Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica per il Management

Federated Learning: un'applicazione per il riconoscimento dell'attività umana

Relatore:
Chiar.mo Prof.
MARCO DI FELICE

Presentata da:
TOMMASO QUINTAVALLI

Sessione IV
Anno Accademico 2022/2023

Hos ego versiculos feci, tulit alter honores
Sic vos non vobis mellificatis apes

Virgilio

Abstract

Il **Federated Learning** è un approccio moderno e innovativo all'Apprendimento Automatico che si pone come obiettivo quello di risolvere i problemi relativi alla **privacy** e alla **tutela dei dati** che caratterizzano inevitabilmente i classici algoritmi di Machine o Deep Learning.

Questi sono infatti basati sulla centralizzazione totale dei dati, processo che mette a rischio gli utenti, cioè gli stessi produttori di dati, spesso e volentieri ignari dell'utilizzo che i servizi effettivamente ne faranno.

L'apprendimento federato promette di superare l' approccio tradizionale, evitando la centralizzazione dei dati e sfruttando le capacità di calcolo dei singoli dispositivi e la loro collaborazione per allenare il modello di turno.

Un'architettura del genere, sulla carta, dovrebbe permette di ottenere gli stessi risultati senza andare a minare la sicurezza di chi i dati li crea, evitando il rischio di perdite o di utilizzo illecito.

Questa Tesi ha l'obiettivo di capire vantaggi e limiti della classificazione tramite un approccio federato, mediante un caso di studio basato sul riconoscimento delle attività motorie umane da dati dell'Internet of Things (IoT).

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Negli ultimi anni la produzione di dati da parte di smartphone, sensori, telecamere, veicoli, così come da dispositivi tradizionali, è crescita esponenzialmente.

I cosiddetti big data però non sono fini a loro stessi: hanno un **valore**, anche intrinseco, molto elevato, *in primis* di tipo **economico**.

Uno studio pubblicato dal Fondo Monetario Internazionale ha evidenziato come non esista, nel mondo della *digital economy*, nulla di veramente gratuito. Le piattaforme online offrono beni e servizi apparentemente gratis, in cambio dei dati dei consumatori, che vengono trasformati in modo da renderli monetizzabili.

Per osservare questo basta guardare al valore di *Meta/Facebook*: si stima che gli asset dell'azienda valessero circa 6,3 miliardi di dollari al momento della sua quotazione in borsa nel 2011, ma la valutazione di mercato della società ha raggiunto velocemente i 104 miliardi di dollari, evidenziando il contributo delle risorse fornite dai propri utenti[1].

Il secondo "tipo" di valore che possiamo attribuire ai nostri dati è quello della **sicurezza** e, in parte, anche quello **etico**: i dati che produciamo contengono, anche se non di facciata, tantissime informazioni su di noi, anche e spesso **sensibili**. Il Garante della Privacy identifica nei dati sensibili quelli che permettono l'identificazione diretta o indiretta di un individuo, quelli rientranti in particolari categorie come quelli che rivelano l'origine razziale o etnica, le convinzioni religiose, filosofiche, le opinioni politiche, l'appartenenza sindacale, relativi alla salute o alla vita sessuale, o ancora i dati genetici, i dati biometrici

e quelli relativi all'orientamento sessuale; quelli relativi alle comunicazioni elettroniche (via Internet o telefono) e quelli che consentono la geolocalizzazione. Dovrebbe bastare questo elenco per capire che la necessità di strumenti per aumentare la protezione dei dati sensibili.

Negli ultimi anni tuttavia i dati degli utenti vengono spesso e volentieri, anche senza la loro consapevolezza, convogliati verso dei *data center* cloud, dove vengono raccolti e analizzati. Il fine ultimo è spesso quello di addestrare modelli di **Machine e Deep Learning** al fine di catturare pattern o profilare gli utenti.

L'idea di questo progetto di Testi nasce con l'obiettivo di identificare e testare una soluzione alternativa al classico approccio centralizzato di addestramento per un modello di apprendimento automatico. Tale soluzione viene individuata in una tecnica di recente introduzione nello stato dell'arte dell'intelligenza artificiale, denominata **Federated Learning**.

Il **Federated Learning** è una tecnica di apprendimento automatico che permette di addestrare un algoritmo attraverso la collaborazione dei nodi aderenti al processo, **senza** la necessità di scambiare i dati stessi, e quindi **decentralizzando** la raccolta dei dati.

I **nodi**, chiamati anche *client* od organizzazioni, possono essere rappresentati dai vari dispositivi che creano effettivamente i dati, fino ad arrivare a intere strutture che li raccolgono (un esempio è una rete di ospedali che raccoglie i dati delle proprie cartelle cliniche e condivide i risultati che ottiene con altri ospedali nella stesse rete). I client sono solitamente gestiti da un **server** che ha l'unico compito di **coordinare** il processo.

La Tesi si compone di una parte di rassegna dello **Stato dell'Arte** dell' intelligenza artificiale, in modo da introdurre il concetto di apprendimento automatico e di capire i motivi per i quali un approccio "classico" potrebbe essere problematico.

Si spiega poi il concetto di **riconoscimento dell'attività umana**, area di ricerca di cui fa parte il caso di studio scelto per la parte "sperimentale" della tesi.

Viene quindi illustrato nel dettaglio il funzionamento dell'apprendimento federato, oltre ai vantaggi e svantaggi che esso porta.

Successivamente si arriva alla **Progettazione** della parte sperimentale, dove vengono elencati e "raccontati" i passaggi seguiti. Vengono testati alcuni modelli di Machine Learning e, successivamente, viene fatto un confronto tra questi e un modello di Federated Learning. Inoltre, vengono dimostrate quelle che sono le possibili limitazioni di questo secondo approccio.

Si descrivono quindi brevemente le **Tecnologie** utilizzate nella sezione di **Implementazione**.

I capitoli riguardanti **Implementazione** e **Risultati** contengono le scelte tecniche, il codice sviluppato e gli esiti degli esperimenti.

La parte di **Conclusione** raccoglie le osservazioni a cui si è arrivati analizzando i risultati.

Capitolo 2

Stato dell'arte

2.1 Machine Learning

"L'obiettivo principale dell'apprendimento automatico è che una macchina sia in grado di generalizzare dalla propria esperienza": è questa, secondo la **teoria dell'apprendimento**, la spiegazione che permette di iniziare a capire cos'è il Machine Learning, branca dell'Informatica e della Scienza dei Dati, che, dall'inizio della seconda metà del XX secolo, ha preso sempre più spazio in discussioni, dibattiti, lezioni e articoli.

Con **generalizzazione** si intende l'abilità da parte della macchina di eseguire compiti nuovi, che non ha mai "visto" prima, dopo aver fatto esperienza su altri dati più o meno correlati.

L'informatico Tom M. Mitchell, uno dei più importanti contributori nello sviluppo di questi meccanismi, definisce[12] il Machine Learning dicendo che:

> *"Un programma apprende dall'esperienza E con riferimento ad alcune classi di compiti T e con misurazione della performance P, se le sue performance nel compito T, come misurato da P, migliorano con l'esperienza E"*,

oppure, più in breve, che una macchina apprende se riscontra un **miglioramento nelle prestazioni** dopo aver svolto un compito.

Il processo di apprendimento si basa su dati caratterizzati da una **distribuzione** probabilistica non nota ma considerata **rappresentativa** dello spazio dei campioni del caso di studio: l'obiettivo della macchina è creare un modello generale riguardante questo spazio che le permetta di predire in modo efficace (ed efficiente) il comportamento di un nuovo caso.

2.2 Storia

L'inizio della ricerca e delle sperimentazioni in questo ambito scientifico va ricercata nei primi anni '40, quando **Walter Pitts**, matematico, e **Warren McCulloch**, neurofisiologo, ne iniziarono a porre le basi, ideando il primo modello matematico di una **rete neurale** rappresentante la struttura del cervello umano[4].

I risultati ottenuti sono estremamente attuali: l'unità di base della struttura, cioè la rappresentazione di un singolo **neurone**, è ancora il riferimento standard nel campo delle reti neurali; inoltre, ha influenzato vari ambiti, come le scienze cognitive, la filosofia, la cibernetica e, appunto, l'intelligenza artificiale.

Nel 1943 i due scienziati svilupparono la prima rete neurale con un circuito elettrico incorporato, con l'obiettivo di fare comunicare due computer per la prima volta. Alla fine degli anni '40, lo psicologo **Donald Olding Hebb** introdusse il concetto della comunicazione fra neuroni, ispirando la ricerca sui neuroni artificiali basandosi su quelli del cervello umano.

Nel 1950 **Alan Turing** introduce l'omonimo "**Test di Turing**" [20]: questo si compone di una persona impegnata in una conversazione scritta con un'altra persona e con una macchina, con l'obiettivo di capire quale dei due è un umano. Questo, nonostante le sue criticità, è stato il primo esempio di **pensiero critico** sviluppato da un computer.

Pochi anni dopo, nel 1952, l'informatico **Arthur Samuel** sviluppa un programma per giocare a dama[18]. Molto interessato nello studio dei giochi per lo sviluppo del Machine Learning, piuttosto che da problemi "reali", la dama permette di studiare come le

classiche **tecniche euristiche** e l'apprendimento possano essere combinate per ottenere risultati migliori rispetto all'utilizzo mutualmente esclusivo dei due approcci.

Nel 1957 lo psicologo **Frank Rosenblatt** implementa la prima versione hardware del **percettrone** ideato da Pitts e McCulloch nella sua macchina *Mark I Perceptron*, progettata per il riconoscimento di immagini. Nonostante le limitazioni tecniche, fu il primo con la capacità di calibrare i propri pesi e parametri in risposta ai dati in ingresso.

Dopo uno stallo nello sviluppo di queste tecnologie, dagli anni ottanta il Machine Learning vira verso la **probabilità** e la **statistica**. Dal decennio successivo, con la crescita della **disponibilità dei dati** e con la capacità di **distribuzione dei servizi** grazie a Internet, vengono sviluppati nuovi algoritmi ed entrano in gioco nuove risorse, come la GPU.

2.3 Dati

In generale, un algoritmo di apprendimento automatico necessita di alcuni componenti fondamentali: vediamo nello specifico cosa si intende con **dati** e **modelli**, i due più importanti.

I dati sono parte essenziale di ogni algoritmo di Apprendimento Automatico, ciò che viene fornito in input al processo per ottenere un output. Sono organizzati in **datasets**, cioè sotto forma di tabelle dove ogni colonna rappresenta una **variabile** o *feature* e ogni riga rappresenta una **osservazione** o *record*.

Generalmente il dataset viene suddiviso in tre sottoinsiemi per ottimizzare le prestazioni degli algoritmi:

- **Training Set:** il sottoinsieme che viene dato in pasto all'algoritmo per addestrarsi. Ad esempio, potrebbe contenere circa il 70% del dataset di partenza.
- **Validation Set:** il sottoinsieme usato per capire le performance dell'algoritmo in confronto ad altri tipi di modello e/o a diverse configurazioni degli iper-parametri. Ad esempio, contiene circa il 15% dei dati di partenza.

- **Test Set:** il sottoinsieme usato per valutare l'affidabilità dell'algoritmo. Come il Validation Set, contiene circa il 15% dei dati di partenza.

2.4 Approcci

Per entrare nel dettaglio della progettazione di un meccanismo di Machine Learning, è doveroso evidenziare che l'approccio per far sì che una macchina impari non è univoco; nonostante ciò, la struttura generale è comune:

1. Raccolta e preparazione dei dati
2. Creazione del modello
3. Addestramento del modello
4. Ottimizzazione degli iper-parametri
5. Validazione del modello

Esistono tre macro-categorie di approcci, corrispondenti alla natura del segnale o *feedback* a disposizione della macchina:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

2.4.1 Supervised learning

Con l'approccio supervisionato gli oggetti in input (i vari *record*) e il rispettivo output (chiamato **label**) sono **noti** alla macchina. Questo tipo di problema prende il nome di **classificazione**, cioè creare un modello che possa imparare a distinguere tra diverse categorie di dati e successivamente assegnare automaticamente una categoria a nuovi dati non etichettati. I dati di training vengono processati per creare una funzione che mappi dati nuovi con uno degli output attesi.

2.4.2 Unsupervised learning

L'approccio non supervisionato consiste nella classificazione di dati dei quali la macchina **non conosce l'output**.

L'idea è quella di sfruttare il concetto di **mimica**, modalità di apprendimento tipica dell'essere umano per la quale la macchina costruisce una rappresentazione del mondo a cui appartengono dati di training, permettendo di generare dei nuovi contenuti propri di questo "mondo".

Nella fase di training, il modello prende in input i dati "grezzi" e deve dedurre le proprie regole e strutturare le informazioni sulla base di somiglianze, differenze e pattern senza istruzioni esplicite su come lavorare con ciascun dato.

Gli algoritmi di apprendimento non supervisionato sono più adatti per attività di elaborazione complesse, come l'organizzazione di set di dati in *cluster* [5].

2.4.3 Reinforcement learning

È un processo che si focalizza sull'ottimizzazione delle decisioni adottate da un agente al fine di massimizzare una determinata "**ricompensa**". Utilizza approcci basati sulla programmazione dinamica, specialmente nell'analisi del *prossimo stato generato*.

Tali algoritmi trovano applicazione in contesti in cui la rappresentazione dei dati attraverso un modello matematico specifico risulta impraticabile, come nel caso dei sistemi di veicoli a guida autonoma.

2.5 Modelli

Il **modello** è il risultato di un algoritmo di Machine Learning ed è la rappresentazione matematica di una funzione che deduce dei pattern dai dati, capace di prendere decisioni riguardanti nuovi dati mai visti.

Esistono vari tipi di modello, che si differenziano in base al problema da risolvere e all'approccio scelto. Qui ne vediamo alcuni, quelli usati nella parte sperimentale di questa Tesi.

2.5.1 Decision Trees

In un contesto supervisionato, sono la più semplice implementazione di Apprendimento Automatico, basati su un processo di **scelte successive**[17].

Si tratta di un modello non parametrico che permette di eseguire sì operazioni di **classificazione** (la ricerca di un output discreto) ma anche di **regressione** (la ricerca di un output continuo).

Si struttura gerarchicamente come un **grafo** in:

- Radice
- Rami
- Nodi interni
- Nodi foglie

La **radice** rappresenta l'interno dataset. I **nodi interni** rappresentano ognuno una variabile e i **rami** un possibile valore per quella *feature*. Infine, i nodi **foglia** corrispondono al valore predetto per la variabile obiettivo.

L'algoritmo di un *decision tree* si articola così:

1. Si parte dalla radice, che rappresenta l'intero dataset;
2. Vengono analizzate le features e si individua quella che divide il dataset massimizzandone l'**entropy gain**;

3. In base alla risposta, suddivide il dataset in porzioni più piccole, creando nuovi rami: ognuno di questi rappresenta una ***route*** all'interno dell'albero;

4. Si ripetono i passaggi $2 - 3$ fino a che non si raggiungono i nodi *foglia*, che rappresentano il risultato predetto (ad esempio, la classe risultante).

L'algoritmo rappresenta le decisioni e il loro risultato, considerandone la probabilità, le risorse spese e l'utilità dell'esito, seguendo un ***path*** dalla radice alla foglia.

