



UNIVERSITÀ DEGLI STUDI DI MILANO - BICOCCA
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea Magistrale in Data Science

Modelli di Machine Learning federato e distribuito per la previsione del carico all'edge

Relatore: Dott. Michele Ciavotta

Tesi di Laurea Magistrale di:
Daniele Mingolla
Matricola 866167

Anno Accademico 2021-2022

Ringraziamenti

Vorrei innanzitutto ringraziare il mio relatore di tesi, il dottore Michele Ciavotta, per avermi accompagnato nel corso dello sviluppo della tesi. Un ringraziamento anche a tutti coloro i quali mi sono stati vicini anche a distanza. Un ultimo ringraziamento lo rivolgo a Rondo, Shiva, Mambo e Paky. Non è mai semplice trovare le parole adatte a esprimere la profondità dell'animo umano, ma voi ci siete riusciti.

Indice

Elenco delle tabelle	v
Elenco delle figure	vi
Acronimi	viii
1 Introduzione	1
1.1 Motivazione e Contesto	1
1.2 Obiettivi della Tesi	3
1.3 Domande di Ricerca	3
1.4 Contributo	4
1.5 Struttura dell'Elaborato di Tesi	4
2 Il Contesto	6
2.1 Cloud Computing	6
2.1.1 Caratteristiche Essenziali	6
2.2 Fog & Edge Computing	7
2.3 Dal Centralized Machine Learning al Federated Learning	8
2.3.1 Centralized Machine Learning	10
2.3.2 Distributed On-Site Learning	11
2.3.3 Federated Learning	11
2.4 Modelli predittivi	15
2.4.1 AutoRegressive Integrated Moving Average	15
2.4.2 Prophet	16
2.4.3 Long Short-Term Memory	16
2.4.4 Diffusion Convolutional Recurrent Neural Network	18
3 Stato dell'Arte	20
4 Assunzioni e Definizione del Problema	22
4.1 Definizione del problema	22
4.2 Generazione delle richieste per ogni nodo	24
4.2.1 Distribuzione di Poisson	24
4.2.2 Markov Chain	24

5 Approcci Proposti	27
5.1 Approcci federati e distribuiti: caratteristiche comuni	27
5.2 Algoritmo federato e distribuito a una fase	29
5.3 Algoritmo federato e distribuito a due fasi	31
6 Valutazione Sperimentale	34
6.1 Creazione del dataset	34
6.2 Esplorazione dei dati	37
6.3 Campagna sperimentale	39
6.3.1 Confronto tra i modelli nei diversi scenari	41
7 Conclusioni	49
7.1 Sviluppi futuri	49
Bibliografia	51

Elenco delle tabelle

6.1 Dataset contenente il numero di richieste per funzione di un nodo della rete 37

Elenco delle figure

2.1	Edge Computing VS Fog Computing [1]	8
2.2	Centralized VS Distributed VS Federated Learning [2]	10
2.3	Fasi principali del FL [3]	12
2.4	Illustrazione del framework FedAvg [4]	13
2.5	Componenti all'interno di una unità LSTM [5]	17
2.6	Architettura del modello DCRNN [6]	19
4.1	Esempio di grafo i cui nodi rappresentano le diverse celle telefoniche . . .	23
4.2	Esempio di catena di Markov [7]	25
5.1	La fase 0 viene eseguita un'unica volta da tutti i nodi e consiste nell'allenamento in locale del proprio modello senza alcuna interazione con i dispositivi nella rete. La fase 1 viene eseguita in sequenza da tutti i nodi fino a quando nessuno di essi migliora le proprie performance. Il nodo riceve i pesi dai propri vicini, gli aggrega e partendo da questi esegue il fine-tuning. Se le sue performance migliorano rispetto a quelle precedenti, memorizza i pesi in una lista locale e invia i nuovi ai nodi vicini.	30
5.2	La fase 0 viene eseguita un'unica volta da tutti i nodi e consiste nell'allenamento in locale del proprio modello senza alcuna interazione con i dispositivi nella rete. La fase 1 e la fase 2 vengono eseguite in sequenza da tutti i nodi fino a quando tutti smettono di migliorare le proprie performance. Tutti i nodi ricevono contemporaneamente i pesi dai propri vicini e ognuno di essi li memorizza all'interno di una lista temporanea. Successivamente e sempre contemporaneamente, nella fase 2 ogni nodo esegue il fine-tuning del proprio modello e aggiorna eventualmente la lista dei pesi che gli hanno garantito le migliori performance.	32
6.1	Moving average e standard deviation relative alle funzioni dell'antenna numero 0 calcolate su una finestra temporale di 2 ore.	38
6.2	Esempio del confronto tra la serie temporale relativa alla funzione F_1 con un lag pari a 16 e la serie temporale relativa a F_0 dello stesso nodo . .	39
6.3	Scenari considerati con un numero di antenne pari a 5	42
6.4	Scenari considerati con un numero di antenne pari a 10	43
6.5	Scenari considerati con un numero di antenne pari a 15	44
6.6	Miglioramento in percentuale delle performance dei modelli proposti al variare del numero di vicini per nodo con numero di antenne pari a 5 . .	45

6.7	Miglioramento in percentuale delle performance dei modelli proposti al variare del numero di vicini per nodo con numero di antenne pari a 10 . . .	45
6.8	Miglioramento in percentuale delle performance dei modelli proposti al variare del numero di vicini per nodo con numero di antenne pari a 15 . . .	46
6.9	La figura mostra come il modello con le migliori performance medie globali sia il modello DCRNN, seguito dall'approccio federato e distribuito a una fase e dal modello Prophet.	47
6.10	La figura mostra chiaramente come il modello DCRNN che ha una visione completa del sistema, ha ottenuto delle performance migliori dei modelli che non sfruttano le informazioni dei vicini. Subito dopo seguono i due approcci federati e distribuiti proposti in questo elaborato e infine il modello Prophet.	48

Acronimi

CML Centralized Machine Learning

FL Federated Learning

GDPR General Data Protection Regulation

IoT Internet of Things

ML Machine Learning

Capitolo 1

Introduzione

Negli ultimi anni siamo stati partecipi di una rivoluzione tecnologica indicata con il nome *Internet of Things (IoT)* la cui origine è da ricercare nell'aumento dei dispositivi connessi alla rete e della quantità di dati prodotti [8]. Ad oggi è il paradigma del *Cloud Computing* ad essere principalmente utilizzato nella realizzazione di piattaforme IoT grazie alla sua capacità di fornire in modo elastico le risorse computazionali e di storage necessarie alle diverse applicazioni, sollevando gli utilizzatori dall'incarico di occuparsi della manutenzione e della gestione dell'hardware [9]. Il modello del Cloud Computing è però fortemente dipendente dalla qualità della connessione a disposizione dei dispositivi connessi alla rete e soffre di tempi di latenza alti, tipicamente causati dal tempo necessario per lo scambio di grandi quantità di dati tra i suoi diversi utilizzatori [10] e dalla distanza tra di essi. Tali problematiche rendono tale modello poco adatto ad essere utilizzato in contesti in cui vi è una molteplicità di dispositivi connessi, molti dei quali necessitano di comunicare con i dispositivi vicini per poter fornire i propri servizi in tempo reale. Per questi motivi recentemente è emerso un nuovo paradigma di calcolo distribuito noto con il nome di *Edge Computing* che prevede lo spostamento del carico di lavoro dal server centrale ai nodi periferici della rete [11]. A partire da tale logica, nei recenti anni si è venuto a sviluppare un nuovo framework per l'allenamento dei modelli di *Machine Learning (ML)* denominato *Federated Learning (FL)* che garantisce tempi di latenza ridotti, una maggior tutela della privacy e una ridotta quantità di dati scambiati tra dispositivi [12].

1.1 Motivazione e Contesto

Nell'approccio tradizionale incentrato sul Cloud Computing, i dati (foto, video, informazioni sulla posizione, ecc.) raccolti dai dispositivi mobili come smartphone o sensori, vengono caricati ed elaborati da un server centrale. Successivamente, tali dati vengono processati e, in contesti particolari, utilizzati anche per lo sviluppo di modelli predittivi. È importante soffermarsi su quest'ultimo punto, in quanto come evidenziato nella Sezione 1.2, l'obiettivo di questo elaborato è quello di proporre e discutere due approcci di Machine Learning federato e distribuito, entrambi in contrapposizione al concetto di *Centralized Machine Learning* discusso nella Sezione 2.3.1.

Tale approccio non è più sostenibile per i seguenti motivi. In primo luogo, i proprietari dei dati e i regolamentatori sono sempre più sensibili alla privacy dei utenti. A seguito delle preoccupazioni sulla privacy dei consumatori nell'era dei big data, i responsabili politici

hanno risposto con la definizione e l'attuazione di normative specifiche sulla privacy dei dati, come il *Regolamento generale sulla protezione dei dati (GDPR)* della Commissione europea [13]. In particolare, il principio del consenso (articolo 6 del GDPR) e della minimizzazione dei dati (articolo 5 del GDPR) limita la raccolta e l'archiviazione dei dati solo a quanto acconsentito dal consumatore e assolutamente necessario per l'elaborazione. In secondo luogo, un approccio incentrato sull'utilizzo di un sistema centralizzato comporta lunghi ritardi nella propagazione delle informazioni all'interno di una rete composta da diversi dispositivi connessi, e ha dei tempi medi di latenza che risultano essere inaccettabili in molti scenari, specialmente in quelli in cui devono essere prese decisioni in tempo reale, ad esempio nei sistemi di auto a guida autonoma [14]. In terzo luogo, il trasferimento dei dati a un server centrale per l'elaborazione può aumentare considerevolmente il carico sulla rete, soprattutto nel caso di attività che coinvolgono l'invio da parte dei nodi periferici di dati non strutturati, come ad esempio accade nell'analisi video.

In tale contesto, nel lavoro di McMahan et al. è stato introdotto un approccio di ML decentralizzato chiamato *Federated Learning (FL)*. In breve, il FL prevede l'addestramento dei modelli predittivi sui dispositivi periferici e garantisce che i dati di training dei modelli rimangano sui dispositivi su cui sono stati raccolti. Inoltre, al di là dei chiari vantaggi in termini di privacy e riduzione del carico di rete, alcuni studi sembrano evidenziare come l'apprendimento federato possa effettivamente portare anche ad un miglioramento della qualità delle previsioni rispetto all'utilizzo di un classico approccio centralizzato [16].

Il processo di addestramento nel FL si può riassumere in due macro fasi:

1. **Allenamento locale:** ogni agente partecipante utilizza rispettivamente i propri dati e le proprie risorse computazionali per aggiornare i parametri del proprio modello locale con l'obiettivo di minimizzare la propria funzione di costo.
2. **Aggregazione globale:** il server aggrega i pesi dei modelli locali inviatigli dai dispositivi e aggiorna un modello globale.

Rispetto ai tradizionali approcci di addestramento incentrati sulla concentrazione dei dati in un punto e sull'esecuzione di modelli sul cloud, l'utilizzo del Federated Learning per l'addestramento dei modelli in maniera distribuita (es. nelle reti mobili periferiche) presenta i seguenti vantaggi:

- **Uso efficiente della banda di trasmissione della rete:** è necessario trasmettere meno informazioni al server e sui pacchetti inviati è possibile applicare degli algoritmi di compressione. Di conseguenza, questo riduce significativamente i costi di comunicazione dei dati.
- **Privacy:** ogni dispositivo non invia i propri dati al server ma soltanto i pesi ottenuti in seguito all'allenamento del proprio modello su tali dati.
- **Bassa latenza:** ogni dispositivo ha a disposizione il proprio modello e dunque può prendere decisioni autonomamente e in tempo reale. Pertanto, la latenza è molto più bassa rispetto al calcolare l'informazione sul server per poi trasmetterla al dispositivo periferico. Questo è fondamentale per le applicazioni critiche dal punto di vista temporale, come i sistemi di auto a guida autonoma, in cui il minimo ritardo può essere potenzialmente pericoloso per la vita.

- **Performance superiori:** supponendo che i dispositivi partecipanti e i server non siano malintenzionati, un maggior numero di utenti disposto a partecipare all’addestramento e la capacità dell’approccio FL di far leva sulle informazioni locali, consentirebbero lo sviluppo di modelli predittivi migliori [17].

1.2 Obiettivi della Tesi

Come descritto nella sezione precedente, l’utilizzo del CML in contesti dinamici caratterizzati da un numero variabile nel tempo di dispositivi eterogenei, soffre di diverse problematiche tra cui:

- utilizzo eccessivo della banda di trasmissione della rete;
- scarsa considerazione della privacy degli utenti.

In virtù di tali problemi, in questo elaborato di tesi si propongono due approcci di ML federato e distribuito applicati in un contesto urbano modellato costruendo una rete di nodi edge; quest’ultimi in grado di eseguire delle funzionalità richieste dagli utenti. All’interno di tale rete, le singole celle telefoniche dislocate nel centro della città di Porto (Portogallo) costituiranno i nodi edge. Tali modelli sono pensati per consentire ad ogni cella di inviare alle celle vicine le informazioni apprese sul proprio traffico in ingresso, simulato a partire dai dati presenti nel Taxy Trajectory Data¹. Le informazioni scambiate comprenderanno i soli pesi calcolati dai modelli, garantendo dunque di salvaguardare la privacy degli utenti e di ridurre il workload sulla rete. Ogni cella avrà un suo modello predittivo e i modelli proposti consentiranno a ognuno di essi di migliorare la propria capacità predittiva. L’obiettivo di questo elaborato si riassume dunque nel misurare l’efficacia di ambedue i metodi descritti nel Capitolo 5 nel prevedere il traffico in ingresso in ogni nodo edge all’interno della rete in diversi scenari urbani. Questi differiscono tra loro nel numero di antenne e nella quantità di traffico in ingresso nel tempo in ognuna di esse.

1.3 Domande di Ricerca

Due nodi edge (o celle telefoniche) appartengono allo stesso vicinato e sono dunque in grado comunicare, se la loro distanza in metri calcolata utilizzando la formula di Haversine², è minore di uno specifico valore definito in fase di sperimentazione. La comunicazione tra due o più nodi consisterà nello scambio delle informazioni apprese sul proprio traffico in ingresso, consentendo al modello di ognuna di esse di migliorare la propria capacità predittiva e al contempo di tutelare la privacy dei propri dati. Per maggiori dettagli sul concetto di vicinato si rimanda alla lettura del Capitolo 4. Le domande principali a cui si cercherà di dare una risposta attraverso questo lavoro di tesi sono le seguenti:

¹<https://www.kaggle.com/datasets/crailtap/taxi-trajectory>

²https://en.wikipedia.org/wiki/Haversine_formula

1. È possibile sviluppare un approccio di Machine Learning distribuito e decentralizzato che permetta al modello di ogni cella di far leva sulle informazioni ricevute dai nodi appartenenti allo stesso vicinato^a?
2. L'utilizzo di tale approccio consente di ottenere delle performance superiori rispetto a un approccio isolato in cui ogni nodo allena il suo modello utilizzando esclusivamente le informazioni a propria disposizione?

^aPer maggiori dettagli si rimanda alla lettura del Capitolo 4

1.4 Contributo

L'aumento dei dispositivi connessi alla rete, assieme all'incremento delle loro capacità computazionali e comunicative, ha reso il paradigma del Cloud Computing obsoleto in contesti dinamici che richiedono tempi di latenza bassi, risposte in real-time e comunicazione tra i diversi dispositivi della rete, come accade nei contesti urbani. In questo elaborato di tesi sono stati proposti due approcci distribuiti e federati che consentono a un insieme di celle telefoniche dislocate nel territorio della città di Porto di poter migliorare le proprie performance predittive relative al proprio traffico in ingresso futuro, sulla base dello scambio delle informazioni con le celle vicine. La definizione completa del problema è presente nel Capitolo 4. I principali contributi del seguente elaborato sono riassumibili come segue:

- Sono stati sviluppati due differenti approcci federati e distribuiti che consentono ai modelli LSTM presenti su ogni cella di sfruttare le informazioni storiche sul carico di lavoro delle celle vicine per poter migliorare le proprie performance predittive, consentendo al contempo anche ai propri vicini di usufruire di quanto appreso. Tali approcci sono spiegati nel Capitolo 5.
- Rispetto agli approcci proposti in lavori simili come quello di Nguyen et al., gli approcci proposti in questo elaborato non richiedono che i dispositivi della rete si scambino direttamente i dati ma solamente i pesi calcolati dai loro modelli allenati sui dati in locale, consentendo una comunicazione più veloce tra i nodi, un carico sulla rete minore e una maggiore salvaguardia della privacy di ogni dispositivo.
- L'efficacia degli approcci proposti è stata valutata in diversi scenari sulla base dei dati reali relativi alle antenne al traffico dei taxi presenti nella città di Porto. Maggiori dettagli nel Capitolo 6.

1.5 Struttura dell'Elaborato di Tesi

Si vuole ora presentare la struttura della tesi. I Capitoli 2, 3, 4 presentano le informazioni necessarie per introdurre il lettore al problema. Seguono poi altri due capitoli (5 e 6) più specifici prima delle conclusioni. La struttura della tesi è la seguente:

- il Capitolo 2 fornisce informazioni di base sulla teoria e sulle tecnologie che sono fondamentali per la comprensione degli argomenti trattati nella tesi. In particolare

viene presentata una panoramica del Cloud computing, del Fog & Edge computing e dell’evoluzione dal CML al FL;

- il Capitolo 3 presenta le ricerche e i progressi fatti finora nell’ambito della previsione del carico all’edge attraverso approcci di ML federati e distribuiti;
- il Capitolo 4 definisce il problema affrontato e la sua formulazione matematica;
- il Capitolo 5 presenta i due approcci di ML federati e distribuiti sviluppati per il seguente lavoro di tesi;
- il Capitolo 6 descrive la creazione del dataset, la sua esplorazione e l’analisi dei risultati ottenuti nella campagna sperimentale effettuata;
- il Capitolo 7 viene proposta una lettura critica dei risultati con una conseguente riflessione sui limiti e sui possibili studi futuri.

Capitolo 2

Il Contesto

L'obiettivo di questo capitolo è illustrare i concetti base necessari al lettore per comprendere l'elaborato. Ne descrive il contesto e l'evoluzione nelle sue caratteristiche essenziali. Infine, vengono riportati alcuni esempi di applicazioni reali.

2.1 Cloud Computing

Cloud Computing fa riferimento sia alle applicazioni che vengono fornite come servizi tramite internet sia all'hardware e al sistema software presente nei Data Center. Può essere considerato come un nuovo paradigma di computing, un nuovo modo di pensare all'industria dei servizi software ma generalmente non a una tecnologia specifica. Esistono varie definizioni del Cloud Computing, ma la più accurata è probabilmente quella data dal *National Institute of Standards and Technology* (Nist) [19]:

[Cloud Computing] Modello che consente in modo onnipresente, conveniente e su richiesta di accedere a un pool condiviso di risorse di elaborazione configurabili (ad es. Reti, server, archiviazione, applicazioni e servizi) che possono essere rapidamente forniti e rilasciati con il minimo sforzo di gestione o interazione con i fornitori di servizi.

2.1.1 Caratteristiche Essenziali

La completezza e l'accuratezza della definizione presentata precedentemente permettono di ricavare gli aspetti principali che caratterizzano il paradigma del Cloud Computing. Le caratteristiche essenziali sono:

1. **On-demand self-service.** Il consumatore può configurare le impostazioni di capacità, come il tempo del servizio e l'archiviazione di rete, secondo le proprie necessità senza richiedere l'interazione umana con un fornitore del servizio.
2. **Broad network access.** Le funzionalità sono disponibili sulla rete e sono accessibili tramite meccanismi standard che promuovono l'uso di piattaforme eterogenee: thin client o thick client.
3. **Resource pooling.** Le risorse informatiche del fornitore del servizio sono messe in comune per servire più consumatori, utilizzando un modello *multi-tenant*, con

diverse risorse fisiche e virtuali assegnate dinamicamente e riassegnate in base alla domanda dei consumatori. Quest'ultimi condivideranno le stesse risorse hardware ma i loro dati saranno memorizzati separatamente. Esempi di risorse includono la memorizzazione, l'elaborazione, la memoria e la larghezza di banda della rete.

