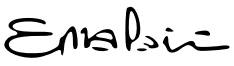
SAPIENZA
UNIVERSITÀ DI ROMA

Sviluppo di un Modello di Machine Learning per l'Applicazione di Smart Parking 'GeneroCity': Creazione di Dati Sintetici e Analisi Preliminare

Facoltà di Ingegneria dell'informazione, informatica e statistica
Informatica

Francesco D'Aprile
Matricola 2016953

Relatore
Prof. Emanuele Panizzi



Anno Accademico 2023/2024

Sviluppo di un Modello di Machine Learning per l'Applicazione di Smart Parking 'GeneroCity': Creazione di Dati Sintetici e Analisi Preliminare

Sapienza Università di Roma

Questa tesi è stata composta con LaTeX e la classe Sapthesis.

Email dell'autore: daprile.francesco02@gmail.com

Sommario

La presente relazione presenta il lavoro svolto durante il tirocinio presso il GamificationLab, un laboratorio del Dipartimento di Informatica dell'Università La Sapienza di Roma, specializzato nella ricerca nei campi della Gamification e dell'Interazione Uomo-Macchina. L'obiettivo principale del tirocinio è stato contribuire allo sviluppo dell'applicazione di smart parking *GeneroCity*, con un focus sulla costruzione preliminare di un modello di Machine Learning basato sui dati forniti dai sensori dell'applicazione.

GeneroCity è un'applicazione per il parcheggio che utilizza sensori virtuali per stimare se l'utente si trovi nel proprio veicolo o meno. Questi sensori non sono dispositivi hardware tradizionali, ma modelli software che analizzano i dati raccolti da diverse fonti hardware per determinare lo stato dell'utente in modo intelligente. In tempo reale, l'applicazione monitora lo stato di ciascun sensore e decide se l'utente è a piedi o alla guida.
Poiché i sensori di *GeneroCity* erano ancora in fase di sviluppo e non erano completamente operativi, non è stato possibile utilizzare un modello di apprendimento automatico basato su dati reali per prendere decisioni finali. Pertanto, è stato necessario creare un dataset di dati sintetici che emulasse i dati raccolti dai sensori dell'applicazione.
Per generare questi dati sintetici, sono stati somministrati questionari a un gruppo di utenti automobilisti. Questi questionari hanno raccolto informazioni sulle abitudini degli utenti, come l'utilizzo della macchina, la frequenza degli spostamenti, l'impiego lavorativo e i dispositivi utilizzati. L'obiettivo di questa procedura era di creare dati sintetici quanto più possibile verosimili rispetto a quelli che sarebbero stati ottenuti dai sensori reali durante l'uso dell'applicazione.

Una volta creato il dataset, è stato possibile sperimentare diversi modelli di Intelligenza Artificiale per classificare lo stato dell'utente in base alle letture simulate dei sensori. Questo approccio permetterà di valutare e migliorare l'efficacia dei modelli di Machine Learning nell'interpretare correttamente i dati raccolti, una volta che i sensori reali saranno pienamente operativi.

Indice

Capitolo 1

Introduzione

I dati raccolti dall'Istituto nazionale di statistica (ISTAT) forniscono una visione dettagliata del parco veicolare in Italia. Secondo le elaborazioni dell'ISTAT per l'anno 2022, il numero totale di veicoli in circolazione nel nostro Paese si attesta a 53,7 milioni di unità, di cui 40,2 milioni sono autovetture. Inoltre, secondo le statistiche di Eurostat, nel 2022 l'Italia contava in media 684 veicoli ogni 1000 abitanti. Questo elevato numero di veicoli ha comportato un aumento del traffico urbano e un significativo impatto ambientale e sociale, soprattutto nelle grandi città.[1]

Effettuando una analisi sui dati storici relativi alla crescita del numero di veicoli in circolazione censiti dall'ANFIA[1], si evidenzia un trend significativo che mette in luce una necessità sempre più urgente: la gestione efficace e sostenibile dei parcheggi urbani, il cosiddetto smart parking, ossia sistemi che attraverso una rete di sensori e telecamere monitorano la disponibilità degli spazi di parcheggio. Gli utenti possono consultare questi dati direttamente da un'applicazione consentendo di trovare parcheggio risparmiando tempo e riducendo il traffico veicolare.

Tra il 1980 e il 2021, il numero totale di veicoli si è più che raddoppiato. Ad esempio, nel 1980 si contavano 19 milioni di veicoli, mentre nel 2021 si è arrivati a oltre 44 milioni di veicoli. Questo aumento vertiginoso è accompagnato da una densità veicolare sempre più elevata, con un conseguente impatto significativo sulle infrastrutture urbane, in particolare sulla disponibilità di parcheggi.[2]

Con l'incremento del numero di veicoli, la congestione del traffico è diventata una problematica di primaria importanza, soprattutto nelle aree metropolitane. Un aspetto spesso trascurato ma estremamente rilevante è la ricerca del parcheggio. Secondo studi recenti, una percentuale significativa del traffico urbano è generata da veicoli che cercano parcheggio. Questo fenomeno non solo aumenta la congestione, ma contribuisce anche all'inquinamento atmosferico e acustico.
Per affrontare questi problemi, l'adozione di sistemi di *smart parking* diventa cruciale. Un sistema di parcheggio intelligente utilizza tecnologie avanzate, come sensori IoT, intelligenza artificiale e app mobile, per ottimizzare l'uso degli spazi di parcheggio disponibili e ridurre il tempo di ricerca da parte degli automobilisti. Tali sistemi

[1] ANFIA - Associazione Nazionale Filiera Industria Automobilistica - è una delle maggiori associazioni di categoria in Italia. Nata nel 1912, da oltre 100 anni ha l'obiettivo di rappresentare gli interessi delle Associate nei confronti delle istituzioni pubbliche e private, nazionali e internazionali e di provvedere allo studio e alla risoluzione delle problematiche tecniche, economiche, fiscali, legislative, statistiche e di qualità del comparto automotive. Chi siamo ANFIA

possono ottimizzare l'uso degli spazi tramite i sensori che monitorano in tempo reale la disponibilità dei posti auto, permettendo una distribuzione più efficiente. Inoltre si ridurrebbe il traffico indicando agli automobilisti i parcheggi disponibili, si riduce il traffico legato alla ricerca di posti liberi e di conseguenza si ridurrebbe l'inquinamento.

La crescita storica del parco veicoli suggerisce che, senza interventi mirati, il problema della gestione dei parcheggi è destinato a peggiorare. Se si rapporta il numero di posti auto a pagamento con la popolazione residente, esistono 9 città (tra cui Genova, Milano, Palermo) dove l'esigenza di parcheggio è più sentita in quanto ci sono 100 abitanti per posto auto a pagamento; la situazione migliore - che vede meno di 20 abitanti per posto auto - riguarda nove città (tra cui Bologna, Bolzano, Pisa), mentre sono 24 quelle dove il numero medio di abitante per posto auto a pagamento varia da 20 a 50 mentre sono 12 (tra cui Roma, Foggia, Nuoro) quelle dove varia da 50 a 100[3]. L'implementazione di sistemi di smart parking non è solo una risposta efficace ai problemi attuali, ma rappresenta una componente fondamentale per lo sviluppo di città intelligenti e sostenibili in futuro.

In conclusione l'analisi dei dati storici evidenzia un aumento impressionante del numero di veicoli e una conseguente necessità di gestire efficacemente gli spazi di parcheggio. La soluzione risiede nell'adozione di tecnologie di smart parking, che non solo migliorano l'efficienza, ma contribuiscono anche a una mobilità urbana più sostenibile. Con l'aumento continuo del numero di veicoli, l'implementazione di tali sistemi diventa non solo auspicabile, ma essenziale per il futuro delle città.

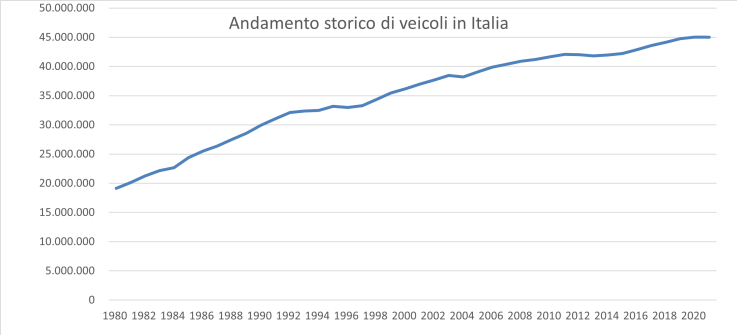


Figura 1.1. Andamento storico del parco veicoli in Italia

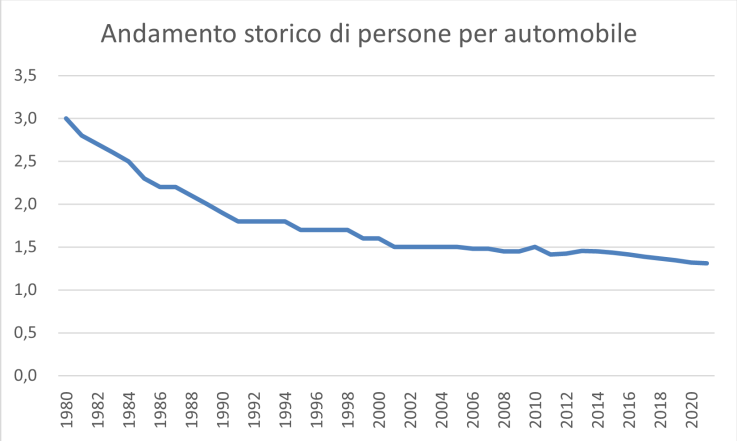


Figura 1.2. Andamento storico di persone per automobile

1.1 GeneroCity

GeneroCity è un'app mobile di smart parking progettata per migliorare l'esperienza di guida degli utenti rendendola più fluida e meno stressante. Attualmente, un gruppo di studenti e ricercatori del Dipartimento di Informatica presso La Sapienza Università di Roma sta lavorando al suo sviluppo. Quando sarà completata, l'app sarà disponibile pubblicamente su App Store e Play Store. L'applicazione ha come obiettivo principale quello di facilitare lo scambio di parcheggi tra utenti. Il processo è semplice: un utente, chiamato *offerente*, segnala la disponibilità del proprio posto auto in un luogo specifico. Un altro utente, denominato *richiedente*, può quindi esprimere interesse per quel parcheggio. Il sistema si occupa di abbinare gli offerenti e i richiedenti, e se l'accordo viene finalizzato, il match viene considerato completato.

Figura 1.3. Logo GeneroCity

Per garantire una maggiore sicurezza durante la guida, GeneroCity adotta un'interazione implicita, riducendo al minimo la necessità di interventi manuali. L'app sfrutta i sensori del telefono, come Bluetooth, GPS, accelerometro etc., per monitorare l'attività dell'utente. Grazie ai modelli di machine learning implementati, l'app può determinare se l'utente è fermo, in movimento a piedi, in auto o se è attivamente alla ricerca di un parcheggio.
Inoltre, per aumentare il coinvolgimento degli utenti, GeneroCity integra un sistema di gamification. Gli utenti accumulano monete virtuali, chiamate *GCoins*, attraverso la cessione dei loro parcheggi ad altri. Queste monete possono essere utilizzate per prenotare un parcheggio, rendendo l'intero processo più interattivo e incentivante.

In futuro, l'app potrebbe anche introdurre funzionalità avanzate, come notifiche in tempo reale per opportunità di parcheggio, integrazioni con altri servizi di mobilità urbana e analisi dei dati per migliorare ulteriormente l'esperienza dell'utente.

Gli ambiti su cui si è concentrato il lavoro del tirocinio sono diversi:

- Analisi e rappresentazione grafica dei dati restituiti dal sensore Motion Sensor (capitolo 3).
- Generazione di dati sintetici dei sensori di Generocity (capitolo 4).

- Progettazione e sviluppo di un modello di Machine Learning che, utilizzando i dati sintetici dei sensori, identifichi se l'utente è alla guida in un dato momento (capitolo 5).

Capitolo 2

Architettura di GeneroCity

2.1 Struttura generale dell'applicazione

Generocity è una applicazione mobile che ha come struttura quella di un modello client-server.
E' composta da frontend e backend, che comunicano tramite un'API REST.

- **Client:** Questa è la parte dell'applicazione con cui l'utente interagisce direttamente, ovvero l'app mobile installata sul dispositivo. Il client si occupa di presentare l'interfaccia utente, raccogliere i dati inseriti dall'utente e inviare richieste al server. Sul client è installata la parte del programma frontend. Nel caso di Generocity il client è l'applicazione Android o IOS che raccoglie informazioni dall'esterno (tramite i sensori) e gli input dell'utente per poi interfacciarsi tramite richieste HTTP con il backend sul server.
- **Server:** Il server è la parte centrale dell'applicazione, che gestisce i dati, esegue la logica di business e fornisce i servizi necessari al client. Il server è solitamente ospitato su un computer o in un cloud e non è direttamente accessibile dall'utente. Su di esso è posta la sezione backend del programma. E' in ascolto per richieste da parte dei client e in più può inviare informazioni anche se non richieste in modalità *push*.

2.1.1 HTTP

HTTP (HyperText Transfer Protocol) è il protocollo alla base del World Wide Web, che regola la comunicazione tra client e server su Internet per la trasmissione di documenti ipertestuali. È un protocollo applicativo basato su TCP per garantire l'affidabilità e il riordinamento dei pacchetti.

Figura 2.1. Logo HTTP

Principi Fondamentali e Funzionamento

La comunicazione HTTP avviene attraverso un ciclo di richieste e risposte:

- **Richiesta:** Il client invia una richiesta al server utilizzando un Uniform Resource Identifier (URI).
- **Risposta:** Il server elabora la richiesta e restituisce una risposta con il codice di stato HTTP, gli header e il corpo del messaggio.

Un messaggio HTTP è composto da:

- **Riga di richiesta/risposta:** Specifica il tipo di richiesta o il codice di stato della risposta.
- **Intestazioni (headers):** Forniscono metadati aggiuntivi, come il tipo di contenuto e la lunghezza del contenuto.
- **Corpo (body):** Contiene i dati effettivi, come il contenuto di una pagina HTML o i dati di un file caricato.

Metodi delle Richieste HTTP

I principali metodi HTTP includono:

Metodo	Descrizione
GET	Richiede una risorsa senza modificarla.
POST	Invia dati al server per creare una nuova risorsa.
PUT	Aggiorna o crea una risorsa esistente.
DELETE	Elimina la risorsa selezionata.
HEAD	Richiede solo le intestazioni della risorsa.
OPTIONS	Descrive i metodi consentiti per la risorsa.

Tabella 2.1. Descrizione dei Metodi HTTP

Codici di Risposta HTTP

I codici di risposta sono suddivisi in classi:

1xx Informativo: Indicano che la richiesta è in corso.

- 100 Continue: La richiesta è parzialmente ricevuta.
- 101 Switching Protocols: Il server cambia protocollo.

2xx Successo: Indicano che la richiesta è stata completata con successo.

- 200 OK: La richiesta è riuscita.
- 201 Created: Nuova risorsa creata.
- 204 No Content: Richiesta completata senza contenuti da restituire.

3xx Redirezione: Richiedono un'azione su una nuova risorsa.

- 301 Moved Permanently: Risorsa spostata definitivamente.
- 302 Found: Risorsa spostata temporaneamente.
- 304 Not Modified: Risorsa non modificata, usa la cache.

4xx Errore del Client: La richiesta è errata.

- 400 Bad Request: Richiesta malformata.
- 401 Unauthorized: Autenticazione necessaria.
- 404 Not Found: Risorsa non trovata.

5xx Errore del Server: Errore interno del server.

- 500 Internal Server Error: Errore imprevisto sul server.
- 502 Bad Gateway: Risposta non valida ricevuta da un altro server.
- 503 Service Unavailable: Server temporaneamente non disponibile.

2.1.2 API

Le API, acronimo di *Application Programming Interface*, sono strumenti fondamentali nell'ingegneria del software, progettati per permettere la comunicazione tra sistemi differenti. Si configurano come un insieme di protocolli, routine e definizioni che consentono a un'applicazione di interagire con un'altra, garantendo l'accesso controllato a funzionalità o dati. Le API rappresentano il canale attraverso il quale software eterogenei possono cooperare, senza esporre direttamente il codice sorgente, migliorando così la modularità e la sicurezza.
Un'API stabilisce le modalità di interazione tra componenti software, permettendo a un'applicazione di effettuare richieste di dati o di eseguire operazioni su un'altra applicazione. Questo tipo di interfaccia è cruciale per l'integrazione di servizi esterni, per l'interazione con servizi web e per il coordinamento tra i vari moduli di un sistema complesso come il client e il server di una applicazione.