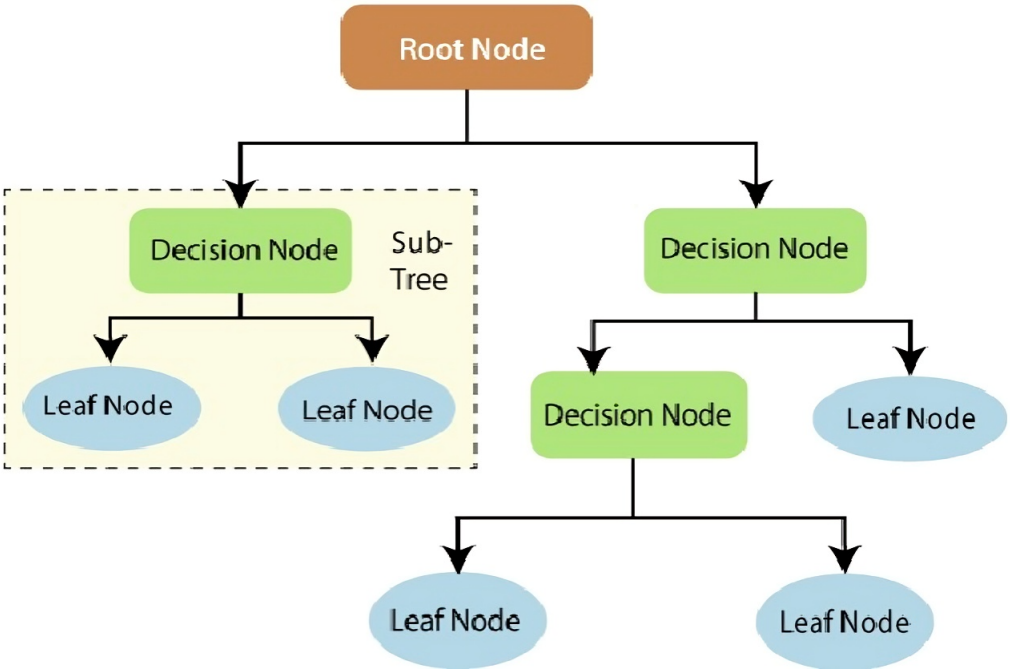

Figura 2.1: Struttura di un albero decisionale

2.5.2 Random Forests

Modelli per l'apprendimento supervisionato, sono **collezioni di alberi decisionali**. Rendono il processo più robusto e affidabile usando più classificatori, con la decisione finale che viene poi presa, solitamente, a maggioranza[3].

Le foreste sono utili perché limitano il rischio di **overfitting**: questo è un classico problema dei classificatori, che avviene quando il modello si "lega" tropo ai dati di training, perdendo la capacità di generalizzazione sui dati nuovi.

Le foreste evitano questo problema sfruttando e combinando varie tecniche:

- **Bootstrapping**: tecnica per la quale ogni albero riceve in input un sottoinsieme del dataset di partenza, con possibile ripetizione dei *record*: questo permette alla foresta di essere meno sensibile ai dati di partenza;
- **Feature Randomness**: tecnica per quale ogni albero analizza solo un sottoinsieme delle features del dataset di partenza (da cui anche il nome "*random*" forests); diminuendo la correlazione tra gli alberi;
- **Bagging**: unione di **Bootstrapping** e **Aggregation**, cioè la ricerca del risultato della classificazione, scegliendo per maggioranza in base ai risultati dei singoli alberi.

Una caratteristica importante di un modello di random forests, ma in generale di ogni modello di Machine Learning, è rappresentata dai suoi **iper-parametri**: questi sono parametri esterni all'apprendimento automatico e devono essere impostati prima dell'avvio del processo, tramite un processo chiamato ***Hyper-parameters Tuning***.

2.6 Deep Learning

Il **Deep Learning** è una branca del Machine Learning che avvicina le metodologie di apprendimento delle macchine a quello del cervello umano. Lo strumento usato dal Deep Learning è la **Rete Neurale Artificiale** (*ANN, Artificial Neural Network*), o brevemente Rete Neurale.

2.6.1 Reti Neurali

Le Reti Neurali sono modelli costruiti sui principi dell'organizzazione delle reti neurali biologiche che costituiscono il cervello degli animali[6].

L'unità base che le costituisce è il **neurone artificiale**, una rappresentazione del neurone biologico, connesso con gli altri neuroni tramite archi che imitano invece le

sinapsi. Il neurone artificiale è una funzione matematica che riceve uno o più input, li elabora applicando dei pesi e li somma per costruire un output. Sia input che output sono numeri reali.

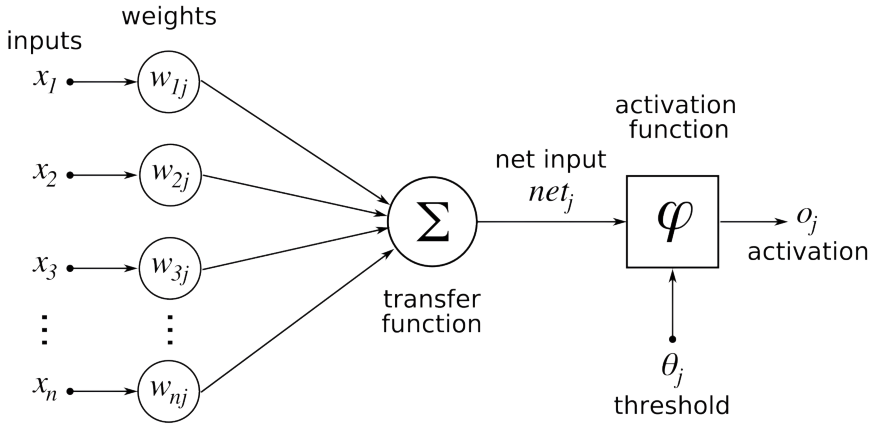

Figura 2.2: Struttura di un neurone artificiale

Ogni input viene pesato separatamente, e il risultato viene sommato ad un termine chiamato *bias*. Il risultato viene poi passato in una funzione non lineare che si chiama **funzione di attivazione** (solitamente, è una *sigmoide* oppure una $ReLU$).

Il numero reale che passa tra le varie componenti e, di conseguenza, tra i vari neuroni prende il nome di **segnale**, lo stesso che ha l'impulso elettrico che attraversa il cervello: i pesi dei neuroni e degli archi influiscono in positivo o in negativo sulla *forza* del segnale in una connessione, cioè la sua importanza.

I neuroni in una rete sono disposti su più livelli o *layers*. Ogni livello può eseguire trasformazioni e operazioni diverse sui suoi input. Una rete è composta almeno da due *layers*: quello di ***input***, che riceve i dati dall'esterno, e quello di ***output***, che produce il risultato finale. È comune trovare anche livelli intermedi: se una rete ne ha almeno due si dice che è **profonda**: da qui il nome **deep learning**.

È importante sottolineare che i neuroni di un livello comunicano solo con quelli del livello strettamente precedente o successivo: il modo in cui sono organizzate le connessioni determina il tipo di rete risultante:

- Rete **Feedforward**: in questo tipo di rete i neuroni sono organizzati in un grafo **aciclico** e **diretto**, in modo che il flusso del segnale sia unidirezionale. È il tipo di rete utilizzato nella Tesi.

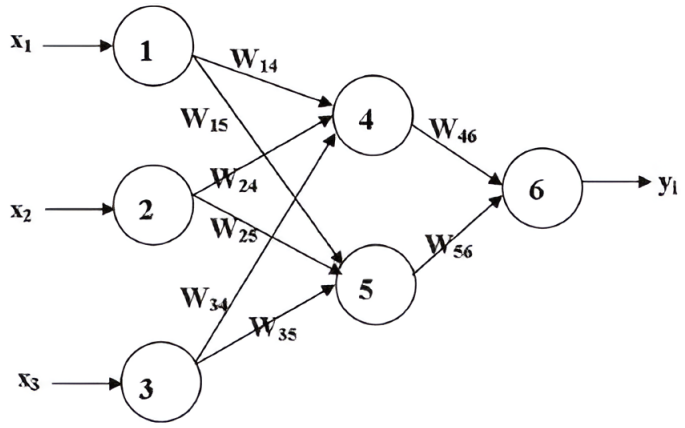

Figura 2.3: Struttura di una rete neurale artificiale *Feedforward* [11]

- Rete **Ricorrente**: al contrario della precedente, il flusso del segnale è bidirezionale, permettendo all'output di un nodo di influenzare direttamente l'input dello stesso.

2.7 Misurazione delle Performance

Fino a ora abbiamo parlato di ricerca e scelta del modello migliore in base al contesto in cui opera. Questa scelta viene effettuate andando a guardare le **performance** di un modello, quantificate da misurazioni statistiche basate sulla probabilità. Vediamo nel dettaglio le metriche usate in questo progetto, che fanno particolare riferimento alle situazioni di **classificazione**.

Matrice di Confusione Una **matrice di confusione**, nel contesto di una classificazione con più classi, è una tabella che mostra le prestazioni di un modello di classificazione rispetto alle diverse classi di output.

La matrice di confusione è organizzata in modo che le righe rappresentino le **classi reali** e le colonne rappresentino le **classi predette** dal modello. Ogni cella della matrice

contiene il numero di istanze appartenenti alla classe corrispondente alla riga e predette come appartenenti alla classe corrispondente alla colonna.

Le celle della matrice contengono la quantificazione 4 fattori di valutazione:

- **True Positive (TP)**: numero di istanze che il modello ha correttamente predetto come appartenenti a una classe specifica.
- **True Negative (TN)**: numero di istanze che il modello ha correttamente predetto come **non** appartenenti a una classe specifica. Questa grandezza è in realtà poco rilevante nel contesto della classificazione, al di fuori di quella binaria.
- **False Positive (FP)**: numero di istanze che il modello ha erroneamente predetto appartenenti a una classe.
- **False Negative (FN)**: numero di istanze che il modello ha erroneamente predetto **non** appartenenti a una classe.

Una matrice di confusione, nel caso di classificazione multi-classe, si presenta visivamente così:

Matrice di Confusione	Classe A	Classe B	Classe C
Classe A	TP	FP	FN
Classe B	FN	TP	FP
Classe C	FP	FN	TP

Tabella 2.1: Esempio di Matrice di Confusione

Grazie a questi concetti, possiamo introdurre il calcolo delle effettive metriche di valutazione.

Accuratezza L'accuratezza è la misurazione più semplice ed è definita come il numero di predizione corrette diviso il numero di predizioni totali.

Precisione È il rapporto tra i *True Positive* e il totale dei Positivi:

$$\frac{TP}{TP + FP} \tag{2.1}$$

Recall È il rapporto tra i *True Positive* e tutti i **veri** positivi:

$$\frac{TP}{TP + FN} \tag{2.2}$$

F1-score È una metrica che rappresenta una combinazione tra **precisione** e **recall** (è la loro *media armonica*):

$$\frac{2}{\frac{1}{precision} + \frac{1}{recall}} \tag{2.3}$$

Intervallo di Confidenza Essendo basati su calcoli probabilistici, le analisi di algoritmi di Machine Learning non forniscono come risultato valori esatti.

Gli **intervalli di confidenza** servono a definire un limite inferiore e uno superiore rispetto alla misurazione media di accuratezza, precisione, recall ed F1-score entro i quali è *probabile* che il vero valore dei risultati si trovi.

L'ampiezza di questo intervallo dipende strettamente dal **livello di confidenza** scelto: un livello di confidenza al 95% indica che in circa il 95% dei casi l'intervallo conterrà il valore vero del parametro.

Dato il livello di confidenza, l'**ampiezza** risultante dell'intervallo fornisce informazioni sull'affidabilità dell'algoritmo: più è stretto intorno al valore medio, più la metrica è significativa.

In definitiva, un algoritmo che fornisce delle buone metriche (vicine a 0,9) con un intervallo di confidenza "piccolo" è, in generale, considerato affidabile: non c'è però un valore "globale" che va ricercato, data la grande quantità di variabili in gioco.

2.8 Data Mining

Con **data mining** si intende il processo che, tramite algoritmi e modelli di Machine Learning, permette di individuare pattern, correlazioni e anomalie in grandi fonti di dati per produrre un risultato: in generale, è la pratica con la quale è possibile estrarre informazioni dai dati, trasformando un input grezzo in conoscenza pratica[19].

2.9 Big Data

Il termine **Big Data** fa riferimento a enormi ed eterogenee fonti di dati strutturati, non strutturati e semi-strutturati che continuano a crescere esponenzialmente nel tempo[10]. È importante distinguere i big data da i dati "normali", in quanti i sistemi di ***data management*** che gestiscono questi ultimi, nella maggior parte dei casi, non sono in grado di arrivare a buoni risultati con i primi.

Il concetto di **Big Data**, per quanto possa variare da caso in caso, è sempre descritto da 3 termini, chiamati "le 3 V":

- **Volume**, inteso come la grande quantità di dati da memorizzare, gestire e analizzare;
- **Velocità**, in quanto i dati vengono generati e aggiornati in tempo reale, e, di conseguenza, devono anche essere processati allo stesso ritmo;
- **Varietà**, perché i dati generati sono eterogenei, provengono da fonti diverse e possono essere organizzati diversamente.

Negli ultimi anni, con la creazione e disponibilità dei dati che è cresciuta in maniera esponenziale, si sono aggiunti altri tre termini:

- Veracità: considerato che dataset grandi possono avere un grande rumore statistico ed essere più inclini all'errore, e dataset piccoli possono essere poco significativi, è importante che la veracità sia alta.
- Variabilità: il continuo aggiornamento dei dati porta a variazioni e cambiamenti sia nel contesto e nell'interpretazione, sia nei modelli di collezione e gestione dei dati, in base a quali informazioni se ne vogliono estrarre.
- Valore: ovviamente, i dati raccolti devono contenere, anche implicitamente, le informazioni per ottenere il risultato desiderato.

Applicabile a innumerevoli contesti, tra cui le telecomunicazione, la finanza e il settore bancario, l'istruzione, la medicina, i Big Data trovano spazio anche in quello dell'**Internet of Things**, cioè quelle reti di dispositivi interconnessi che generano dati tramite sensori e, in particolare, nell'ambito del riconoscimento dell'attività umana (chiamato ***HAR - Human Activity Recognition***).

2.9.1 Human Activity Recognition

Il **riconoscimento dell'attività umana** è una branca dell'ingegneria e dell'informatica che ha l'obiettivo di creare sistemi e tecniche capaci di riconoscere e categorizzare autonomamente le azioni dell'uomo sulla base dei segnali di vari sensori [9].

Le fonti di dati per questa pratica sono molteplici: possono essere immagini o video, oppure dati provenienti dai sensori di *smartphone*, *wearables* e *smartwatch*, come il sensore GPS, l'accelerometro, il magnetometro, il giroscopio, il pedometro, il rilevatore della frequenza cardiaca e del livello di ossigeno e molti altri.

Una volta ottenuti i dati, si segue la classica procedura di data mining:

1. **Pre-processing dei dati**
   (a) *Filtraggio*: pratica che rimuove il rumore e i dati indesiderati.
   (b) *Estrazione*: in questa fase, si estrae dai dati grezzi l'informazione che si sta cercando.
   (c) *Selezione delle features*: vengono selezionate le features più significative, in modo da minimizzare lo spazio utilizzato e migliorare l'accuratezza.
   (d) *Segmentazione*: in questa fase si estraggono gli aspetti temporali delle attività, dividendoli in segmenti.
   (e) *Normalizzazione*: pratica nella quale le features vengono scalate per avere una media neutra e una varianza di valore 1, in modo da rendere l'analisi bilanciata.

