ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Un framework per la predizione del contesto utente

Relatore:
Dott.
Luca Bedogni

Presentata da:
Nicola Buono

Sessione III
Anno Accademico 2017/2018

Abstract

Il seguente lavoro di tesi, vuole mostrare come la grande mole di dati, definita *Big Data*, generata dai numerosi dispositivi tecnologici creati negli ultimi anni (*Smartphone*, dispositivi *IoT*, etc.) e dalle varie *web application* (*social network*, servizi di messagistica, servizi di posta elettronica, etc.), possa essere sfruttata per ottenere informazioni relative alla vita privata degli utenti. Infatti, nonostante nella letteratura siano presenti una lunga serie di lavori su come utilizzare questi dati in modo efficiente, al fine di creare applicazioni in grado di fornire servizi innovativi basati sulle preferenze dei singoli utenti (applicazioni *context-aware*), l'ambito relativo al rapporto tra le operazioni di raccolta e analisi delle informazioni e la *privacy* degli singoli utenti resta largamente inesplorato.

Inizialmente introdurremo i principali concetti che ci saranno utili per comprendere al meglio l'argomento trattato, per poi mostrare il modo in cui, avendo a disposizione una raccolta di dati, sia possibile risalire a un'ampia serie di informazioni di varia natura relative ai singoli utenti considerati.

Descriveremo dunque il processo di creazione di un *framework* ideato con l'obiettivo di inferire varie informazioni sulla base di una serie di *dataset* relativi ad un gruppo di persone, che hanno acconsetito al trattamento dei propri dati personali per lo scopo di questo progetto.

Infine, presenteremo e analizzeremo i risultati ottenuti al variare di una serie di parametri.

Keyword: *privacy, Big Data, IoT, Context-Aware computing*

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

In seguito ai progressi ottenuti nel campo dell'Informatica negli ultimi anni, l'intero pianeta è stato investito da un'autentica rivoluzione tecnologica. Il mondo di *Internet* ha profondamente rivoluzionato la vita di ognuno di noi, mettendo a disposizione strumenti per comunicare con persone sparse sull'intero globo terrestre e fornendo una moltitudine di informazioni provenienti da ogni zona del pianeta. Inoltre, con la diffusione degli *smartphone* e la nascita del *IoT (Internet of Things)*, i dispositivi informatici sono stati integrati praticamente ovunque, contribuendo alla creazione di numerose innovazioni. Un ruolo fondamentale in questo processo di "interconnessione mondiale" è svolto sicuramente dai *social network*, piattaforme web in cui le persone possono comunicare, condividere foto, video e interessi comuni. Tra i più famosi *social network* troviamo *Facebook* (2.2 miliardi di utenti attivi), *Instagram* (1 miliardo di utenti attivi) e *Twitter*(330 milioni di utenti attivi). [1]

L'insieme di questi elementi genera una quantità di dati enorme, per cui si viene a creare la necessita di uno spazio di archiviazione e di tecniche di analisi adeguate al fine di memorizzare e sfruttare al meglio questa grane mole di informazioni. È così che nasce il settore dei *Big Data*, il quale si occupa delle operazioni precedentemente citate.

Questo sviluppo rapido e profondo, ha apportato numerosi benefici in vari settori della società, e in generale nella vita quotidiana di ognuno di noi, ma allo stesso tempo ha contribuito alla nascita di numerose problematiche legate alla *privacy*. Infatti, è molto importante che le informazioni raccolte dai *social network*, dai servizi di posta elettronica (*Gmail, Yahoo*, etc.) o di messagisti-

[1] https://juliusdesign.net/28700/lo-stato-degli-utenti-attivi-e-registrati-sui-social-media-in-italia-e-mondo-2015

ca (*Whatsapp, Telegram*, etc.), ma in generale da qualsiasi dispositivo o *web application*, vengano trattate nel massimo rispetto della *privacy* degli utenti.

Oltretutto, capita spesso che un utente non sia a conoscenza dei dati che sta effettivamente fornendo ad un servizio durante lo svolgimento di operazioni di varia natura.

È così che, durante la fase di analisi dei *Big Data*, possono essere ottenute informazioni sensibili di un utente o di un'organizzazione, la cui scoperta può produrre conseguenze molti gravi per l'utente o l'organizzazione stessa. Dunque, nonostante i progressi negli algoritmi di crittografia e nelle techiche di *PPDM (Privacy Preserving Data Mining)*, siamo ancora ben lontani da un concetto assoluto di sicurezza legata all'ambito informatico.

Il punto di partenza di questa tesi è costituito dal gran numero di informazioni che un individuo genera quotidianamente mediante una moltitudine di servizi. Possiamo dividere questi dati in due categorie principali:

- informazioni fornite in modo esplicito, come una registrazione presso un luogo su un *social network*, la condivisione di una foto o del percorso effettuato durante una corsa al parco
- informazioni fornite in modo implicito, come ad esempio i dati registrati mediante la connessione alla rete *GSM*

Capita spesso, in modo errato, di sottovalutare il potenziale che risiede in questi dati. Mostreremo quindi in che modo, essendo a conoscenza e combinando queste informazioni, si possa creare una vera e propria profilazione di un utente, fino a risalire al suo stile di vita e alle sue abitudini. Inoltre, avendo a disposizione informazioni relative a più utenti, è possibile evienziare le differenze nei loro stili di vita.

Costruiremo quindi un modello probabilistico, scritto nel linguaggio di programmazione *Python*, in grado di predire, in diversi casi con elevata precisione, un'ampia serie di informazioni legate agli spostamenti e all'attività svolta da un utente, mostrando come la routine quotidiana di una persona sia altamente predicibile.

Il problema principale che si vuole evidenziare in questa tesi risiede nel fatto che un eventuale parte maliziosa, se in possesso di un *dataset* simile a quello utilizzato per questo lavoro, potrebbe essere in grado di costruire il medesimo modello probabilistico. Ne conseguirebbe un'evidente violazione della

privacy che, nella maggior parte dei casi, arreccherebbe conseguenze devastanti nella vita privata di una persona. Risolvere questo problema inoltre risulterebbe molto difficile, in quanto alcuni aspetti della vita quotidiana di una persona sarebbero molto difficili da modificare.

L'obiettivo principale a cui ambisce questo progetto dunque, oltre a mostrare l'alta predicibilità della ruotine quotidiana mediante l'analisi dei risultati ottenuti, è quello di sensibilizzare gli utenti affinchè siano a conoscenza dei rischi relativi alla *privacy* che può comportare l'eccessiva quantità di dati forniti alle varie piattaforme web.

Inoltre, si vuole mostrare il valore reale e il potenziale che i dati stessi costituiscono.

Il capitolo 2 descrive lo stato dell'arte utile per comprendere al meglio il lavoro svolto, nel capitolo 3 viene invece descritta nei particolari la fase di costruzione e valutazione del modello probabilistico, mentre il capitolo 4 presenta un'analisi dei risultati ottenuti.

Capitolo 2

Stato dell'arte

2.1 Sviluppo tecnologico

Gli *smartphone* sono senza dubbio la tecnologia che è stata adottata e si è diffusa più velocemente rispetto a qualsiasi altro prodotto tecnlogico nella storia dell'essere umano. Basti pensare che nel 2016, il numero di *smartphone users* era circa 2.1 miliardi (quasi il doppio rispetto al numero di *desktop users*), mentre nel 2019 il numero totale potrebbe superare i 5 miliardi [1].

Al fine di trarre profitto dalla continua espansione di questo mercato, le principali aziende del settore come *Google, Microsoft e Apple*, hanno sviluppato i propri sistemi operativi in ambiente *mobile*, fornendo una vasta serie di *APIs*, affinchè gli sviluppatori *software* fossero in grado di creare nuove e innovative applicazioni. La principale potenzialità di questi dipositivi, è data soprattutto dalla presenza dei numerosi sensori integrati (*GPS*, accellerometro, giroscopio, etc...) che permettono di rilevare e produrre informazioni di diverse tipologie. Inoltre la loro dimensione ridotta rispetto ai *PC*, permette ad un utente di trasportare il proprio *smartphone* ovunque esso voglia. In questo modo, ogni individuo dotato di un dispositivo di questo tipo, ha la possibilità, mediante l'accesso alla rete mobile, di navigare sul web, leggere le email o effettuare altre operazioni durante l'intero corso della giornata.

Anche la rete mobile, al fine di adattarsi all'aumento esponenziale dei dispositivi (e quindi del volume del traffico di rete), ha subito negli ultimi anni un processo di crescita e sviluppo notevole, fino ad arrivare alla sua quinta generazione (5G).

[1] https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

È stato stimato che nel 2020 saranno più di 50 miliardi i dispositivi connessi, e che il volume del traffico di rete sarà circa 1000 volte più ampio rispetto a quello attuale [14].

Fornire un servizio stabile e ottimale ad un numero così ampio di utenti è sicuramente una sfida non semplice, soprattutto perchè la quantità di informazioni che devono essere gestite al fine di ottenere un'alta *QoE (Quality of Experience)* è molto ampia.

Il settore che si occupa di gestire, analizzare e trarre vantaggio da questa grande mole di dati, relativa non solo alle informazioni prodotte dai dispositivi mobile ma anche dai dispositivi *IoT (Internet of things)* o da qualsiasi web application (*social network*, servizi di posta elettronica, etc...) è quello dei *Big data*.

2.2 Big Data

Col termine *Big Data* si intende una quantità di dati talmente grande e complessa da richiedere la definizione di nuovi strumenti e metodologie per estrapolare, gestire e processare informazioni entro un tempo ragionevole. Non esiste una quantità di riferimento, in quanto all'aumentare della potenza di calcolo dei dispositivi informatici, corrisponde un'aumento della dimensione dei dataset. [2]

Un concetto fondamentale nell'ambito dei *Big Data* è quello di *data mining*, con il quale si indica l'insieme delle tecniche e metodologie che hanno per oggetto l'estrazione di informazioni utili da grandi quantità di dati attraverso metodi automatici o semi-automatici e l'utilizzo scientifico, aziendale/industriale o operativo delle stesse. [3]
La seguente figura rappresenta la quantità media di dati generati al minuto. [4]

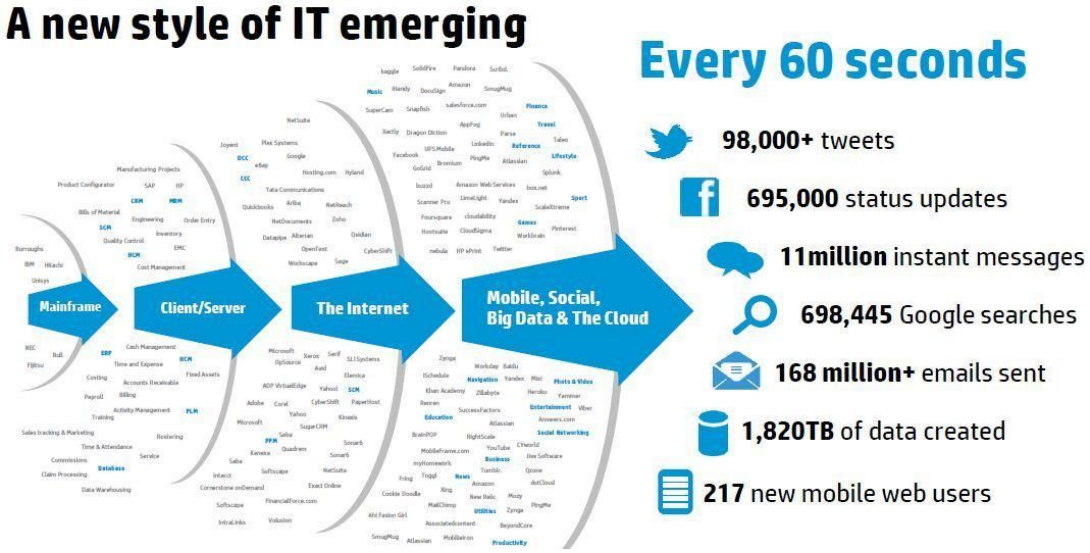

Figura 2.1: Quantità media di dati generati al minuto

[2] https://it.wikipedia.org/wiki/Big_data

[3] https://it.wikipedia.org/wiki/Data_mining

[4] Fonte: https://www.datasciencecentral.com/profiles/blogs/basic-understanding-of-big-data-what-is-this-and-how-it-is-going

Modello dei Big Data

Nel 2001, l'analista Doug Laney pubblicò uno studio in cui definiva il modello di crescita di questa grande mole di dati come tridimensionale (modello delle 3V), ritenuto ancora valido nonostante sia stato ulteriormente esteso. Il modello attuale è noto come Modello delle 5V, ed è composto dalle seguenti features:

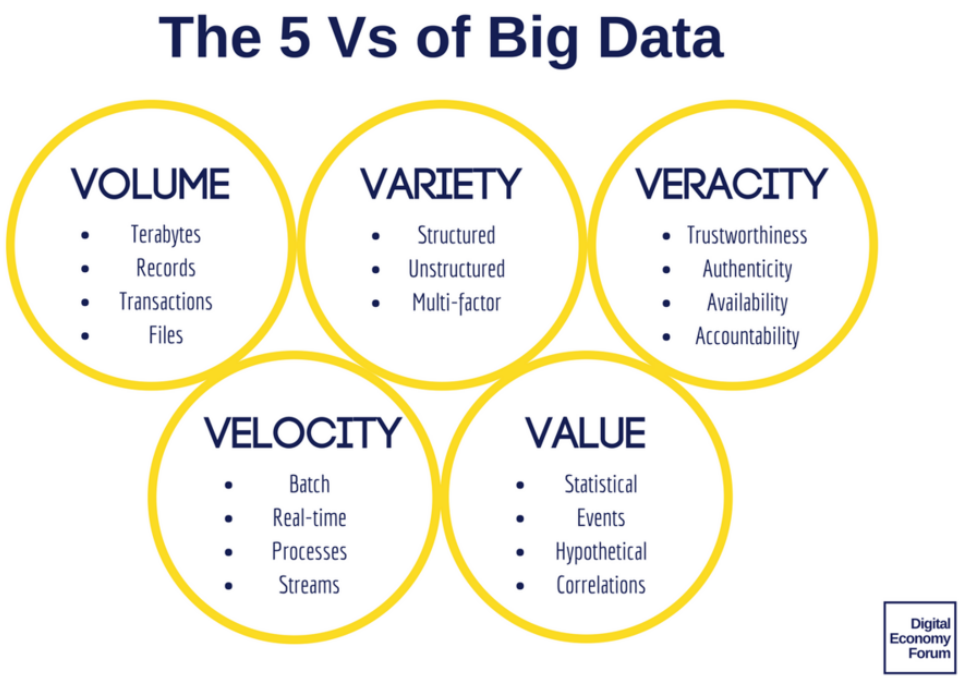

Figura 2.2: Modello dei Big Data. Fonte: https://digitaleconomyforum.org/chapter-2-how-your-digital-footprint-generates-big-data

1. ***Volume***: riferito alla quantità di dati (strutturati e non) prodotti ogni secondo. Tali dati vengono generati da sorgenti eterogenee quali: sensori, log, eventi, email, social media e database tradizionali. È stato stimato che il volume raddoppia circa ogni 40 mesi

2. ***Varietà***: è riferito alle differenti tipologie di dati che vengono generati, collezionati e utilizzati al fine di ottenere analisi più accurate. vengono presi in considerazione oltre ai dati strutturati, anche quelli semi-struturati e non strutturati.

Analizziamone in breve le differenze:

a) dati strutturati: sono le informazioni che solitamente si trovano nei *database relazionali*, infatti vengono definiti anche *dati relazionali*. Possono essere gestiti abbastanza facilmente .

b) dati semi-strutturati: sono dati non presenti nei database relazionali, ma caratterizzati da alcune proprietà che li rendono semplici da anlizzare. Fanno parte di questa categoria,ad esempio, i documenti *XML* e i database *NoSQL*.

c) dati non strutturati: dati che non appartengono alle precedenti categorie, non facilmente interpretabili da un elaboratore. Possono essere considerati dati di questo tipo i log dei *web server* o i log dei *firewall*, In quanto sono informazioni espresse in linguaggio naturale.

3. ***Velocità***: velocità con cui i nuovi dati vengono generati. È un fattore spesso considerato di maggiore importanza rispetto al volume. Infatti, nella maggior parte dei casi, è preferibile avere una quantità di dati limitata in *real time*, rispetto a una mole maggiore di dati ad una bassa velocità.

4. ***Veridicità***: considerando la varietà dei dati e la velocità alla quale tali dati possono variare, è molto importante assegnare ad essi un indice di veridicità su cui si basano le analisi, in modo da avere una misura dell'affidabilità. È evidente infatti, che se i dati alla base delle analisi sono poco accurati, i risultati finali non saranno ottimali.

5. ***Valore***: riferito alla capacità di trasformare i dati in valore. Infatti un progetto *Big Data* necessita di investimenti per la raccolta granulare dei dati e la loro analisi.

2.3 Internet of Things (IoT)

Con il termine *IoT* si intende una rete composta da dispositivi elettronici, oggetti, animali o persone dotati di un identificativo unico (*UIDs*) e dell'abilità di trasferire dati tramite la rete *Internet* senza richiedere l'interazione *human-to-human* o *human-to-computer*. La connettività a *Internet* viene quindi estesa oltre i classici device come *pc, laptop, smartphone* e *tablet*, fino a una moltitudine di oggetti differenti. I requisiti fondamentali relativi ad un dispositivo *IoT* sono:

1. possibilità di connettersi alla rete *Internet*;
2. possibilità di trasmettere e ricevere dati;

Altra caratteristica molto importante, è che oggetti di questo tipo possono essere monitorati e controllati da remoto. Esempi di *device IoT*, possono essere un "frigorifero intelligente" che notifica la scadenza di un prodotto al suo interno, o una macchina del caffè che in base alla sveglia impostata sullo *smartphone*, è in grado di attivarsi in automatico e far trovare il caffè pronto al risveglio. Ancora, in Svizzera sono presenti dei semafori intelligenti, che diventano verdi quando si accorgono che una macchina si trova nei pressi del semaforo e dall'altra parte della strada non è in transito nessun veicolo.

I principali campi di applicazione dell'*IoT* sono:

- casa, *smart home*, domotica;
- edifici intelligenti, *smart building, building automation*;
- monitoraggio in ambito industriale, Robotica, Robotica collaborativa;
- industria automobilistica, *automotive, self driving car*;
- *smart healt*, sanità, ambito biomedico;
- ambiti della telemetria;
- ambiti di sorveglianza e sicurezza;
- *smart city,smart mobility*
- forme di *digital payment* innovative

Dispositivi di questo tipo, forniscono la possibilità di collegarsi a *software* di analisi (ad esempio *Google Universal Analytics*) e in questo modo trasmettere dati ed informazioni dalla vita reale direttamente agli elaboratori, aprendo la strada ai *Big Data*. Infatti, la nascita dell' *IoT* ha contribuito notevolmente alla crescita del volume e della varietà di quest'ultimi. È stato stimato che entro il 2020 si arriverà a oltre 25 miliardi di apparati *IoT*, e al crescere del numero di *device*, ovviamente cresce anche la mole di dati che dovranno essere gestiti.