Componenti fondamentali di un'API

- **Endpoint**: Gli endpoint costituiscono i punti di accesso attraverso cui le risorse o i servizi messi a disposizione dall'API possono essere utilizzati. Ogni endpoint è solitamente identificato da un URL univoco che indirizza l'applicazione richiedente alla risorsa desiderata o all'operazione da eseguire.
- **Formato dei dati**: Le risposte fornite da un'API, contenenti i dati richiesti, sono generalmente strutturate in formati standard come JSON, un formato basato su testo per archiviare e scambiare dati in un modo che sia leggibile dall'uomo e analizzabile dalla macchina; oppure XML, acronimo di Extensible Markup Language che consente di definire e archiviare i dati in modo condivisibile. Questi formati garantiscono un'interoperabilità elevata tra diversi sistemi, semplificando la manipolazione e la gestione dei dati ricevuti.
- **Protocollo di comunicazione**: Le API si basano su protocolli di comunicazione che specificano le modalità con cui i sistemi si scambiano informazioni. Tali protocolli definiscono i formati dei messaggi, i metodi di trasmissione, l'ordine delle operazioni e altri dettagli cruciali che assicurano una comunicazione efficace e standardizzata tra le parti coinvolte. Il protocollo HTTP è uno dei più comuni, ma non l'unico; altri includono HTTPS, SMTP, e WebSocket.
- **Token di autenticazione**: Per garantire l'accesso sicuro alle risorse, molte API implementano meccanismi di autenticazione tramite token. Questi token, che sono stringhe uniche e segrete, permettono di verificare l'identità del richiedente e di autorizzare l'accesso solo a utenti legittimi.
- **Documentazione**: Una API ben strutturata è accompagnata da una documentazione completa e chiara, che descrive in dettaglio come utilizzare ogni endpoint, specificando i parametri richiesti, le possibili risposte, e gli errori comuni. La documentazione è essenziale per facilitare l'integrazione e l'uso corretto dell'API da parte degli sviluppatori.

Approcci principali nella progettazione di API

La progettazione delle API può essere realizzata seguendo diversi modelli architetturali, tra cui i più diffusi sono il modello *SOAP* e il modello *REST*.

- **SOAP (Simple Object Access Protocol)**: SOAP è un protocollo di messaggistica altamente strutturato che permette l'interazione tra applicazioni via rete. Le comunicazioni SOAP possono avvenire su diversi protocolli di trasporto, come HTTP o SMTP, e utilizzano XML per la codifica dei messaggi. Questo modello è caratterizzato da una rigorosa aderenza a standard formali, rendendolo particolarmente adatto per ambienti in cui la sicurezza, l'affidabilità e la transazionalità sono critici. Tuttavia, la complessità e la verbosità di SOAP lo rendono meno flessibile rispetto ad altre soluzioni.
- **REST (Representational State Transfer)**: REST è un modello architetturale che sfrutta le potenzialità del protocollo HTTP per la gestione delle risorse. In REST, le operazioni sono tipicamente mappate sui metodi HTTP (GET, POST, PUT, DELETE), con una chiara corrispondenza alle operazioni CRUD (Create, Read, Update, Delete). Le risorse sono rappresentate attraverso stati trasferibili, solitamente in formato JSON, che ne consentono una manipolazione diretta e intuitiva. REST è ampiamente apprezzato per la sua semplicità, flessibilità e facilità di implementazione, oltre che per la sua capacità di scalare facilmente in ambienti distribuiti. Questo modello è stato scelto come base architetturale per lo sviluppo dell'applicazione *GeneroCity*, sfruttando la sua efficienza e leggerezza per garantire una comunicazione rapida e sicura tra i vari componenti del sistema.

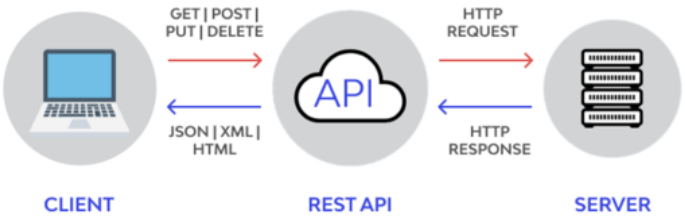


Figura 2.2. Schema di funzionamento delle API REST

2.2 Tecnologie utilizzate da Generocity

Le tecnologie alla base di GeneroCity sono molteplici e vengono impiegate per garantire prestazioni ottimali. Di seguito viene fornita una descrizione delle principali tecnologie utilizzate per lo sviluppo del backend.

OpenAPI, utilizzato per definire in modo standardizzato le API RESTful. Questo approccio permette di progettare, documentare e gestire le interfacce in modo preciso e uniforme, facilitando l'integrazione tra i diversi componenti del sistema. Le specifiche delle API sono definite in formato YAML o JSON, e strumenti come Swagger UI generano automaticamente una documentazione interattiva in HTML,

rendendo più semplice la comprensione e l'utilizzo delle API.

Go, un linguaggio di programmazione open-source sviluppato da Google, è stato selezionato per lo sviluppo del backend di GeneroCity in virtù delle sue caratteristiche distintive. Caratterizzato da una sintassi ispirata al linguaggio C, Go si pone come obiettivo la realizzazione di software efficiente, scalabile e facile da mantenere.
Un elemento di differenziazione fondamentale di Go risiede nell'approccio alla concorrenza. Mediante l'utilizzo di goroutine e channel, il linguaggio offre un modello di concorrenza leggero e intuitivo, che consente di gestire efficacemente un elevato numero di attività contemporanee senza incorrere nei classici problemi legati alla gestione dei thread tradizionali. Le goroutine, in particolare, sono funzioni che permettono l'esecuzione concorrenziale, mentre i channel permettono la sincronizzazione e la comunicazione tra queste.

L'utilizzo di Go per il backend di GeneroCity è motivato dalla necessità di sviluppare un'applicazione in grado di gestire un carico di lavoro elevato e di scalare in modo efficiente. Le caratteristiche di Go, come la gestione della memoria automatica, la tipizzazione statica e la compilazione rapida, si rivelano fondamentali per raggiungere questi obiettivi.
Inoltre, l'ecosistema di librerie di terze parti per Go è in continua crescita, offrendo un ampio ventaglio di strumenti e funzionalità. In particolare, l'esistenza di numerose librerie HTTP semplifica significativamente lo sviluppo delle API RESTful di GeneroCity, consentendo di implementare un'interfaccia di programmazione robusta e performante mantenendo l'astrazione ad alto livello.

Figura 2.3. Logo GO

Docker, una piattaforma di containerizzazione che offre un meccanismo efficiente per isolare e distribuire applicazioni. Un container Docker, essenzialmente un processo isolato, include tutto il necessario per l'esecuzione di un'applicazione: codice, runtime, librerie e configurazioni. Questa granularità consente di creare ambienti di esecuzione consistenti e riproducibili, minimizzando i conflitti tra applicazioni e garantendo una portabilità elevata.
Rispetto alle tradizionali macchine virtuali, i container Docker condividono il kernel del sistema operativo host, offrendo un livello di isolamento inferiore ma con un impatto significativamente minore sulle risorse di sistema. Questa caratteristica li rende particolarmente adatti per la distribuzione di microservizi e per l'orchestrazione di ambienti complessi.

GeneroCity sfrutta Docker per standardizzare l'ambiente di sviluppo e produzione, garantendo che l'applicazione funzioni in modo coerente in ogni fase del suo ciclo di vita. La containerizzazione consente inoltre di semplificare i processi di deployment e di scaling, favorendo l'agilità e la flessibilità dello sviluppo.

MariaDB, un sistema di gestione di database relazionale (RDBMS) open-source originato da un fork di MySQL, è stato scelto per gestire i dati di GeneroCity grazie alla sua combinazione di affidabilità, performance e flessibilità.
Una caratteristica fondamentale di MariaDB, e più in generale dei database relazionali, è il supporto alle transazioni. Una transazione è una sequenza di operazioni che viene trattata come un'unità atomica, ovvero o viene eseguita completamente o non viene eseguita affatto. Questo garantisce la coerenza e l'integrità dei dati, anche in caso di errori o interruzioni. Le proprietà ACID (Atomicità, Consistenza,

Isolamento, Durabilità) definiscono le caratteristiche essenziali di una transazione in un database relazionale.

Per gestire fisicamente i dati, MariaDB utilizza diversi storage engine. In GeneroCity, è stato scelto InnoDB, un motore di archiviazione general-purpose noto per la sua robustezza e affidabilità. InnoDB supporta pienamente le transazioni ACID, garantendo così la sicurezza e l'integrità dei dati. Inoltre, offre funzionalità avanzate come l'isolamento a livello di riga, la gestione dei blocchi e la ripresa in caso di guasto, rendendolo adatto a un'ampia gamma di applicazioni.

Figura 2.4. Logo MariaDB

2.3 Sensori di Generocity

Un sensore è un componente hardware o software che è in ascolto sullo stato del dispositivo o sull'ambiente esterno. I sensori vengono utilizzati per raccogliere informazioni per poi effettuare operazioni, raccoglimento di dati o prendere decisioni. I sensori di GeneroCity non sono sensori hardware, ma modelli software che, scegliendo un elemento hardware, danno una loro stima sullo stato dell'utente (in macchina o no) in modo intelligente. I sensori sono ancora in fase di sviluppo e non tutte le loro modalità operative sono state completamente definite. Di conseguenza, verrà successivamente illustrata l'analisi condotta sul loro funzionamento prima della generazione dei dati sintetici (sezione 4.2.2).

Capitolo 3

Analisi e rappresentazione grafica dei dati restituiti dal sensore Motion Sensor

Questa sezione illustra il lavoro preliminare svolto durante il periodo di tirocinio, antecedente allo sviluppo del progetto relativo alla generazione di dati sintetici dei sensori e alla progettazione di modelli di Machine Learning. Tale attività si è concentrata sull'analisi dei dati forniti dal sensore Apple Motion Sensor e sulla successiva rappresentazione grafica dei risultati ottenuti.

3.1 Apple Motion Sensor

Il sensore di movimento integrato nei dispositivi Apple rappresenta un complesso sistema di rilevamento che, combinando dati provenienti da diversi sensori hardware, è in grado di fornire informazioni dettagliate sull'attività dell'utente, in particolare lo stato di movimento e la modalità di esso (a piedi, in bicicletta, in automobile ecc).

3.1.1 Principio di funzionamento

Il cuore del sistema è costituito da un'*Inertial Measurement Unit* (IMU), un componente elettronico di misurazione inerziale che comprende un insieme di dispositivi hardware:

- **Accelerometro:** Misura le variazioni di velocità lungo tre assi spaziali, fornendo informazioni sulla forza e la direzione dei movimenti.
- **Giroscopio:** Rileva la rotazione angolare del dispositivo attorno a tre assi, consentendo di determinare l'orientamento nello spazio.
- **Magnetometro:** Misura l'intensità e la direzione del campo magnetico terrestre, fornendo un riferimento per l'orientamento.

A questi sensori si aggiunge spesso un barometro, che misura la pressione atmosferica e può essere utilizzato per stimare variazioni di altitudine.

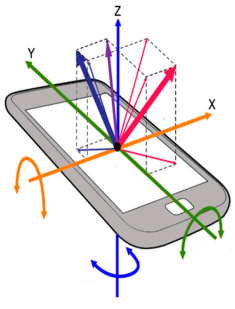


Figura 3.1. Accelerometro

3.1.2 Riconoscimento dell'attività

Attraverso l'integrazione dei dati forniti da questi sensori, il sensore di movimento di Apple, supportato da algoritmi avanzati, è in grado di:

- Classificare l'attività: Distinguere tra diverse attività fisiche come camminata, corsa, guida, ciclismo, ecc.
- Calcolare parametri fisiologici: Stimare la distanza percorsa, il ritmo cardiaco (in combinazione con altri sensori), le calorie bruciate e altri parametri rilevanti per il fitness.
- Determinare il contesto: Riconoscere se l'utente si trova in un ambiente statico o dinamico, se è seduto, in piedi o in movimento.

3.1.3 Considerazioni

Nonostante gli algoritmi implementati nel software del Motion Sensor di Apple presentino un elevato livello di accuratezza, in alcune circostanze non riescono a fornire risultati corretti, generando falsi positivi e falsi negativi rispetto allo stato reale dell'utente. Inoltre, sebbene sia possibile accedere alle letture del Motion Sensor, Apple non fornisce una completa trasparenza riguardo al processo di acquisizione e gestione dei dati. Non è chiaro, ad esempio, se queste letture possano essere modificate successivamente o quale sia il tempo di campionamento tra le rilevazioni. A causa di questa opacità, è stato necessario condurre uno studio approfondito per comprendere meglio il funzionamento del sensore. Tale indagine ha anche evidenziato che l'impiego del set di sensori utilizzati dall'applicazione comporta un consumo energetico significativo, riducendo l'efficienza complessiva del sistema e influendo negativamente sulla durata della batteria del dispositivo.

3.2 Logs Motion Sensor

Un log è un registro sequenziale di eventi o messaggi generati da un sistema, applicazione software o dispositivo. Esso cattura informazioni dettagliate su operazioni eseguite, errori, avvisi, stati e altri eventi significativi che avvengono durante l'esecuzione di un sistema. I log sono strumenti essenziali per il monitoraggio, la diagnosi, e la manutenzione di sistemi complessi, poiché consentono di tracciare il comportamento del sistema nel tempo e di identificare anomalie o malfunzionamenti. I log possono contenere diverse informazioni, tra cui timestamp (marcatori temporali) per indicare quando un evento si è verificato, messaggi descrittivi per spiegare la natura dell'evento, e, in alcuni casi, dati tecnici più specifici come codici di errore, livelli di gravità, o dettagli sulle risorse coinvolte.

Nell'applicazione *GeneroCity*, oltre ai log standard utilizzati per monitorare il funzionamento del sistema e le operazioni eseguite, sono presenti anche i log specifici del *Motion Sensor*, i quali vengono utilizzati principalmente a scopo di debug. Questi log consentono agli sviluppatori di tracciare e diagnosticare eventuali anomalie nel comportamento del sensore, fornendo informazioni dettagliate sui dati raccolti e sul processo di rilevazione del movimento. In tal modo, è possibile ottimizzare le prestazioni del sistema, identificando e correggendo eventuali errori o inefficienze nell'interpretazione dei dati del sensore. All'interno dell'interfaccia grafica dell'applicazione, attualmente in fase di sviluppo, è presente un pulsante che permette ai dispositivi Apple di inviare i dati raccolti dal *Motion Sensor* nelle ultime 24 ore.

Questi dati, una volta inviati, vengono memorizzati in un'apposita tabella dei log all'interno del database relazionale, consentendo un'analisi dettagliata delle informazioni relative al movimento rilevato dal sensore.
Attraverso l'endpoint HTTPS '/logs/' utilizzando il metodo `HTTP GET`, è possibile effettuare richieste aggiungendo parametri specifici, tra cui il token di autenticazione, il limite di record da restituire, il codice identificativo dell'utente associato al log e l'intervallo di tempo desiderato.

3.3 Analisi in locale dei dati remoti

Il team di sviluppo dell'applicazione aveva delle incertezze riguardo l'affidabilità e la tempestività dei valori restituiti da questo sensore. In particolare, si voleva verificare che il sensore non modificasse lo stato dell'utente con un ritardo rispetto a quanto indicato e gli intervalli di aggiornamento dei dati. Pertanto, il lavoro si è concentrato sull'analisi approfondita del comportamento del sensore, al fine di comprenderne meglio le dinamiche operative e valutare la precisione delle informazioni fornite.
Successivamente, si è rivelato utile visualizzare graficamente i valori restituiti dal sensore, aggregandoli per formare periodi di tempo durante i quali l'utente si trovava nello stesso stato. Questa visualizzazione ha permesso di valutare l'affidabilità dei dati forniti, facilitando l'individuazione di eventuali falsi positivi e falsi negativi.
Inizialmente, l'analisi dei dati è stata effettuata implementando un programma in linguaggio Python su una macchina locale. Questo programma ha permesso di esaminare i log del Motion Sensor salvati sul database del server di GeneroCity, accedendo ai dati tramite le API fornite.

3.3.1 Estrazione dei dati dai log

Poiché i log sono salvati in formato testuale, con ogni valore restituito dal *Motion Sensor* associato a un timestamp specifico per tutta la giornata e separato dagli altri tramite il carattere '|', la prima fase del processo consiste nell'estrazione dei dati dalla stringa contenuta nella sezione *message* del log. Questa fase prevede una serie di operazioni di suddivisione del testo. Al termine di questo processo, si ottiene un array di dizionari in cui le chiavi rappresentano le informazioni essenziali per ciascun valore restituito:

- ID: codice identificativo del dato
- Timestamp: data e ora specificata del dato
- Stato: stato dell'utente in quel determinato timestamp, i valori possibili sono molteplici:
  - unknown: stato sconosciuto
  - stationary: l'utente è fermo
  - walking: l'utente sta camminando a piedi
  - running: l'utente sta correndo
  - automotive: l'utente sta viaggiando in automobile
  - cycling: l'utente sta viaggiando in bicicletta
- Confidence: indica quanto il sensore sia confidente in quel valore.

3.3.2 Calcolo delle statistiche

Per analizzare il comportamento di questo sensore, sono state calcolate delle statistiche riguardanti la frequenza con cui i valori vengono restituiti, considerando sia gli stati stabili sia le transizioni tra diversi stati. Inoltre, sono stati esaminati anche i cambiamenti nei valori restituiti dal sensore per verificare se, una volta dichiarato uno stato in un determinato timestamp, questo venisse successivamente modificato.

Per avviare questa analisi, è stata preliminarmente esaminata la frequenza di ripetizione dei timestamp, ovvero le rilevazioni di stato che condividono lo stesso istante temporale. I risultati hanno evidenziato una media significativa di tre ripetizioni per ciascun valore temporale, suggerendo una certa ridondanza nei dati raccolti dal *Motion Sensor*. Analizzando le differenze tra i valori associati a queste ripetizioni, è emerso che lo stato dichiarato non veniva modificato successivamente al timestamp iniziale. Tuttavia, si osservavano variazioni nella confidence, nell'ID del dato, mentre nella maggior parte dei casi i dati restavano totalmente invariati.
Per analizzare i tempi di risposta del sensore, è stata effettuata una semplificazione raggruppando tutti gli stati sotto la categoria *walking*, ad eccezione dello stato *automotive*, in modo da ridurre l'analisi a due soli stati distinti. Successivamente, sono stati esaminati i tempi intercorrenti tra due timestamp consecutivi, focalizzandosi sulle quattro possibili transizioni tra questi stati: walking-walking, walking-automotive, automotive-walking, e automotive-automotive.