2. **Selezione del modello**: in questa fase si possono testare i vari meccanismi di Machine e Deep Learning: oltre a quelli citati precedentemente, si trovano anche le *Support Vector Machines* e le Reti Neurali Convoluzionali. Quest'ultima tecnica *deep* si rivela particolarmente efficace nell'ambito di riconoscimento di immagini e video.

3. **Distribuzione del modello**: nell'ambito HAR il modello scelto viene distribuito in due modi:

   - Esternamente: i sensori esterni, come camere o rilevatori di movimento, vengono posizionati nelle aree circostanti quella che si vuole analizzare; i dati che producono vengono inviati e processati da una macchina "centrale".
   - *"On body"*: in questo caso, i sensori (come l'accelerometro) sono posizionati direttamente sul dispositivo (come uno smartwatch). I dati vengono processati direttamente sul dispositivo oppure su una macchina a esso collegata, come lo smartphone dell'individuo o una struttura esterna.

2.10 Privacy e Tutela dei Dati

Come appurato precedentemente, un modello di Machine Learning necessita *in primis* di due risorse[7]:

- Grandi quantità di dati accessibili, eterogenei e significativi
- Potenti risorse computazionali

Se il secondo requisito solleva questioni puramente tecniche e pratiche, il primo, soprattutto negli ultimi anni, fa in modo che i sistemi di Machine Learning creati dai programmatori siano trasparenti, giusti e capaci di proteggere i dati dei ***data owner*** e dei ***model users***.

Queste idee fanno parte del concetto di ***data protection***, cioè quell'insieme di proprietà basate sulla confidenzialità e la privacy: l'applicazione di questi concetti negli algoritmi e nelle architetture è sia corretto da un punto di vista **etico** che obbligatorio

da quello **legale**. Esistono infatti regolamentazioni specifiche, come il ***GDPR*** (*General Data Protection Regulation*) redatto dall'Unione Europea, che raccoglie le leggi sulla protezione dei dati.

In un classico algoritmo di Machine Learning i dati usati per il *training* del modello vengono inviati dai dispositivi degli utenti a un server centralizzato (che sia fisico o in *cloud*), che li raccoglie e li utilizza per addestrare il modello. Questo processo è però rischioso per la tutela dei dati, in particolare quelli sensibili: anche se crittografati e, almeno teoricamente, "al sicuro", i dati sono esposti ad attacchi (sia durante il trasferimento verso il server, sia una volta memorizzati) da parte di terzi.

Riprendendo il caso del riconoscimento dell'attività umana: anche questi dati possono rivelare informazioni dettagliate su abitudini e spostamenti di una persona, sul suo stato di salute, condizioni mediche specifiche, informazioni contestuali come spostamenti o contesti sociali in cui avvengono determinate attività.

In generale, tutti i tipi di dato, più o meno sensibili, posso essere oggetto di *leak* da parte di malintenzionati e "merce di scambio" da parte dei gestori del sito web del caso verso gli inserzionisti, una pratica a oggi estremamente sottovalutata da parte dei consumatori stessi che, senza esserne consapevoli, usufruiscono di servizi apparentemente gratuiti pagando con la rinuncia totale o parziale della propria privacy.

È necessario andare alla ricerca di alternative rispetto alla soluzione che vede i dati continuamente trasferiti verso un sistema esterno. Uno dei paradigmi di programmazione che sta prendendo sempre più spazio è quello del **Federated Learning** o **Apprendimento Federato**.

2.11 Federated Learning

Il Federated Learning è un paradigma di progettazione di ambienti di Machine Learning che si basa su una struttura nella quale più entità o ***client***, coordinati da un **server**, allenano in modo collaborativo un modello assicurando che i dati prodotto da ognuno rimangano **decentralizzati**[13].

Il suo obiettivo è eseguire algoritmi di Machine Learning su più dataset locali (cioè sui vari client) senza scambiarsi esplicitamente i dati, bensì i risultati della fase di training, come i pesi o i *bias* di una rete neurale.

2.11.1 Algoritmo

1. **Inizializzazione del modello globale**: il server inizializza il modello, impostandone i parametri e altre informazioni utili ai client.

2. **Invio del modello**: il modello creato (**non** ancora addestrato) viene inviato ai vari **nodi**, cioè i client che compongono la nostra "rete" (nella pratica, il modello iniziale viene inviato solo ad alcuni client).

3. **Training del modello**: ora che tutti i client selezionati hanno le stesse informazioni, cioè la "configurazione" del modello, iniziano ad addestrarlo nel modo classico *sui propri dati locali*, basandosi sulle istruzioni passata dal server, come il numero di epoche o i livello minimo di convergenza.

4. **Ritorno dei risultati**: una volta finito il training, ogni client avrà una versione leggermente diversa del modello. Invia quindi i parametri che considera ottimali verso il server.

5. **Aggregazione**: il server riceve da ogni client il modello aggiornato. A questo punto, **aggrega** i risultati ottenuti (ad esempio facendone la media pesata) per creare un unico modello che rappresenti tutti i nodi.

6. **Ciclo**: il server ripete i passaggi **2-5** fino a che il modello non converge o fino a che non raggiunge il risultato minimo ricercato.

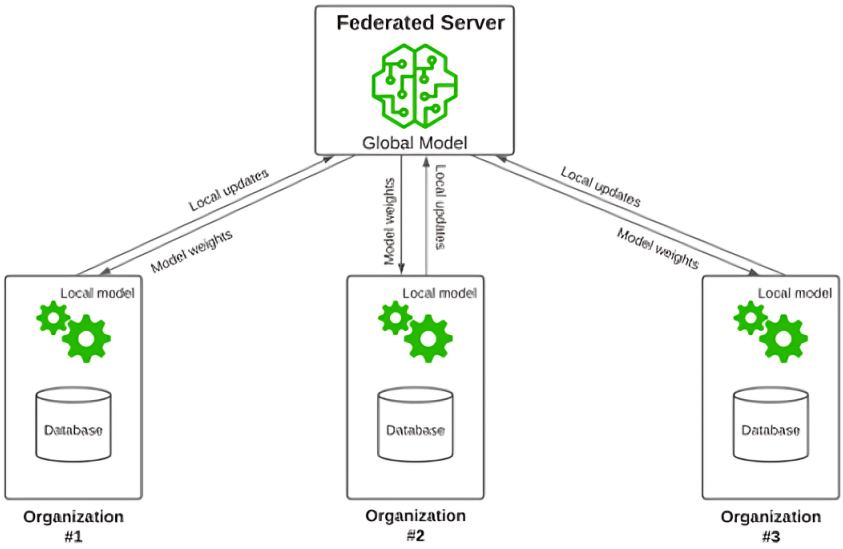

Figura 2.4: Flusso di dati dell'algoritmo di apprendimento federato [8].

Per far sì che il risultato sia un unico modello, preciso e affidabile, l'algoritmo segue un processo iterativo diviso in sezioni atomiche di iterazioni client-server che vengono chiamati ***federated learning round*** (round di apprendimento federato). Ogni round consiste nel trasmettere lo stato attuale del modello, addestrare i modelli locali e aggiornarli, e aggregare i risultati nel modello centralizzato.

2.11.2 Iper-parametri

Oltre ai vari parametri tipici del modello di Machine Learning scelto, è importante che il server fornisca una serie di informazioni riguardanti la parte "federata" dell'addestramento.

- numero di round di apprendimento federato
- numero di nodi
- frazione di nodi usata a ogni iterazione
- *batch size* locale usato a ogni iterazione

2.11.3 Concetti

Elenchiamo ora alcuni concetti che permettono di capire quali sono le potenzialità e le funzionalità di un algoritmo di apprendimento federato[2].

Federated evaluation Allo stesso modo in cui si allena il modello in maniera decentralizzata, è ovviamente possibile anche valutare il risultato di questo processo tramite **metriche di valutazione** "collaborative".

Federated analytics Non sempre è necessario ricorrere a tecniche di apprendimento automatico, che siano *deep* o meno, per estrarre informazioni dai dati o per far sì che questi abbiano un valore intrinseco. Spesso si può ricorrere a semplici *queries* per trovare la risposta che si sta cercando. Allo stesso modo in cui i dati si analizzano quando sono centralizzati, è possibile effettuare tali interrogazioni anche ai dataset locali dei nodi.

Differential Privacy È, in generale, una tecnica di tutela della privacy usato quando si analizzano e si condividono dati statistici. La privacy viene massimizzata aggiungendo rumore statistico agli update del modello che vengono passati tra server e client, assicurando che le informazioni degli individui non possano essere distinguibili e che le persone non siano identificabili. Questa tecnica può essere vista come una misurazione quantitativa e quantificabile di tutela della privacy.

2.11.4 Tipologie

Esistono vari tipi di apprendimento federato, che si possono classificare in due macroaree: **incentrati sul modello** e **incentrati sui dati**.

Incentrati sul modello È la categoria più comune, e raccoglie tutti quegli approcci che hanno lo scopo di costruire un modello centrale. I tipi appartenenti a questo gruppo di distinguono in **omogenei** ed **eterogenei**.

I modelli **omogenei** lavorano su dati organizzati in modo **orizzontale**, nel senso che si basano sulle stesse features. Significa che i vari record fanno riferimento a un set di caratteristiche consistente tra tutti i dataset locali.

I modelli **eterogenei** invece operano su dati **verticali**: si intende che i dataset condividono lo stesso spazio di istanze, ma descritto da features diverse.

Incentrati sui dati I modelli **incentrati sui dati** sono un paradigma recente di progettazione. In breve, l'idea di base è che i possessori dei dati permettano a organizzazioni esterne di accedervi per creare i loro modelli, senza condividere questi dati.

Inoltre, i modelli si suddividono in altre due categorie, dipendentemente dal modo di comunicazione dei nodi:

Decentralizzati I nodi si coordinano tra loro, senza la necessita di un punto centrale comune.

Centralizzati Un server centrale orchestra e gestisce i vari passi dell'algoritmo e coordina i nodi partecipanti. È responsabile della selezione dei nodi all'inizio del processo di training e per l'aggregazione dei risultati. Questa versione è la più semplice ed è quella che viene approfondita e implementata in questo lavoro di tesi.

2.11.5 Vantaggi

Come visto precedentemente, in un classico algoritmo di Machine Learning:

1. Si parte da un modello e da un dataset
2. Si addestra il modello usando i dati per eseguire un'azione utile, come una classificazione

Questo è un *iter* ideale nella teoria: nella pratica però i dati di training non vengono generati dalla macchina che effettivamente allena il modello; bensì, vengono creati da una moltitudine di smartphone, sensori, dispositivi di domotica... per poi essere inviati a un server centrale per usare effettivamente gli algoritmi di Machine Learning.

Questo approccio, che chiamiamo "classico" oppure "centralizzato", può presentare una serie di problematiche, dipendentemente dal contesto in cui ci si trova:

- **Regolamentazioni**: le leggi riguardanti la collezione e la protezione dei dati non sono uguali in tutto il mondo e, da zona a zona, cambia l'autorità che le controlla. Molte di queste non permettono ai dati sensibili di essere spostati e in casi specifici impediscono di unire i dati provenienti da diverse parti del mondo perché, appunto, gestite da autorità e legislazioni diverse.

- **Preferenze dei singoli utenti**: in alcuni casi, l'utente (colui che effettivamente genera i dati) non si aspetta o non vuole che i propri dati lascino il proprio dispositivo, o che vengano usati per addestrare modelli di IA.

- **Volume dei dati**: alcuni sensori producono dati grezzi di grandi dimensioni e con una altissima frequenza (si pensi alle videocamere di sorveglianza): non è conveniente immagazzinare tutti i dati in un punto centralizzato, per ragioni di velocità di trasferimento e di spazio fisico di archiviazione, considerando anche il fatto che buona parte di questi non risulterà utile al fine dell'addestramento.

Un'applicazione basata su apprendimento federato permette di evitare i cosiddetti *membership inference attacks*, cioè quelli che si verificano quando un attaccante cerca di determinare se un particolare record, istanza o tabella è stato utilizzato per addestrare un modello,

Inoltre, l'utilizzo di questi strumenti permette di venire incontro a quelle che sono le condizioni del GDRP[14]. Facciamo riferimento in particolare all'articolo 5.1.(c), con il principio della *minimizzazione dei dati*. Questo richiede che i dati raccolti siano *"adeguati, pertinenti e limitati a quanto necessario rispetto alle finalità per le quali sono trattati"*.

Tramite apprendimento federato inoltre è facile andare incontro all'articolo 5.1.(b) (*limitazione delle finalità*) cioè che i dati personali sono "*raccolti per finalità determinate, esplicite e legittime, e successivamente trattati in modo che non sia incompatibile con tali finalità [...].*"

Decentralizzando i dati infatti si limita il rischio per il quale i dati vengano riutilizzati per fini esterni a quelli originali.

Combinando le considerazioni appena fatte con le conclusioni sulla privacy e tutela dei dati alle quali siamo arrivati precedentemente, possiamo elencare alcuni esempi di contesti in cui l'approccio classico non è la soluzione ottimale:

- Dati sensibili, tra cui immagini e video, riguardanti la sanità e l'assistenza sanitaria;
- Informazioni finanziarie provenienti da diverse organizzazioni per la rilevazione di frodi;
- Geolocalizzazione di veicoli elettrici per ottimizzarne l'autonomia oppure dati riguardanti l'utilizzo energetico di una abitazione (si pensi a come questi dati potrebbero essere utili per i motivi sbagliati se dovessero finire nelle mani di malintenzionati, ad esempio rivelando quando una persona è a casa o meno);
- Messaggi crittografati per addestrare modelli di autocompletamento.

Il paradigma federato ha lo scopo di risolvere queste situazioni e lo fa "invertendo" l'*iter* di quello classico: se il Machine Learning centralizzato sposta i dati verso la computazione, il Federated Learning *sposta la computazione verso i dati*.

I passaggi di addestramento di un modello federato sono gli stessi dell'approccio classico, ma "scambiati" di posto. Anche gli strumenti e le entità in gioco sono le stesse: abbiamo un **server** centralizzato, una serie di **client** che vi fa riferimento (ad esempio i singoli dispositivi o le varie organizzazioni), un insieme di **dati** (ogni client avrà il suo dataset, con i dati prodotti esclusivamente da sé stesso) e un **modello** da addestrare.

2.11.6 Limitazioni

Nonostante i vantaggi presentati in precedenza, il Federated Learning presenta una serie di limitazioni sia **tecniche**, sia di natura **statistica** sia a livello di **sicurezza**.

Limitazioni tecniche Dato che il training dei modelli viene effettuato sui nodi, va considerato il fatto che questi dispositivi non sono, nella maggior parte dei casi, progettati per svolgere questi compiti (si pensi ai sensori IoT). Inoltre, un'implementazione efficace richiede una comunicazione non costante ma molto frequente tra client e server, che porta a richiedere risorse non solo a livello di computazione e memoria, ma anche a livello di banda di rete.

Limitazioni statistiche Queste riguardano principalmente i dati, la loro forma e la loro distribuzione, sia tra i client che all'interno dei singoli dataset.

- Eterogeneità tra i dataset locali: ognuno può avere dei *bias* rispetto alla popolazione globale, che diventano difficili da individuare una volta arrivati al modello globale.
- Dimensione: i dataset locali possono essere poco omogenei a livello di quantità di dati.
- Difficoltà nel *pre-processing* dei dati a livello di nodo.
- Sbilanciamento dei dataset locali.

Queste criticità, che si traducono in performance peggiori rispetto all'approccio centralizzato in tema di accuratezza e precisione, possono essere riassunte nella caduta dell'ipotesi dei campioni **indipendenti e distribuiti in modo identico** *(iid)* tra i nodi locali.

