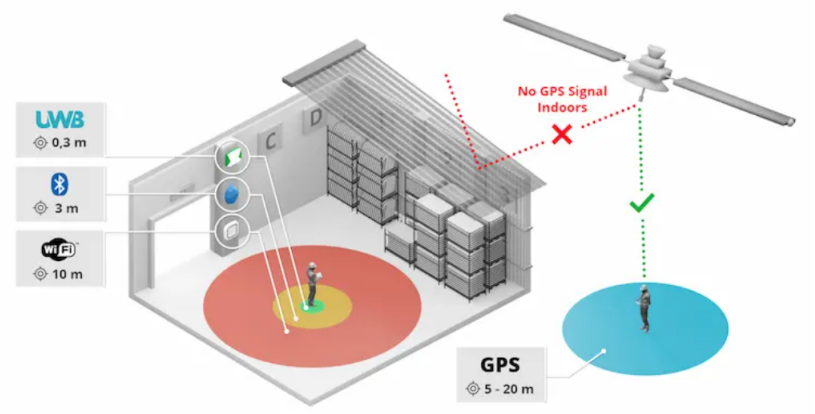

FIGURA 2.5: Tecnologie radio-based in ambienti indoor

2.4 Algoritmi di posizionamento

Dopo aver esaminato il processo di acquisizione dei dati per il tracciamento della posizione, possiamo ora concentrarci sull'analisi di come queste informazioni vengano utilizzate all'interno di un algoritmo per prevedere la posizione di un dispositivo in un ambiente indoor. In [24], viene presentata una categorizzazione di tali algoritmi:

- *Fingerprinting:* Comunemente noto anche come *mapping*, questo approccio si suddivide in una fase offline e una fase online. La fase offline comporta la creazione di una mappa dei punti di riferimento (reference points), disposti idealmente in una griglia omogenea [25]. Questa mappa viene quindi rappresentata tramite un database che contiene i valori di RSS (Received Signal Strength) di ogni nodo rispetto agli altri. Nella fase online, avviene la misurazione effettiva: viene rilevato il RSS del dispositivo corrente nell'ambiente e confrontato con i valori memorizzati nella fase offline, al fine di determinare la posizione del soggetto [7]. Il calcolo della posizione può essere eseguito tramite due approcci principali:
  - *Deterministico:* Viene calcolata la distanza euclidea della differenza tra i valori di distanza tra i tag e i nodi, e viene applicato un algoritmo di prossimità, come il nearest-neighbor [22], [25].
  - *Probabilistico:* Nella fase offline, si utilizzano anche la media e la varianza dei valori dei nodi, al fine di modellare gli RSS come distribuzioni di probabilità gaussiane [26]. Questi modelli possono poi essere utilizzati nella fase online, ad esempio, attraverso l'uso di Particle Filter [24].
- *Path-loss:* Tecniche come la trilaterazione [27] e la triangolazione [28] sono suscettibili alle riflessioni del segnale e all'assorbimento nelle pareti. In questo approccio, anziché utilizzare direttamente il RSS per il calcolo della posizione, si fa uso del risultato di un modello di path-loss [29], che si basa sulla perdita del segnale. Questo tenta di mitigare i problemi precedentemente menzionati.

- *Pedestrian Dead Reckoning (PDR):* Sfruttando i sensori inerziali, è possibile localizzare un dispositivo tramite il conteggio dei passi compiuti dall'utente, a condizione che si conosca il punto di partenza [30]. Questo approccio si articola in tre fasi: innanzitutto, vengono conteggiati i passi effettuati dall'utente, quindi si stima la direzione dei passi e infine si calcola la distanza percorsa per passo. Quest'ultima può essere determinata utilizzando un approccio statico (dove ogni passo ha la stessa lunghezza) o dinamico (dove la lunghezza del passo può variare) [31].

2.5 Applicazione del machine learning

Il machine learning è ampiamente adottato per automatizzare soluzioni, in quanto offre la capacità di generalizzare l'informazione, rendendo la tecnologia efficiente e scalabile per risolvere una vasta gamma di problemi, tra cui quello della localizzazione interna [32]. Poiché gli ambienti indoor sono mutevoli e soggetti a cambiamenti nel tempo, come la disposizione degli oggetti all'interno delle stanze, il machine learning rappresenta l'approccio consigliato per l'elaborazione dei dati in questo contesto [32], [33].

Nel machine learning, la predizione è preceduta da una fase di training, durante la quale il modello viene addestrato su un insieme di dati noti, chiamato training set, che consiste in input con i corrispondenti output desiderati (dati etichettati) [34]. Questo processo può avvenire attraverso quattro tipi principali di apprendimento:

- *Supervisionato:* Si utilizzano dati etichettati per trovare la correlazione tra input e output, minimizzando sia gli errori di varianza che di bias [34].
- *Non supervisionato:* Si addestra il modello su input non etichettati per identificare pattern intrinseci nei dati [35].
- *Semi-supervisionato:* Si utilizza una combinazione di dati etichettati e non etichettati per addestrare il modello [36].
- *Per rinforzo:* Il modello apprende a prendere azioni in un ambiente basato sui feedback ricevuti [37].

Dopo la fase di training, avviene la fase di test, durante la quale il modello viene valutato su un insieme di dati separato noto come test set, al fine di valutare se il modello ha generalizzato correttamente l'informazione [38]. Successivamente, il modello allenato può essere utilizzato per classificare o prevedere valori numerici per nuovi dati.

Nel contesto della localization indoor, è essenziale conoscere la posizione del dispositivo. Questo può essere trattato come un problema di regressione, in cui si cerca di prevedere un valore numerico, come la latitudine e la longitudine del dispositivo, oppure come un problema di classificazione, in cui si predice la classe o la categoria della posizione del dispositivo, come un determinato ambiente o area all'interno di un edificio [34].

2.6 Linguaggi e librerie per sviluppare il modello

Una volta compreso il funzionamento generale del machine learning nel contesto della localizzazione interna, è fondamentale esaminare i linguaggi e le librerie utilizzate per lo sviluppo del modello.

Tra i linguaggi più diffusi, Python risulta essere la scelta predominante per affrontare problemi di localizzazione indoor, grazie alla sua sintassi chiara e alla vasta selezione di librerie disponibili [39]–[41]. Le principali librerie utilizzate in Python includono:

- *Scikit-learn*: una libreria di machine learning semplice ed efficiente, ampiamente utilizzata per la sua facilità d'uso e la vasta gamma di algoritmi disponibili [39].
- *Keras (Tensorflow)*: una libreria di alto livello per la creazione di reti neurali, spesso eseguita su Tensorflow, il backend di deep learning di Google [40].
- *PyTorch*: una libreria open source sviluppata da Facebook, nota per la sua flessibilità nell'addestramento di reti neurali e per la facilità di debugging rispetto ad altri framework [41].

Oltre a Python, il linguaggio R è ampiamente utilizzato per lo sviluppo di modelli statistici e di data mining, offrendo implementazioni di algoritmi di machine learning attraverso librerie come *caret* e *randomForest* [42].

Capitolo 3

Metodologia

3.1 Dataset e ambiente di sviluppo

Effettueremo i nostri test utilizzando il dataset *UJIndoorLoc* [43], un dataset pubblico WLAN fingerprint-based sviluppato per comparare diverse metodologie di localization indoor basate sul Wi-Fi fingerprinting.

Il dataset descrive tre edifici dell'Universitat Jaume I a Madrid, in Spagna, di cui due con 4 piani ciascuno e uno con 5 piani, coprendo complessivamente 110000 m^2 [43]. Le informazioni raw sono state raccolte nel 2013 da oltre 20 utenti e dai segnali di 25 dispositivi Android.

Il dataset è già suddiviso in training set e test set, con 19937 record nel file *trainingData.csv* per il training e 1111 record nel file *validationData.csv* per il test.

Ogni record dei due dataset è composto da 529 attributi:

- 520 RSSI provenienti da altrettanti WAPs (Wireless Access Points), dove il valore positivo 100 indica che il segnale non è stato rilevato da una WAP in quella determinata posizione. Questi attributi rappresentano le features del modello.
- LATITUDE, LONGITUDE, FLOOR e BUILDING_ID per descrivere le coordinate della posizione. Questi attributi sono i target da predire.
- Altri attributi come SPACE, POSITION, USER, DEVICE e TIMESTAMP sono raccolti per approcci di classificazione e analisi temporali, ma non sono rilevanti per il nostro progetto.

È importante notare che le coordinate di latitudine e longitudine sono rappresentate in metri utilizzando il sistema di coordinate UTM (Universal Transverse Mercator) basato sul datum WGS84 (World Geodetic System 1984). Ciò significa che le coordinate geografiche sono convertite in coordinate proiettate, esprimendo i valori in metri anziché gradi decimali.

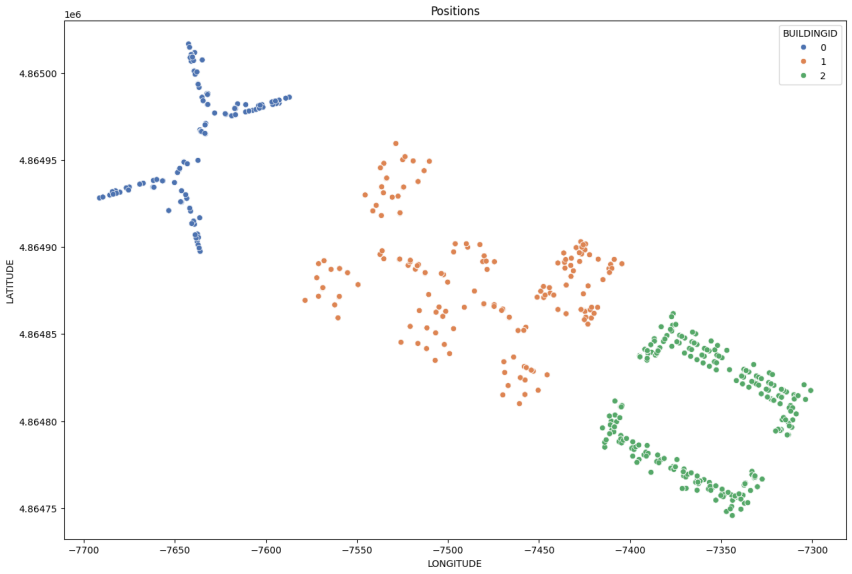

FIGURA 3.1: Visione dall'alto del dataset UJIndoorLoc

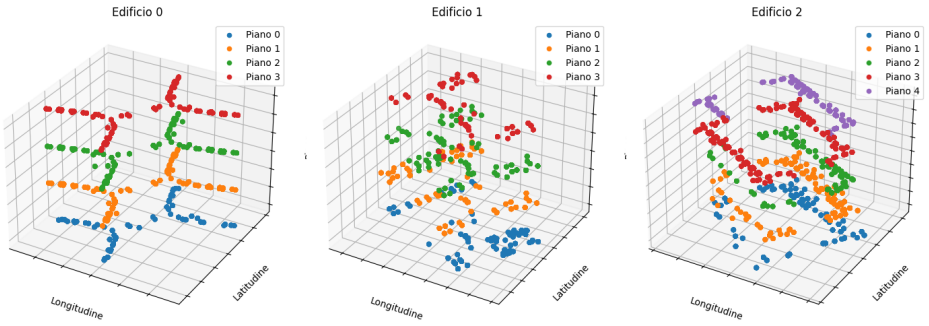

FIGURA 3.2: Visione 3D del dataset UJIndoorLoc

L'ambiente di sviluppo scelto per questa tesi è Google Colaboratory [44], che consente di combinare testo formattato con codice eseguibile in un unico ambiente, sfruttando rispettivamente Markdown e Python. Poiché utilizzeremo Scikit-learn per la realizzazione del progetto, non sfrutteremo la GPU del servizio, limitandoci all'utilizzo della CPU. Tuttavia, per migliorare le prestazioni del progetto, è possibile considerare l'utilizzo di Scikit-learn-extra, Keras o PyTorch, poiché supportano nativamente l'utilizzo della GPU.