Per ciascuna di queste combinazioni, sono state calcolate le seguenti statistiche: la media degli intervalli di tempo, lo scostamento medio dalla media, la durata minima e la durata massima. I risultati dell'analisi hanno evidenziato che, mediamente, tra due stati consecutivi di tipo walking-walking trascorrono diversi minuti. Il valore massimo rilevato per questa transizione supera spesso un'ora, un risultato che si verifica tipicamente durante il periodo notturno, quando lo smartphone rimane completamente fermo, non richiedendo quindi aggiornamenti di stato.

Per le transizioni tra gli stati walking e automotive, l'intervallo temporale è risultato essere molto breve, con una media di 50 secondi e un massimo di 2-3 minuti. La transizione automotive-automotive ha mostrato una media simile di 50 secondi, ma con picchi massimi che possono raggiungere i 10 minuti.
Questi risultati sono stati ottenuti analizzando un campione composto da una dozzina di log registrati da 4 utenti diversi.

3.3.3 Rappresentazione grafica

Dopo aver condotto l'analisi statistica sui dati, si è ritenuto opportuno rappresentare graficamente i *trip*[1] registrati dal *Motion Sensor*. Questo passaggio è stato intrapreso al fine di ottenere una visione più chiara dell'affidabilità dei valori rilevati e per identificare e analizzare eventuali falsi positivi e falsi negativi. Per costruire i trip, sono stati raggruppati i dati con lo stesso stato, combinandoli in un unico record con un tempo di inizio e un tempo di fine. I falsi positivi e falsi negativi sono stati identificati come quei trip in cui l'utente rimane in un determinato stato per un periodo inferiore a una soglia preimpostata di 120 secondi. Questo criterio è stato adottato per considerare errati quei trip che indicano un cambiamento di stato in un intervallo di tempo troppo breve, il che risulterebbe incoerente con un comportamento realistico dell'utente, suggerendo un'errata rilevazione da parte del sensore.

[1] viaggio, intervallo di tempo in cui l'utente ha lo stesso stato

Ad esempio, se i dati indicano che l'utente è stato in macchina per soli 50 secondi, ciò viene classificato come un falso positivo. Analogamente, se durante un trip in macchina si registra un breve intervallo di 10 secondi in cui l'utente è considerato a piedi, questo viene considerato un falso negativo.

Per la rappresentazione grafica dei dati è stata utilizzata la libreria Matplotlib di Python. È stato creato un istogramma con altezza della barra fissa, in modo da garantire una leggibilità ottimale degli stati nel grafico. Sull'asse delle ascisse è stato riportato il tempo, mentre i diversi colori utilizzati identificano lo stato dell'utente in quel momento. Per distinguere visivamente i falsi positivi e negativi, sono stati assegnati colori differenti a ciascuno di essi.

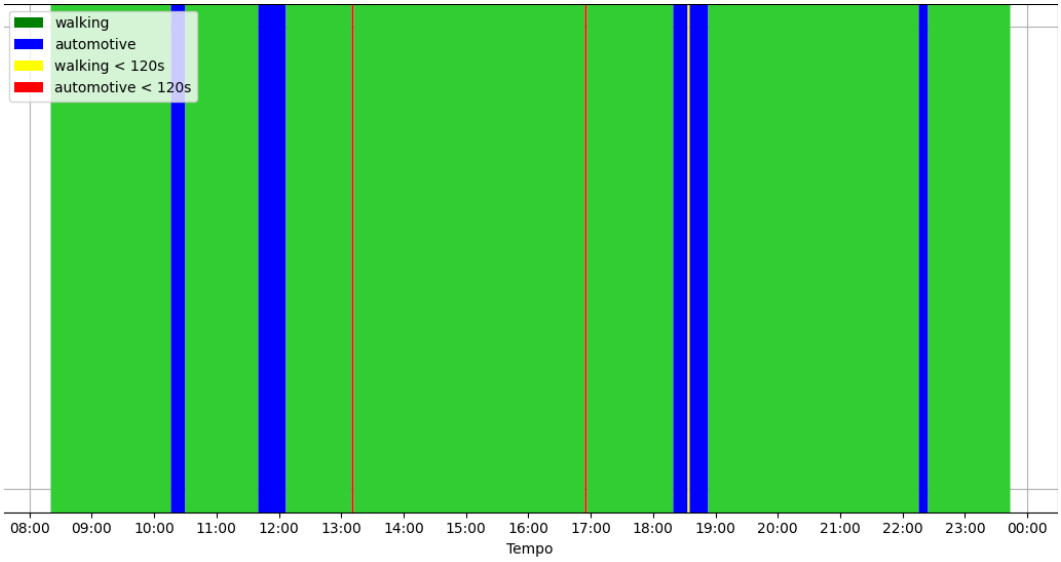


Figura 3.2. Rappresentazione grafica dei dati restituiti dal sensore Motion Sensor

Come illustrato nella figura 3.2, è possibile che il *Motion Sensor* restituisca cambi di stato errati. Ad esempio, tra le 18:30 e le 19:00, il sensore ha registrato che l'utente fosse in stato di walking per un breve intervallo di 30 secondi, mentre per il resto del tempo ha indicato che l'utente fosse in macchina. Questo cambiamento di stato, rilevato in un intervallo di tempo troppo breve, suggerisce un'anomalia nei dati, che può essere classificata come un falso negativo.

3.4 Rappresentazione grafica tramite Grafana

A seguito dell'analisi preliminare dei dati e della generazione dei primi grafici, è emerso un particolare interesse nella rappresentazione interattiva e in tempo reale dei dati forniti dal sensore Motion Sensor, con riferimento specifico al tempo di generazione del log. Il grafico risultante è stato integrato in un pannello all'interno del laboratorio Gamification Lab, consentendo un monitoraggio costante e immediato dell'andamento dei dati raccolti.

3.4.1 Grafana

Grafana è una piattaforma open-source utilizzata per l'analisi, la visualizzazione e il monitoraggio dei dati, largamente adottata in ambienti di ingegneria del software e DevOps per migliorare l'osservabilità di sistemi complessi. La sua principale caratteristica è la capacità di aggregare, esplorare e visualizzare in tempo reale metriche, log e flussi di dati provenienti da diverse applicazioni e infrastrutture. Una delle funzionalità più distintive di Grafana è la possibilità di creare dashboard personalizzate, che consentono di rappresentare i dati attraverso vari tipi di grafici e indicatori interattivi. Queste dashboard offrono una panoramica chiara e immediata delle prestazioni del sistema o dei processi monitorati, aggiornandosi automaticamente per permettere un monitoraggio continuo.

Figura 3.3. Logo Grafana

Nel contesto del progetto GeneroCity, l'acquisizione dei dati avviene attraverso due principali modalità: mediante una connessione diretta al sistema di gestione del database (DBMS) dell'applicazione, oppure tramite l'esposizione e la raccolta delle metriche utilizzando Prometheus.
La piattaforma è inoltre estensibile tramite plugin, permettendo l'integrazione con ulteriori sorgenti di dati e aumentando la sua flessibilità e personalizzazione.

3.4.2 Prometheus

Prometheus è un sistema open source avanzato per il monitoraggio e l'alerting, progettato per raccogliere, archiviare e gestire metriche sotto forma di *time series data*. Questo sistema consente di arricchire le metriche, che sono misurazioni numeriche indipendenti dal sistema monitorato, attraverso l'uso di *label*, elementi aggiuntivi che forniscono un contesto più dettagliato. Il termine time series indica che le metriche vengono registrate nel tempo, permettendo di osservare e analizzare i cambiamenti nel corso del tempo.

Figura 3.4. Logo Prometheus

Prometheus offre librerie client che facilitano l'integrazione con applicazioni sviluppate in vari linguaggi di programmazione, come Go, Java, Python, e molti altri. Un altro componente chiave è il push gateway, che consente a workload specifici di esporre le proprie metriche a Prometheus.
L'architettura di Prometheus è relativamente semplice: il server Prometheus centrale raccoglie le informazioni tramite librerie, exporter e push gateway utilizzando un modello pull. A questo server si collegano l'alert manager, per la gestione degli alert, e strumenti di visualizzazione come Grafana, nonché servizi API personalizzati per l'esportazione delle metriche raccolte.

Nel contesto del progetto Generocity, le metriche sono state esposte tramite il server Prometheus utilizzando librerie client integrate nel backend dell'applicazione, sviluppato in Go.

3.4.3 Esposizione delle metriche

La logica dell'esposizione delle metriche avviene dunque nel backend. Il codice è strutturato per acquisire, elaborare e esporre metriche provenienti dal Motion Sensor,

utilizzando Prometheus come strumento principale per la raccolta e la gestione dei dati di monitoraggio. Inoltre, il codice fa uso di una cache per ottimizzare le operazioni di raccolta e ridurre i tempi di risposta.

Uso della Cache

All'inizio della funzione principale, viene effettuato un controllo sulla cache per determinare se esistono già metriche aggiornate per il sensore di movimento. La cache funge da strato intermedio tra il database e il sistema di monitoraggio, riducendo la necessità di interrogazioni frequenti al database e migliorando l'efficienza complessiva. Se la cache non contiene le informazioni necessarie, il sistema procede con l'aggiornamento delle metriche, memorizzandole poi nuovamente nella cache per utilizzi futuri.
L'uso della cache è strategico per evitare carichi inutili sul database e per garantire che le metriche esposte siano aggiornate, minimizzando allo stesso tempo la latenza. Questo è particolarmente utile in scenari in cui le metriche non cambiano frequentemente o quando è tollerabile un certo ritardo nella visualizzazione dei dati aggiornati.

In questo contesto, la cache è stata utilizzata anche per consentire l'aggiornamento delle metriche una sola volta al giorno, impostando una scadenza di 24 ore per il contenuto memorizzato. Questo approccio garantisce che i grafici rappresentino i viaggi dell'ultimo giorno di un utente per un periodo di 24 ore. Allo scadere di questo intervallo, le metriche vengono aggiornate con nuovi dati, se disponibili; in caso contrario, rimangono invariate fino al successivo aggiornamento.

Raccolta delle Metriche con Prometheus

Una volta determinato che le metriche devono essere aggiornate, il codice interroga il database per ottenere l'ultimo log salvato contenente i dati del sensore di movimento. La funzione che gestisce questa operazione effettua una query sul database, estraendo specifici log etichettati con il tag *Retrieve*. Questi log contengono informazioni cruciali che verranno successivamente elaborate per costruire le metriche da esporre.

La parte centrale del codice relativa a Prometheus si concentra sulla registrazione di un collector personalizzato. Questo collector è responsabile della raccolta e descrizione delle metriche. Viene utilizzata la funzione `prometheus.MustRegister()` per assicurarsi che il collector venga registrato correttamente con Prometheus, rendendo disponibili le metriche per il sistema di monitoraggio.
È stato implementato un nuovo endpoint API denominato 'metrics', che consente l'accesso alle metriche esposte dal sistema. Oltre a queste metriche personalizzate, l'endpoint rende disponibili anche le metriche standard fornite da Prometheus.

Elaborazione delle Informazioni

La fase di elaborazione dei dati provenienti dal log riveste un ruolo cruciale nel processo. La logica del codice per la costruzione dei trip è simile alla versione precedente sviluppata in Python (sezione 3.3.1).

Gestione delle metriche con Timestamp

Una volta elaborati i dati, il sistema genera una metrica per ciascun trip, associando lo stato del movimento a un valore numerico specifico (1 per walking, 2 per

automotive, 3 per falso-walking, e 4 per falso-automotive).
Un problema significativo riscontrato con Prometheus riguarda il suo design, che è principalmente orientato all'esposizione di metriche in tempo reale attraverso contatori o valori numerici aggiornabili. La principale sfida affrontata è stata quella di esporre più dati contemporaneamente, garantendo al contempo una corretta formattazione e visualizzazione all'interno di un unico grafico. Per risolvere questa difficoltà, è stato implementato un collector personalizzato (figura 3.5) in grado di raccogliere e descrivere queste metriche utilizzando un tipo meno comune, denominato `MetricWithTimestamp`. Questo approccio consente di impostare manualmente il timestamp di aggiornamento della metrica, assegnando un timestamp distinto per ogni trip. Il timestamp corrisponde alla data e ora di inizio del trip stesso. Dunque aggirando la problematica riscontrata.

```
type myCollector struct {
        metric    *prometheus.Desc
        value     float64
        timestamp time.Time
}

func (c *myCollector) Describe(ch chan<- *prometheus.Desc) {
        ch <- c.metric
}
func (c *myCollector) Collect(ch chan<- prometheus.Metric) {
        s := prometheus.NewMetricWithTimestamp(c.timestamp,
        ↪  prometheus.MustNewConstMetric(
                c.metric, prometheus.GaugeValue, c.value,
        ))

        ch <- s
}

```

Figura 3.5. Definizione del collector personalizzato

```
func (c *myCollector) UpdateMetricValue() {
        for _, mt := range motionsTrips {
                var value float64

                switch mt.State {
                case "walking":
                        value = 1
                case "automotive":
                        value = 2
                case "FP-walking":
                        value = 3
                case "FP-automotive":
                        value = 4
                }

                c.value = value
                c.timestamp = mt.Start
        }
}
```

Figura 3.6. Assegnazione dei trip al collector

Visualizzazione delle Metriche Estratte in Grafana

Per rappresentare i trip su un grafico all'interno del laboratorio Gamification Lab, è stato utilizzato Grafana. La connessione tra Grafana e i dati esposti da Prometheus è stata effettuata configurando Grafana per interfacciarsi con l'API 'metrics' di Generocity e selezionando la metrica dalla quale prelevare i dati.

Per ottenere un grafico simile a quello prodotto precedentemente utilizzando librerie Python, è stato scelto il modello di Grafana denominato 'State Timeline'. Questo modello consente di visualizzare gli stati attraverso l'uso di colori differenti lungo l'asse temporale.

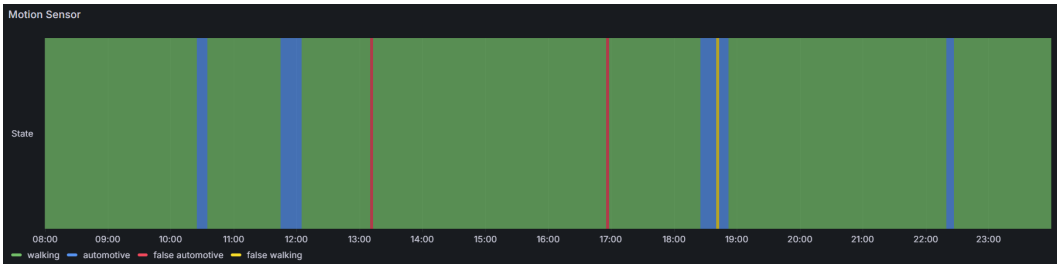


Figura 3.7. Grafico Grafana dei dati restituiti dal Motion Sensor

Capitolo 4

Generazione di dati sintetici dei sensori di Generocity

La generazione di dati sintetici è una tecnica sempre più utilizzata in ambito accademico e industriale per affrontare la carenza di dati reali o per ampliare i dataset esistenti, garantendo al contempo la privacy degli utenti e riducendo i costi associati alla raccolta dei dati senza compromettere la qualità o l'integrità delle informazioni originali[4]. Nel contesto dell'applicazione Generocity, la generazione di dati sintetici assume un ruolo cruciale, poiché permette di simulare scenari realistici prima che termini l'effettivo sviluppo dei sensori e il rilascio dell'applicazione. Questa tecnica non solo consente di testare e validare algoritmi di classificazione preliminarmente, ma anche di esplorare casi d'uso estremi o rari che potrebbero non essere facilmente osservabili nei dati reali.

La generazione di dati sintetici può essere effettuata utilizzando diverse metodologie, tra cui modelli probabilistici, tecniche di simulazione basate su agenti, e approcci di machine learning come le Generative Adversarial Networks (GANs)[5]. Gli ultimi due approcci sono stati esclusi in quanto si basano sull'utilizzo di dati reali come punto di partenza per la generazione di dati sintetici. Tuttavia, nel presente contesto, tali dati non erano disponibili, rendendo necessaria la generazione dei dati sintetici ex novo.
Per il progetto in esame, è stato sviluppato uno script denominato *GeneraSet*[1] che utilizza dati provenienti da questionari somministrati a utenti automobilisti per simulare le loro abitudini e di conseguenza i valori dei sensori nei vari momenti della giornata.

L'approccio adottato in GeneraSet è in linea con le pratiche descritte in letteratura, dove i dati sintetici sono generati partendo da modelli probabilistici costruiti su dati raccolti tramite sondaggi o altre fonti di dati eterogenei[6]. In questo caso, i dati di base provengono da questionari che sondano le abitudini di spostamento degli utenti, compresi dettagli sull'uso di tecnologie come il Bluetooth e Wi-Fi.
Un'ulteriore vantaggio della generazione di dati sintetici è che si garantisce la privacy degli utenti, poiché i dati utilizzati non sono direttamente riconducibili a individui reali.

[1] Nome derivato dalla generazione di dataset per Generocity

4.1 Progettazione e sviluppo dei questionari

I questionari sono stati progettati per raccogliere informazioni dettagliate sulle abitudini di spostamento e sull'utilizzo di tecnologie durante la guida e in altre attività. Questi dati sono stati utilizzati per costruire una simulazione realistica delle settimane tipo di ciascun utente.