2.4 Context-aware computing

Grazie ai *Big Data* e alle informazioni generate dai sensori integrati negli *smartphone* (giroscopio, accellerometro, *GPS*, etc...) e dai dispositivi *IoT*, negli ultimi anni si sono diffuse notevolmente le applicazioni basate sul paradigma di *context-aware computing*, ovvero applicazioni in grado di effettuare operazioni in modo autonomo in base al contesto rilevato. In questo caso, il contesto può essere definito come come *"qualsiasi informazione che può essere utilizzata per caratterizzare la situazione di un'entità. Un'entità è una persona, un luogo, o un oggetto che è considerato rilevante nell'interazione tra l'utente e l'applicazione, inclusi l'utente e l'applicazione stessi" [13].* Un'applicazione di questo tipo dunque, è in grado di sfruttare le numerose informazioni a disposizione (sia quelle *real-time* che quelle storiche), al fine di fornire un'esperienza d'uso unica basata sulle preferenze dell'utente, ottimizzare le *performance*, e fornire la migliore *QoE*.

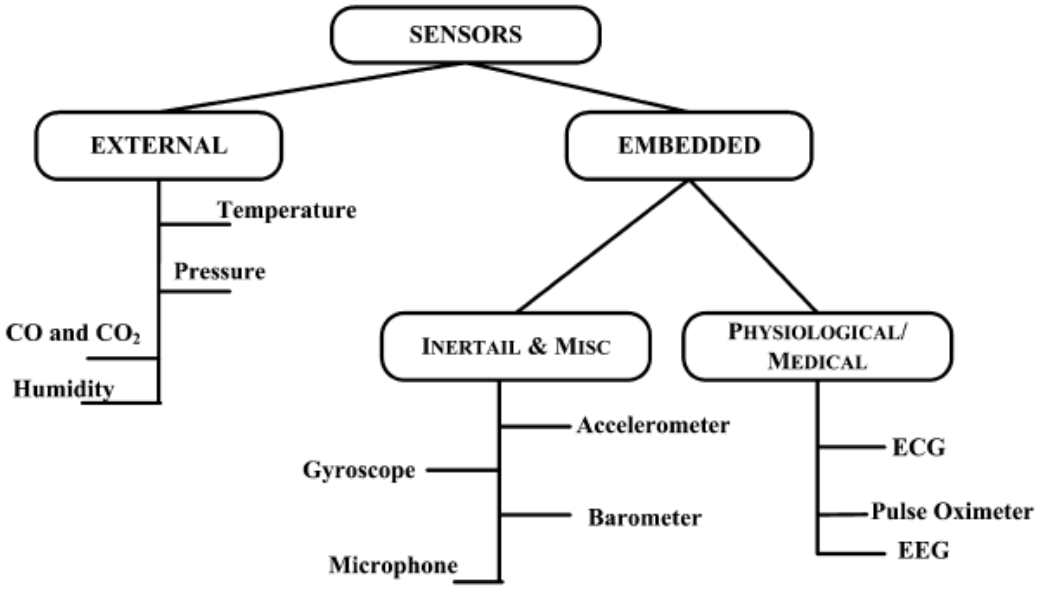

Figura 2.3: Fetching dei dati tramite sensori. Fonte: [15]

Ciclo di vita del contesto

Andiamo ora ad analizzare le varie fasi che costituiscono il ciclo di vita del contesto e che ne denotano l'evoluzione, partendo dai dati grezzi che necessitano di essere raffinati al fine di fornrie informazioni significative.

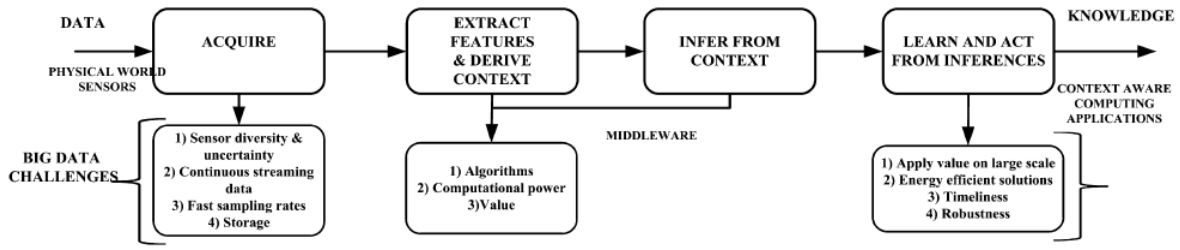

Figura 2.4: Context life cycle. Fonte: [15]

1) Data acquisition: i dati vengono aquisiti da vari sensori utilizzati per misurare una quantità fisica e convertirla in un segnale digitale processabile e memorizzabile dall'elaboratore .

2) data awareness: costituisce la fase di preprocessing e di identificazione del contesto. A causa della presenza di "rumore (dati fittizi)", errori dei sensori e *"motion artifacts"* è necessario un lavoro di *preprocessing/filtering* dei dati grezzi. Tecniche spesso adottate sono quelle di *power spectral density (PSD)*, o *Fast Fourier trasforms (FFT)*. Inoltre quando i dati vengono ricavati da numerosi sensori di tipo *wearable*, è richiesto anche un'operazione di normalizzazione e sincronizzazione dei dati.

3) Data analytics: inferire informazioni utili dai dati contestuali. Generalmente è molto utile creare un'astrazione dei dati grezzi e costruire un modello che permetta di ricavare informazioni di rilevanza. L'obiettivo di questa fase è quello di identificare le *features* che ci permettano di apprendere le caratteristiche principali di un *dataset*. Le *features estratte*, forniscono una rappresentazione significativa dei dati elaborati dai sensori, e ci permettono di formulare una relazione tra i dati grezzi e la conoscenza necessaria per il processo decisionale.

2.5 Privacy, Big Data e Context-awareness

Come collezionare, analizzare e processare questa grande mole di dati fornendo una *QoE* di alto livello, tutelando allo stesso tempo la *privacy* dell'utente finale, costitusice una delle maggiori problematiche per gli analisti e gli sviluppatori *software*.

Infatti, nonostante il *data mining* occupi un ruolo fondamentale in molte applicazioni, la *privacy* individuale di un utente può essere violata a causa dell'accesso non autorizzato a dati personali, o a causa dell'utilizzo di questi dati per scopi non inerenti a quelli per cui sono stati collezionati.

Bisogna tener presente che lo smarrimento o l'uso improprio di dati sensibili può arrecare danni molto gravi alla persona o all'organizzazione a cui questi dati appartengono. È molto comune infatti che un *dataset* includa informazioni sensibili degli utenti, e spesso anche le modalità con cui queste informazioni vengono estrapolate non siano del tutto trasparenti al soggetto considerato. Oltretutto, non sempre un utente è consapevole di quante e quali informazioni sta fornendo durante lo svolgimento di determinate operazioni [14].

Privacy Preserving Data Mining (PPDM)

Lo sviluppo di tecniche sempre più avanzate di *data mining* costituisce una seria minaccia alla sicurezza dei dati sensibili degli utenti. Proprio per questo motivo, uno dei settori emergenti negli ultimi anni è quello conosciuto come *Privacy Preserving Data Mining (PPDM)*. L'idea alla base del *PPDM*, è quella di modificare i dati al fine di utilizzare algoritmi di *data mining* in modo efficiente senza compromettere la sicurezza delle informazioni sensibili contenute dei dati.

Il *PPDM* è basato su due principi fondamentali:

1. i dati grezzi relativi a informazioni sensibili dell'utente, come il numero di carta d'identità, codice fiscale o numero di cellulare, non dovrebbero essere utilizzati direttamente nelle operazioni di *mining*;

2. I risultati delle operazioni di *mining* che potrebbero costituire una violazione della *privacy* dell'utente dovrebbero essere esclusi.

È importante, infine, evidenziare che problemi legati alla *privacy* possono verificarsi durante tutte le varie fasi del processo di *data mining*.

2.6 Lavori correlati

Nonostante *Big Data e Context-awareness* siano ambiti relativamente recenti, la letteratura prodotta negli ultimi anni è molto ampia. Oltre alla nascita di numerose innovazioni, moltissimi studiosi hanno cercato di analizzare i benefici in termini di *performace* o le varie problematiche relative alla *privacy* venutesi a creare in seguito alla notevole espansione di queste due tematiche.

2.6.1 Creazione di nuovi servizi

Il *Context-aware computing* ha permesso lo sviluppo di un'ampia serie di applicazioni innovative. In seguito al notevole aumento dei veicoli presenti in circolazione, le congestioni stradali sono sempre più frequenti. Vi è dunque, la necessità di progettare nuovi sistemi al fine di rendere efficiente la rete dei trasporti e migliorare in generale la vita quotidiana dei cittadini. Inoltre ottenere informazioni sulla mobilità urbana, ci permette di comprendere le dinamiche economiche, sociali e demografiche di una città. Nell'articolo [3], viene mostrato come sfruttare un *dataset* di dominio pubblico (in particolare i dati di *OpenStreetMap*) per effettuare una stima della mobilità urbana della città di Bologna, con dei costi relativamente bassi e un impatto ambientale minimo. La stima sulla mobilità, viene effettuata costruendo una matrice origine-destinazione, che descrive i vari travel patterns mostrando il numero di viaggi effettuati tra le varie zone della città. Terminata la costruzione della *ODM (Origin-Destination Matrix)*, ulteriori open data provenienti da diverse sorgenti , sono stati utilizzati per validare il modello e mostrarne l'effettiva validità.

Oltre ad effettuare una stima della mobilità urbana, questo studio mostra la potenzialità offerta dagli open data nel contesto delle *smart cities*.

Nel 2016, viene pubblicato il lavoro [2] di Luca Bedogni, Marco Di Felice e Luciano Bononi, in cui viene descritta un'applicazione *Context-aware android* in grado di identificare la modalità di trasporto corrente (biciletta, auto, treno, etc.) con cui si sta spostando il possessore dello *smartphone*. Un servizio di questo tipo, può essere molto utile ad una moltitudine di sistemi *mobility-aware*. La maggior parte delle applicazioni di questo tipo utilizzano i dati forniti dal

GPS e tecniche di *machine learning*, senza tener conto delle risorse limitate di uno *smartphone*. Inoltre, bisogna tener conto anche dell'impatto prodotto sulla durata della batteria, oltre al fatto che in alcuni luoghi il *GPS* può non essere disponibile.

Bisogna dunque trovare un compromesso tra la precisione del processo di rilevazione e il consumo energetico.

La principale innovazione proposta da [2], deriva dal fatto che il *framework* fa leva sui dati generati dai sensori integrati nel dispositivo (accellerometro/giroscopio), in seguito processati tramite un algoritmo di classificazione, proponendo un servizio che è in grado di adattarsi dinamicamente alle capacità computazionali del dispositivo, preservando allo tesso tempo la durata della batteria. Per questo motivo, il processo di classificazione, definito "a cascata", combina i risultati ottenuti da una serie di algoritmi relativamente meno precisi , piuttosto che utilizare un singolo algoritmo che fornirebbe da una precisione maggiore, arrivando ad ottenere una precisione molto vicina ai *framework GPS-based* di questo tipo. Inoltre, le applicazioni di terze parti che necessitano di questo tipo di informazioni, possono ottenerle in maniera completamente trasparente rispetto al *framework* proposto, al fine di implementare servizi *context-aware*.

In [8] viene proposto un *framework* in grado di rilevare l'utilizzo dello *smartphone* da parte del guidatore di un veicolo. È fondamentale quindi, identificare la posizione dell'utente all'interno del veicolo. Vengono utilizzati i sensori integrati del dispositivo (accellerometro/giroscopio), per ottenere le variazioni dell'accellerazione centripeta relativa alla dinamica del veicolo. Da queste informazioni, combinate con la velocità angolare, possiamo dedurre se l'utente si trova nel lato destro o in quello sinistro del veicolo. Nonostante non sia semplice, a causa di vari fattori legati all'ambiente, ottenere dei valori precisi dell'accellerazione centripeta tramite i sensori, l'applicazione proposta fornisce ottimi risultati in termini di precisione. Infatti, in seguito a numerosi test effettuati su due veicoli in due città differenti, il *framework* ottiene una precisione di oltre il 90% con un numero di falsi positivi molto basso.

Negli ultimi anni, si sono verificati migliaia di incidenti stradali dovuti all'utilizzo dei cellulari alla guida, e il lavoro svolto, può essere di grande aiuto per moltissimi servizi legati alla sicurezza stradale.

2.6.2 Ottimizzazione delle performance

Nel 2008, Stephen Herborn, Henrik Petander e Max Ott, pubblicano uno studio [4] su come le performance dei *device multi-homing* possano essere ottimizzate sfruttando informazioni relative al contesto. Per *multihoming*, si intende l'operazione di connettere un *host* o una *computer network* a più di una (singola) rete, al fine di incrementarne affidabilità e *performance*.

La scelta di quale interfaccia di rete utilizzare, viene effettuata in base ai risultati forniti da algoritmo statistico di *machine learning* che sfrutta sia informazioni semplici come le coordinate *GPS* o l'ora del giorno, sia informazioni più complesse come lo stato interno del sistema e i segnali *Bluetooth* presenti nelle vicinanze.

Nel 2009, il ricercatore Yang Ye del team *Microsoft Research Asia*, presenta alla *Tenth International Conference on Mobile Data Management: System, Services and Middleware*, un *framework* che permette di ottenere informazioni sullo stile di vita di un individuo, basandosi sui dati grezzi forniti dalle *location-aquisition technologies (GPS, GSM network, etc.)* [7]. Il concetto fondamentale su cui si basa lo studio di Yang, è quello di *life pattern*. Viene ottenuto analizzando i posti più significativi che un utente visita durante la vita quotidiana, in quanto questi luoghi rappresentano l'attivita svolta tipicamente. Un *life pattern* dunque, è un importante indicatore della regolarità dello stile di vita di un utente.

Il lavoro svolto dal *framework*, chiamato *LP-Mine*, avviene in due fasi:

1. fase di *pre-processing* dei dati grezzi
2. applicazione di tecniche di *mining sui dati ottenuti* per la scoperta di *life pattern*

Nello stesso anno, il ricercatore Murat Ali Bayir nel trattato [1], presenta un *framework* chiamato *Mobility Profiler*, utilizzato per ottenere, partendo dai log relativi alla posizione forniti dai telefoni cellulare, una sorta di profilazione dell'utente relativa agli spostamenti che egli compie più frequentemente. Questo *framework* nasce principalmente per sopperire il distacco presente tra le informazioni di basso livello dei log delle posizioni e le informazioni di alto livello supportate dalle *mobile application*.

2.6.3 Privacy

Nell'ambito della *privacy* troviamo l'analisi [6], condotta da jierui Xie e Hongxia Jin (*Samsung Research Center*) in collaborazione con Bart Piet Knijnenburg (*University of California, Irvine*), su come preservare la *privacy* degli utenti nell'uso di *location-based system*. In particolare, viene mostrato in che modo fattori come la compagnia, lo stato emotivo e il contesto influenzino la scelta di condividere o meno infomazioni legate al luogo in cui ci si trova mediante *location-sharing services (LLS, i.e. Facebook, Instragram, etc.)*, e come , nonostante la natura altamente dinamica e dipendente dal contesto dei servizi *LLS*, questi elementi possano essere sfruttati per predire informazioni utili all'utente (*sharing setting*). L'algoritmo proposto, chiamato *PPRec*, fornisce all'utente consigli personalizzati su i vari *"sharing setting"* da utilizzare, con l'obiettivo di ridurre problematiche legate alla *privacy*.

Uno studio molto importante nel campo della *mobility prediction* è stato svolto da Qiujian Lv, Yuanyuan Qiao, Nirwan Ansari, Jun Liu e Jie Yang. L'articolo [5] pubblicato nel 2016, descrive un *framework* basato su *Hidden Markov Model (HMM)*, che identifica una serie di *Point of Interest (POI)* di un utente, ed effettua operazioni di *mobility prediction*. La predizione di un luogo che un utente visiterà in futuro, è un fattore che può essere di fondamentale importanza per un'ampia serie di applicazioni (ad esempio applicazioni che forniscono informazioni sul traffico o consigli su punti di interesse).

Il *framework* è costituito da due *predictor* differenti:

- *spatio-temporal predictor*: cerca di predire dove si troverà un utente in un dato momento in futuro;
- *next-place predictor*: cerca di predire dove si recherà l'utente dopo aver lasciato il luogo attuale

Nello studio [9], viene analizzata la *privacy* nei contesto dei *cyber physical systems*. Questo tipo di sistemi, sono formati dall'unione tra dispositivi informatici e componenti fisiche, e rappresentano una notevole innovazione, in quanto stanno pian piano rimpiazzando le infrastrutture di base esistenti. Il loro potenziale è dato soprattutto dalla possibilità di comunicare mediante lo scambio di diverse informazioni. Tuttavia, dalla scoperta di tali informazioni da parte di terzi, potrebbero scaturire gravi problematiche fino a compromettere lo sta-

to interno del sistema. Basti pensare ai danni che potrebbero essere arrecati a impianti elettrici o centrali nucleari.

Viene descritto dunque un *framework* matematico basato su un processo Markoviano, che ha l'obiettivo di analizzare il *design* dei *controller* che governano questi sistemi, nel caso la *privacy* abbia un ruolo fondamentale ai fini dei sistemi stessi.

Il lavoro descritto in [10] fornisce importanti informazioni ai fini di questo progetto di tesi. Viene effettuato uno studio relativo alla mobilità di un utente, cercando di identificare i casi in cui il suo comportamento differisce dalla classica ruotine quotidiana.

Il processo di analisi della mobilità di un untente è diviso in due fasi:

1. costruzione di una metrica relativa al grado di entropia di un utente per l'identificazione di *life pattern* e l'individuazione di *"routine departures"*

2. utilizzo di un modello Bayesiano al fine di predire eventuali *"routine departures"* future.

Per valutare l'efficienza dell'analisi effettuata, è stato utilizzato il dataset di *Nokia Lausanne*, comprensivo di dati raccolti dal *GPS*, log delle chiamate e utilizzo delle applicazioni, in merito a 38 utenti che sono stati monitorati per la durata di un anno.

I risultati ottenuti, mostrano inoltre come l'attività quotidiana di un utente sia strettamente correlata all'interazione con il proprio *device*.

Uno dei principali progetti di cui bisogna tener conto, è il lavoro [11] realizzato da Luca Bedogni (*Università di Bologna*) e Marco Levorato (*University of California*). In particolare, viene mostrato come ottenere diverse informazioni relative all'attività o agli spostamenti di un utente, anche avendo a disposizione un insieme ristretto di dati relativi al contesto.

Il dataset utilizzato è composto dai dati raccolti dal *framework* di *Google* "cronologia delle posizioni", che provvede a collezionare informazioni su luoghi e attivà svolte da un utente.

Vengono in seguito descritti tre classificatori basati sull'algoritmo di *machine learning Random Forest*, utilizzati a seconda delle informazioni iniziali a disposizione. Come *training set* sono stati utilizzati i dati relativi ai primi due mesi, mentre i dati del mese successivo sono stati utilizzati come *test set* al fine di verificare l'efficienza del modello.