4. **Rapid elasticity.** Le capacità possono essere fornite elasticamente ed essere rilasciate, in alcuni casi, automaticamente, per effettuare scaling out o scaling up in funzione della domanda. Per il consumatore, la capacità disponibile per l'approvvigionamento spesso sembra essere illimitata e le risorse possono essere utilizzate in qualsiasi quantità ed in qualsiasi momento.
5. **Measured service.** I sistemi cloud controllano e ottimizzano automaticamente l'uso delle risorse sfruttando una capacità di misurazione a un livello di astrazione appropriato al tipo di servizio (ad esempio, archiviazione, elaborazione, larghezza di banda e account utente attivo). L'utilizzo delle risorse può essere monitorato, controllato e riportato, fornendo trasparenza sia al fornitore che al consumatore del servizio utilizzato.

2.2 Fog & Edge Computing

Il cloud, per sua definizione, fa riferimento a un insieme di componenti che non si trovano vicino al suo utilizzatore ma sono allocati in varie zone del pianeta e controllate da diverse organizzazioni commerciali (*Amazon, Google, Microsoft*). Pertanto, per sopperire ai crescenti problemi di latenza e privacy, si stanno sviluppando due tendenze per il computing: Fog Computing & Edge Computing. Non esiste in letteratura una loro definizione univoca, ma soprattutto la differenza tra di esse molto spesso risulta essere sottile o controversa. Secondo L. M. Vaquero [20]:

"scenario where a huge number of heterogeneous (wireless and sometimes autonomous) ubiquitous and decentralised devices communicate and potentially cooperate among them and with the network to perform storage and processing tasks without the intervention of third parties. These tasks can be for supporting basic network functions or new services and applications that run in a sandboxed environment. Users leasing part of their devices to host these services get incentives for doing so'.

Secondo il *National Institute of Standards and Technology* (Nist) [21]:

Il Nodo Fog è la componente centrale dell'architettura. Posso essere componenti fisiche (gateway, switch, router, server) o virtuali (switch virtualizzati, macchine virtuali, cloudlet^a) che sono strettamente collegati con gli smart end devices o le reti di accesso e forniscono risorse computazione a questi dispositivi. Un Nodo Fog è consapevole della sua distribuzione geografica e della sua collocazione logica nel contesto del suo cluster. Per implementare una determinata capacità di Fog Computing, i nodi operano in modo centralizzato o decentralizzato e possono essere configurati come nodi autonomi che comunicano tra loro per fornire il servizio o possono essere federati per formare cluster che forniscono scalabilità orizzontale su aree diverse, attraverso meccanismi di mirroring^b o di estensione.

^a<https://en.wikipedia.org/wiki/Cloudlet>

^btecnica che consiste nella duplicazione dei dati su più dischi fissi, principalmente allo scopo di preservarne l'integrità in caso di guasto di uno dei dischi

Il Fog computing è spesso confuso erroneamente con l'Edge computing, ma si possono evidenziare differenze chiave tra i due concetti, come mostrato in Figura 2.1. Il primo approccio è pensato per eseguire le applicazioni in un'architettura multistrato che disaccoppia e mette in rete le funzioni hardware e software, consentendo riconfigurazioni dinamiche per diverse applicazioni mentre si eseguono servizi di calcolo e trasmissione intelligenti. Edge Computing esegue applicazioni specifiche in una posizione logica fissa e fornisce un servizio di trasmissione diretta. Il Fog computing è gerarchico, mentre l'edge computing tende ad essere limitato ad un piccolo numero di dispositivi periferici. Inoltre, oltre al calcolo e al networking, il Fog computing si include aspetti di archiviazione, controllo e velocizzazione dell'elaborazione dei dati [21].

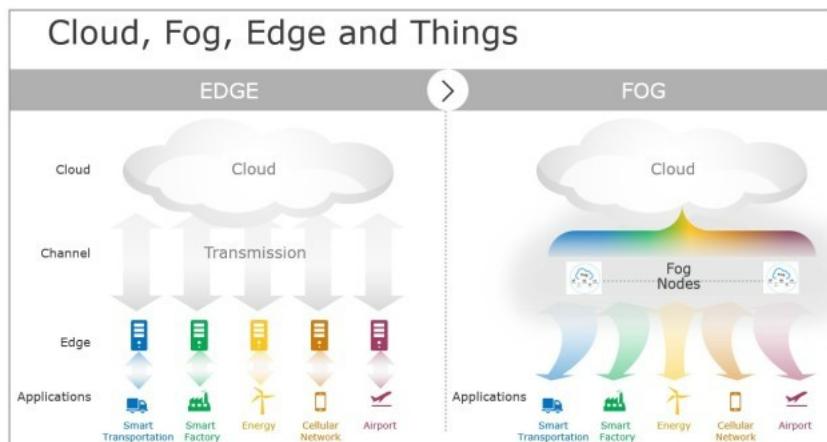


Figura 2.1: Edge Computing VS Fog Computing [1]

2.3 Dal Centralized Machine Learning al Federated Learning

Gli algoritmi di *Machine Learning* hanno dimostrato di avere un grande potenziale se applicati nella risoluzione di problemi di classificazione e regressione, grazie alla loro capacità di identificare i pattern presenti nei dati. La loro applicazione è oramai vastissima

e include diversi ambiti tra cui quello del Cloud e dell’Edge Computing. Infatti, in ambedue gli ambiti sono diversi i lavori in cui algoritmi di ML e DL vengono utilizzati per migliorare la sicurezza dei sistemi [22] e la distribuzione del carico di lavoro attraverso delle previsioni del carico futuro più accurate [23]. Di conseguenza, questi metodi *data-driven* sono tipicamente impiegati per affrontare le sfide che emergono negli ambienti mobile e richiedono lo scambio di informazioni tra i dispositivi di rete e un’entità centrale che realizza il processo di apprendimento [3]. Le applicazioni rilevanti spaziano in diversi domini (veicoli a guida autonoma, sensori intelligenti, IoT) e servono vari obiettivi (riconoscimento di immagini, elaborazione del linguaggio naturale, sanità elettronica, ecc.). L’approccio tradizionale, denominato *Centralized Machine Learning (CML)*, si basa sulla raccolta dei dati grezzi, provenienti per esempio da dispositivi IoT (misurazioni, audio, immagini, ecc.), in un nodo di calcolo centrale, il cui compito è quello di raccogliere tali dati per poi utilizzarli per l’addestramento di un modello predittivo o di classificazione. Nel caso di un scenario IoT, tale processo può generare un notevole carico sulla rete in quanto l’allenamento di un modello complesso richiede generalmente grandi quantità di dati, nonché la loro aggregazione e pulizia, attività tipicamente compiute centralmente in un nodo a livello di cloud. Per far fronte a tale problema, nel corso degli anni è stato sviluppato un nuovo paradigma per l’addestramento di modelli denominato *Distributed On-Site Learning* [2]. In tale approccio ogni dispositivo interagisce una volta sola con il server centrale per il download del modello predittivo, dopodichè quest’ultimo viene allenato sui dati in locale. È importante sottolineare che il Distributed On-Site Learning si differenzia dal Centralized Machine Learning in quanto nel secondo vi è una comunicazione continua tra il server e i dispositivi che richiedono il calcolo di una previsione, mentre nel primo la comunicazione tra server e dispositivi è limitata al solo scaricamento del modello predittivo da parte di quest’ultimi. Nel Distributed On-Site Learning non vi è dunque lo sviluppo di un modello globale che includa i contributi di ogni dispositivo, come mostrato in Figura 2.2.

Per superare tale limitazione, il *Federated Learning* si propone come un nuovo modello di apprendimento distribuito in grado non solo di sviluppare un modello globale, ma anche di salvaguardare la privacy di ogni dispositivo. Il server funge da controllore occupandosi della gestione dei round di allenamento e dell’aggregazione dei soli pesi calcolati dai modelli eseguiti sui singoli dispositivi, evitando il trasferimento di grandi quantità di dati come avviene nel CML. Nella Figura 2.2 viene graficamente riassunto quanto appena descritto. Di seguito vengono approfonditi i diversi approcci.

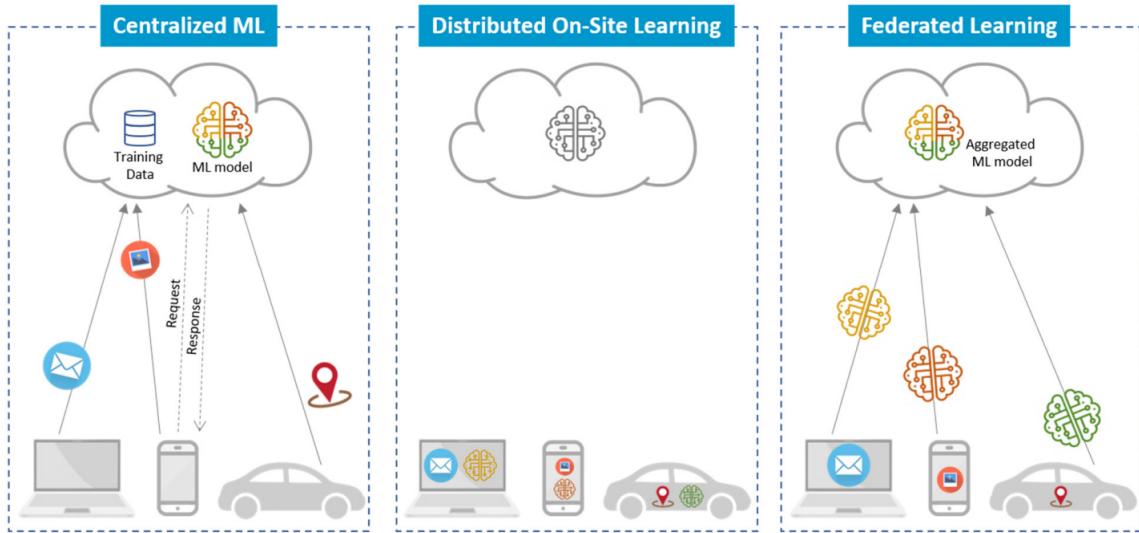


Figura 2.2: Centralized VS Distributed VS Federated Learning [2]

2.3.1 Centralized Machine Learning

Per l’addestramento di un modello universale che possa essere utilizzato da tutti i dispositivi connessi all’interno di una rete, gli approcci tipicamente utilizzati nell’ambito del *Machine Learning* e dell’*Internet of Things (IoT)* prevedono il caricamento su un server centralizzato dei dati appartenenti a ogni dispositivo. Tale forma di apprendimento centralizzato ha il vantaggio di permettere al modello una migliore generalizzazione basandosi sui pattern locali presenti nei dati provenienti da ogni dispositivo, consentendogli di migliorare le proprie performance [12]. Si noti che affinché si verifichi l’apprendimento centralizzato, il modello deve essere in grado di tenere conto della specificità del dispositivo da cui i dati provengono e di comprendere la maniera in cui quest’ultimo è influenzato dal suo ambiente circostante.

Recentemente sono stati evidenziati alcuni svantaggi derivanti dall’utilizzo di una strategia di apprendimento centralizzata ed è emerso anche che quest’ultima incontra ostacoli più ostici se applicata all’ambito *IoT*. Innanzitutto, la larghezza di banda è spesso limitata e il volume di dati potrebbe essere troppo elevato per poter essere inviato al cloud senza causare ritardi o latenze eccessive; come nel caso riportato in [24] in cui viene presentata una rete composta da diverse sottostazioni elettriche ABB in grado di produrre 5 GB/s di dati. Inoltre, i dispositivi *IoT* sono spesso di piccole dimensioni e trasportabili da un punto all’altro, il che rende difficile per quest’ultimi mantenere una connessione Internet costante. Infine, per le applicazioni che operano in tempo reale, l’eccessiva latenza derivante dal trasferimento dei dati da e verso il cloud è spesso un problema importante. Riassumendo, i problemi principali dell’apprendimento centralizzato tradizionale sono i seguenti:

- le informazioni devono essere inviate attraverso un collegamento Internet affidabile;
- la larghezza di banda potrebbe costituire il collo di bottiglia del processo di invio dei dati da e verso il server, soprattutto nel caso di dati multimediali (audio, video, immagini in alta definizione, ecc.);

- per le applicazioni in tempo reale, utilizzate ad esempio nell'ambito dell'industria automobilistica, è necessaria una latenza molto bassa [14];
- i dati sensibili devono rimanere sui dispositivi locali per salvaguardare la privacy degli utenti.

2.3.2 Distributed On-Site Learning

Il *Distributed On-Site Learning (DOL)* [2], ovvero un paradigma di apprendimento automatico che si basa sull'eseguire i modelli localmente su ogni dispositivo connesso, elimina i problemi discussi nella Sezione 2.3.1 relativi al CML. Anzichè inviare i dati raccolti a un server centrale o richiedergli di svolgere una previsione tramite API¹, ogni dispositivo riceve da quest'ultimo un modello (Figura 2.2). Tali modelli vengono allenati sui dati che il dispositivo (sensore, dispositivo indossabile, ecc.) stesso raccoglie semplificando il processo di apprendimento. Infatti, ogni modello si concentrerà soltanto sull'apprendere i pattern presenti nei propri dati, eliminando le fasi di invio dei pesi al server e di sviluppo di un modello globale presenti invece nel FL (presentato nel dettaglio nella sezione 2.3.3). Il DOL ha il vantaggio di non necessitare del trasferimento di dati sensibili al cloud alleggerendo pertanto il carico sulla rete e aumentando il livello di tutela della privacy degli utenti.

Diversi sono i suoi utilizzi soprattutto nell'ambito del *mHealth*². Come mostrato nel lavoro di Dai et al. [25], il DOL può essere utilizzato per permettere a un modello per l'identificazione del cancro della pelle pre-allenato di essere scaricato dagli utenti i quali possono usufruirne senza la necessità di inviare i propri dati al cloud. Ciononostante, la mancata comunicazione tra server e dispositivi limita l'esperienza dell'utente il cui modello predittivo non potrà beneficiare delle informazioni apprese dagli altri utenti [2].

2.3.3 Federated Learning

Il *Federated Learning (FL)* è una tecnica di ML che permette l'allenamento di un modello di ML, comune a tutti i nodi, attraverso la collaborazione dei diversi agenti (dispositivi) situati all'interno di una rete decentralizzata. Tale modello è allenato senza la necessità da parte di ogni dispositivo di condividere i propri dati, permettendo così di salvaguardare la privacy di ognuno. Introdotto per la prima volta da Google in un articolo scientifico pubblicato nel 2017 [15] con l'obiettivo di sviluppare delle tastiere virtuali per smartphone in grado di suggerire agli utenti la prossima parola da scrivere, l'utilizzo del FL si è successivamente diffuso in diversi settori a partire da quello della telefonia mobile, fino a giungere al settore della sicurezza informatica [26], dell'Industrial IoT (IIot) [27] e di altri ancora, meglio descritti in seguito.

In generale, nel FL si susseguono tre fasi principali (Figura 2.3):

1. *Inizializzazione*: il server, che svolge il ruolo di orchestratore coordinando i dispositivi partecipanti al processo di allenamento collaborativo, seleziona il modello predittivo iniziale e la funzione obiettivo da minimizzare o massimizzare.

¹<https://en.wikipedia.org/wiki/API>

²<https://en.wikipedia.org/wiki/MHealth>

2. *Allenamento in locale e invio dei pesi*: ogni dispositivo scarica il modello predittivo dal server e lo allena sui dati locali, aggiornando i pesi del proprio modello. Tali pesi saranno successivamente inviati al server.
3. *Aggregazione dei pesi e aggiornamento del modello globale*: il server aggrega i pesi ricevuti da ogni dispositivo e aggiorna il modello globale.
4. Le fasi 2 e 3 si ripetono iterativamente fino al verificarsi della condizione di termine dell'algoritmo decisa inizialmente.

I passi descritti presuppongono la sincronizzazione di tutti i dispositivi per l'invio dei pesi al server. I recenti sviluppi nell'ambito del FL hanno introdotto nuove tecniche per affrontare l'asincronicità durante il processo di allenamento dei modelli e la dinamicità della rete che varia all'aggiungersi o al disconnettersi di nuovi dispositivi [28]. Rispetto agli approcci sincroni, in cui i pesi vengono scambiati una volta che i calcoli sono stati eseguiti, quelli asincroni sfruttano alcune proprietà di alcuni modelli di ML e DL, come le reti neurali ad esempio, per scambiare gli aggiornamenti del modello non appena sono disponibili i pesi di un determinato strato [29]. L'obiettivo ultimo del FL è sviluppare dei modelli predittivi che sfruttando le informazioni apprese da tutti i dispositivi all'interno della rete, siano in grado di raggiungere delle performance superiori rispetto a quelle che si otterrebbero se ogni dispositivo non fosse in grado di comunicare con gli altri.

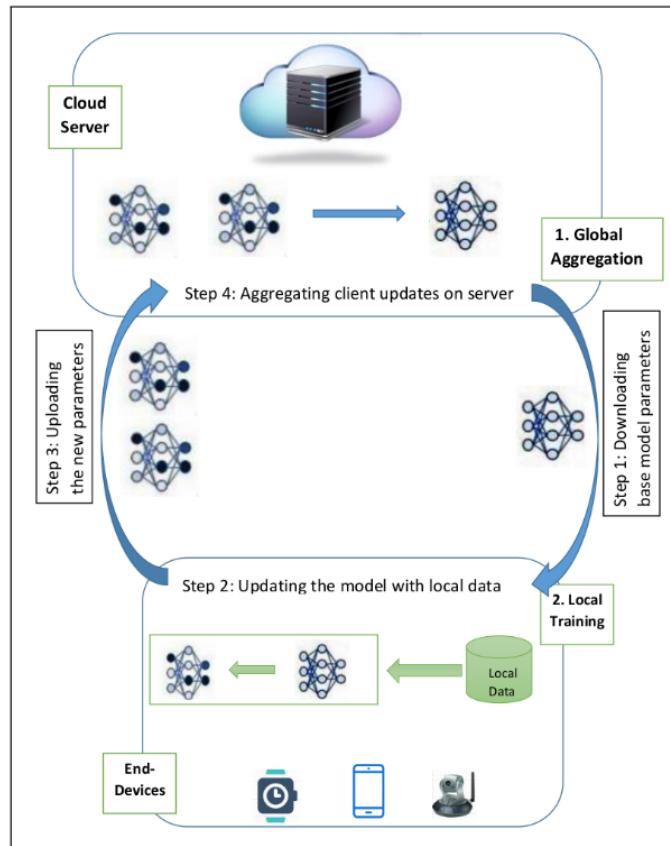


Figura 2.3: Fasi principali del FL [3]

Il sistema sulla quale il FL è stato concepito si fonda essenzialmente su tre componenti: end-devices, server (o manager) e un framework di comunicazione-computazione utilizzato per l’allenamento dei modelli di ML e DL.

- **End-devices:** gli end-devices sono i proprietari dei dati e si dividono principalmente in organizzazioni e dispositivi mobili. A seconda delle capacità computazionali e di storage di quest’ultimi, diversi sono stati gli approcci presentati negli ultimi anni con il fine di massimizzare i risultati ottenibili dai singoli dispositivi con le risorse a disposizione. Ad esempio, Wang et al. [30] hanno proposto un algoritmo che in virtù delle risorse a disposizione di ogni dispositivo per poter allenare il proprio modello, determina in real-time la frequenza di aggregazione globale dei pesi con l’obiettivo di minimizzare una specifica funzione obiettivo sotto determinati vincoli dettati dall’hardware e dalla larghezza di banda.
- **Server:** il server è un dispositivo dotato di elevate capacità computazionali e di storage che gli consentono di eseguire l’allenamento del modello globale e di gestire la comunicazione con e tra gli end-devices. La stabilità e l’affidabilità del server rivestono un ruolo importante in quanto si rischierebbe di ottenere dei modelli predittivi non accurati in caso di problemi che lo riguardassero. Per questo motivo negli ultimi anni sono stati proposti diversi approcci basati sull’utilizzo della *blockchain* per garantire una maggiore decentralizzazione del sistema e incrementarne di conseguenza la stabilità [31].
- **Framework di comunicazione-computazione:** in un sistema federato il framework di comunicazione-computazione adottato riveste un ruolo fondamentale in quanto detta le modalità con cui gli end-devices alleneranno il proprio modello e come comunicheranno tra di loro per scambiarsi i pesi calcolati. Il framework ad oggi più popolare nell’ambito del FL è denominato *Federated Averaging (FedAvg)* [15] (Figura 2.4). Ad ogni iterazione, il server invia il modello globale corrente agli end-devices che provvederanno ad allenarlo sui propri dati locali. Successivamente tutti i nuovi pesi calcolati verranno inviati nuovamente al server che si occuperà di aggregarli e di ottenere un nuovo modello globale. Tale procedura termina una volta raggiunto il limite di iterazioni massime e restituisce il modello globale finale.