3.2 Preprocessing dei dati

In questa sezione andremo ad analizzare ogni azione effettuata sul dataset per l'apprendimento automatico, con lo scopo di ridurre il costo computazionale dello sviluppo di quest'ultimo.

3.2.1 Gestione dei dati mancanti

Come possiamo vedere da questo snippet di codice:

```
print('Numero di dati mancanti nel training set: ',trainingData.isnull().sum().sum())
print('Numero di dati mancanti nel test set: ',validationData.isnull().sum().sum())
```

```
Numero di dati mancanti nel training set:  0
Numero di dati mancanti nel test set:  0
```

in questo dataset non risultano valori mancanti, il che è importante in quanto la presenza di valori mancanti potrebbe causare problemi durante il processo di apprendimento. Pertanto, non è necessario gestirli. Tuttavia, nel caso in cui fossero presenti valori mancanti, avremmo potuto utilizzare strumenti come SimpleImputer per gestirli in modo appropriato.

3.2.2 Creazione dell'attributo BUILDING_FLOOR

Nel dataset, i valori del piano e dell'edificio sono rappresentati rispettivamente dagli attributi FLOOR e BUILDING_ID. In particolare, l'attributo FLOOR assume valori discreti nel range da 0 a 4, mentre l'attributo BUILDING_ID assume valori discreti nel range da 0 a 2. Pertanto, è necessario utilizzare due modelli di classificazione separati, uno per il piano e uno per l'edificio.

È possibile combinare entrambe le informazioni utilizzando la seguente formula:

$$BUILDING\_FLOOR = BUILDING\_ID \cdot 10 + FLOOR \tag{3.1}$$

In questo modo, si lega l'informazione relativa all'edificio all'ordine delle decine e l'informazione relativa al piano all'ordine delle unità. Tuttavia, è importante notare che i valori risultanti sono comunque discreti e adatti per l'uso in algoritmi di classificazione. È importante notare che questa rappresentazione è discontinua: ad esempio, l'ultimo piano del primo edificio è rappresentato dal valore 03, mentre il primo piano del secondo edificio è rappresentato dal valore 10. Sebbene questi numeri siano consecutivi nella successione, non sono consecutivi nel dominio dei numeri naturali. Questa discontinuità potrebbe portare ad errori durante l'allenamento dei modelli, a seconda del tipo di algoritmo utilizzato.

3.2.3 Miglioramento della continuità dei dati

Le features del dataset consistono nelle potenze del segnale delle WAPs, le quali saranno utilizzate per predire la posizione. Il range di valori di queste informazioni è compreso tra -104 e 0, dove -104 rappresenta il valore più basso di una WAP che è ancora in range, mentre 0 rappresenta il valore più alto. Se una WAP è fuori dal range relativo ad una posizione, viene assegnato il valore 100. Questa discontinuità tra il valore più basso nel range e il valore fuori dal range può creare problemi durante la preparazione dei dati. Per affrontare questa problematica, abbiamo sostituito il valore 100 con il valore -111 in tutto il dataset. Questo non solo riduce il gap tra i valori sopracitati, ma rende anche il valore fuori dal range il più piccolo possibile tra tutti i valori possibili. Pertanto, l'intervallo di valori relativo alle potenze del segnale delle WAPs è ora compreso tra -111 e 0.

3.3 Normalizzazione dei dati

Utilizzando la tecnica di normalizzazione Minmax, è possibile ridimensionare l'intervallo dei valori delle potenze del segnale delle WAPs in un intervallo specifico. Questa tecnica segue la seguente formula:

$$x_n = \frac{x_i - \min(x)}{\max(x) - \min(x)} \tag{3.2}$$

Dove:

- x_n è il valore normalizzato;
- x_i è il valore originale da normalizzare;
- $\min(x)$ è valore minimo del segnale, che è -111;
- $\max(x)$ è valore massimo del dataset, che è 0.

In questo modo, l'intervallo dei valori normalizzati sarà compreso tra 0 e 1, dove 0 indicherà che la potenza del segnale della WAP non copre la posizione cercata, mentre 1 indicherà il massimo segnale della WAP. L'applicazione di questa operazione accelera significativamente il processo di apprendimento automatico per tutti i modelli che verranno sviluppati successivamente.

3.4 Selezione delle features

Per ridurre il numero di features e accelerare il processo di apprendimento automatico, possiamo utilizzare la feature selection tramite il metodo Variance Threshold. Questo metodo consente di rimuovere le features con bassa varianza, poiché tali features potrebbero non essere significative per l'apprendimento. Il calcolo della varianza può essere eseguito utilizzando la seguente formula:

$$\sigma^2 = \frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N} \tag{3.3}$$

Dove:

- σ^2 rappresenta la varianza della feature;
- x_i è il valore i-esimo della feature;
- μ è la media delle feature;
- N è il numero di campioni del dataset.

Dopo aver calcolato la varianza per ciascuna feature, vengono rimosse tutte le features che presentano una varianza pari a 0, poiché non aggiungono alcuna informazione utile al modello di apprendimento automatico.

3.5 Riduzione della dimensionalità

Per migliorare ulteriormente le prestazioni del modello di apprendimento automatico, possiamo applicare l'analisi delle componenti principali (PCA, Principal Component Analysis) al nostro dataset. Questo ci permette di trasformare le features originali in un nuovo set di features, riducendo la dimensionalità dei dati mantenendo al contempo la maggior parte della varianza.Il processo PCA comprende i seguenti passaggi:

1. Standardizzazione delle features: Le features vengono standardizzate per avere una media zero e una deviazione standard unitaria. Questo passaggio è già stato eseguito applicando Minmax e Variance Threshold.

2. Calcolo della matrice di covarianza:

$$\Sigma = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \overline{x})(x_i - \overline{x})^T \tag{3.4}$$

   dove:

   - Σ è la matrice di covarianza;
   - N è il numero di campioni del dataset;
   - x_i è il vettore dei valori i-esimo;
   - $\overline{x}$ è il vettore medio dei valori;

3. Calcolo degli autovalori e autovettori della matrice di covarianza;

4. Ordinamento del dataset in ordine decrescente degli autovalori: Le nuove features sono ordinate in base alla loro importanza, rappresentata dagli autovalori maggiori, che indicano una maggiore varianza.

Dopo aver creato questo nuovo set di dati trasformato, possiamo selezionare solo le componenti principali che spiegano una percentuale significativa della varianza totale del dataset. In questo caso, è stato scelto di mantenere solo le features che spiegano l'85% della varianza totale.

Capitolo 4

Sviluppo dei modelli

4.1 Strumenti per lo sviluppo

Nella presente sezione, sarà esposto il processo di sviluppo dei modelli adottati nella tesi, con un'analisi dettagliata dei metodi impiegati e delle ragioni che hanno guidato tali scelte, sino alla fase di creazione e archiviazione dei modelli stessi.

4.1.1 Grid Search

L'uso di Grid Search consente di individuare gli iperparametri ottimali in un modello di machine learning. Il suo funzionamento può essere riassunto nei seguenti punti:

1. Viene definito lo spazio di ricerca, che può comprendere valori continui o discreti, a seconda delle esigenze del problema;
2. Vengono esplorate tutte le combinazioni possibili di iperparametri all'interno dello spazio definito;
3. Viene valutata ogni combinazione, restituendo come output il modello con i migliori iperparametri individuati nello spazio di ricerca.

4.1.2 Cross-Validation

La cross-validation rappresenta una tecnica statistica ampiamente utilizzata nell'ambito del machine learning con l'obiettivo di valutare accuratamente le prestazioni di un modello, riducendo il rischio di overfitting. Nell'ambito di questa tesi, sono state impiegate le tecniche *KFold* per gli algoritmi di regressione e *Stratified KFold* per quelli di classificazione.

Il funzionamento di KFold può essere sintetizzato nei seguenti passaggi:

1. Divisione dei dati in fold, in questo contesto il dataset è stato suddiviso in 5 fold;

2. Selezione di un fold come set di convalida e dei restanti come set di addestramento;

3. Ripetizione del processo K volte, dove K rappresenta il numero di fold.

Per configurare KFold, è stato utilizzato il seguente frammento di codice:

```
kfold = KFold(n_splits=5, random_state=77, shuffle=True)
```

FIGURA 4.1: Parametri per la cross validation

Dove il parametro *n_splits* indica il numero di fold, *random_state* consente di controllare la casualità della suddivisione dei fold, rendendo i risultati della cross-validation riproducibili in diverse circostanze, mentre *shuffle* si occupa di mescolare i dati prima della suddivisione dei fold, evitando che l'ordine di acquisizione influenzi la cross-validation.

Tuttavia, utilizzare KFold come metodo di cross-validation per tutti gli algoritmi di machine learning può comportare una limitazione: una suddivisione sbilanciata dei fold può causare notevoli imprecisioni durante la validazione incrociata, poiché alcuni fold potrebbero mancare di informazioni cruciali a causa della loro distribuzione non omogenea. Questa problematica si presenta soprattutto nell'addestramento di modelli per la previsione di piano e edificio della posizione.

Per affrontare questa criticità, viene impiegato Stratified KFold, che arricchisce KFold con una stratificazione: la suddivisione dei fold tiene conto della distribuzione dei dati, garantendo la creazione di fold omogenei sotto questo aspetto. I parametri per la configurazione di Stratified KFold sono gli stessi di KFold, come illustrato nel frammento di codice precedentemente riportato, con l'unica differenza rappresentata dall'applicazione della stratificazione, che caratterizza la differenza tra le tecniche di cross-validation per algoritmi di classificazione e regressione.

4.1.3 GridSearchCV e pipeline

L'implementazione di *GridSearchCV* costituisce un approccio che combina i vantaggi di Grid Search e della cross-validation. Per ogni combinazione di iperparametri nello spazio di ricerca, viene eseguita una cross-validation, aumentando ulteriormente la precisione di Grid Search e garantendo la selezione dei migliori iperparametri senza incorrere nell'overfitting.

In questa tesi, il range di valori selezionati per GridSearch sarà relativamente limitato, prendendo ispirazione da risultati ottenuti in tentativi precedenti sul medesimo dataset. Questa scelta è motivata dalle restrizioni temporali relative all'addestramento dei modelli. L'obiettivo principale è quindi illustrare il funzionamento di GridSearch, pur consapevoli che un range più ampio di iperparametri potrebbe condurre a risultati più accurati. Invitiamo pertanto il lettore a esplorare combinazioni di iperparametri con range più estesi rispetto a quelli proposti nella tesi, al fine di ottenere risultati più approfonditi e dettagliati.

Nel capitolo precedente abbiamo esaminato le trasformazioni dati per migliorare le prestazioni dell'addestramento, e nel paragrafo precedente abbiamo illustrato come verrà utilizzato GridSearchCV per ottimizzare l'output.

In uno dei parametri di GridSearchCV, denominato *estimator*, viene specificato il tipo di algoritmo che il modello dovrà utilizzare. In questo contesto, il parametro *estimator* non verrà utilizzato solo a tale scopo, ma anche per applicare tecniche come MinMax, Variance Threshold e PCA, grazie all'uso di una pipeline.

L'utilizzo di una pipeline, anziché eseguire tutte le trasformazioni del dataset prima dell'addestramento, offre due vantaggi:

- Ogni trasformazione viene applicata separatamente a ciascun fold, evitando così la contaminazione dei dati tra il set di addestramento e quello di test;
- I parametri ottimali identificati da GridSearchCV non saranno basati esclusivamente sul classificatore/regressore, ma terranno conto anche delle trasformazioni applicate.