Per alcuni utenti è più semplice effettuare predizioni di questo tipo, mentre per altri risulta più complesso. Tuttavia le percentuali di accuratezza media, soprattutto durante i giorni lavorativi, sono molto alte, e in alcuni casi vicine al 100%. Inoltre, analizzando i risulati ottenuti, si possono inferire anche le differenze tra i differenti stili di vita degli utenti.

Il punto di riferimento per la realizzazione di questa tesi, è rappresentato dallo studio [12], sempre a cura di Luca Bedogni e Marco Levorato, che ha l'obiettivo di mostrare come le numerose informazioni prodotte dalle *piattaforme mobile* utilizzate dalle applicazioni basate su *context-aware computing*, se acquisite da malintenzionati, possano arrecare gravi rischi per la *privacy*. Nell'articolo [12] , viene descritto un *framework* basato su un processo di Markov e su tecniche di programmazione dinamica, creato con lo scopo di ridurre l'accuratezza delle predizioni effettuate da un eventuale "parte maliziosa". Piuttosto che cercare di "nascondere" le informazioni, tra l'altro spesso rese pubbliche dall'utente stesso, il *framework* genera del "rumore" (dati fittizi) con l'obbiettivo di limitare la precisione relativa alla predizione di un contesto futuro di un utente, preservando allo stesso tempo la funzionalità dei servizi. Il rumore inoltre viene inserito in relazione all'attività svolta tipicamente dall'utente, in quanto dati fittizzi troppo espliciti potrebbero essere facilmente identificati dall'attaccante e quindi evitati.

Capitolo 3

Modello probabilistico

Nel seguente capitolo andremo a descrivere le fasi di costruzione e valutazione dell modello probabilistico ottenuto sulla base delle informazioni estrapolate dai datasets a nostra disposizione.

3.1 Costruzione del modello

Il modello probabilistico utilizzato, è basato sul processo stocastico noto come Catena di Markov. Andremo ora ad analizzare questo modello, descrivendone prima le caratteristiche principali per poi mostrare dettagliatamente come viene costruito a partire dal *dataset* a disposizione.

Markov process

Si definisce *processo stocastico di Markov*, un processo aleatorio in cui la probabilità di transizione che determina il passaggio a uno stato di sistema dipende solo dallo stato del sistema immediatamente precedente (proprietà di Markov) e non da come si è giunti in questo stato. Il primo a sviluppare questa teoria fu il matematico russo Andrej Andreevič Markov.

Un processo markoviano è caratterizzato dalla proprietà di Markov, detta anche condizione di *"assenza di memoria"*, la quale può essere descritta come:

$$P(X(t_{n+1}) = x_{n+1} | X(t_n) = x_n, X(t_{n-1}) = x_{n-1}, ..., X(t_0) = x_0) =$$

$$P(X(t_{n+1}) = x_{n+1} | X(t_n) = x_n)$$

o più semplicemente

P("FUTURO" | "PRESENTE + PASSATO") = P("FUTURO" | "PRESENTE)

Si definisce *catena di Markov* un processo di Markov con spazio degli stati discreto, quindi un processo stocastico che assume valori in uno spazio discreto e che gode della proprietà di Markov.

Nella vita reale le catene di Markov vengono utilizzate come modello statistico per modellare una moltitudine di processi in diversi ambiti, come lo studio di sistemi cruise control nel campo della motorizzazione, o algoritmi previsionali in ambito economico, finanziario e di dinamica dei sistemi.

Sono numerosissimi anche gli algoritmi di *Link Analysis Ranking* basati sulla teora di processi markoviani. Ad esempio l'algoritmo *PageRank* di *Google*, si basa sulla frequenza a posteriori di transizione degli utenti da un sito web A ad un sito B tramite i *link* che da A conducono a B e non sul semplice numero e tipo di collegamenti da A a B, in modo da considerare principalmente la popolarità del legame per gli utenti piuttosto che l'importanza per il creatore del sito web.

3.2 Dataset

Il *dataset* utilizzato, è composto da una raccolta di informazioni relative a una serie di mobile user, i quali hanno acconsentito al trattamento dei propri dati per lo scopo di questo progetto. Nello specifico, i dati sono ottenuti mediante il servizio di *Google "cronologia delle posizioni"*, che ha l'obiettivo di registrare il percorso e i luoghi visitati da un utente durante il corso della giornata. Parsando dai dati grezzi rilevati da *Google*, per ogni utente sono stati creati *dataset* di dimensioni differenti (1 mese, 3 mesi, 6 mesi, 12 mesi) sottoforma di *file csv*, in cui ogni riga rappresenta un *record* composto dalle seguenti *features*:

- latitudine e longitudine;
- giorno della settimana;
- data
- ora
- attività svolta (può assumere uno tra i seguenti valori: *STILL, ON_FOOT, IN_VEHICLE, EXITING_VEHICLE, ON_BICYCLE*)
- attività svolta nel record precedente
- *deltaActivity*, ovvero la differenza espressa in secondi tra l'attività attuale e l'attività svolta precedentemete
- informazioni sulla posizione espresse come coordinate *MGRS* di differenti granularità;
- valori *MGRS* relativi al record precedente;
- *timeSlot*
- *halfTimeSlot*
- stagione corrente

Esempio della stuttura di un dataset

	A	B	C	D	E	F	G	H	I	J	K	L
1	lat	long	dayWeek	date	time	activity	prevActivity	deltaActivity	mgrs100km	mgrs10km	mgrs1km	mgrs100m
2	62.6693845	23.5434523	Friday	26-02-2019	06:16:00 AM	STILL	IN_VEHICLE	236	24PQ	24PQ04	24PQ0402	24PQ040206

	L	M	N	O	P	Q	R	S	T	U	V	W
1	mgrs100m	mgrs10m	mgrs1m	prevMgrs100km	prevMgrs10km	prevMgrs1km	prevMgrs100m	prevMgrs10m	prevMgrs1m	timeSlot	halfTimeSlot	season
2	24PQ040206	24PQ04020666	24PQ0402066601	24PQ	24PQ04	24PQ0402	24PQ040216	24PQ04021662	24PQ0402166234	6	06:00:00 AM	winter

Figura 3.1: Esempio di record contenuto nel dataset

MGRS - Military Grid Reference System

Il *Military Grid Reference System (MGRS)* è uno standard di coordinate geografiche, usato dalla *NATO* in ambito militare, per localizzare vari punti della superfice terrestre. A differenza delle coordinate geografiche tipicamente utilizzate (latitudine e longitudine) per identificare un punto preciso del globo terrestre, un valore *MGRS* è riferito ad una zona più ampia rappresentata da un quadrato (da cui deriva il termine *"grid"*), la cui dimensione dipende dalla precisione fornita dalle coordinate. Coordinate di questo tipo possono essere composte da 0, 2, 4, 6, 8 o 10 cifre, a seconda della precisione desiderata.

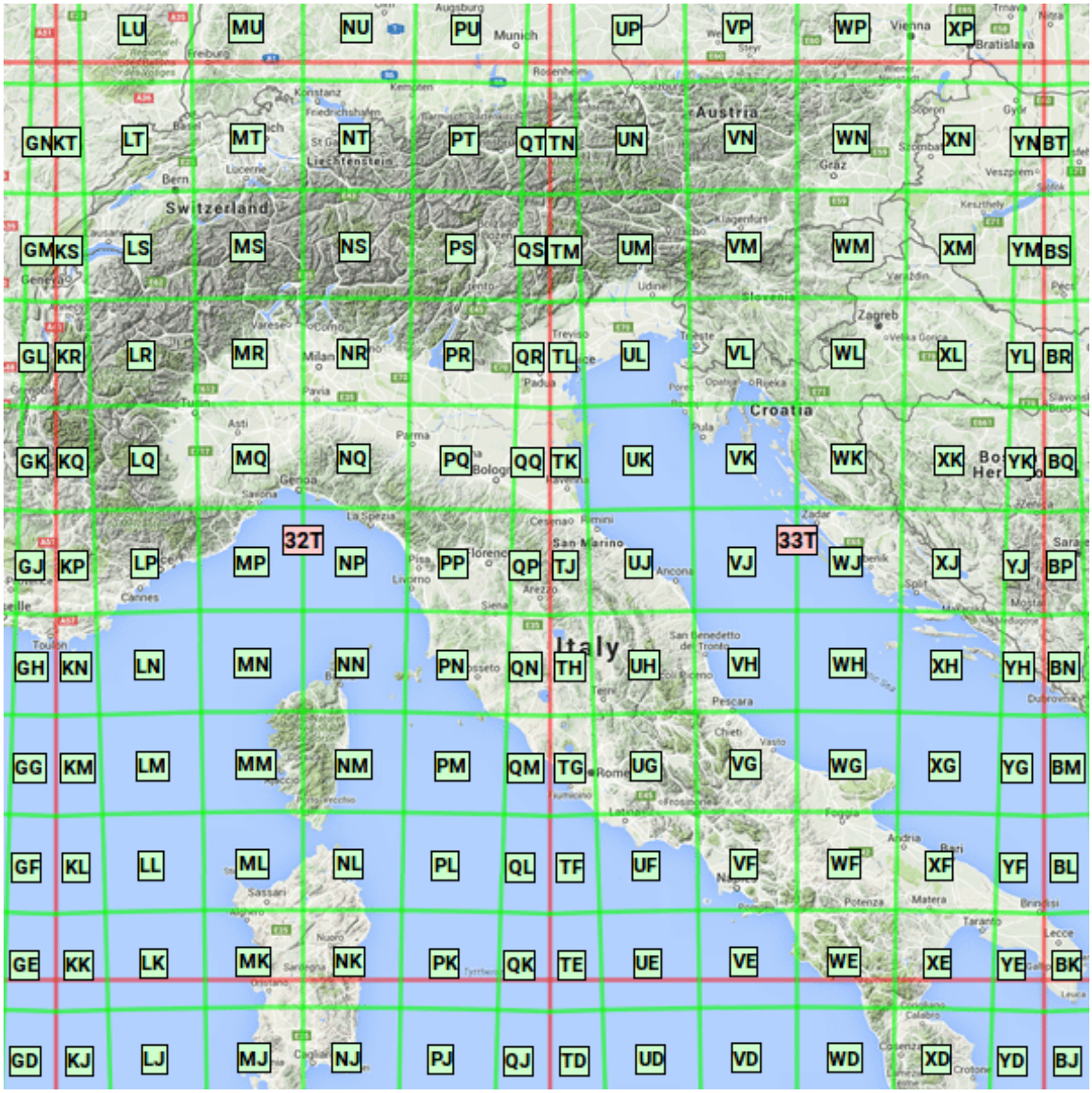

Figura 3.2: Coordinate MGRS del territorio italiano. Fonte: https://www.researchgate.net/figure/Italy-and-the-city-of-Bologna-represented-through-the-MGRS-scheme-In-Figure-2a-the_fig1_313543044

3.3 Librerie utilizzate

Descriviamo per prima cosa le librerie utilizzate che ci forniranno il supporto necessario per svolgere una vasta serie di operazioni. In particolare, verranno utilizzate per interagire col *dataset* e per costruire e valutare la catena di Markov in maniera efficiente, oltre a fornire supporto nella fase di debug.

```
import sys
import csv
import networkx as nx
import matplotlib.pyplot as plt
import json
```

- *sys*: fornisce funzioni per interagire direttamente col sistema operativo su cui stiamo lavorando. In questo caso, viene utilizzata per leggere i parametri passati da riga di comando, che saranno fondamentali al fine di determinare le differenti modalità in cui può operare l'algoritmo.

- *csv*: Viene utilizzata per effettuare operazioni sul *dataset*, fornendo funzioni specifiche per leggere/scrivere su file di tipo *csv* (*Comma Separated Values*). Nello specifico, permette di gestire un file di questo tipo in due modi differenti:

- *reader:* il file viene trattato come una matrice, e tramite un *ciclo for* è possibile iterare sulle varie righe. Ogni singolo campo di un *record* (riga della matrice), può essere acceduto facilmente tramite un indice corrispondente alla colonna in cui si trova la *feature* desiderata. È utile quando bisogna calcolare il numero di *record* totali del *dataset*, o quando bisogna ottenere gli *header*;

- *DictReader:* il file viene trattato come una struttura dati di tipo dizionario. In questo modo, utilizzando una lista contenente gli *header*, ci permette di ottenere con facilità le *feature* desiderate durante la fase di *data fetching* e di valutazione del modello.

- *networkx*: permette di creare, manipolare e analizzare strutture dati di tipo *graph* (grafo), *DiGraph* (grafo orientato) e *MultiGraph* (grafo multidirezionale). A partire dai vari *cluster*, e sfruttando le funzioni fornite da questa libreria, costruiremo un grafo direzionato che rappresenterà la nostra catena di Markov. Ci sarà molto utile anche durante la fase di valutazione del modello mettendo a disposizione funzioni per navigare la catena e ottenere i successori relativi ad un determinato nodo.
Inoltre supporta molti algoritmi standard relativi ai grafi che possono essere utilizzati per effettuare operazioni di analisi con una complessità ottimale.

- *matplotlib.pyplot*: permette la creazione di un'ampia serie di grafici, mostrandoli semplicemente a video, o esportandoli in un file esterno. È stata utilizzata principalmente in fase di *debug* per fonire una rappresentazione grafica della catena (costruita su una piccola parte del *dataset*) e verificarne la correttezza. Non viene utilizzata per mostrare la catena completa di tutti gli stati in quanto non sarebbe possibile visualizzare una struttura dati di tale dimensione

-*json*: fornisce funzioni per interagire con file di tipo *JSON (JavaScript Object Notation*. Una volta terminata la fase di *data fetching*, il primo campo di ogni *record* sarà costituito da un indice, che determinerà una precisa fascia oraria in base al *cluster* in cui si trova il record stesso. Le varie corrispondenze indice-intervallo di tempo sono contenute all'interno di un *file json*, il cui *parsing* viene effettuato mediante le funzioni fornite da questa libreria al fine di effettuare l'operazione di *mapping*.

3.4 Modalità d'uso

Il *framework* può costruire il modello ed effettuare predizioni secondo diverse modalità, a seconda dei parametri passati da riga di comando, in questo modo l'analisi può essere effettuata in modo più ampio, considerando una vasto insieme di casi differenti. Ad esempio, può essere utile effettuare un confronto tra i risultati ottenuti cercando di predire determinate informazioni nei giorni lavorativi, e i risultati ottenuti considerando soltanto i giorni festivi. Oppure, analizzare come varia il grado di precisione incrementando o diminuendo l'intervallo di tempo specificato, o in base alla quantità di dataset utilizzata per apprendere le informazioni di un utente e costruire il relativo modello.

Combinando i risultati generati eseguendo l'algoritmo in modalità differenti, è possibile infine risalire a diversi aspetti relativi alla routine di un utente.

Nello specifico, i parametri richiesti sono i seguenti:

```
$ python3 script.py userN -mode -t_window -t_setSize -datasetLength
```

-*userN*: utente per cui si vuole effettuare la predizione. N può assumere valore compreso tra 1 e 5.

-*mode*: può essere -w (*weekDays*) o -f (*festiveDays*). A seconda del parametro indicato, la catena verrà costruita e valutata considerando soltanto i giorni lavorativi o i giorni festivi.

-*time_window*: rappresenta l'intervallo di tempo da considerare per il processo di *clustering*. Può assumere uno tra i seguenti valori:

15min, 30min, 1h, 2h, 3h, 4h,6h, 12h.

Determina inoltre, il numero di *cluster* da creare in cui verranno inseriti i vari *record*. Ad esempio, se assume valore *15min*, il numero di *cluster* sarà

uguale a 96. Supponiamo di leggere dal *dataset* tre *record* di questo tipo:

1)<lat = 44.6466597, long = 11.5352053, ... , **time = 14.10** , ...>

2)<lat = 44.6466512, long = 11.5352039, ... , **time = 14.12** , ...>

3)<lat = 44.6466556, long = 11.5352026, ... , **time = 14.20** , ...>

I primi due *record* verranno inseriti nel *cluster* corrispondente alla fascia oraria **14:00-14:15**, mentre il terzo verrà inserito nel *cluster* corrispondente alla fascia oraria **14:15-14:30**.

Nel caso invece avessimo un qualsiasi valore di *time window* superiore a *15min*, tutti e tre i *record* verrebbero inseriti nel medesimo *cluster* (**14:00-14:30** per *time window = 30min*, **14:00-15:00** per *time window = 1h*, etc.);

-training_set_size: indica, in percentuale, la dimensione di *dataset* da utilizzare come *training set* al fine di costruire la *catena di Markov*. La parte di dataset restante, sarà utilizzata come *test set* al fine di verificare la precisione del modello costruito.

-datasetLength: può assumere uno tra i seguenti valori: *1m, 3m, 6m, 12m*. Permette di selezionare il *dataset* di lunghezza desiderata relativamente ad un dato utente.

3.5 Data fetching

Durante questa fase, avviene l'estrazione dal *dataset* di dati relativi all'utente specificato precedentemente, affinchè in seguito le informazioni possano essere processate e *clusterizzate* in modo efficiente. Quest'operazione rappresenta una fase di *clustering preliminare*, utile a facilitare la fase di *clustering* vera e propria che effettueremo successivamente.

La prima funzione ad essere chiamata all'interno dello script, è la funzione *ReadData()*

```
def ReadData():
```

che ha il compito di leggere i *record* presenti nel dataset utilizzato, ed inserirli in una struttra dati organizzata come una matrice multidimensionale, composta da un numero di *cluster* calcolato in base al valore della *time window* scelta. Ad ogni record inserito nel cluster corrispondente, viene associato un intero, che rappresenta il numero di occorrenze di quel *record* all'interno del *training set*. Inoltre, a seconda dei parametri letti da riga di comando, per la costruzione della matrice possono essere inseriti nei *cluster* soltanto i *record* corrispondenti ai giorni lavorativi o ai giorni festivi.