Nella teoria della probabilità, una sequenza di variabili casuali è detta **indipendente e identicamente distribuita** se le variabili hanno tutte la stessa distribuzione di probabilità e sono tutte statisticamente indipendenti.

Si pensi a un algoritmo di Federated Learning con lo scopo di effettuare una classificazione: se i client presentano istanze di osservazione appartenenti a una certa classe ma non hanno mai "visto" *record* appartenenti a un'altra, che invece è più presente nel dataset di un altro client, la classificazione potrebbe risultare più complessa, e si dovrebbe dare particolare attenzione al metodo di aggregazione lato server.

Limitazioni di Sicurezza Se i modelli di apprendimento federato sono meno vulnerabili agli attacchi inferenziali, come descritto precedentemente, possono essere invece

esposti ad attacchi **attacchi di inferenza sulle proprietà** o *property inference attacks*[14]. Questi attacchi vengono eseguiti da nodi "malevoli" che partecipano al training, andando a intaccare gli update che vengono passati verso il server. Il loro obiettivo è dedurre caratteristiche e proprietà dai dati di training che, se sono uniche rispetto a un individuo o a un gruppo di essi, saranno facili da individuare. Nonostante questi attacchi siano teoricamente possibili, è stato dimostrato[15] che nella pratica hanno sempre avuto poco spazio e poco successo.

Capitolo 3

Progettazione

In questo capitolo vengono elencati e descritti i passaggi e le scelte di progettazione effettuate nell'elaborato. Una effettiva implementazione di questi passaggi si trova nel capitolo successivo.

3.1 Obiettivi

L'**obiettivo** di questo progetto consiste nel valutare le **prestazioni di classificazione** di alcuni algoritmi di apprendimento automatico su un dataset di Riconoscimento di Attività Umana, con particolare attenzione verso le differenze di prestazioni tra un algoritmo classico di **Machine** e **Deep Learning** e uno di **Federated Learning**.

Inoltre, vengono analizzate le **limitazioni** di tipo statistico di questo secondo approccio, in presenza di dataset bilanciati e non bilanciati. Quest'ultimo caso rappresenta una vulnerabilità degli algoritmi di Federated Learning parzialmente investigata in letteratura.

Ciò che ci si aspetta è un risultato leggermente peggiore in termini di prestazioni per il modello federato, in confronto a quello centralizzato, a favore della preservazione delle privacy.

3.2 Dataset e pre-processing

Il dataset selezionato si chiama *Human Activity Recognition Using Smartphones* [16] ed è costituito dalle misurazioni dei vari sensori degli smartphone di 30 individui mentre eseguivano attività quotidiane.

L'esperimento è stato condotto su un gruppo di 30 persone tra i 19 e i 48 anni, con ognuna di esse che stava eseguendo una di queste 6 attività:

- **WALKING** - Camminare
- **WALKING UPSTAIRS** - Percorrere una scala verso l'alto
- **WALKING DOWNSTAIRS** - Percorrere una scala verso il basso
- **SITTING** - Stare seduti
- **STANDING** - Stare in piedi
- **LAYING** - Stare sdraiati

Le misurazioni sono state effettuate tenendo uno smartphone alla vita e sfruttando il suo **accelerometro** e **giroscopio**.

Il dataset è composto da 10299 record.

Ogni record contiene la **accelerazione tri-assiale** misurata dall'accelerometro, la **accelerazione del corpo** stimata, le **velocità tri-assiale angolare** misurata dal giroscopio, un vettore di 561 features, la label dell'attività e un identificatore del soggetto.

Il dataset viene fornito già suddiviso in due parti in modo casuale, facendo sì che il 70% dei soggetti faccia parte del *training set* e il restante 30% componga il *test set*.

Una volta ottenuto il dataset, il *pre-processing* da effettuare è minimo: non ci sono *missing values* e i dati sono già normalizzati e delimitati nell'intervallo [-1,1].

Vengono però definite alcune funzioni di *data analysis* per calcolare il **bilanciamento** del dataset a livello di istanze per ogni classe.

3.3 Approccio Centralizzato

La prima parte del progetto consiste nel creare algoritmi di classificazione e valutare le prestazioni sul dataset di due modelli di Machine Learning (*Decision Trees* e *Random Forests*) e di uno di Deep Learning (una rete neurale *feedforward*) con due obiettivi:

1. **Valutare** le prestazioni di questi modelli, in modo da avere una base di confronto per i passaggi successivi;

2. **Identificare** il modello e la configurazione migliore, così da partire da esso per la costruzione dell'algoritmo federato.

Chiaramente, in tutti i modelli testati in questa sezione, i dati vengono gestiti come se fossero effettivamente centralizzati, quindi uniti in unico dataset.

3.3.1 Decision Trees

Il primo modello è un **albero decisionale**. Come descritto nelle sezioni precedenti, questi modelli hanno l'obiettivo di creare meccanismi di classificazione che predicono un output imparando da semplici regole decisionali dedotte dalle variabili osservate.

Hanno il vantaggio di essere modelli semplici e accessibili, anche grazie alla loro visualizzazione grafica. Inoltre sono adatti al contesto in quanto possono gestire sia dati numerici che categorici.

Le decisioni vengono prese basandosi sul concetto di **entropia**, che è la misura della casualità e del disordine dei dati. In questo contesto viene utilizzato per approssimare la "purezza" di un nodo. Un nodo **puro** contiene misurazioni di una sola classe: l'obiettivo dell'albero è quindi quello di minimizzare l'entropia e aumentare la purezza, in modo da arrivare a un **nodo foglia**.

La differenza di purezza tra un nodo padre e un nodo figlio, e quindi la quantità di informazioni ottenuta da una variabile osservandone un'altra, è chiamato ***information gain*** ed è il criterio per il quale viene deciso se una feature viene usata per lo *splitting* o meno.

Gli alberi decisionali sono però limitati dall'**overfitting**, che porta a una cattiva generalizzazione, dalla loro scarsa stabilità di risultati e dal costo computazionale, in quanto l'algoritmo che li gestisce segue un approccio ***greedy***.

3.3.2 Random Forests

Il secondo approccio testato è quello delle **Random Forests**. Una **foresta** è una collezione di alberi decisionali che risolve almeno in parte i problemi di questi ultimi, in quanto si usano più classificatori e si sceglie il risultato "a maggioranza". Inoltre, sfrutta alcune tecniche, spiegate nelle sezioni precedenti, per ottimizzare i risultati, tra cui **Bootstrapping**, **Feature Randomness** e **Bagging**.

3.3.3 Rete Feedforward

L'ultimo modello testato è una rete *deep*, in particolare una rete ***feedforward***. Queste sono ottimi strumenti di classificazione, anche con grandi quantità di dati e features, e sono composte da strati di **neuroni artificiali** che comunicano tra loro. Una descrizione più accurata si può trovare nella apposita sezione.

Hyper-parameters Tuning

Una volta creata e analizzati i risultati della rete, viene eseguita una fase di aggiustamento degli iper-parametri per ottimizzare le prestazioni.

Queste serve perché, eseguendo il training più volte sui dati di addestramento, è possibile che i risultati non rispecchino quelli del mondo reale. Una tecnica per non mancare in generalizzazione è la **Cross Validation**.

Cross Validation

È un tecnica usata per valutare le performance di un modello su dati non visti. Consiste nel dividere i dati disponibili in più **sottoinsiemi**, usando uno di questi come *validation set*, e gli altri come set di addestramento. Questo processo viene eseguito più volte, e per ognuna viene usato un *validation set* differente.

Alla fine del processo viene calcolata la media dei risultati di ciascuna fase di validazione per produrre una stima più affidabile delle prestazioni del modello.

Una volta misurate le prestazioni di tutti i modelli, si sceglie quello migliore e si passa alla fase successiva, cioè quella di **apprendimento federato**.

3.4 Approccio Federato

In questa sezione viene simulata una situazione in cui disponiamo di più dataset in più organizzazioni addestrando il modello individuato precedentemente su questi client utilizzando l'apprendimento federato.

Questa sezione inizia creando i vari **client**. Questo viene fatto suddividendo il dataset iniziale in varie parti, ognuna con la sua porzione di training, di testing e di validation. La suddivisione tra i vari client avviene in modo casuale, facendo sì che i vari dataset locali siano bilanciati tra loro sia a livello di distribuzione delle osservazioni tra le classi sia a livello di dimensione.

In una architettura di Federated Learning il **server** invia i parametri del modello globale ai **client**, che aggiorna quindi quelli del suo modello locale. A questo punto, allena il proprio modello sui propri dati, aggiorna i parametri del suo modello e li manda indietro al server.

È quindi necessario che il server e il client espongano dei metodi per **inviare** i parametri, **ricevere** i parametri ed effettuare la **valutazione** dei risultati.

Uno dei compiti principali del server è definire la **strategia** che verrà seguita durante il processo.

La **strategia** è quell'insieme di istruzioni che include le decisioni su come selezionare i dispositivi partecipanti, come gestire la comunicazione e l'aggiornamento dei modelli, come gestire la privacy e la sicurezza dei dati locali e come effettuare l'aggregazione dei pesi del modello centralmente.

La strategia comprende anche il metodo di scelta dei primi parametri, che può essere casuale oppure avvenire richiedendoli a un client.

Il server inoltre espone anche una funzione che permette, alla fine di ogni round di addestramento, di calcolare le misurazioni di accuratezza, precisione, recall e F1-score.

Un fattore importante è che la **valutazione** dei risultati può avvenire sia lato server che lato client.

- **Valutazione lato Server**: funziona come la classica valutazione in ambito Machine Learning. Se è presente un dataset nel server con l'unico scopo di effettuare valutazioni, è una buona soluzione perché permette di valutare il nuovo modello senza doverlo mandare indietro al client.

- **Valutazione lato Client**: è la soluzione più complessa, ma anche quella più potente. Non necessita infatti di un dataset centralizzato e permette di valutare il modello su un set di dati più significativo.

  Questa seconda soluzione però è anche più rischiosa: una volta iniziato il processo di valutazione, è possibile che il dataset cambi o si aggiorni, aggiungendo, modificando o eliminando dei record. Inoltre, il dataset potrebbe variare anche tra un round e l'altro. Questo potrebbe portare a risultati poco stabili.

La prima analisi viene fatta mantenendo il numero di client fisso e pari a 10; nella fase successiva, si osservano i risultati al variare di questo numero.

3.4.1 Sbilanciamento del dataset

L'ultimo passaggio consiste nel simulare quelle che potrebbero essere le limitazioni statistiche descritte nella sezione precedente nel modello federato. È difficile infatti che i dati del mondo reale siano equamente distribuiti tra le classi osservate e che ogni client abbia la stessa quantità di dati: viene quindi definito un nuovo metodo per suddividere le osservazioni tra le classi, in modo da simulare questo comportamento. Viene inoltre definito un indice per osservare i risultati al variare di questo "indice di sbilanciamento".

Anche in questo caso, si effettuano varie prove e si osservano i risultati.

Alla fine di tutti i passaggi, vengono confrontati i risultati sulla precisione e l'affidabilità del modello, confrontando i vari metodi di apprendimento centralizzato con quello federato e valutando quanto effettivamente le limitazioni sulla distribuzione dei dati possano influenzare i risultati di quest'ultimo.

Capitolo 4

Strumenti

In questo capitolo vengono descritti gli strumenti, linguaggi, *framework* e librerie utilizzati nella stesura di questo progetto di Tesi.

4.1 Python

Python è un linguaggio di programmazione ad alto livello, tipato dinamicamente e orientato agli oggetti. Gli ambiti di applicazione sono plurimi: sviluppo di siti o applicazioni Web e desktop, realizzazione di interfacce grafiche, amministrazione di sistema, calcolo scientifico e numerico, database, giochi, grafica 3D e, come nel nostro caso Data Science e Machine Learning.

Python viene scelto in quanto presenta una serie di libreria costruite *ad hoc* per l'analisi dei dati e l'apprendimento automatico, tra cui *Pandas*, *TensorFLow* e *NumPy*.

4.1.1 Google Colab

Google Colab, o "*Colaboratory*" è un editor che permette di scrivere ed eseguire codice Python tramite browser.

Un documento Colab si chiama **blocco note**. I blocchi note Colab permettono di combinare codice eseguibile e RTF in un unico documento, insieme a immagini, HTML, LaTeX e altro ancora.

Un blocco note è un ambiente interattivo composto da una serie di **celle di codice**. Ogni cella è eseguibile e le variabili, che siano oggetti o funzioni, definite in ognuna, possono essere usate nelle altre.

Tramite Colab è semplice sfruttare tutte le librerie necessarie per importare, gestire, analizzare e visualizzare dati, così come quelle per la creazione di algoritmi di Machine Learning.

4.2 NumPy

NumPy è un pacchetto per il calcolo scientifico. Fornisce **array dinamico multi-dimensionale** e una serie di metodi per gestirli. Queste permette di manipolare l'array sotto l'aspetto matematico e logico ed effettuare operazioni algebriche e statistiche.

4.3 Pandas

Pandas è una libreria Python che fornisce strutture dati semplici, flessibili ed espressive per lavorare con i dati e i dataset. Raccoglie gli strumenti più importanti per effettuare analisi e manipolazione dei dati.

Uno degli oggetti più importanti tra quelli forniti da Pandas è `DataFrame`, cioè una struttura bidimensionale che crea un **dataset** a partire da una matrice. È un oggetto versatile perché espone i metodi per effettuare i primi controlli e le prime operazioni sui dati, come il controllo dei valori `null` o `NaN`, o i metodi per iterare sui dati.

Inoltre, è l'oggetto che viene dato in input ad alcuni modelli di apprendimento automatico, come vedremo successivamente.

4.4 Scikit-learn

Scikit-learn è libreria open-source per il Machine Learning che fornisce un set di strumenti semplici ed efficienti per l'analisi dei dati, tramite un ampio parco di algoritmi

e modelli.

4.5 TensorFlow

TensorFlow è un'altra libreria open-source sviluppata da Google per il Machine Learning e il Deep Learning, che fornisce un ambiente flessibile per la creazione e l'addestramento di modelli di apprendimento automatico più complessi e completi di quelli di *Scikit-learn*.

Uno dei punti chiavi di questa libreria è il **grafo dei tensori**: i modelli vengono rappresentati attraverso un grafo computazionale, nel quale i nodi rappresentano operazioni matematiche e gli archi rappresentano i tensori (strutture dati multi-dimensionali simili ad array).

4.6 PyTorch

PyTorch è un *framework* open-source di Machine Learning sviluppato principalmente da Facebook. La sua caratteristica principale estende il grafo dei tensori descritto precedentemente. Qui infatti si parla di **grafo computazionale dinamico**: questo viene costruito e modificato dinamicamente durante l'esecuzione dell'algoritmo e, quindi, durante l'apprendimento.

PyTorch inoltre fornisce una libreria molto ampia e dettagliata di reti neurali, adatte a ogni contesto, con moduli predefiniti per strati, funzioni di attivazione, ottimizzatori e criteri di perdita.

4.7 Flower

Flower è un framework per costruire sistemi di Federated Learning[2]. Le funzioni che fornisce vengono descritte nel capitolo sulla **Implementazione**. Il design di Flower è basato su alcuni principi:

- Personalizzabile: Flower fornisce un'ampia gamma di configurazioni diverse a seconda delle esigenze di ogni singolo caso d'uso, in quanto l'apprendimento federato è applicabile a innumerevoli campi.
- Estendibile: nato da un progetto di ricerca presso l'Università di Oxford, Flower è stato costruito pensando alla ricerca sull'intelligenza artificiale. Molti componenti possono essere estesi e sostituiti per creare nuovi sistemi all'avanguardia.
- Scalabile: Flower è direttamente applicabile a contesti reali, anche con decine di migliaia di client.
- Indipendente dal contesto: Flower può essere usato insieme a tutti i framework di Machine Learning disponibili per Python, come PyTorch e TensorFlow.
- Comprensibile: Flower è scritto pensando alla manutenibilità, con la community è incoraggiata a leggere e contribuire al codice.