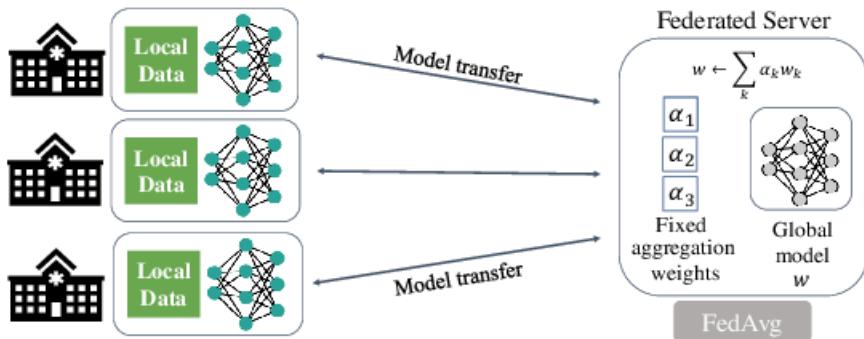


Figura 2.4: Illustrazione del framework FedAvg [4]

I vantaggi derivanti dall’utilizzo del FL sono molteplici:

- Mantenimento dei dati in locale e conseguente riduzione dei problemi derivanti dalla violazione della privacy degli utenti.
- Previsioni in tempo reale dovute al risiedere del modello predittivo e dei dati sullo stesso dispositivo.
- Non richiede capacità computazionali particolarmente elevate in quanto tale approccio delega buona parte del carico computazionale ai nodi periferici della rete.

Il FL presenta ovviamente delle limitazioni che variano da applicazione ad applicazione. Le principali difficoltà tecniche riguardano:

- Il minimizzare il numero di round necessari durante la fase di allenamento del modello e la quantità di dati inviati al cloud.
- Il far fronte all'eterogeneità dei dati raccolti da dispositivi l'uno diverso dall'altro. Infatti, la maggioranza dei modelli di ML suppone che il campione utilizzato in fase di allenamento sia costituito da osservazioni indipendenti e identicamente distribuite.

Grazie a un utilizzo efficiente della banda, alla riduzione dei tempi di risposta da parte dei modelli predittivi e alla gestione di una vasta eterogeneità di dispositivi tra loro differenti, le sue aree di applicazione sono molteplici:

- **Mobile:** il Federated Learning permette di sfruttare la sempre crescente potenza computazionale dei dispositivi cellulari. Ne è un esempio Gboard³, la tastiera sviluppata da Google⁴ per i dispositivi Android⁵ che consente agli smartphone di collaborare tra di loro con il fine di allenare un modello predittivo in grado di consigliare la parola successiva da scrivere all'interno di una frase, evitando lo scambio di dati sensibili [32].
- **Cyberattack Detection:** nei lavori di Abeshu and Chilamkurti [33] e di Nguyen et al. [26] sono stati presentati dei modelli di previsione di attacchi informatici all'interno di una rete basati sul *Federated Learning*. In tali modelli, ogni nodo ha a disposizione un insieme di dati relativi a diversi tipi di intrusioni informatiche e dopo aver allenato il proprio modello e calcolati i pesi, quest'ultimi vengono inviati al server che si occuperà di allenare un modello globale. Questi modelli sono in grado di classificare con successo il 95.6% degli attacchi azzerando il numero di falsi allarmi [26].
- **Privacy protection:** alcune organizzazioni come ad esempio gli ospedali, hanno la necessità di condividere i propri dati con altre organizzazioni, come ad esempio le agenzie organizzative o altri ospedali. Tale procedura deve però avvenire nel rispetto del Regolamento generale sulla protezione dei dati (GDPR) della Commissione Europea. Per questo motivo sono stati sviluppati degli approcci che basandosi sul *Federated Learning* che permettono ad esempio di facilitare il lavoro dei radiologi riducendo il tempo richiesto per l'analisi delle radiografie grazie ai dati condivisi da vari ospedali, rispettando al contempo la privacy dei pazienti [13].

³<https://play.google.com/store/apps/details?id=com.google.android.inputmethod.latin>

⁴https://about.google/intl/ALL_it/

⁵<https://www.android.com/>

- **Previsione dei consumi energetici:** la corretta previsione del consumo energetico è un compito fondamentale per una pianificazione, una produzione e un trasporto sostenibile dell'elettricità nei moderni sistemi energetici [34]. Saputra et al. [35] hanno proposto un nuovo approccio predittivo basato sul FL per la stima della domanda di energia all'interno di una rete composta da veicoli elettrici. Rimanendo in tale ambito, Savi and Olivadese [36] si sono avvalsi del FL per far fronte al problema dell'elevata volatilità della domanda di corrente elettrica nelle abitazioni residenziali.
- **Industrial Internet of Things (IIoT):** l'IIoT svolgerà un ruolo sempre più importante nell'ambito dello *smart manufacturing*, dello *smart factory* e dell'*Industria 4.0* grazie ai recenti avanzamenti tecnologici in ambito industriale che hanno favorito l'incremento della produttività e dell'efficienza dei sistemi aziendali [37]. Zhang et al. [38] propongono un approccio basato sul FL e sul DL per la diagnosi dei guasti dei macchinari industriali, mentre il lavoro di Liu et al. [39] combina i vantaggi del FL e del RL per permettere ai robot di potersi adattare e muovere all'interno di un nuovo ambiente in maniera produttiva e in minor tempo.

2.4 Modelli predittivi

In questa sezione verranno presentati i modelli di ML e DL applicati durante lo svolgimento del presente lavoro di tesi. I risultati ottenuti a partire da tali modelli saranno presentati e confrontati all'interno del Capitolo 6.

2.4.1 AutoRegressive Integrated Moving Average

Il modello predittivo denominato AutoRegressive Integrated Moving Average (ARIMA), è un modello di analisi statistica utilizzato per l'analisi delle serie temporali e per lo svolgimento di previsioni future a partire dai valori passati assunti dalla serie stessa. Tale modello è particolarmente indicato per l'analisi di serie temporali non stazionarie nel senso di media, dove può essere dunque applicata una fase di differenziazione iniziale (che corrisponde alla parte integrata del modello) più volte per eliminare la non stazionarietà della funzione (il trend e la stagionalità). Un modello ARIMA si basa principalmente su 3 componenti:

- *Autoregression (AR):* è un modello che permette di prevedere gli istanti temporali futuri di una serie storica utilizzando come input valori assunti in passato da quest'ultima.

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \dots + \beta_p Y_{t-p} + \epsilon_t \quad (2.1)$$

- *Integrated (I):* consente di far fronte alla non stazionarietà di alcune serie storiche andando a sostituire ogni loro valore con la differenza tra questi ultimi e quelli precedenti. Tale processo di differenziazione può essere eseguito più di una volta in base al *grado di differenziazione* che si sceglie di utilizzare.

- *Moving Average (MA)*: è un modello statistico utilizzato per modellare le serie storiche sulla base della media mobile dei loro termini passati.

$$Y_t = \alpha + \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + \dots + \phi_q \epsilon_{t-q} \quad (2.2)$$

2.4.2 Prophet

Prophet è un modello di time series forecasting sviluppato da due ricercatori di Facebook come progetto open source per rendere più semplice l'analisi e il forecasting di serie temporali [40]. Il valore della variabile target viene calcolato come la somma di tre componenti: trend, stagionalità e “vacanze”.

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t \quad (2.3)$$

Nella formula di cui sopra, il termine $g(t)$ modella le variazioni non periodiche del valore delle serie temporale, il termine $s(t)$ rappresenta le variazioni periodiche (ad esempio, la stagionalità settimanale e annuale), e $h(t)$ rappresenta gli effetti delle festività che si verificano in modo potenzialmente irregolare nell'arco di uno o più giorni. Il termine di errore ϵ_t rappresenta qualsiasi variazione idiosincratica che non viene presa in considerazione dal modello; si assume che il termine di errore ϵ_t sia normalmente distribuito.

Come descritto in dettaglio nell'articolo originale [40], il problema del calcolo delle previsioni è inquadrato come un problema di adattamento della curva, che è intrinsecamente diverso dai modelli di serie temporali che tengono esplicitamente conto della struttura di dipendenza temporale dei dati. Il problema così formulato offre una serie di vantaggi pratici:

- Diversamente dai modelli ARIMA, non è necessario che le misurazioni siano regolarmente distanziate e non è necessario interpolare i valori mancanti, ad esempio per rimuovere gli outlier.
- L'allenamento del modello è molto veloce e consente all'analista di esplorare interattivamente molte specifiche del modello.
- I parametri del modello sono facilmente interpretabili e possono essere modificati dall'analista per imporre ipotesi sulla previsione.

2.4.3 Long Short-Term Memory

Il modello LSTM è una rete neurale artificiale ricorrente in grado di elaborare intere sequenze di dati. Grazie alla sua capacità di apprendere i pattern storici all'interno di una sequenza temporale di osservazioni, tale modello ha acquisito una certa popolarità nell'ambito delle previsioni delle serie temporali. Nel caso in cui i valori futuri della serie storica dipendano dai valori assunti da quest'ultima nell'immediato passato, ovvero poco prima del valore che cerchiamo di predire, allora i classici modelli neurali di tipo Recurrent Neural Networks *RNNs* ottengono delle buone performance. Vi sono però delle situazioni in cui è necessario avere una visuale più ampia del contesto storico per poter effettuare una previsione, come ad esempio nel caso in cui volessimo prevedere una valore che sappiamo

dipendere fortemente dai valori assunti dalla serie temporale nel lontano passato. In tali situazioni, le performance derivanti dall'utilizzo di approcci ricorrenti tradizionali composti ad esempio da unità utilizzanti la sola funzione $tanh$ diminuiscono, mentre aumentano quelle ottenute dall'utilizzo di unità più complesse come *LSTM* e *GRU* [41].

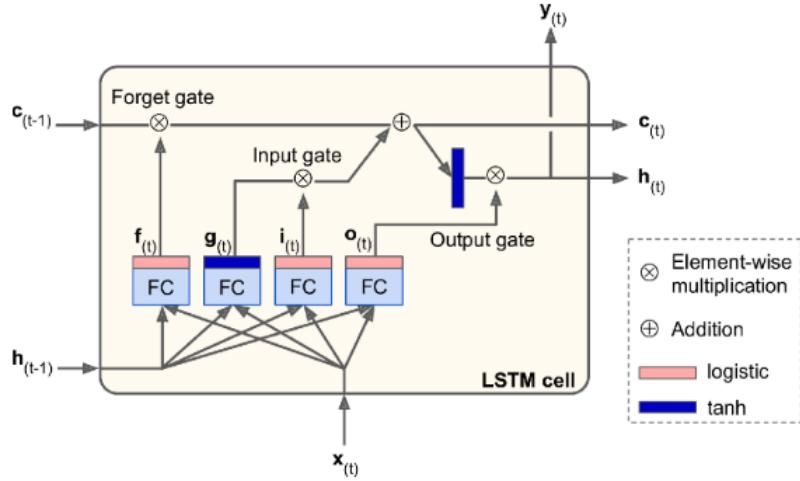


Figura 2.5: Componenti all'interno di una unità LSTM [5]

Il modello LSTM è composto da diverse unità (si faccia riferimento alla Figura 2.5) in grado di apprendere quali informazioni memorizzare nella memoria a lungo termine c_t , quali scartare e quali restituire come input all'unità successiva. c_t attraversa la rete e passa dapprima attraverso il *forget gate*, il cui compito è scartare le informazioni non ritenute utili, e in seguito li vengono aggiunte nuove informazioni selezionate dall'*input gate*. Ad ogni passaggio vengono aggiunte e scartate delle informazioni, il risultato finale è passato alla funzione *tanh* per poi essere filtrato dall'*output gate* e restituito all'unità successiva. Di seguito sono riassunte le funzionalità di ogni livello:

- Il livello principale il cui output è g_t ha il compito di analizzare l'input corrente x_t e il precedente stato h_{t-1} e dedurre le informazioni più importanti da memorizzare nel lungo termine;
- Gli altri livelli sono detti *gate controllers* e, utilizzando la *funzione logistica*, restituiscono un valore compreso tra 0 e 1. Come mostrato in Figura 2.5, il loro output viene poi moltiplicato con gli altri valori calcolati all'interno dell'unità. Si deduce che se i gate restituiscono 0 l'informazione viene scartata, se restituiscono 1 viene memorizzata. I gate controllers si dividono in:
 - *forget gate* (f_t): specifica quali informazioni devono essere eliminate dalla memoria a lungo termine;
 - *input gate* (i_t): specifica quali informazioni devono essere aggiunte nella memoria a lungo termine;
 - *output gate* (o_t): aggiorna lo stato precedente determinandone uno nuovo.

Ogni unità LSTM j mantiene dunque una memoria c_t^j al tempo t . L'output h_t^j dell'unità LSTM è dunque calcolata come segue:

$$h_t^j = o_t^j \tanh(c_t^j), \quad (2.4)$$

dove o_t^j è l'*output gate* ed è così calcolato:

$$o_t^j = \sigma(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + V_o \mathbf{c}_t)^j, \quad (2.5)$$

e σ rappresenta la funzione logistica mentre V_o è una matrice diagonale. La memoria dell'unità LSTM c_t^j è aggiornata scartando delle informazioni ottenute al tempo $t-1$ e aggiungendo \tilde{c}_t^j , ovvero quanto appreso al tempo t :

$$c_t^j = f_t^j c_{t-1}^j + i_t^j \tilde{c}_t^j. \quad (2.6)$$

La misura in cui le informazioni esistenti vengono dimenticate è modulata dal *forget gate* f_t^j , mentre la misura in cui le nuove informazioni apprese sono aggiunte allo stato attuale della cella sono modulate dall'*input gate* i_t^j . I due gate sono così calcolati:

$$\begin{aligned} f_t^j &= \sigma(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + V_f \mathbf{c}_{t-1})^j, \\ i_t^j &= \sigma(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + V_i \mathbf{c}_{t-1})^j. \end{aligned} \quad (2.7)$$

Si noti che V_f e V_i sono matrici diagonali.

A differenza delle unità *RNN* tradizionali che sovrascrivono il proprio contenuto a ogni time-step, un'unità LSTM è in grado di decidere se mantenere la memoria esistente attraverso i *gate*. Intuitivamente, se l'unità LSTM rileva una caratteristica importante da una sequenza di input in una fase iniziale, può facilmente memorizzare tale informazione e utilizzarla in futuro, catturando così potenziali dipendenze a lunga distanza.

2.4.4 Diffusion Convolutional Recurrent Network

Relativamente al problema delle previsioni del traffico, nel lavoro di Li et al. [6] gli autori presentano un modello di deep learning denominato Diffusion Convolutional Recurrent Neural Network (DCRNN) i cui risultati rispetto all'attuale stato dell'arte, sono stati superiori del 12%-15% circa. Tali performance sono state ottenute grazie alla capacità del modello di acquisire in input le dipendenze spaziali e temporali tra i dati raccolti dai sensori posizionati in diversi punti stradali. L'intero problema è stato modellato attraverso l'utilizzo di un grafo composto da nodi, ovvero i sensori, e da archi che rappresentano le dipendenze spaziali tra di essi. A ogni arco tra due nodi è stato poi associato un peso di valore pari alla loro distanza fisica. Il cuore del modello DCRNN è il *Diffusion Convolutional Recurrent Layer* in quanto al suo interno avviene la modellizzazione matematica delle dipendenze spaziali e temporali tra i nodi. Le dipendenze spaziali sono modellate come dei *processi di diffusione (diffusion process)* in quanto tale tipologia di processi è utilizzata per modellare dei fenomeni che evolvono casualmente in determinate condizioni e sotto specifiche restrizioni [42], come il traffico ad esempio. In tal modo è possibile assegnare a ogni sensore l'influenza che il tratto di strada su cui egli è posizionato ha nei confronti dei dati acquisiti dai sensori vicini. Le dipendenze temporali sono modellate grazie all'utilizzo delle *recurrent neural networks (RNNs)* e delle *Gated Recurrent Units (GRU)*, questo

assicura al modello la capacità di apprendere i pattern storici presenti nei dati e di poterli utilizzare per migliorare le previsioni future. Per consentire al modello di poter svolgere le previsioni su un intervallo di tempo futuro (multi-step ahead) più ampio, gli autori si sono affidati all'architettura chiamata *Sequence to Sequence* realizzata da Sutskever et al. [43]. Durante la fase di allenamento del modello, le serie storiche dei dati raccolti (ground truth) vengono date in input a un encoder i cui stati finali, dati in input al decoder, permetteranno a quest'ultimo di generare le previsioni. Durante la fase di allenamento, il numero di osservazioni ground truth che vengono passate al modello si riduce gradualmente in modo tale che la rete migliora sempre più la sua abilità nell'apprendere la distribuzione dei dati di test.

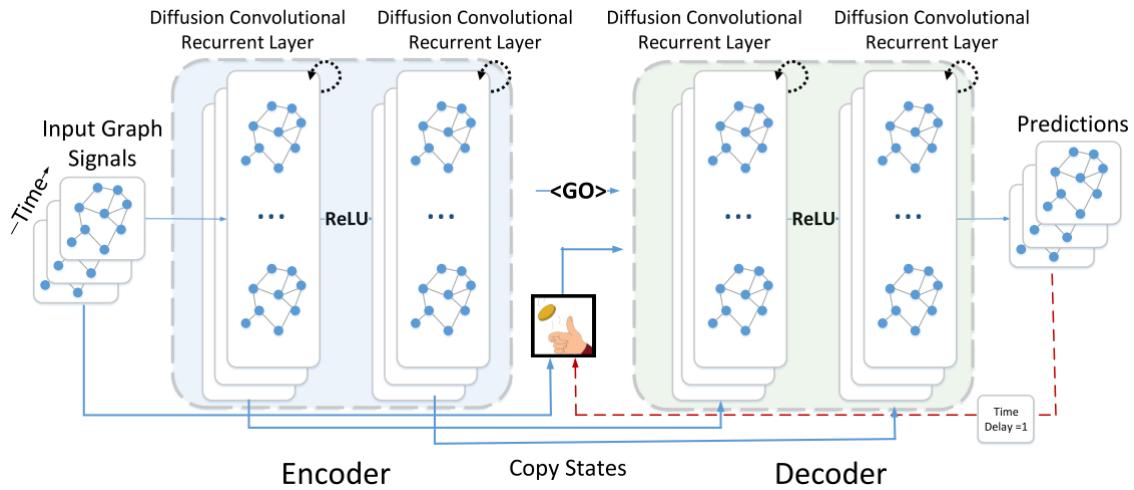


Figura 2.6: Architettura del modello DCRNN [6]

Capitolo 3

Stato dell'Arte

Questo capitolo presenta i concetti e le tecniche dello stato dell'arte relativi al problema studiato e in particolare verranno discussi i punti principali di alcuni studi presenti nella letteratura scientifica relativi all'applicazione del distributed learning per la previsione del carico all'edge. I lavori di ricerca attuali si stanno indirizzando su due aspetti principali:

1. migliorare le abilità predittive dei modelli a disposizione dei singoli dispositivi all'interno della rete, sfruttando le informazioni topologiche dell'ambiente in cui quest'ultimi risiedono e i pattern locali contenuti all'interno dei dati a loro disposizione;
2. superare gli svantaggi imposti dai tipici approcci centralizzati come ad esempio l'alta latenza e la riduzione della quantità di dati trasferiti sulla rete.