Un esempio di pipeline che verrà inserita nel parametro *estimator* di GridSearchCV è presentato nel seguente snippet di codice, in cui viene utilizzato un classificatore basato sull'algoritmo Random Forest (uno degli algoritmi trattati nella tesi):

```
pipeline_example = Pipeline([
    ('data_scaling', MinMaxScaler()),
    ('feature_selection_1', VarianceThreshold()),
    ('dimension_reduction', PCA(0.85)),
    ('model', RandomForestClassifier())
    ])
```

FIGURA 4.2: Esempio di costruzione della pipeline

4.2 Algoritmi di machine learning

In questa sezione esamineremo gli algoritmi impiegati nella presente tesi, illustrando il loro funzionamento insieme ai relativi range di parametri selezionati per l'utilizzo con GridSearch. È importante sottolineare che, oltre agli algoritmi trattati, esistono numerosi altri approcci, alcuni dei quali possono dimostrarsi più o meno performanti nel contesto specifico di questo dataset.

Invitiamo quindi il lettore a esplorare ulteriori algoritmi seguendo la stessa metodologia proposta, al fine di ottenere risultati più completi e migliorare la comprensione delle dinamiche del dataset in esame.

4.2.1 Random Forest

Il funzionamento del Random Forest è descritto dai seguenti passaggi una volta che l'input è stato inserito e i dati sono stati preprocessati:

1. Viene generato un insieme di alberi decisionali;
2. Ciascun albero viene addestrato su un sottoinsieme casuale delle features;

3. Ogni albero del Random Forest fornisce un'etichetta con la propria predizione;

4. Viene restituita l'etichetta con la predizione più frequente tra tutti gli alberi di decisione del Random Forest.

Gli iperparametri settati per Random Forest, sia per problemi di classificazione che di regressione, sono i seguenti:

```
param_grid_randomforest = {
    'model__max_depth': [10],
    'model__n_estimators': [i for i in range(300,500,100)],
    'model__min_samples_leaf': [i for i in range(2,4,1)]
    }
```

FIGURA 4.3: Parametri della griglia per Random Forest

Dove:

- *max_depth*: indica il limite della profondità di un albero decisionale per evitare l'overfitting;
- *n_estimators*: indica il numero di alberi decisionali nel Random Forest. Un valore maggiore rende l'addestramento più robusto, ma comporta un aumento del costo computazionale;
- *min_samples_leaf*: indica il numero minimo di campioni che una foglia dell'albero deve contenere. Un valore adeguato permette di ottenere suddivisioni omogenee dei dati negli alberi, ma un valore troppo grande può generare alberi troppo semplici, causando la perdita di dettagli nei dati.

In Python, il ciclo for inizia dal primo valore (incluso) e termina con il secondo valore (escluso), con un incremento di unità pari al terzo valore ad ogni iterazione. In questo caso, avremo 2 valori possibili per *n_estimators* e 2 valori possibili per *min_samples_leaf*, considerando 5 fold dati dalla cross-validation. Pertanto, GridSearchCV eseguirà un totale di 20 fit con i seguenti settaggi.

4.2.2 K-Nearest Neighbor

Il funzionamento di K-Nearest Neighbor è descritto dai seguenti passaggi una volta che l'input è stato inserito e i dati sono stati preprocessati:

1. Viene calcolata la distanza euclidea tra il punto di test e tutti i punti del set di addestramento;

2. Vengono selezionati i K vicini con la minore distanza calcolata nel passaggio precedente;

3. Viene contato il numero di punti di addestramento per ogni classe tra i K punti più vicini e viene assegnata un'etichetta con il maggior numero di voti;

4. Viene restituita l'etichetta con il valore stimato.

Gli iperparametri impostati per K-Nearest Neighbor, sia per problemi di classificazione che di regressione, sono i seguenti:

```
param_grid_knn = {
    "model__n_neighbors": [i for i in range(3,6,1)],
    "model__leaf_size": [i for i in range(5,20,5)],
}
```

FIGURA 4.4: Parametri della griglia per K-Nearest Neighbor

Dove:

- *n_neighbors*: rappresenta il valore di K, ossia il numero di vicini da considerare per un punto;
- *leaf_size*: indica la dimensione della foglia dell'albero utilizzata per organizzare i dati durante la ricerca dei vicini. Più piccola è la dimensione, più precisa sarà la ricerca, ma i tempi di addestramento saranno maggiori.

Avremo 3 valori possibili per *n_neighbors* e 3 valori possibili per *leaf_size*, considerando 5 fold dati dalla cross-validation. Con i seguenti settaggi, GridSearchCV eseguirà un totale di 45 fit.

4.2.3 Support Vector Machine

Il funzionamento di SVM è il seguente: una volta inserito l'input e preprocessato i dati:

1. viene definita la funzione di decisione, che calcola la distanza dal punto al piano di decisione;
2. lo spazio delle features viene divisa tramite iperpiani, viene quindi assegnato il valore in base alla posizione del punto rispetto agli iperpiani;
3. tramite i vettori di supporto, si tenta di massimizzare il margine con gli iperpiani, che permettono la corretta generalizzazione dell'informazione;
4. quando tutto è stato settato, l'algoritmo predirà la posizione in base al lato dove si trova il punto rispetto agli iperpiani.

Gli iperparametri settati in questa tesi, sia per problemi di classificazione che di regressione, sono i seguenti:

```
param_grid_svm = {
    'model__C': [10**i for i in range(0,3,1)],
    'model__gamma': [10**i for i in range(-1,2,1)]
}
```

FIGURA 4.5: Parametri della griglia per Support Vector Machine

Dove:

- *C* controlla il tradeoff tra complessità del modello e accuratezza nell'addestramento. Un valore più basso permette al modello di essere semplice ma impreciso, un valore più alto permette una precisione maggiore, a discapito della complessità del modello e quindi aumentando il costo computazionale.

- *gamma* influenza il peso di un singolo esempio nell'iperpiano. Un valore basso comporta che esempi lontani dalla regione di decisione hanno un forte impatto, mentre un valore alto porta ad avere un impatto maggiore gli esempi vicini.

Avremo 3 valori possibili per *C* e 3 valori possibili per *gamma*, considerando 5 fold dati dalla cross-validation. Con i seguenti settaggi, GridSearchCV eseguirà un totale di 45 fit.

4.2.4 XGBoost

Il funzionamento di XGBoost è il seguente: una volta inserito l'input e preprocessato i dati:

1. viene creato un singolo albero decisionale di base e viene addestrato;

2. vengono calcolati i residui, ossia la differenza tra i valori osservati e quelli predetti;

3. viene creato un albero decisionale con lo scopo di predire i residui dell'albero precedente, con lo scopo di ridurre i residui;

4. viene eseguito il processo iteramente per vari alberi, che si uniscono al modello;

5. una volta finito, la predizione finale del modello è una somma di tutte le predizioni degli alberi decisionali creati.

Gli iperparametri settati in questa tesi, sia per problemi di classificazione che di regressione, sono i seguenti:

```
param_grid_xgboost = {
    "model__n_estimators": [i for i in range(100,250,50)],
    "model__max_depth": [10],
    "model__learning_rate": [i for i in np.arange(0.1,0.4,0.1)]
}
```

FIGURA 4.6: Parametri della griglia per XGBoost

Dove:

- *n_estimators* e *max_depth* sono i stessi parametri visti per Random Forest, poiché anche XGBoost utilizza alberi decisionali per la predizione;

- *learning_rate* indica quanto veloce deve essere l'apprendimento di ogni albero, è un valore che varia tra 0 a 1. Un valore alto del learning rate accelera l'adattamento del modello ma può aumentare il rischio di overfitting e oscillazioni durante l'ottimizzazione. Al contrario, un

valore basso rallenta l'adattamento del modello ma può contribuire a una maggiore regolarizzazione e precisione.

Avremo 3 valori possibili per *n_estimators* e 4 valori possibili per *learning_rate*: essendo valori decimali, è necessario utilizzare la funzione *arange* di Numpy. Importante notare che il secondo parametro passato alla funzione viene incluso, e non escluso come nel ciclo for di Python visti in precedenza. Considerando i 5 fold dati dalla cross-validation, GridSearchCV eseguirà un totale di 60 fit con i seguenti settaggi.

4.3 Addestramento del classificatore

Per l'attributo BUILDING_FLOOR, essendo una suddivisione di piani ed edifici, trattiamo il problema come un problema di classificazione. Di seguito è presente un esempio di settaggio e avvio dell'allenamento.

```
classificatore = GridSearchCV(estimator = pipeline_classificatore,
                               param_grid = param_grid_classificatore,
                               cv = skfold,
                               return_train_score=True,
                               verbose=3)
classificatore.fit(X_train, y_train['BUILDING_FLOOR'])
classificatore.best_params_
```

FIGURA 4.7: Applicazione e allenamento di un modello di classificazione

Dove nel dettaglio:

- in *estimator* abbiamo inserito la pipeline dei processi vista in 4.2;
- in *param_grid* abbiamo inserito i parametri da utilizzare, visti in 4.3, 4.4, 4.5, 4.6;
- in *cv* inseriamo la cross validation, vista in 4.1, in questo caso Stratified KFold;
- in *return_train_score* settiamo True, così da avere un feedback del training durante quest'ultimo;
- in *verbose* impostiamo la verbosità del feedback, col valore 3 visualizzerò informazioni sul modello, lo score e il tempo impiegato.

Tramite la funzione *fit*, il modello inizierà a tutti gli effetti l'allenamento, eseguendo tutte le combinazioni settate in precedenza, dove sono dichiarati come parametri le features (*X_train*) e il target (*y_train*, in questo caso BUILDING_FLOOR). Infine, tramite la funzione *best_params_*, verranno stampati l'eventuale combinazione di iperparametri con le migliori performance valutate durante l'allenamento.

4.3.1 Aggiunta di Label Encoder per XGBoost

Nella sezione 3.2.2, abbiamo notato un dettaglio, ossia che i valori, seppur discreti, di BUILDING_FLOOR non sono continui per via della sua costruzione:

dall'ultimo piano del primo edificio (03) al piano terra del secondo edificio (10) abbiamo un salto, e questo avviene ad ogni cambio di edificio. Applicare la fit su XGBoost comporta un errore durante l'allenamento, dobbiamo quindi risolvere questo problema di discontinuità. Per ovviare a ciò, utilizziamo Label Encoder, il cui funzionamento consiste nell'associare a classi discontinue, classi nuove continue, che si riferiscono alle classi originali.

Nel dettaglio, viene associato il valore 0 (nuova classe) alla valore 00 (vecchia classe, che indica primo edificio piano terra), 1 al valore 01, e così via fino ad arrivare a 3 per 03. Nel momento del cambio di edificio, quindi per il valore 10, verrà condotto al valore 4, 11 diventerà 5, e così via. Le nostre classi saranno quindi sempre 13, ma con Label Encoder saranno valori discreti da 0 a 12, che sono trattabili da XGBoost.

L'applicazione di Label Encoder sul target per il training è visibile nella figura 4.8

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
y_train_encoded = le.fit_transform(y_train['BUILDING_FLOOR'])
```

FIGURA 4.8: Applicazione di Label Encoder per il target di XGBoost

4.4 Addestramento dei regressori

Per gli attributi LATITUDE e LONGITUDE, essendo valori in metri continui, trattiamo i problemi come problemi di regressione. Di seguito è presente un esempio di settaggio e avvio dell'allenamento.