Capitolo 5

Implementazione

Questo capitolo descrive nel dettaglio i vari step seguiti nella creazione dei modelli, nel loro addestramento e nella loro valutazione, compresi gli strumenti utilizzati e le scelte tecniche.

Vengono analizzate in particolare le porzioni di codice Python e le librerie utilizzate, con l'obiettivo finale di creare una architettura di apprendimento federato e confrontarne i risultati con quelli di una di apprendimento classico.

5.1 Pre-processing dei dati

Una volta caricato il dataset **Human Activity Recognition Using Smartphones** nel proprio *Google Drive*, è semplice importarlo nel blocco note di *Colab* e inserirlo in una struttura dati.

I file del dataset utilizzati sono X_train.txt, y_train.txt, X_test.txt e y_test.txt.

Questi file contengono i record del dataset, suddivisi in righe. X_train.txt e X_test.txt contengono i valori delle **features**, mentre y_train.txt e y_test.txt contengono la **variabile di output** di quella osservazioni, quindi la classe a cui appartiene.

Il contenuto di ogni file viene inserito in una struttura dati, per poi essere trasformato in un DataFrame di Pandas.

```
import pandas as pd

# train set
x_train = []
with open(X_train_file, 'r') as file:
    for line in file:
        data = line.strip().split()
        x_train.append(data)

train_df = pd.DataFrame(x_train)
```

Una volta eliminati i valori mancanti:

```
train_df = train_df.dropna()
test_df = test_df.dropna()
```

vediamo come i dati si dividono tra *training set* e *test set*:

```
Training Features Shape: (7352, 561)
Training Labels Shape: (7352,)
Testing Features Shape: (2947, 561)
Testing Labels Shape: (2947,)
```

5.2 Approccio Centralizzato

Una volta raccolti i dati, si inizia la creazione del primo modello di apprendimento, cioè un **Albero Decisionale**.

5.2.1 Classificazione tramite Decision Trees

Per creare un albero decisionale, viene scelto il modello fornito da *scikit-learn*.

L'inizializzazione è semplice: basta selezionare il criterio di suddivisione. In questo caso, viene scelta l'**entropia** al posto del criterio di **Gini** o della **log_loss**, dato che l'obiettivo è massimizzare l'informazione nei nodi dell'albero. Anche gli altri criteri sono stati testati, portando tuttavia a risultati simili.

Viene poi eseguita la *build* del classificatore fornendo in input i valori e le labels del set di addestramento.

```
# import the model
from sklearn import tree

# create the tree
clf = tree.DecisionTreeClassifier(criterion='entropy')

# build the tree
clf = clf.fit(x_train, y_train)
```

Successivamente, viene chiamata la funzione per effettuare le predizioni sul *test set*, da cui poi calcolare la **accuratezza** del classificatore confrontando i risultati della predizione con i dati di output effettivi.

```
# predictions array
predictions = clf.predict(x_test)

# get the accuracy
accuracy = accuracy_score(y_test, prediciotns)
```

```
Model accuracy score: 0.8446
```

L'albero decisionale arriva a una accuratezza dell'**84.46%**.

5.2.2 Classificazione tramite Random Forests

Il secondo modello testato è una **Random Forest**. Il processo è molto simile a quello del punto precedente, utilizzando come modello RandomForestClassifier fornito da *scikit-learn*.

```
# import the model
from sklearn.ensemble import RandomForestClassifier

# create the forest
rf = RandomForestClassifier(n_estimators=1000)
```

Il parametro fornito in input indica il numero di **classificatori** (gli alberi decisionali) usato.

Anche in questo caso, si costruisce il modello per allenarlo sui dati di training e si crea un vettore di **predizioni** per controllare le sue prestazioni sul test set. Una volta ottenuto questo array, è possibile calcolare l'accuratezza della foresta.

```
# train the model on training data
rf.fit(train_df, train_labels)

# predictions array
predictions = rf.predict(test_df)

# get the accuracy
accuracy = accuracy_score(y_test, predictions)
```

```
Model accuracy score: 0.9274
```

La **Random Forest** arriva a una accuratezza del **92.74%**, un valore più alto rispetto a quello del semplice albero.

5.2.3 Classificazione tramite Rete Neurale

L'ultimo modello centralizzato è un modello *deep* e si tratta di una **rete neurale feedforward**. Lo strumento utilizzato è il MLPClassifier fornito da scikit –learn. Il termine sta per *Multi-layer Perceptron classifier* e rappresenta, appunto, una rete neurale su più livelli. La prima versione del modello viene implementata scegliendo i parametri in modo casuale:

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import mean_squared_error, mean_absolute_error

myNet = MLPClassifier(max_iter=100, activation="relu",
                                    hidden_layer_sizes = (100, 100))
```

I parametri sono:

- **max_iter=100**: Questo parametro imposta il numero massimo di iterazioni durante l'ottimizzazione. In questo caso, l'addestramento della rete neurale si interromperà dopo 100 iterazioni.
- **activation="relu"**: Questo parametro specifica la funzione di attivazione da utilizzare nei nodi della rete neurale. In questo caso, è impostata su "relu" che sta per *Rectified Linear Unit*, una funzione di attivazione comune che introduce non-linearità alla rete.
- **hidden_layer_sizes=(100, 100)**: Questo parametro imposta la struttura della rete neurale specificando il numero di nodi in ciascuno degli strati nascosti. In questo caso, ci sono due strati nascosti, ognuno con 100 nodi.

I passaggi successivi sono medesimi a quelli dei modelli precedenti:

```
# fit
myNet.fit(train_df, train_labels)

# predict
predictions = myNet.predict(test_df)

# accuracy
accuracy = accuracy_score(test_labels, predictions)
```

```
Model accuracy score: 0.9477
```

Il risultato di accuratezza è del **94.77%**.

È possibile cercare di migliorare le prestazioni di questo classificatore effettuando **hyper-parameters tuning**. Come esplicato precedentemente, l'ottimizzazione degli iper-parametri si ottiene valutando il modello sempre più volte sul set di addestramento, il che può portare al problema dell'overfitting. Una tecnica per evitare questo inconveniente è chiamata Cross-Validation; in particolare, qui viene usata la sua versione **K-Fold**.

Con questo meccanismo, suddividiamo ulteriormente il set di addestramento in K sottoinsiemi, chiamati *fold*. Iterativamente addestriamo il modello su *K-1* fold, considerando quello non utilizzato come il set di **validazione**.

Per l'ottimizzazione degli iper-parametri, eseguiamo molte iterazioni dell'intero processo di K-Fold CV, utilizzando ogni volta **diverse impostazioni del modello**. Confrontiamo quindi tutti i modelli, selezioniamo il migliore, lo addestriamo sull'intero set di addestramento e poi lo valutiamo sul set di testing.

Questo processo si può automatizzare grazie alla funzione GridSearchCV fornita da scikit –learn. La funzione esegue una ricerca esaustiva su una griglia predefinita di valori dei parametri forniti, testando tutte le possibili combinazioni di parametri.

```
# Hyper-parameters grid

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100, 100, 100)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}

from sklearn.model_selection import GridSearchCV

# use CV = 3 for cross-validation -> three iterations, with 3 folds,
                                    each iteration uses two training
                                    folds and the other as validation
                                    fold

clf = GridSearchCV(myNet, parameter_space, n_jobs=-1, cv=3)
clf.fit(train_df, train_labels)
```

Una volta trovata la migliore configurazione possibile, si possono effettuare di nuovo le misurazioni dell'accuratezza e si arriva a un risultato del 95.22%, il più alto. Abbiamo quindi individuato il migliore modello, che si riassume così:

```
{'activation': 'tanh',
 'alpha': 0.05,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'adaptive',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
```

5.3 Approccio Federato

In questa sezione viene creato l'architettura di **apprendimento federato** per utilizzare il modello di **rete neurale** individuato precedentemente. La struttura viene costruita sulla base del framework **Flower**.

Il primo passo consiste nel definire una funzione che permetta di suddividere il dataset tra i vari *client*. Questa funzione prende il nome di load_datasets e prende il input il **numero di partizioni** nelle quali si vuole frazionare il dataset.

```
def load_datasets(NUM_CLIENTS):

    partition_size = len(train_dataset) // NUM_CLIENTS
    lengths = [partition_size] * NUM_CLIENTS
    # Randomly split a dataset into non-overlapping new datasets of
                                        given lengths.
    datasets = random_split(train_dataset, lengths, torch.Generator().
                                        manual_seed(42))

    test_partition_size = len(test_dataset) // NUM_CLIENTS
    test_lengths = [test_partition_size] * NUM_CLIENTS
    test_datasets = random_split(test_dataset, test_lengths, torch.
                                        Generator().manual_seed(42))

    val_partition_size = len(val_dataset) // NUM_CLIENTS
    val_lengths = [val_partition_size] * NUM_CLIENTS
    val_datasets = random_split(val_dataset, val_lengths, torch.
                                        Generator().manual_seed(42))

    # Split each partition into train/val/test and create DataLoader
    trainloaders = []
    valloaders = []
    testloaders = []
    for ds in datasets:
        trainloaders.append(DataLoader(ds, batch_size=BATCH_SIZE,
                                            shuffle=True))

    for ds in test_datasets:
        testloaders.append(DataLoader(ds, batch_size=BATCH_SIZE,
                                            shuffle=True))

    for ds in val_datasets:
        valloaders.append(DataLoader(ds, batch_size=BATCH_SIZE, shuffle
                                            =True))

    return trainloaders, valloaders, testloaders
```

Questa funzione viene poi richiamata per creare liste di oggetti *loader*. Un loader è un oggetto specifico per iterare sulle righe di un dataset. La lista trainloaders conterrà un numero di *loader* pari al numero di client. Come prima prova, si è deciso di utilizzare 10 client.

```
NUM_CLIENTS = 10
trainloaders, valloaders, testloaders = load_datasets(NUM_CLIENTS);
```

A questo punto, abbiamo simulato la divisione dei dati tra i vari nodi.

Dato che ogni client ha il compito di addestrare e valutare il proprio modello locale, necessita di due funzioni: una di **training** e una di **testing**.

```
def train(net, trainloader, epochs: int, verbose=False, device='cuda'):
    """Train the network on the training set."""
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(net.parameters())
    net.to(device)
    net.train()

    for epoch in range(epochs):
        correct, total, epoch_loss = 0, 0, 0.0
        for record, labels in trainloader:
            record, labels = record.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = net(record)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            # Metrics
            epoch_loss += loss.item()
            total += labels.size(0)
            _, predicted = torch.max(outputs.data, 1)
            correct += (predicted == labels).sum().item()

        epoch_loss /= len(trainloader.dataset)
        epoch_acc = correct / total
```

Oltre ad allenare la rete, la funzione permette di visualizzare, a ogni **epoca**, l'**errore** della previsione.

La funzione di testing, invece, ritorna le varie metriche, cioè accuratezza, precisione, recall, ed F1-score:

```
def test(net, testloader, device='cuda'):
    """Evaluate the network on the entire test set."""
    criterion = nn.CrossEntropyLoss()
    correct, total, loss = 0, 0, 0.0
    all_labels, all_predictions = [], []
    net.to(device)
    net.eval()

    with torch.no_grad():
        for record, labels in testloader:
            record, labels = record.to(device), labels.to(device)
            outputs = net(record)
            loss += criterion(outputs, labels).item()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

            all_labels.extend(labels.cpu().numpy())
            all_predictions.extend(predicted.cpu().numpy())

    loss /= len(testloader.dataset)
    accuracy = correct / total

    # Additional metrics
    precision = precision_score(all_labels, all_predictions, average='
                                        weighted')
    recall = recall_score(all_labels, all_predictions, average='
                                        weighted')
    f1 = f1_score(all_labels, all_predictions, average='weighted')

    return loss, accuracy, precision, recall, f1
```

Server-Side

Per effettuare la comunicazione **client-server**, è necessario che quest'ultimo esponga due metodi: uno per **ricevere** i parametri dal client e uno per **inviare** questi parametri, aggiornando quelli dei modelli locali.

```
# returns an array of the model's (the net given in input) parameters,
                                    that are in the state_dict schema
from collections import OrderedDict
def get_parameters(net) -> List[np.ndarray]:
    parameters = [val.cpu().numpy() for _, val in net.state_dict().
                                        items()]
    return parameters

# creates a dictionary to update the net params
def set_parameters(net, parameters: List[np.ndarray]):
    params_dict = zip(net.state_dict().keys(), parameters)
    state_dict = OrderedDict({k: torch.Tensor(v) for k, v in
                                        params_dict})
    net.load_state_dict(state_dict, strict=True)
```

Queste operazioni sono fattibili grazie al dizionario state_dict, una struttura dati che contiene i parametri del modello che lo richiama.

Client-Side

Dal lato delle organizzazioni, invece, è necessario creare degli oggetti che rappresentino i vari client durante la simulazione. **Flower** fornisce una classe che, estesa, permette di eseguire questo compito.

Per creare un client, è necessario implementare tre metodi: uno per ritornare i propri parametri, uno che riceva i parametri, aggiorni il modello e faccia partire il training locale per poi inviare indietro i parametri aggiornati e uno per valutare i parametri sui propri dati locali.

Ogni client è caratterizzato da un **client id** univoco (cid).

```
class FlowerClient(fl.client.NumPyClient):
    def __init__(self, cid, net, trainloader, valloader):
        self.cid = cid
        self.net = net
        self.trainloader = trainloader
        self.valloader = valloader

    def get_parameters(self, config):
        print(f"[Client {self.cid}] get_parameters")
        return get_parameters(self.net)

    def fit(self, parameters, config):
        print(f"[Client {self.cid}] fit, config: {config}")
        set_parameters(self.net, parameters)
        train(self.net, self.trainloader, epochs=1)
        return get_parameters(self.net), len(self.trainloader), {}

    def evaluate(self, parameters, config):
        print(f"[Client {self.cid}] evaluate, config: {config}")
        set_parameters(self.net, parameters)
        loss, accuracy, precision, recall, f1 = test(self.net, self.
                                                valloader)
        return float(loss), len(self.valloader), {"accuracy": float(
                                                accuracy), "loss": float(
                                                loss), "precision": float(
                                                precision), "recall":float(
                                                recall), "f1_score": float(
                                                f1)}
```

Virtual Client Engine

Considerando il fatto che stiamo simulando questa architettura, quindi il server e tutti i client, su una sola macchina, è facile incorre in problemi di memoria. Il **Virtual Client Engine** è un meccanismo fornito da **Flower** ed è un "motore" virtuale che crea i client *solo quando necessari*, cioè quando sono effettivamente utili per addestramento e valutazione.

Il **Virtual Client Engine** crea una FlowerClient quando necessita di un particolare nodo per richiamare una delle sue funzioni, grazie al suo identificatore.

I client virtuali vengono scartati e rimossi dalla memoria dopo l'uso, eliminando ogni stato.

