UNIVERSITÀ
DEGLI STUDI
DI PALERMO

Progetto e sviluppo di un sistema per il riconoscimento delle attività svolte dagli utenti tramite dispositivi mobili

Tesi di Laurea Magistrale in Ingegneria Informatica

F. Concone

Relatore: Prof. Giuseppe Lo Re

Correlatore: Ing. Marco Morana

UNIVERSITÀ DEGLI STUDI DI PALERMO

SCUOLA POLITECNICA

LAUREA MAGISTRALE INGENGERIA INFORMATICA

PROGETTO E SVILUPPO DI UN SISTEMA PER IL RICONOSCIMENTO DELLE ATTIVITÀ SVOLTE DAGLI UTENTI TRAMITE DISPOSITIVI MOBILI

Tesi di laurea di:

Dott. Federico CONCONE

Relatore:

Ch.mo Prof. Giuseppe LO RE

Controrelatore:

Ing. Alessandra DE PAOLA

Correlatore:

Ing. Marco MORANA

Sommario

Negli ultimi anni gli smartphone sono diventati uno strumento indispensabile per la vita di tutti i giorni. Infatti, oltre alle semplici funzioni che consentono di effettuare chiamate o inviare messaggi, questi dispositivi consentono di estrarre informazioni sull'utente grazie ad un insieme di sensori, sia fisici che virtuali. La presente tesi descrive la progettazione e realizzazione di un framework per la Human Activity Recognition (HAR), utilizzando i dati raccolti da due sensori di movimento: accelerometro e giroscopio. A partire da questi dati grezzi vengono calcolati valori di tipo statistico al fine di modellare quattro differenti attività *Fermo*, *Cammino*, *Corro* e *Veicolo*. Successivamente la classificazione real-time viene effettuata tramite il K-NN, una tecnica di machine learning tanto diffusa in letteratura quanto affidabile. Il sistema, inoltre, adotta un paradigma di Participatory Sensing in cui ogni utente contribuisce al corretto funzionamento del framework fornendo feedback sull'attività riconosciuta dallo smartphone e inviando i dati sensoriali raccolti al fine di aggiornare i modelli interni del sistema. Per verificare l'efficienza del sistema proposto, sono stati condotti degli esperimenti per confrontare quanto si è realizzato con altre tecniche allo stato dell'arte.

Indice

Capitolo 1

Introduzione

Negli ultimi anni la percentuale di popolazione che possiede uno smartphone è aumentata vertiginosamente. Questi dispositivi forniscono funzionalità che vanno oltre i semplici scopi di chiamare e mandare messaggi. Infatti, grazie a sensori installati a bordo, possono essere considerati delle vere e proprie *piattaforme di sensing* in grado di estrarre informazioni di particolare rilevanza riguardo un certo contesto. In letteratura tali informazioni possono essere utilizzate per differenti applicazioni. Si passa da scenari in cui i dati sensoriali vengono elaborati al fine di tenere sotto controllo persone che soffrono di patologie mediche, quali diabete, melanomi, handicap, fino ad arrivare a scenari di smart cities e smart home in cui gli smartphone vengono adoperati per gestire situazioni di emergenza, per facilitare la viabilità e così via. Tutti questi scenari sono possibili grazie ad alcuni sensori come quelli di posizione, quale il GPS che consente di tener traccia della posizione dell'utente, e sensori di movimento, come accelerometro e giroscopio che in base al tipo di scenario possono avere usi differenti.

Sempre più ricerche si sono concentrate sull'utilizzo degli smartphone applicati al problema della **Human Activity Recognition (HAR)**, dove i dati raccolti da sensori di movimento possono essere analizzati per inferire l'attività svolta da un individuo. Questo scenario risulta essere particolarmente interessante poiché il riconscimento delle attività umane può essere intercalato in altri contesti. Per esempio in uno scenario di gestione del flusso di gente sapere se un individuo stia camminando o correndo può aiutare il sistema nella gestione della folla. Un altro esempio in cui il riconoscimento delle attività può giocare un ruolo cruciale riguarda uno scenario in ambito medico. Per esempio monitorare l'attività fisica degli utenti tramire gli smartphone è utile per la gestione della salute dei pazienti affetti da malattie croniche, ad esempio la malattia di Parkinson [1].

In letteratura si trovano svariate tecniche che si approcciano all'HAR e tra le più interessanti vi è sicuramente il sistema nativamente reso disponibile da Google su dispositivi android, tramite le *Activity Recognition API* [2], mette a disposizione uno strumento con cui è possibile inferire alcune attività svolte dagli individui.

In scenari come quello dell'HAR giocano un ruolo fondamentale gli algoritmi di *machine learning* utilizzati per l'analisi dei dati. Formalmente "l'Apprendimento automatico rappresenta una delle aree fondamentali dell'Intelligenza Artificiale e si occupa della realizzazione di sistemi che si basano su osservazioni o esempi come dati per la sintesi di nuova conoscenza (classificazioni, generalizzazioni, riformulazioni)" [3]. Per poter riconoscere una certa attività, l'approccio maggiormente adoperato è legato al concetto di *classificazione*. Quest'ultimo si basa sulla costruzione di un modello che permetta di classificare dati appartenenti alla stessa categoria. In particolare, in uno scenario come la HAR, viene considerata la classificazione multi-classe, poiché generalmente è necessario riconoscere più di due attività. Infine le tecniche di classificazione possono diversificarsi in base a due tipi di approcci: *supervisionato* e *non supervisionato*. L'approccio maggiormente adoperato in letteratura è il primo poiché la complessità della tecnica non supervisionata risulta essere maggiore e i dati relativi a più classi potrebbero essere accorpati tra loro creando problemi nella distinzione delle classi stesse.

Tra le più importanti funzionalità degli smartphone vi è la possibilità di potersi connettere alla rete tramite Wi-Fi o operatore telefonico. Ciò dà la possibilità di condividere informazioni raccolte localmente nei singoli dispositivi e consente agli utenti di essere considerati essi stessi come veri e propri "sensori" che generano dati utilizzabili per diversi scopi [4]. Questo fenomeno prende il nome di **Participatory Sensing** e, formalmente, può essere considerato come il concetto di una comunità che contribuisce fornendo informazioni sensoriali per ricavare certi contenuti informativi. Partendo dalla sua definizione è possibile integrare tale approccio nei sistemi che si occupano di HAR tramite utilizzo dello smartphone.

In questo lavoro di tesi viene presentata la progettazione e realizzazione di un framework per il riconoscimento in real-time delle attività svolte dagli utenti. L'elemento fondamentale del sistema è rappresentato dal processo di classificazione che avviene tramite l'uso del *k-nearest neighbors* (K-NN), un algoritmo di machine learning molto conosciuto in letteratura, che mostra migliori performance in termini di efficienza e accuratezza di riconoscimento. In particolare i dati grezzi, estratti dai sensori di movimento, vengono elaborati al fine di individuare sequenze di pattern ricorrenti, rappresentate da vettori di feature che descrivono le attività svolte dagli individui. Successivamente, sulla base di un approccio supervisionato,

tali vettori di feature verranno passati in input al K-NN per scopi di classificazione. Inoltre nel documento si è considerato l'utilizzo dell'applicazione in uno scenario di *Participatory Sensing* in cui ogni utente contribuisce fornendo i dati raccolti attraverso un'architettura Client/Server e feedback riguardo la correttezza dell'attività riconosciuta.