```
coordinate = GridSearchCV(estimator = pipeline_regressore,
                          param_grid = param_grid_regressore,
                          cv = kfold,
                          return_train_score=True,
                          verbose=3)
coordinate.fit(X_train, y_train['LATITUDE'])
coordinate.best_params_
```

FIGURA 4.9: Applicazione e allenamento di un modello di regressione

Dove nel dettaglio:

- in *estimator* abbiamo inserito la pipeline dei processi vista in 4.2, ma adesso in model sarà presente un regressore anziché un classificatore;
- in *param_grid* abbiamo inserito i parametri da utilizzare, visti in 4.3, 4.4, 4.5, 4.6;
- in *cv* inseriamo la cross validation, vista in 4.1;
- *return_train_score* e *verbose* sono identici a 4.3;

Tramite la funzione *fit*, il modello inizierà a tutti gli effetti l'allenamento, eseguendo tutte le combinazioni settate in precedenza, dove sono dichiarati come parametri le features (*X_train*) e il target (*y_train*, in questo caso LATITUDE in figura 4.9 e LONGITUDE con lo stesso pattern). Infine, tramite la funzione *best_params_*, verranno stampati l'eventuale combinazione di iperparametri con le migliori performance valutate durante l'allenamento.

4.5 Salvataggio dei modelli addestrati

Per il deploy dei modelli, per utilizzi futuri e per evitare di allenare nuovamente il modello (ricordiamo che tutti i modelli sono stati allenati su una sessione di Google Colab, una volta chiuso il runtime tutte le informazioni vengono cancellate, lasciando solamente gli output stampati sul foglio ma senza alcuna variabile allocata di ogni tipo, compresi quindi i modelli addestrati) utilizziamo la libreria Pickle per il salvataggio di questi ultimi.

Come è possibile vedere in figura 4.10, i file vengono salvati all'interno del percorso *modelli/modello_addestrato* col parametro *wb*, che indica la scrittura del file. Tramite la funzione *dump* di Pickle, viene salvato il modello allenato. Se vogliamo richiamare il modello per predizioni future o per il deploy, come vedremo nel capitolo 6, basta richiamare il percorso del file tramite il parametro *rb*, che indica la lettura di un file, e possiamo allocare il modello a una variabile tramite la funzione *load* di Pickle.

```
import pickle

#Salvataggio del modello
with open('modelli/modello_addestrato.pkl','wb') as file:
        pickle.dump(modello_addestrato, file)

#Caricamento del modello
percorso_file = open('modelli/modello_addestrato.pkl','rb')
modello = pickle.load(percorso_file)
```

FIGURA 4.10: Salvataggio e caricamento del modello addestrato

Una volta eseguita la funzione di salvataggio di tutti i modelli allenati sul dataset UJIndoorLoc preprocessato, ecco di seguito elencati i file di output di quanto illustrato in questo capitolo:

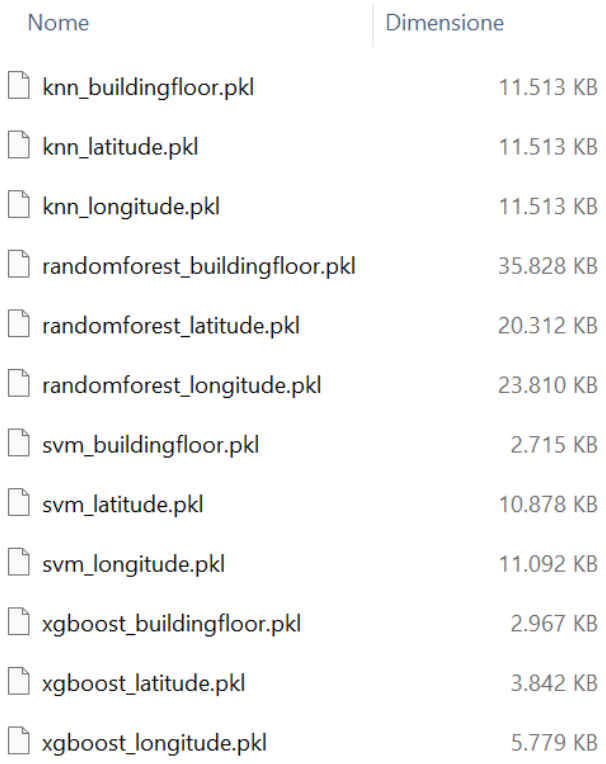

FIGURA 4.11: Modelli addestrati salvati con Pickle

Notiamo già, prima di confrontare la differenza di prestazioni nel prossimo capitolo, la differenza di dimensione dei modelli, notando come i modelli basati su XGBoost siano file di dimensioni ridotte rispetto agli altri, specie per i modelli dedicati a LATITUDE e LONGITUDE.

Capitolo 5

Confronto dei modelli

5.1 Predizione dei targets

Allenati i modelli, è il momento di eseguire le previsioni, per poter testare e confrontare i modelli allenati nel capitolo precedente. L'azione di predizione è presente in figura 5.1. Possiamo notare come prima di eseguire *predict* inseriamo *best_estimator_*. Lo scopo è di prendere automaticamente i migliori iperparametri visti nella sezione 4.2, trovati poi durante l'allenamento ed infine eseguire la predizione su quest'ultimi.

```
y_predizione = classificatore_regressore.best_estimator_.predict(X_test)
```

FIGURA 5.1: Esempio di predizione del modello con i migliori parametri

5.1.1 Reverse di Label Encoder per XGBoost

La metodologia vista in figura 5.1 è stata applicata in tutti i modelli, ad eccezione di XGBoost per l'attributo BUILDING_FLOOR. Come visto nella sezione 4.3.1, abbiamo applicato Label Encoder per risolvere il problema della continuità delle classi che XGBoost non riusciva a gestire. Tuttavia, applicare la previsione come per gli altri modelli, seppur contenente l'informazione di edificio e piano della posizione, non è congruo con i valori delle classi di tutti gli altri modelli. In figura 5.2 è possibile vedere la predizione nello specifico e il reverse alle classi originali.

```
y_pred_buildingfloor_xgboost_le = xgboost_buldingfloor.best_estimator_.predict(X_test)
#reverse dell'encoding per ritornare alle classi desiderate
y_pred_buildingfloor_xgboost = le.inverse_transform(y_pred_buildingfloor_xgboost_le)
```

FIGURA 5.2: Predizione del modello per BUILDING_FLOOR basato su XGBoost

5.2 Metriche per la valutazione

Effettuate le previsioni, dobbiamo trovare le metriche per poter confrontare direttamente i modelli, stabilendo qual è il più performante nello specifico e in media.

5.2.1 Classificazione

Per l'attributo BUILDING_FLOOR, che ricordiamo essere un problema di classificazione, definendo *True Positive (TP)* e *True Negative (TN)* le corrispondenze corrette tra le previsioni di un modello e i veri valori di una classe, *False Positive (FP)* e *False Negative (FN)* che indicano errori di classificazione, sono state scelte le seguenti metriche per la valutazione delle prestazioni:

- *Accuracy-score*: si trova dividendo il numero totale di previsioni corrette fatte dal modello per il numero totale di previsioni effettuate. La formula è la seguente:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{5.1}$$

  Dove il risultato è un valore compreso tra 0 e 1. Un valore più alto dell'accuracy-score indica una maggiore capacità del modello di classificare correttamente i dati. Ad esempio, un'accuracy-score di 0.7 indica che il modello ha classificato correttamente il 70% dei casi nel set di dati.

- *F1-score*: rappresenta l'armonica media tra le metriche *precision* e *recall*, ed è utile quando le classi nel dataset sono sbilanciate. La formula è la seguente:

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.2}$$

  Dove:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{5.3}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5.4}$$

  L'F1-score varia da 0 a 1, dove un valore alto indica un migliore bilanciamento tra Precision e Recall, il che significa che il modello ha una migliore capacità di classificare correttamente le istanze positive e di evitare falsi positivi e falsi negativi.

- *Confusion Matrix*: consiste in una tabella che mostra il numero di predizioni corrette e errate fatte dal modello nel test set, confrontando le predizioni effettuate dal modello con i valori reali dei dati di test. Una rappresentazione di una matrice di confusione è presente in figura 5.3:

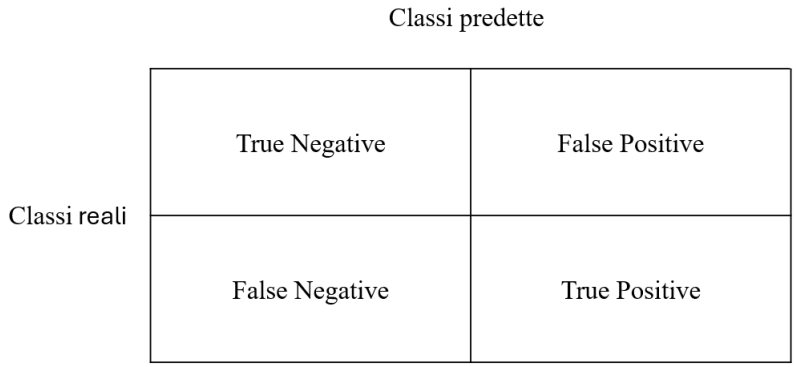

FIGURA 5.3: Lettura di una Confusion Matrix

Nella diagonale, dove sono presenti i valori di TP e TN, si desidera avere valori predominanti poiché indicano correttezza nelle previsioni, mentre è preferibile minimizzare i valori nelle altre celle. In una matrice più grande (come una 13x13 per BUILDING_FLOOR nel nostro caso), la presenza di valori significativi è rilevante solo lungo la diagonale principale, poiché ogni classe è confrontata solo con se stessa. Pertanto, si desidera che la diagonale della matrice di confusione sia la più ricca possibile, con il resto nullo, indicando che le previsioni sono state tutte corrette. Inoltre, è possibile analizzare quali errori ha commesso il modello e con quale frequenza, per poter intervenire di conseguenza se necessario.

Di seguito viene presentato il codice con il quale verrà effettuato il calcolo delle metriche dei modelli di classificazione allenati.

```
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix

#accuracy
accuracy = accuracy_score(y_test['BUILDING_FLOOR'], y_pred_classificatore)
print("Accuracy:", accuracy_svm)

#F1-score
f1 = f1_score(y_test['BUILDING_FLOOR'], y_pred_classificatore, average='weighted')
print("F1-score:", f1)

#confusion matrix
confusion_matrix = confusion_matrix(y_test['BUILDING_FLOOR'], y_pred_classificatore)
print("Confusion Matrix:\n", confusion_matrix)
```

FIGURA 5.4: Calcolo delle metriche di un classificatore

5.2.2 Regressione

Per gli attributi LATITUDE e LONGITUDE, che sono problemi di regressione, definendo y_i il valore osservato, $\hat{y}_i$ il valore predetto, $\bar{y}$ il valore medio di quelli osservati, n il numero totale delle osservazioni, sono state scelte le seguenti metriche per la valutazione:

- R^2*-score*: quantifica quanto delle variazioni osservate nella variabile dipendente può essere attribuito alla variazione nelle variabili indipendenti considerate nel modello. La formula è la seguente:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{5.5}$$

  La metrica varia da 0 a 1, dove un valore maggiore indica una maggiore capacità del modello di adattarsi ai dati;

- *Mean Squared Error (MSE)*: calcola la media dei quadrati delle differenze tra i valori previsti dal modello e i valori reali osservati. La formula è la seguente:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{5.6}$$

  Il MSE fornisce una misura della dispersione dei dati intorno alla linea di regressione, dove valori più elevati indicano una maggiore dispersione. Invece, una diminuzione del MSE implica una maggiore precisione del modello nella previsione dei valori osservati.

- *Mean Absolute Error (MAE)*: misura la media delle differenze assolute tra i valori previsti e i valori reali. La formula è la seguente:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{5.7}$$

  Essendo una misura della media degli errori di previsione, il MAE fornisce una valutazione diretta della precisione del modello. Una diminuzione del MAE indica una maggiore precisione del modello nell'approssimare i valori reali, evidenziando una migliore capacità predittiva.

Di seguito viene presentato il codice con il quale verrà effettuato il calcolo delle metriche dei modelli di regressione allenati, in questo caso per un modello basato sulla predizione dell'attributo LONGITUDE.