La funzione *ReadData()* si basa su una serie di procedure ausiliarie:

```
def getHeader():
```

-*getHeader()*: legge la prima riga del dataset per ottenere gli *header* che rappresentano la tipologia delle varie *features (latitude, longitude, activity, etc.)*. Ritorna una lista contenente una serie di parametri che saranno successivamente utilizzati come chiavi per accedere ai singoli campi dei *record*.

```
def getTrainingSetSize():
```

-*getTrainingSetSize()*: effettua una prima scansione del *dataset* utilizzato, al fine di ottenere il numero di *record* totali da cui è composto. Utilizza poi il parametro letto da riga di comando per stabilire le percentuali di file da utilizzare come *training* e come *test set*. Successivamente, in base alla dimensione del

training set desiderata, ritorna un intero rappresentante il numero di *record* da prendere in considerazione per la costruzione del modello probabilistico.

```
def buildMatrix(matrix, flag, f_number):
```

-*buildMatrix(matrix, flag, f_number)*: inizializza la struttura dati in base al numero di *cluster* da cui deve essere composta. Il parametro *matrix*, che riceve in input, costituisce la struttura dati in cui verranno inseriti i *record* durante la fase preliminare di *clustering*. In questa chiamata i parametri *flag* e *f_number* sono entrambi settati a 0, e verranno utilizzati principalmente nelle chiamate successive a questa funzione. Se la matrice viene creata con successo la funzione ritorna *true*, altrimenti ritorna *false*.

```
def getIndex(timeSlot, time):
```

-*getIndex(timeSlot, time)*: prende in input i campi *timeSlot* e *time*, in base al valore della *time window* indicata calcola l'indice relativo al *cluster* in cui deve essere inserito il *record* appena letto dal *dataset*. All'interno di questa funzione viene definito un dizionario chiamato *blockDict*, le cui chiavi rappresentano i valori che può assumere il parametro *time_window*, mentre i valori corrispondono al numero di *cluster* da cui sarà composta la matrice. In caso di intervalli di tempo uguali o maggiori a un'ora, l'indice corrispondente sarà uguale a :

$$Index = \frac{timeSlot}{blockDict[t\_window]}$$

Il secondo parametro viene preso in considerazione nel caso il valore di *time window* sia uguale a 15min o 30 min. In questo caso, l'indice corrispondente sarà calcolato in questo modo:

$$Index = timeSlot * blockDict[t\_window] + n$$

dove n assume valore compreso tra 0 e 4, a seconda del quarto orario o della mezz'ora considerati.

```
def lookForPattern(index, record):
```

-*lookForPattern(index, record)*: verifica la presenza del *record* all'interno del *cluster* di indice *index*. Questa funzione ha l'obiettivo di indentificare e accorpare i *record* comuni, in modo da facilitare la fase successiva di *clustering*, in caso abbia già incontrato precedentemente un *record* con quella struttura, ritorna *true* e aggiorna il numero di occorrenze. Altrimenti, vuol dire che è la prima volta che quel *record* viene letto dal *dataset* e quindi ritorna *false*, inserendolo nel *cluster* appropriato con contatore delle occorrenze uguale a 1.

Al termine della funzione *readData()*, tutti i *record* necessari alla creazione del *training set* sono stati letti dal *dataset* ed inseriti nei corrispettivi *cluster*.

3.6 Costruzione matrice intermedia

Una volta terminata la fase di *data fetching* o fase di *clustering preliminare* è il momento di iniziare la fase di data *clustering* vera e propria, in cui costruiremo una una "matrice intermedia" considerando soltanto un sottoinsieme di tutte le *features* da cui è composto il *dataset*.

L'idea di base, è che non sempre siamo a conoscenza di una serie così ampia di informazioni relative ad un utente, quindi dobbiamo essere in grado di creare un modello probabilistico sfruttando soltanto le *features* a nostra disposizione.

Le informazioni di cui siamo a conoscenza vengono indicate in un file di testo (*tests.txt*), e possono essere espresse mediante notazioni differenti:

- valori interi separati da una virgola, ad esempio 1,3,5,6.
- specificate tramite un intervallo, ad esempio 2:6, 9:12
- tramite le *keyword*:
  - mgrs1km, mgrs100m, mgrs10m, mgrs1m)*SPACE_ALL (lat, long, mgrs100km, mgrs10km, mgrs1km, mgrs100m, mgrs10m, mgrs1m)*
  - SPACE_PREV (prevMgrs100km, prevMgrs10km, prevMgrs1km, prevMgrs100m, prevMgrs10m, prevMgrs1m)
  - TIME_ALL (date, time, dayWeek, timeSlot, halfTimeSlot, season)
  - ACTIVITY (activity)
  - ACTIVITY_PREV (prevActivity)
  - ACTIVITY_DELTA (deltaActivity)
- una combinazione delle notazioni precedenti;

La creazione di questa "matrice intermedia", avviene mediante la chiamata alla funzione *getFeatures()*

```
def getFeatures():
```

che svolge anche l'operazione di *mapping delle features*.

3.6.1 Features mapping

Al fine di ottenere un elenco delle informazioni di cui siamo a conoscenza in un formato processabile in maniera più semplice, bisogna effettuare una conversione dalle differenti notazioni in cui possono essere espresse le *features* verso i nomi effettivi delle *features* stesse.

Utilizziamo quindi la funzione *getLabel()*

```
def getLabel():
```

che si occupa di effettuare la fase di mapping. Le varie corrispondenze <indice : feature >sono contenute in un file chiamato *features.txt*.

Per prima cosa, viene letta dal *file tests.txt* una stringa che rappresenta l'insieme delle varie *features* in uno dei formati precedentemente citati. Questa stringa viene poi splittata nelle componenti rappresentanti le singole *label* e , per ognuna di esse, partendo dall'indice corrispondente viene ricavato il nome completo della *feature*.

Per le feature dichiarate secondo la notazione *keyword*, viene utilizzata la funzione ausiliaria *getKeyword()*.

```
def getKeyword():
```

che genera una struttura dati di tipo dizionario le cui chiavi rappresentano le *keyword*, mentre i valori sono costituiti dalle *features* corrispondenti. Le varie corrispondeze <keyword : lista di features >vengono ottenute tramite la lettura di un file denominato *binding.txt*.

Infine, la funzione *getLabel()* termina ritornando una lista composta dalle *features* da utilizzare nella costruzione della matrice intermedia.

3.6.2 Data clustering

In seguito alla *fase di mapping* avviene la creazione della struttura dati da cui deriveremo successivamente la catena di Markov, raggruppando i *record* comuni presenti nella matrice iniziale comprensiva di tutte le *features* in base alle informazioni a nostra disposizione. Ogni *record* presente nei vari *cluster*, il cui numero, come per la matrice iniziale, dipenderà dal valore della *time window* scelta, rappresenterà uno stato differente della catena.

In *getFeatures()*, viene chiamata nuovamente la funzione *buildMatrix()* con i seguenti parametri:

```
    data = getLabel()                    #features mapping
    buildMatrix(mid_matrix,1,len(data))
```

dove *mid_matrix* rappresenta la nostra matrice intermedia. La differenza rispetto alla chiamata precedente, notificata alla funzione *buildMatrix()* tramite il parametro *flag*, sta nel fatto che in ogni *cluster* deve essere presente un record di questo tipo:

<cluster_index, -1, ... , -1, 1 >

dove il numero di valori -1 sarà uguale al numero delle *features* di cui siamo a conoscenza, per questo motivo il valore *len(data)* viene fornito come paramentro alla funzione *buildMatrix*, mentre l'ultima cifra rappresenta il contatore delle occorrenze. Questo *record* viene inserito per considerare il caso in cui non fossimo a conoscenza di nessuna informazione relativa a quella precisa fascia oraria.

A questo punto, è arrivato il momento di riempire i *cluster* con i dati di nostro interesse.

```
    for index in range(len(matrix)):
        for block in range(len(matrix[index])):
            aux = []
            for label in data:
                if(label == 'time' or label == 'deltaActivity'):
                    continue

                aux.append(matrix[index][block][label])

            c = getCounter(aux, data, index)
```

Iniziamo a quindi ad iterare sulla matrice principale. I *record* comprensivi di tutte le *feature*, devono essere accorpati secondo le informazioni lette dal file *test.txt*. Ad esempio, supponiamo di avere a disposizione soltanto attività e valori delle coordinate *MGRS* di granularità 100m.

Nel medesimo *cluster* della matrice iniziale sono presenti i seguenti *record*:

<lat:44.5324, long:11.5352,...,activity:**STILL**,mgrs100m:**32TQ010384**,...,counter:**2** >

<lat:44.21321, long:11.21342,...,activity:**STILL**,mgrs100m:**32TQ010384**,...,counter:**3** >

<lat:44.53234, long:11.64392,...,activity:**STILL**,mgrs100m:**32TQ010384**,...,counter:**1** >

Nella matrice intermedia, andremo dunque ad inserire un unico *record*, strutturato in questo modo:

<activity:STILL, mgrs100m:32TQ010384, counter:**6** >

dove il numero delle occorrenze di quel *record* è dato dalla somma dei valori dei singoli campi *counter*.

Le indicazioni su quali record accorpare, vengono fornite tramite la chiamata alla funzione *getCounter()*

```
def getCounter(record, data, index):
```

che ricerca ,all'interno del *cluster* (della matrice principale) di indice *index*, le occorrenze del *record* ricevuto come parametro e ,per ogni *match* avvenuto aggiorna, il contatore delle occorrenze. Inoltre, ogni qualvolta avviene un *match*, il campo *counter* del *record* appartenente alla matrice principale viene settato a zero affinchè uno stesso *pattern* non sia considerato più volte.

Terminata la fase di ricerca, ritorna il numero di occorrenze del *record*.

Se il valore ritornato da *getCounter()* è diverso da zero, significa che quel *pattern* è stato trovato per la prima volta dunque inserisco, nell'apposito *cluster* della matrice intermedia, il *record* attuale comprensivo di un campo *counter* uguale al numero di occorrenze appena calcolato.

Altrimenti, se il valore ritornato da *getCounter()* è uguale a zero, vuol dire che ho già incontrato precedentemente quel *record* e quindi non effettuo alcuna operazione.

La cardinalità dei vari *cluster* viene memorizzata in un vettore chiamato *cluster_size*, che verrà utilizzato in seguto per il calcolo delle probabilità di transizione.

Infine, una volta terminata la fase di costruzione della matrice intermedia, viene chiamata la funzione *addTime()*

```
def addTime():
```

che ha il compito di assegnare ai vari *record*, una sorta di "etichetta" relativa alla relativa fascia oraria .

Per effettuare questa operazione utilizza le informazioni contenute in un *file JSON*, chiamato *time.json*, in cui ad ogni *time window* viene associato l'insieme delle diverse corrispondenze <indice del cluster : fascia oraria >

```
        "4h":{
                        "0":"00-04", "1":"04-08", "2":"08-12", "3":
                           "12-16", "4":"16-20", "5":"20-00"
        },
        "6h":{
                        "0":"00-06", "1":"06-12", "2":"12-18", "3":
                           "18-00"
        },
        "8h":{
                        "0":"00-08", "1":"08-16", "2":"16-00"
        },
        "12h":{
                        "0":"00-12", "1":"12-00"
        }
```

Inizialmente il contenuto del file *time.json* viene caricato all'interno di una variabile definita come *time*. In seguito, la funzione *addTime()* inizia ad iterare sui *cluster* della matrice intermedia e, per ogni *record* contenuto in ognuno di essi, estrapola la relativa fascia oraria nel modo seguente:

time[t_window][cluster_index]

aggiungendo il valore ottenuto nella prima posizione di ogni *record*.

3.7 Costruzione della catena

Una volta completata la *fase di clustering* siamo in grado di costruire il modello vero e proprio, su cui faremo affidamento per effettuare le predizioni.

L'idea è quella di implementare la catena come un *grafo orientato* in cui ogni singolo *record* della matrice intermedia precedentemente costruita, andrà a formare uno stato. Ogni stato poi, rappresenterà, a seconda delle informazioni a nostra disposizione, una configurazione in cui l'utente si è trovato in passato almeno una volta. In seguito, ogni nodo relativo ad una specifica fascia oraria, verrà connesso a tutti i nodi della fascia oraria successiva, associando ai corrispettivi archi un peso che rappresenterà la probabilità di transizione.

Questa fase viene svolta dalla funzione *buildChain()*

```
def buildChain():
```

che sfrutta le funzioni fornite dalla libreria *networkx* per la costruzione della *catena di Markov*.

Per prima cosa, definiamo la struttura dati come un grafo orientato:

```
MarkovChain = nx.DiGraph()
```

per andare a creare successivamente gli stati della catena partendo dalla matrice intermedia.

```
            node = tuple(mid_matrix[index][block][0:-1])
            MarkovChain.add_node(node)
```

Ogni *record* presente nei vari *cluster* costituirà uno stato, rappresentato da un nodo del grafo definito come una tupla.

Infine, aggiungiamo al grafo gli archi a cui associamo un peso, che rappresenta la probabilità di transizione tra due stati

```
            first_node = tuple(mid_matrix[index][block][0:-1])

            try:
                for j in range(len(mid_matrix[index+1])):
                    second_node = tuple(mid_matrix[index+1][j
                        ][0:-1])
                    probability=round(mid_matrix[index+1][j][-1]/
                        cluster_size[index+1],3)
                    MarkovChain.add_edge(first_node,second_node,
                        weight=probability)
            except:
                for j in range(len(mid_matrix[0])):
                    second_node = tuple(mid_matrix[0][j][0:-1])
                    probability=round(mid_matrix[0][j][-1]/
                        cluster_size[0],3)
                    MarkovChain.add_edge(first_node,second_node,
                        weight=probability)
```

Dati due stati i e j tra loro connessi, la probabilità di transizione *P(i,j)* da *i* a *j* è definita come:

$$\frac{casi\ favorevoli}{casi\ possibili}$$

quindi

$$P(i,j) = \frac{occorenze\ dello\ stato\ j}{cardinalità\ del\ cluster\ corrispondente}$$

dove il numero di occorrenze dello stato j è dato dal campo *counter* del corrispettivo *record*, mentre la cardinalità del *cluster* viene ottenuta dal vettore *cluster_size*.

Esempio: Catena di Markov con MGRS1000 e ACTIVITY

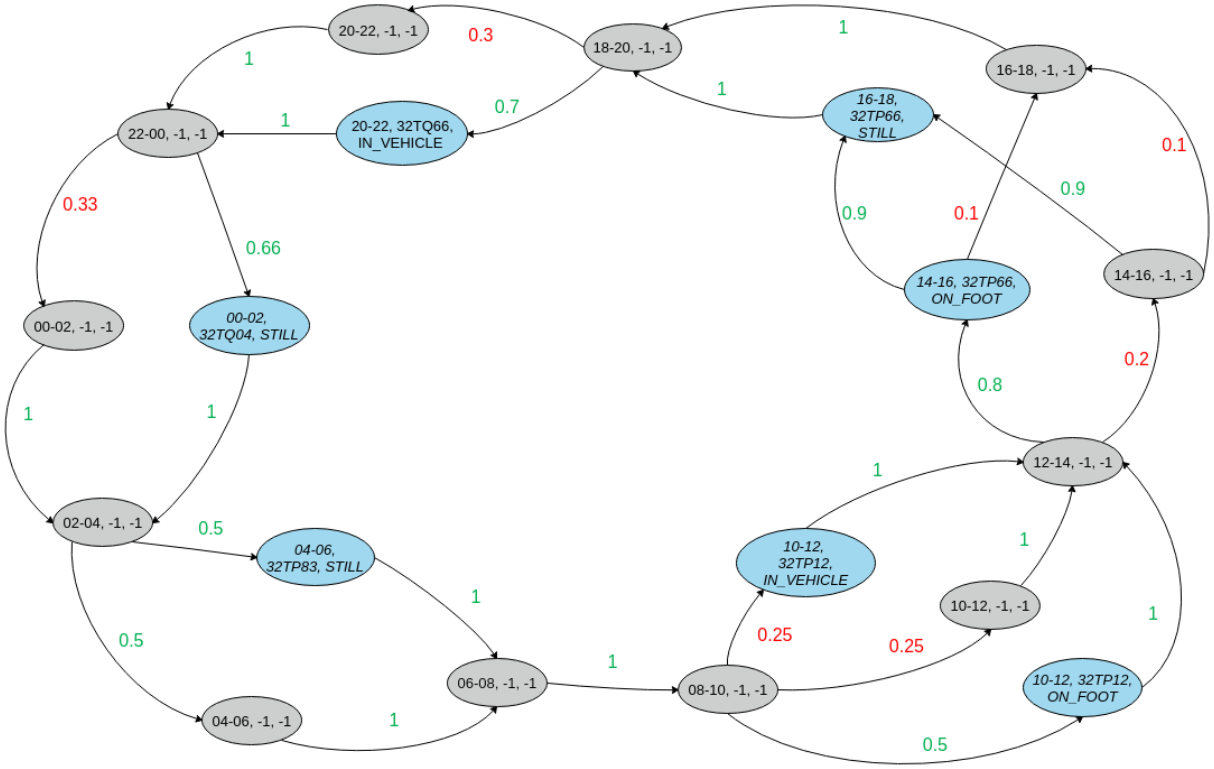

Figura 3.3: Esempio di catena di Markov con due features

3.8 Valutazione del modello

Terminata la fase di costruzione del modello probabilistico, inizia la fase di valutazione, in cui viene utilizzato il test set, ovvero la porzione di dataset restante, non utilizzata per costruire la catena di Markov, al fine di ottenere il numero di stati che l'algoritmo è stato in grado di predire e verificarne l'accuratezza. Il concetto di base è che, basandomi sugli eventi registrati in passato (training set), devo essere in grado di prevedere, con determinate probabilità, eventi futuri (test set) che si verificano nella routine di un utente.

Il lavoro di valutazione e predizione, viene svolto dalla funzione *doValuation()*:

```
def doValuation():
```

in cui l'algorimo cerca di predire gli stati relativi alle 4 ore successive rispetto ad ogni *record* letto dal *test set*. La funzione verifica se è presente nella catena uno stato composto dalle medesime *features*, chiamando la funzione *successors()* della libreria *networkx*, e fornendogli come parametro il *pattern* appena letto.

```
        second_node_set = list(MarkovChain.successors(first_node))
```

Se lo stato è presente nella catena, vuol dire che quel determinato evento si è già verificato precedentemente. La funzione *successors* ritorna una lista composta da tutti i suoi nodi successori. A partire dallo stato corrente, avanzo nella catena fino agli stati corrispondenti alle 4 ore successive, memorizzando gli eventi che ho predetto in una struttura dati chiamata *"PreDict"*,e associando le relative probabilità ad ognuno di essi.

In caso contrario, la ricerca fallisce e l'algoritmo avanza al prossimo *record* presente nel *test set*.

Gli stati appena memorizzati in *PreDict* dunque, rappresentano gli eventi che intendo predire.

Per ogni record letto dal *test set*, verifico se in *PreDict* è presente uno stato corrispondente alla configurazione attuale. Se la ricerca da esito positivo, vuol dire che precedentemente sono riuscito a predire lo stato attuale, quindi incremento la variabile che rappresenta il numero dei *match* avvenuti.

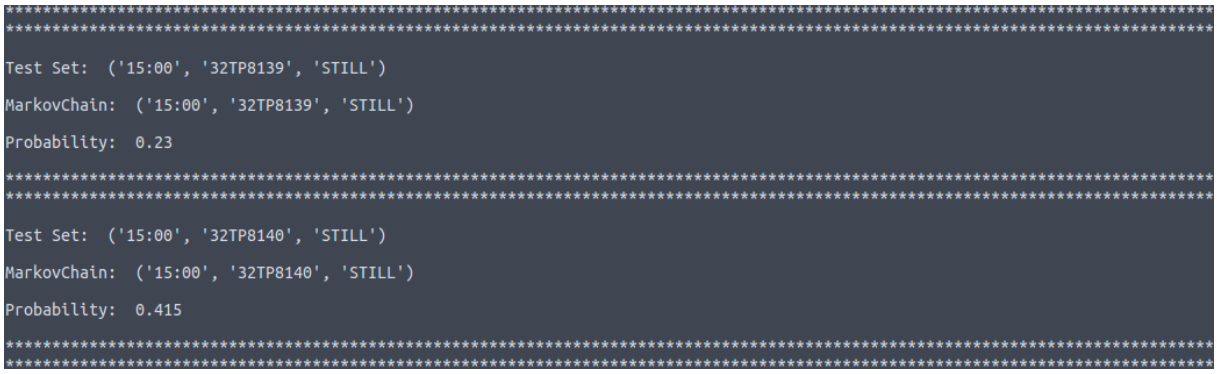

Figura 3.4: Esempio di match avvenuto.