Il resto del documento è organizzato nel modo seguente: lo stato dell'arte viene presentato nel Capitolo 2. Esso descrive quali approcci sono stati utilizzati per riconoscere le attività svolte dall'utente. Si partirà dai metodi vision-based evidenziando i loro svantaggi, passando alle tecniche wearable sensor-based considerate scomode per l'utente, per arrivare agli approcci smartphone-based che rappresentano l'attuale stato dell'arte per il problema dell'HAR. Inoltre si descriveranno alcuni lavori riguardanti il participatory sensing.

Nel Capitolo 3 si presenta e definisce il sistema proposto per l'Activity Recognition mostrando le sue componenti principali. In particolare verranno prese in esame le varie fasi di progettazione inerenti sia il processo di estrazione delle feature, sia la scelta dell'algoritmo di classificazione, sia l'implementazione dell'architettura Client/Server.

Al fine di confrontare il sistema con alcune tecniche allo stato dell'arte e le sue prestazioni, nel Capitolo 4 sono tracciati diversi esperimenti e ne sono stati analizzati i risultati ottenuti. In particolare, confrontando i vari sistemi, si vedrà che il framework proposto nel seguente lavoro di tesi risulta essere più efficiente e accurato rispetto agli altri.

Si giunge così alle conclusioni, trattate nel Capitolo 5. Quindi verranno proposti dei possibili lavori futuri che consentiranno di apportare migliorie al sistema.

Capitolo 2

Stato dell'arte

Grazie alla potenza di calcolo e alla molteplicità di sensori installati a bordo di ogni dispositivo, gli smartphone sono considerati dai ricercatori come degli strumenti a partire dai quali è possibile creare sistemi di ultima generazione che facilitano la vita dell'uomo. Negli ultimi anni la comunità di ricerca ha sviluppato framework di vario tipo in base al tipo di scenario a cui ci si vuole rivolgere.

Per esempio in [5] viene descritto un sistema per uno scenario di *smart cities* che, a partire dall'analisi dei dati raccolti dai sensori di posizione, scopre automaticamente quali siano i luoghi di interesse di ogni individuo. A tal fine si considera un clustering a due livelli in cui, rispettivamente, vengono raggruppate le posizioni, adoperando una tecnica di time-based clustering, e i punti in cui l'utente trascorre maggiormente il suo tempo, utilizzando un grid-based clustering. Un altro sistema viene descritto in [6] dove si considera il problema del monitoraggio del traffico e delle condizioni di una strada in una città. Tale scenario risulta essere particolarmente interessante poichè permette di effettuare delle analisi in real-time che possono essere utilizzate al fine di garantire la sicurezza del conducente e la manutenzione delle strade. In particolare il sistema, tramite l'uso dell'accelerometro, è in grado di analizzare e riconoscere alcuni pattern di guida e condizioni della strada, quali buche, frenate, urti. Altro scenario interessante è quello che prevede l'uso delle informazioni sensoriali in ambito *medico* al fine di migliorare la qualità di vita delle persone assistendole durante il giorno e tenere sotto controllo il loro stato di salute. Un esempio è descritto in [7] dove i sensori installati a bordo dello smartphone vengono utilizzati per inferire lo stato emotivo dei pazienti facendo attenzione al livello di stress degli stessi. In particolare, analizzando il modo in cui le persone interagiscono con il loro smartphone, si vuole capire la condizione psicologica dell'utente. In [8] viene presentato un framework per il monitoraggio

comportamentale dei pazienti da remoto. In particolare, al fine di capire il livello di stress e depressione, per ognuno di essi si monitorano i movimenti (utilizzando GPS e WiFi), attività cinestetiche (utilizzando accelerometri multiassiali), durata del sonno (modellata utilizzando dati di utilizzo del dispositivo, caratteristiche del suono ambientale e livelli di luce ambientale), e il tempo trascorso al telefono. Altri lavori in letteratura utilizzano gli smartphone per la *gestione delle folle e delle situazioni di emergenza*. In [9] si propone un sistema di navigazione, tramite smartphone, che consente a utenti che partecipano a eventi (come concerti e spettacoli) di evitare aree affollate o strade strette. Per il controllo della folla per prima cosa vengono acquisite delle immagini da un sistema esterno per stimare una mappa di densità di folla e, successivamente, analizzando i dati sensoriali ricavati dallo smartphone, vengono estratte informazioni riguardo la posizione e la velocità di movimento dell'utente stesso. Un altro lavoro è descritto in [10] dove viene presentata una tecnica per dedurre la densità della folla analizzando le posizioni condivise dagli utenti, supponendo che la velocità a piedi dei utenti sia dipendente dalla densità della folla stessa. La peculiarità di questo lavoro, a differenza di quelli proposti in letteratura, risiede nella possibilità di dedurre la densità di folla avendo un numero ridotto di utenti che condividono i dati.

In letteratura uno scenario particolarmente analizzato da studiosi e ricercatori consiste nel fornire informazioni accurate riguardo le attività svolte dagli utenti. Questo è dovuto, principalmente, al fatto che è possibile sviluppare applicazioni che possono essere adattate a contesti come quelli sopra descritti (medicina, gestione situazioni di emergenza ecc.). Molti dei metodi presentati in letteratura riguardanti la Human Activity Recognition possono essere raggruppati in due macro-categorie: *Vision-based* e *Sensor-based*.

Il primo è in grado di inferire le attività svolte dagli utenti tramite l'analisi di sequenze video catturate da diversi dispositivi [11]. Inizialmente tali tecniche si basavano sull'estrazione delle silhouette da immagini RGB. Purtroppo l'alto costo computazione di elaborazione e la necessità di ripulire le immagini da informazioni rumorose rendono questo tipo di approccio non adatto per applicazioni in real-time. Studi più recenti si sono concentrati su dispositivi, come il *Kinect* di Microsoft, poichè è stato dimostrato che i dati RGB-D migliorano il processo di riconoscimento consentendo un'analisi in real-time della scena osservata. Sulla base di questo approccio, in [12] viene presentato un metodo per il riconoscimento delle attività svolte dagli utenti utilizzando le informazioni sensoriali raccolte da una camera RGB-D (la Kinect di Microsoft). L'approccio proposto modella un'attività come evoluzioni spazio-temporali di posture conosciute e fa uso di tre differenti tecniche di machine learning, quali K-Means clustering, Support Vector Machine e Hidden Markov Models, al fine

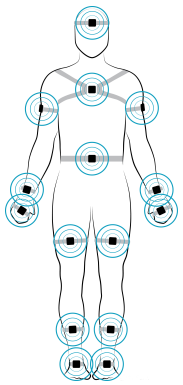


Figura 2.1: Sensori wearable dislocati in varie parti del corpo.

di combinarle tra loro per riconoscere le posture degli individui. Un altro lavoro che fa uso del Kinect, applicato ad uno scenario di Ambient Intelligence, viene presentato in [13]. L'approccio proposto sfrutta la posizione di alcune parti del corpo umano stimate a partire dalle informazioni di profondità provenienti dal Kinect. In tale sistema vengono rilevati schemi significativi di giunzioni (cioè posture) applicando in successione una tecnica di clustering, una SVM multi-classe e una HMM per modellare sequenze di posture note. Infine in [14] si descrive un sistema per l'activity recognition basato dati scheletrici estratti da una camera di profondità. A partire da alcune posture di base, il sistema sfrutta metodi di machine learning per classificare le varie azioni. Di particolare interesse è l'utilizzo dell'algoritmo X-means che permette di trovare il numero ottimale di cluster in modo dinamico. Le tecniche vision-based hanno due svantaggi: sono invasive, poichè registrano tutto ciò che fa l'utente durante il giorno, e il loro utilizzo è limitato solamente agli ambienti indoor (come casa, ufficio ecc). Tuttavia, se la necessità è quella di utilizzare questo tipo di approccio, le tecniche basate su dati RGB-D sono da preferire rispetto a quelle che si basano sull'analisi dei dati RGB poichè sono più adatte per un riconoscimento in real-time.