```
# Virtual Client Engine: Create a Flower client representing a single
                                    organization.

def client_fn(cid: str) -> FlowerClient:

    # Note: each client gets a different trainloader/valloader, so each
                                        client
    # will train and evaluate on their own unique data
    trainloader = trainloaders[int(cid)]
    valloader = valloaders[int(cid)]

    # Create a single Flower client representing a single organization
    return FlowerClient(cid, net, trainloader, valloader)
```

Metriche

Prima di avviare la simulazione, definiamo una funzione che permetta di misurare le prestazioni del modello:

```
# Aggregate and return custom metric (weighted average)
def weighted_average(metrics: List[Tuple[int, Metrics]]) -> Metrics:
    accuracies = [num_examples * m["accuracy"] for num_examples, m in
                                        metrics]
    examples = [num_examples for num_examples, _ in metrics]

    weighted_average_metrics = {
        "Accuracy": sum(accuracies) / sum(examples),
        "Precision": sum(num_examples * m.get("precision", 0) for
                                            num_examples, m in metrics)
                                             / sum(examples),
        "Recall": sum(num_examples * m.get("recall", 0) for
                                            num_examples, m in metrics)
                                             / sum(examples),
```

```
        "F1-score": sum(num_examples * m.get("f1_score", 0) for
                                                num_examples, m in metrics)
                                                 / sum(examples),
    }

    return weighted_average_metrics
```

Simulazione

A questo punto è possibile iniziare la simulazione tramite la funzione flwr.simulation.start_simula che accetta alcuni argomenti in input:

- la funzione client_fn usata per creare le istanze del FlowerClient
- il numero di client da simulare
- il numero di round
- la **strategia**, che incapsula l'approccio selezionato

Si definisce quindi prima la strategia:

```
strategy = fl.server.strategy.FedAvg(
fraction_fit=1.0,  # Sample 100% of available clients for training
fraction_evaluate=0.5,  # Sample 50% of available clients for
                                    evaluation
min_fit_clients=NUM_CLIENTS,
min_evaluate_clients=NUM_CLIENTS,
min_available_clients=NUM_CLIENTS,
evaluate_metrics_aggregation_fn=weighted_average,  # <-- pass the
                                    metric aggregation function
)
```

Quindi si fa partire la simulazione:

```
fl.simulation.start_simulation(
    client_fn=client_fn,
    num_clients=NUM_CLIENTS,
    config=fl.server.ServerConfig(num_rounds=3),
    strategy=strategy,
```

```
    client_resources=client_resources,
)
```

Risultati

I risultati di questa simulazione sono riassunti qui:

Accuracy	0.8585
Precision	0.8703
Recall	0.8585
F1-score	0.8553

Tabella 5.1: Risultati della simulazione di Apprendimento Federato

Vediamo subito che i risultati sono peggiori rispetto a quelli della rete neurale nell'architettura centralizzata, ma rimandiamo i risultati e i confronti nella apposita sezione.

Un'altra analisi che viene fatta riguarda **i risultati al variare del numero di nodi**. Anche in questo caso, si eseguono più simulazioni e al variare del numero di client si riportano i risultati su un grafo.

5.3.1 Sbilanciamento del dataset

Fino a ora, i dataset tra i vari client sono sempre stati abbastanza bilanciati tra di loro. Questo viene verificato dalla funzione check_balancement.

```
from collections import Counter

def check_balancement(i, loader):
    print("\nDataframe: {i}, Size: {size}".format(i=i, size=len(loader.
                                        dataset)))

    all_classes = [0, 1, 2, 3, 4, 5]

    etichette = []
    for batch in loader:
```

```
        etichette.extend(batch[1].numpy())

    counter = Counter(etichette)

    fractions = []
    for className in all_classes:
        count = counter[className] if className in counter else 0
        fraction = count / len(loader.dataset)
        fractions.append(fraction * 100)

        print("Class: {className}, Occurrences: {count}, Percentage: {
                                                percentage:.2%}".format(
            className=className, count=count, percentage=fraction))

    standard_deviation = np.std(fractions)
    balance = 1 - standard_deviation / 100.0
    print("Balancement: {balance:.2%}".format(balance=balance))
    counter.clear()
    fractions = []
```

Questa funzione, applicata ai vari *DataLoaders* creati, torna per ognuno un livello di bilanciamento di circa **0.95**. Questo significa che le varie organizzazione si suddividono equamente le osservazioni delle varie classi.

Questo, però, è un comportamento raramente osservabile nella realtà: è infatti più probabile che alcune organizzazioni creino più frequentemente dati appartenenti a una classe, mentre potrebbero non trovarsi mai o di rado a crearne altre.

Per simulare questo comportamento, definiamo una funzione generate_high_std_weights. Questo metodo sfrutta la **distribuzione di Pareto** per generare un vettore di numeri con una elevata deviazione standard tra loro. Questi numeri verranno poi usati come pesi per distribuire in modo più o meno equo le varie classi tra i dataset "locali".

```
def generate_high_std_weights(num_classes, alpha):
    weights = np.random.pareto(a=alpha, size=num_classes)

    normalized_weights = weights / np.sum(weights)
```

```
    return normalized_weights

class_weights = [generate_high_std_weights(NUM_CLASSES) for _ in range(
                                    NUM_CLIENTS)]

train_samplers = [WeightedRandomSampler(weights, len(ds), replacement=
                                    False) for weights, ds in zip(
                                    class_weights, train_datasets)]