Inoltre, l'algoritmo fornisce in *output* lo stato della catena, con la relativa probabilità, e la configurazione attuale tra cui è avvenuto il *match*.

Terminata la fase di valutazione vengono mostrati a video il numero di *match* avvenuti, il numero di casi totali analizzati e l'*accuracy* risultante.

Capitolo 4

Risultati

In questo capitolo, andremo ad analizzare i risultati prodotti dall'algoritmo di predizione. Per la fase di *test*, sono stati utilizzati *datasets* di dimensioni differenti (1 mese, 3 mesi, 6 mesi, 12 mesi) relativi a cinque utenti. Abbiamo cercando di predire i luoghi (in termini di coordinate *MGRS* di differenti granularità) e l'attività svolta da un utente nelle quattro ore successive rispetto alle informazioni riportate dal *test set*. Per ogni *test* effettuato il *dataset* utilizzato, indipendentemente dalla sua durata temporale, viene diviso in questo modo:

- *training set:* 70%;
- *test set:* 30%;

Partiremo dall'analisi dei risultati ottenuti cercando di predire le coordinate *mgrs10*, per poi trattare i casi successivi come un'estensione del caso appena citato mostrando le variazioni risultanti in termini di incremento o perdita di precisione.

4.1 Entropia

Esclusi i dati da cui è composto il *dataset*, non siamo a conoscenza di alcuna informazione in merito ai vari utenti (età, sesso, posizione lavorativa, etc.).

Introduciamo dunque, un concetto che sarà molto utile per comprendere al meglio i risultati ottenuti.

Entropia: indice del grado di regolarità della vita quotidiana di un utente. In questo caso, deriva dal numero luoghi visitati e/o attività svolte durante il periodo preso in considerazione.

-utente con *alto grado di entropia:* conduce uno stile di vita meno regolare, compie spostamenti frequenti, e per cui sarà quindi più complesso identificare dei *life pattern*. Per questo tipo di utenti, il nostro algoritmo produrrà risultati meno accurati, in quanto avrà una maggiore difficoltà nell'effettuare le predizioni.

-utente con *basso grado di entropia:* caratterizzato da *life pattern* ricorrenti durante la sua ruotine quotidiana, in genere tende a visitare un numero minore di luoghi. I risultati prodotti per questo tipo di utenti, saranno quindi più precisi.

4.2 Working days

Il primo test è stato effettuato impostando il parametro *time window* a 1 ora e considerando soltanto i giorni lavorativi. I risultati ottenuti, evidenziano come nei giorni lavorativi, gli utenti tendano ad avere una routine più regolare, con un risultante grado di entropia minore.

Analizziamo dunque, la precisione fornita dall'algoritmo nei diversi casi considerati.

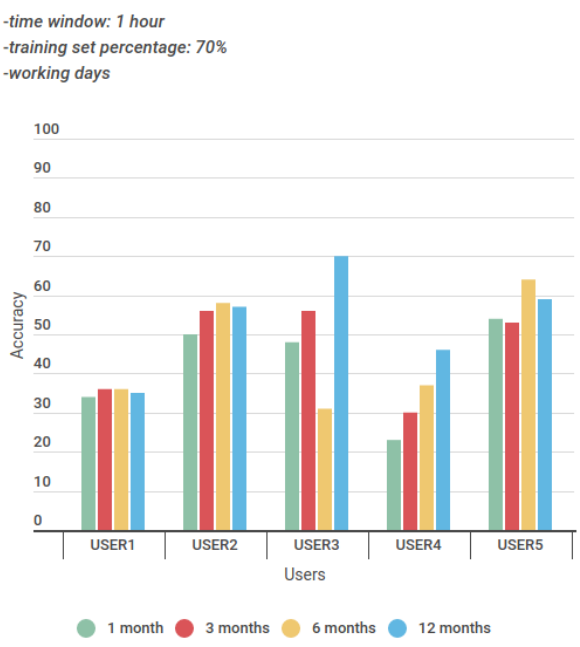

Figura 4.1: Risultati mgrs10 (1h, w.days)

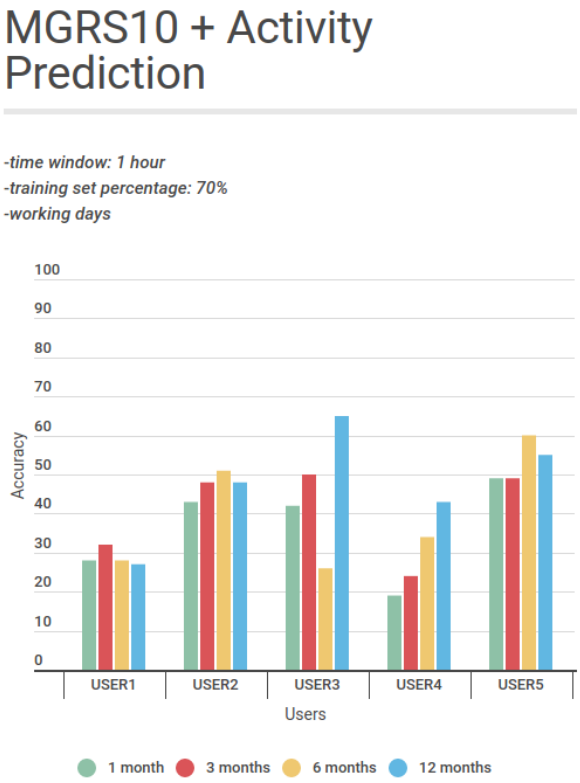

Figura 4.2: Risultati mgrs10 e activity (1h, w.days)

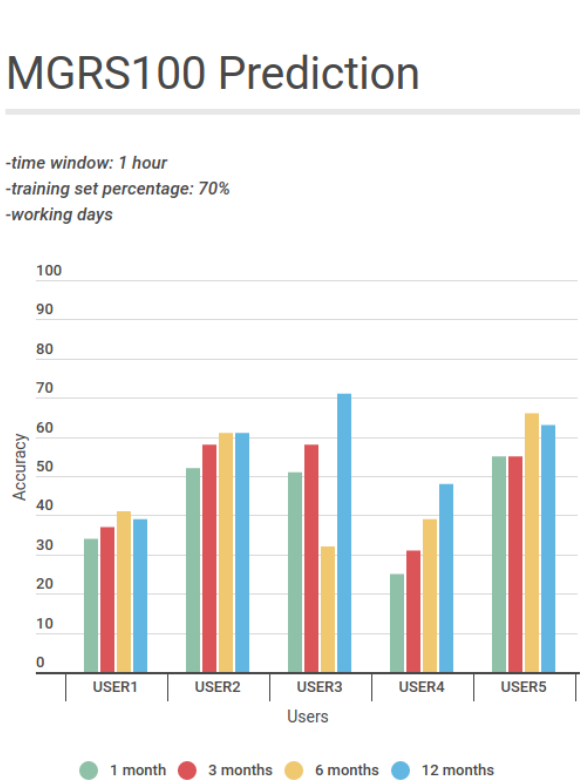

Figura 4.3: Risultati mgrs100 (1h, w.days)

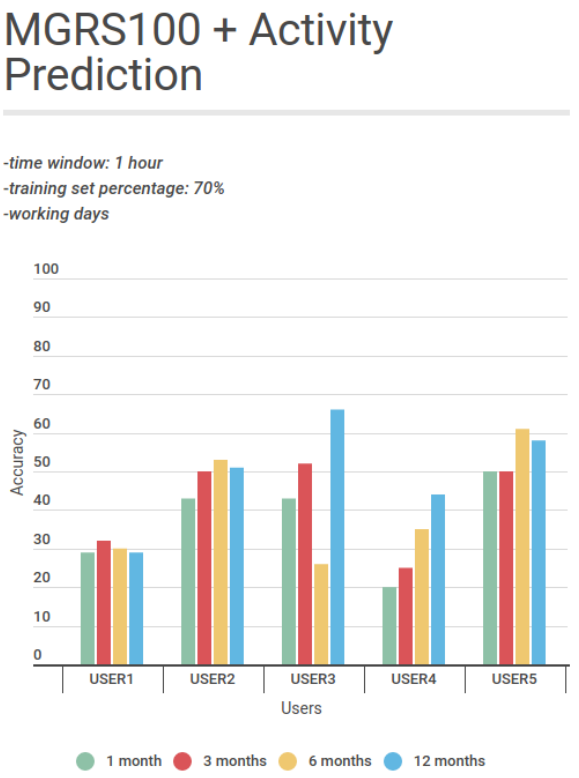

Figura 4.4: Risultati mgrs100 e activity (1h, w.days)

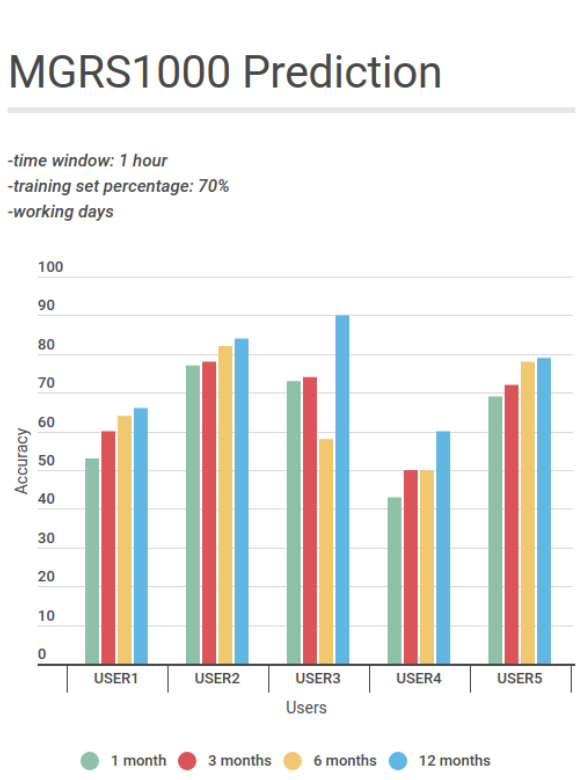

Figura 4.5: Risultati mgrs1000 (1h, w.days)

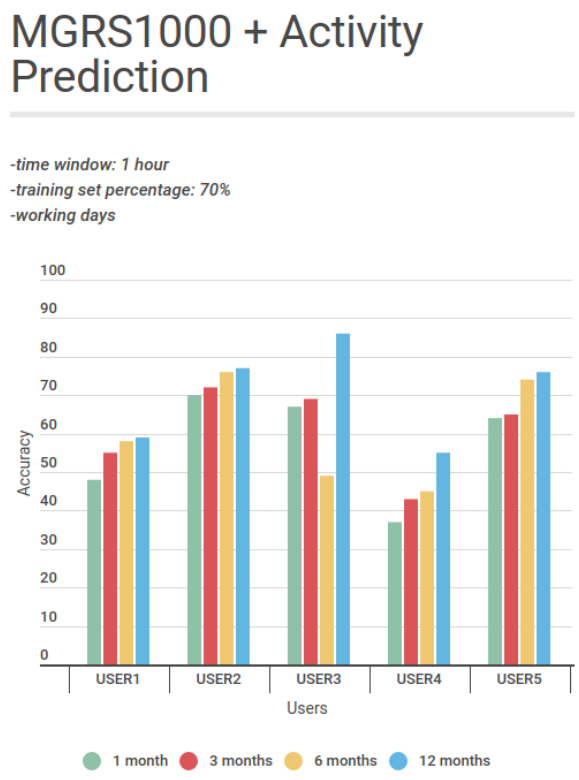

Figura 4.6: Risultati mgrs1000 e activity (1h, w.days)

Giorni lavorativi						
	mgrs10	***mgrs100***	***mgrs1000***	***mgrs10 + A***	***mgrs100 + A***	***mgrs1000 + A***
User 1	*35,7*	*38,1*	*61,3*	*29,3*	*30,4*	*55,5*
User2	*55,6*	*58,6*	*80,8*	*48,1*	*49,9*	*74,4*
User3	*51,7*	*53,4*	*74,3*	*46,4*	*47*	*68,2*
User4	*34,6*	*36,1*	*51,1*	*30,5*	*31,3*	*45,49*
User5	*57,9*	*60,2*	*75,1*	*53,6*	*55*	*70,2*

Tabella 4.1: Valori medi di precisione dell'algoritmo sui giorni lavorativi

MGRS10

Per l'utente 1, nel caso dei *dataset* della durata di 1 e 3 mesi si ottengono rispettivamente il 34,39% e il 36,57% . Sui *dataset* di dimensione 6 e 12 mesi, le performance peggiorano leggermente, con precisioni di 36,48% e 35,69% .

Per l'utente 2, abbiamo un notevole miglioramento delle performance rispetto ai risultati precedenti. Nel caso di *dataset* di 1 e 3 mesi abbiamo rispettivamente una precisione del 50.13% e del 56.04% , mentre sui 6 mesi mesi del 58.87% . La precisione diminuisce leggermente al 57.48% sui 12 mesi.

Nel caso dell'utente 3, i risultati ottenuti su 1 e 3 mesi sono molto simili a quelli dell'utente precedente: 48,89% su 1 mese e 56,64% su 3 mesi. Sui 6 mesi abbiamo una diminuzione drastica fino al 31,35% , per poi ritornare sul 70,7% sui 12 mesi.

Per l'utente 4, l'aumento della durata del *dataset* porta a costanti miglioramenti. Partiamo da 1 mese con il 23,81% , per poi arrivare al 30,43% e al 37,69% su 3 e 6 mesi. Il valore massimo è ottenuto sulla durata dei 12 mesi con un valore del 46,52% .

Nel caso dell'utente 5, l'algoritmo produce discreti risultati. Con 1 mese e 3 mesi otteniamo i valori del 54,09% e 53,81% , mentre sulla durata dei 6 mesi una precisione del 64,15% . Sui 12 mesi c'è un leggero calo delle performance, ottenendo un valore pari al 59,81% .

MGRS100

Aumentando la granularità delle coordinate *mgrs*, notiamo un leggero miglioramento per tutti gli utenti. Nel caso dell'utente 1, la precisione media aumenta del 2,4% , ottenendo i seguenti risultati:

- 1 mese: 34,61%;
- 3 mesi: 37,43%;
- 6 mesi: 41 %;
- 12 mesi: 39,6 %;

Per l'utente 2 si verifica un miglioramento della precisione media del 3% circa, con i seguenti valori individuali:

- 1 mese: 52,40%;
- 3 mesi: 58,73%;
- 6 mesi: 61,74%;
- 12 mesi: 61,26%;

Per l'utente 3 l'aumento della precisione media corrisponde al 1,7%, con i seguenti risultati ottenuti:

- 1 mese: 51,30%;
- 3 mesi: 58,44%;
- 6 mesi: 32,3%;
- 12 mesi: 71,63%;

Per l'utente 4 si verifica invece un aumeto del 1,5%, ottenendo i seguenti risultati:

- 1 mese: 25,26%;
- 3 mesi: 31,7%;
- 6 mesi: 39,32%;

- 12 mesi: 48,09%;

Mentre per l'utente 5 l'aumento in termini di precisione media corrisponde al 2,3%:

- 1 mese: 55,23%;
- 3 mesi: 55,65%;
- 6 mesi: 66,37%;
- 12 mesi: 63,76%;

MGRS1000

Cercando di predire la posizione in termini di coordinate di questo tipo, abbiamo un notevole aumento delle *performance* generali. Per l'utente 1 la precisione media assume un valore intorno al 61,3% (+25,6% su *mgrs10* e +23,2% su *mgrs100*), per l'utente 2 si arriva al 80,8% (+25,2% su *mgrs10* e +22,2% su *mgrs100*), per l'utente 3 al 74,3% (22,9+% su *mgrs10* e +20,9% su *mgrs100*), per l'utente 4 al 51,1 % (+16,5% su *mgrs10* e +15% su *mgrs100*), mentre per l'utente 5 la precisione media risultante è del 75,1% (+17,2% su *mgrs10* e +14,9% su *mgrs100*).

Di seguito vengono elencati i singoli risultati ottenuti:

- User1:
  - 1 mese: 53,67%;
  - 3 mesi: 60,57%;
  - 6 mesi: 64,5%;
  - 12 mesi: 66,42%;
- User2:
  - 1 mese: 77,95%;
  - 3 mesi: 78,17%;
  - 6 mesi: 82,71%;
  - 12 mesi: 84,38%;

- User3:
  - 1 mese: 73,23%;
  - 3 mesi: 74,42%;
  - 6 mesi: 58,97%;
  - 12 mesi: 90,87%;
- User4:
  - 1 mese: 43,07%;
  - 3 mesi: 50,06%;
  - 6 mesi: 50,58%;
  - 12 mesi: 60,74%;
- User5:
  - 1 mese: 69,77%;
  - 3 mesi: 72,33%;
  - 6 mesi: 78,97%;
  - 12 mesi: 79,33%;

MGRS 10,100,1000 e Activity

Vediamo ora come cambiano i risultati, in relazione ai valori precedentemente ottenuti, se oltre al luogo si cerca di predire anche l'attività svolta in quel preciso momento. In media, la perdita di precisione non è eccessiva, e i risultati ottenuti restano comunque discreti.

- User1:
  - mgrs10 + activity: precisione media del 29,3% (-6,4%);
  - mgrs100 + activity: precisione media del 30,4% (-7,7%);
  - mgrs1000 + activity: precisione media del 55,51% (-5,8%);
- User2:
  - mgrs10 + activity: precisione media del 48,1% (-7,5);

  - mgrs100 + activity: precisione media del 49,9 (-8,7);
  - mgrs1000 + activity: precisione media del 74,4% (-6,9%);

- User3:
  - mgrs10 + activity: precisione media del 51,7% (-5,3%);
  - mgrs100 + activity: precisione media del 53,4% (-6,4%);
  - mgrs1000 + activity precisione media del 74,3% (-6,1%)

- User4:
  - mgrs10 + activity: precisione media del 34,6% (-4,1%);
  - mgrs100 + activity: precisione media del 36,1% (-4,8%);
  - mgrs1000 + activity: precisione media del 51,1% (-5,61%)

- User5:
  - mgrs10 + activity: precisione media del 57,9% (-4,3%);
  - mgrs100 + activity: precisione media del 60,2% (-5,2%);
  - mgrs1000 + activity: precisione media del 75,1% (-4,9%);

In genere, l'utilizzo di *dataset* più ampi porta ad un aumento della precisione.

Tuttavia è interessante analizzare i risultati relativi all'utente 3: nel passaggio dall'utilizzo del *dataset* di lunghezza 1 mese a quello di lunghezza 3 mesi, si ha un incremento medio della precisione del 6% circa.