La seconda macro-categoria si basa sul posizionare vari tipi di sensori in diverse parti del corpo umano. Questo approccio supera i limiti dei sistemi *Vision-based* poiché sono meno intrusivi e richiedono uno sforzo computazionale molto minore. I primi esperimenti sono stati effettuati usando solamente un accelerometro per catturare i valori dell'accelerazione sui tre assi [15]. Purtroppo un singolo sensore non è sufficiente per descrivere attività molto complesse e, di conseguenza, molti studi si sono concentrati su tecniche che facessero uso di sensori multipli. Per esempio, in [16] il sistema descritto acquisisce dati da cinque accelerometri biassiali, indossati simultaneamente su differenti parti del corpo, al fine di riconoscere attività semplici e complesse. Un altro lavoro che fa uso di diversi accelerometri per riconoscere la attività di *walking*, *running*, *sitting* e *standing* è descritto in [17]. Infine altri studi

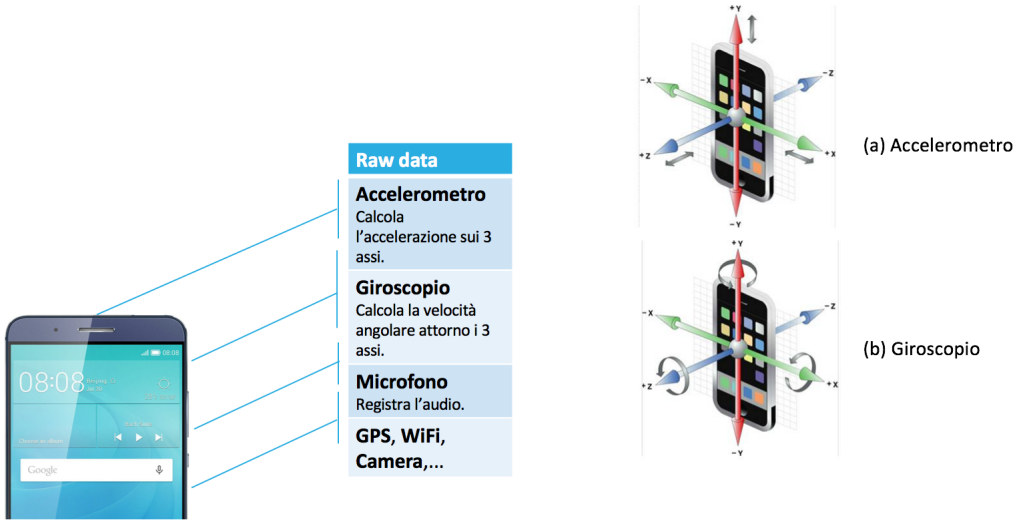


Figura 2.2: Sensori a bordo degli smartphone.

hanno cercato di migliorare i sistemi di riconoscimento adoperando combinazioni di sensori eterogenei, come accelerometri, giroscopi, GPS e così via. Un esempio di tale approccio è descritto in [18], dove gli autori propongono un nuovo modello, basato su una rete Bayesiana dinamica, in grado di riconoscere le attività svolte e il luogo in cui l'utente si trova facendo uso di un accelerometro triassiale, due microfoni, un barometro, GPS e dei fototransistori. In [19] viene utilizzata una scheda sensoriale multi-modal composta da accelerometro triassiale, barometro, fototransitori, compasso e così via per cercare di riconoscere otto differenti attività tra quelle più comuni svolte quotidianamente.

Per i metodi appena descritti, l'utente deve fisicamente indossare i sensori tramite fasce elastiche dislocate in varie parti del corpo come mostra la Fig.2.1. Questo risulta essere particolarmente scomodo poichè ogni individuo dovrebbe svolgere attività all'aperto indossando questi dispositivi. Per ovviare a questo problema, negli ultimi anni, sempre più lavori si sono concentrati sul problema dell'HAR con applicazioni smartphone-based. Questa scelta è dovuta principalmente alla grande diffusione degli smartphone e alle loro intrinseche caratteristiche, quali sensori embedded (vedere Fig.2.2), facile portabilità del dispositivo, connettività di rete e una potenza di calcolo sempre più alta.

In [20] gli autori presentano un'approccio che combina tecniche di machine learning e symbolic reasoning per migliorare la qualità di vita di pazienti diabetici. L'intero sistema è basato sull'analisi dei dati sensoriali provenienti dallo smartphone al fine di tenere traccia della fatica e della depressione durante le attività svolte quotidianamente. Un altro sistema è quello descritto in [21], dove reti neurali, implementate sullo smartphone, permettono di riconoscere cinque attività: *static*, *walking*, *running*, *upstairs* e *downstairs*. In [22] viene

Attività	Descrizione
In vehicle	L'utente si trova in un veicolo.
On bicycle	L'utente è in bicicletta.
On foot	L'utente sta camminando o correndo.
Running	L'utente sta correndo.
Still	L'utente è fermo.
Tilting	L'utente si sta alzando in piedi.
Unknown	Impossibile riconoscere l'attività.
Walking	L'utente sta camminando.

Tabella 2.1: Attività riconosciute dalle Google Activity Recognition API.

implementato un sistema di fall detection tramite sensori dello smartphone. In particolare la caduta viene riconosciuta se viene superato un certo valore di threshold. Infine è importante sottolineare che esistono tecniche che utilizzano un approccio non supervisionato come quella descritta in [23]. Purtroppo in questi casi il processo di riconoscimento è dipendente dal numero di cluster scelti durante la fase di design. Di conseguenza attività differenti potrebbero essere erroneamente combinate non supervisionate in una e viceversa.

Uno dei framework più interessanti allo stato dell'arte è quello presentato da Google, il quale nel 2013 ha lanciato le Activity Recognition API [2]. Tale strumento fornisce agli sviluppatori una libreria con cui è possibile riconoscere quale attività l'utente stia attualmente svolgendo. Il tutto senza preoccuparsi delle complesse tecniche necessarie per ottenere i dati grezzi dai sensori e per estrarre informazioni tali che permettano di identificare l'attività stessa. Purtroppo una delle più grandi limitazioni di queste API è rappresentata dall'impossibilità di sapere quali e come siano implementati i meccanismi utilizzati all'interno della libreria. Questo da un lato permette di sviluppare in modo semplice e veloce applicazioni Android con differenti funzionalità, dall'altro non consente a ricercatori e studiosi di poter analizzare e modificare il codice per apportarne migliorie. La libreria di Google è in grado di riconoscere le attività mostrate in Tabella 2.1.

Infine tra i lavori più interessanti in letteratura vi è un piattaforma chiamata *Mobile Phone Sensing* (MSF) [24]. Essa è un framework general-purpose che permette la creazione di applicazioni Android in grado di poter leggere, in modo semplice ed efficiente, diverse informazioni ricavate dai sensori montati a bordo di ogni dispositivo. I principali vantaggi di questo sistema sono dovuti alla facilità con cui gli sviluppatori possono includere questa libreria nella propria app e al fatto che, a differenza di Google, il codice è open-source, il che consente di gestire, modificare e migliorare l'intero framework. A partire dal lavoro appena descritto è stata sviluppato un sistema di ultima generazione, denominato *Mobile Sensing*

Technology (MoST) e presentato in [25], che fornisce un meccanismo, ottimizzato per gli smartphone, per il riconoscimento delle attività svolte dagli utenti e per il geofencing. Il processo di activity recognition viene effettuato elaborando i dati grezzi raccolti dai sensori montati a bordo del dispositivo, e consente di discriminare tra tre tipi di attività: *Fermo*, *Corro* e *Cammino*. Il geofancing ha lo scopo di trovare e delimitare aree geografiche in cui vengono svolte attività, eventi o altro. L'architettura di MoST è stata progettata in modo da garantire diverse proprietà: modularità, ovvero la possibilità di poter personalizzare quali sensori utilizzare in base allo scopo finale dell'applicazione; non intrusività grazie alla gestione dinamica degli eventi; leggerezza poichè tale framework viene implementato su dispositivi con risorse limitate.