trainloaders = [DataLoader(ds, batch_size=BATCH_SIZE, sampler=sampler)
                                    for ds, sampler in zip(datasets,
                                    train_samplers)]
```

La funzione prende in input un valore **alpha**, che definiamo **indice di sbilanciamento**, che permette di variare il livello di sbilanciamento, dove un valore prossimo allo 0 indica un dataset estremamente sbilanciato e un valore più vicino ad 1 porta a una distribuzione più equilibrata.

Vengono poi eseguite varie simulazioni, al variare dell'indice di sbilanciamento, e si riportano i risultati in un grafico.

Capitolo 6

Risultati

In questo capitolo vengono visualizzati, analizzati e confrontati i risultati ottenuti nei passaggi descritti nel capitolo precedente.

Per ogni modello sarà possibile visualizzare le metriche di **accuratezza**, **precisione**, **recall** ed **f1-score**, nonché **l'intervallo di confidenza** e la **matrice di confusione**.

Confronteremo i risultati tra questi modelli per trarne le conclusioni.

Infine, tramite una serie di istogrammi, visualizzeremo i risultati dell'approccio federato ottenuti al variare dell'**indice di sbilanciamento** e al variare del **numero di client**.

6.1 Approccio centralizzato

6.1.1 Decision Trees

Nella seguente tabella elenchiamo i risultati ottenuti addestrando e valutando il primo modello, cioè l'**albero decisionale**.

Metriche	
Accuracy	0.8554
Precision	0.8556
Recall	0.8554
F1-score	0.8546

Tabella 6.1: Risultati dell'albero decisionale

Vediamo l'intervallo di confidenza del 95% dell'accuratezza:

Intervallo di confidenza	
inf	0.8552
sup	0.8557

Tabella 6.2: Intervallo di confidenza dell'albero decisionale

I risultati evidenziano una performance abbastanza positiva, anche se non ottimale, del modello. Le metriche, infatti, tutte vicine all'85%, evidenziano come l'albero decisionale e la simulazione effettuata siano affidabili, risultato osservabile anche dall'intervallo di confidenza risultante molto limitato.

Infine visualizziamo la **matrice di confusione**, cioè i dati da cui sono state calcolate le metriche delle tabelle precedenti.

$$\begin{bmatrix} 459 & 24 & 13 & 0 & 0 & 0 \\ 74 & 353 & 44 & 0 & 0 & 0 \\ 22 & 50 & 348 & 0 & 0 & 0 \\ 0 & 2 & 0 & 379 & 110 & 0 \\ 0 & 0 & 0 & 87 & 445 & 0 \\ 0 & 0 & 0 & 0 & 0 & 537 \end{bmatrix}$$

Tabella 6.3: Matrice di confusione dell'albero decisionale

Possiamo notare come l'algoritmo presenti difficoltà nel classificare alcune osservazioni: c'è molta incertezza infatti sulle classi **3** e **4**, rispettivamente **SITTING** e **STANDING**, come si può notare dalla terza e quarta riga. L'ultima classe (**LAYING**) è la più precisa, con tutte le predizioni tra i "*true positives*".

6.1.2 Random Forest

Visualizziamo le stesse tabelle per la **Random Forest**, iniziando dalle metriche:

Metriche	
Accuracy	0.9226
Precision	0.9234
Recall	0.9226
F1-score	0.9225

Tabella 6.4: Risultati della random forest

Quindi l'intervallo di confidenza:

Intervallo di confidenza	
inf	0.9225
sup	0.9228

Tabella 6.5: Intervallo di confidenza della random forest

Possiamo notare come i risultati siano nettamente migliori rispetto ai precedenti, con metriche superiori al 92%. Inoltre, anche l'intervallo di confidenza risulta più stretto (pari a 3×10^{-4}, rispetto allo 5×10^{-4} dell'albero decisionale), dimostrando l'affidabilità della random forest.

Vediamo la matrice di confusione:

$$\begin{bmatrix} 475 & 10 & 11 & 0 & 0 & 0 \\ 36 & 427 & 8 & 0 & 0 & 0 \\ 17 & 44 & 359 & 0 & 0 & 0 \\ 0 & 2 & 0 & 436 & 55 & 0 \\ 0 & 0 & 0 & 47 & 485 & 0 \\ 0 & 0 & 0 & 0 & 0 & 537 \end{bmatrix}$$

Tabella 6.6: Matrice di confusione della random forest

Possiamo notare come le incertezze descritte precedentemente vengano ampiamente limitate, rendendo le misurazioni effettivamente più accurate rispetto a quelle dell'albero decisionale.

6.1.3 Rete Feedforward

Gli ultimi risultati che visualizziamo sono quelli relativi alla **rete neurale feedforward**.

Metriche	
Accuracy	0.9599
Precision	0.9619
Recall	0.9599
F1-score	0.9598

Tabella 6.7: Risultati della rete neurale

Vediamo l'intervallo di confidenza del 95% dell'accuratezza:

Intervallo di confidenza	
inf	0.9598
sup	0.9601

Tabella 6.8: Intervallo di confidenza della rete neurale

Osserviamo che le misurazioni sono ulteriormente migliorate, arrivando a un livello del 96%. L'intervallo di confidenza è uguale a quello della random forest, quindi pari a 3×10^{-4}, il che rende il risultato ottenuto estremamente preciso e affidabile.

$$\begin{bmatrix} 494 & 0 & 2 & 0 & 0 & 0 \\ 20 & 450 & 1 & 0 & 0 & 0 \\ 3 & 18 & 399 & 0 & 0 & 0 \\ 0 & 3 & 0 & 432 & 56 & 0 \\ 2 & 0 & 0 & 8 & 522 & 0 \\ 0 & 0 & 0 & 0 & 5 & 532 \end{bmatrix}$$

Tabella 6.9: Matrice di confusione della rete neurale

Dalla matrice vediamo che il rapporto delle predizioni corrette rispetto a quelle errate è molto alto, con l'incertezza più grande che rimane quella tra le classi **SITTING** e **STANDING**.

6.1.4 Confronto dei risultati

Vediamo i risultati appena elencati raggruppati in un istogramma:

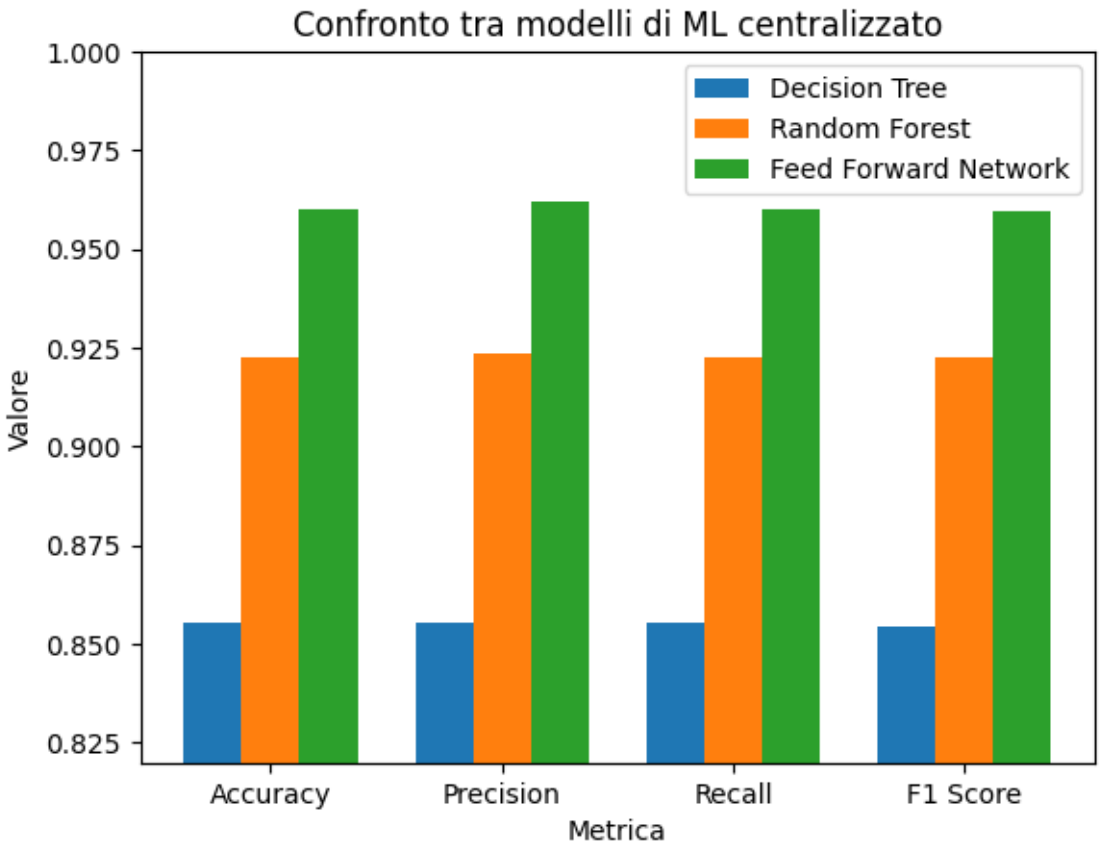

Figura 6.1: Confronto tra le performance dei modelli centralizzati

Possiamo vedere come il modello deep, cioè la **rete neurale**, abbia performance molto migliori sui dati centralizzati, arrivando a risultati anche di dieci punti percentuali in più del singolo albero decisionale.

Confermato quindi che la rete neurale feedforward sia il modello migliore, analizziamo i risultati dell'approccio federato.

6.2 Approccio federato

Guardiamo nel dettaglio le misurazioni, lasciando da parte la matrice di confusione ma analizzando nel dettaglio gli intervalli di confidenza delle metriche svolgendo l'analisi con **10 client**.

Metrica	Valore	CI-inf	CI-sup	CI-ampiezza
Accuratezza	0,8415	0,8267	0,8563	0,0296
Precisione	0,8604	0,8466	0,8741	0,0275
Recall	0,8415	0,8267	0,8563	0,0296
F1-score	0,8388	0,8233	0,8544	0,0311

Tabella 6.10: Metriche risultanti dalla simulazione di Federated Learningx

Notiamo come la media dei risultati sia intorno all'84%, un risultato paragonabile a quello degli alberi decisionali dell'approccio centralizzato. Vediamo però come l'intervallo di confidenza sia più ampio, intorno a 2×10^{-2}, quindi di due ordini di grandezza più grande. Questo sta a significare che i risultati sono in media buoni, ma con una affidabilità più scarsa.

Visualizziamo questo comportamento su un istogramma, evidenziando l'intervallo di confidenza:

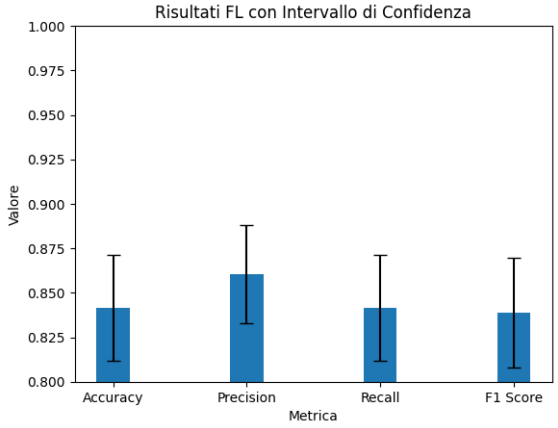

Figura 6.2: Performance del modello federato

6.2.1 Confronto con i modelli centralizzati

Vediamo come i risultati ottenuti dal modello federato si comportano rispetto a quelli precedenti:

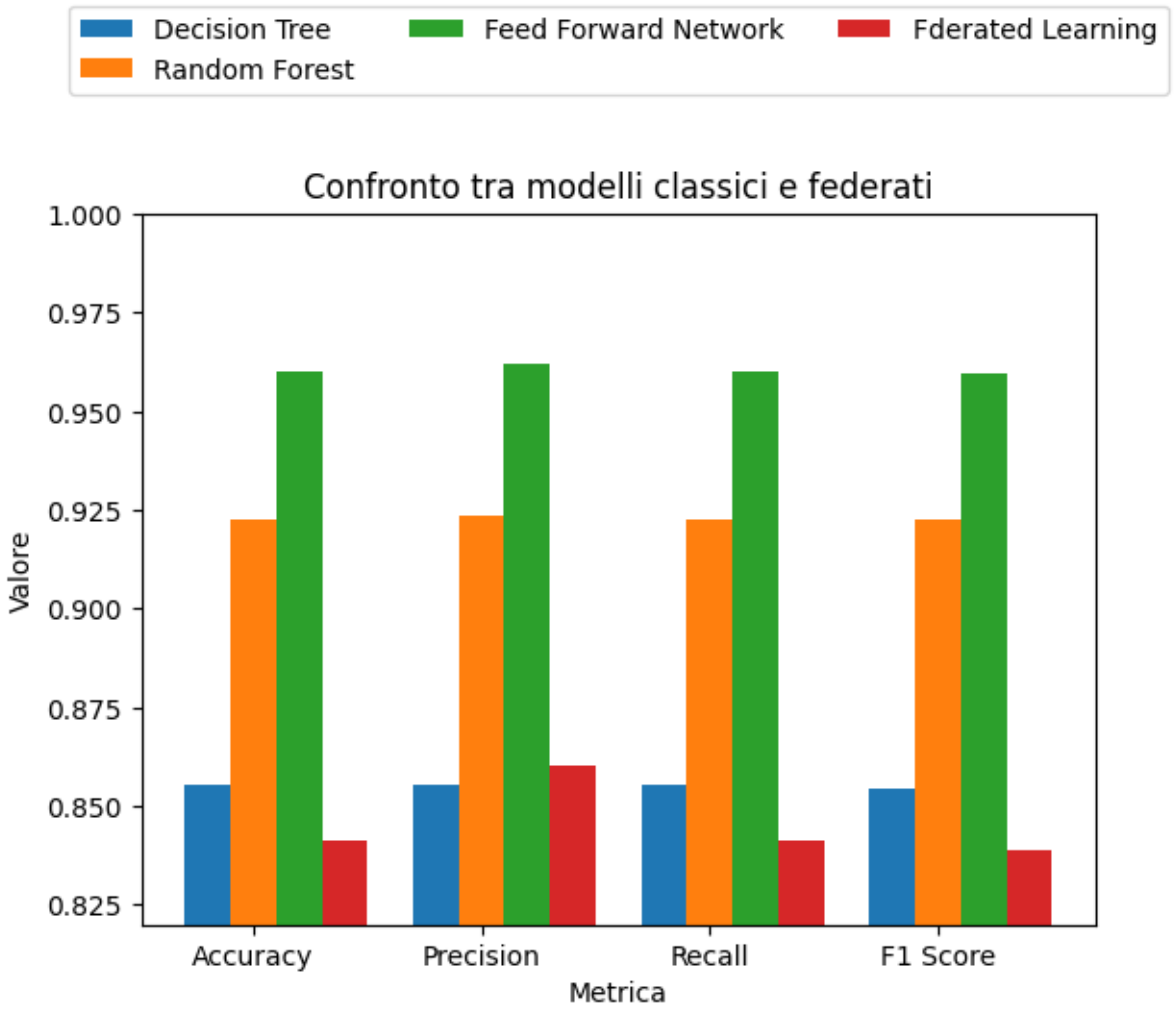

Figura 6.3: Confronto tra le performance dei modelli centralizzati e federato, con enfasi sulle differenze

Notiamo come le performance del modello nell'ambito federato siano più scarse, come ci si poteva aspettare, di quelle dello stesso modello nell'architettura centralizzata. Osservando il grafico, bisogna considerare il fatto che la scala dell'asse verticale parte dal valore di 0.8, per evidenziare le differenze tra i risultati. Questi però non sono scarsi in senso assoluto: si avvicinano molto, infatti a quelli dell'**albero decisionale**.

Se guardiamo lo stesso istogramma nella sua versione integrale, possiamo notare come le differenze non siano così accentuate:

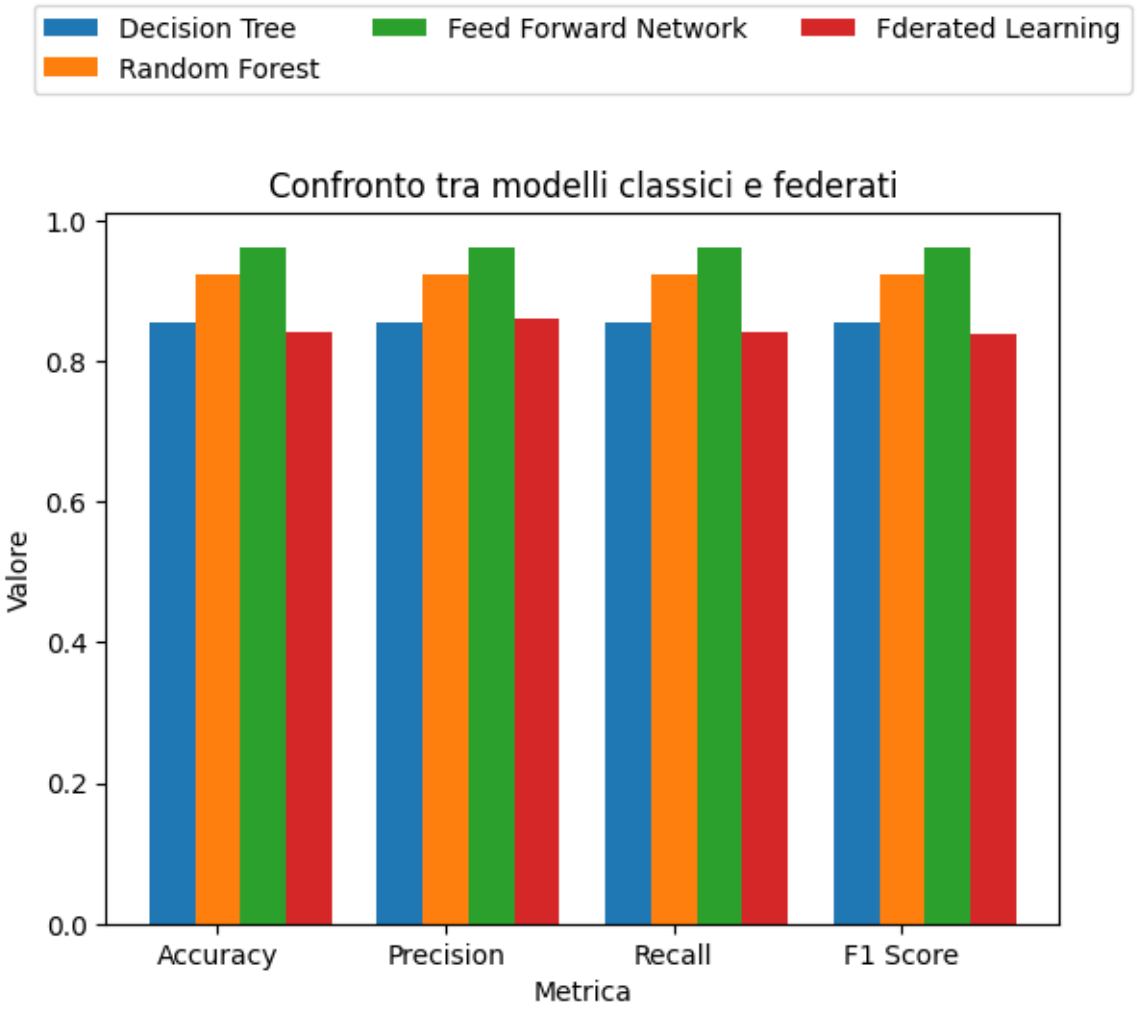

Figura 6.4: Confronto tra le performance dei modelli centralizzati e federato con la scala completa

6.2.2 Risultati al variare del numero di nodi

Fino a ora, le misurazioni sono state effettuate con il numero di client fisso a 10. Ovviamente, in un contesto reale, i nodi dovrebbero essere molti di più e il loro numero potrebbe anche variare nel tempo. Ogni client dovrebbe produrre circa la stessa quantità

di dati; se così non dovesse essere, il compito di "aggiustare" il peso di ognuno sarà in mano alla strategia di aggregazione lato server.

Possiamo però valutare, mantenendo sempre l'intero dataset, come si comporta l'algoritmo al variare del numero effettivo di client, quindi con un numero relativamente sempre più basso di osservazioni per ogni dataset locale.

Vediamo prima la tabella e successivamente i risultati riportati sul grafico.

num. client	**Accuratezza** **int. conf.**	**Precisione** **int. conf.**	**Recall** **int. conf.**	**F1-score** **int. conf.**
2	0,9204 0,0080	0,9160 0,0075	0,9109 0,0079	0,9113 0,0088
5	0,9075 0,0124	0,9131 0,0118	0,9075 0,0124	0,9059 0,0127
10	0,9109 0,0089	0,9161 0,0075	0,9109 0,0089	0,9113 0,0088
20	0,8158 0,0198	0,8310 0,0185	0,8158 0,0198	0,8140 0,0203
30	0,8007 0,0216	0,8422 0,0223	0,8001 0,0216	0,7947 0,0222
40	0,7694 0,0264	0,8091 0,0288	0,7694 0,0264	0,7484 0,0307
40	0,7694 0,0264	0,8091 0,0288	0,7694 0,0264	0,7484 0,0307
50	0,7697 0,0201	0,8107 0,0210	0,7696 0,0201	0,7441 0,0215

Tabella 6.11: Performance del modello federato al variare del numero di client

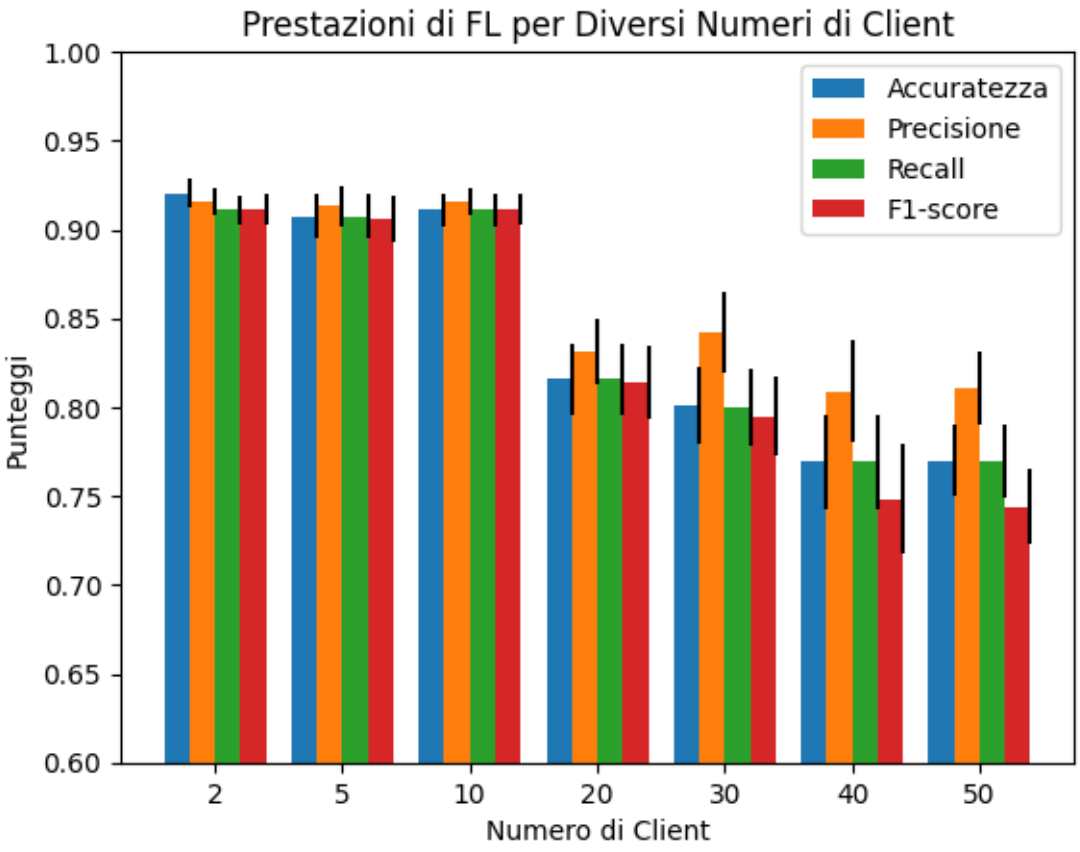

Figura 6.5: Confronto tra le performance del modello di FL al variare del numero di client

Come ci si poteva aspettare, le prestazioni del modello calano all'aumentare del numero di client, fenomeno visibile sia dal valore medio che si abbassa sia dall'ampiezza del intervallo di confidenza sempre più grande. Notiamo però che, anche con dataset notevolmente più piccoli, (con 50 client, i dataset sono grandi 0,04 volte quelli avendo a disposizione due client), le performance non sono così scarse, con una precisione dell'81%.

6.3 Dataset sbilanciati

L'ultimo risultato che guardiamo è quello riguardante i dataset sbilanciati a livello di distribuzioni delle classi tra i client.