```python
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

#R2
r2 = r2_score(y_test['LONGITUDE'], y_pred_regressore)
print("R2:", r2)

# Mean squared error (MSE)
mse = mean_squared_error(y_test['LONGITUDE'], y_pred_regressore)
print("MSE:", mse)

# Mean absolute error (MAE)
mae = mean_absolute_error(y_test['LONGITUDE'], y_pred_regressore)
print("MAE:", mae)
```

FIGURA 5.5: Calcolo delle metriche di un regressore

5.3 Confronto dei modelli

Confrontiamo adesso i modelli allenati nella fase sperimentale della tesi, decretando quali sono i migliori nelle rispettive metriche. Nei grafici a seguire verrà colorato in arancione il modello che possiede la metrica migliore.

5.3.1 BUILDING_FLOOR

Accuracy

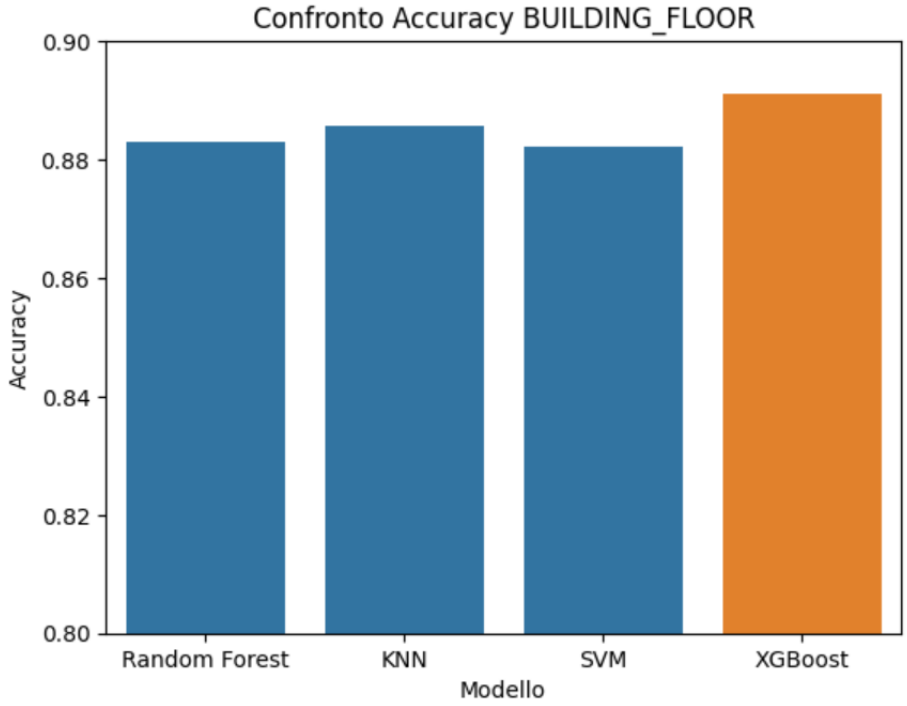

Figura 5.6: Confronto accuracy-score per BUILDING_FLOOR

F1

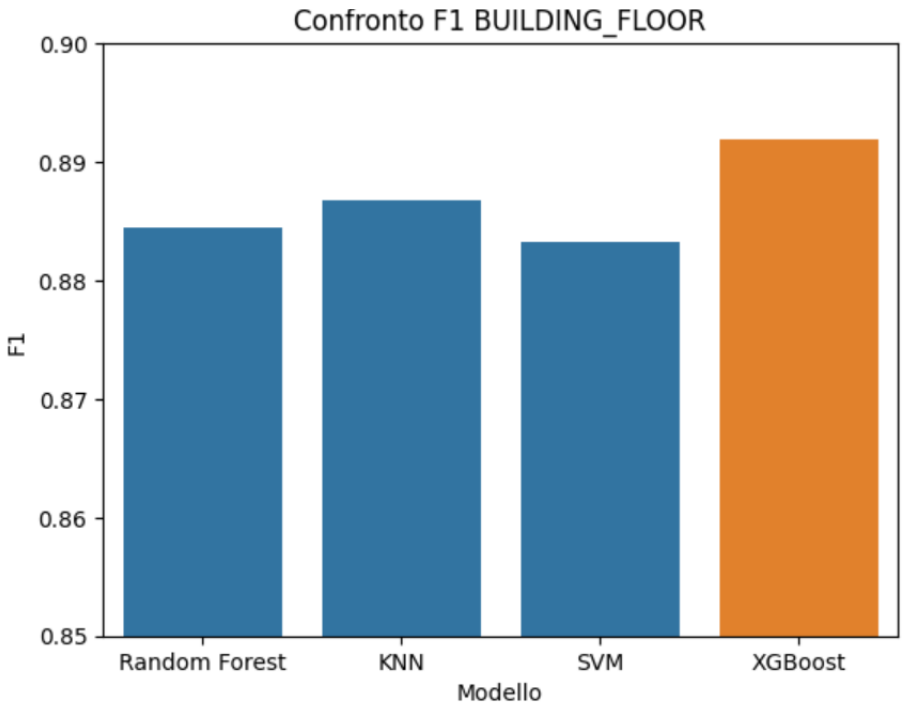

FIGURA 5.7: Confronto F1-score per BUILDING_FLOOR

Confusion Matrix

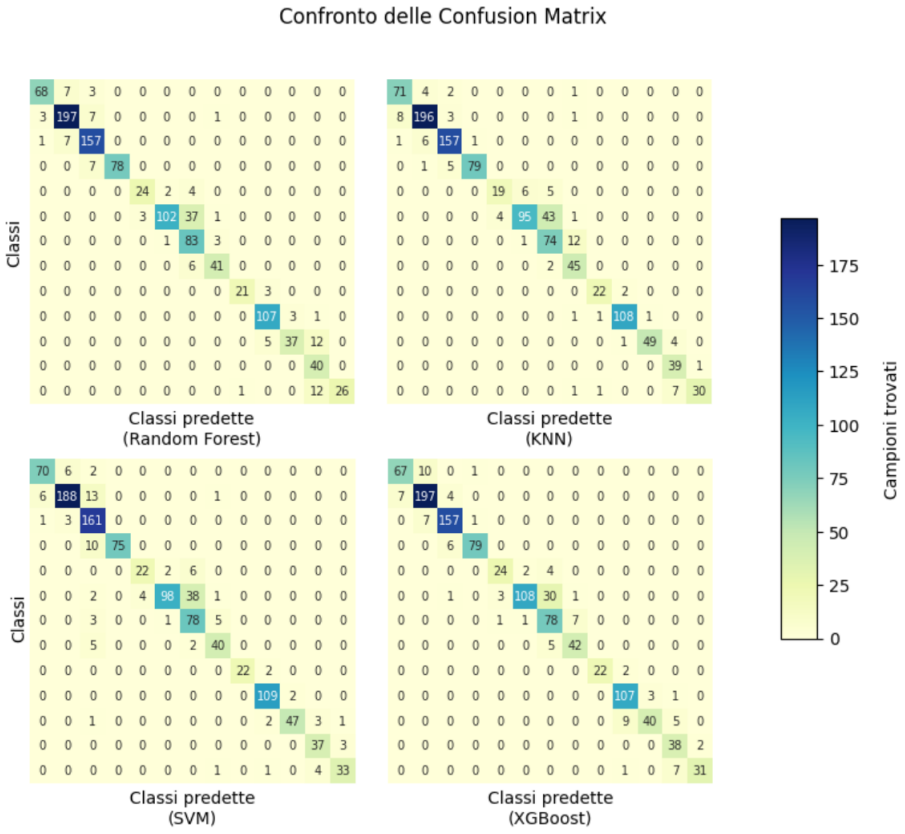

FIGURA 5.8: Confronto delle Confusion Matrix per BUILDING_FLOOR

5.3.2 LATITUDE

R^2-score

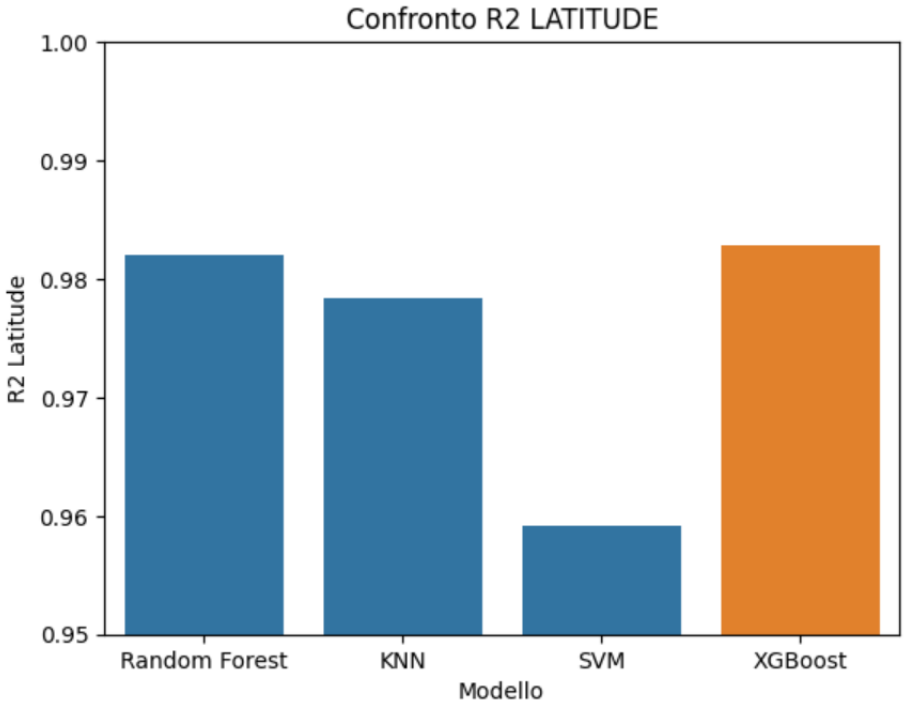

FIGURA 5.9: Confronto R^2-score per LATITUDE

MSE

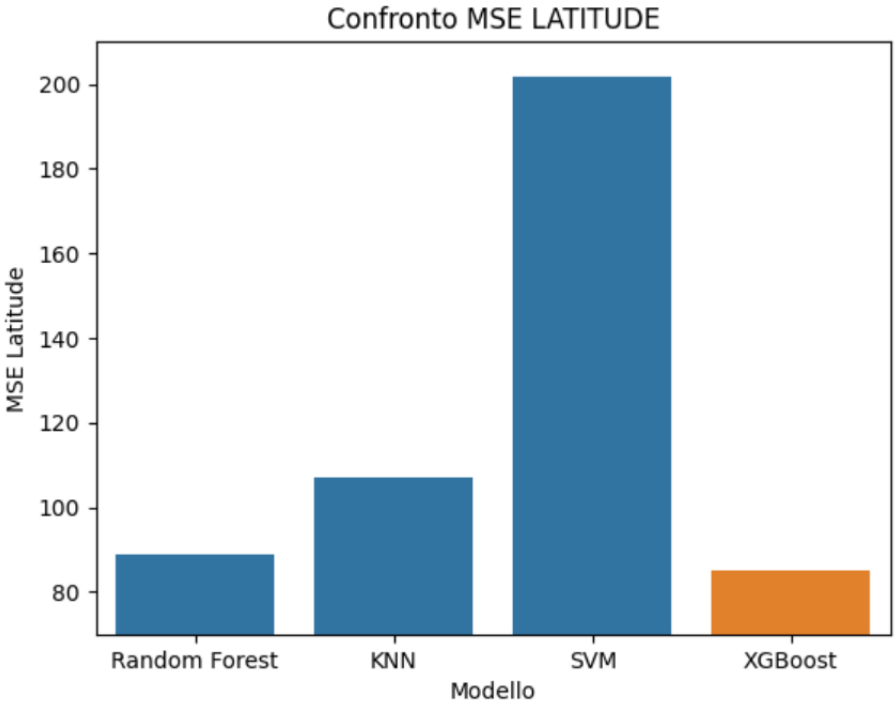

FIGURA 5.10: Confronto MSE per LATITUDE

MAE

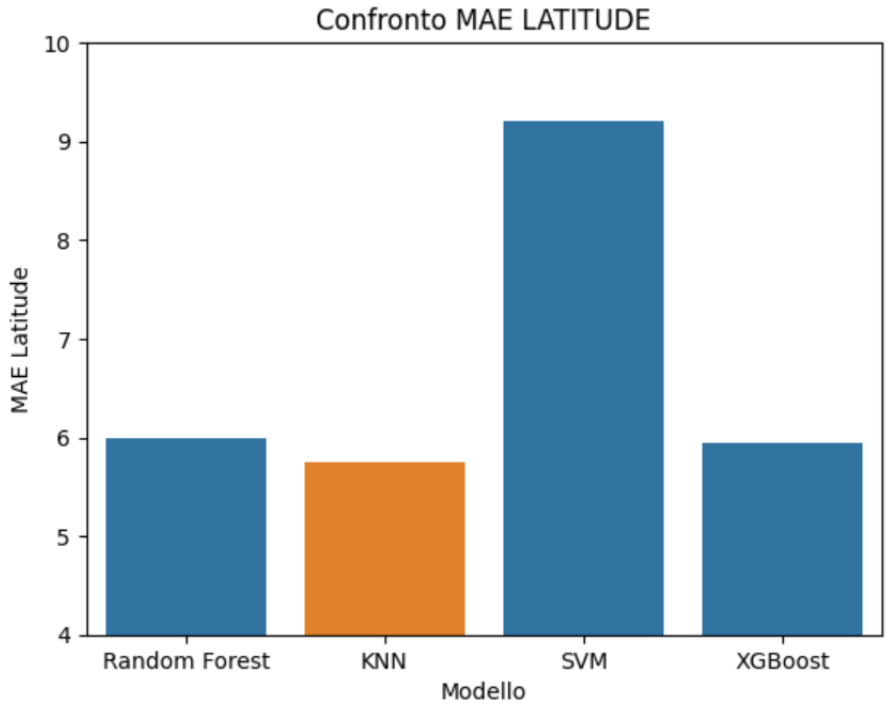

FIGURA 5.11: Confronto MAE per LATITUDE

5.3.3 LONGITUDE

R^2-score

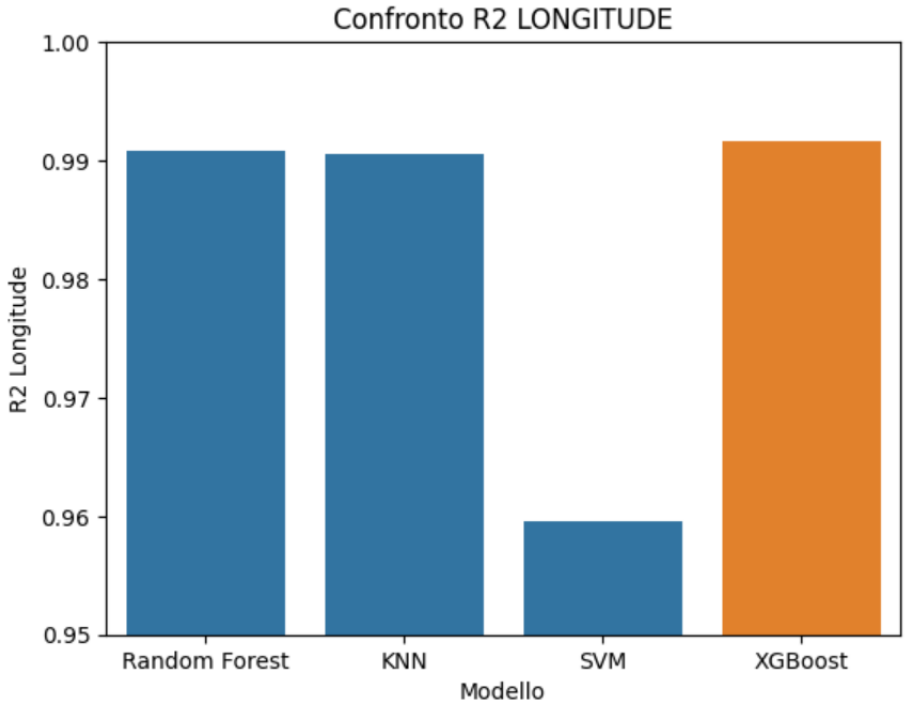

FIGURA 5.12: Confronto R^2-score per LONGITUDE

MSE

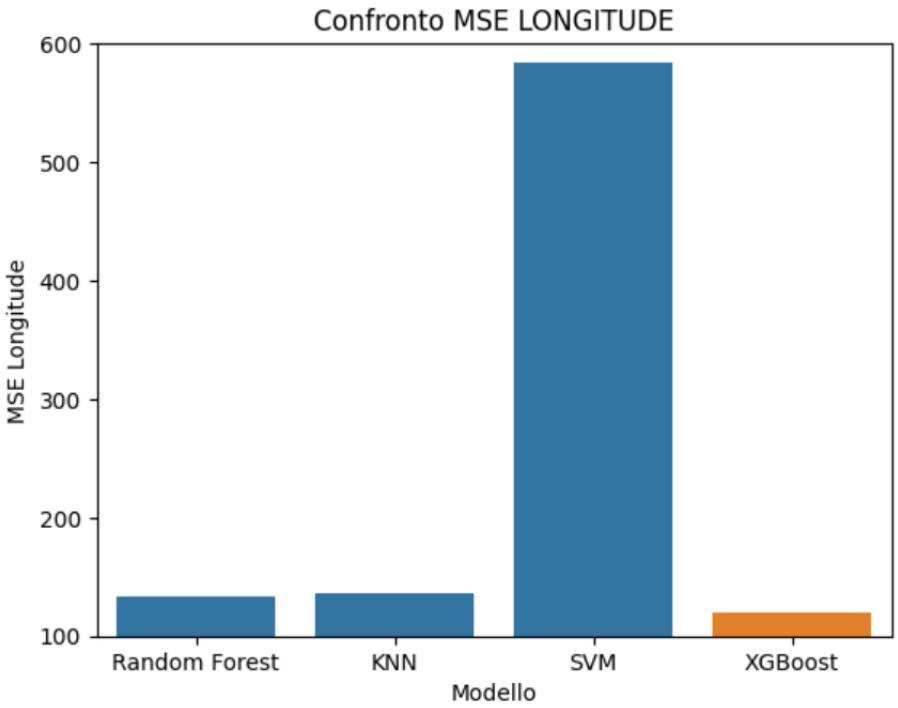

FIGURA 5.13: Confronto MSE per LONGITUDE

MAE

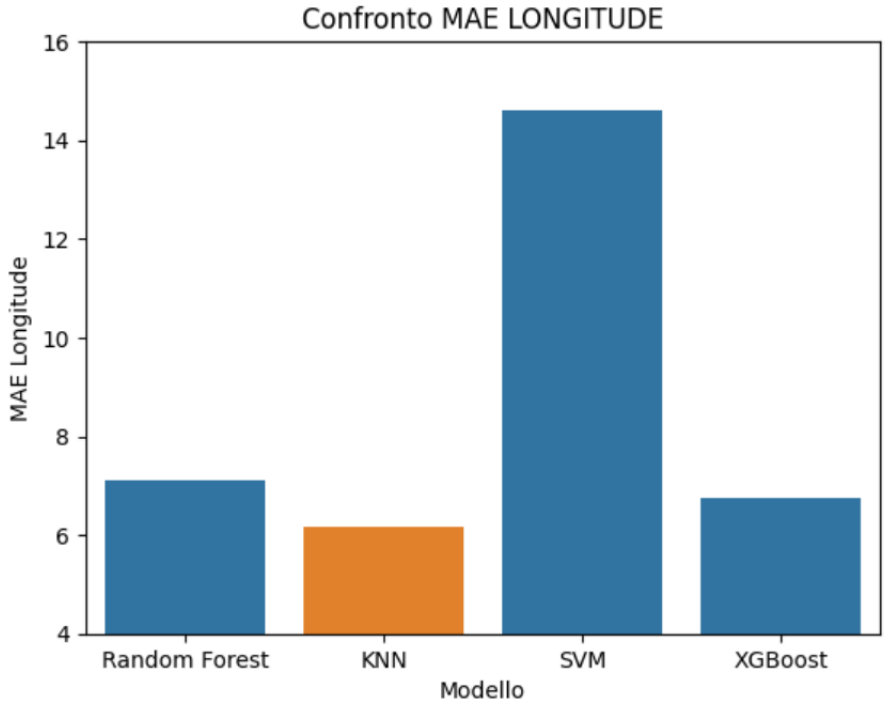

FIGURA 5.14: Confronto MAE per LONGITUDE

Capitolo 6

Deploy dei modelli performanti