Inoltre, al fine di estendere il lavoro proposto in [25] ad uno scenario su larga scala, in [26] viene descritto un progetto il cui scopo è quello di analizzare e studiare il potenziale rappresentato dalla collaborazione di utenti, che sfruttando gli smartphone come mezzo di interconnessione, condividono con un sistema più grande tutte le informazioni riguardanti certe attività. In particolare, grazie alla libreria MoST, è stata sviluppata un'applicazione Android che permette agli utenti di eseguire alcune azioni, come misurare la qualità della ricezione del telefono o l'inquinamento acustico, e inviare i dati ad un server centrale, che elabora, aggrega, e analizza le informazioni ricevute al fine di inferire un certo contenuto informativo non ottenibile da un singolo utente. L'idea appena presentata, in cui la comunità contribuisce condividendo informazioni di un certo tipo, apre lo scenario a nuove applicazioni che ben si adattano a ciò che in letteratura viene chiamato Participatory Sensing. Molti studi si sono concentrati per creare sistemi che permettessero di gestire le problematiche relative all'invio collettivo di dati (o crowdsensing). In [27] si descrive un sistema, chiamato Vita, il quale, attraverso l'uso di dispositivi mobili, fornisce un approccio sistematico per supportare gli sviluppatori per la creazione di applicazioni adatte al mobile crowdsensing. In particolare ad ogni utente, in base al proprio profilo, viene associato un certo task tramite ciò che viene indicato come «social vector», ovvero una rappresentazione sintetica delle risorse e conoscenze degli utenti. In [28], invece, viene adottato un modello sensing-based per uno scenario di participatory sensing nell'ambito della gestione delle situazioni di emergenza. Nel dettaglio si effettuano delle analisi semantiche su dati di microblog e le parole più frequenti, relative a diversi luoghi e/o eventi, vengono estratte per poi essere classificate. Quindi, generalmente, le applicazioni in cui si richiede un paradigma di participatory sensing non si limitano ad un particolare scenario ma possono avere usi differenti in base al tipo di applicazione che si vuole creare.

Capitolo 3

Il sistema di activity recognition

In questo capitolo verrà individuato un modello di sistema per il riconoscimento delle attività svolte dagli utenti tramite l'analisi dei dati sensoriali, i quali verranno successivamente condivisi attraverso l'architettura Client/Server. Nel dettaglio si spiega come, a partire dai dati grezzi ottenuti dall'accelerometro e dal giroscopio, avvengono i processi di estrazione delle feature e di classificazione a bordo del dispositivo. Inoltre verrà fornita una breve descrizione sui meccanismi di sicurezza informatica, necessari per la protezione dei dati scambiati tra client e server, fino ad arrivare alle tecnologie implementate dal sistema per raggiungere i suddetti scopi. Tale modello, dopo essere stato formalizzato, verrà utilizzato come linea guida per l'implementazione del framework finale.

Si ritiene necessario, per una maggiore chiarezza, individuare quali siano le componenti essenziali comuni alla maggior parte dei sistemi proposti in letteratura [29]. La Figura 3.1 riassume tali componenti: lo smartphone, mentre un *partecipante* (utente nel contesto del Participatory Sensing) effettua una delle quattro attività tra *Fermo*, *Cammino*, *Corro* e *Veicolo*, raccoglie dati grezzi dai sensori per la successiva fase di elaborazione. Dopodiché le informazioni elaborate vengono inviate al server, che ne analizzerà il contenuto.

Figura 3.1: Componenti base per il Participatory Sensing

3.1 Architettura

L'architettura proposta ha lo scopo di inferire automaticamente l'attività svolta dall'utente analizzando i dati raccolti dallo smartphone. Come mostra la Fig.3.2, le entità che entrano in gioco nel sistema sono tre: *partecipante*, *smartphone* e *server*.

Nell'ambito del Participatory Sensing il partecipante è l'utente che per l'appunto *partecipa* all'esperimento proposto. L'intero sistema viene avviato dall'utente (**passo 1**) nel momento in cui si accinge a svolgere una delle quattro attività.

Successivamente il controllo passa allo smartphone che si occuperà principalmente di svolgere tre operazioni. La prima è responsabile della *data collection*, che si occupa di raccogliere dati grezzi dai sensori dello smartphone mentre un'attività viene svolta. La collezione dei dati è resa possibile grazie alla libreria MoST semplicemente richiedendo che ogni utente effettui una delle quattro attività mentre lo smartphone è posto nella tasca dei pantaloni. Inoltre, a differenza di [23], il sistema proposto non prende in considerazione l'accelerazione di gravità consentendo all'utente di posare lo smartphone senza preoccuparsi del suo orientamento. Successivamente tali informazioni vengono inviate al modulo di *feature detection*, dove viene estratto un insieme di punti n-dimensionali al fine di riconoscere una particolare attività. Queste vengono, poi, classificate usando un algoritmo di machine learning che, come si vedrà nelle sezioni successive, sarà il K-NN. Dopo che il processo di riconoscimento avrà restituito una delle quattro classi (**passo 2**), l'utente avrà il compito di validare la correttezza dell'output inserendo un feedback positivo o negativo sull'attività riconosciuta (**passo 3**).

Quindi tutte le informazioni raccolte dallo smartphone, che comprendono sia le feature estratte sia il feedback rilasciato, vengono inviate al server (**passo 4**), il quale ha il compito di svolgere delle elaborazioni al fine di aggiornare i modelli interni del sistema e valutare l'efficacia di riconoscimento dell'algoritmo utilizzato. Oltre a ciò, lato server, vi è l'*amministratore* che può accedere ai dati del server e leggere informazioni rilevanti, quali il pattern delle attività descritte dai dati sensoriali grezzi, i modelli aggiornati, le confusion matrix ecc. (**passo 5**). Inoltre ogni volta che i modelli vengono aggiornati il server si preoccuperà di inviarli al client in modo da poter essere adoperati nelle successive elaborazioni effettuate dallo smartphone (**passo 6**).

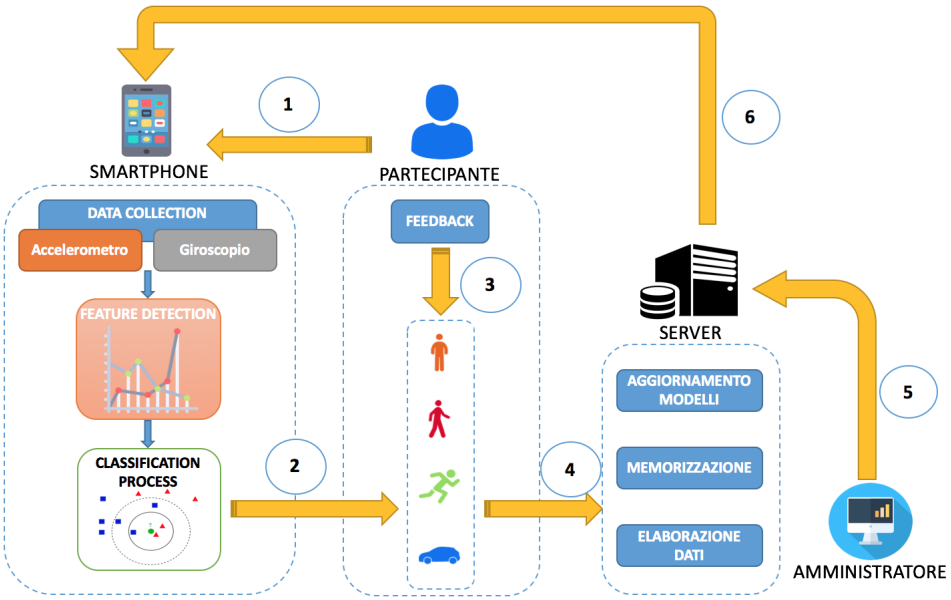


Figura 3.2: Architettura del sistema.