È importante sottolineare che un livello dell'indice di sbilanciamento pari o vicino

ad 1 non corrisponda a un livello bilanciamento perfetto, ma vicino al 85-90% in media. D'altra parte, la suddivisione casuale delle osservazioni tra i vari dataset portava a un livello di bilanciamento vicino al 95%.

Visualizziamo nella tabella i valori medi di ogni metrica al variare dell'indice di sbilanciamento, cioè il valore alpha che viene dato in input alla funzione generate high std weights; negli invece vediamo anche l'intervallo di confidenza.

Indice di sbilanciamento	**Accuratezza**	**Precisione**	**Recall**	**F1-score**
0,1	0,6122	0,6791	0,6122	0,6429
0,2	0,5936	0.7429	0,5936	0,6152
0,3	0,6373	0,6550	0,6863	0,6373
0,4	0,7495	0,7483	0,7495	0,7714
0,5	0,7175	0,8283	0,7175	0,7451
0,6	0,7321	0,8451	0,7321	0,7482
0,7	0,8072	0,8563	0,8072	0,8007
0,8	0,8821	0,8759	0,8821	0,8747
0,9	0,9135	0,9017	0,9135	0,9366
1	0,8719	0,9015	0,8719	0,8753

Tabella 6.12: Metriche risultanti dalla simulazione di Federated Learning con dataset locali sbilanciati.

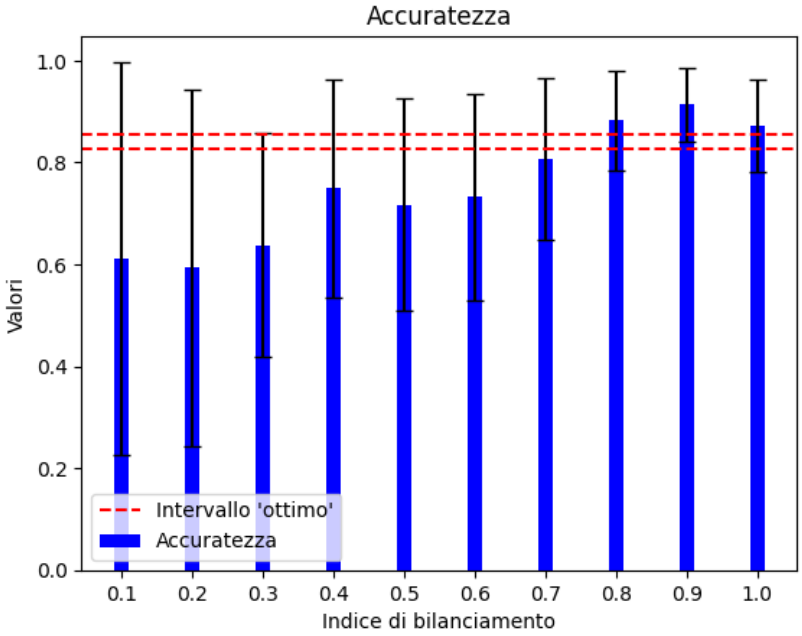

Figura 6.6: Performance del modello federato al variare del bilanciamento - Accuratezza

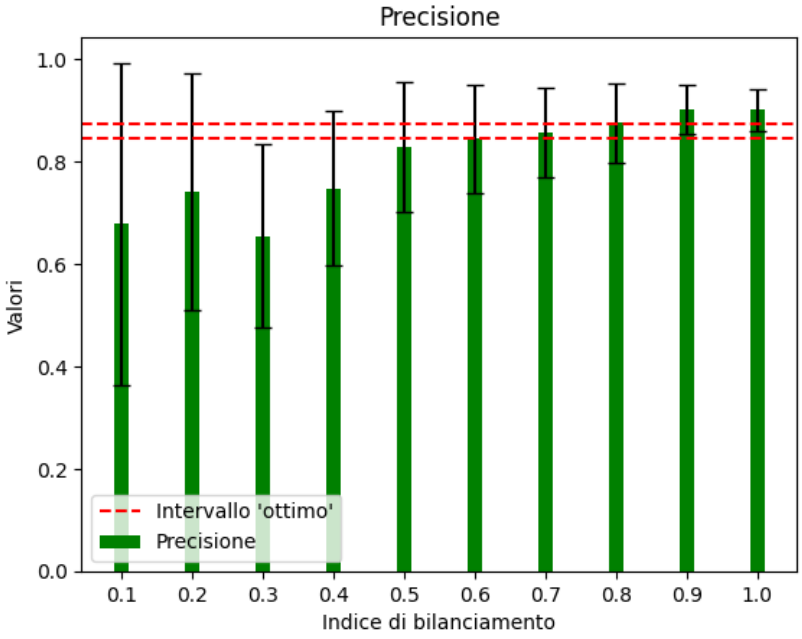

Figura 6.7: Performance del modello federato al variare del bilanciamento - Precisione

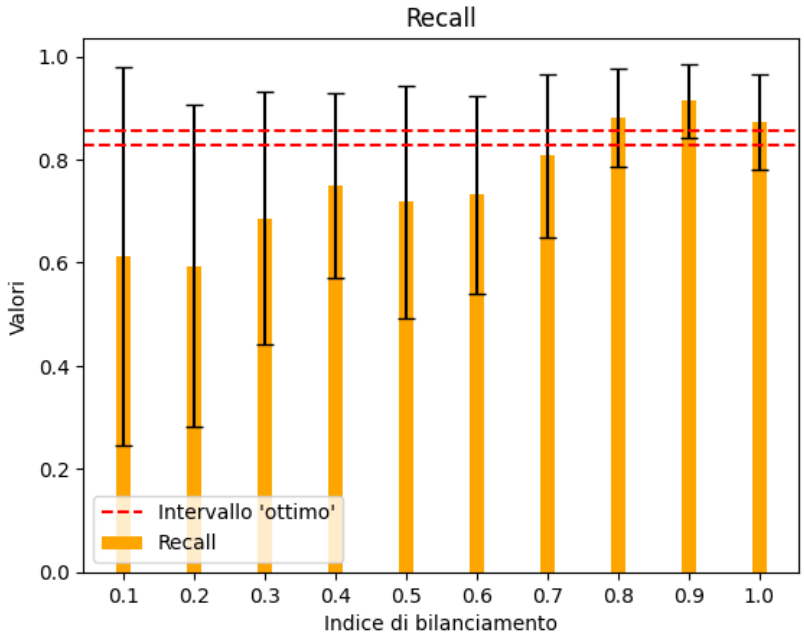

Figura 6.8: Performance del modello federato al variare del bilanciamento - Recall

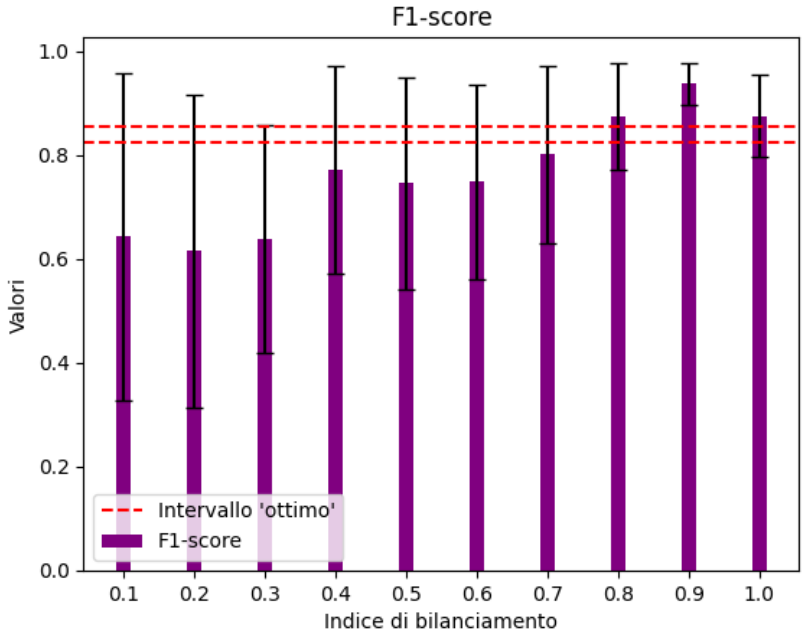

Figura 6.9: Performance del modello federato al variare del bilanciamento - F1-score

Possiamo notare come le performance del modello siano diverse al variare del fattore di sbilanciamento. Il fenomeno è lo stesso su tutte le metriche: quando il fattore è vicino a 0, la media delle misurazioni è scarsa, e il risultato si può vedere anche dall'ampio **intervallo di confidenza**.

Man mano che i dataset diventano più bilanciati, il valore delle misurazioni si alza e l'ampiezza dell'intervallo di confidenza diminuisce, rendendo le previsioni più precise e le misurazioni più accurate.

Notiamo infine il confronto con l'intervallo di confidenza trovato testando il modello sui dataset con un livello di bilanciamento "ottimo": possiamo vedere come, con un dataset leggermente sbilanciato, le performance siano addirittura leggermente migliori, a scapito di un intervallo di confidenza leggermente più ampio.

Capitolo 7

Conclusioni e Sviluppi Futuri

Il **Federated Learning** rappresenta un approccio contemporaneo motivato da principi fondamentali, i quali devono essere prioritariamente considerati da coloro che implementano sistemi di apprendimento automatico e raccolta dati. Questo vale soprattutto nel rispetto degli utenti finali, i quali, spesso inconsapevolmente, possono compromettere il proprio diritto alla **privacy** e alla sicurezza dei dati durante l'utilizzo di tali sistemi. La scelta di un dataset riguardante il riconoscimento dell'attività umana è strettamente correlata allo spazio di manovra di questo argomento.

Un classico algoritmo di Machine Learning o Deep Learning, basato sulla centralizzazione totale dei dati, presenta sicuramente un vantaggio importante a livello di affidabilità del modello, peccando in altri ambiti. Questi posso essere, appunto, la privacy dei dati utente finale oppure le richieste eccessive di potenza di calcolo o di storage.

Un approccio di tipo federato, invece, ha il vantaggio di preservare completamente la privacy e i dati sensibili dei propri utenti, Inoltre, permette di lavorare su dati più vari e significativi.

D'altra parte, però, presenta altre problematiche, cioè la sfida di utilizzare macchine con risorse limitate per l'addestramento e la valutazione di modelli di apprendimento automatico, e limitazioni relative alla distribuzione dei dati tra i vari dataset locali.

In questo progetto di Tesi ci siamo concentrati specificatamente sulle performance puramente "statistiche" di questo nuovo approccio.

Come possiamo vedere nella sezione dei **Risultati**, con il dataset a disposizione, le metriche di valutazione non si discostano più di tanto da quelle risultanti dai due approcci, con i risultati di apprendimento federato assimilabili a quelli di un **albero decisionale** centralizzato.

Il riscontro è diverso quando si parla di dati non *iid*: abbiamo visto come client con dataset sbilanciati portino a risultati peggiori durante la simulazione. Il risultato ottenuto è vicino a quanto ci si aspettava: se alcuni client presentano solo osservazioni relativi ad alcune classi, è difficile per loro valutare il risultato degli altri modelli e di conseguenza aggiornare i propri parametri.

Uno sbilanciamento leggero però non è per forza un fattore negativo: potrebbe infatti rappresentare più realisticamente i dati e la loro distribuzione tra le organizzazioni, permettendo al modello di imparare meglio. Inoltre, se la classe meno rappresentata è quella di maggiore interesse o rilevanza nel contesto, un dataset sbilanciato può essere utile. Ad esempio, nel riconoscimento di frodi in transazioni finanziarie, la classe delle transazioni fraudolente potrebbe essere di maggiore interesse rispetto a quelle legittime.

Una continuazione di questo progetto potrebbe comprendere la gestione di questi dataset sbilanciati a livello statistico, ad esempio utilizzando tecniche come il *campionamento* o la pesatura delle classi; oppure, modificando la **strategia** di aggregazione usata dal **server** di Federated Learning.

Un'altra sfida riguardante l'*Internet of Things* potrebbe essere l'analisi del consumo di risorse relativo ai dispositivi client che devono eseguire gli algoritmi di apprendimento, oppure un confronto tra i due approcci di apprendimento riguardante l'utilizzo di energia, memoria e archiviazione per la gestione degli algoritmi. Infine, si potrebbero anche analizzare tecniche ibride tra le due, come l'**edge computing**.

Dato che abbiamo visto che le performance di apprendimento federato non sono così distanti da quelle di apprendimento centralizzato, possiamo concludere che questo nuovo approccio è sicuramente una valida alternativa e, in futuro, con le dovute migliorie, potrebbe essere l'architettura ordinaria di tutti i sistemi di apprendimento automatico.

Riferimenti bibliografici

[1] K. Barefoot, D. Curtis, W. Jolliff, J. R. Nicholson, R. Omohundro, et al., *Defining and measuring the digital economy*, US Department of Commerce Bureau of Economic Analysis, Washington, DC, 15 (2018), p. 210.

[2] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, and N. D. Lane, *Flower: A friendly federated learning research framework*, 2022.

[3] L. Breiman, *Random forests*, Machine learning, 45 (2001), pp. 5–32.

[4] P. Christen and O. Del Fabbro, *Cybernetical concepts for cellular automaton and artificial neural network modelling and implementation*, in 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), IEEE, 2019, pp. 4124–4130.

[5] G. Cloud, *What is unsupervised learning?*

[6] I. B. M. Corporation, *Cosa sono le reti neurali?*

[7] S. Z. El Mestari, G. Lenzini, and H. Demirci, *Preserving data privacy in machine learning systems*, Computers Security, 137 (2024), p. 103605.

[8] gemmo.ai, *Federated learning: Predictive model without data sharing.*

[9] Google, *Human activity recognition (har): Fundamentals, models, datasets.*

[10] ——, *What is big data?*

[11] M. Islam and M. R. Islam, *Modeling Spammer Behavior: Artificial Neural Network vs. Nai?ve Bayesian Classifier*, 04 2011.

[12] M. Learning, *Tom mitchell*, Publisher: McGraw Hill, (1997).

[13] K. (Medium.com), *Federated learning: Advancing machine learning while protecting privacy.*

[14] M. (Medium.com), *Benefits and challenges of federated learning under the gdpr.*

[15] L. Melis, C. Song, E. D. Cristofaro, and V. Shmatikov, *Exploiting unintended feature leakage in collaborative learning*, 2018.

[16] A. D. G. A. O. L. Reyes-Ortiz, Jorge and X. Parra, *Human Activity Recognition Using Smartphones*. UCI Machine Learning Repository, 2012. DOI: https://doi.org/10.24432/C54S4K.

[17] L. Rokach and O. Maimon, *Decision Trees*, vol. 6, 01 2005, pp. 165–192.

[18] C. Sammut and G. I. Webb, eds., *Samuel's Checkers Player*, Springer US, Boston, MA, 2010, pp. 881–881.

[19] A. W. Services, *Che cos'è il data mining?*

[20] A. M. TURING, *I.—COMPUTING MACHINERY AND INTELLIGENCE*, Mind, LIX (1950), pp. 433–460.

Ringraziamenti

Credo che questa sia una sezione doverosa per chiudere al meglio un ciclo. Sarò breve, perché certe cose preferisco dirle a voce.

Ringrazio il professor Marco Di Felice, che mi ha seguito durante la stesura della Tesi, per avermi introdotto a un argomento così vicino ai miei interessi.

Voglio ringraziare la mia famiglia, in particolare ai miei genitori e mia sorella, che mi hanno sempre sostenuto, nonostante abbia raccontato poco a casa: probabilmente non avete ancora capito bene cosa ho studiato, ma ammetto sia colpa mia.

Un ringraziamento doveroso va a Nonna Lella: casa tua è stata la sede della scrittura di buona parte di questa Tesi, che senza di te non avrebbe visto luce. Grazie.

Un grazie ai miei amici "di vecchia data" e anche a quelli un po' più recenti: come da proverbio, siete pochi (per fortuna!), ma molto, molto buoni.

Grazie ai miei "colleghi": so quanto vi piace essere chiamati così! Non ho ancora chiaro cosa abbiamo in comune, ma so che avete reso questa esperienza infinite volte migliore.

Infine, un ringraziamento a Bologna. Sono certo di non averti vissuta al cento per cento, ma mi hai messo così tanta curiosità che non vedo l'ora di farlo. Grazie per aver reso magico questo viaggio.