In seguito all'analisi dei risultati ottenuti nella fase di sviluppo del modello, è emerso chiaramente che l'algoritmo XGBoost costituisce la scelta più adatta tra le selezionate in questa tesi per affrontare il problema di localizzazione indoor. Questa conclusione si basa sia sulla dimensione dei modelli generati (Figura 4.11) che sulle prestazioni metriche viste nel capitolo precedente, ad eccezione della metrica MAE, in cui KNN sembra presentare un vantaggio marginale (Figure 5.11 e 5.14).

Per il deploy del modello, si è fatto ricorso alla piattaforma Streamlit, un framework che consente la creazione di dashboard interattive in modo semplice e veloce. Utilizzando Streamlit Share come piattaforma di hosting, è stato possibile rendere l'applicazione accessibile online tramite una repository su GitHub.

6.1 Creazione della dashboard

La dashboard è stata progettata per offrire una visualizzazione chiara e intuitiva delle predizioni generate dal modello. Il codice relativo alla fase di predizione (vista nella sezione 5.1) è stato integrato nella dashboard, sfruttando una struttura modulare basata su funzioni richiamate nel main. Le funzioni aggiunte sono relative al caricamento dei dati (Figura 6.1), al caricamento dei modelli (Figura 6.2) e alla preparazione per le predizioni (Figura 6.3), dove possiamo notare il comando *@st.cache_data* all'inizio di ognuna di queste funzioni. Ciò ha ottimizzato le prestazioni dell'applicazione, memorizzando i dati, i modelli e le predizioni in modo da evitare di dover eseguire le funzioni e ricalcolare gli output di quest'ultime ad ogni avvio, che invece verranno semplicemente caricati dalla cache.

Per l'esecuzione degli algoritmi di predizione saranno chiamate le funzioni in Figura 6.4; per il modello di classificazione per la predizione di BUILDING_FLOOR verrà richiamata la funzione con l'encoding. Inoltre, l'utilizzo

di una pipeline per la preparazione dei dati di input, come visto nella sezione 4.1.3 ha semplificato il processo di sviluppo dell'applicazione, consentendo al modello di gestire automaticamente le trasformazioni viste nelle sezioni 3.3, 3.4, 3.5.