3.2 Analisi on board

In questa sezione si forniranno le modalità con cui vengono estratti i dati sensoriali che verranno utilizzati nella seguente fase di classificazione che si descriverà nel paragrafo successivo. L'obiettivo è quello di ottenere un modello matematico che possa descrivere il pattern di una certa attività svolta da un utente [30, 31, 32]. Al fine di costruire un modello completo basato su un approccio supervisionato, è necessario definire tre elementi essenziali: le *classi*, l'*insieme di addestramento* e la *funzione obiettivo*.

Dal momento che si ci muove nello scenario della Human Activity Recognition, le classi sono rappresentate dalle attività svolte dagli utenti. Per rendere più chiaro il concetto di attività si faccia riferimento alla seguente:

Definizione 1. Un'attività a è un comportamento effettuato da un utente per un certo periodo di tempo $[t_s, t_e]$ e denotata come $a = (id, t_s, t_e)$.

dove:

- id: identificativo dell'attività svolta;
- t_s: istante in cui viene iniziata l'attività;
- t_e: istante in cui viene terminata l'attività.

Nell'esperimento presentato le classi sono le quattro attività di base:

- a_1: *Fermo*;

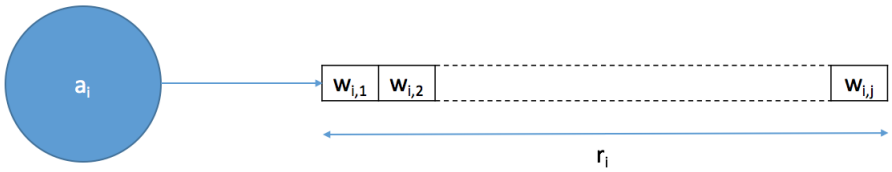


Figura 3.3: Ogni attività viene raccolta per un tempo r_i e suddiviso in $w_{i,j}$ finestre.

- a_2: *Camminare*;
- a_3: *Correre*;
- a_4: *Veicolo*.

La scelta è giustificata dal fatto che queste sono quelle effettuate più frequentemente dagli utenti durante la vita quotidiana.

*** OMISSIS ***

Per assicurare un riconoscimento delle attività in real-time il sistema deve essere in grado di elaborare i dati grezzi in input dentro finestre temporali, al fine di estarre le feature che verrano utilizzate nella successiva fase di classificazione. Scegliere la giusta lunghezza per le finestre temporali è un fattore da non sottovalutare: finestre brevi potrebbero da un lato migliorare le performance del sistema in termini di CPU load e tempo di esecuzione ma, dall'altro, non raccogliere sufficienti informazioni per discriminare l'attività; finestre lunghe potrebbero contenere informazioni riguardo più attività e di conseguenza alterare le performance del sistema in termini di riconoscimento delle stesse.

Per capire come avviene l'estrazione delle feature, si definiscano i seguenti parametri:

- a_i: attività i-esima con i $\in \{1,\ldots,4\}$;
- r_i: minuti di registrazione dell'attività i-esima nella fase di training;
- $w_{i,j}$: finestra j-esima di campionamento costante relativa all'attività i-esima con $i \in \{1,\ldots,4\}$ e $j \in \{1,\ldots,k\}$, dove k è il numero massimo di finestre di campionamento contenute in r_i.

Supponendo che il periodo di tempo in cui viene registrata l'attività i-esima sia proprio r_i, allora si ottiene la configurazione mostrata in Fig.3.3. Ogni finestra $w_{i,j}$ contiene una sequenza di valori che provengono dall'accelerometro e dal giroscopio. In particolare tale sequenza è

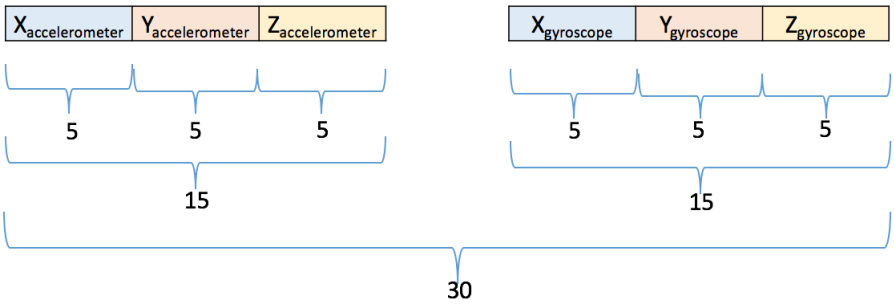


Figura 3.4: Struttura di un vettore di feature.

formata dalla successione di n campioni per ognuno degli assi cartesiani ottenendo, per una certa $w_{i,j}$, una sequenza composta da $3 * n$ elementi per ogni sensore.

A partire dalle sequenze descritte, vengono estratte delle feature di tipo statistico calcolando i seguenti valori per ogni asse:

- *massimo*;
- *minimo*;
- *media*;
- *deviazione standard*;
- *scarto quadratico medio*.

Ne consegue che un singolo vettore di feature $f_{i,j}$ sarà composto da $P = 30$ elementi di tipo statistico: 15 per l'accelerometro e 15 per il giroscopio (si veda Fig.3.4).

*** OMISSIS ***

Come precedentemente anticipato, altra componente fondamentale è il *training set* ed è necessaria per poter riconoscere correttamente un'attività nella successiva fase di test. L'insieme di addestramento è costruito a partire da un set di feature calcolati per ognuna delle singole attività. Formalmente si supponga che per ogni classe a_i l'insieme delle feature sia il seguente:

$$TS(a_i) = \begin{vmatrix} f_{1,1} & f_{1,2} & \dots & f_{1,n} \\ f_{2,1} & f_{2,2} & \dots & f_{2,n} \\ \vdots & \vdots & \dots & \vdots \\ f_{j,1} & f_{j,2} & \dots & f_{j,n} \end{vmatrix}$$

dove j indica il numero di finestre per una data r_i. Il training set risultante sarà:

$$S = \bigcup_{i=1,\ldots,4} TS(a_i)$$

L'ultimo elemento necessario affinché il problema sia ben posto è la funzione di stima, la quale viene utilizzata come metrica per poter classificare un nuovo elemento. In particolare nel presente documento è stata considerata la *distanza euclidea*. A partire da ciò, se i vettori di feature vengono considerati come punti n-dimensionali dello spazio, allora è possibile utilizzare come funzione di stima la distanza euclidea tra due unità i ed h nel modo seguente:

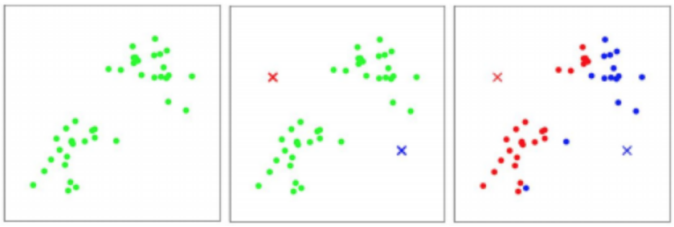
$$\sqrt{\sum_j (x_{ij} - x_{hj})^2}$$

3.3 Sistema di classificazione

La componente principale del sistema presentato in Fig.3.2 è il processo di classificazione. In questo lavoro sono stati presi in esame due algoritmi di machine learning molto conosciuti in letteratura: *K-Means* e *K-NN*. Come sarà chiaro nel Capitolo 4, nel sistema finale il processo di classificazione si basa sul K-NN poichè risulta essere migliore in termini di performance e accuratezza del riconoscimento delle attività. A prescindere dalla scelta presa, al fine di capire come sono state implementate le due tecniche, si fornisce una breve descrizione dei due algoritmi.