Per la realizzazione dei questionari è stata impiegata la piattaforma *Google Form*, uno strumento versatile che offre numerose funzionalità per la creazione di questionari personalizzati. Grazie a Google Form, è stato possibile strutturare i questionari in modo dettagliato, includendo una varietà di tipologie di domande, come risposte a scelta multipla e domande aperte, per raccogliere in modo efficace informazioni specifiche sulle abitudini di spostamento degli utenti e sull'uso delle tecnologie durante la guida. Inoltre, la piattaforma ha facilitato la distribuzione dei questionari e la raccolta delle risposte, le quali sono state automaticamente organizzate e rese disponibili in formato *CSV*. Questo formato ha permesso un'elaborazione successiva dei dati in modo efficiente e preciso.

4.1.1 Struttura del sondaggio

All'interno del sondaggio, sono presenti domande progettate specificamente per fornire dati utilizzati all'interno di GeneraSet. Queste domande sono essenziali per generare i dati sintetici necessari per la simulazione. Tuttavia, sono incluse anche altre domande che, pur non essendo direttamente utilizzate da GeneraSet, servono a creare un contesto più completo per l'utente, migliorando l'affidabilità delle risposte. Inoltre, alcune domande sono state inserite in previsione di un possibile utilizzo futuro, permettendo un eventuale ampliamento delle funzionalità del programma.

Di seguito viene presentato l'elenco delle domande incluse nel questionario, accompagnato da una spiegazione per ciascuna di esse.

Domanda	Motivazione della domanda
Utilizzi una macchina in maniera regolare?	Esclusione dal sondaggio in caso di risposta negativa
Sesso	Profilazione
Età	Profilazione
Hai un telefono Android o Apple?	Profilazione
Ti connetti mai al Bluetooth della tua macchina? Si → Quanto spesso? Si connette in maniera manuale o automatica? Ti ricordi l'ultima volta?	Comprendere l'abitudine dell'uso della tecnologia Bluetooth alla guida.
Quanto spesso ti capita di collegarti al Bluetooth di una macchina altrui come ospite?	Comprendere l'uso della tecnologia in auto non propria, utile per eventuali sviluppi futuri.
La tua macchina ha un sistema di Apple Car Play/Android Auto? Si → Quanto spesso lo utilizzi? Utilizza un sistema con il cavo o wireless?	Comprendere l'abitudine dell'uso della tecnologia Apple Car Play/Android Auto alla guida. La tipologia di connessione influenza i valori di altri sensori durante l'uso.
Quanto spesso ti capita di collegarti a Apple Car Play/Android Auto di una macchina altrui come ospite? Quanto spesso ti capita di mettere in carica il telefono in macchina?	Comprendere l'uso della tecnologia in auto non propria, utile per eventuali sviluppi futuri. Comprendere l'abitudine dell'utente nel ricaricare il dispositivo all'interno del veicolo.
Quando lo carichi in macchina dove lo colleghi?	Comprendere il dispositivo di ricarica all'interno dell'auto. Se il dispositivo è l'auto stessa il sensore questo influenza il valore del sensore di ricarica.
Hai un Wi-Fi in casa che usi spesso dal telefono?	Comprendere l'abitudine dell'uso della tecnologia Wi-Fi nella propria abitazione
Quanto spesso ti capita di collegarti a una rete Wi-Fi con il telefono quando sei a lavoro/Università?	Comprendere l'abitudine dell'uso della tecnologia Wi-Fi durante le attività lavorative o accademiche.
Quanto spesso vai a lavoro/Università in macchina?	Comprendere le abitudini sull'utilizzo dell'automobile per gli spostamenti verso attività lavorative o accademiche
Quanto è lungo approssimatamente il tragitto in macchina? Quanto ci metteresti?	Comprendere la durata dello spostamento in automobile verso attività lavorative o accademiche
Quanti giorni a settimana vai indicativamente a lavoro/università?	Comprendere la frequenza settimanale delle attività lavorative o accademiche
Qual è l'orario in cui parti da casa per andare al lavoro/università solitamente? E l'orario in cui solitamente ritorni a casa?	Comprendere gli orari di spostamento per le attività lavorative e accademiche per generare precisamente i viaggi all'interno della settimana più fedelmente possibile a quelli dell'utente.
Fai qualche esempio di attività che svolgi regolarmente andandoci con la tua auto indicando il giorno e l'ora. Per esempio: venerdì alle 18, sabato alle 12. Oppure lunedì martedì venerdì alle 20. Oppure 3 volte alle 21.	Domanda a risposta aperta per conoscere eventuali ulteriori spostamenti in auto che effettua l'utente regolarmente nell'arco della settimana. Vengono specificati degli esempi per suggerire dei pattern di risposta all'utente.

4.2 GeneraSet

In questa sezione verrà descritta la logica e la struttura del programma Generaset.

4.2.1 Simulazione delle routine settimanali

A partire dai risultati ottenuti tramite i questionari somministrati, è stata costruita una rappresentazione verosimile della routine settimanale dell'utente, coprendo l'intervallo temporale dal lunedì alla domenica. Ogni minuto della settimana è stato associato a uno specifico stato dell'utente, in base all'attività che seguirebbe in quel momento.

Lo stato dell'utente al momento della formazione degli spostamenti verosimili può essere assegnato a:

- Walk (L'utente sta camminando)
- Automotive (L'utente sta guidando)
- Public (L'utente si trova sui mezzi publici)
- Home (L'utente si trova a casa)
- Act (L'utente sta svolgendo una attività)
- Work (L'utente si trova a lavoro)

Assegnazione degli spostamenti per attività regolari

Per l'ultima domanda è stato necessario un processo di raffinamento per strutturare i dati in modo adeguato, poiché si trattava di una domanda a risposta aperta, consentendo all'utente di esprimersi in maniera arbitraria. Per affrontare questo problema, si è inizialmente considerato l'impiego di modelli di Intelligenza Artificiale basati sul linguaggio naturale (LLM)[7]. Tuttavia, tale approccio è stato scartato poiché ritenuto eccessivo e non giustificato rispetto alla complessità del compito. In alternativa, si è optato per l'uso di espressioni regolari (regex), una soluzione più semplice e adeguata per l'estrazione e la strutturazione delle informazioni rilevanti.

Le regex sono state utilizzate come filtri per identificare pattern specifici all'interno delle risposte testuali, al fine di estrarre informazioni sui giorni e sugli orari degli spostamenti dell'utente. In particolare, sono state applicate a tre strutture lessicali principali:

1. Giorno specifico (figura 4.1): Ad esempio, "Domenica alle 19". In questo caso, l'utente indica un giorno della settimana specifico associato a un orario, generalmente accompagnato da una preposizione (come "dalle" o "alle").

2. Giorni generici: Ad esempio, "tre volte alla settimana alle 18 e 30". Qui l'utente fornisce un'indicazione generica sul numero di giorni. L'assegnazione dell'orario a un giorno specifico avviene in modo casuale, selezionando un giorno libero da altre attività, spostamenti o impegni lavorativi.

3. Tutti i giorni: Ad esempio, "tutte le mattine alle 10" oppure "ogni settimana dalle 19:20". In questo caso, l'attività viene assegnata all'orario indicato ogni giorno, garantendo che non vi siano conflitti con altre attività già pianificate.

```
(piu o meno |quasi tutti i |quasi tutte le |quasi ogni |quasi tutti i )*(lunedi|martedi|mercoledi|giovedi|venerdi|sabato|domenica)(alle|per le|a|dalle|le)*\s*(\d{1,2}(?:[:.]\d{0,2})?)\s*(circa)*
```

Figura 4.1. Regex per i giorni specifici

Le espressioni come "quasi", "più o meno" e "circa" che appaiono adiacenti all'indicazione dell'orario o dei giorni vengono catturate per introdurre variabilità e incertezza nella simulazione. In particolare, queste espressioni consentono di applicare una casualità all'orario, introducendo un intervallo variabile di minuti compreso tra -20 e +20. Inoltre, si considera una probabilità del 20% che l'attività pianificata non si svolga, rendendo la simulazione ancora più realistica e rappresentativa delle incertezze e delle variazioni tipiche nella pianificazione e nell'esecuzione delle attività quotidiane.

Per aumentare ulteriormente il livello di verosimiglianza, ogni orario viene modificato aggiungendo un intervallo casuale di minuti compreso tra -10 e +10. Questo aggiustamento aleatorio contribuisce a rendere l'orario risultante più realistico, simulando le variazioni naturali e imprevedibili che possono verificarsi nel mondo reale. In questo modo, si ottiene una rappresentazione temporale più accurata e aderente alle fluttuazioni quotidiane.

Inizialmente, vengono attribuiti gli orari di inizio delle attività. Successivamente, viene determinato un intervallo casuale per il ritorno, che varia tra mezz'ora e tre ore dopo l'inizio dell'attività. Durante l'intervallo compreso tra l'orario di inizio e quello di fine dell'attività, l'utente viene assegnato allo stato "act".
Successivamente, viene calcolato il tempo di percorrenza necessario per raggiungere l'attività e ritornare presso l'abitazione. Tale intervallo di tempo viene determinato mediante l'utilizzo di un valore randomico, la cui distribuzione segue una gaussiana caratterizzata da una media di 20 minuti e una deviazione standard pari a 5 minuti. Questo approccio statistico permette di simulare la variabilità delle condizioni reali di spostamento, tenendo in considerazione eventuali fattori che potrebbero influire sul tempo impiegato, come il traffico, le condizioni meteorologiche o eventuali ritardi. Poiché non sono disponibili dati statistici riguardanti i tempi di percorrenza degli automobilisti per raggiungere le attività extralavorative, tali valori sono stati determinati in maniera aleatoria, ritenendo che siano plausibili sulla base di considerazioni empiriche.

Assegnazione degli spostamenti lavorativi/accademici

Nel presente studio, sono state utilizzate domande del questionario relative agli orari lavorativi e al tempo di percorrenza per raggiungere il luogo di lavoro. La domanda riguardante la frequenza dell'utilizzo dell'automobile per recarsi al lavoro prevedeva cinque opzioni di risposta: sempre, spesso, qualche volta, raramente, mai. Tali risposte sono state tradotte in valori percentuali corrispondenti: 100%, 75%, 50%, 25%, 0%. Questi valori sono stati impiegati per determinare, nei periodi e nei giorni indicati dall'utente, la probabilità di assegnare uno spostamento in automobile o con i mezzi pubblici. Ad esempio, se l'utente ha dichiarato di utilizzare spesso l'auto per recarsi al lavoro, vi sarà una probabilità del 75% di assegnazione dello spostamento in automobile e una probabilità del 25% per l'uso dei mezzi pubblici per ogni giornata lavorativa.

Inoltre, qualora non venga assegnato all'utente uno spostamento in auto per il tragitto casa-lavoro, viene eseguita una verifica sulla distanza indicata. Se la distanza tra la residenza e il luogo di lavoro risulta inferiore a un chilometro, lo spostamento verrà automaticamente assegnato come tragitto pedonale anziché con i mezzi pubblici.
Il tempo di percorrenza dall'abitazione al luogo di lavoro viene modificato con un valore casuale compreso tra il -20% e il +20% della durata indicata, al fine di garantire una maggiore verosimiglianza, considerando la variabilità del traffico e altri fattori.

I tempi di partenza in auto vengono anch'essi variati tramite un valore casuale tra -5 e +15 minuti, includendo anche il tempo necessario per raggiungere l'automobile a piedi. Analogamente, per gli spostamenti con i mezzi pubblici, viene aggiunta una variazione casuale compresa tra 5 e 25 minuti, che tiene conto del tempo impiegato per raggiungere la fermata e per l'attesa del mezzo.

Risoluzione di conflitti e raffinamento

Dopo l'assegnazione degli intervalli di tempo per gli stati working, automotive e public, si procede alla risoluzione di eventuali conflitti tra gli spostamenti simulati. Per conflitti si intendono le sovrapposizioni tra gli intervalli assegnati all'utente, in particolare tra quelli relativi agli spostamenti (automotive/public) o agli stati (act/work). Non possono verificarsi sovrapposizioni tra due intervalli work, poiché ne viene assegnato al massimo uno per ciascun giorno.
Per identificare tali conflitti, vengono esaminati tutti gli intervalli dello stato act e verificata l'eventuale collisione con altri stati nello stesso giorno. Nel caso si rilevi una sovrapposizione, l'intervallo in questione viene riposizionato nel primo giorno disponibile, selezionato in modo casuale, che risulti libero in quella fascia oraria. Se non è possibile riprogrammare l'intervallo, quest'ultimo viene eliminato.

```python
def move_trip(trip, source_day, trips):
    """ Sposta il viaggio a un altro giorno senza conflitti """
    # prende un giorno random
    for day in random.sample(days, 7):
        if day != source_day:
            if not any(is_conflict(trip, other_trip) for other_trip in trips['automotive'][day]) and \
                    not any(is_conflict(trip, other_trip) for other_trip in trips['public'][day]) and \
                    not any(is_conflict(trip, other_trip) for other_trip in trips['walking'][day]):
                trips['automotive'][day].append(trip)  # se è possibile lo aggiunge a quel giorno
                return
    return


def resolve_conflicts(trips):
    """ Verifica conflitti tra act e work (automotive, public, walking) e in caso li sposta """
    for day in days:
        i = 0
        while i < len(trips['automotive'][day]):
            trip1 = trips['automotive'][day][i]
            if trip1[4] == 'act':
                states = ['automotive', 'public', 'walking']
                # Controlla act contro tutti i work
                for s in states:
                    if any(is_conflict(trip1, work_trip) for work_trip in trips['automotive'][day]):
                        move_trip(trip1, day, trips)
                        trips['automotive'][day].pop(i)
                        break
            i += 1

    return trips
```

Figura 4.2. Codice di risoluzione dei conflitti degli intervalli in GeneraSet

Dopo aver risolto i conflitti tra gli intervalli, vengono assegnati gli stati di walking che precedono l'ingresso nello stato automotive o public. Prima di entrare nello stato automotive, il periodo di walking viene determinato tramite un valore casuale compreso tra 2 e 10 minuti. Analogamente, il periodo di walking che precede lo stato public viene assegnato con un valore casuale compreso tra 10 e 20 minuti, includendo anche il tempo di attesa per il mezzo pubblico.
Può verificarsi che un'attività sia programmata in orario serale, con un intervallo che oltrepassi la mezzanotte, ad esempio con inizio alle 23:30 e fine alle 01:20, nello stato act. In tali casi, l'intervallo viene suddiviso in due parti: il primo segmento si estende fino alle 23:59, mentre il secondo inizia alle 00:00 e prosegue fino all'orario di fine indicato.

Infine, il tempo rimanente di ciascun giorno, nei periodi in cui l'utente non risulta assegnato a uno stato specifico, viene attribuito allo stato home. Questo implica che, nell'ambito della simulazione, l'utente viene collocato nella propria abitazione quando non è impegnato in attività lavorative, spostamenti in automobile, utilizzo di mezzi pubblici, camminate a piedi, o altre attività extralavorative. In particolare, lo stato home rappresenta il momento in cui l'utente si trova nella propria abitazione. Questa assegnazione viene eseguita automaticamente per evitare lacune temporali nel profilo giornaliero dell'utente, garantendo una continuità nella simulazione che riflette la realtà del quotidiano, in cui il tempo trascorso a casa gioca un ruolo significativo nella vita delle persone. Tale approccio consente di completare il quadro complessivo della giornata, fornendo una visione completa e accurata delle dinamiche giornaliere tra lavoro, spostamenti, attività e vita domestica.

4.2.2 Generazione dei dati sintetici dei sensori

La simulazione degli stati dell'utente nel tempo è generata sulla base delle risposte fornite attraverso il questionario. Il periodo di simulazione è definito da due variabili modificabili, che determinano la data e l'orario di inizio e di fine della simulazione. Nel caso in cui il periodo di simulazione superi i sette giorni, viene applicata una procedura per la generazione di una settimana "verosimile" per l'utente, in modo tale da variare orari e attività tra i diversi giorni, evitando la ripetizione meccanica e aumentando il realismo della simulazione.

Inoltre, è possibile specificare il valore della variabile *step*, che determina l'intervallo temporale tra i campionamenti dei valori dei sensori. Per esempio, impostando il periodo di simulazione dal 01-11-2024 alle ore 00:01 al 14-11-2024 alle ore 23:59, con uno *step* di 5 minuti, si genereranno 4032[2] campionamenti per ogni utente.

Analisi dei valori dei sensori

È stata condotta un'analisi preliminare per definire il funzionamento e la logica operativa dei sensori che verranno implementati nell'applicazione Generocity. Questi sensori forniranno un output in corrispondenza di specifici timestamp, restituendo un valore numerico compreso nell'intervallo [0,1]. Tale valore rappresenta una stima probabilistica dello stato dell'utente, in cui più il valore si avvicina ad 1 e maggiore è la certezza che l'utente si trovi all'interno della propria automobile.

Bluetooth

Si tratta di un sensore progettato per rilevare la connessione dell'utente a un dispositivo Bluetooth.
Il suo scopo principale è identificare se il dispositivo collegato allo smartphone dell'utente è un'automobile o un dispositivo generico, come ad esempio delle cuffie. Il riconoscimento del dispositivo come automobile avviene mediante l'analisi del tipo di connessione stabilita; in assenza di informazioni specifiche, viene esaminato il nome del dispositivo per determinare la sua natura.
Inoltre, il sensore conserva uno storico delle connessioni a dispositivi automobilistici, permettendo così di identificare in modo univoco il veicolo associato all'utente.