```
@st.cache_data
def carica_dataset_test():
    #Caricamento del dataset.
    data = pd.read_csv('./Data/validationData.csv')
    #Drop colonne superflue
    data.drop(['SPACEID','RELATIVEPOSITION','USERID','PHONEID','TIMESTAMP']
              , axis=1, inplace=True )
    #Creazione BUILDING_FLOOR
    data['BUILDING_FLOOR'] = data['BUILDINGID']*10 + data['FLOOR']
    return data
```

FIGURA 6.1: Funzione di caricamento del test set su *dashboard.py*

```
# Funzione caricamento modelli basati su XGBoost
@st.cache_data()
def carica_modelli():
    with open('modelli/xgboost_buildingfloor.pkl', "rb") as f1:
        buildingfloor_model = pickle.load(f1)
    with open('modelli/xgboost_latitude.pkl', "rb") as f2:
        latitude_model = pickle.load(f2)
    with open('modelli/xgboost_longitude.pkl', "rb") as f3:
        longitude_model = pickle.load(f3)

    return buildingfloor_model, latitude_model, longitude_model
```

FIGURA 6.2: Funzione di caricamento dei modelli performanti su *dashboard.py*

```python
@st.cache_data
def carica_target_train():
    data = pd.read_csv('./Data/trainingData.csv')
    data['BUILDING_FLOOR'] = data['BUILDINGID']*10 + data['FLOOR']
    data = data[['BUILDING_FLOOR']]
    return data

# Funzione creazione features e modifiche
@st.cache_data
def estrapola_features(data):
    X_test = data[[i for i in data.columns if 'WAP' in i]]
    X_test = X_test.replace(to_replace=100,value=-111)
    return X_test

# Funzione creazione targets e modifiche
@st.cache_data
def estrapola_targets(data):
    y_test = data[[i for i in data.columns if not 'WAP' in i]]
    y_test.drop(['FLOOR','BUILDINGID'], axis=1, inplace=True)
    y_test = y_test.replace(to_replace=100,value=-111)
    return y_test
```

FIGURA 6.3: Funzioni di preparazione alla predizione su *dashboard.py*

```python
def predizione_encode(buildingfloor_model, X_test):
    #avvio encoding
    le = LabelEncoder()
    target = carica_target_train()
    le.fit_transform(target['BUILDING_FLOOR'])

    #predizione BUILDING_FLOOR
    predizione_encoded = buildingfloor_model.best_estimator_.predict(X_test)

    #reverse encoding
    predizione = le.inverse_transform(predizione_encoded)