Per prima cosa si prenda in considerazione il *K-Means*. Esso è una tecnica di *clustering partizionante* dove ad ogni cluster viene associato un punto rappresentativo, chiamato **centroide**. Una peculiarità di questo algoritmo risiede nella necessità di conoscere a priori il numero di classi, ovvero il valore di K. Anche se in molti casi questa rappresenta una limitazione, qui non lo è poiché il numero di classi è esattamente uguale al numero di attività che si vuole riconoscere. L'idea generale è quella di suddividere un insieme di oggetti in cluster in modo tale da minimizzare una certa funzione obiettivo. In alcune applicazioni, come nel caso della HAR, il K-Means viene utilizzato come tecnica di classificazione supervisionata più che come algoritmo di clustering. A tal proposito possono essere utilizzate due diverse configurazioni:

- **Configurazione 1**: consiste nel dividere i dati secondo la classe di appartenenza e, in un secondo momento, applicare il K-Means separatamente su ogni classe. Così facendo si otterrà un insieme di centroidi per ciascuna classe, per un totale di $c * K$



(a) Passo 1 (b) Passo 2 (c) Passo 3

Figura 3.5: Iterazioni K-Means.

centroidi dove c è il numero di centroidi per ogni classe K. Successivamente un nuovo campione verrà assegnato alla classe che minimizzerà la funzione obiettivo calcolata tra il nuovo punto e i centroidi generati nella fase precedente.

- **Configurazione 2**: consiste nel mettere tutti i dati insieme e nell'applicare il K-Means una volta sola. Non viene in alcun modo garantito che tutti i punti nello stesso gruppo appartengano alla stessa classe. Per poter associare un prototipo a una classe, si devono contare il numero di punti in ogni classe che sono assegnati a quel prototipo. La classificazione di un nuovo punto avviene allo stesso modo in cui è stata descritta nella *Configurazione 1*.

Nella sua implementazione più generale, l'algoritmo segue le fasi mostrate in Fig.3.5. Inizialmente prenderà in input un insieme di punti che appartengono al dataset, come mostrato in 3.5(a). Successivamente verranno generati, in maniera random, un numero di centroidi pari alle classi da analizzare. Infine verranno fatte un numero di iterazioni tale che l'algoritmo soddisferà la funzione obiettivo presa in esame. Seguendo la logica esposta fino ad ora lo pseudo-codice del K-Means viene presentato di seguito.

Algorithm 1: KMeans

Input : $E = \{e_1, e_2, \ldots, e_n\}$
k (number of clusters)
$MaxIters$ (limit of iteration)
Output: $C = \{c_1, c_2, \ldots, c_k\}$ (set of cluster centroids)
$L = \{l(e)|e = 1, 2, \ldots, n\}$ (set of cluster labels of E)

```
foreach c_i ∈ C do
    c_i ← e_j ∈ E (e.g. random selection)
end
foreach e_i ∈ E do
    l(e_i) ← argminDistance(e_i, c_j) j ∈ 1, ..., k
end
changed ← false;
iter ← 0;
repeat
    foreach c_i ∈ C do
        UpdateCluster(c_i)
    end
    foreach e_i ∈ E do
        minDist ← argminDistance(e_i, c_j) j ∈ 1, ..., k
    end
    if minDist ≠ l(e_i) then
        l(e_i) ← minDist;
        changed ← true;
    end
    iter++;
until changed = true and iter < MaxIters;
```

Entrambe le configurazioni possono risultare efficaci ai fini del riconoscimento delle attività. Per poter confrontare la migliore implementazione del K-Means con quella del K-NN, nel paragrafo 4.2 verranno implementati entrambi gli schemi al fine di scegliere quello che ha prodotto risultati migliori in termini di complessità computazionale e accuratezza.

Da questo momento in poi si consideri il *K-NN*. È una tecnica utilizzata nel riconoscimento di pattern per la classificazione di oggetti e si basa sulle caratteristiche degli oggetti vicini a quello considerato. Con questo algoritmo un nuovo elemento è classificato in base alla maggioranza dei voti dei suoi K vicini, dove K è un intero positivo tipicamente non molto grande. Uno degli inconvenienti di questo algoritmo è causato dalla predominanza delle classi con più oggetti. Per ovviare il problema si è soliti pesare i contributi dei vicini in base alla distanza dall'oggetto considerato. Per una buona implementazione dell'algoritmo un fattore cruciale è la scelta di K. Generalmente l'aumento di K riduce il rumore che compromette la classificazione, sebbene il criterio di scelta per la classe diventi più labile. Per capire meglio il funzionamento di questo algoritmo basti guardare la Fig.3.6.

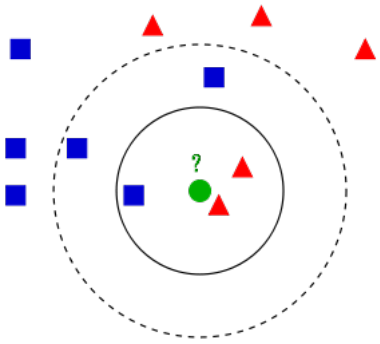


Figura 3.6: Classificazione K-NN.

Supponendo che il pallino verde sia il punto di osservazione e che gli oggetti blu e rossi siano le classi da analizzare:

- se $K = 3$, allora l'algoritmo avrebbe considerato i tre oggetti più vicini e assegnato il punto verde alla classe rossa.
- se $K = 5$, allora l'algoritmo avrebbe considerato i cinque oggetti più vicini e assegnato il punto verde alla classe blu.

Per la scelta di K possono essere adottate diverse strade. La *Cross-Validation* è una tecnica molto usata che serve a ottenere una stima dei parametri del modello che non sono noti. L'idea generale di questo metodo consiste nel dividere l'insieme dei campioni in sottoinsiemi, indicati con v e chiamati **fold**. Fissato K, si applica il K-NN per effettuare previsioni sul segmento v e per valutare l'errore. Questo processo viene successivamente applicato a tutte le possibili v. Gli errori calcolati nei sottoinsiemi, alla fine del processo, vengono mediati per ottenere una misura della stabilità del modello. Le fasi sopra descritte vengono poi ripetute per vari K ed il valore con l'errore più basso viene selezionato come ottimale, nel senso di *convalida incrociata*. Questo tipo di misura dà informazioni sulla *predictive accuracy*. Per una Cross-Validation il valore di v viene impostato a dieci. Questa scelta viene fatta poiché esperimenti approfonditi hanno dimostrato che questo valore è il migliore per ottenere una stima accurata [33]. La Fig.3.7 mostra un esempio di procedura Cross-Validation con $v = 3$. Molto spesso, per avere maggiori informazioni sulla robustezza di un determinato algoritmo, questa misura viene accompagnata da un'altra, ovvero la *Resubstitution*. Questa indica quanto buoni siano i risultati facendo dei test sull'insieme di addestramento stesso ed è tipicamente usata come una misura della performance dell'algoritmo. Lo pseudo-codice dell'algoritmo appena descritto è presentato di seguito.