Nel lavoro svolto da Miao et al. [44] si è cercato di applicare un modello basato sulle *Graph Neural Network (GNN)* e sulle *LSTM* per prevedere il futuro consumo di risorse di ogni edge server. La scelta di utilizzare la GNN è dovuta al loro essere in grado di catturare le informazioni topologiche del grafo analizzato e di sfruttarle in modo da migliorare le abilità predittive del modello. In tale contesto si è assunto che edge server vicini avessero dei carichi di lavoro simili e dunque che il loro vicinato potesse fornire informazioni utili al miglioramento delle performance predittive dei modelli. Ogni edge server è visto come nodo di un grafo ed edge server rientranti all'interno di uno specifico raggio sono considerati tra loro collegati. All'interno del grafo, tale collegamento è rappresentato da un arco e a ogni nodo del grafo sono associati degli attributi specifici come ad esempio le risorse computazionali disponibili o i valori della serie storica rappresentante il numero di richieste nei confronti dell'edge server. La GNN aggrega a ogni step le informazioni rappresentate come vettori di ogni nodo e dei suoi vicini. Queste informazioni vengono infine passate come input a un modello LSTM che verrà utilizzato per prevedere il traffico in entrata a ogni nodo negli istanti temporali successivi. Il lavoro termina paragonando le performance predittive ottenute dagli autori utilizzando un modello basato su GNN e LSTM, chiamato GraphEdge, con quelle ottenute utilizzando la sola LSTM. I risultati, oltre a dimostrare che il modello GraphEdge ottiene MSE e WMAPE bassi, concordano nel dire che quest'ultimo ha una capacità di molto maggiore rispetto al modello LSTM nello spiegare la varianza del carico di richieste in ingresso a ogni edge server.

Il lavoro di Nguyen et al. [18] si è invece posto come obiettivo principale il superamento degli svantaggi imposti dai tipici approcci centralizzati come ad esempio l'alta latenza.

Hanno dunque sviluppato un modello distribuito chiamato *Mobile Edge Clouds (MECs)* in grado di delegare il carico computazionale ai dispositivi di rete periferici tipicamente vicini all’utente. Attraverso l’utilizzo di un modello LSTM multivariato allenato sullo storico del traffico in ingresso in ogni antenna e di quello dei suoi vicini, si sono raggiunte delle performance predittive superiori rispetto a quelle ottenute considerando le singole antenne come entità isolate l’una dalle altre. L’approccio che Nguyen et al. [18] hanno utilizzato per la creazione del dataset per l’allenamento dei modelli è molto simile a quello utilizzato per lo svolgimento di questo lavoro di tesi. Nello specifico gli autori hanno deciso di unire le informazioni contenute in due dataset differenti: uno contenente la posizione delle antenne telefoniche nella città di San Francisco e l’altro contenente la descrizione dei viaggi di 536 taxi nella medesima città. A partire da questi dati, il traffico in ingresso per ogni antenna è stato simulato calcolando la distanza tra taxi e antenna e associando a quest’ultima un numero di richieste pari al numero di taxi nel suo specifico raggio di copertura. Ottenuto il dataset, si è deciso di utilizzare un modello ARIMA allenato sui soli dati del traffico in entrata nella singola antenna e di paragonare i risultati ottenuti con quelli derivanti dall’utilizzo di un modello LSTM multivariato che invece considerava anche il traffico dei nodi vicini a quest’ultima. I risultati hanno dimostrato come il secondo approccio si sia rivelato essere più efficace.

Nel lavoro di Chen et al. [45] si è affrontato il problema della riduzione della latenza comunicativa e dell’ottimizzazione della distribuzione del carico di lavoro tra dispositivi edge ed edge data centers in ambito industriale. A detta degli autori, l’approccio discusso in tale lavoro è però adattabile anche ad altri ambiti diversi da quello industriale tra cui l’ambito delle smart city e delle smart home. Inizialmente i dati sono stati raccolti da molteplici sensori lungo una catena di montaggio per poi essere spediti agli edge data centers che, dopo averli elaborati, inviavano le indicazioni ad alcuni robot che eseguivano gli ordini ricevuti. Gli edge data centers memorizzavano le informazioni relative al carico di lavoro elaborato sotto forma di serie storiche e su tali dati sono stati in seguito applicati differenti modelli predittivi nel tentativo di ottimizzare l’utilizzo delle risorse. Gli autori hanno deciso di concentrare le previsioni del modello strettamente sull’utilizzo della CPU in quanto ritenuta una delle risorse più critiche e rappresentative del carico di lavoro. Il modello proposto in questo lavoro è stato denominato *SG-CBA* ed è stato allenato sul dataset di Alibaba [46] contenente lo storico del carico di lavoro di circa 4000 macchine. I risultati ottenuti da tale modello sono risultati essere migliori in termini di Mean Square Error (MSE), Mean Absolute Error (MAE) ed R^2 , rispetto a quelli ottenuti dai classici algoritmi di deep learning (RNN, LSTM, GRU). Alla base di tale modello vi è l’utilizzo congiunto di layer convoluzionali e unità LSTM: i primi utilizzati per estrarre le informazioni topologiche relative all’ambiente in cui i robot risiedevano, le seconde utilizzate per memorizzare l’influenza di pattern passati sulle previsioni future relative al carico di lavoro.

Capitolo 4

Assunzioni e Definizione del Problema

Il Capitolo 2 ha fornito i concetti base sulla tecnologia studiata nella tesi, mentre il Capitolo 3 ha contestualizzato lo studio nel panorama scientifico. Questo capitolo definisce il problema affrontato e propone una possibile formulazione.

4.1 Definizione del problema

Immaginiamo il caso in cui all'interno di una città siano posizionate un numero limitato e definito di celle telefoniche localizzabili sulla base delle coordinate geografiche di ognuna. Ogni cella è in grado di comunicare con tutte le celle all'interno di un determinato raggio (in metri) e di offrire, all'interno di un'ottica di tipo Edge Computing, la stessa tipologia di servizi ai diversi utenti che decidono di connettersi a esse. Ogni cella sarà dotata dunque di risorse computazionali e di storage che le permetteranno non solo di offrire un servizio a coloro i quali lo richiederanno, ma anche di memorizzare lo storico delle richieste nel tempo. La situazione di cui sopra è modellata in questo lavoro di tesi facendo uso di un grafo non orientato $G = (V, E)$, dove V è l'insieme dei vertici (celle telefoniche) che chiameremo *nodi* per chiarezza, e v_i con $i \in \{1, \dots, n\}$ rappresenta il nodo i -esimo, dato $n = |V|$ il numero di nodi di cui il grafo è composto (Figura 4.1).

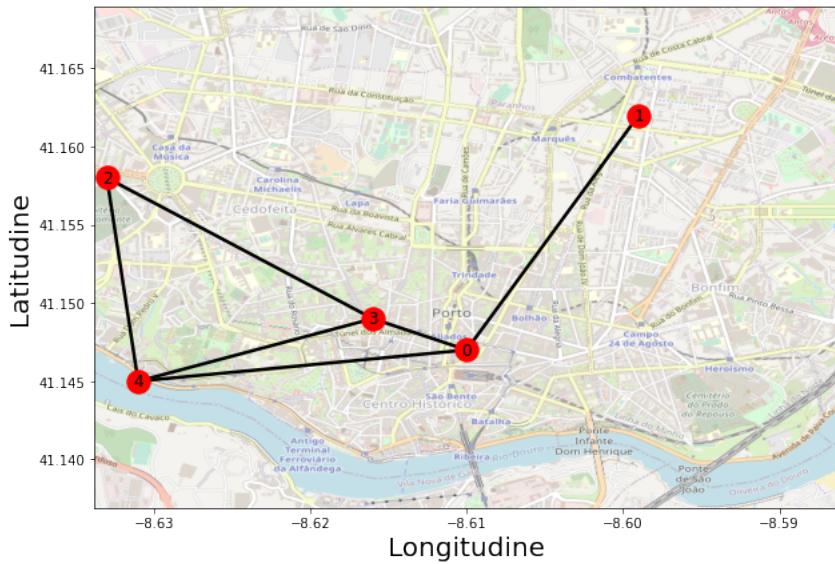


Figura 4.1: Esempio di grafo i cui nodi rappresentano le diverse celle telefoniche

Siano φ_i e φ_j rispettivamente la latitudine del nodo i e del nodo j e siano ψ_i e ψ_j rispettivamente la longitudine del nodo i e del nodo j , $\forall i, j \in \{1, \dots, n\}$, l'insieme degli archi E è descritto nella Formula 4.1:

$$E \subseteq \left\{ (V_i, V_j) \mid (V_i, V_j) \in V^2 \vee V_i \neq V_j \right. \\ \left. \vee \text{hav}(\varphi_j - \varphi_i) + \cos(\varphi_i) \cos(\varphi_j) \text{hav}(\psi_j - \psi_i) \leq m \right\} \quad (4.1)$$

La formula di cui sopra rappresenta l'insieme degli archi all'interno del grafo G ognuno dei quali è composto da una coppia di nodi V_i e V_j legati da una relazione spaziale. Tale relazione tra due nodi sussiste se la loro distanza di Haversine¹ è inferiore a un dato valore m . Tale tipologia di distanza è stata utilizzata in quanto consente il calcolo della distanza più breve tra due coordinate tenendo in considerazione la sfericità della Terra, al contrario della distanza euclidea². Dato il nodo V_i , per vicinato di V_i si intende l'insieme

$$H(V_i) = \{V_j \in V \mid i \neq j \vee \text{hav}(\varphi_j - \varphi_i) + \cos(\varphi_i) \cos(\varphi_j) \text{hav}(\psi_j - \psi_i) \leq m\}, \quad (4.2)$$

$$\forall i \in \{1, \dots, n\}.$$

Date K differenti funzioni offerte da tutti i nodi, sia $f_{i,k,t} = \langle f_{i,0,t}, f_{i,1,t}, \dots, f_{i,K,t} \rangle$ $\forall i \in \{1, \dots, n\}$ il vettore rappresentante il numero di richieste ingresso per ogni funzione offerta dal nodo i al tempo t . L'obiettivo di tale lavoro è predire $\tilde{f}_{i,k,t+d}$, dove d rappresenta un dato orizzonte temporale, minimizzando l'RMSE _{i} per ogni nodo i (Equazione 4.3).

$$\text{RMSE}_i = \sqrt{\frac{\sum_{t=1}^d \sum_{k=1}^K (|f_{i,k,t} - \tilde{f}_{i,k,t}|)}{d}} \quad (4.3)$$

Si è scelto di utilizzare come metrica di valutazione delle performance l'RMSE invece del MAE in quanto errori di grandi dimensioni, che sono i più indesiderati nel nostro contesto, sono meglio evidenziati dal primo anziché dal secondo.

¹https://en.wikipedia.org/wiki/Haversine_formula

²https://en.wikipedia.org/wiki/Euclidean_distance

4.2 Generazione delle richieste per ogni nodo

In questa sezione vengono descritti gli strumenti statistici utilizzati per la modellizzazione del processo di inter-arrivo delle richieste per ogni nodo e per la descrizione della correlazione tra i servizi offerti da ognuno di essi.

4.2.1 Distribuzione di Poisson

Il processo di Poisson è un processo estremamente utile per la modellazione in molte applicazioni pratiche, come ad esempio la modellazione dei processi di arrivo per i modelli di coda o i processi di domanda per i sistemi di inventario. È stato riscontrato empiricamente che in molte circostanze i processi stocastici che si presentano possono essere ben approssimati da un processo di Poisson [47]. Dal punto di vista degli utenti di una rete, i messaggi inviati sono tipicamente innescati da eventi particolari. Ma dal punto di vista della rete, tali eventi che producono la trasmissione di un messaggio sono in qualche modo arbitrari e imprevedibili. Pertanto, la sequenza dei tempi di arrivo dei messaggi per una determinata sessione può essere modellata per mezzo di un processo casuale. Inoltre, il processo di arrivo dei messaggi all'interno di una sessione deve essere indipendente dalle altre sessioni. Quando gli eventi si verificano secondo una distribuzione esponenziale, si dice che si verificano in modo completamente casuale. Pertanto, il tempo di arrivo dell'evento successivo non è influenzato dal tempo trascorso dall'evento precedente. Al fine di modellare i tempi di inter-arrivo delle richieste per ogni cella, si è utilizzato il processo di Poisson con parametro λ definibile dall'utente in fase di sperimentazione. Una variabile aleatoria che segue una distribuzione di probabilità esponenziale di media λ (con $\lambda > 0$) ha la seguente funzione di densità di probabilità:

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x > 0 \\ 0 & x \leq 0 \end{cases}, \quad (4.4)$$

il cui integrale ci permette di ottenere la probabilità che il tempo di attesa tra una richiesta e l'altra da parte di un nodo sia pari a:

$$F(x) = \begin{cases} 1 - e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases} \quad (4.5)$$

Pertanto la media dei tempi di attesa di un messaggio e la rispettiva varianza sono pari a:

$$\begin{aligned} E(x) &= \int_0^\infty x \cdot \lambda \cdot e^{-\lambda x} dx = \frac{1}{\lambda} \\ \text{Var}(x) &= \int_0^\infty x^2 \cdot \lambda \cdot e^{-\lambda x} dx - \left(\frac{1}{\lambda}\right)^2 = \frac{1}{\lambda^2} \end{aligned} \quad (4.6)$$

4.2.2 Markov Chain

Le catene di Markov (esempio in Figura 4.2) sono un particolare tipo di processo stocastico discreto dove la transizione allo stato successivo dipende esclusivamente dallo stato attuale, per questo motivo possono essere chiamate processo stocastico senza memoria. Esse trovano applicazione in vari campi: analisi dell'andamento dei mercati, studio della gestione delle

code nei server web e del traffico all'interno di una rete. In tale lavoro si è utilizzata una catena di Markov a tempo e stati discreti per modellare la correlazione tra i servizi offerti da ogni nodo. Si è assunto che il processo di richiesta di una funzione specifica da parte di un utente nei confronti di un nodo della rete, a partire dalla funzione richiesta precedentemente, sia di tipo markoviano. Quest'ultima assunzione si basa sullo scenario in cui i servizi richiesti da ogni utente variano al variare delle applicazioni utilizzate dal medesimo sul proprio smartphone (o qualsivoglia dispositivo mobile). In tale contesto, sono diverse le ricerche che mostrano come la scelta dell'applicazione da utilizzare successivamente dipende in larga parte da quella attualmente in uso dall'utente [48].

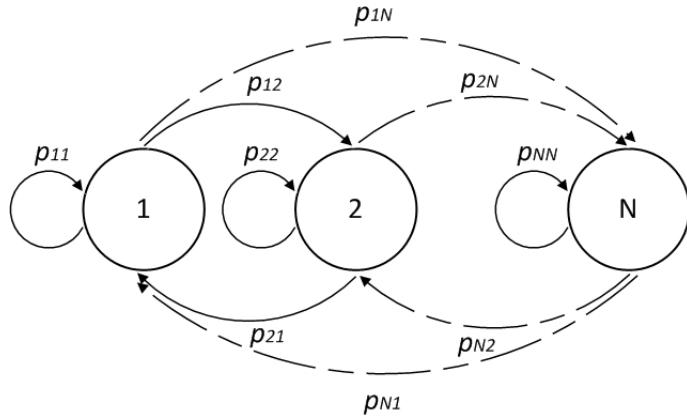


Figura 4.2: Esempio di catena di Markov [7]

Formalmente una catena di Markov è un processo stocastico Markoviano caratterizzato da un parametro $t_i \in T$, da un insieme S di stati e da una funzione probabilità di transizione $P : S \times S \times T \mapsto [0, 1]$ [49]. Un processo Markoviano P gode della proprietà di Markov:

$$\begin{aligned} P(X(t_{n+1}) = x_{n+1} | X(t_n) = x_n, X(t_{n-1}) = x_{n-1}, \dots, X(t_0) = x_0) &= \\ &= P(X(t_{n+1}) = x_{n+1} | X(t_n) = x_n) \end{aligned} \quad (4.7)$$

ovvero la distribuzione di probabilità del processo in un tempo futuro, condizionata a tutta la storia fino all'istante presente, dipende solo dal valore dello stato presente e non dalla storia passata [50]. La catena di Markov utilizzata per descrivere il processo di richiesta di un servizio al tempo $t + 1$ sulla base del servizio richiesto al tempo t , è di tipo omogeneo. In tal caso, la probabilità di richiedere un servizio al tempo t_i e dunque di passare allo stato $S(t_i)$ all'interno della catena di Markov, dipende solamente dallo stato del sistema al tempo immediatamente precedente $S(t_{i-1})$. Vale dunque la seguente condizione:

$$P(X_{n+1} = x | X_n = y) = P(X_n = x | X_{n-1} = y) \quad (4.8)$$

L'insieme degli stati S è finito e discreto in quanto coincide con l'insieme delle funzioni appartenenti a classi differenti F_k dove $k \in K := 1, \dots, K$. Tale proprietà assieme alla condizione descritta nell'eq. 4.9, rendono il processo markoviano P rappresentabile mediante una matrice di transizione $A \in \mathbb{R}^{N \times N}$. Gli elementi di A rappresentano le probabilità di transizione tra gli stati della catena: una catena che si trova nello stato i ha probabilità

a_{ij} di passare allo stato j nel passo immediatamente successivo. In particolare gli elementi a_{ii} sulla diagonale principale di A indicano le probabilità di rimanere nello stesso stato i . Si ha dunque:

$$a_{ij} = P(X_{n+1} = j \mid X_n = i) \quad (4.9)$$

dove:

$$a_{ij} \geq 0, \forall i, j \in \{1 \dots N\}, \quad (4.10)$$

$$\sum_{j=1}^N a_{ij} = 1, \forall i \in \{1 \dots N\}. \quad (4.11)$$

Capitolo 5

Approcci Proposti

Nel Capitolo 5 vengono illustrati due approcci federati e distribuiti progettati con l'intento di far fronte ad alcune problematiche relative al CML discusse nella Sezione 2.3.1. Laddove il CML basa la propria architettura sull'utilizzo di un server centrale poco adatto a contesti IoT che richiedono un'architettura dinamica e che consenta a ogni edge node di fornire previsioni in tempo reale; nelle sezioni seguenti saranno presentati due approcci pensati per consentire a ogni dispositivo della rete di comunicare agli altri le informazioni apprese a partire dai dati raccolti localmente, tutelandone la privacy e consentendo a ognuno di essi di svolgere previsioni in real-time. Sfruttando la comunicazione tra i nodi, ci si aspetta che gli approcci descritti nel Capitolo 5 ottengano delle performance superiori rispetto all'utilizzo isolato del modello LSTM da parte dei singoli nodi. I risultati ottenuti sono stati conformi alle nostre aspettative. Infatti, nel confronto dei modelli si 'e ottenuto un valore della funzione obiettivo maggiore nel caso lineare e questo 'e dipeso dal modo in cui si 'e linearizzato il vincolo sulla latenza

5.1 Approcci federati e distribuiti: caratteristiche comuni

Il primo e il secondo approccio risolutivo presentati nelle sezioni successive condividono alla base tre caratteristiche importanti. La prima è relativa alla mancata presenza di un nodo centrale che svolga il ruolo da orchestratore, tipico negli approcci basati sul FL. In nessuno dei due approcci vi è un nodo *master* e ciò evita dunque di avere un *single point of failure* in caso di guasti e/o errori al server centrale. La seconda caratteristica è relativa allo scambio di informazioni tra nodi appartenenti allo stesso vicinato. Similmente a quanto accade nel FL, ambedue gli approcci discussi in questo lavoro di tesi permettono a ogni nodo di comunicare i pesi che il proprio modello ha calcolato in seguito all'allenamento eseguito sui dati in locale. Inoltre ogni nodo ha un modello LSTM dall'architettura identica a quella messa a disposizione del modello LSTM presente negli altri nodi e quest'ultima caratteristica consente a ogni nodo di sfruttare le informazioni apprese dai modelli dei propri vicini con l'obiettivo ultimo di migliorare le proprie performance predittive. La terza e ultima caratteristica riguarda il concetto di *fine-tuning*¹, ovvero il processo attraverso il quale si utilizzano i pesi di un modello precedentemente addestrato

¹<https://en.wikipedia.org/wiki/Fine-tuning>

come valori di partenza per l’addestramento di un nuovo modello. A ogni iterazione i pesi dei modelli di ogni nodo vengono aggiornati utilizzando la media dei propri pesi e di quelli ricevuti dai nodi appartenenti al vicinato. Successivamente ogni nodo allena il proprio modello utilizzando i dati di train memorizzati localmente, costringendo quest’ultimo a specializzarsi sull’apprendimento dei pattern presenti nei propri dati locali mantenendo al contempo al suo interno la conoscenza di quelli individuati dalle celle vicine. Rispetto al lavoro di Nguyen et al. [18] (Capitolo 3) che sfrutta un modello LSTM multivariato allenato sullo storico del traffico in ingresso in ogni antenna e di quello dei suoi vicini, gli approcci presentati in questo elaborato non richiedono che ogni nodo condivida in chiaro i propri dati locali con i nodi vicini. Come è possibile notare analizzando lo pseudocodice dell’Algoritmo 1 e dell’Algoritmo 2, soltanto i pesi calcolati in seguito all’allenamento che il modello di ogni nodo compie sui propri dati locali vengono condivisi, diminuendo notevolmente il rischio che un nodo malevolo entri in possesso di informazioni riservate. Supponendo inoltre che la quantità di dati raccolti da ogni antenna aumenti notevolmente con il passare del tempo, aumenterà di conseguenza il carico sulla rete causato dal trasferimento di dati tra nodi vicini. Questo non accade negli approcci 1 e 2 che si basano sulla sola comunicazione dei pesi dei modelli su cui sarebbe inoltre possibile applicare degli algoritmi di compressione per ridurre ulteriormente la dimensione dei dati scambiati, velocizzando la comunicazione [17].