Utilizzando in seguito il *dataset* di lunghezza 6 mesi, avviene una perdita di precisione notevole, per poi riscontrare un netto aumento quando si considera il *dataset* di lunghezza 12 mesi. Da questi risultati si può dedurre che l'utente ha modificato in maniera netta la propria routine quotidiana, in relazione ai luoghi visitati, probabilmente cambiando città o posto di lavoro.

4.3 Festive days

Come mostrano i risultati seguenti, durante i giorni festivi predire informazioni relative alla mobilità di un utente è molto più complesso. Questo avviene principalmente perchè spesso un individuo sfrutta il tempo libero a disposizione per visitare nuovi posti o effettuare attività che normalmente non avrebbe il tempo di svolgere.

In generale, non solo nel caso dei giorni festivi, l'agoritmo tenderà a fornire un valore di precisione inversamente proporzionale al grado di entropia dell'utente considerato.

Per gli utenti 1 e 4 in particolare, il numero di stati predetti risulta essere davvero basso, ottenendo in alcuni casi una precisione molto vicina allo zero.

Nonostante i risultati siano decisamente migliori, anche per gli utenti 2, 3 e 5 la precisione diminuisce notevolmente.

Possiamo affermare dunque che gli utenti 2,3 e 5 sono caratterizzati da un grado di entropia minore, mentre gli utenti 1 e 4, avendo una routine meno regolare durante i giorni festivi, tendono ad avere un grado di entropia maggiore.

Come mostrato dai seguenti risultati, possiamo affermare che il grado di entropia di ogni utente aumenta durante i giorni festivi, mentre diminuisce nei giorni lavorativi.

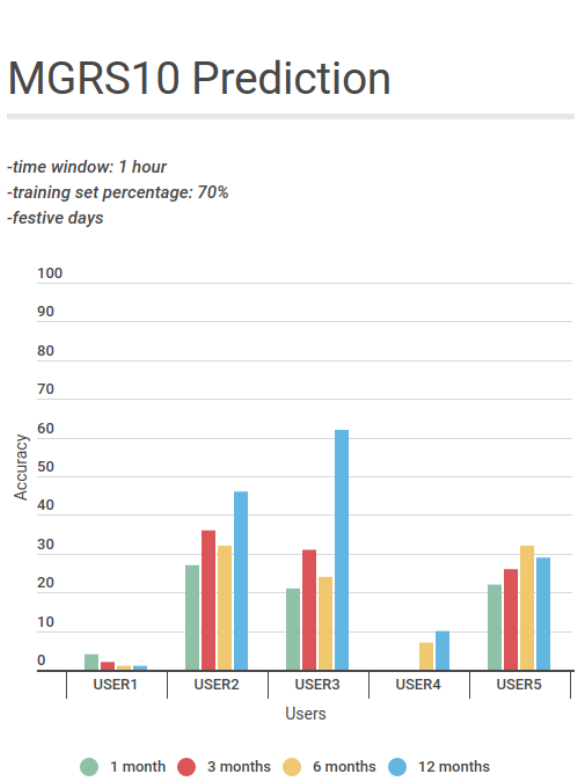

Figura 4.7: Risultati mgrs10 (1h, f.days)

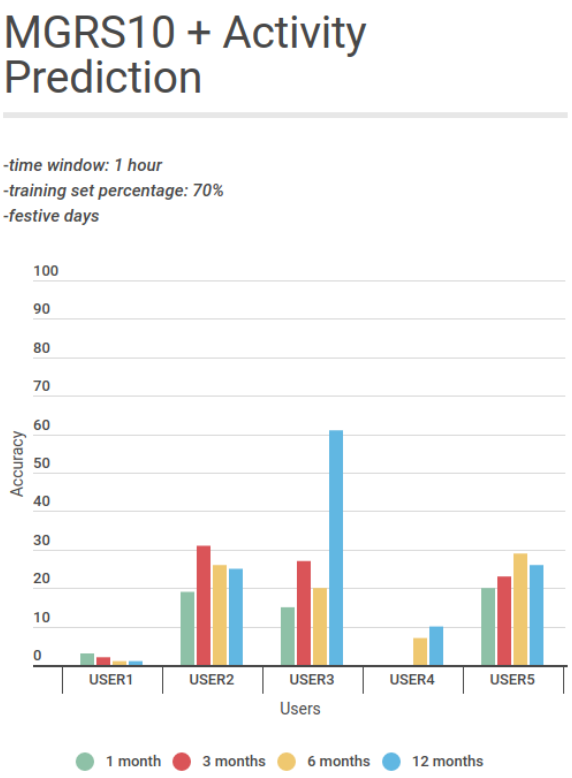

Figura 4.8: Risultati mgrs10 e activity (1h, f.days)

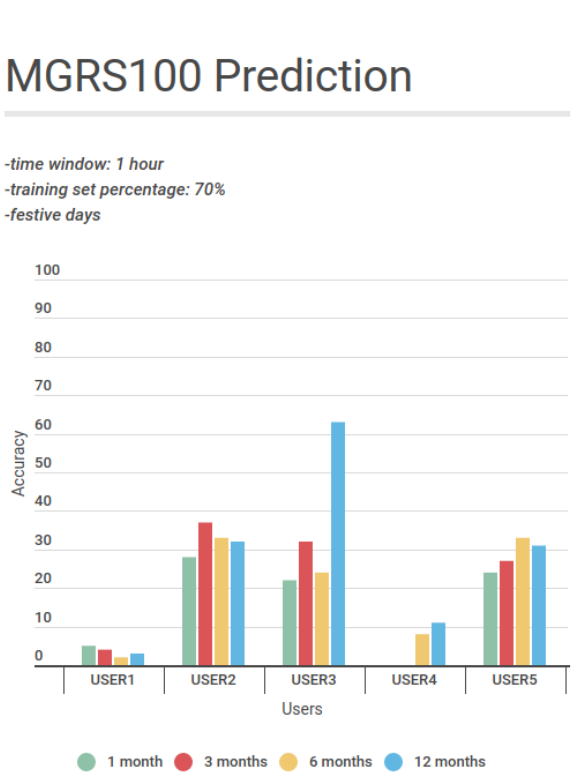

Figura 4.9: Risultati mgrs100 (1h, f.days)

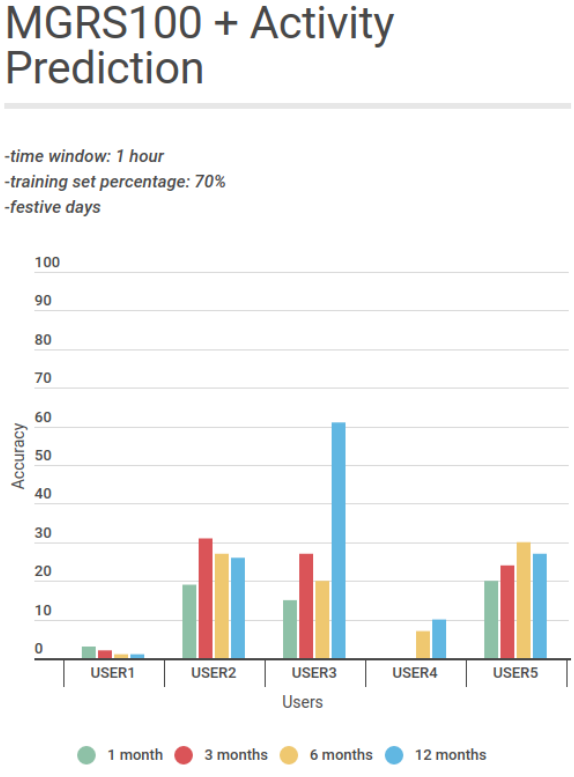

Figura 4.10: Risultati mgrs100 e activity (1h, f.days)

Giorni festivi						
	mgrs10	***mgrs100***	***mgrs1000***	***mgrs10 + A***	***mgrs100 + A***	***mgrs1000 + A***
User 1	*2,4*	*3,7*	*18,5*	*1,9*	*2,5*	*15,5*
User2	*31,9*	*33*	*52*	*25,6*	*26,3*	*45,7*
User3	*35*	*35,5*	*59,3*	*31,2*	*31,5*	*53,4*
User4	*4,5*	*5*	*8,7*	*4,4*	*4,7*	*7,1*
User5	*27,7*	*28,9*	*46,6*	*25,1*	*25,7*	*41,8*

Tabella 4.2: Valori medi di precisione dell'algoritmo sui giorni festivi

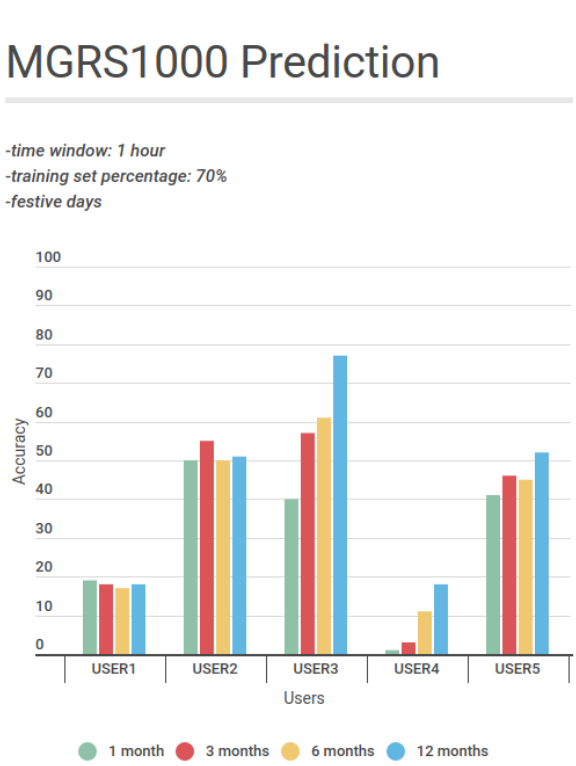

Figura 4.11: Risultati mgrs1000 (1h, f.days)

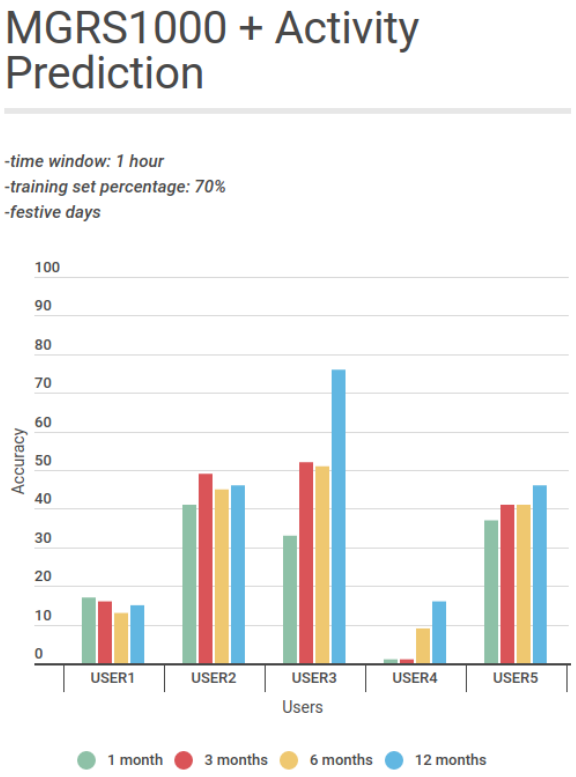

Figura 4.12: Risultati mgrs1000 e activity (1h, f.days)

MGRS10

Per l'utente 1 risulta molto difficile predire informazioni relative ai giorni festivi, soprattutto se si considera le coordinate mgrs10. Per il *dataset* di 1 mese si ha la precisione di predizione maggiore, con un risultato del 4,27%, mentre le *performance* peggiorano all'aumentare della dimensione del *dataset* (2,23% su 3 mesi, 1,48% su 6 mesi e 1,62% su 12 mesi). Per l'utente 2, su un mese otteniamo un valore del 27,36%, su 3 mesi le *performance* migliorano fino ad una precisione del 36,1%, mentre su 6 e 12 mesi calano (rispettivamente 32,62% e 31,5%). Per l'utente 3, considerando il *dataset* di 1 mese ci troviamo intorno al 21,69%, mentre il risultato migliora considerando i 3 mesi (31,33%), per poi

avere un calo sulla durata di 6 mesi (24,24%). Infine, sul *dataset* di dimensione maggiore, otteniamo il valore più alto tra i vari utenti relativamente ai giorni festivi, con una precisione di circa 63%. Relativamente all'utente 4 invece, possiamo notare come sui *dataset* di 1 e 3 mesi, l'algoritmo non riesca ad effettuare nessuna predizione, con una precisione dello 0% in entrambi i casi. Nonostante ciò, considerando i *dataset* di 6 e 12 mesi, si ha un leggero miglioramento, con i rispettivi valori di 7,51% e 10,68%. Per l'utente 5 i risultati ottenuti possono essere considerati discreti: su un mese la precisione ottenuta è del 22,96%, su 3 mesi del 26,24%, mentre raggiunge il valore massimo sulla durata dei 6 mesi con un 32,23%. In seguito, considerando un periodo di 12 mesi, avviene una leggera perdita di precisione (29,6%).

MGRS100

In questo caso, all'aumento della granularità *mgrs*, corrisponde un aumento minimo della precisione generale di predizione. Per l'utente 1, si verifica un aumento di precisione media del 1,3%, dove la precisione relativi ai singoli *dataset* assume i seguenti valori:

- 1 mese: 5,45%;
- 3 mesi: 4,29%;
- 6 mesi: 2,16%;
- 12 mesi: 3,1%;

Per l'utente 2 si verifica un miglioramento della precisione media del 1,1% circa, con i seguenti valori individuali:

- 1 mese: 28,2%;
- 3 mesi: 37,1%;
- 6 mesi: 33,67%;
- 12 mesi: 32,97%;

Per l'utente 3 l'aumento della precisione media corrisponde allo 0,5%, con i seguenti risultati ottenuti:

- 1 mese: 22,35%;
- 3 mesi: 32,21%;
- 6 mesi: 24,47%;
- 12 mesi: 63,06%;

Anche per l'utente 4, l'aumento del valore medio di precisione è dello 0,5%, e sui singoli dataset otteniamo i seguenti risultati:

- 1 mese: 0,06%;
- 3 mesi: 0,09%;
- 6 mesi: 8,27%;
- 12 mesi: 11,61%;

Mentre per l'utente 5 si ha un aumento medio del 1,2% complessivo, con i seguenti valori relativi ai singoli dataset:

- 1 mese: 24,23%;
- 3 mesi: 27,16%;
- 6 mesi: 33,19%;
- 12 mesi: 31,37%;

MGRS1000

Cercando di predire la posizione in termini di coordinate di questo tipo, abbiamo un notevole aumento delle *performance* generali per quanto riguarda gli utenti 2,3 e 5, un discreto aumento per l'utente 1 e un leggero aumento per l'utente 4. Riguardo l'utente 1 la precisione media assume un valore intorno al 18,5% (+14,8% su *mgrs100* e +16,1 su *mgrs10*), per l'utente 2 si arriva al 52% (+19% su *mgrs100* e +20,1% su *mgrs10*), per l'utente 3 al 59,3% (+23,8% su *mgrs100* e +24,3% su *mgrs10*), per l'utente 4 al 8,7 % (+3,7% su *mgrs100* e +4,2% su *mgrs10*), mentre per l'utente 5 la precisione media risultante è del 46,6% (+17,7% su *mgrs100* e +18,9% su *mgrs10*).

Di seguito vengono elencati i singoli risultati ottenuti:

- User1:
  - 1 mese: 19,19%;
  - 3 mesi: 18,52%;
  - 6 mesi: 17,65%;
  - 12 mesi: 18,73%;
- User2:
  - 1 mese: 50,89%;;
  - 3 mesi: 55,93%;
  - 6 mesi: 50,27%;
  - 12 mesi: 51,15%;
- User3:
  - 1 mese: 40,44%;
  - 3 mesi: 57,72%;
  - 6 mesi: 61,52%;
  - 12 mesi: 77,72%;
- User4:
  - 1 mese: 1,88%;
  - 3 mesi: 3,55%;
  - 6 mesi: 11,01%;
  - 12 mesi: 18,62%;
- User5:
  - 1 mese: 41,41%;
  - 3 mesi: 46,76%;
  - 6 mesi: 45,91%;
  - 12 mesi: 52,48%;

MGRS 10,100,1000 e Activity

Analizziamo ora i risultati relativi alla predizione di luogo e attività nei giorni festivi. Per gli utenti 1 e 4 la precisione dell'algoritmo è molto bassa, con una diminuzione media della precisione, rispetto ai risultati precedenti, compresa tra 0,5% e 1,6%. Per gli utenti 2,3 e 5 i risultati sono migliori, ma decisamente minori rispetto a quelli riscontrati considerando i giorni lavorativi. I valori medi della precisione dell'algoritmo per ogni utente sono i seguenti:

- User1:
  - mgrs10 + activity: precisione media del 1,9% (-0,5%);
  - mgrs100 + activity: precisione media del 2,5% (-1,2%);
  - mgrs1000 + activity: precisione media del 15,5% (-3%);
- User2:
  - mgrs10 + activity: precisione media del 25,6% (-6,3%);
  - mgrs100 + activity: precisione media del 26,3% (-6,7%);
  - mgrs1000 + activity: precisione media del 45,7% (-6,3%);
- User3:
  - mgrs10 + activity: precisione media del 31,2% (-3,8%);
  - mgrs100 + activity: precisione media del 31,5% (-4%);
  - mgrs1000 + activity precisione media del 53,4% (-5,9%)
- User4:
  - mgrs10 + activity: precisione media del 4,4% (-0,1%);
  - mgrs100 + activity: precisione media del 4,7% (-0,3%);
  - mgrs1000 + activity: precisione media del 7,1% (-1,6%)
- User5:
  - mgrs10 + activity: precisione media del 25,1% (-2,6%);
  - mgrs100 + activity: precisione media del 25,7% (3,2%);
  - mgrs1000 + activity: precisione media del 41,8% (-4,8%);

4.4 Diminuzione del parametro *time window*

Analizziamo ora i risultati ottenuti in seguito alla diminuzione dell'intervallo di tempo. È stato utilizzato un valore pari a 15 minuti, in modo tale da ottenere predizioni relative ad un preciso momento della giornata.

Nel complesso, si ottiene un grado di precisione minore rispetto all'uso di intervalli di tempo maggiori, ma allo stesso tempo un considerevole incremento dei valori delle probabilità di transizione dei vari stati della catena di Markov.