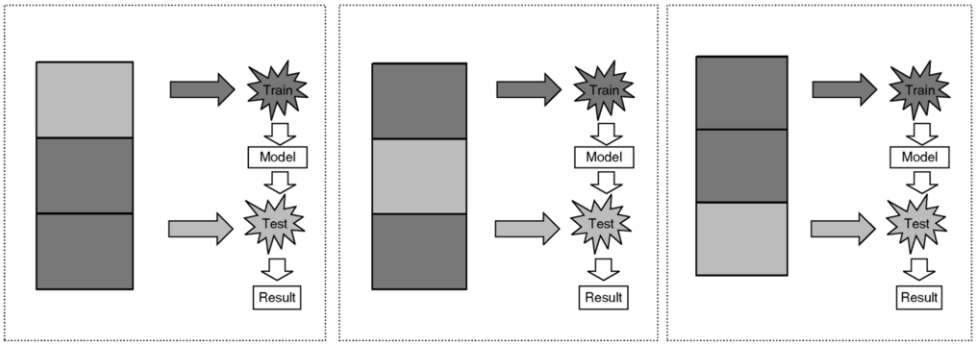


Figura 3.7: Processo di cross vaidation.

Algorithm 2: KNN

Input : D (finite set of point to be classified) T (finite set of point)
c (function $c : T \rightarrow 1,\ldots,m$)
k (natural number)

Output: function $r : D \rightarrow 1,\ldots,m$

```
1  begin
2      foreach x ∈ D do
3          let U ← {}
4          foreach t ∈ T do
5              Add the pair {d(x,t), c(t)} to U
6              Sort the pairs in U using the first components
7              Count the class labels form the first k elements from U
8              Let r(x) be the class with the highest number of occurrence
9          end
10     end
11     Return r;
12 end
```

Come si vedrà nel Capitolo 4 l'intero processo di classificazione si baserà sul K-NN poichè risulta essere più performante sia in termini di carico della memoria, che in termini di accuracy, precision e recall.

Prima di procedere è fondamentale capire come avviene il processo di classificazione a bordo del dispositivo. Formalmente si consideri l'insieme di addestramento costituito da un insieme di vettori di feature $(f_1, f_2, \ldots, f_n)$, come definito nel paragrafo 3.2. L'idea si basa sul prendere un nuovo vettore di feature f di cui non si conosce la classe. Dopodiché, se questo nuovo campione viene proiettato nello spazio dei punti definito dall'insieme di addestramento, idealmente, sulla base del funzionamento del K-NN, verrà circondato dai campioni della stessa classe. Se ciò avviene, l'algoritmo assegnerà ad f la stessa classe dei suoi vicini.

3.4 Meccanismi di sicurezza

Come menzionato precedentemente, il sistema adotta un paradigma per il participatory sensing che consente di sfruttare le informazioni provenienti da una comunità di utenti al fine di migliorare le performance del sistema. Per tale motivo è stata progettata un'architettura

Client/Server che consente ad ogni utente di condividere i dati raccolti tramite smartphone e rilasciare un feedback riguardo al processo di riconoscimento, indicando se la classe in output è corretta oppure no. Nella seguente sezione viene introdotto il problema di come rendere sicura l'architettura Client/Server e, in particolare, verranno forniti gli strumenti per assicurare autenticazione e confidenzilità.

Un approccio largamente utilizzato prevede l'utilizzo del **Transport Layer Security** (TLS). Infatti questo protocollo non solo permette un meccanismo di autenticazione tramite l'utilizzo dei certificati dello standard X.509, ma garantisce anche la segretezza delle comunicazioni tra client e server. In particolare, per quanto concerne il meccanismo di autenticazione, da un lato il server si autenticherà col client inviando il proprio certificato mentre, dall'altro, il client utilizzerà un approccio che prevede l'utilizzo di password.

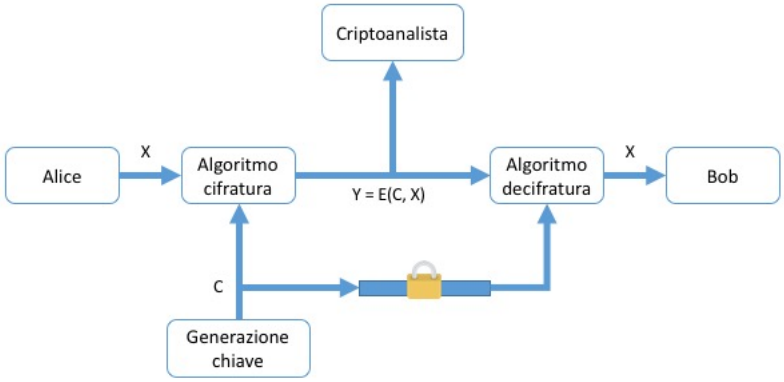
3.4.1 Cenni su crittografia a chiave simmetrica e asimmetrica

Il TLS al suo interno fa uso di algoritmi di crittografia simmetrica, al fine di celare il contenuto di un messaggio, e di algoritmi asimmetrici, per lo scambio delle chiavi tra le due entità comunicanti. Per rendere più completa la trattazione, di seguito verranno analizzate entrambe le famiglie fornendo, inoltre, una breve descrizione degli algoritmi analizzati per il framework.

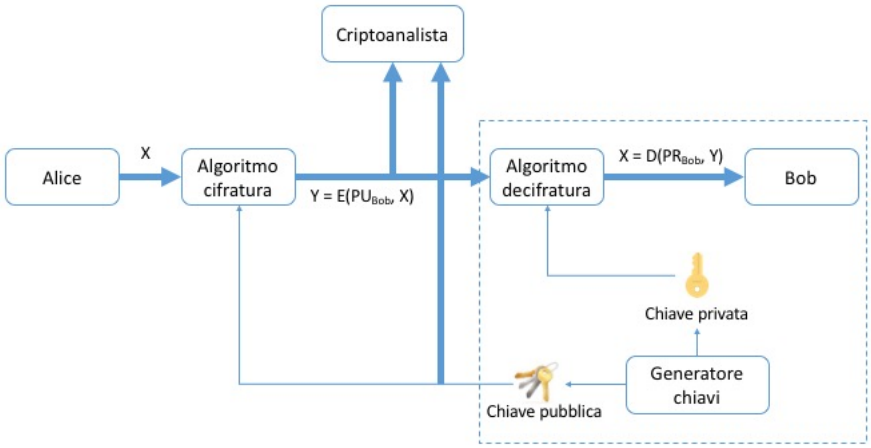
Si inizi mettendo a diretto confronto gli schemi generali delle due famiglie mostrate in Fig.3.8. Dalle immagini si evince che entrambi gli schemi presentano elementi comuni, quali:

- *testo in chiaro*: è il testo originale;
- *algoritmo di cifratura*: effettua sostituzioni e trasformazioni sul testo in chiaro;
- *testo cifrato*: è il testo ritornato in output e dipende dalla chiave segreta e dal testo in chiaro iniziale;
- *algoritmo di decifratura*: effettua delle operazioni inverse rispetto all'algoritmo di crittografia.