Gli approcci descritti in questo capitolo si basano sulle seguenti assunzioni:

1. Ogni nodo è in grado di comunicare con almeno un altro nodo, ciò implica che tutti i nodi partecipano all’allenamento dei modelli.
2. I nodi possono scambiarsi i dati tra di loro in quanto appartenenti alla medesima infrastruttura e la comunicazione è stabile, cioè non è soggetta a interferenze dovute, per esempio, a condizioni climatiche avverse (o altri eventi esterni). Non vi sono inoltre ritardi nell’invio e nella ricezione delle informazioni.
3. Esiste un meccanismo di sincronizzazione tra i nodi che implementa una relazione di precedenza temporale nella selezione dei nodi per il loro riallenamento. Questa assunzione è necessaria per poter simulare utilizzando un algoritmo sequenziale, il comportamento di un algoritmo distribuito.
4. Ogni nodo ha a disposizione un dataset locale contenente le informazioni sul proprio numero di richieste in ingresso per ognuna delle funzioni da lui offerte. Tale dataset verrà utilizzato da ogni nodo per l’allenamento e la valutazione del proprio modello predittivo.

Per facilitare la lettura degli pseudocodici presenti nelle sezioni successive, di seguito viene inserita una lista di variabili di supporto utilizzate in ambedue gli approcci con annessa spiegazione:

- *curr_performance*: variabile di supporto utilizzata per contenere il valore dell’indicatore di performance del modello del nodo i -esimo successivamente al fine-tuning;
- *l_performance*: lista dalle dimensioni di lunghezza pari al numero di nodi, contenente alla posizione di indice i il miglior valore dell’indicatore di performance ottenuto dal nodo i -esimo;

- e_s : variabile di supporto utilizzata per implementare il metodo *early stopping*, il cui obiettivo è quello di escludere certi nodi dall'algoritmo se per un certo numero di iterazioni le performance del loro modello non migliora;
- $l_e_stopping$: lista dalle dimensioni pari al numero di nodi contenente alla posizione di indice i il numero di iterazioni entro i quali il nodo $_i$ deve migliorare le proprie performance per evitare di essere inserito nella lista $l_stopped$. Il numero massimo di iterazioni senza alcun miglioramento delle performance consentite a ogni nodo, è pari a e_s per tutti i nodi e ogniqualvolta il nodo i dovesse ottenere delle performance superiori rispetto a quelle precedentemente raggiunte, il corrispettivo contatore viene ripristinato;
- $l_stopped$: lista inizialmente vuota che conterrà gli indici dei nodi che avranno smesso di migliorare dopo e_s iterazioni;
- $l_weights$: lista dalle dimensioni pari al numero di nodi contenente alla posizione di indice i , i pesi del modello del nodo i -esimo che hanno consentito di ottenere le performance memorizzate in $l_performance[i]$.

5.2 Algoritmo federato e distribuito a una fase

Nell'algoritmo presentato in questa sezione vi è innanzitutto una fase di inizializzazione (linea 1 - linea 5) nella quale ogni nodo addestra un proprio modello utilizzando i dati di train locali (fase 0, Figura 5.1). Nella medesima fase le liste di supporto all'interno delle quali vengono salvate le performance dei modelli valutati sui dati di test locali e i rispettivi pesi calcolati successivamente all'allenamento dei modelli, vengono inizializzate. Successivamente (linea 6 - linea 26), viene selezionato un nodo dopo aver permutato la lista dei nodi partecipanti all'allenamento in modo tale che quest'ultimo non segua sempre lo stesso ordine. Si è così in grado di approssimare il comportamento di un approccio distribuito attraverso l'utilizzo di un algoritmo sequenziale. Ogni nodo della permutazione riceve i pesi dai nodi vicini e, attraverso un'apposita funzione, ne calcola la media che utilizzerà per aggiornare il proprio modello. Successivamente esegue il fine-tuning di quest'ultimo in modo tale che egli apprenda i pattern presenti nei propri dati locali sulla base delle informazioni acquisite dagli altri nodi. Dopodichè valuta la propria performance predittiva sui dati di test locali (linea 12 - linea 23) e se le performance ottenute superano la migliore performance ottenuta fino a quel momento, il modello corrente di quel nodo viene aggiornato (fase 1, Figura 5.1). In tal modo nelle iterazioni successive i nodi suoi vicini saranno in grado di ricevere i pesi aggiornati successivamente al fine-tuning. Per evitare di bloccare l'allenamento del modello qualora non dovesse migliorare a ogni iterazione, si utilizza un valore di *early stopping* associato a ogni nodo impostato manualmente. Se in seguito a un'iterazione le performance di un modello non dovessero migliorare rispetto alla sua migliore performance ottenuta in un'iterazione precedente, il valore di early stopping associato al rispettivo nodo viene decrementato di una unità. L'algoritmo termina la sua esecuzione nel momento in cui tutti i modelli di ciascun nodo smettono di migliorare, ovvero quando la somma di tutti gli indicatori di early stopping è pari a zero. È importante notare che tale approccio si differenzia dall'Algoritmo 2 in quanto a ogni iterazione varia l'ordine con il quale i nodi comunicano tra di loro e allenano il proprio modello.

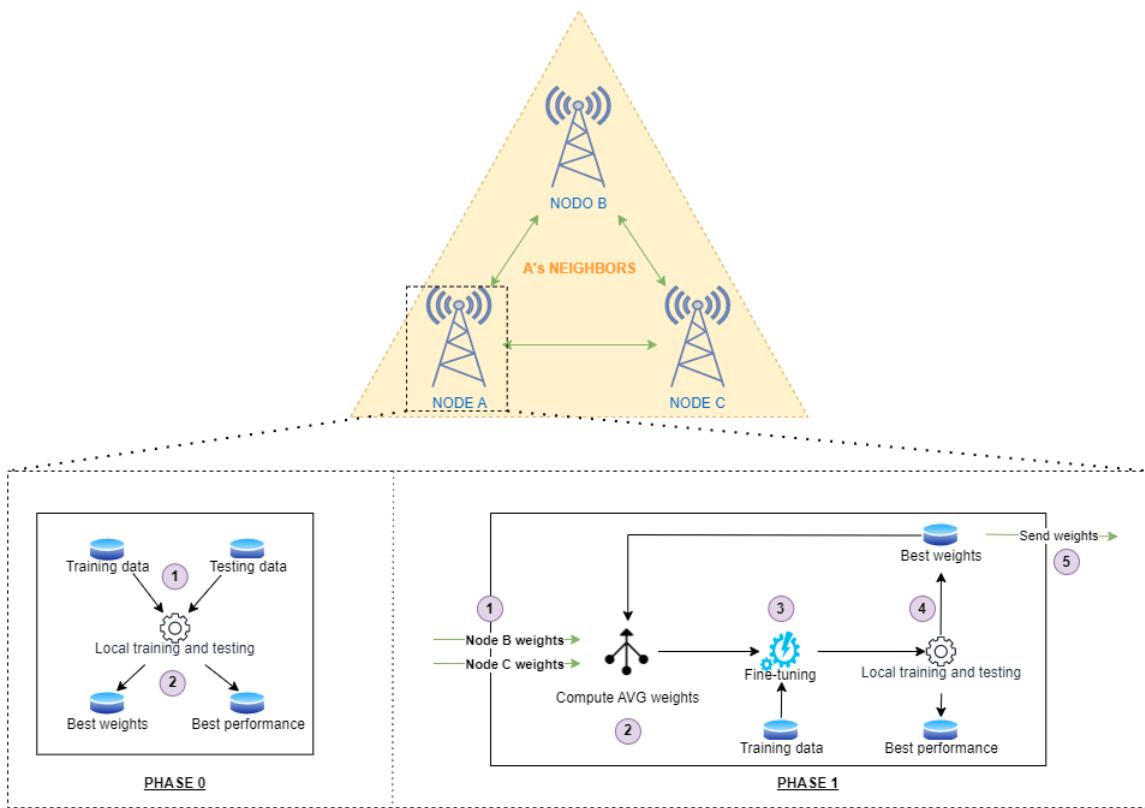


Figura 5.1: La fase 0 viene eseguita un'unica volta da tutti i nodi e consiste nell'allenamento in locale del proprio modello senza alcuna interazione con i dispositivi nella rete. La fase 1 viene eseguita in sequenza da tutti i nodi fino a quando nessuno di essi migliora le proprie performance. Il nodo riceve i pesi dai propri vicini, gli aggrega e partendo da questi esegue il fine-tuning. Se le sue performance migliorano rispetto a quelle precedenti, memorizza i pesi in una lista locale e invia i nuovi ai nodi vicini.

Algorithm 1 Algoritmo federato e distribuito a una fase

Input: lista dei nodi (*l_master*), lista dei vicini per ogni nodo (*l_neighborhood*)
Output: lista contenente il valore di RMSE più basso per ottenuto da ogni nodo (*l_performance*), lista contenente per ogni nodo i pesi che hanno consentito le performance migliori (*l_weights*)

```
1: for nodoi in l_master do
2:   nodoi allena il proprio modello sui dati locali
3:   l_performance[i]  $\leftarrow$  performance valutata sui dati di test locali
4:   l_weights[i]  $\leftarrow$  pesi ottenuti in seguito all'allenamento
5: end for
6: while almeno un valore in l_e_stopping è diverso da 0 do
7:   l_master  $\leftarrow$  shuffle(l_master)
8:   for nodoi in l_master do
9:     if nodoi not in l_stopped then
10:      n_w  $\leftarrow$  nodoi riceve i pesi dei vicini contenuti in l_neighborhood[i]
11:      nodoi esegue fine-tuning del proprio modello a partire da n_w
12:      curr_perf  $\leftarrow$  valuto il modello del nodoi
13:      if curr_perf > l_performance[i] then
14:        l_performance[i]  $\leftarrow$  curr_perf
15:        l_weights[i]  $\leftarrow$  n_w
16:        l_e_stopping[i]  $\leftarrow$  e_s
17:      else
18:        if l_e_stopping[i]  $\geq$  1 then
19:          l_e_stopping[i]  $\leftarrow$  l_e_stopping[i] - 1
20:        else
21:          l_stopped.append(nodoi)
22:        end if
23:      end if
24:    end if
25:  end for
26: end while
```

5.3 Algoritmo federato e distribuito a due fasi

Nell'algoritmo presentato in questa sezione vi è innanzitutto una fase di inizializzazione (linea 1 - linea 5) nella quale ogni nodo addestra un proprio modello utilizzando i dati di train locali (fase 0, Figura 5.2). Diversamente da quanto avviene nell'Algoritmo 1, in questo secondo approccio lo scambio dei pesi e l'allenamento in locale dei modelli da parte di ogni nodo avviene in due ulteriori fasi ben distinte. Nella prima fase tutti i nodi richiedono i pesi dei propri vicini e li memorizzano in un'apposita lista (fase 1, Figura 5.2). Nella seconda fase tutti i nodi a partire dai pesi ricevuti nella fase precedente, eseguono il fine-tuning del proprio modello e ne testano le performance (fase 2, Figura 5.1). Rispetto all'Algoritmo 1, l'Algoritmo 2 prevede che tutti i nodi si scambino i pesi e allenino i propri modelli contemporaneamente assumendo che sia presente un meccanismo

di sincronizzazione in grado di separare temporalmente le due fasi. Rispetto al caso precedente, però, non esiste una rigida dipendenza temporale tra gli eventi (scambio pesi e riallenamento) all'interno della stessa fase. Anche in questo secondo approccio l'algoritmo termina la sua esecuzione nel momento in cui tutti i modelli di ciascun nodo smettono di migliorare, ovvero quando la somma di tutti gli early stopping è pari a zero.

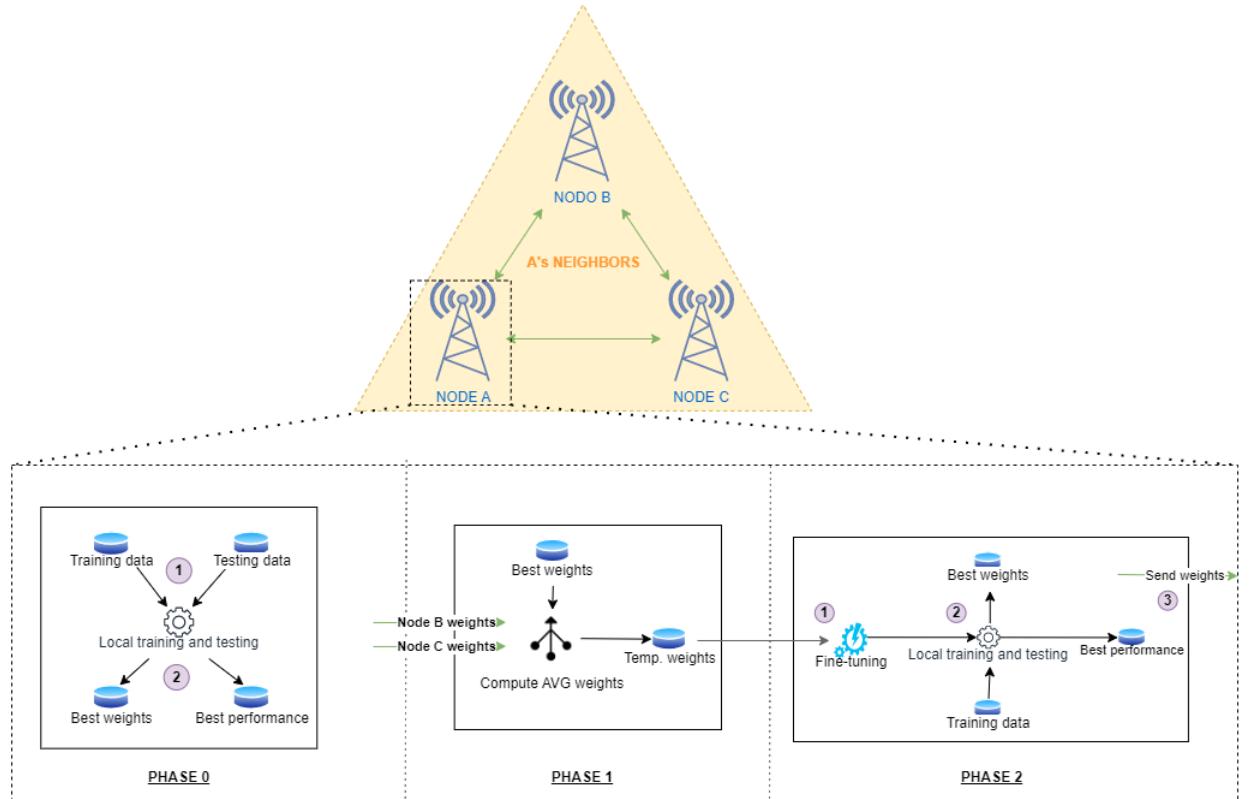


Figura 5.2: La fase 0 viene eseguita un'unica volta da tutti i nodi e consiste nell'allenamento in locale del proprio modello senza alcuna interazione con i dispositivi nella rete. La fase 1 e la fase 2 vengono eseguite in sequenza da tutti i nodi fino a quando tutti smettono di migliorare le proprie performance. Tutti i nodi ricevono contemporaneamente i pesi dai propri vicini e ognuno di essi li memorizza all'interno di una lista temporanea. Successivamente e sempre contemporaneamente, nella fase 2 ogni nodo esegue il fine-tuning del proprio modello e aggiorna eventualmente la lista dei pesi che gli hanno garantito le migliori performance.

Algorithm 2 Algoritmo federato e distribuito a due fasi

Input: lista dei nodi (*l_master*), lista dei vicini per ogni nodo (*l_neighborhood*)
Output: lista contenente il valore di RMSE più basso ottenuto da ogni nodo (*l_performance*), lista contenente per ogni nodo i pesi che hanno consentito le performance migliori (*l_weights*)

```
1: for i in l_master do
2:     nodoi allena il proprio modello sui dati locali
3:     l_performance[i]  $\leftarrow$  performance valutata sui dati di test locali
4:     l_weights[i]  $\leftarrow$  pesi ottenuti in seguito all'allenamento
5: end for
6: while almeno un valore in l_e_stopping è diverso da 0 do
7:     for i in l_master do
8:         if nodoi not in l_stopped then
9:             n_w  $\leftarrow$  nodoi riceve i pesi dei vicini contenuti in l_neighborhood[i]
10:            l_temp_weights[i]  $\leftarrow$  n_w
11:        end if
12:    end for
13:    for i in l_master do
14:        if nodoi not in l_stopped then
15:            fine-tuning del proprio modello a partire da l_temp_weights[i]
16:            curr_perf  $\leftarrow$  valuto il modello del nodoi
17:            if curr_perf > l_performance[i] then
18:                l_performance[i]  $\leftarrow$  curr_perf
19:                l_weights[i]  $\leftarrow$  n_w
20:                l_e_stopping[i]  $\leftarrow$  e_s
21:            else
22:                if l_e_stopping[i]  $\geq$  1 then
23:                    l_e_stopping[i]  $\leftarrow$  l_e_stopping[i] - 1
24:                else
25:                    l_stopped.append(nodoi)
26:                end if
27:            end if
28:        end if
29:    end for
30: end while
```

Capitolo 6

Valutazione Sperimentale

Il Capitolo 6 illustra il processo di creazione del dataset a partire dal quale sono state svolte le diverse valutazioni sperimentali e i risultati ottenuti dall'applicazione degli algoritmi predittivi descritti nel Capitolo 2 e nel Capitolo 5.

6.1 Creazione del dataset

Il dataset che si è utilizzato come base di partenza per la simulazione del traffico è il *Taxy Trajectory Data*¹ contenente i viaggi svolti nella città di Porto (Portogallo) da 442 taxi dal 01/07/2013 al 30/06/2014. A ogni viaggio è associata una lista contenente un insieme di coordinate (latitudine e longitudine) che descrivono la posizione del rispettivo taxi ogni 15 secondi lungo il tragitto. Alla base vi è l'idea di simulare uno scenario reale in cui vi siano delle antenne posizionate nell'area della città Porto e un certo numero di dispositivi (taxi) che in base alla loro posizione si agganciano all'antenna più vicina richiedendo una funzione tra quelle offerte.