[2] *(60 minuti * 24 ore * 14 giorni) / 5 minuti di step*

Valore	Descrizione
1	Automobile tramite tipo della connessione
0.8	Automobile tramite il nome
0.5	Indecisione

Wi-Fi

Si tratta di un sensore in grado di rilevare la connessione dell'utente a una rete Wi-Fi o di analizzare le reti Wi-Fi disponibili per la connessione.
Il suo utilizzo è finalizzato a identificare se la rete a cui lo smartphone dell'utente è collegato sia una rete fissa, come quella domestica o aziendale, oppure un hotspot mobile. Questa distinzione consente di ottenere informazioni utili per dedurre il contesto di utilizzo della rete da parte dell'utente.
Permette di identificare una connessione al Wi-Fi dell'automobile analizzando il nome della rete a cui lo smartphone dell'utente è collegato. Questa analisi si basa sul riconoscimento di specifici pattern o denominazioni tipiche delle reti Wi-Fi veicolari, consentendo di distinguere tali connessioni da altre reti disponibili.

Valore	Descrizione
0.8	Collegato a un Wi-Fi di una macchina tramite il nome
0.5	Collegato a un hotspot o non collegato
0.3	Una rete disponibile riconosciuta è disponibile ma il dispositivo è connesso ai dati mobili
0	Collegato a una rete Wi-Fi fissa riconosciuto

Apple Car Play/Android Auto

Il sensore consente di rilevare se l'utente è connesso alla propria automobile tramite le tecnologie Apple CarPlay o Android Auto.
Analogamente al sensore Bluetooth descritto in precedenza, questo strumento mantiene uno storico delle connessioni avvenute con veicoli, al fine di identificare e associare con precisione l'auto di appartenenza dell'utente.

Valore	Descrizione
1	Collegato alla propria macchina
0.5	Collegato a una macchina non riconosciuta come propria
0	Non collegato

Motion Sensor

Il sensore Motion Sensor è un'aggregazione di diversi sensori forniti da Apple, che include GPS, giroscopio e accelerometro.
Questo sensore consente di interrogare direttamente lo stato dell'utente a un determinato timestamp, fornendo informazioni dettagliate sul movimento e la posizione dell'utente.

Valore	Descrizione
1	Automotive
$[0.5, 1]$	Automotive con bassa confidence
$[0, 0.5]$	Walking con bassa confidence
0	Walking

Cell Change

Il sensore analizza le connessioni alle reti cellulari (satellitari) e monitora i cambiamenti di connessione per stimare lo spostamento e la velocità dell'utente. Questa analisi si basa sulla variazione della rete a cui l'utente è connesso, permettendo di determinare la sua velocità di movimento.

Valore	Descrizione
1	L'utente sta cambiando cella telefonica molto spesso
0.5	L'utente non è connesso a una rete cellulare (non prende)
0	L'utente è connesso alla stessa rete cellulare da molto tempo

Charger

È un sensore che esamina la modalità di ricarica del dispositivo dell'utente per determinare il suo stato.
Si presume che il sensore sia in grado di identificare se la modalità di ricarica è rapida o standard; tuttavia, questa capacità non è stata implementata nella versione attuale del programma, lasciando spazio per potenziali aggiornamenti futuri.

Valore	Descrizione
1	Il dispositivo è in carica collegato ad un'auto
0.5	Il dispositivo non è in carica o è in carica non veloce
0	Il dispositivo è in carica tramite una ricarica veloce

GPS/Navigation

È un sensore che monitora la posizione dell'utente utilizzando il sistema GPS satellitare.
Rileva continuamente la posizione del dispositivo, consentendo di approssimare la velocità istantanea dell'utente in un dato timestamp.

Valore	Descrizione
1	Il dispositivo si sta spostando con una velocità moderata
0.5	Il dispositivo si sta spostando ad una bassa velocità o è fermo da poco tempo
0	Il dispositivo è da molto tempo fermo

BLE

Bluetooth Low Energy (BLE) è una tecnologia di comunicazione wireless progettata per offrire un consumo energetico ridotto rispetto al Bluetooth tradizionale, mantenendo al contempo una connessione affidabile e a bassa latenza. BLE opera su frequenze radio a 2.4 GHz e utilizza un protocollo di comunicazione che permette di stabilire e mantenere connessioni efficienti per dispositivi che richiedono bassa potenza e trasmissioni intermittenti.
Nel contesto del programma, BLE può essere successivamente integrata come un sensore per analizzare i dispositivi Bluetooth rilevati. Utilizzando la tecnologia BLE, è possibile determinare lo stato dell'utente in base ai dispositivi Bluetooth circostanti. Questo sensore fornirà dati essenziali per identificare e monitorare lo stato dell'utente, contribuendo a una valutazione più accurata delle sue attività quotidiane.

Attribuzione dei valori dei sensori relativamente allo stato dell'utente

In questa sezione verranno delineate le metodologie logiche implementate per l'assegnazione dei valori ai sensori in specifici timestamp, in relazione allo stato dell'utente in quel preciso momento.

Stato Home
Questo stato indica che l'utente si trova nella sua abitazione.

- Wi-Fi: Il valore è determinato dalla variabile *Wi-Fi_home*, che indica se l'utente utilizza una rete Wi-Fi domestica nella propria abitazione. Se la variabile *Wi-Fi_home* è positiva, il valore di default è impostato a 0, con un'ulteriore probabilità del 5% che, durante un campionamento specifico, l'utente non sia connesso alla propria rete Wi-Fi a causa di un passaggio alla rete mobile o di un allontanamento dal router. In tale eventualità, il valore sarà pari a 0.3, poiché la rete è disponibile ma il dispositivo non è attualmente collegato. Qualora l'utente non utilizzi una rete Wi-Fi domestica, il valore rimarrà costantemente 0.
- Bluetooth: 0.5
- Charger: 0.5
- Motion Sensor: Di default 0, in aggiunta è introdotta una probabilità del 5% che in un determinato campionamento il sensore restituisca 0.3 derivato da piccoli spostamenti all'interno dell'abitazione.
- GPS: 0
- Cambio cella: 0
- Carplay: 0

Stato Work

Questo stato indica che l'utente è coinvolto in una attività lavorativa o accademica.

- Wi-Fi: Il valore del sensore è influenzato dalla variabile *work_frequency*, che determina la probabilità di connessione alla rete Wi-Fi in un intervallo specifico. Se l'utente è connesso alla rete Wi-Fi durante l'intervallo, viene introdotta una probabilità del 20% per ogni campionamento che l'utente non utilizzi la rete. Questo approccio simula un uso intermittente della tecnologia, rendendo i risultati più realistici, e in tale caso il valore del sensore è pari a 0.3. Se, invece, l'utente è connesso alla rete Wi-Fi, il valore del sensore è impostato a 0.
- Bluetooth: 0.5
- Charger: 0.5
- Motion Sensor: Il sensore assume il valore 0, con una probabilità del 10% per ogni campionamento in cui il valore restituito è compreso tra 0.1 e 0.5.
  Questo intervallo di valori riflette piccole variazioni nel comportamento dell'utente all'interno di un contesto lavorativo o accademico, come movimenti minori o cambiamenti nella posizione che influenzano leggermente la rilevazione del sensore senza modificare sostanzialmente il suo stato complessivo.

- GPS: 0
- Cambio cella: 0
- Carplay: 0

Stato Act

Questo stato è associato all'intervallo durante il quale l'utente sta svolgendo un'attività regolare, come descritto e generato in base alle sue abitudini personali.

- Wi-Fi: Per semplicità, il valore è fissato costantemente a 0.5.
- Bluetooth: 0.5
- Charger: 0.5
- Motion Sensor: Il sensore assume il valore 0, con una probabilità del 10% per ogni campionamento in cui il valore restituito è compreso tra 0.1 e 0.5.
- GPS: Il sensore assume il valore 0, con una probabilità del 10% per ogni campionamento in cui il valore restituito è compreso tra 0.1 e 0.5.
- Cambio cella: 0
- Carplay: 0

Stato Walk

Questo stato indica che l'utente sta camminando a piedi per andare a lavoro, per andare ad una attività, per raggiungere la propria macchina parcheggiata o per raggiungere la fermata del mezzo pubblico.

- Wi-Fi: 0.5
- Bluetooth: 0.5
- Charger: 0.5
- Motion Sensor: Il sensore assume il valore 0, con una probabilità del 40% per ogni campionamento in cui il valore restituito è compreso tra 0.1 e 0.5.
- GPS: Il sensore assume il valore 0, con una probabilità del 10% per ogni campionamento in cui il valore restituito è compreso tra 0.1 e 0.5.
- Cambio cella: Il sensore assume il valore 0, con una probabilità del 5% per ogni campionamento in cui il valore restituito è 1.
- Carplay: 0

Stato Automotive

Questo stato indica che l'utente è nella propria automobile e sta guidando. (Figura 4.3)

- Wi-Fi: La tecnologia Wi-Fi in auto è strettamente correlata all'utilizzo di Apple Car Play o Android Auto. Questo sensore può rilevare la presenza dell'utente nella propria automobile solo se è connesso a tali tecnologie in modalità wireless. Di conseguenza, la frequenza di utilizzo del Wi-Fi dipende dalla probabilità (variabile *carplay_frequency*) che, in un determinato intervallo, l'utente faccia uso di Apple Car Play o Android Auto.
  Quando l'utente è collegato all'auto tramite queste tecnologie in modalità wireless o wireless con cavo, il valore del sensore è impostato a 0.8. Se, invece, la connessione Wi-Fi è configurata per il riconoscimento dell'auto ma non viene effettivamente utilizzata, il valore del sensore scende a 0.3 per tutta la durata dell'intervallo. Nel caso in cui la connessione avvenga esclusivamente tramite cavo o l'automobile non disponga di tali tecnologie, il valore del sensore è fissato a 0.5.

- Bluetooth: Nel questionario non è presente una domanda specifica che chiarisca se l'automobile dell'utente sia identificabile dal sensore Bluetooth attraverso il tipo di connessione o il nome del dispositivo, poiché si tratta di un aspetto troppo tecnico per essere incluso in questo contesto. Pertanto, prima della generazione dei dati sintetici, si imposta una variabile per ciascun utente che definisce se la connessione Bluetooth al veicolo viene riconosciuta tramite il nome o la tipologia di connessione. Si è scelto di assegnare una probabilità del 90% al riconoscimento tramite il nome del dispositivo, considerato che è più comune che i dispositivi Bluetooth dell'autoradio abbiano nomi identificabili. Solo il 10% dei casi prevede un riconoscimento tramite la tipologia di connessione, un'opzione ritenuta meno frequente ma possibile.
  La frequenza di connessione dell'utente alla propria automobile tramite Bluetooth è determinata dalla variabile *Bluetooth_frequency*, la quale stabilisce la probabilità di utilizzo della tecnologia Bluetooth durante un determinato intervallo. Se il veicolo è identificabile tramite la tipologia di connessione Bluetooth, il sensore restituisce un valore pari a 1. In alternativa, se l'automobile viene riconosciuta tramite il nome del dispositivo Bluetooth, il valore del sensore sarà pari a 0.8. Tale valore rimane costante per l'intera durata dell'intervallo in cui l'utente utilizza il Bluetooth in auto.
  Durante i periodi in cui il Bluetooth non viene utilizzato, il valore del sensore si riduce a 0.5.

- Charger: La probabilità che l'utente utilizzi la ricarica del telefono mentre si trova in automobile è determinata dalla variabile *charging_frequency*. All'interno del questionario vengono poste domande riguardanti la modalità di ricarica del dispositivo all'interno del veicolo. Se l'utente ha indicato che il telefono viene ricaricato direttamente tramite l'auto, il sensore assegnerà un valore di 1 durante l'intervallo di ricarica, poiché sarà in grado di rilevare la connessione con il veicolo. In caso contrario, se l'utente ha dato una risposta differente, il valore del sensore sarà fissato a 0.5. Inoltre, se l'utente ha segnalato l'uso della tecnologia Apple CarPlay o Android Auto per la connessione al veicolo, sia tramite cavo che tramite una combinazione di cavo e wireless, il sensore restituirà un valore di 1 ogni volta che viene avviene la connessione. Se il sensore non può riconoscere il collegamento al veicolo, il suo valore rimarrà a 0.5

- Motion Sensor: Identificando che l'utente si sta spostando ad una certa velocità il sensore restituisce il valore 1. Tuttavia per ogni singolo campionamento sono state introdotte delle variabili che modellano il comportamento del sensore

rendendolo più realistico. E' stato introdotta la possibilità che si verifichi un rumore con probabilità di 5% dovuto magari a rallentamenti o stop nella normale viabilità nel traffico, in questa circostanza il valore restituisce un valore compreso tra 0.1 e 0.5 per quel campionamento. Inoltre è stata introdotta anche una logica per la stabilizzazione del valore del sensore, ossia che all'inizio dell'intervallo in macchina il valore è più probabile che sia tra 0.5 e 0.9 rispetto ai minuti a venire, questo dato appunto dalla stabilizzazione del sensore. Nello specifico la probabilità che restituisca un valore non 1 è data dal 30% sottratto dell'un terzo dei minuti trascorsi nell'intervallo nel veicolo. Ad esempio dopo 15 minuti trascorsi in automobile la probabilità che il valore del sensore sia compreso tra 0.5 e 0.9 sarà 25[3].

- GPS: Il sensore assume il valore 1, con una probabilità del 30% per ogni campionamento in cui il valore restituito è compreso tra 0.5 e 0.9.
- Cambio cella: 1
- Carplay: Il valore dipende dalla varibile *carplay_frequency* che detta la probabilità di utilizzo della tecnologia Apple Car Play/Android Auto in auto. Il sensore restituisce il valore 1 durante l'intervallo in cui l'utente utilizza questa tecnologia, altrimenti 0.

Stato Public

Questo stato indica che l'utente si trova su un mezzo pubblico.

- Wi-Fi: 0.5
- Bluetooth: 0.5
- Charger: 0.5
- Motion Sensor: Il sensore assume il valore 1, con una probabilità del 30% per ogni campionamento in cui il valore restituito è compreso tra 0.5 e 0.9. Non è stato implementato un meccanismo di stabilizzazione del valore del sensore, come accade nello stato "automotive", poiché in questo contesto i rallentamenti del mezzo pubblico sono molto più frequenti. Questi rallentamenti derivano dalle numerose fermate previste lungo il tragitto, rendendo meno coerente l'assegnazione di un valore stabile al sensore. Di conseguenza, il valore può subire variazioni più frequenti, riflettendo una situazione di spostamento meno fluida rispetto a quella che si verifica con l'uso di un'automobile privata.
- GPS: Il sensore assume il valore 1, con una probabilità del 30% per ogni campionamento in cui il valore restituito è compreso tra 0.5 e 0.9.
- Cambio cella: Il sensore assume il valore 1, con una probabilità del 5% per ogni campionamento in cui il valore restituito è 0.
- Carplay: 0

[3] *30 - 15/3 = 25*

```
elif state == 'automotive':
    if old_state != state:
        minutes_in_automotive = 0
        carplay_value = randomize_sensor_value(0, 1, user['carplay_frequency'])
        if auto_bluetooth_by_name:
            bluetooth_value = randomize_sensor_value(0.5, 0.8,
            ↪ user['bluetooth_frequency'])
        else:
            bluetooth_value = randomize_sensor_value(0.5, 1,
            ↪ user['bluetooth_frequency'])
        if user['charging_place'] == 'Direttamente all\'auto':
            charger_value = randomize_sensor_value(0.5, 1,
            ↪ user['charging_frequency'])
        else:
            charger_value = 0.5
    else:
        minutes_in_automotive += step
        bluetooth_value = data_sensor[-1][8]
        charger_value = data_sensor[-1][9]
        carplay_value = data_sensor[-1][13]

    if carplay_value:
        if user['carplay_connection'] == 'Wireless' or user['carplay_connection']
        ↪ == 'Entrambi':
            data_timestamp.append(0.8)  # Wi-Fi
        else:
            data_timestamp.append(0.5)  # Wi-Fi

        if user['carplay_connection'] == 'Cavo' or user['carplay_connection'] ==
        ↪ 'Entrambi':
            charger_value = 1  # charger
    else:
        data_timestamp.append(0.5)  # Wi-Fi

    data_timestamp.append(bluetooth_value)  # Bluetooth
    data_timestamp.append(charger_value)  # charger
    if random.random() > 0.05:
        data_timestamp.append(randomize_sensor_value(0, random.randint(1, 5) / 10,
        ↪ 1))  # Motion Sensor
    else:
        prob = 0.3 - minutes_in_automotive/30
        if prob < 0:
            prob = 0
        data_timestamp.append(randomize_sensor_value(1, random.randint(5, 10) / 10,
        ↪ prob))  # Motion Sensor

    if no_gps:
        data_timestamp.append(0.5)  # GPS
    else:
        data_timestamp.append(randomize_sensor_value(1, random.randint(5, 10) / 10,
        ↪ 0.3))  # GPS

    if no_signal:
        data_timestamp.append(0.5)  # CellChange
    else:
        data_timestamp.append(1)  # CellChange

    data_timestamp.append(carplay_value)  # carplay
    data_timestamp.append(1)  # stato automotive
```

Figura 4.3. Sezione di codice per l'assegnamento dei valori dei sensori per lo stato automotive