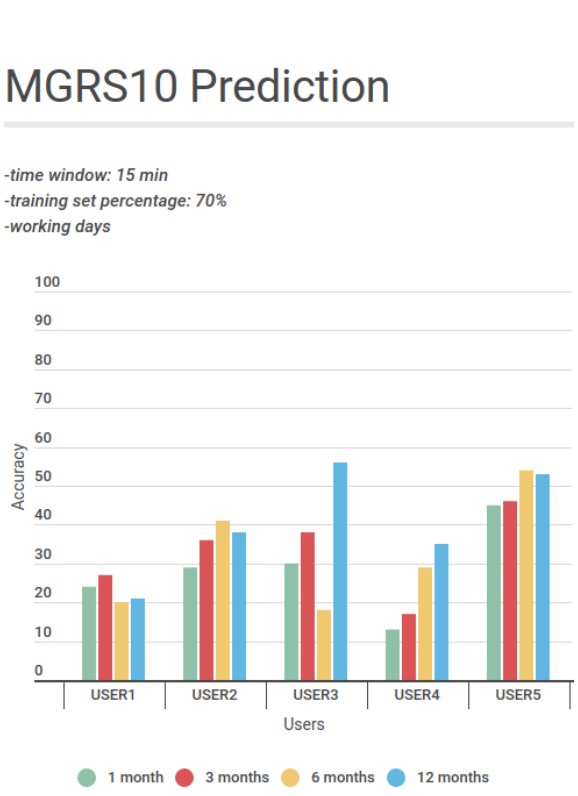

Figura 4.13: Risultati mgrs10 (15min, w.days)

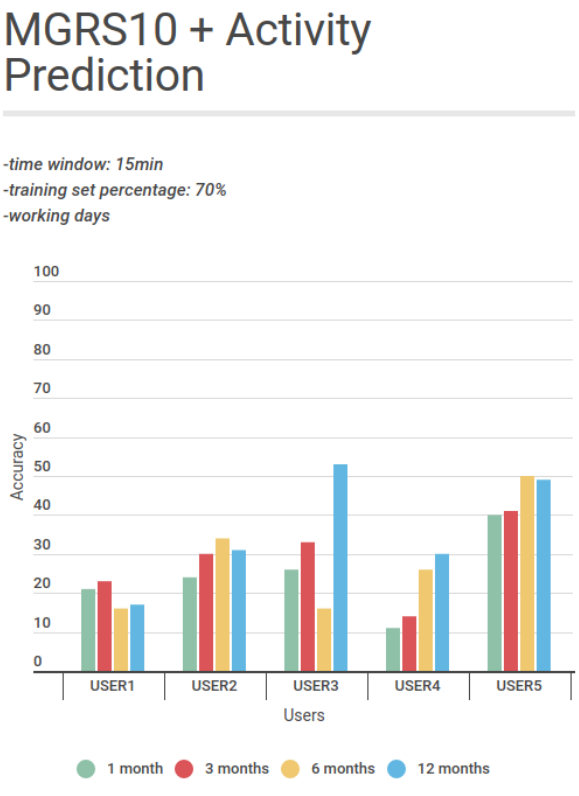

Figura 4.14: Risultati mgrs10 e activity (15min, w.days)

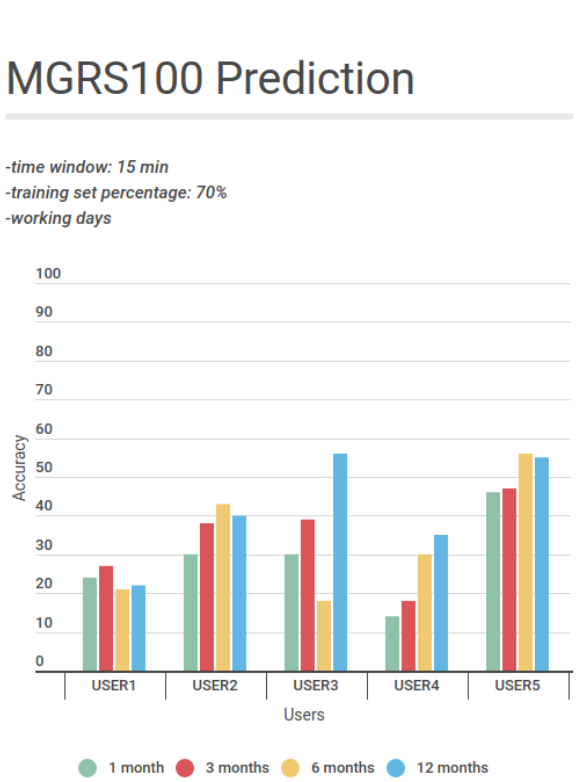

Figura 4.15: Risultati mgrs100 (15min, w.days)

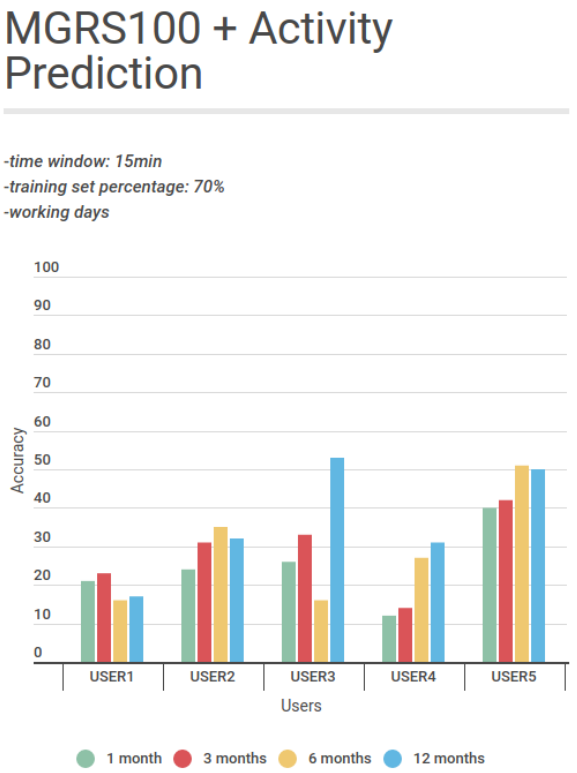

Figura 4.16: Risultati mgrs100 e activity (15min, w.days)

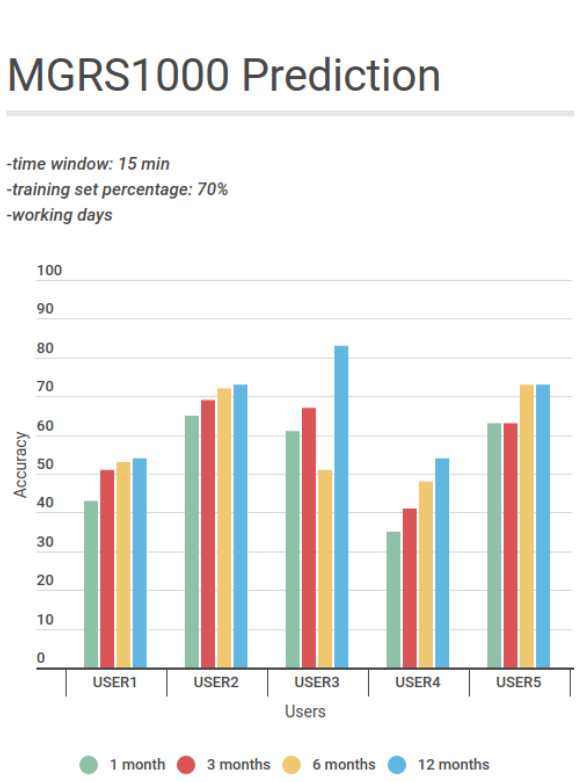

Figura 4.17: Risultati mgrs1000 (15min, w.days)

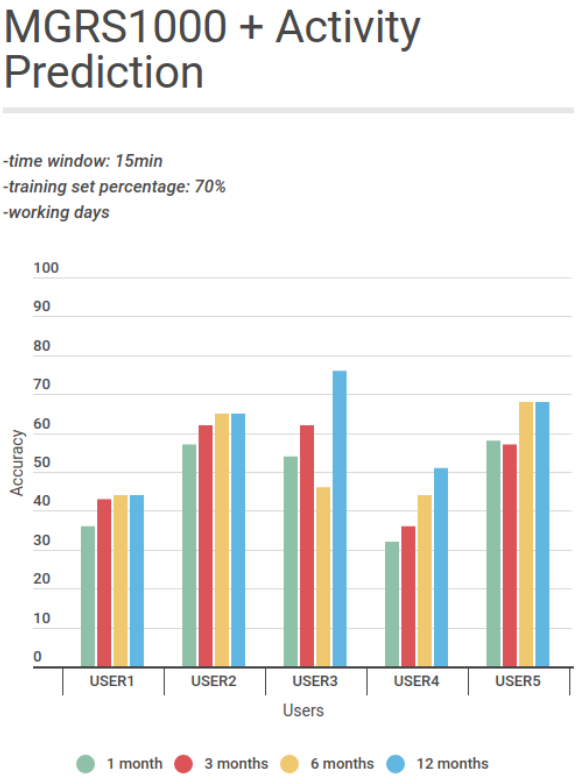

Figura 4.18: Risultati mgrs1000 e act. (15min, w.days)

Diminuzione dell'intervallo di tempo						
	mgrs10	***mgrs100***	***mgrs1000***	***mgrs10 + A***	***mgrs100 + A***	***mgrs1000 + A***
User 1	*23,5*	*24*	*50,89*	*19,5*	*19,8*	*42,3*
User2	*36,7*	*37,9*	*70.19*	*30,4*	*31*	*62,6*
User3	*36,15*	*36,5*	*65,9*	*31,4*	*32,3*	*60,1*
User4	*24,1*	*24,8*	*44,9*	*20,9*	*21,3*	*41,2*
User5	*50,2*	*51,3*	*68,4*	*45,71*	*46,21*	*63,2*

Tabella 4.3: Valori medi di precisione dell'algoritmo su intervallo di tempo di 15 minuti

MGRS10

Per l'utente 1, si ottiene una precisione del 24,39% sul *dataset* di 1 mese, che aumenta al 27,13% nel caso del *dataset* si consideri il *dataset* di 3 mesi. Aumentando ulteriormente la dimensione del *dataset*, la precisione diminuisce al 20,97% e al 21,85% nel caso dei *dataset* di 6 e 12 mesi. Per l'utente 2, la precisione generale è maggiore. Su 1 mese otteniamo il 29,79% di casi predetti, per poi avere valori di precisione maggiori fino al 36,97% e al 41,44% nel caso di *dataset* di 3 e 6 mesi, mentre considerando una durata di 12 mesi scende al 38,6%. Nel caso dell'utente 3, si parte da una precisone del 30,7% su 1 mese e del 38,89% su una durata di 6 mesi, mentre la precisione diminuisce in modo netto fino al 18,59% nel caso venga utilizzato il *dataset* di durata 6 mesi. Sul *dataset* relativo ai 12 mesi invece, si ha un notevole incremento del valore di precisione, che assume valore 56,66%. Nel caso dell'utente 4, i valori di precisione sono particolarmente bassi. Sulla durata di 1 mese, si ottiene il 13,95%, mentre le performance tendono a crescere per i *dataset* di durata maggiore, con valori rispettivi di 17,63%, 29,81% e 35,04% su 3,6 e 12 mesi. Le migliori performance vengono ottenute utilizzando il *dataset* dell'utente 5. Escludendo il *dataset* di dimensione maggiore, il valore della precisione aumenta in rapporto all'incremento della durata di tempo considerata, partendo da un 45,8% per un mese, 46,68% sui 3 mesi, arrivando poi al 54,64% se si considera il *dataset* di 6 mesi. Infine, sui 12 mesi diminuisce leggermente, dove otteniamo un valore del 53,65%.

MGRS100

Nel caso dell'utilizzo di una finestra di tempo minore, nel passaggio da mgrs10 a mgrs100, l'aumento in termini di performance è minimo. Per l'utente 1, si verifica un aumento di precisione media dello 0,5%, dove la precisione relativa ai singoli dataset assume i seguenti valori:

- 1 mese: 24,39%;
- 3 mesi: 27,26%;
- 6 mesi: 21,85%;
- 12 mesi: 22,68%;

Per l'utente 2 si verifica un miglioramento della precisione media del 1,2% circa, con i seguenti valori individuali:

- 1 mese: 30,17%;
- 3 mesi: 38,18%;
- 6 mesi: 43,03%;
- 12 mesi: 40,43%;

Per l'utente 3 l'aumento della precisione media corrisponde allo 0,35%, con i seguenti risultati ottenuti:

- 1 mese: 30,84%;
- 3 mesi: 39,58%;
- 6 mesi: 18,99%;
- 12 mesi: 56,96%;

Anche per l'utente 4, l'aumento del valore medio di precisione è dello 0,7%, e sui singoli dataset otteniamo i seguenti risultati:

- 1 mese: 14,32%;
- 3 mesi: 18,21%;

- 6 mesi: 30,87%;
- 12 mesi: 35,91%;

Mentre per l'utente 5 si ha il maggior beneficio in termini di valore medio, con un aumento del 1,1% complessivo, seguenti valori relativi ai singoli dataset:

- 1 mese: 46,36%;
- 3 mesi: 47,36%;
- 6 mesi: 56,21%;
- 12 mesi: 55,52%;

MGRS1000

Aumentando ulteriormente la granularità mgrs, avviene un netto aumento della precisione generale. In particolare, per i primi quattro utenti il valore della precisione tende quasi a raddoppiare. Per l'utente 1 la precisione media raggiunge il 50,8% (+27,3% su *mgrs10* e + 26,8% su *mgrs100*), per l'utente 2 la precisione media è del 70,1% (+33,4% su *mgrs10* e + 32,3% su *mgrs100*), per l'utente 3 del 65,9% (+29,8% su *mgrs10* e + 29,4% su *mgrs100*), mentre per l'utente 4 il valore della precisione è del 44,9% (+20,8% su *mgrs10* e + 20,1% su *mgrs100*). Infine, per l'utente 5 si ottiene un valore del 68,4% (+18,2% su *mgrs10* e + 17,1% su *mgrs100*).

I singoli risultati ottenuti sono i segenti:

- User1:
  - 1 mese: 43,34%;
  - 3 mesi: 51,6%;
  - 6 mesi: 53,76%;
  - 12 mesi: 54,86%;
- User2:
  - 1 mese: 65,51%;;
  - 3 mesi: 69,72%;

  - 6 mesi: 72,36%;
  - 12 mesi: 73,17%;

- User3:
  - 1 mese: 61,17%;
  - 3 mesi: 67,53%;
  - 6 mesi: 51,99%;
  - 12 mesi: 83,13%;

- User4:
  - 1 mese: 35,47%;
  - 3 mesi: 41,29%;
  - 6 mesi: 48,24%;
  - 12 mesi: 54,96%;

- User5:
  - 1 mese: 63,93%;
  - 3 mesi: 63,34%;
  - 6 mesi: 73,38%;
  - 12 mesi: 73,04%;

MGRS 10,100,1000 e Activity

Considerando anche l'attività, nel complesso si ottiene una diminuzione media della precisione compresa tra il 3,2% e l 8,5%. Per l'utente 4 si ottiene la perdita di precisione minore, mentre la predizione dell'attività influisce in negativo sulle per formance in modo particolare per l'utente 2 .

I valori medi della precisione dell'algoritmo per ogni utente sono i seguenti:

- User1:
  - mgrs10 + activity: precisione media del 19,5% (-4%);
  - mgrs100 + activity: precisione media del % 19,8(-4,2%);
  - mgrs1000 + activity: precisione media del % 42,3(-8,5%);

- User2:
  - mgrs10 + activity: precisione media del 30,4% (-6,3%);
  - mgrs100 + activity: precisione media del 31% (-6,9%);
  - mgrs1000 + activity: precisione media del 62,6% (-7,5%);
- User3:
  - mgrs10 + activity: precisione media del 31,4% (-4,7%);
  - mgrs100 + activity: precisione media del 32,3% (-4,2%);
  - mgrs1000 + activity precisione media del 60,1% (-5,8%)
- User4:
  - mgrs10 + activity: precisione media del 20,9% (-3,2%);
  - mgrs100 + activity: precisione media del 21,3% (-3,5%);
  - mgrs1000 + activity: precisione media del 41,2% (-3,7%)
- User5:
  - mgrs10 + activity: precisione media del 45,7% (-4,5%);
  - mgrs100 + activity: precisione media del 46,2% (-5,1%);
  - mgrs1000 + activity: precisione media del 63,2% (-5,2%);

4.5 Incremento del parametro *time window*

Ad un aumento dell'intervallo di tempo considerato, in questo caso utilizzando un'ampiezza di 4 ore, segue un aumento notevole della precisione, che in alcuni casi oltrepassa il 90%. Allo stesso tempo, bisogna tener conto della drastica diminuzione delle singole probabilità di transizione, in quanto avremo un numero minore di *cluster*, ma ognuno di essi sarà costituito da un numero maggiore di *record*.