Il dataset pubblico *OpenCellID*² contiene migliaia d'informazioni relative alle antenne per cellulari sparse nel mondo e alla loro posizione. Di tale dataset si andranno a utilizzare le sole informazioni relative alle antenne installate nella città di Porto. Nel suo formato CSV originale, tale dataset si presenta come mostrato di seguito:

```
radio ,mcc,mnc,area ,cellon ,lat ,range ,samples ,changeable ,created ,updated ,averagesignal  
GSM,268,1,31,43141,-16.910.019,32.653.428,1295,2,1,1303558361,1446746924,0  
GSM,268,3,7000,20101,-16.915.685,32.649.413,7817,321,1,1303558361,150305122,0  
GSM,268,1,31,36071,0,-16.919.403,32.664.299,1421,12,1,1303558361,1492427481,0
```

Ogni riga del file ha 13 campi:

- *radio*: identifica lo standard utilizzato per la gestione delle comunicazioni mobili;
- *mcc*: è un codice di due o tre cifre che identifica l'area geografica della rete GSM;
- *mnc*: è un codice di due o tre cifre che identifica l'operatore telefonico;
- *area*: è il Location Area Code (LAC), ovvero un codice che identifica in maniera univoca un'area geografica all'interno della rete;

¹<https://www.kaggle.com/crailtap/taxi-trajectory>

²<https://opencellid.org/#zoom=16&lat=41.14945&lon=-8.61079>

- *cell*: è il Cell tower code (CID), ovvero l'identificatore univoco della cella;
- *lon*: longitudine della cella;
- *lat*: latitudine della cella;
- *range*: a partire dal punto valore definito dai valori di longitudine e latitudine, il valore in tale colonna rappresenta il raggio entro la quale si trova la reale posizione della cella con un intervallo di confidenza pari al 98%;
- *samples*: numero di dispositivi (contributi) utilizzati per calcolare i dati relativi alla cella;
- *changeable*: assume il valore 0 se la posizione dell'antenna è stata calcolata eseguendo una media delle posizioni ricevute dagli utenti, mentre assume il valore 1 se tale posizione è stata data direttamente dall'azienda proprietaria dell'antenna;
- *created*: UNIX timestamp della prima volta in cui l'antenna è stata inserita nel database sorgente;
- *updated*: UNIX timestamp indicante la data dell'ultimo aggiornamento dei dati relativi all'antenna;
- *averagesignal*: media delle potenze di segnale dell'antenna ricevute dagli utenti.

In modo da rendere il dataset *OpenCellID* utile ai nostri fini, su di esso sono state eseguite diverse operazioni di preprocessing:

1. Eliminazione delle righe duplicate.
2. Eliminazione delle colonne non utili per i fini del lavoro (*mcc*, *mnc*, *area*).
3. Filtraggio delle antenne non rientranti nell'area geografica considerata.

Successivamente questo dataset verrà utilizzato per il posizionamento di un campione di antenne sul territorio della città di Porto e la successiva modellazione di queste ultime attraverso l'utilizzo di un grafo.

La generazione del traffico entrante in ogni cella si deve invece all'utilizzo del dataset *Taxy Trajectory Data* contenente 1.710.670 righe e 5 campi:

- *trip_id*: codice alfanumerico che identifica univocamente la tratta percorsa dal taxi;
- *taxi_id*: codice numerico che identifica in maniera univoca il taxi;
- *timestamp*: stringa numerica che rappresenta il tempo in secondi trascorso dalla mezzanotte del 1º gennaio 1970 fino all'inizio del viaggio;
- *missing_data*: valore binario (true/false) che identifica la presenza o meno di errori nell'acquisizione dei dati sulla posizione del veicolo da parte del dispositivo GPS;
- *polyline*: lista di tuple nel formato (longitudine, latitudine) contenente la posizione geografica del taxi ogni 15 secondi dall'inizio del tragitto fino al suo termine.

Come nel caso dell’*OpenCellID* dataset, anche sul dataset *Taxy Trajectory Data* è stato necessario svolgere operazioni di preprocessing:

1. Eliminazione delle righe contenenti misurazioni GPS parziali.
2. Feature engineer dei valori contenuti nella colonna *polyline* in modo da rendere le future operazioni su tale colonna più agevoli. Si è passati infatti dall’avere un insieme di liste separate del tipo [LONGITUDINE, LATITUDINE], ad avere un’unica lista contenente un insieme di tuple (LONGITUDINE, LATITUDINE) che ha permesso di facilitare alcuni controlli come ad esempio quelli relativi al filtraggio dei viaggi troppo corti e di facilitare la creazione delle grafiche come quelle presenti in Figura 6.3.
3. Creazione della colonna *trip_length* contenente la durata in secondi del rispettivo viaggio.
4. Esclusione dei viaggi anomali (contenenti spostamenti quasi istantanei da un punto geografico a quello successivo) e dei viaggi iniziati o terminati al di fuori dell’area geografica di riferimento.
5. Aggiunta della posizione interpolata dei taxi nei 13 secondi che intercorrono tra ogni coppia successiva di coordinate a nostra disposizione all’interno della colonna *polyline*. Tale operazione è stata svolta facendo uso della libreria *pyproj4*³ che ha permesso di agevolare il calcolo dei 13 punti equamente distanti lungo la geodetica che unisce la posizione del taxi con la stessa nei 15 secondi successivi.

Sulla base dei due dataset così ottenuti, si è poi proceduto alla generazione del traffico in ingresso in ogni cella. L’assegnamento di una richiesta alla cella più vicina al taxi in ogni istante del suo viaggio, la gestione dei tempi di inter-arrivo delle richieste per ogni cella e la correlazione tra le richieste delle diverse funzioni all’interno dello stesso nodo, sono stati rispettivamente ottenuti utilizzando la distanza di Haversine, la distribuzione di Poisson e il processo Markoviano descritto nel Capitolo 4. Il risultato finale è una lista di dataset rappresentanti lo storico del traffico in ingresso per ogni cella (Tabella 6.1). Su ognuno di questi dataset sono in seguito stati comparati i diversi approcci predittivi presentati nel Capitolo 2 e nel Capitolo 5.

³<https://github.com/pyproj4/pyproj>

index	F_0	F_1	F_2	F_3	F_4
2013-07-01 06:17:25	0	1	0	0	1
2013-07-01 06:17:26	0	2	0	0	0
2013-07-01 06:17:27	1	0	1	0	2
2013-07-01 06:17:28	0	0	0	0	0
2013-07-01 06:17:29	0	0	2	0	0
2013-07-01 06:17:30	0	1	0	0	0
2013-07-01 06:17:31	1	0	2	0	0
2013-07-01 06:17:32	0	1	0	0	0
2013-07-01 06:17:33	0	0	1	0	0
2013-07-01 06:17:34	1	0	0	0	2

Tabella 6.1: Dataset contenente il numero di richieste per funzione di un nodo della rete

6.2 Esplorazione dei dati

Per ogni dataset rappresentante il traffico in ingresso in ogni nodo, si sono svolte le necessarie analisi statistiche atte a fornire una comprensione migliore dei dati sulla quale saranno in seguito comparati i modelli predittivi. Innanzitutto si è verificato che le serie temporali degli andamenti del traffico in ingresso in ogni antenna fossero stazionari. La stazionarietà dei dati è un'assunzione alla base del modello ARIMA, utilizzato in questo lavoro di tesi. Come mostrato in Figura 6.1, la media e la deviazione standard si mantengono costanti nel tempo, convalidando l'ipotesi di stazionarietà dei delle serie.

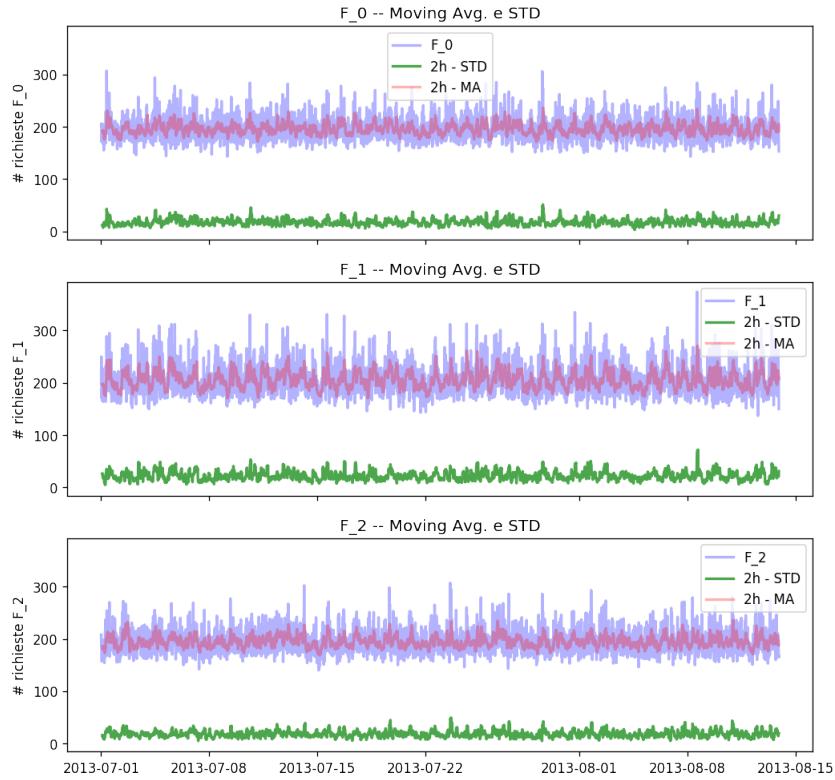


Figura 6.1: Moving average e standard deviation relative alle funzioni dell’antenna numero 0 calcolate su una finestra temporale di 2 ore.

Per ottenere un’ulteriore conferma della veridicità di tale ipotesi, è stato applicato un test statistico denominato *Augmented Dickey-Fuller* per verificare o meno la presenza di una radice unitaria. I valori del p-value ottenuti dal test sono stati di molto inferiori al livello di significatività selezionato (0.05). Questo significa che i dati a nostra disposizione avvalorano la tesi della mancanza di una radice unitaria all’interno delle nostre serie storiche e che dunque queste ultime sono stazionarie. In seguito si è analizzata l’interdipendenza tra le serie storiche degli andamenti del traffico in ingresso di ogni funzione messa a disposizione da ogni nodo. A tal fine è stato utilizzato il test statistico di *causalità di Granger* che permette di determinare l’utilità di una serie storica nel prevederne un’altra. Tale analisi si traduce nel comprendere se serie storica rappresentante il numero di richieste per una specifica funzione offerta dal nodo può essere utile a prevedere il numero di richieste future di un’altra funzione. Tale test ha restituito risultati statisticamente significativi fino al lag 16, ovvero il numero di richieste in ingresso nel nodo relative a una specifica funzione nelle 4 ore (con frequenza di raggruppamento dei dati pari a 15 minuti) precedenti, è utile a prevedere il numero di richieste future di un’altra funzione offerta dallo stesso nodo. Come mostrato nella Figura 6.2, se considero la serie storica F_1 traslata all’indietro di 4 ore, essa è quasi perfettamente sovrapponibile alla serie storica relativa a F_0 .

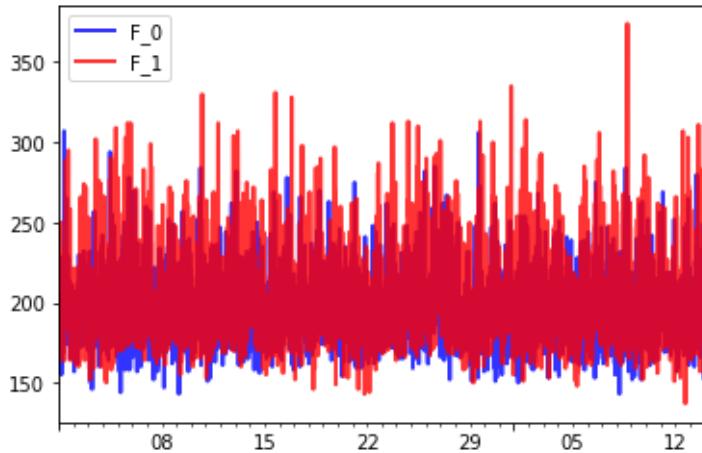


Figura 6.2: Esempio del confronto tra la serie temporale relativa alla funzione F_1 con un lag pari a 16 e la serie temporale relativa a F_0 dello stesso nodo

6.3 Campagna sperimentale

La campagna sperimentale atta alla valutazione dell’efficacia degli approcci proposti in questo elaborato di tesi sulla base dei dati descritti nella sezione precedente, è stata svolta attraverso l’utilizzo di un simulatore totalmente sviluppato in *Python*⁴ eseguito sulla piattaforma *Google Colab*⁵. Quest’ultimo si compone di due fasi, ognuna delle quali ha dei parametri selezionabili dall’utente in base alla tipologia di valutazione sperimentale che intende svolgere. Le due fasi sono le seguenti:

1. **Generazione del traffico:** in questa fase avviene il processo descritto nella Sezione 6.1 in cui a partire dai dataset *Taxy Trajectory Data* e *OpenCellID*, si generano i dataset contenenti le informazioni sul traffico in ingresso in ogni cella in uno specifico periodo temporale. Sempre in questa fase avviene la creazione del grafo che conterrà al suo interno le informazioni relative alla posizione geografiche delle antenne (nodi) e alla loro relazione spaziale (archi). All’utente è data la possibilità di modificare i seguenti parametri:
 - *numero di antenne*: numero di antenne che andranno a comporre i nodi del grafo (maggiori dettagli nel Capitolo 4);
 - *raggio*: definisce l’area attorno a ogni antenna all’interno della quale le antenne presenti verranno considerate appartenenti allo stesso vicinato e dunque in grado di comunicare tra di loro. Due o più nodi appartenenti allo stesso vicinato sono uniti da un arco all’interno del grafo;
 - *numero di viaggi*: definisce la grandezza del campione di viaggi estratto dal dataset Taxy Trajectory Data da utilizzare per la generazione del traffico;
 - *numero di funzioni*: definisce il numero di servizi che ogni antenna espone al pubblico.

⁴<https://www.python.org/>

⁵<https://research.google.com/colaboratory/faq.html>

2. **Applicazione dei modelli:** a partire dai dataset creati nella fase precedente, in questa fase si svolge l'applicazione dei modelli descritti nel Capitolo 2 e nel Capitolo 5, nonchè il calcolo delle rispettive metriche e la generazione dei grafici. In questa fase i parametri modificabili dall'utente sono:

- *frequenza*: stringa indicante la frequenza temporale di raggruppamento dei dati di ogni nodo. Può assumere valori del tipo: *15min*, *30min*, ecc.;
- *percentuale di training*: valore numerico indicante per ogni nodo la percentuale di dati che quest'ultimo utilizzerà per il training dei modelli;
- *n. steps in*: valore numerico indicante il numero di righe che ogni modello utilizzerà per allenarsi a prevedere le *n. steps out* righe successive. Nel caso ad esempio si decida di utilizzare una frequenza di raggruppamento pari a 15 minuti e di assegnare un valore del parametro *n. steps in* pari a 4, ciò significherebbe che verrebbero create delle *sliding windows*⁶ di un'ora sui cui allenare i modelli;
- *n. steps out*: valore numerico indicante il numero di righe successive alle *n. steps in* che ogni modello cercherà di prevedere e sulla quale verrà calcolato l'RMSE.

Sulla base dei parametri precedentemente descritti, questo elaborato si concentra sull'analisi delle performance dei diversi approcci proposti all'aumentare del numero delle antenne. Nello specifico sono state svolte tre sperimentazioni, variando ogni volta il parametro *numero di antenne* e facendoli assumere i seguenti valori: 5, 10, 15. Le tre sperimentazioni condividono tutte i seguenti parametri e i seguenti valori a loro assegnati:

- *raggio*: 1000;
- *numero di viaggi*: 10000;
- *numero di funzioni*: 3;
- *frequenza*: 15min,
- *percentuale di training*: 80%;
- *n. steps in*: 16;
- *n. steps out*: 1.

Per quanto riguarda il valore dei parametri, per quanto ne sappiamo non esiste in letteratura un riscontro sulla loro entità; per tale motivo sono stati scelti secondo le nostre necessità.

Si è deciso inoltre di utilizzare il seguente processo con il fine di rendere più robuste le metriche raccolte per ognuna delle tre sperimentazioni:

1. Per ogni combinazione di parametri della simulazione si sono generati 5 diversi scenari il cui numero di antenne è invariato, ma varia la loro locazione geografica.

⁶<https://machinelearningmastery.com/time-series-forecasting-supervised-learning/>

- Per ogni scenario l’allenamento e la valutazione di ognuno dei modelli in analisi è avvenuta 5 volte con il fine di calcolare un intervallo di confidenza utile per la loro comparazione.

Attraverso le metriche raccolte in ogni sperimentazione si cercherà di rispondere alle seguenti domande di ricerca:

- Gli approcci che permettono ai modelli predittivi di ogni nodo di sfruttare le informazioni ricevute dal proprio vicinato, ottengono performance superiori rispetto agli approcci che non tengono conto di queste informazioni?*
- Come variano le performance degli approcci federati e distribuiti proposti (Capitolo 5) al variare del numero di vicini per nodo?*

6.3.1 Confronto tra i modelli nei diversi scenari

Come descritto nella Sezione 6.3, sono state svolte tre sperimentazioni variando di volta in volta il parametro *numero di antenne* e facendoli assumere i seguenti valori: 5, 10, 15. Sono stati in seguito generati 5 diversi scenari per ognuno dei tre valori di tale parametro e quest’ultimi sono rappresentati in Figura 6.3, in Figura 6.4 e infine in Figura 6.5.