Per ogni campionamento di qualsiasi stato, una volta che i valori dei vari sensori sono stati assegnati, viene introdotto un ulteriore valore binario (1 o 0) che rappresenta se l'utente si trova all'interno della propria automobile o meno. Il valore 1 viene associato esclusivamente ai campionamenti che rientrano nell'intervallo temporale definito come stato "automotive", mentre in tutti gli altri stati il valore sarà 0. Questo dato aggiuntivo viene denominato "valore di vera verità" (ground truth) e rappresenta l'etichetta che indica lo stato effettivo dell'utente.
L'importanza di questo valore risiede nel suo successivo utilizzo nell'addestramento di modelli predittivi. Questi modelli saranno in grado di riconoscere schemi nei valori restituiti dai sensori e associare tali schemi all'etichetta corrispondente.
(Ultima riga di codice nella figura 4.3)

Per aumentare il realismo dei dati sintetici generati, è stata introdotta una logica di malfunzionamento casuale dei sensori GPS e CellChange per ogni campionamento. Questa logica prevede una probabilità che, per un intervallo di tempo casuale, il sensore restituisca un valore costante di 0.5, simulando una perdita temporanea di segnale. Come mostrato nelle righe 40-48 della figura 4.3, nel caso del sensore GPS, c'è una probabilità del 5% che restituisca il valore 0.5 per un periodo variabile tra i 5 e i 10 minuti. Analogamente, per il sensore CellChange, vi è una probabilità del 5% di restituire 0.5 per un periodo compreso tra i 5 e i 20 minuti. Nel momento in cui questo malfunzionamento viene attivato, tutti i campionamenti successivi restituiscono il valore 0.5 fino al termine del periodo impostato. Tale periodo viene decrementato a ogni campionamento secondo l'intervallo definito dalla variabile step, garantendo un malfunzionamento simulato più realistico. Questa introduzione migliora l'affidabilità della simulazione, tenendo conto di possibili interruzioni del segnale tipiche dei contesti reali, ad esempio in aree con scarsa copertura di rete o satellitare.

```
1  if not no_signal: # variabile inizializzata a 0
2      no_signal = randomize_sensor_value(0, 1, 0.05)  # 5% di probabilità che il
       ↪  telefono non prenda per [5, 20] minuti
3      no_signal_minutes = random.randint(5, 20)
4  else:
5      no_signal_minutes -= step # a ogni campionamento il numero di minuti viene
       ↪  decrementato di step minuti
6      if no_signal_minutes <= 0:
7        no_signal = 0
8
9  if not no_gps: # variabile inizializzata a 0
10     no_gps = randomize_sensor_value(0, 1, 0.05)  # 5% di probabilità che il gps
       ↪  non funzioni per [5, 10] minuti
11     no_gps_minutes = random.randint(5, 10)
12 else:
13     no_gps_minutes -= step # a ogni campionamento il numero di minuti viene
       ↪  decrementato di step minuti
14     if no_gps_minutes <= 0:
15       no_gps = 0
```

Figura 4.4. Sezione di codice per l'aggiunta di rumore nel funzionamento dei sensori GPS e CellChange

I campionamenti dei valori dei sensori per ciascun utente, riferiti a specifici timestamp, vengono salvati in un file CSV di output. Ogni riga di questo file rappresenta un singolo campionamento, organizzato in modo che ogni colonna contenga il valore corrispondente a un determinato sensore. Oltre ai valori dei sensori, nella riga ven-

gono riportate informazioni aggiuntive, come il timestamp (suddiviso in anno, mese, giorno, ora, minuto, e giorno della settimana), il codice identificativo dell'utente e lo stato dell'utente stesso (1 per automotive e 0 per non-automotive).

Test effettuati

Per testare il programma, il questionario è stato somministrato a un campione di 53 utenti, variabili per sesso, età e abitudini, al fine di ottenere un quadro rappresentativo della popolazione. I tempi di esecuzione del programma sono stati monitorati, includendo fasi quali l'apertura e la lettura del file CSV contenente le risposte, la generazione delle settimane verosimili degli utenti e la successiva produzione dei dati sintetici dei sensori.

Viene inoltre sottolineata l'importanza dei parametri di generazione, come lo step e il periodo di simulazione, poiché essi influiscono significativamente sulla quantità di dati generati. Questi parametri determinano la granularità del campionamento e la durata della simulazione, incidendo sui tempi complessivi di elaborazione.

I test sono stati condotti utilizzando la piattaforma Google Colab, in modalità di runtime CPU.
Per ogni serie di parametri, il programma è stato eseguito dieci volte consecutivamente, e la media dei tempi di esecuzione risultanti è stata calcolata e riportata. Questo approccio consente di ottenere una stima più stabile e accurata dei tempi di esecuzione, riducendo l'impatto di eventuali variazioni temporanee o casuali legate alle risorse computazionali disponibili al momento della prova.

N Utenti	step(m)	Periodo(gg)	Tempo(s)
53	2	14	8.4
53	2	28	15.3
53	5	14	2.9
53	5	28	6.2
10	2	14	1.6
10	2	28	3.7
10	5	14	0.6
10	5	28	1.8

Capitolo 5

Sviluppo di modelli di Machine Learning

In questo capitolo saranno illustrate le metodologie adottate per la costruzione di modelli predittivi basati sui dati sintetici generati dal programma GeneraSet. L'obiettivo principale è quello di sviluppare modelli capaci di predire lo stato dell'utente, ovvero se esso si trova all'interno della propria automobile o meno, a partire dai valori dei sensori simulati.
Si farà riferimento al dataset generato utilizzando come parametri step pari a 2, un periodo di simulazione di due settimane e 53 utenti.

5.1 Pre-processamento dei dati

Il codice implementa una serie di operazioni volte a pre-elaborare i dati raccolti dai sensori e generati dal programma GeneraSet. Data la natura dei dati, la variabile target è rappresentata da due classi: automotive (1) e non-automotive (0). Tuttavia, il dataset risulta significativamente sbilanciato, con una predominanza di campioni appartenenti alla classe non-automotive (walking) dato dal fatto che nelle abitudini giornaliere di un individuo, il tempo trascorso all'interno della proria automobile è irrisorio rispetto all'intero periodo giornaliero.
Per migliorare le prestazioni del modello predittivo è necessario correggere tale sbilanciamento, utilizzando tecniche di bilanciamento delle classi[8].
In particolare, vengono considerate tre strategie:

- Nessun bilanciamento ($balance\_class = 0$)
- Upsampling della classe minoritaria ($balance\_class = 1$)
- Downsampling della classe maggioritaria ($balance\_class = 2$)

Nessun bilanciamento

In questa prima modalità, i dati non vengono bilanciati, mantenendo lo sbilanciamento originale (9280 campioni per automotive, 514880 per walking). In un contesto di Machine Learning, l'addestramento di un modello con un dataset così sbilanciato potrebbe portare il modello a prediligere la classe maggioritaria (walking). Questo si tradurrebbe in un alto accuracy score per la classe dominante, ma bassa performance (precision, recall, F1-score) per la classe minoritaria (automotive). Un esempio di errore in questo caso potrebbe essere un classificatore cieco, che

prevede sempre "walking" senza mai riconoscere correttamente la classe "automotive" ottenendo comunque una buona accuracy.

Upsampling

In questo approccio, i record appartenenti alla classe minoritaria (automotive) vengono ripetuti fino a raggiungere una dimensione comparabile a quella della classe maggioritaria. Il risultato finale presenta 505600 campioni per automotive e 514880 per walking. L'upsampling bilancia le due classi, riducendo il rischio che il modello ignori la classe minoritaria.
Tuttavia, questa metodologia introduce campioni duplicati, il che può portare a overfitting. In altre parole, il modello potrebbe imparare a riconoscere pattern ripetitivi legati ai dati duplicati piuttosto che generalizzare correttamente.

Downsampling

In questo approccio, il numero di campioni appartenenti alla classe maggioritaria ("walking") è stato ridotto al fine di riequilibrare la distribuzione delle classi. Tale riduzione è stata ottenuta fissando un limite massimo di 500 record per ciascun insieme distinto di valori dei sensori relativi alla classe "walking".
Dunque, il dataset risultante comprende 23.648 record per la classe "walking" e i 9.280 campioni originali per la classe "automotive". Questo metodo ha permesso di mantenere l'integrità dei dati originali relativi allo stato "walking", limitandosi a ridurre il numero di record con configurazioni ridondanti di valori.

Dopo il bilanciamento del dataset, sono state eseguite diverse operazioni, tra cui:

- Separazione del dataset in feature e label: le feature includono tutte le informazioni relative al timestamp e ai valori dei sensori, mentre la label rappresenta lo stato del campionamento (automotive o walking), ovvero la variabile che i modelli dovranno predire a partire dalle feature.
- Estrazione dei dati appartenenti a un singolo utente, utilizzati esclusivamente nella fase di test, per evitare che tali dati vengano esposti durante la fase di addestramento.
- Rimozione delle informazioni relative all'ID utente (userID) dal dataset per prevenire l'introduzione di bias nel modello.
- Suddivisione del dataset in training set e test set, con una proporzione del 70% per il training set e del 30% per il test set.
  La separazione è stata eseguita in modo casuale, utilizzando un *seed* predefinito (valore 42) per garantire la riproducibilità della suddivisione randomica.

5.2 Creazione del modello

In questa sezione verranno dettagliatamente illustrati i modelli predittivi implementati al fine di effettuare la previsione delle label a partire dalle feature estratte dai dati. L'obiettivo principale di tali modelli è quello di apprendere relazioni significative tra le feature e le label corrispondenti. Ogni modello impiegato sarà descritto in termini di architettura, iperparametri scelti e metodologia di addestramento,

con particolare attenzione alle tecniche di valutazione e alle metriche utilizzate per misurare le prestazioni predittive.

5.2.1 Random Forest

Il Random Forest[9] è un algoritmo di Machine Learning basato su un insieme di alberi decisionali, ciascuno dei quali è addestrato su diversi sottoinsiemi del dataset mediante la tecnica del *bootstrap sampling*[1]. Ogni albero prende decisioni in base ai dati su cui è stato addestrato, producendo una predizione autonoma.

In ogni nodo di ciascun albero, viene selezionato casualmente un sottoinsieme di variabili (*features*) da considerare per determinare la suddivisione (*split*) migliore dei dati. Questa selezione casuale ha lo scopo di ridurre la correlazione tra gli alberi, incrementando così la robustezza complessiva dell'algoritmo.

Una volta che gli alberi sono stati addestrati, i risultati ottenuti da ciascun albero vengono combinati. Nelle operazioni di classificazione, ogni albero fornisce un voto per la classe predetta, e la classe finale viene determinata tramite un meccanismo di maggioranza (*voting*). Nel caso della regressione, le predizioni dei vari alberi vengono mediate per ottenere il risultato finale.

Un aspetto cruciale del Random Forest è la capacità di misurare l'importanza delle variabili predittive. Questa capacità lo rende uno strumento molto interpretabile rispetto ad altri algoritmi di Machine Learning avanzati. La misura dell'importanza delle variabili si basa sull'impatto che l'eliminazione o la perturbazione di una variabile specifica ha sull'accuratezza del modello. Una riduzione significativa della precisione del modello suggerisce che la variabile in questione ha un ruolo rilevante nel determinare le predizioni finali. Questo metodo fornisce una chiara indicazione delle variabili più influenti nel dataset, facilitando l'interpretazione del modello.

In aggiunta, ciascun albero decisionale è interpretabile di per sé, in quanto segue una logica di regole "se-allora" per effettuare le predizioni. Sebbene l'insieme di alberi presenti nella foresta possa risultare numeroso, rendendo complessa l'interpretazione di ciascun albero singolarmente, è comunque possibile isolare gli alberi più significativi o analizzare l'influenza di variabili specifiche tramite strumenti come i grafici di *partial dependence*. Questi grafici consentono di visualizzare l'effetto di una singola variabile sulle predizioni, mantenendo costanti le altre, migliorando la comprensione delle relazioni non lineari e delle interazioni tra le variabili.

Per la costruzione del modello nel presente progetto, sono stati configurati tre parametri fondamentali che caratterizzano l'algoritmo Random Forest:

- `n_estimators`: Rappresenta il numero di alberi decisionali che compongono la foresta. Un valore maggiore di `n_estimators` tende a migliorare la robustezza del modello, riducendo la varianza, ma aumenta il costo computazionale. E' stato assegnato il valore 5 per questo parametro.
- `max_depth`: Definisce la profondità massima degli alberi. Limitare la profondità può prevenire l'overfitting, poiché riduce la complessità del modello, ma un valore troppo basso può portare a underfitting. E' stato assegnato il valore 7 per questo parametro.

[1] Campionamento con sostituzione

- `min_samples_split`: Indica il numero minimo di campioni richiesto per suddividere un nodo. Questo parametro controlla la crescita degli alberi, influenzando la granularità delle suddivisioni; valori più alti possono semplificare il modello, prevenendo una suddivisione eccessiva. E' stato assegnato il valore 2 per questo parametro.

Per ottimizzare gli iperparametri del modello, è stata utilizzata una griglia di ricerca (*Grid Search*) avente come obiettivo la massimizzazione dell'`average_precision`[2] sul dataset di train. La griglia contiene una serie di valori predefiniti per ciascun iperparametro, e il processo consiste nel testare tutte le combinazioni possibili di questi valori. L'algoritmo seleziona quindi la combinazione che ottimizza la metrica obiettivo, identificando i valori degli iperparametri che massimizzano le prestazioni del modello.

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [3, 5, 10, 15, 20, 30, 50, 60, 80, 100, 200],
    'max_depth': [3, 5,  10],
    'min_samples_split': [2, 5, 10],
}

grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5,
↪  scoring='average_precision', verbose=3)
grid_search.fit(X_train, y_train)

print("Best Parameters:", grid_search.best_params_)
```

Figura 5.1. Griglia di ricerca per il Random Forest

Dopo la creazione del modello Random Forest, quest'ultimo viene addestrato utilizzando le *features* e le *labels* della porzione di *train* del dataset. Successivamente, il modello viene testato prevedendo le etichette per i dati della porzione di *test* del dataset, utilizzando le sole *features* di quest'ultimo.
In fase di testing, le performance del modello vengono quindi valutate attraverso diverse metriche. In primo luogo, si calcola l'accuratezza, che rappresenta la percentuale di predizioni corrette rispetto al totale dei campioni nel set di test. Questa metrica fornisce un'indicazione generale dell'affidabilità del modello.
Successivamente, viene calcolata la *average_precision*, che tiene conto sia della precisione che del richiamo, fornendo una visione più equilibrata delle performance del modello, soprattutto in presenza di classi sbilanciate. Inoltre, viene generato un report di classificazione dettagliato, il quale include metriche come la precisione, il richiamo e il punteggio F1 per ciascuna delle classi considerate, in questo caso "Walking" e "Automotive".

[2]La metrica `average_precision` misura la precisione media ponderata su tutte le soglie di classificazione.

```
Accuracy: 0.986
Average precision: 0.962
Classification Report:
                precision    recall  f1-score   support

     Walking       0.99      0.99      0.99      7078
  Automotive       0.98      0.97      0.98      2801

    accuracy                           0.99      9879
   macro avg       0.99      0.98      0.98      9879
weighted avg       0.99      0.99      0.99      9879
```

Figura 5.2. Classification Report di Random Forest utilizzando il dataset con downsampling

La valutazione prosegue con la creazione di una matrice di confusione, che consente di visualizzare i risultati delle predizioni rispetto alle etichette vere. Questa matrice offre un'analisi approfondita delle classificazioni corrette e degli errori del modello, facilitando l'identificazione di aree di miglioramento.

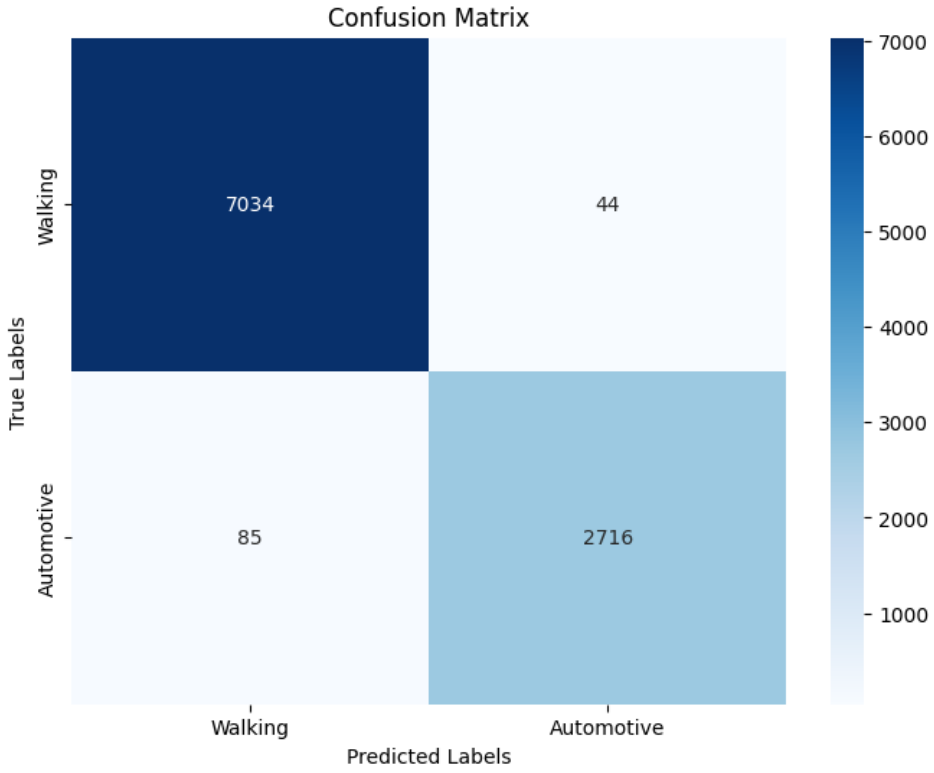


Figura 5.3. Confusion Matrix di Random Forest in fase di test

Dopo aver utilizzato il set di test del dataset per valutare il modello, può essere impiegato l'insieme di dati dell'utente, che inizialmente è stato escluso dal dataset al fine di evitare che il modello venga esposto a questi dati durante la fase di addestramento. Questa tecnica di testing è chiamata *out-of-sample* testing. Essa permette di valutare le performance del modello su dati completamente nuovi,

garantendo una misura più affidabile della sua capacità di generalizzare su dati mai visti prima.