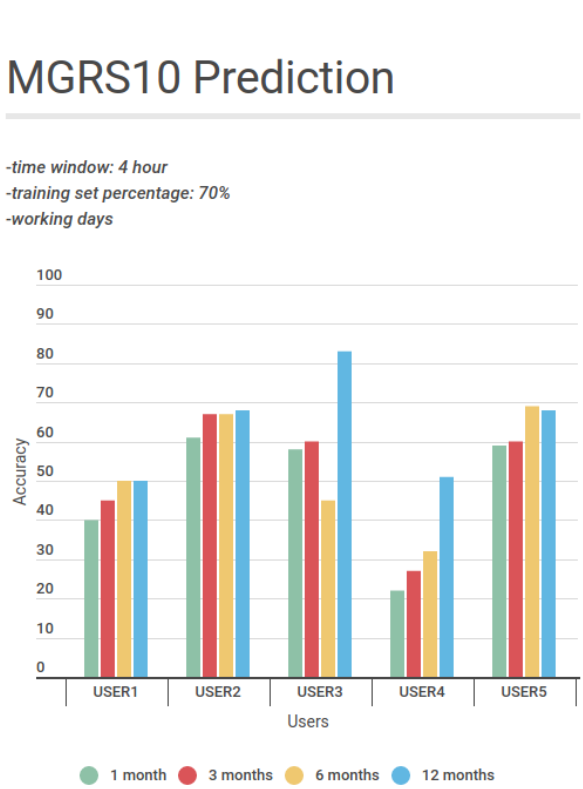

Figura 4.19: Risultati mgrs10 (4h, w.days)

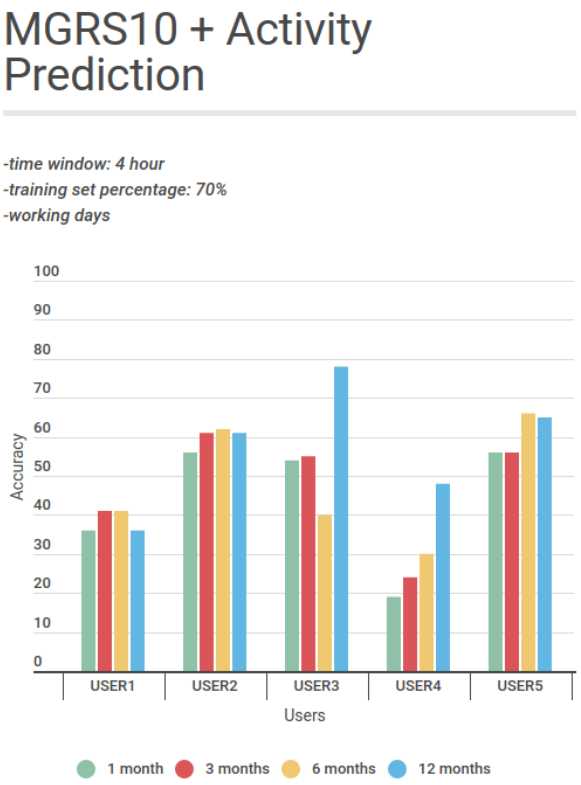

Figura 4.20: Risultati mgrs10 e activity (4h, w.days)

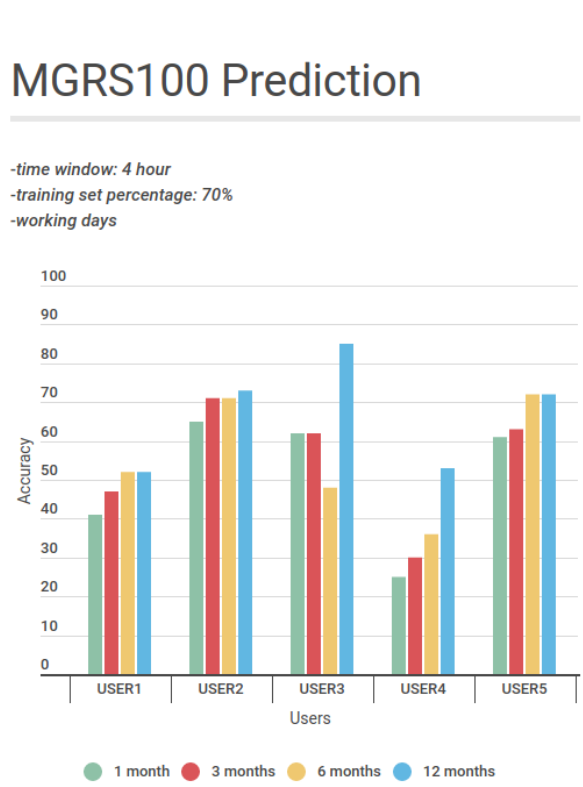

Figura 4.21: Risultati mgrs100 (4h, w.days)

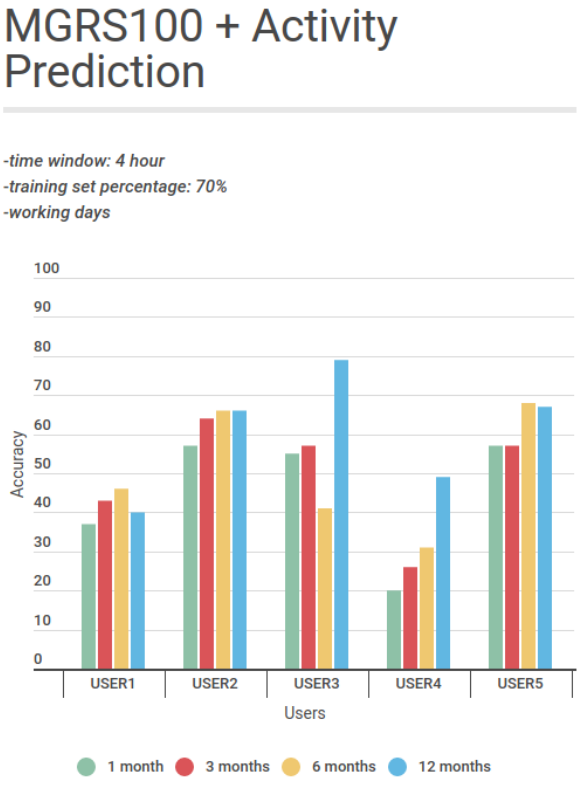

Figura 4.22: Risultati mgrs100 e activity (4h, w.days)

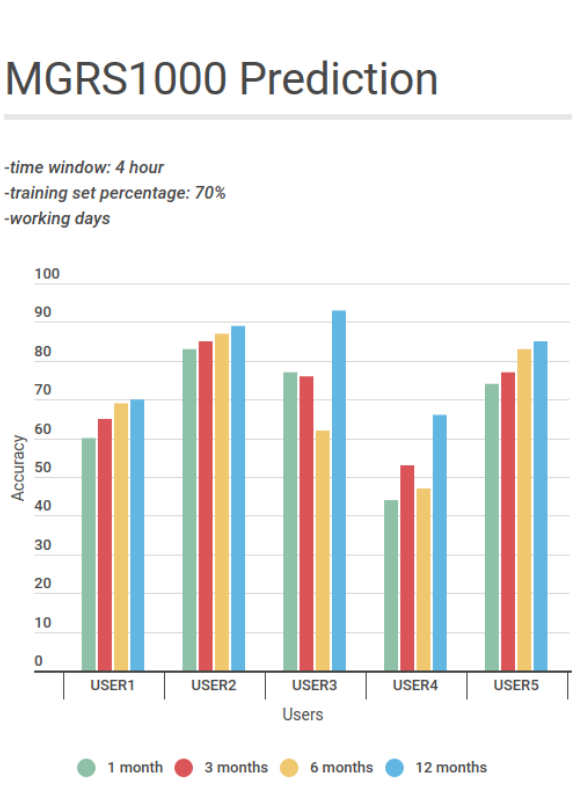

Figura 4.23: Risultati mgrs1000 (4h, w.days)

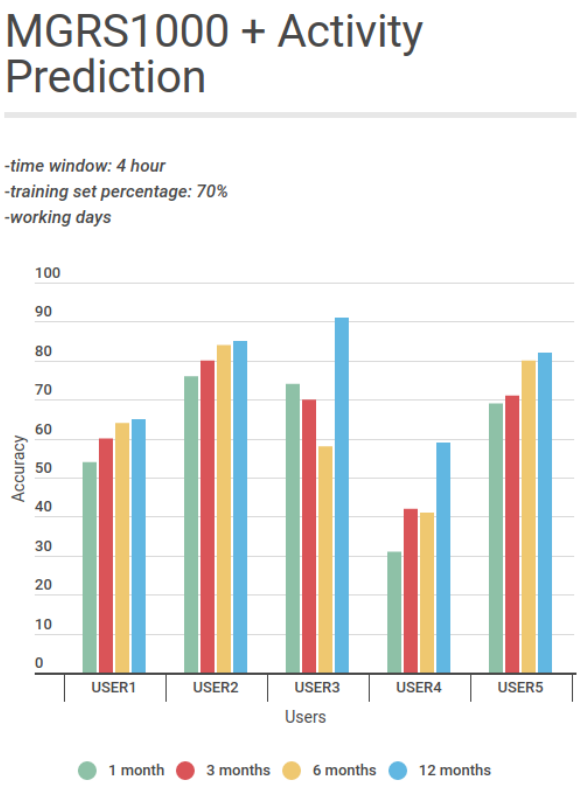

Figura 4.24: Risultati mgrs1000 e activity (4h, w.days)

	Incremento dell'intervallo di tempo					
	mgrs10	***mgrs100***	***mgrs1000***	***mgrs10 + A***	***mgrs100 + A***	***mgrs1000 + A***
User 1	*46,6*	*48,6*	*66,6*	*38,9*	*41,9*	*61,1*
User2	*66,2*	*70,5*	*86,5*	*60,6*	*63,6*	*81,4*
User3	*62,3*	*64,6*	*77,6*	*57,3*	*58,3*	*73,6*
User4	*33,5*	*36,3*	*52,7*	*30,9*	*32,1*	*44*
User5	*65,4*	*67,4*	*80,3*	*61,1*	*62,6*	*75,8*

Tabella 4.4: Valori medi di precisione dell'algoritmo su intervallo di tempo di 4 ore

MGRS10

Per l'utente 1, la precisione ottenuta sul *dataset* di un mese è di 40,47%, che aumenta sui *dataset* di 3 e 6 mesi fino a raggiungere i corrispettivi valori del 45,55% e 50,65%. Sul *dataset* di 12 mesi invece, si ha una leggera perdita di precisione ottenendo un valore di 50,02%. Per l'utente 2, un aumento della dimensione del *dataset* porta ad un incremento delle performance, e l'algoritmo ottiene una precisione iniziale di 61,45% su un mese ,per poi salire al 67,35% e al 67,7% nel caso si considerino i *dataset* della durata di 3 e 6 mesi. Il valore massimo è ottenuto sui 12 mesi, dove la precisione è uguale al 68,62%.

Per l'utente 3, su un mese otteniamo il 58,83% di *match* avvenuti, con un incremento sui 3 mesi per un valore di 60,49%. Considerando il *dataset* di 6 mesi, la precisione scende al 46,57%, per poi ottenere un ottimo risultato in relazione ai 12 mesi, con un valore di 83,49%. L'utente 4, è quello per cui l'algoritmo ottiene i risultati peggiori. Considerando il *dataset* di un mese, la precisione assume un valore del 22,88%, per poi aumentare fino al 27,57% sui 3 mesi e al 32,81% sui 6 mesi. Infine, le *perfomance* migliorano notevolmente se si considera il periodo di tempo di 12 mesi, ottenendo una precisione del 51,08%. Per l'utente 5, sul *dataset* di un mese l'algoritmo fornisce una precisione del 59,51%, che aumenta a 60,22% e 69,45% se si considerano rispettivamente i *dataset* di 3 e 6 mesi. Nel caso del *dataset* di 12 mesi, la precisione diminuisce leggermente assumendo un valore del 68,56%.

MGRS100

Anche in questo caso, non si hanno grandi miglioramenti utilizzando una granularità *mgrs* maggiore. Per l'utente 1, si verifica un aumento di precisione media del 2%, dove la precisione relativi ai singoli *dataset* assume i seguenti valori:

- 1 mese: 41,75%;
- 3 mesi: 47,87%;
- 6 mesi: 52,78%;
- 12 mesi: 52,26%;

Per l'utente 2 si verifica un miglioramento della precisione media del 4,3% circa, con i seguenti valori individuali:

- 1 mese: 65,56%;
- 3 mesi: 71,03%;
- 6 mesi: 71,85%;
- 12 mesi: 73,57%;

Per l'utente 3 l'aumento della precisione media corrisponde allo 2,3%, con i seguenti risultati ottenuti:

- 1 mese: 62,26%;
- 3 mesi: 62,61%;
- 6 mesi: 48,82%;
- 12 mesi: 85,09%;

Per l'utente 4, l'aumento del valore medio di precisione è dello 0,8%, e sui singoli dataset otteniamo i seguenti risultati:

- 1 mese: 25,79%%;
- 3 mesi: 30,33%;
- 6 mesi: 36,11%;
- 12 mesi: 53,18%;

Mentre per l'utente 5 si ha un incremento medio del 2% ,con i seguenti valori relativi ai singoli *dataset*:

- 1 mese: 61,67%;
- 3 mesi: 63,03%;
- 6 mesi: 72,87%;
- 12 mesi: 72,16%;

MGRS1000

Anche in questo caso, cercando di predire la posizione in termini di coordinate *mgrs1000*, si ottengono ottimi risultati.

Per l'utente 1, la precisione media risultante raggiunge il 66,6% (+20% su *mgrs10* e +18% su *mgrs100*), per l'utente 2 otteniamo un valore medio del 86,5% (+20,3% su *mgrs10* e +16% su *mgrs100*), mentre per l'utente 3 il valore della precisione raggiunge il 77,6% (+15,3% su *mgrs10* e +13% su *mgrs100*). Per l'utente 4 otteniamo il grado di affidabilità minore, con un valore del 52,7% (+19,2% su *mgrs10* e +16,4% su *mgrs100*). Infine per l'utente 5, il valore medio di precisione è uguale a 80,3% (+14,9% su *mgrs10* e +12,5% su *mgrs100*).

I singoli risultati ottenuti sono i segenti:

- User1:
  - 1 mese: 60,92%;
  - 3 mesi: 65,43%;
  - 6 mesi: 69,54%;
  - 12 mesi: 70,64%;
- User2:
  - 1 mese: 83,72%;;
  - 3 mesi: 85,52%;
  - 6 mesi: 87,72%;
  - 12 mesi: 89,38%;
- User3:
  - 1 mese: 77,92%;
  - 3 mesi: 76,04%;
  - 6 mesi: 62,64%;
  - 12 mesi: 93,82%;
- User4:
  - 1 mese: 44,43%;
  - 3 mesi: 53,03%;
  - 6 mesi: 47,44%;
  - 12 mesi: 66,16%;
- User5:
  - 1 mese: 74.49%;
  - 3 mesi: 77.57%;
  - 6 mesi: 83.66%;
  - 12 mesi: 85.66%;

MGRS 10,100,1000 e Activity

Considerando anche l'attività, nel complesso si ottiene una diminuzione media della precisione compresa tra il 3,2% e l 8,5%. Per l'utente 4 si ottiene la perdita di precisione minore, mentre la predizione dell'attività influisce in negativo sulle *performance* in modo particolare per l'utente 2 .

I valori medi della precisione dell'algoritmo per ogni utente sono i seguenti:

- User1:
  - mgrs10 + activity: precisione media del 38,9% (-7,7%);
  - mgrs100 + activity: precisione media del 41,9% (-6,7%);
  - mgrs1000 + activity: precisione media del 61,1% (-5,5%);
- User2:
  - mgrs10 + activity: precisione media del 60,6% (-5,6%);
  - mgrs100 + activity: precisione media del 63,6% (-6,9%);
  - mgrs1000 + activity: precisione media del 81,4% (-5,1%);
- User3:
  - mgrs10 + activity: precisione media del 57,3% (-5%);
  - mgrs100 + activity: precisione media del 58,3% (-6,3%);
  - mgrs1000 + activity precisione media del 73,6% (-4%)
- User4:
  - mgrs10 + activity: precisione media del 30,9% (-2,6%);
  - mgrs100 + activity: precisione media del 32,1% (-6,9%);
  - mgrs1000 + activity: precisione media del 44% (-8,7%)
- User5:
  - mgrs10 + activity: precisione media del 61,1% (-4,3%);
  - mgrs100 + activity: precisione media del 62,6% (-4,8);
  - mgrs1000 + activity: precisione media del 75,8% (-4,5);

Conclusioni

In questo lavoro di tesi, abbiamo inizialmente analizzato il notevole sviluppo tecnologico che sta investendo il pianeta negli ultimi anni, analizzando le principali tecnologie che hanno contribuito a questo processo e l'impatto che hanno prodotto nella vita quotidiana di ognuno di noi. Abbiamo mostrato poi, come l'insieme di queste tecnologie (dispositivi *IoT*, *smartphone*, etc..), sia in grado di generare una mole di dati enorme, che necessita di essere analizzata e processata al fine di erogare numerosi servizi.

Successivamente, ci siamo focalizzati sui rischi per la *privacy* derivanti dal processo di raccolta e analisi dei dati stessi, di cui spesso gli utenti non sono a conoscenza. In particolare, si è voluto evidenziare come, nel caso determinati dati fossero ottenuti da terzi, sarebbe possibile arrecare gravi danni in termini di *privacy*.

Abbiamo mostrato quindi come sia possibile, a partire da un *dataset* contenente informazioni relative agli spostamenti e all'attività svolta da un individuo, predire i luoghi e le attività svolte in futuro.

A tal scopo, abbiamo descritto nei particolari le varie fasi di creazione di un *framework* scritto nel linguaggio di programmazione *Python* , basato su un un *processo Markoviano* atto ad effettuare questo lavoro di predizione. Nello specifico, il *framework* costruisce un modello probabilistico utilizzando le informazioni del passato, al fine di infrerire informazioni relative al futuro.

Nella parte finale di questo lavoro, abbiamo analizzato i risultati forniti dall'algoritmo in termini di precisione ottenuta relativamente ad una serie di utenti, mostrando come, specialmente nei giorni lavorativi, sia possibile predire con un'elevata accuratezza i luoghi visitati e l'attività svolta da una persona. Nei giorni festivi, effettuare questa operazione risulta più difficile, ma per gli utenti caratterizzati da un grado di entropia basso il *framework* fornisce comunque un discreto valore medio di precisione.

Inoltre, combinando i risultati siamo riusciti a mostrare le differenze, in termini di entropia, dello stile di vita relativo ai vari utenti.

Gli obiettivi prefissati inizialmente, sono stati quindi raggiunti con ottimi risultati.

In seguito ai risultati mostrati da questa analisi, si spera che ogni singola persona comprenda, oltre numerosi servizi offerti dai vari dispositivi tecnologici, il grande potenziale ma allo stesso tempo i grandi rischi che risiedono nella grande mole di dati raccolti, che giorno dopo giorno, continua a crescere a dismisura.

Bibliografia

[1] Murat Ali Bayir, Murat Demirbas, and Nathan Eagle. "Mobility profiler: A framework for discovering mobility profile of cell phone users." *Human Behavior in Ubiquitous Environments: Modeling of Human Mobility Patterns, 2010.*

[2] Luca Bedogni, Marco Di Felice, and Luciano Bononi. " Context-aware Android applications through transportation mode detection techniques." *Wireless Communications and Mobile Computing, 2016.*

[3] V. Caiati, L. Bedogni, L. Bononi, F. Ferrero, M. Fiore, and A.Vesco. " Estimating urban mobility with open data: A case study in Bologna." *IEEE International Smart Cities Conference, 2016.*

[4] S. Herborn, H.Petander, and M.Ott. " Predictive context aware mobility handling." *2008 International Conference on Telecommunications.*

[5] Q. Lv, Y. Qiao, N. Ansari, and J. Yand. "Big data driven hidden markov model based individual mobility prediction at points of interest".
IEEE Transaction on Vehicular Technology, 2017.

[6] Jierui Xie, Bart Piet Knijnenburg, and Hongxia Jin. "Location sharing privacy preference: Analysis and personalized recommendation." *Proceedings of the 19th International Conference on Intelligent User Interfaces, 2014.*

[7] Yang Ye, Yu Zheng, Yukun Chen, Jianhua Feng, and Xing Xie. "Mining individual life pattern based on location history." *Proceedings of the 2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware, 2009.*

[8] Y. Wang, Y.J. Chen, J.Yang, M.Gruteser, R. P. Martin, H. Liu, L- Liu, and C.Karatas. "Determining driver phone use by exploiting smartphone integrated sensors".
IEEE Transaction on Mobile Computing (2016).

[9] P. Venkitasubramaniam. "Privacy in stochastic control: A markov decision process perispective". *51st Annual Allerton Conference on Communication, Control, and Computing (Allerton), 2013.*

[10] James McInerney, Sebastian Stein, Alex Rogers, Nicholas R. Jennings. "Breaking the habit: Measuring and predicting departures from routine in individual human mobility"
Pervasive Mob. Comp. (2013).

[11] L. Bedogni, M. Levorato. *"Forecasting Heterogeneous User Contexts with Incomplete Data".*

[12] L. Bedogni, M. Levorato. *"Rising User Privacy Against Predictive Context Awareness through Adversarial Information Injection".*

[13] Anind K. Dey. "Understanding and Using Context" *Personal Ubiquitous Comput. (2001).*

[14] N. P. Kuruvatti and H. D. Schotten. "Framework to support mobility context awareness in cellular networks". *IEEE 83rd Vehicular Technology Conference (VTC Spring), 2016.*

[15] Kalayan P. Subbu, Athanasios V. Vasilakos. *"Big Data for Context Aware Computing – Perspectives and Challenges".*