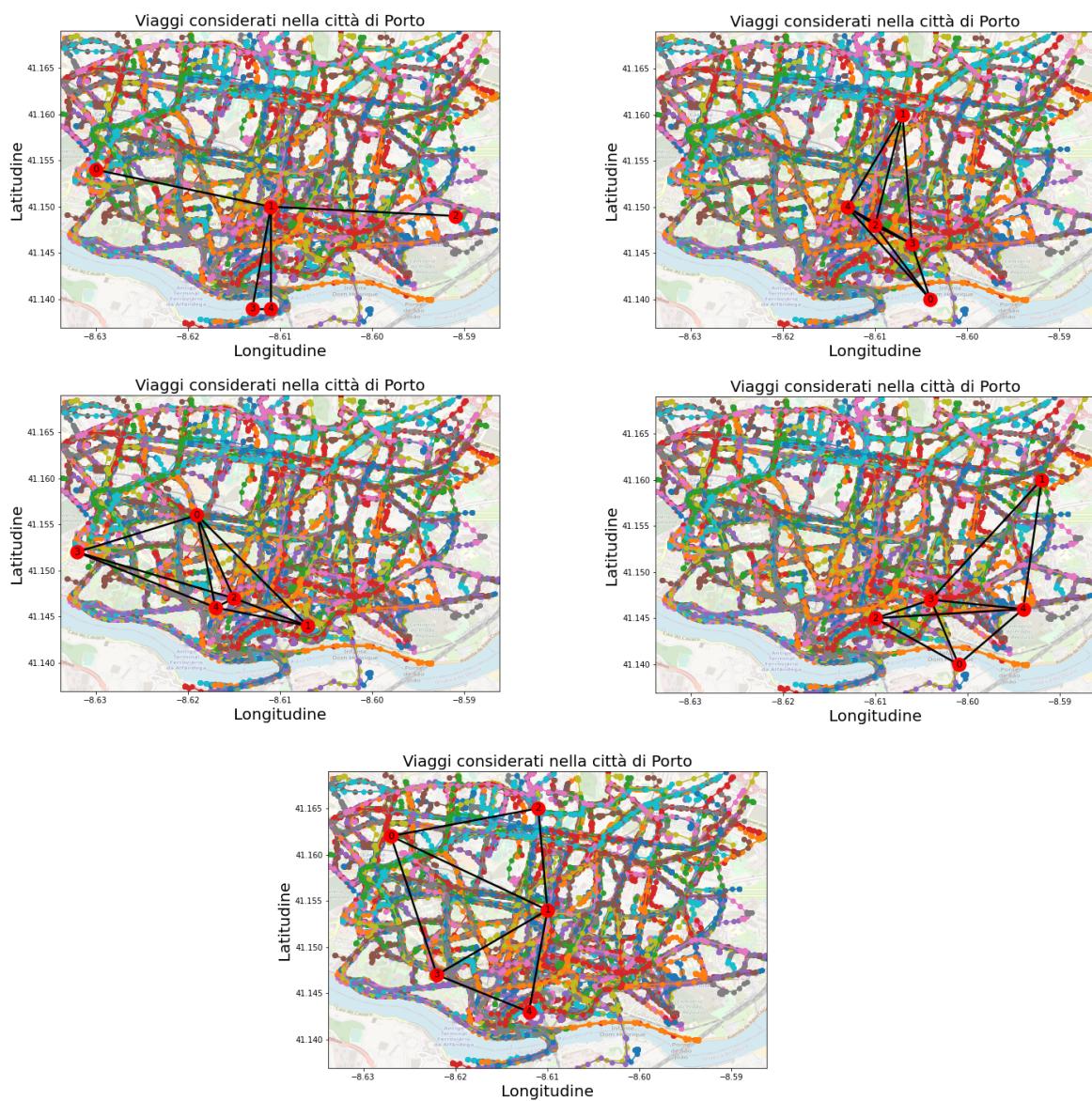


Figura 6.3: Scenari considerati con un numero di antenne pari a 5

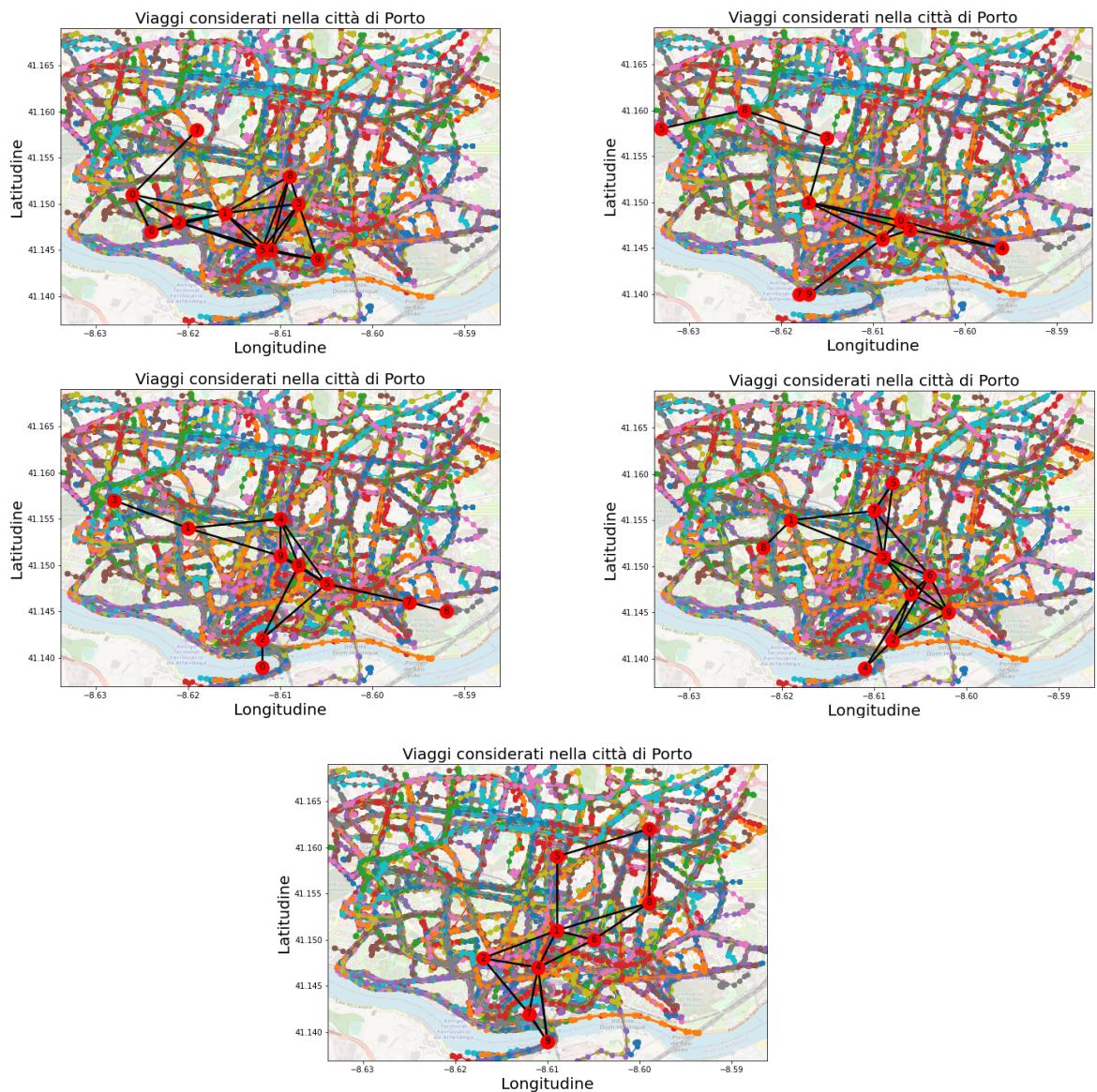


Figura 6.4: Scenari considerati con un numero di antenne pari a 10

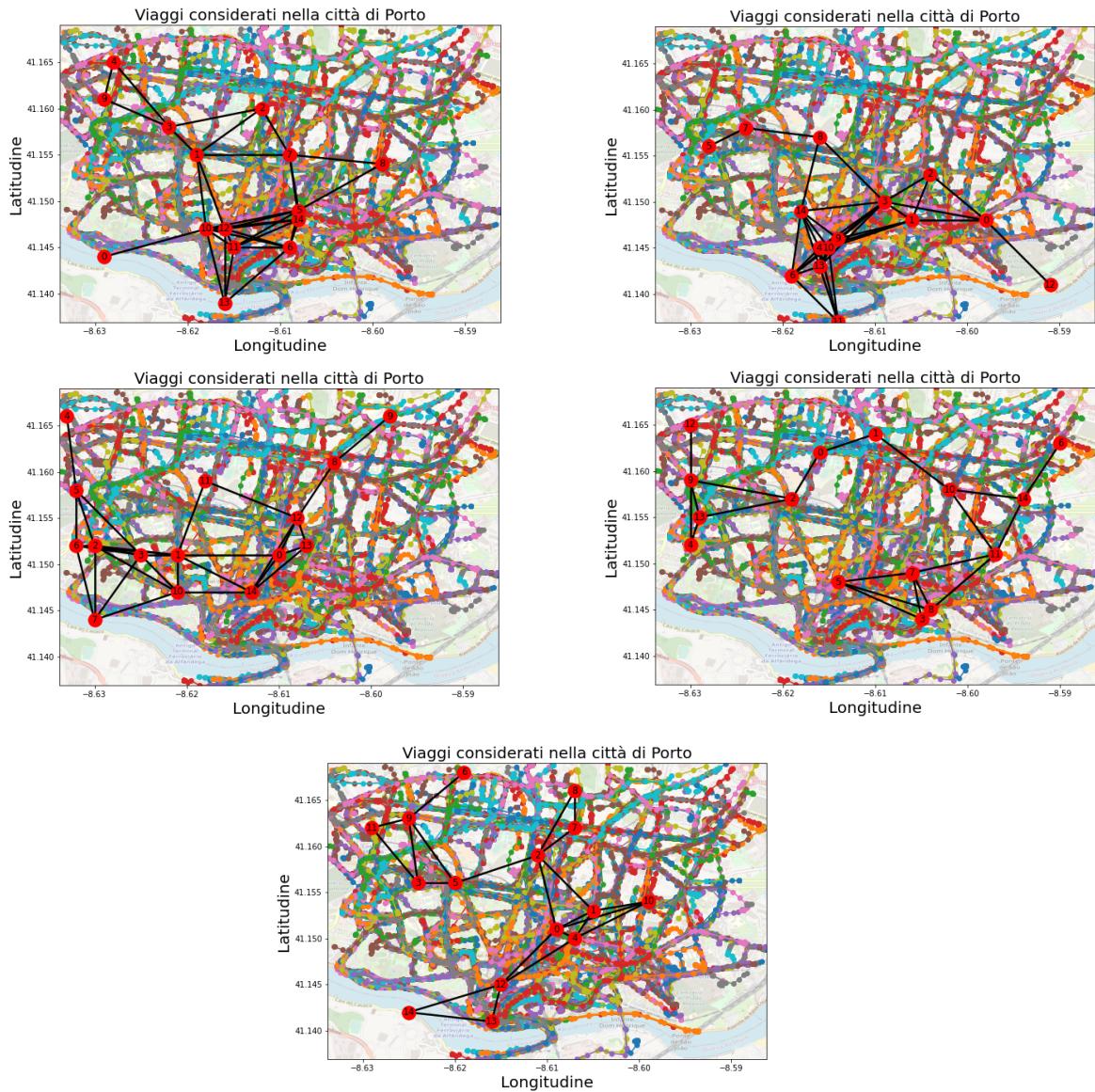


Figura 6.5: Scenari considerati con un numero di antenne pari a 15

Per ognuna delle tre sperimentazioni si è studiato l’andamento medio della percentuale di miglioramento delle performance predittive dei nodi aventi lo stesso numero di vicini in seguito all’utilizzo degli approcci proposti. La percentuale di miglioramento è calcolata attraverso la Formula 6.1 considerando il modello LSTM come modello baseline:

$$\frac{RMSE_{LSTM} - RMSE_{APPROCCIO PROPOSTO}}{RMSE_{LSTM}}. \quad (6.1)$$

Le figure 6.6, 6.7, 6.8 mostrano i box plot degli incrementi in percentuale delle performance dovute all’utilizzo degli approcci federati e distribuiti. Gli incrementi sono stati aggregati per nodo in base al numero di nodi vicini e, per ogni simulazione, tengono in considerazione i 5 diversi scenari.

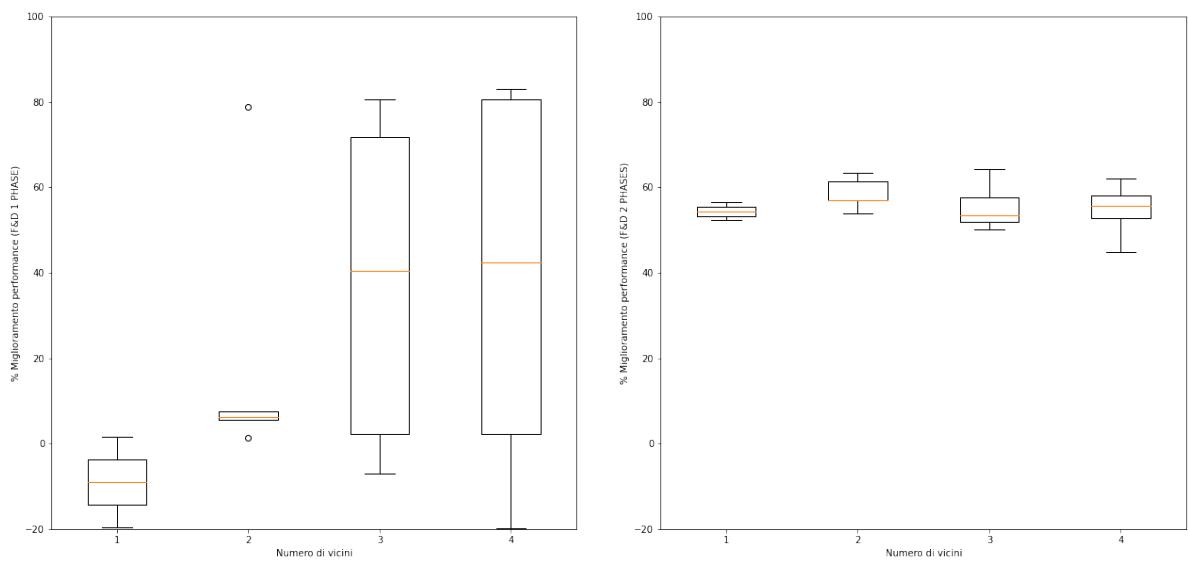


Figura 6.6: Miglioramento in percentuale delle performance dei modelli proposti al variare del numero di vicini per nodo con numero di antenne pari a 5

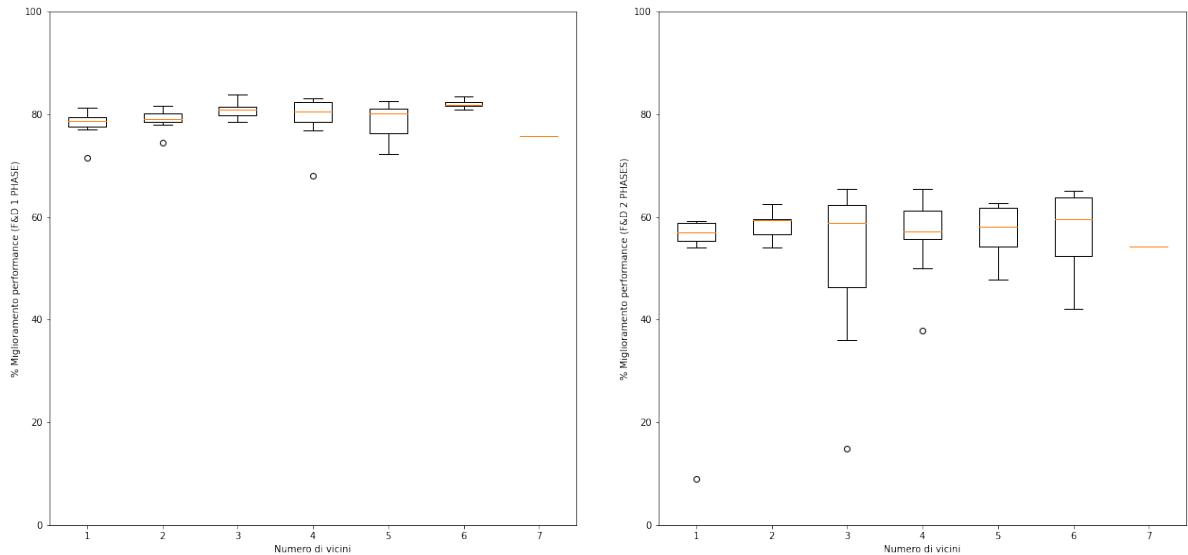


Figura 6.7: Miglioramento in percentuale delle performance dei modelli proposti al variare del numero di vicini per nodo con numero di antenne pari a 10

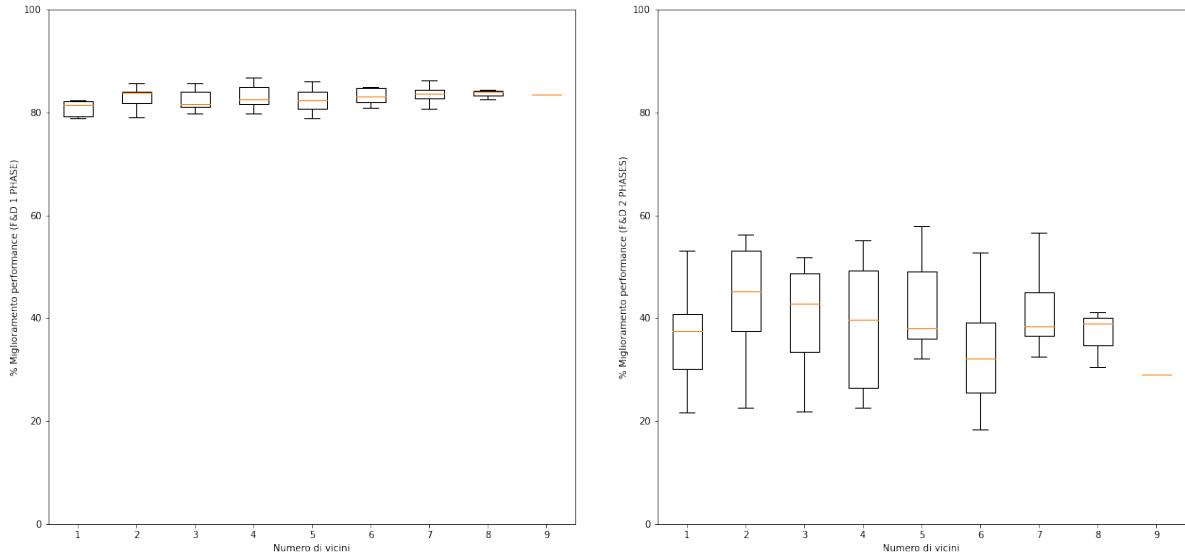


Figura 6.8: Miglioramento in percentuale delle performance dei modelli proposti al variare del numero di vicini per nodo con numero di antenne pari a 15

Al fine di comprendere come variano le performance degli approcci federati e distribuiti proposti al variare del numero di vicini per nodo, è importante evidenziare come in Figura 6.6, 6.7 e 6.8, la percentuale di miglioramento dovuta ad ambedue gli approcci raggiunge un punto di saturazione oltre il quale i nodi smettono di migliorare le proprie performance predittive e, in alcuni casi, queste ultime peggiorano. Tale fenomeno è simile a quanto descritto nel lavoro di Nguyen et al. e potrebbe essere spiegato dal fatto che a un maggior numero di nodi vicini corrisponde anche un maggior rumore nelle informazioni (pesi) scambiate. Essendo ogni antenna posizionata in un diverso punto della città ed essendo la quantità di traffico non omogenea (potrebbero esserci zone meno trafficate di altre), le informazioni apprese da un nodo potrebbero deteriorare le capacità predittive dei suoi vicini. Si suppone dunque che lo scambio di informazioni tra nodi vicini accresca le performance dei singoli nodi solo se questi ultimi sono posizionati in aree geografiche aventi pattern nel traffico simili. In caso contrario, avere un vicinato localizzato in aree il cui traffico è fortemente diverso dal proprio, basti pensare a celle telefoniche molto distanti tra loro, influisce negativamente sulle capacità predittive dei nodi.

In Figura 6.9 sono riassunte le performance predittive aggregate dei modelli descritti nel Capitolo 2 e nel Capitolo 5, ottenute in tutte le sperimentazioni svolte (15 scenari totali). Notiamo come il modello migliore risulti essere il modello DCRNN che essendo un approccio globale centralizzato, ha una visione completa di tutti i nodi all'interno del grafo e del modo in cui interagiscono tra loro e si influenzano l'uno con l'altro. Questo non avviene nel caso dell'Algoritmo 1 e dell'Algoritmo 2 che permettono a ogni nodo di conoscere solo una parte limitata dell'intera rete, ovvero il proprio vicinato. Il secondo miglior risultato è stato ottenuto dall'Algoritmo 1 che ha ottenuto delle performance medie superiori rispetto all'Algoritmo 2. Si noti però come l'approccio distribuito a una fase abbia un notevole numero di outlier e questo oltre a poter dipendere dall'intrinseca differenza nell'ammontare di traffico ricevuto da ogni antenna in base alla propria posizione, potrebbe anche essere dovuta alla componente casuale presente nell'Algoritmo 1 che porta le sue performance a dipendere fortemente dall'ordine e della frequenza (diversi a ogni iterazione)

di allenamento di ogni nodo. Notiamo come il modello Prophet pur essendo stato allenato sui dati locali di ogni nodo senza alcuna possibilità di sfruttare le informazioni dei vicini, abbia ottenuto performance nettamente superiori a quelle avute utilizzando il modello LSTM nelle medesime condizioni. Questo potrebbe essere dovuto alla minor dipendenza del modello Prophet dall'*hyperparameter tuning*⁷ rispetto al modello LSTM [51]. Si noti inoltre come il modello LSTM abbia ottenuto delle performance nettamente peggiori rispetto al medesimo modello dotato della stessa architettura ma allenato in maniera federata e distribuita. Questo risultato può essere spiegato dal fatto che il modello LSTM allenato solamente sullo storico dei dati del rispettivo nodo non sfrutti la correlazione delle variazioni del carico di lavoro dei nodi a distanza fisica ravvicinata.