```
Accuracy: 0.999
Average precision: 0.894
Classification Report:
                precision    recall  f1-score   support

     Walking       1.00      1.00      1.00      9995
  Automotive       0.89      1.00      0.94        85

    accuracy                           1.00     10080
   macro avg       0.95      1.00      0.97     10080
weighted avg       1.00      1.00      1.00     10080
```

Figura 5.4. Classification Report di Random Forest utilizzando *out-of-sample*

È possibile esaminare l'importanza delle variabili (*Feature Importance*), che indica quali feature contribuiscono maggiormente alla riduzione dell'impurità nella costruzione degli alberi decisionali. L'impurità, in questo caso, è definita in base al criterio di suddivisione adottato dall'algoritmo; nello specifico, viene utilizzato l'indice di Gini, che misura il grado di disordine in un nodo e diminuisce ad ogni suddivisione, migliorando così l'accuratezza complessiva del modello.

I due grafici in figura 5.5 mostrano l'impatto delle feature sulla riduzione media dell'accuratezza (primo grafico) e dell'*average precision* (secondo grafico), in seguito all'esclusione di ciascuna di esse dal modello. Dal secondo grafico si nota chiaramente come l'esclusione della feature Bluetooth comporti un calo significativo delle prestazioni, indicando il suo ruolo essenziale nella fase predittiva.
Tuttavia, osservando il primo grafico relativo all'accuratezza, si rileva che le feature influenzano il modello in misura relativamente ridotta, con variazioni inferiori al 3%. Una possibile spiegazione potrebbe risiedere nell'alta correlazione tra le feature: escludendone una tramite permutazione, il modello potrebbe utilizzare un'altra variabile fortemente correlata, mantenendo così un livello di accuratezza pressoché invariato.

Inoltre, l'interpretabilità offerta dal modello Random Forest consente di identificare le relazioni tra le feature, anche nel caso di correlazioni lineari. Come mostrato nella figura 5.6, le coppie di sensori maggiormente correlate sono Apple Car Play e Charger, e Gps e CellChange. La correlazione tra Apple Car Play e Charger è plausibile, poiché la connessione con Apple Car Play avviene tramite cavo, il che attiva anche il sensore di carica. La correlazione tra Gps e CellChange, invece, potrebbe essere spiegata da intersezioni temporali nei periodi di assenza di segnale; inoltre, entrambi i sensori tendono a presentare valori di confidenza simili quando l'automobile è in movimento, suggerendo che condividano condizioni operative simili.

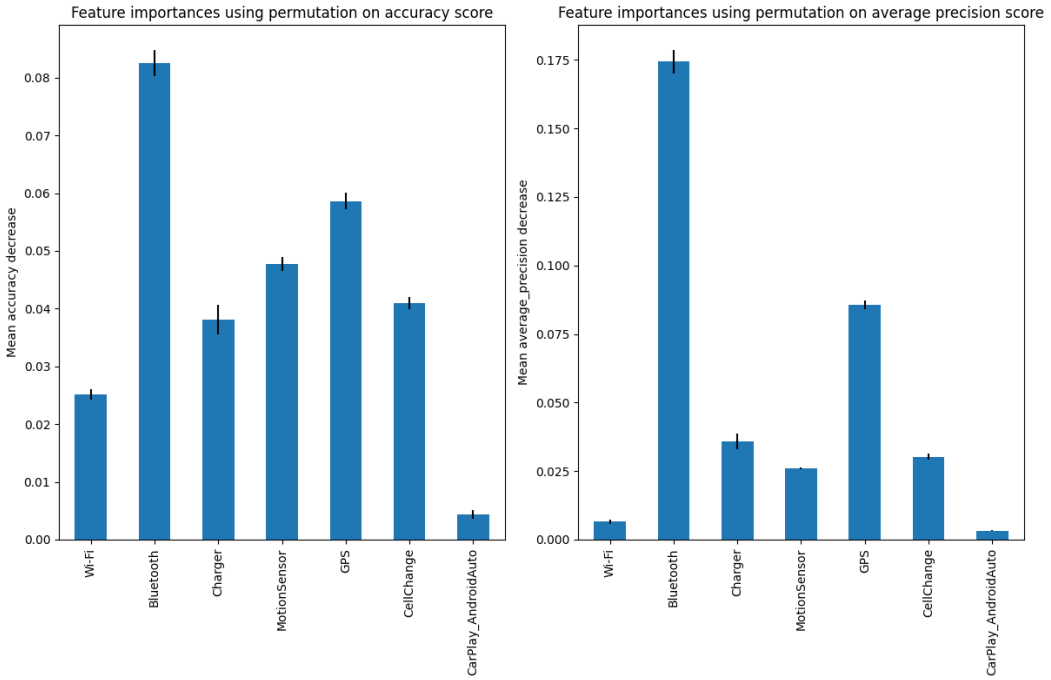


Figura 5.5. Feature Importance di Random Forest

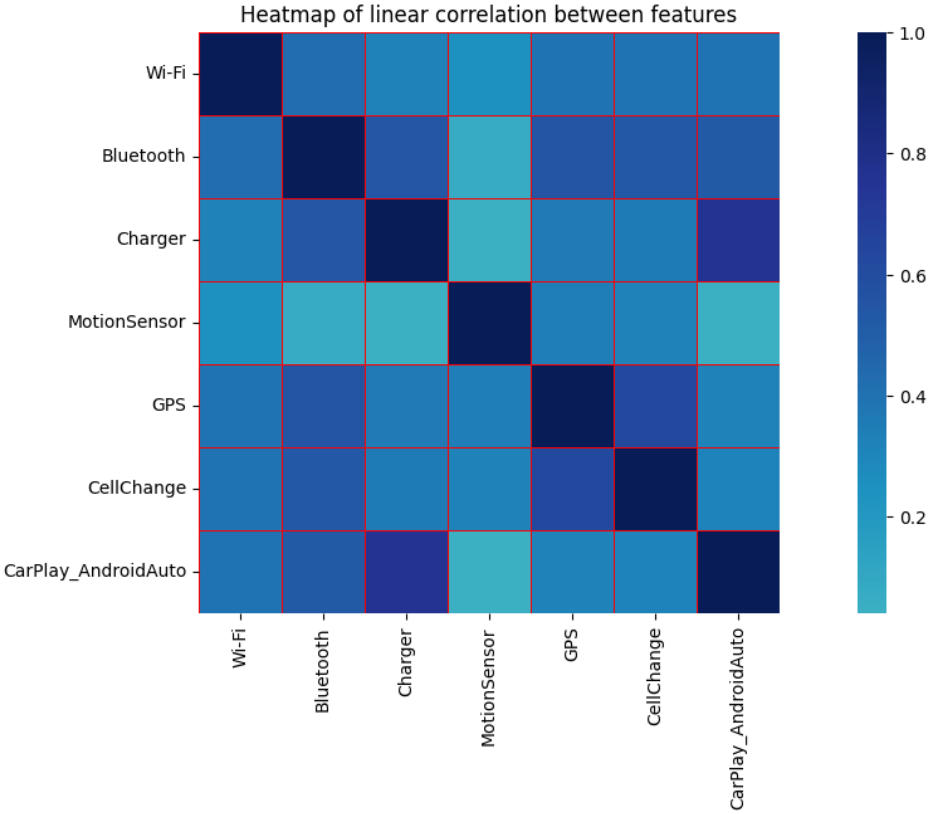


Figura 5.6. Correlazioni lineari tra le feature

5.2.2 Decision Tree

Il Decision Tree[10] è un algoritmo di Machine Learning che organizza il processo decisionale in una struttura ad albero, dove ogni nodo interno rappresenta una condizione su una variabile (*feature*), i rami rappresentano i possibili esiti della condizione, e le foglie rappresentano la classe o il valore predetto.

In fase di addestramento, il modello seleziona la variabile che massimizza la separazione tra le classi, usando l'indice di Gini per i problemi di classificazione. L'albero cresce ricorsivamente, suddividendo i nodi in base alla feature più informativa fino a soddisfare i criteri di arresto, come la profondità massima (`max_depth`) o numero massimo di foglie(`max_leaf_nodes`). Sebbene altamente interpretabili, i Decision Trees sono noti per il rischio di sovradattamento ai dati di addestramento. Questo problema può essere mitigato regolando i parametri di costruzione dell'albero potandolo dopo la costruzione o limitandolo prima.

Per questo progetto è stato implementato un Decision Tree con una profondità massima pari a 3. Come illustrato nella figura 5.7, il modello identifica il valore del sensore Bluetooth come la feature più importante per la classificazione dei record. Successivamente, i sensori GPS, Motion Sensor, e CellChange contribuiscono a raffinare ulteriormente il processo decisionale, migliorando la precisione delle predizioni.

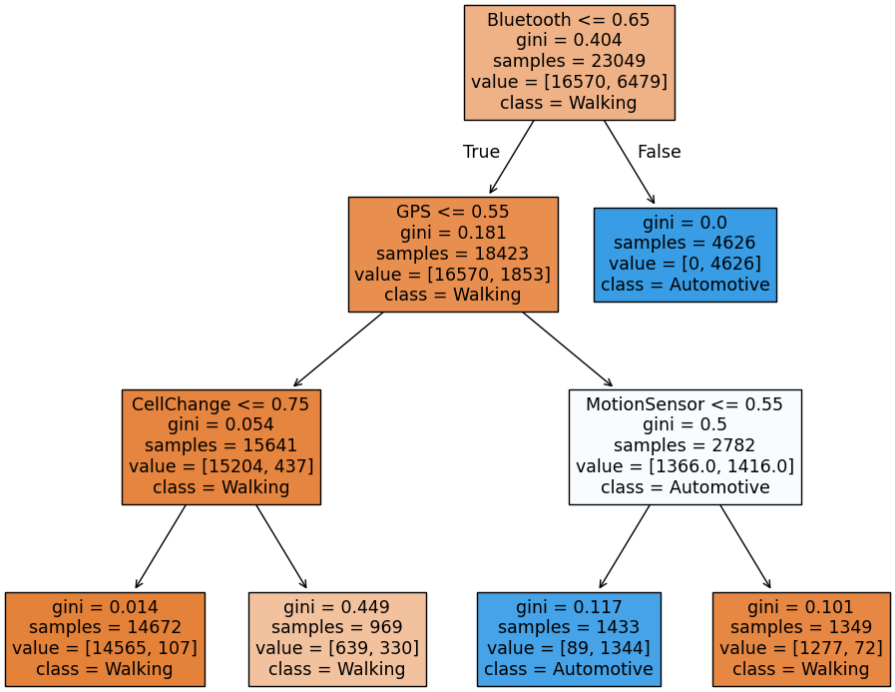


Figura 5.7. Albero decisionale

L'albero è stato costruito utilizzando i dati del set di train, e successivamente impiegato per predire la classe dei record nel set di test. Di seguito (figura 5.8)

vengono riportati i risultati delle metriche valutative del modello, insieme alle feature importance restituite dall'algoritmo. Le metriche forniscono una misura delle performance del modello su dati non visti, mentre le feature importance indicano il contributo di ciascuna variabile nella determinazione delle predizioni finali.

```
Average precision score:  0.933
Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.98      7078
           1       0.99      0.92      0.95      2801

    accuracy                           0.97      9879
   macro avg       0.98      0.96      0.97      9879
weighted avg       0.98      0.97      0.97      9879

Feature Importances:
Bluetooth  ->  0.715
MotionSensor  ->  0.129
GPS  ->  0.130
CellChange  ->  0.024
```

Figura 5.8. Valutazione del Decision Tree e Feature Importances

5.2.3 Neural Networks

Una Neural Network[11] (Rete Neurale) è un modello di apprendimento automatico ispirato alla struttura del cervello umano, progettato per apprendere pattern complessi dai dati attraverso una serie di trasformazioni non lineari. Nel contesto di questo progetto, è stata implementata una rete neurale a strati densi per risolvere il problema di classificazione binaria, che distingue tra due classi: "Walking" e "Automotive". Essa è composta da cinque strati completamente connessi (*fully connected layers*), ciascuno seguito da una funzione di attivazione non lineare `ReLU` (Rectified Linear Unit) tranne per l'ultimo, dove viene impiegata una funzione `Softmax` per la normalizzazione delle probabilità delle classi.

Per l'addestramento del modello, è stata utilizzata la `CrossEntropyLoss`, una funzione di perdita comunemente impiegata per problemi di classificazione multi-classe. L'ottimizzazione dei pesi del modello avviene tramite l'algoritmo `Adam`, una versione migliorata della discesa del gradiente che effettua aggiornamenti dei parametri più efficienti, utilizzando un tasso di apprendimento di 0.001. Durante l'addestramento, la rete neurale esegue 40 epoche, aggiornando i pesi attraverso il *backpropagation*. Al termine dell'addestramento, il modello è in grado di generalizzare le informazioni apprese e viene quindi testato su un set di dati separato per valutarne le prestazioni. Durante questa fase, le metriche di performance, come l'accuratezza e il report di classificazione, vengono calcolate per fornire una misura oggettiva dell'efficacia del modello nel compito di classificazione (figura 5.10).
Per valutare il modello è possibile analizzare il grafico della *loss* che rappresenta il valore della funzione di perdita ad ogni epoca,Un valore decrescente della *loss* suggerisce un miglioramento continuo nell'apprendimento del modello, indicando che esso sta ottimizzando i suoi parametri per fornire predizioni più accurate.

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.fc1 = nn.Linear(X_train.shape[1], 1024)
        self.fc2 = nn.Linear(1024, 512)
        self.fc3 = nn.Linear(512, 256)
        self.fc4 = nn.Linear(256, 64)
        self.fc5 = nn.Linear(64, 2)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.relu(self.fc3(x))
        x = self.relu(self.fc4(x))
        x = self.softmax(self.fc5(x))

        return x
model = NeuralNetwork().to(device)
```

Figura 5.9. Struttura del modello dell'Artificial Neural Network

```
Accuracy: 0.978
              precision    recall  f1-score   support

     Walking       0.98      0.99      0.98      7078
  Automotive       0.96      0.96      0.96      2801

    accuracy                           0.98      9879
   macro avg       0.97      0.97      0.97      9879
weighted avg       0.98      0.98      0.98      9879
```

Figura 5.10. Valutazione del modello artificial neural network in fase di test

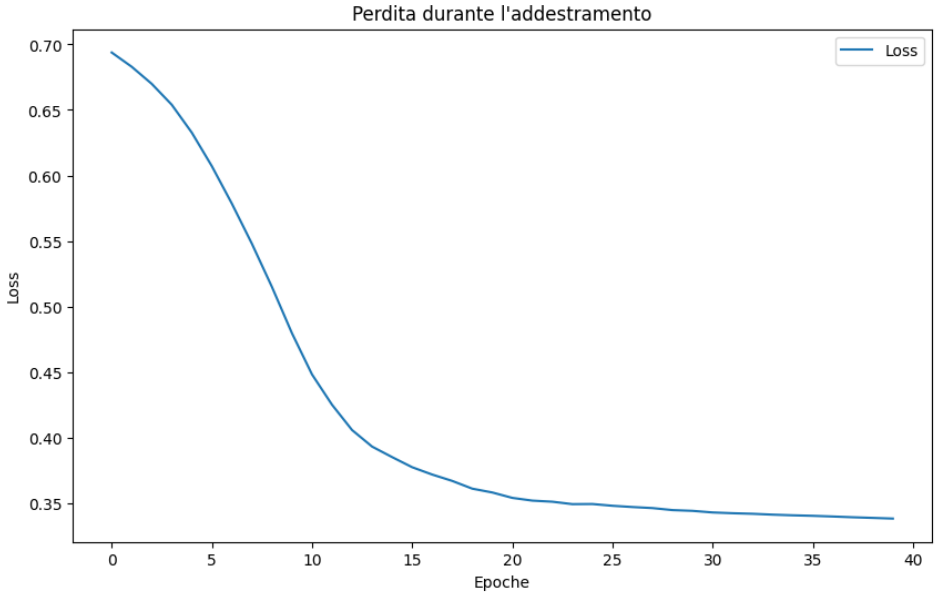


Figura 5.11. Grafico della funzione *loss* della rete neurale

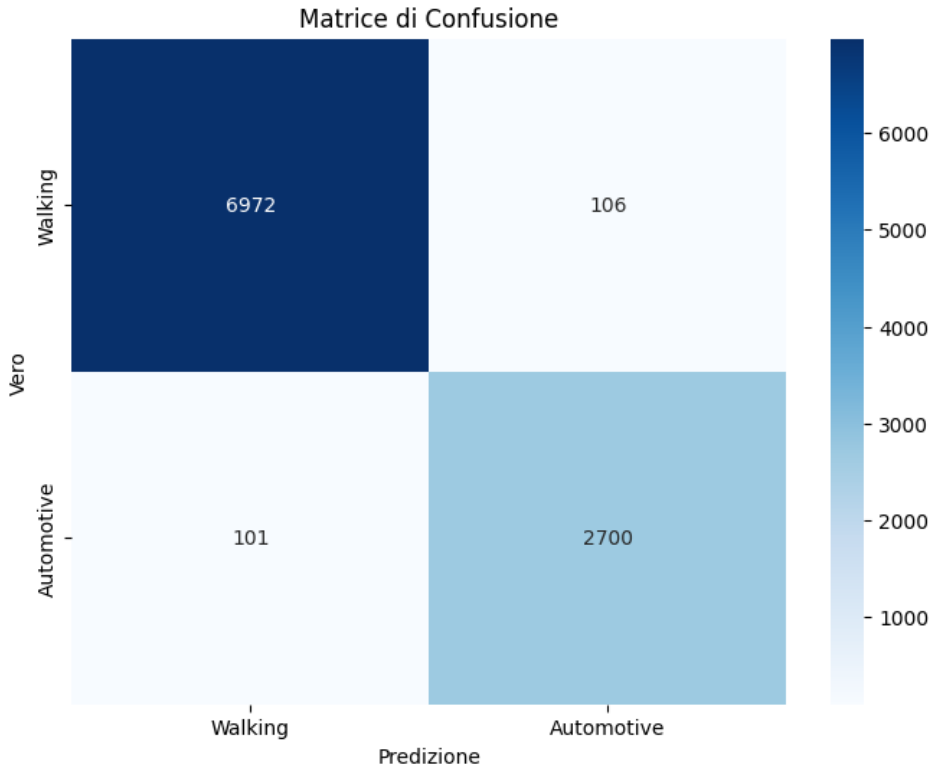


Figura 5.12. Grafico Confusion Matrix della rete neurale in fase di test

5.2.4 Convolutional Neural Networks

Le Convolutional Neural Networks[12] (CNN) sono simili alle reti neurali artificiali tradizionali (NN) in quanto sono composte da neuroni che si ottimizzano autonomamente attraverso l'apprendimento. Ogni neurone riceve un input ed esegue un'operazione (ossia una convoluzione, un prodotto scalare seguito da una funzione non lineare), come avviene nelle NN. Dall'input, che consiste in vettori di dati grezzi, fino all'output finale che rappresenta il punteggio di classificazione, l'intera rete esprime una singola funzione di valutazione percettiva (il peso). L'ultimo strato contiene le funzioni di perdita associate alle classi, e tutte le tecniche comunemente utilizzate nelle NN tradizionali sono applicabili anche qui.