    return predizione

def predizione(model, X_test):
    predizione = model.best_estimator_.predict(X_test)
    return predizione
```

FIGURA 6.4: Funzioni di predizione su *dashboard.py*

6.2 Visualizzazione della dashboard

La dashboard è strutturata in modo intuitivo, consentendo all'utente di selezionare una riga specifica dal dataset di test e visualizzare la posizione reale confrontata con le posizioni predette dai modelli. Un'intera mappa del dataset di test viene generata, con la possibilità di selezionare la riga desiderata tramite un menu a tendina nella sidebar. I risultati vengono visualizzati graficamente, con le posizioni reali rappresentate in nero e le posizioni predette

dai modelli indicate con un punto rosso, come è possibile osservare in figura 6.5.

Ogni volta che viene selezionata una riga, viene generato un grafico che mostra la posizione reale e le posizioni predette dai modelli. Inoltre, una barra inferiore fornisce i dettagli della riga selezionata, inclusi i dati reali e le metriche di valutazione.

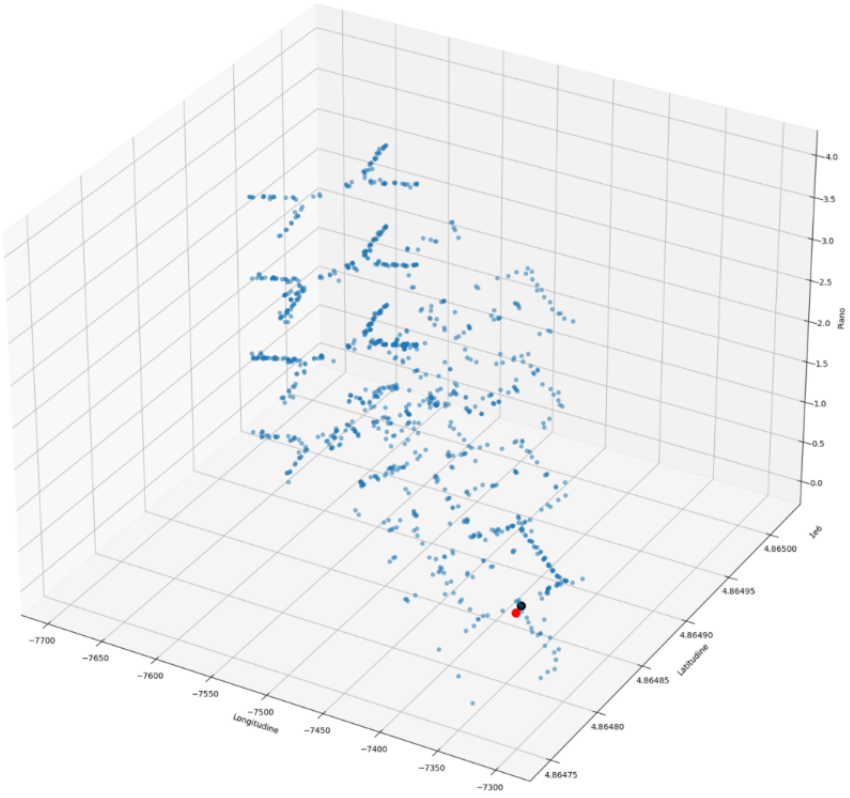

FIGURA 6.5: Visualizzazione della posizione reale e predetta su Streamlit

6.3 Deploy dell'applicazione

Il processo di deploy dell'applicazione è stato agevolato dal pulsante di deploy integrato in Streamlit, che ha semplificato notevolmente il caricamento dell'applicazione su Streamlit Share. Tuttavia, è emersa una sfida relativa alla dimensione del dataset di training, che superava il limite di 25MB imposto da GitHub per il caricamento tramite browser. Per superare questa limitazione, è stata utilizzata la funzionalità di collegamento della cartella locale alla repository GitHub tramite comandi Git, consentendo così il caricamento del dataset di training.

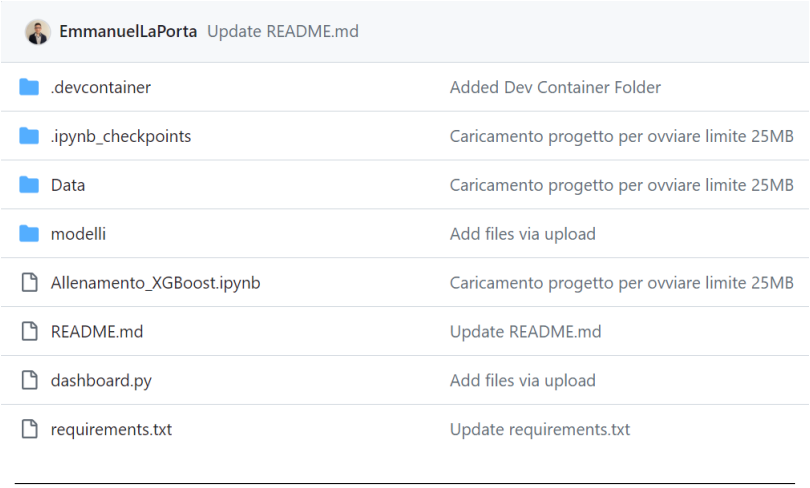

FIGURA 6.6: Repository dei dati necessari per l'applicazione su GitHub

Con questi passaggi, l'applicazione è stata correttamente deployata e resa accessibile online tramite Streamlit Share, visitabile dal seguente link: https://laporta-tesilocalizationindoor.streamlit.app.

Capitolo 7

Conclusione

La ricerca condotta ed i risultati delle predizioni sul test set hanno evidenziato che i modelli basati su XGBoost risultano più performanti nello scenario considerato sia in termini di dimensioni sia in termini di metriche analizzate, sebbene anche alternative come Random Forest e K-nearest Neighbor abbiano dato ottimi risultati, in particolare quest'ultimo per ciò che riguarda il MAE.

Inoltre, l'approccio della pipeline si è dimostrato fondamentale nell'integrazione di tecniche come la normalizzazione, la selezione delle feature mediante il threshold di varianza e l'applicazione della PCA, il che ha portato ad un significativo miglioramento delle prestazioni dei modelli. Questa metodologia, oltre a garantire risultati ottimali, si è dimostrata altamente scalabile per eventuali implementazioni future, in particolare per il processo di deploy, poiché tutto è integrato direttamente con il modello.

Un altro punto chiave del progetto è stata la creazione di una dashboard interattiva utilizzando Streamlit, che ha reso accessibili e comprensibili i risultati ottenuti dai modelli performanti. La distribuzione della dashboard su Streamlit Share e su GitHub ha reso possibile la condivisione e la collaborazione con altri ricercatori e stakeholder, facilitando la diffusione dei risultati e l'accessibilità del progetto.

Guardando al futuro, si aprono interessanti prospettive di sviluppo. Una possibile direzione potrebbe essere l'esplorazione di modelli avanzati o l'implementazione di approcci innovativi per affrontare specifiche sfide nel dominio di studio. Inoltre, la continuazione del lavoro sulla dashboard potrebbe includere l'aggiunta di nuove funzionalità e l'ottimizzazione dell'interfaccia utente per una migliore esperienza d'uso.

Infine, l'integrazione dei principi FAIR (Findable, Accessible, Interoperable, Reusable) nel processo di ricerca e condivisione dei dati rappresenta un aspetto fondamentale per garantire la trasparenza, la riproducibilità e l'accessibilità dei risultati ottenuti. L'adozione di tali principi contribuirà a promuovere una cultura della ricerca aperta e a favorire la collaborazione e lo scambio di conoscenze nella comunità scientifica.

La presente ricerca rappresenta solo un punto di partenza, e sono fiducioso

che i risultati ottenuti possano costituire una base solida per futuri sviluppi e contribuire al progresso del campo di studio.

Bibliografia

[1] T. Kim Geok, K. Zar Aung, M. Sandar Aung et al., ≪Review of Indoor Positioning: Radio Wave Technology,≫ *Applied Sciences*, vol. 11, n. 1, 2021, ISSN: 2076-3417. DOI: 10.3390/app11010279. indirizzo: https://www.mdpi.com/2076-3417/11/1/279.

[2] M. Khalaf-Allah, ≪Time of arrival (TOA)-based direct location method,≫ in *2015 16th International Radar Symposium (IRS)*, 2015, pp. 812–815. DOI: 10.1109/IRS.2015.7226229.

[3] A. R. Kulaib, R. M. Shubair, M. A. Al-Qutayri e J. W. P. Ng, ≪An overview of localization techniques for Wireless Sensor Networks,≫ in *2011 International Conference on Innovations in Information Technology*, 2011, pp. 167–172. DOI: 10.1109/INNOVATIONS.2011.5893810.

[4] R. Peng e M. L. Sichitiu, ≪Angle of Arrival Localization for Wireless Sensor Networks,≫ in *2006 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks*, vol. 1, 2006, pp. 374–382. DOI: 10.1109/SAHCN.2006.288442.

[5] C. Yang e H.-r. Shao, ≪WiFi-based indoor positioning,≫ *IEEE Communications Magazine*, vol. 53, n. 3, pp. 150–157, 2015. DOI: 10.1109/MCOM.2015.7060497.

[6] Y. Ma, B. Wang, S. Pei, Y. Zhang, S. Zhang e J. Yu, ≪An Indoor Localization Method Based on AOA and PDOA Using Virtual Stations in Multipath and NLOS Environments for Passive UHF RFID,≫ *IEEE Access*, vol. 6, pp. 31 772–31 782, 2018. DOI: 10.1109/ACCESS.2018.2838590.

[7] X. Tian, R. Shen, D. Liu, Y. Wen e X. Wang, ≪Performance Analysis of RSS Fingerprinting Based Indoor Localization,≫ *IEEE Transactions on Mobile Computing*, vol. 16, n. 10, pp. 2847–2861, 2017. DOI: 10.1109/TMC.2016.2645221.

[8] N. H. A. Wahab, N. Sunar, S. H. Ariffin, K. Y. Wong, Y. Aun et al., ≪Indoor positioning system: A review,≫ *International Journal of Advanced Computer Science and Applications*, vol. 13, n. 6, 2022.

[9] I. Ashraf, Y. B. Zikria, S. Hur e Y. Park, ≪A comprehensive analysis of magnetic field based indoor positioning with smartphones: Opportunities, challenges and practical limitations,≫ *IEEE Access*, vol. 8, pp. 228 548–228 571, 2020.

[10] B. Li, T. Gallagher, A. G. Dempster e C. Rizos, ≪How feasible is the use of magnetic field alone for indoor positioning?≫ In *2012 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, 2012, pp. 1–9. DOI: 10.1109/IPIN.2012.6418880.

[11] R. Zhang, F. Hoflinger e L. Reindl, ≪Inertial Sensor Based Indoor Localization and Monitoring System for Emergency Responders,≫ *IEEE Sensors Journal*, vol. 13, n. 2, pp. 838–848, 2013. DOI: 10.1109/JSEN.2012.2227593.

[12] J. Moutinho, R. Araújo e D. Freitas, ≪Indoor localization with audible sound — Towards practical implementation,≫ *Pervasive and Mobile Computing*, vol. 29, pp. 1–16, 2016, ISSN: 1574-1192. DOI: https://doi.org/10.1016/j.pmcj.2015.10.016. indirizzo: https://www.sciencedirect.com/science/article/pii/S157411921500200X.

[13] J. Qi e G.-P. Liu, ≪A Robust High-Accuracy Ultrasound Indoor Positioning System Based on a Wireless Sensor Network,≫ *Sensors*, vol. 17, n. 11, 2017, ISSN: 1424-8220. DOI: 10.3390/s17112554. indirizzo: https://www.mdpi.com/1424-8220/17/11/2554.

[14] D. Arbula e S. Ljubic, ≪Indoor Localization Based on Infrared Angle of Arrival Sensor Network,≫ *Sensors*, vol. 20, n. 21, 2020, ISSN: 1424-8220. DOI: 10.3390/s20216278. indirizzo: https://www.mdpi.com/1424-8220/20/21/6278.

[15] W. Ding, F. Yang, H. Yang et al., ≪A hybrid power line and visible light communication system for indoor hospital applications,≫ *Computers in Industry*, vol. 68, pp. 170–178, 2015, ISSN: 0166-3615. DOI: https://doi.org/10.1016/j.compind.2015.01.006. indirizzo: https://www.sciencedirect.com/science/article/pii/S0166361515000160.

[16] U. Bolat e M. Akcakoca, ≪A hybrid indoor positioning solution based on Wi-Fi, magnetic field, and inertial navigation,≫ in *2017 14th Workshop on Positioning, Navigation and Communications (WPNC)*, IEEE, 2017, pp. 1–6.

[17] I.-G. Lee, D. B. Kim, J. Choi et al., ≪WiFi HaLow for Long-Range and Low-Power Internet of Things: System on Chip Development and Performance Evaluation,≫ *IEEE Communications Magazine*, vol. 59, n. 7, pp. 101–107, 2021. DOI: 10.1109/MCOM.001.2000815.

[18] B. Kim, M. Kwak, J. Lee e T. T. Kwon, ≪A multi-pronged approach for indoor positioning with WiFi, magnetic and cellular signals,≫ in *2014 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, 2014, pp. 723–726. DOI: 10.1109/IPIN.2014.7275551.

[19] F. S. Daniş e A. T. Cemgil, ≪Model-Based Localization and Tracking Using Bluetooth Low-Energy Beacons,≫ *Sensors*, vol. 17, n. 11, 2017, ISSN: 1424-8220. DOI: 10.3390/s17112484. indirizzo: https://www.mdpi.com/1424-8220/17/11/2484.

[20] B. Badihi, M. U. Sheikh, K. Ruttik e R. Jäntti, «On Performance Evaluation of BLE 5 In Indoor Environment: An Experimental Study,» in *2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications*, 2020, pp. 1–5. DOI: 10.1109/PIMRC48278.2020.9217132.

[21] C. M. Ramya, M Shanmugaraj e R Prabakaran, «Study on ZigBee technology,» in *2011 3rd International Conference on Electronics Computer Technology*, vol. 6, 2011, pp. 297–301. DOI: 10.1109/ICECTECH.2011.5942102.

[22] J. Zhou e J. Shi, «RFID Localization Algorithms and Applications—A Review,» *Journal of Intelligent Manufacturing*, vol. 20, pp. 695–707, 2009. DOI: https://doi.org/10.1007/s10845-008-0158-5. indirizzo: https://link.springer.com/article/10.1007/s10845-008-0158-5.

[23] S. Gezici, Z. Tian, G. Giannakis et al., «Localization via ultra-wideband radios: a look at positioning aspects for future sensor networks,» *IEEE Signal Processing Magazine*, vol. 22, n. 4, pp. 70–84, 2005. DOI: 10.1109/MSP.2005.1458289.

[24] N. M. Tiglao, M. Alipio, R. Dela Cruz, F. Bokhari, S. Rauf e S. A. Khan, «Smartphone-based indoor localization techniques: State-of-the-art and classification,» *Measurement*, vol. 179, p. 109 349, 2021, ISSN: 0263-2241. DOI: https://doi.org/10.1016/j.measurement.2021.109349. indirizzo: https://www.sciencedirect.com/science/article/pii/S0263224121003444.

[25] Y. Yang, P. Dai, H. Huang, M. Wang e Y. Kuang, «A Semi-Simulated RSS Fingerprint Construction for Indoor Wi-Fi Positioning,» *Electronics*, vol. 9, n. 10, 2020, ISSN: 2079-9292. DOI: 10.3390/electronics9101568. indirizzo: https://www.mdpi.com/2079-9292/9/10/1568.

[26] A. Roxin, J. Gaber, M. Wack e A. Nait-Sidi-Moh, «Survey of Wireless Geolocation Techniques,» in *2007 IEEE Globecom Workshops*, 2007, pp. 1–9. DOI: 10.1109/GLOCOMW.2007.4437809.

[27] Z. Yang e Y. Liu, «Quality of Trilateration: Confidence-Based Iterative Localization,» *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, n. 5, pp. 631–640, 2010. DOI: 10.1109/TPDS.2009.90.

[28] V. Osa, J. Matamales, J. F. Monserrat e J. López, «Localization in wireless networks: the potential of triangulation techniques,» *Wireless personal communications*, vol. 68, pp. 1525–1538, 2013.

[29] X. Zhao, T. Rautiainen, K. Kalliola e P. Vainikainen, «Path-Loss Models for Urban Microcells at 5.3 GHz,» *IEEE Antennas and Wireless Propagation Letters*, vol. 5, pp. 152–154, 2006. DOI: 10.1109/LAWP.2006.873950.

[30] H.-H. Hsu, W.-J. Peng, T. K. Shih, T.-W. Pai e K. L. Man, «Smartphone Indoor Localization with Accelerometer and Gyroscope,» in *2014 17th International Conference on Network-Based Information Systems*, 2014, pp. 465–469. DOI: 10.1109/NBiS.2014.72.

[31] X. Hou e J. Bergmann, «Pedestrian Dead Reckoning With Wearable Sensors: A Systematic Review,» *IEEE Sensors Journal*, vol. 21, n. 1, pp. 143–152, 2021. DOI: 10.1109/JSEN.2020.3014955.

[32] N. Singh, S. Choe e R. Punmiya, «Machine Learning Based Indoor Localization Using Wi-Fi RSSI Fingerprints: An Overview,» *IEEE Access*, vol. 9, pp. 127 150–127 174, 2021. DOI: 10.1109/ACCESS.2021.3111083.

[33] L. Zhang, Y. Li, Y. Gu e W. Yang, «An efficient machine learning approach for indoor localization,» *China Communications*, vol. 14, n. 11, pp. 141–150, 2017. DOI: 10.1109/CC.2017.8233657.

[34] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen e J. S. Rellermeyer, «A Survey on Distributed Machine Learning,» *ACM Comput. Surv.*, vol. 53, n. 2, 2020, ISSN: 0360-0300. DOI: 10.1145/3377454. indirizzo: https://doi.org/10.1145/3377454.

[35] M. Usama, J. Qadir, A. Raza et al., «Unsupervised Machine Learning for Networking: Techniques, Applications and Research Challenges,» *IEEE Access*, vol. 7, pp. 65 579–65 615, 2019. DOI: 10.1109/ACCESS.2019.2916648.

[36] T. Sakai, M. C. du Plessis, G. Niu e M. Sugiyama, «Semi-Supervised Classification Based on Classification from Positive and Unlabeled Data,» in *Proceedings of the 34th International Conference on Machine Learning*, D. Precup e Y. W. Teh, cur., ser. Proceedings of Machine Learning Research, vol. 70, PMLR, 2017, pp. 2998–3006. indirizzo: https://proceedings.mlr.press/v70/sakai17a.html.

[37] L. P. Kaelbling, M. L. Littman e A. W. Moore, «Reinforcement learning: A survey,» *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[38] Y. Reich e S. Barai, «Evaluating machine learning models for engineering problems,» *Artificial Intelligence in Engineering*, vol. 13, n. 3, pp. 257–272, 1999, ISSN: 0954-1810. DOI: https://doi.org/10.1016/S0954-1810(98)00021-1. indirizzo: https://www.sciencedirect.com/science/article/pii/S0954181098000211.

[39] F. Pedregosa, G. Varoquaux, A. Gramfort et al., «Scikit-learn: Machine Learning in Python,» *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[40] F. Chollet et al., *Keras*, https://keras.io, 2015.

[41] A. Paszke, S. Gross, F. Massa et al., «PyTorch: An Imperative Style, High-Performance Deep Learning Library,» in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035. indirizzo: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[42] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2021. indirizzo: https://www.R-project.org/.

[43] J. Torres-Sospedra, R. Montoliu, A. Martnez-Us, T. Arnau e J. Avariento, «UJIIndoorLoc,» 2014. DOI: https://doi.org/10.24432/C5MS59.

[44] E. Bisong, «Google Colaboratory,» in *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Berkeley, CA: Apress, 2019, pp. 59–64, ISBN: 978-1-4842-4470-8. DOI: 10.1007/978-1-4842-4470-8_7. indirizzo: https://doi.org/10.1007/978-1-4842-4470-8_7.

Financial support from: PNRR MUR project PE0000013-FAIR acknowledged.

Ringraziamenti

Desidero esprimere la mia gratitudine al Prof. Patti e al Prof. La Delfa per la loro guida e il loro sostegno durante tutto il percorso di questa tesi.

Un grazie di cuore va a mia mamma e mio papà, per il loro amore incondizionato e il loro continuo supporto, anche nei momenti più difficili.

Ringrazio i miei coinquilini, vecchi e nuovi, per aver reso questi anni della triennale un'esperienza indimenticabile.

Un pensiero speciale va anche ai miei amici del paese, che mi hanno sempre incoraggiato e supportato.

Grazie a tutte le persone che mi hanno supportato attivamente nel percorso universitario, anche solo per un consiglio, e a coloro che mi hanno offerto il loro sostegno morale, anche solo per un momento.

E infine, un grazie a mio nonno. Te l'avevo promesso.