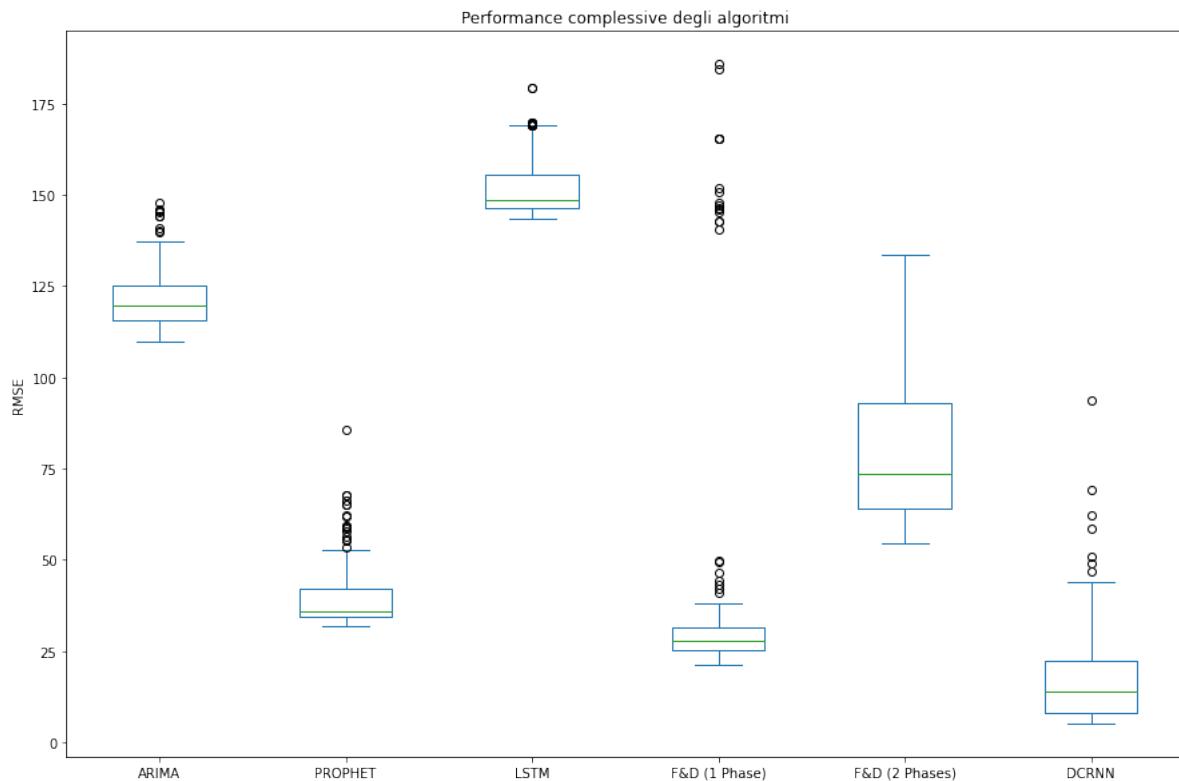


Figura 6.9: La figura mostra come il modello con le migliori performance medie globali sia il modello DCRNN, seguito dall’approccio federato e distribuito a una fase e dal modello Prophet.

La Figura 6.10 mostra invece un resoconto più dettagliato delle performance aggregate di ogni modello al variare del numero di nodi (antenne) nel grafo. Come descritto precedentemente, anche in questo caso emergono le performance superiori del modello DCRNN rispetto agli altri indipendentemente dal numero di nodi considerati. Gli approcci federati e distribuiti proposti nel Capitolo 5 si dimostrano essere maggiormente performanti rispetto ai modelli allenati in maniera isolata, tra cui Prophet, Arima e il modello LSTM.

⁷https://en.wikipedia.org/wiki/Hyperparameter_optimization

Grafico rappresentante le performance dei modelli al variare del numero di antenne nel grafo

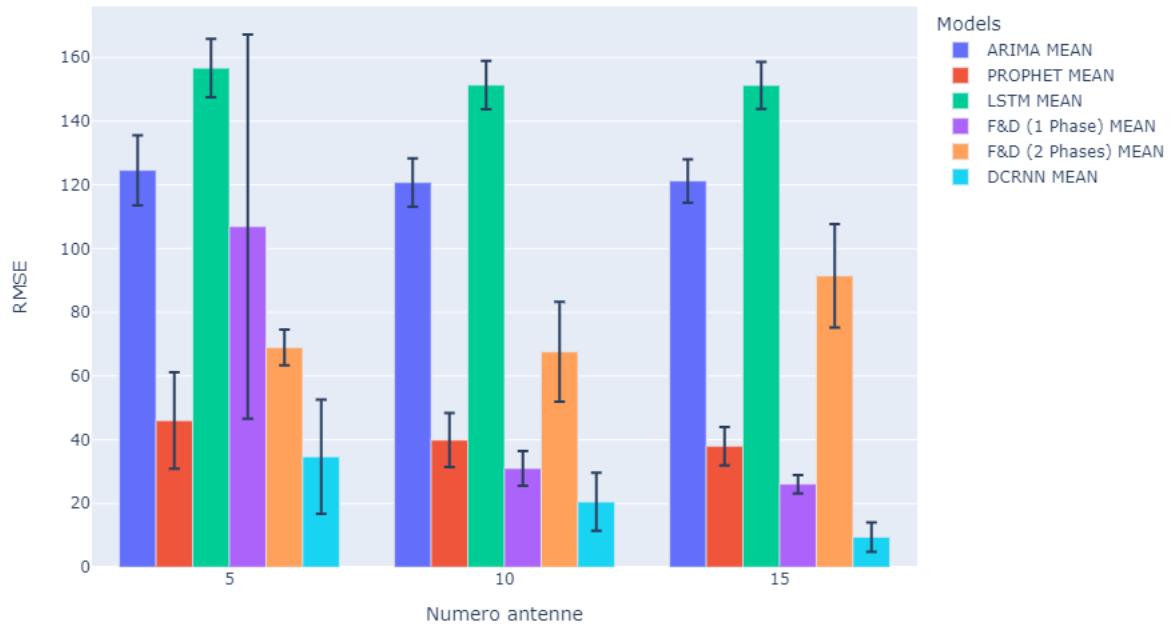


Figura 6.10: La figura mostra chiaramente come il modello DCRNN che ha una visione completa del sistema, ha ottenuto delle performance migliori dei modelli che non sfruttano le informazioni dei vicini. Subito dopo seguono i due approcci federati e distribuiti proposti in questo elaborato e infine il modello Prophet.

Capitolo 7

Conclusioni

L'incremento della diffusione dei dispositivi mobili e la sempre crescente dinamicità delle reti richiedono dei sistemi di apprendimento in grado di potersi adattare a contesti non statici e di sfruttare la collaborazione tra gli utenti. Nonostante il paradigma del Cloud Computing sia ad oggi molto diffuso per la realizzazione di piattaforme IoT, dipende fortemente dalla qualità della connessione a disposizione dei dispositivi connessi alla rete e soffre di tempi di latenza alti. Queste problematiche lo rendono poco adatto ad essere utilizzato in contesti in cui vi è una molteplicità di dispositivi connessi in comunicazione tra loro. In questo lavoro, sono stati proposti due approcci federati e distribuiti per la previsione della quantità di traffico in ingresso in una rete di celle telefoniche all'interno di un contesto urbano. Tali approcci permettono al modello LSTM di ogni cella di comunicare le informazioni apprese ai propri vicini sfruttando i pattern presenti nei dati relativi al traffico in ingresso registrato nelle antenne a lui vicine. Il primo approccio simula un contesto decentralizzato nella quale la comunicazione tra i nodi dei rispettivi pesi calcolati e il loro allenamento avviene seguendo un ordine diverso a ogni iterazione. Il secondo approccio invece si compone principalmente di due fasi in cui nella prima avviene lo scambio dei pesi in contemporanea tra tutti i nodi, mentre nella seconda fase ognuno di essi allena il proprio modello sui dati locali. Il fine ultimo di ambedue gli approcci è migliorare la capacità di ogni cella di prevedere con maggiore accuratezza il carico in ingresso in un breve periodo di tempo. I risultati sperimentali sono stati conformi alle aspettative in quanto hanno dimostrato che i metodi proposti raggiungono dei risultati assimilabili a quelli ottenuti nel lavoro di Nguyen et al. e migliori rispetto all'allenamento dei singoli modelli predittivi in maniera isolata. Ambedue gli approcci federati e distribuiti proposti portano inoltre notevoli vantaggi in termini di incremento nella privacy dei dati e un minore workload a carico della rete.

7.1 Sviluppi futuri

A partire dal lavoro svolto in questa tesi magistrale, sono diversi i suoi possibili sviluppi futuri come ad esempio:

- Progettare un protocollo di sincronizzazione locale tra nodi in modo tale da considerare la concorrenza tra i diversi nodi nell'eseguire la fase di comunicazione dei pesi e la fase di allenamento del proprio modello. Ambedue gli approcci studiati attualmen-

te si basano sull'assunzione di sincronicità tra i processi ma ciò raramente accade in contesti reali nei quali le particolari condizioni climatiche e di traffico mutabili nel tempo, potrebbero interferire con la corretta esecuzione delle fasi precedentemente citate.

- Progettare, applicare e analizzare le performance di nuovi modelli federati e distribuiti in grado di colmare il gap nelle performance tra i gli approcci proposti nel seguente lavoro di tesi e il modello DCRNN. Tale gap può essere infatti visto come indicatore del miglioramento ancora possibile.
- Applicare un algoritmo più avanzato di controllo del miglioramento delle performance predittive dei modelli che riduca il workload sulla rete riducendo le comunicazioni tra i nodi al minimo e solo quando necessarie a migliorare considerevolmente il modello. Un esempio di tale tipologia di algoritmo da considerare potrebbe essere il *Simulated annealing*¹.

¹https://en.wikipedia.org/wiki/Simulated_annealing

Bibliografia

- [1] Need of Cloud,Fog and Edge Computing. <https://forum.huawei.com/enterprise/en/discussion-post-need-of-cloud-fog-and-edge-computing/thread/748771-893>.
- [2] Sawsan Abdulrahman, Hanine Tout, Hakima Ould-Slimane, Azzam Mourad, Chamseddine Talhi, and Mohsen Guizani. A Survey on Federated Learning: The Journey From Centralized to Distributed On-Site Learning and Beyond. *IEEE Internet of Things Journal*, PP, October 2020. doi: 10.1109/JIOT.2020.3030072.
- [3] M. G. Sarwar Murshed, Christopher Murphy, Daqing Hou, Nazar Khan, Ganesh Ananthanarayanan, and Faraz Hussain. Machine Learning at the Network Edge: A Survey. *ACM Computing Surveys*, 54(8):1–37, November 2022. ISSN 0360-0300, 1557-7341. doi: 10.1145/3469029.
- [4] Yingda Xia, Dong Yang, Wenqi Li, Andriy Myronenko, Daguang Xu, Hirofumi Obinata, Hitoshi Mori, Peng An, Stephanie Harmon, Evrim Turkbey, Baris Turkbey, Bradford Wood, Francesca Patella, Elvira Stellato, Gianpaolo Carrafiello, Anna Ierardi, Alan Yuille, and Holger Roth. Auto-FedAvg: Learnable Federated Averaging for Multi-Institutional Medical Image Segmentation, April 2021.
- [5] Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition [Book]. <https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/>.
- [6] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting, February 2018.
- [7] Tarek Alskaf, Wouter Schram, Geert Litjens, and Wilfried van Sark. Smart Charging of Community Storage Units Using Markov Chains. January 2018. doi: 10.1109/ISGTEurope.2017.8260177.
- [8] Nacer Khalil, Mohamed Riduan Abid, Driss Benhaddou, and Michael Gerndt. Wireless sensors networks for Internet of Things. In *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 1–6, April 2014. doi: 10.1109/ISSNIP.2014.6827681.
- [9] Christos Stergiou, Kostas E. Psannis, Byung-Gyu Kim, and Brij Gupta. Secure integration of IoT and Cloud Computing. *Future Generation Computer Systems*, 78: 964–975, January 2018. ISSN 0167-739X. doi: 10.1016/j.future.2016.11.031.
- [10] Top 10 Advantages and Disadvantages of Cloud Computing [2022].

- [11] Blessen Varghese, Nan Wang, Sakil Barbhuiya, Peter Kilpatrick, and Dimitrios S. Nikolopoulos. Challenges and Opportunities in Edge Computing. In *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 20–26, November 2016. doi: 10.1109/SmartCloud.2016.18.
- [12] Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit Niyato, and Chunyan Miao. Federated Learning in Mobile Edge Networks: A Comprehensive Survey. *IEEE Communications Surveys & Tutorials*, 22(3):2031–2063, 2020. ISSN 1553-877X, 2373-745X. doi: 10.1109/COMST.2020.2986024.
- [13] Bart Custers, Alan M. Sears, Francien Dechesne, Ilina Georgieva, Tommaso Tani, and Simone van der Hof. *EU Personal Data Protection in Policy and Practice*, volume 29 of *Information Technology and Law Series*. T.M.C. Asser Press, The Hague, 2019. ISBN 978-94-6265-281-1 978-94-6265-282-8. doi: 10.1007/978-94-6265-282-8.
- [14] Ganesh Ananthanarayanan, Victor Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath Sivalingam, and Sudipta Sinha. Real-time Video Analytics: The killer app for edge computing. *IEEE Computer*, October 2017.
- [15] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data, February 2017.
- [16] Adrian Nilsson, Simon Smith, Gregor Ulm, Emil Gustavsson, and Mats Jirstrand. A Performance Evaluation of Federated Learning Algorithms. In *Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning*, DIDL ’18, pages 1–8, New York, NY, USA, December 2018. Association for Computing Machinery. ISBN 978-1-4503-6119-4. doi: 10.1145/3286490.3286559.
- [17] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Processing Magazine*, 37(3):50–60, May 2020. ISSN 1053-5888, 1558-0792. doi: 10.1109/MSP.2020.2975749.
- [18] Chanh Nguyen, Cristian Klein, and Erik Elmroth. Multivariate LSTM-Based Location-Aware Workload Prediction for Edge Data Centers. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 341–350, Larnaca, Cyprus, May 2019. IEEE. ISBN 978-1-72810-912-1. doi: 10.1109/CCGRID.2019.00048.
- [19] Peter Mell and Tim Grance. The NIST Definition of Cloud Computing. Technical Report NIST Special Publication (SP) 800-145, National Institute of Standards and Technology, September 2011.
- [20] Luis M. Vaquero and Luis Rodero-Merino. Finding your Way in the Fog: Towards a Comprehensive Definition of Fog Computing. *ACM SIGCOMM Computer Communication Review*, 44(5):27–32, October 2014. ISSN 0146-4833. doi: 10.1145/2677046.2677052.

- [21] National Institute of Standards and Technology. *Fog Computing Conceptual Model: NIST SP 500-325 & 500-291*. CreateSpace Independent Publishing Platform, North Charleston, SC, USA, 2018. ISBN 978-1-986596-95-4.
- [22] Umer Butt, Muhammad Mehmood, Syed Bilal Shah, Rashid Amin, M. Shaukat, Syed Raza, Doug Suh, and Md Piran. A Review of Machine Learning Algorithms for Cloud Computing Security. *Electronics*, 9:1379, August 2020. doi: 10.3390/electronics9091379.
- [23] Thang Le Duc, Rafael García Leiva, Paolo Casari, and Per-Olov Östberg. Machine Learning Methods for Reliable Resource Provisioning in Edge-Cloud Computing: A Survey. *ACM Computing Surveys*, 52(5):94:1–94:39, September 2019. ISSN 0360-0300. doi: 10.1145/3341145.
- [24] Substation Automation Protection & Control - Substation Automation Systems Manufacturer from Vadodara. <https://www.indiamart.com/abblimited-3882692/>.
- [25] Xiangfeng Dai, Irena Spasić, Bradley Meyer, Samuel Chapman, and Frederic Andres. Machine Learning on Mobile: An On-device Inference App for Skin Cancer Detection. In *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 301–305, June 2019. doi: 10.1109/FMEC.2019.8795362.
- [26] Thien Nguyen, Samuel Marchal, Markus Miettinen, Minh Dang, N. Asokan, and Ahmad-Reza Sadeghi. DIoT: A Crowdsourced Self-learning Approach for Detecting Compromised IoT Devices. April 2018. doi: 10.48550/arXiv.1804.07474.
- [27] Industrial internet of things. *Wikipedia*, September 2022.
- [28] Enmao Diao, Jie Ding, and Vahid Tarokh. HeteroFL: Computation and Communication Efficient Federated Learning for Heterogeneous Clients, December 2021.
- [29] Otkrist Gupta and Ramesh Raskar. Distributed learning of deep neural network over multiple agents, October 2018.
- [30] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K. Leung, Christian Makaya, Ting He, and Kevin Chan. Adaptive Federated Learning in Resource Constrained Edge Computing Systems, February 2019.
- [31] Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. *On-Device Federated Learning via Blockchain and Its Latency Analysis*. August 2018.
- [32] Federated Learning: Collaborative Machine Learning without Centralized Training Data.
- [33] Abebe Abeshu and Naveen Chilamkurti. Deep Learning: The Frontier for Distributed Attack Detection in Fog-to-Things Computing. *IEEE Communications Magazine*, 56(2):169–175, February 2018. ISSN 1558-1896. doi: 10.1109/MCOM.2018.1700332.

- [34] Ahsan Raza Khan, Anzar Mahmood, Awais Safdar, Zafar A. Khan, and Naveed Ahmed Khan. Load forecasting, dynamic pricing and DSM in smart grid: A review. *Renewable and Sustainable Energy Reviews*, 54:1311–1322, February 2016. ISSN 1364-0321. doi: 10.1016/j.rser.2015.10.117.
- [35] Yuris Mulya Saputra, Dinh Thai Hoang, Diep N. Nguyen, Eryk Dutkiewicz, Markus Dominik Mueck, and Srikathyayani Srikanteswara. Energy Demand Prediction with Federated Learning for Electric Vehicle Networks, September 2019.
- [36] Marco Savi and Fabrizio Olivadese. Short-Term Energy Consumption Forecasting at the Edge: A Federated Learning Approach. *IEEE Access*, 9:95949–95969, 2021. ISSN 2169-3536. doi: 10.1109/ACCESS.2021.3094089.
- [37] Emiliano Sisinni, Abusayeed Saifullah, Song Han, Ulf Jennehag, and Mikael Gidlund. Industrial Internet of Things: Challenges, Opportunities, and Directions. *IEEE Transactions on Industrial Informatics*, 14(11):4724–4734, November 2018. ISSN 1941-0050. doi: 10.1109/TII.2018.2852491.
- [38] Wei Zhang, Xiang Li, Hui Ma, Zhong Luo, and Xu Li. Federated learning for machinery fault diagnosis with dynamic validation and self-supervision. *Knowledge-Based Systems*, 213:106679, February 2021. ISSN 0950-7051. doi: 10.1016/j.knosys.2020.106679.
- [39] Boyi Liu, Lujia Wang, and Ming Liu. Lifelong Federated Reinforcement Learning: A Learning Architecture for Navigation in Cloud Robotic Systems. *IEEE Robotics and Automation Letters*, 4(4):4555–4562, October 2019. ISSN 2377-3766, 2377-3774. doi: 10.1109/LRA.2019.2931179.
- [40] Sean J. Taylor and Benjamin Letham. Forecasting at scale. Technical Report e3190v2, PeerJ Inc., September 2017.
- [41] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, December 2014.
- [42] Diffusion Process - an overview | ScienceDirect Topics. <https://www.sciencedirect.com/topics/mathematics/diffusion-process>.
- [43] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.
- [44] WeiWei Miao, Zeng Zeng, Mingxuan Zhang, Siping Quan, Zhen Zhang, Shihao Li, Li Zhang, and Qi Sun. Workload Prediction in Edge Computing based on Graph Neural Network. In *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 1663–1666, New York City, NY, USA, September 2021. IEEE. ISBN 978-1-66543-574-1. doi: 10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00223.

- [45] Lei Chen, Weiwen Zhang, and Haiming Ye. Accurate workload prediction for edge data centers: Savitzky-Golay filter, CNN and BiLSTM with attention mechanism. *Applied Intelligence*, February 2022. ISSN 1573-7497. doi: 10.1007/s10489-021-03110-x.
- [46] Alibaba Cluster Trace Program. Alibaba, April 2022.
- [47] Jerzy Letkowski. Developing Poisson probability distribution applications in a cloud. *Journal of Case Research in Business and Economics*, Volume 5, December 2014.
- [48] Charles Gouin-Vallerand and Neila Mezghani. An analysis of the transitions between mobile application usages based on Markov chains. September 2014. doi: 10.1145/2638728.2641700.
- [49] Processo markoviano. *Wikipedia*, May 2022.
- [50] proprietà di Markov in "Enciclopedia della Scienza e della Tecnica". [https://www.treccani.it/enciclopedia/proprietà-di-markov_\(Enciclopedia-della-Scienza-e-della-Tecnica\).](https://www.treccani.it/enciclopedia/proprietà-di-markov_(Enciclopedia-della-Scienza-e-della-Tecnica).)
- [51] Konstantin Kutzkov. ARIMA vs Prophet vs LSTM for Time Series Prediction. <https://neptune.ai/blog/arima-vs-prophet-vs-lstm>, January 2022.