La differenza principale tra le CNN e le NN tradizionali risiede nel fatto che le CNN sono impiegate principalmente nel campo del riconoscimento di pattern all'interno delle immagini. Questa peculiarità consente di incorporare caratteristiche specifiche delle immagini nell'architettura della rete, rendendola più adatta a compiti focalizzati sulle immagini, riducendo al contempo il numero di parametri necessari per la configurazione del modello.

Nel contesto del progetto, è stata impiegata una rete convoluzionale 1D, la quale, anziché elaborare matrici di dati come nel caso delle immagini, riceve in input una serie di vettori che rappresentano i valori dei sensori. Questo tipo di rete è in grado di rilevare i pattern non lineari tra i dati provenienti dai sensori.
Come illustrato nella figura 5.13, la rete è composta da tre strati convoluzionali e due strati completamente connessi. Tra questi strati viene applicata la funzione `MaxPool1D`, che riduce progressivamente la dimensione dei dati. Inoltre, ogni strato convoluzionale è seguito dalla funzione di attivazione `ReLU`, che introduce non linearità nel modello e aiuta a migliorare le capacità di apprendimento della rete.

Il modello è stato addestrato per un totale di 80 epoche. Come mostrato nel grafico della *loss* in figura 5.14, è possibile monitorare l'andamento della funzione di perdita durante ogni epoca.

Nella figura 5.15 sono riportate le metriche di valutazione del modello, applicato ai dati del set di test. Inoltre, è visualizzata la matrice di confusione (figura 5.16), che consente di analizzare in dettaglio le prestazioni del modello nella classificazione delle diverse classi, evidenziando eventuali errori e corretta assegnazione delle etichette predette rispetto a quelle reali.

```python
class CNN1D(nn.Module):
    def __init__(self):
        super(CNN1D, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=1, out_channels=64,
        ↪  kernel_size=5, padding=2)
        self.conv2 = nn.Conv1d(in_channels=64, out_channels=128,
        ↪  kernel_size=5, padding=2)
        self.conv3 = nn.Conv1d(in_channels=128, out_channels=256,
        ↪  kernel_size=5, padding=2)
        self.pool = nn.MaxPool1d(kernel_size=2)
        self.fc1 = nn.Linear(256 * (X_train_np.shape[2] // 4), 512)
        self.fc2 = nn.Linear(512, 2)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool(x)
        x = self.relu(self.conv2(x))
        x = self.pool(x)
        x = self.relu(self.conv3(x))
        x = x.view(x.size(0), -1)
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Figura 5.13. Codice della convolutional neural network

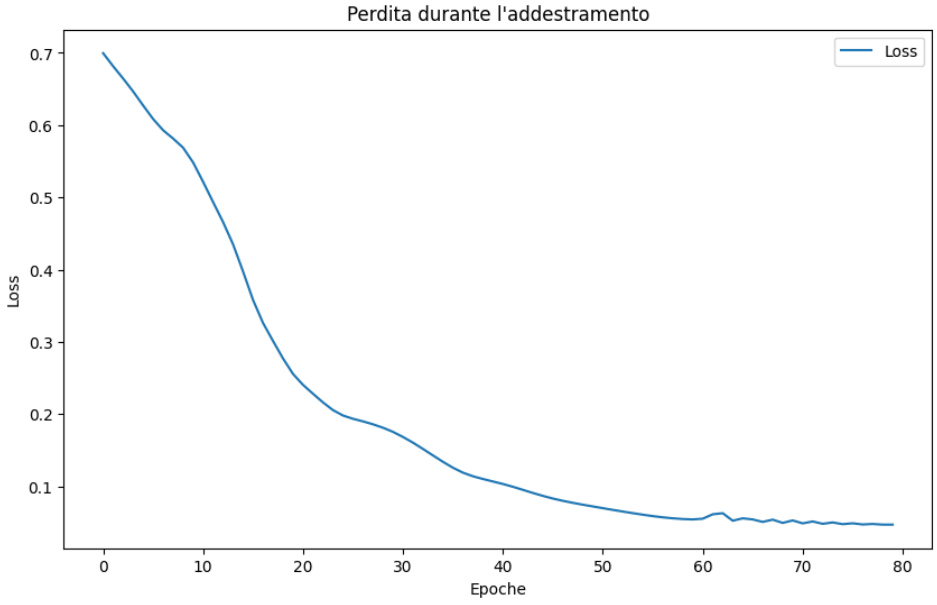


Figura 5.14. Grafico della funzione *loss* della rete neurale convoluzionale

```
Accuracy: 0.983
              precision    recall  f1-score   support

     Walking       0.99      0.99      0.99      7078
  Automotive       0.97      0.97      0.97      2801

    accuracy                           0.98      9879
   macro avg       0.98      0.98      0.98      9879
weighted avg       0.98      0.98      0.98      9879
```

Figura 5.15. Valutazione del modello convolutional neural network in fase di test

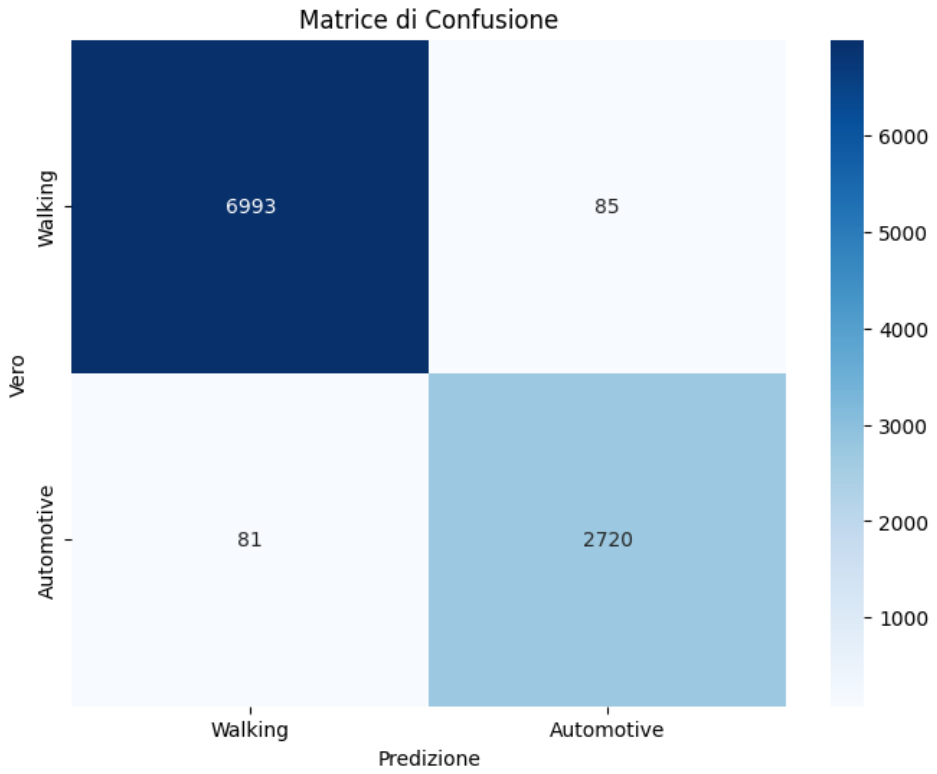


Figura 5.16. Grafico Confusion Matrix del modello convolutional neural network in fase di test

Capitolo 6

Conclusioni

In questo elaborato ho descritto la creazione e l'allenamento di un modello di machine learning per la rilevazione dello stato di guida di un utente che usa un'applicazione per smartphone. Sapendo che i dati per l'allenamento di un modello basato su sensori del telefono, quali bluethooth, gps, giroscopio etc., finalizzati a questo scopo non sono facilmente reperibili, ho deciso di esplorare la possibilità di allenare il modello su dati sintentici. Questi ultimi sono stati creati a partire da questionari di utenti veri, in modo da basare il modello su abitudini di utenti reali.

6.1 Risultati e comparazioni

I modelli implementati per questo progetto mostrano un notevole potenziale, dimostrando la capacità di predire lo stato di utenti non inclusi nel set di addestramento. In particolare, sono stati valutati quattro diversi algoritmi di machine learning: Random Forest, Decision Tree, Artificial Neural Network e Convolutional Neural Network. Di seguito verrà presentata una comparazione delle prestazioni e delle caratteristiche di essi.

Modello	Accuracy	Training(s)	Test(s)	Parametri	Peso(KB)
Random Forest	98.7%	0.051	0.0054	231	18.68
Decision Tree	93.3%	0.009	0.0032	9	2.38
NN	98.4%	1.692	0.0003	680898	2663.5
CNN	98.3%	5.872	0.0003	338178	1324.76

Il numero di parametri di un Decision Tree corrisponde al numero di nodi dell'albero, allo stesso modo i parametri di una Random Forest sono i nodi di ogni albero moltiplicati per il numero di alberi all'interno della foresta.

I test sono stati condotti utilizzando Google Colab con runtime abilitato alla GPU. In particolare, la rete neurale artificiale e la rete convoluzionale sono state addestrate e testate sfruttando la GPU T4. Tuttavia, per i modelli Random Forest e Decision Tree, l'addestramento è avvenuto esclusivamente sulla CPU, in quanto la libreria `sklearn` non supporta l'esecuzione su GPU. Questo ha inevitabilmente influenzato i tempi di addestramento e test di questi due modelli, che tuttavia sono trascurabili.

Come riportato nei report di *feature importance* dei primi due modelli (figura 5.5 e 5.8), Bluetooth risulta essere la feature più importante per la classificazione. Questi dati così estremi potrebbero far intuire che le prestazioni del modello siano

trainate esclusivamente da questo sensore, tuttavia per tutti e quattro i modelli sono stati effettuati test includendo o meno i valori di determinati sensori e i livelli prestazionali non sono calati drasticamente suggerendo una solidità non indifferente dei modelli e dell'algoritmo di generazione del dataset sintetico.

6.2 Lavori futuri

In questa sezione verranno presentati alcuni possibili sviluppi futuri volti a migliorare sia l'efficienza del programma GeneraSet che le prestazioni dei modelli predittivi implementati.

- Per migliorare la qualità dei dati si potrebbero somministrare i questionari a un numero maggiore di utenti. L'inclusione di una popolazione più ampia e diversificata consentirebbe di raccogliere dati più rappresentativi, aumentando così la variabilità influenzando la generalizzabilità dei modelli predittivi. Questo approccio potrebbe migliorare le performance dei modelli in fase di addestramento e test, garantendo una maggiore robustezza nella previsione di nuovi dati non visti.
- Per aumentare il realismo nella simulazione dei dati sensoriali, si potrebbe introdurre un ritardo tra il momento in cui l'utente sale in auto e il momento in cui il sensore lo rileva. Questo ritardo potrebbe derivare da vari fattori tecnici, come la latenza del dispositivo o la trasmissione dei dati, oppure da circostanze pratiche, come il fatto che l'utente colleghi il dispositivo all'auto solo dopo alcuni minuti rispetto a quando vi entra. Implementare questo tipo di ritardo migliorerebbe la rappresentazione delle condizioni reali e di conseguenza i modelli predittivi più robusti.
- Per una maggiore precisione dei dati si potrebbero integrare all'interno del sistema ulteriori sensori avanzati come Bluetooth BLE 4.2.2. Inoltre potrebbero essere implementati sensori che tengono traccia del tragitto che sta svolgendo un utente e capire se esso è correlato ad una automobile o strettamente ad un mezzo pubblico (nel caso di una metropolitana o un treno per esempio).
- E' in progetto di eseguire test su dati reali, raccolti in tempo reale da un'applicazione di test, ciò andrebbe a valutare in maniera appropriata i modelli costruiti dai dati sintetici.
- Un'ulteriore direzione di sviluppo potrebbe includere l'implementazione di modelli di previsione avanzati come le LSTM (Long Short-Term Memory). Questi modelli, grazie alla loro capacità di gestire sequenze temporali e memorizzare informazioni per intervalli di tempo prolungati, sono particolarmente adatti a sfruttare il timestamp associato ai campionamenti dei sensori. Questo approccio permetterebbe di cogliere pattern temporali complessi e migliorare la capacità del sistema di riconoscere variazioni nei dati legate alla dinamica temporale degli eventi.
- Quando l'applicazione GeneroCity sarà operativa, sarà possibile raccogliere i dati forniti dai sensori, offrendo un'opportunità preziosa per migliorare i modelli di machine learning tramite il *fine-tuning*. Questo processo permetterà di aggiornare i modelli già allenati, adattandoli a nuovi dati e ottimizzando ulteriormente le loro prestazioni. In questo modo, sarà possibile rafforzare la capacità predittiva dei modelli, migliorando la loro accuratezza e generalizzazione rispetto a nuove situazioni o variazioni nei dati raccolti.

Bibliografia

[1] ISTAT: **Pubblico registro automobilistico** http://dati.istat.it/Index.aspx?DataSetCode=DCIS_VEICOLIPRA

[2] ANFIA: **Dati storici sull'uso dei veicoli in Italia** https://www.anfia.it/it/attivita/studi-e-statistiche/automobile-in-cifre/statistiche-italia/parco-circolante/autoveicoli-motor-vehicles

[3] Confcommercio: Prima **indagine nazionale sulla Sosta e Parcheggi** realizzata dall'Aipark https://www.confcommercio.it/-/centri-urbani-cercasi-parcheggi-disperatamente

[4] Nikolenko, S.I. (2021). Synthetic-to-Real Domain Adaptation and Refinement. In: **Synthetic Data for Deep Learning**. Springer Optimization and Its Applications, vol 174. Springer, Cham.

[5] Goodfellow, I., et al. (2014). **Generative Adversarial Nets**. Advances in Neural Information Processing Systems

[6] Dube, K., Gallagher, T. (2014). Approach and Method for **Generating Realistic Synthetic Electronic Healthcare Records** for Secondary Use. In: Gibbons, J., MacCaull, W. (eds) Foundations of Health Information Engineering and Systems. FHIES 2013. Lecture Notes in Computer Science, vol 8315. Springer, Berlin, Heidelberg.

[7] Humza Naveeda, Asad Ullah Khana, Shi Qiub, Muhammad Saqibc, Saeed Anware, Muhammad Usmane, Naveed Akhtarg, Nick Barnesh, Ajmal Mian (2024). A Comprehensive Overview of **Large Language Models**

[8] Poolsawad, N., Kambhampati, C., & Cleland, J. G. F. (2014). **Balancing class** for performance of classification with a clinical dataset. In proceedings of the World Congress on Engineering (Vol. 1, pp. 1-6).

[9] Breiman, L. (2001). **Random Forests**.

[10] De Ville, B. (2013). **Decision trees**. Wiley Interdisciplinary Reviews: Computational Statistics, 5(6), 448-455.

[11] Grossi, Enzo & Buscema, Massimo. (2008). Introduction to **Artificial Neural Networks**. European journal of gastroenterology & hepatology. 19. 1046-54.

[12] O'Shea, K. (2015). An introduction to **Convolutional Neural Networks**.