La differenza principale risiede nel numero di chiavi e nel come queste vengono utilizzate. Infatti mentre gli schemi simmetrici fanno uso di una singola chiave, indicata come *chiave segreta*, quelli asimmetrici fanno uso di una coppia di chiavi, conosciute in letteratura come *chiave pubblica* e *chiave privata*. Sulla base di ciò, la logica con cui vengono implementati



(a) *A chiave simmetrica.*



(b) *A chiave asimmetrica*

Figura 3.8: Schema algoritmi di crittografia.

gli schemi simmetrici e quelli asimmetrici risulta essere totalmente differente. A linee generali le tecniche simmetriche si basano sulla condivisione, tramite un canale sicuro, della chiave segreta. Questo compito si può adempiere in due modi: il mittente genera la chiave e in qualche modo la condivide segretamente con il destinatario oppure una terza parte genera tale chiave per poi condividerla in modo sicuro con le entità che devono comunicare. Per quanto riguarda le tecniche asimmetriche, invece, ogni entità sulla rete crea una coppia di chiavi, di cui una verrà comunicata alle altre entità (la chiave pubblica), mentre l'altra rimarrà segreta e conosciuta solamente da chi l'ha generata (chiave privata). Seguendo questa logica se un utente A volesse inviare un messaggio a B, allora dovrà cifrarlo con la chiave pubblica di B. Infine è importante evidenziare che, a differenza degli schemi simmetrici, gli algoritmi asimmetrici possono essere utilizzati per molteplici scopi: per cifrare/decifrare messaggi, per lo scambio della/e chiave/i, per le firme digitali.

Nel resto del seguente paragrafo verranno descritte brevemente alcune tra le più famose tecniche di crittografia. In particolare si porrà attenzione all'AES e al 3DES, per quanto concerne gli algoritmi a chiave simmetrica, e al RSA per gli algoritmi asimmetrici. Ovviamente esistono altre tecniche in letteratura ma, ai fini di una maggiore comprensione degli esperimenti introdotti nel Capitolo 4, verranno presi in considerazione solamente quelli appena citati.

DES e 3DES

Prima dell'avvento dell'AES nel 2001, il **Data Encryption Standard** (DES) è stato lo schema più utilizzato, soprattutto nelle applicazioni finanziarie.

Lo schema del DES per un testo in chiaro a 64-bit e una chiave a 56-bit è quello mostrato in Fig.3.9. Come qualsiasi altro schema troviamo due input: il testo da cifrare e la chiave. Il lato sinistro dell'immagine mostra che il processo di trasformazione, dal testo in chiaro in cifrato, si divide in tre fasi differenti:

- *prima fase*: il testo in chiaro a 64-bit subisce una permutazione iniziale (IP) che riordina i bit per un input permutato;
- *seconda fase*: vengono effettuati 16 round consistenti di operazioni di permutazione e sostituzione. L'output del sedicesimo round è formato da 64-bit che sono funzione del testo in chiaro in input e della chiave. Infine le metà di destra e sinistra dell'output vengono sottoposte ad uno swap per creare un preoutput;

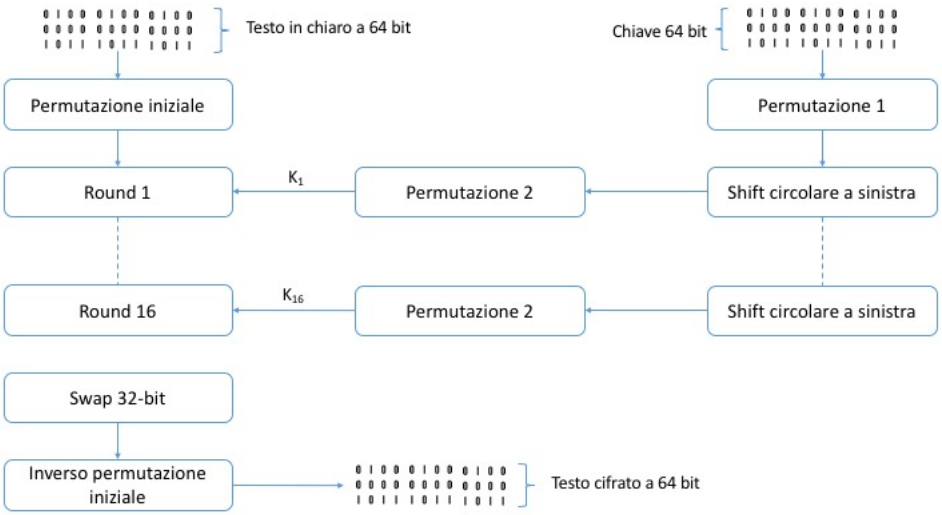


Figura 3.9: Schema funzionamento DES.

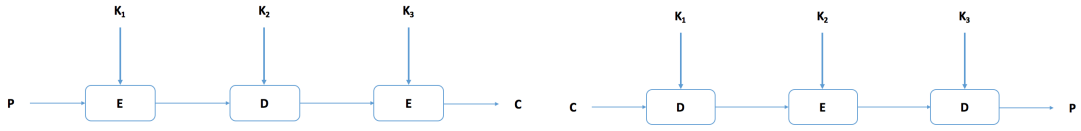
- *terza fase*: al preoutput viene applicata una permutazione inversa a quella iniziale producendo un testo cifrato a 64-bit.

Nella parte destra di Fig.3.9 viene utilizzata la chiave a 56-bit. Per prima cosa viene effettuata una permutazione. Successivamente per ogni round viene prodotta una subkey prodotta dalla combinazione di uno shift circolare a sinistra e una permutazione. Nella fase opposta semplicemente viene eseguito lo stesso procedimento solo che l'uso delle chiavi viene effettuato al contrario. Il DES è considerato insicuro a causa di molteplici vulnerabilità scoperte negli anni successivi alla sua uscita. La sua debolezza principale risiede nella lunghezza della chiave segreta utilizzata per cifrare il testo in chiaro. Nel 1998 alcune compagnie riuscirono in sole 22 ore e 15 minuti a rompere l'algoritmo. Con le potenze di calcolo disponibili ai nostri giorni e, grazie alle architetture parallele, la chiave del DES può essere forzata in poche ore esaminando tutte le possibili combinazioni.

Per migliorare lo schema sono state effettuate diverse proposte tra cui quella maggiormente conosciuta è il *Triple DES* (3DES), ovvero il DES iterato tre volte con due o tre chiavi. Il caso più semplice è il 3DES a due chiavi. Esso viene definito nel modo seguente:

$$C = E(K_1, D(K_2, E(K_1, P)))$$

Per capire meglio tale sequenza di operazioni, si può fare riferimento alla Fig.3.10 che mostra lo schema del 3DES. È importante sottolineare che l'operazione interna non viene eseguita con la finalità di aumentare la robustezza del 3DES, ma viene adoperata per rendere tale schema retro-compatibile con il DES classico. La decifratura del DES viene effettuata in



(a) *Cifratura.* (b) *Decifratura*

Figura 3.10: Logica 3DES.

modo inverso, come mostra la Fig.3.10(b). Anche se il Triple DES risulta essere efficace contro certi tipi di attacchi, come quelli basati sulla crittoanalisi, presenta alcuni svantaggi. Tra tutti sicuramente vi è quello che riguarda l'uso di blocchi a 64-bit, invece che di almeno 128-bit. Un altro punto a sfavore riguarda la complessità di implementazione a livello software, poiché il DES è stato sviluppato in modo da essere eseguito facilmente sui dispositivi hardware. Questo comporta una scarsa efficienza a livello software.

AES

L'**Advanced Encryption Standard** (AES) venne pubblicato dal National Institute of Standards and Technology (NIST) nel 2001. L'AES è un algoritmo di cifratura a blocchi simmetrico, che è subentrato al DES come standard approvato per svariate applicazioni. Nel seguente algoritmo tutte le operazioni vengono effettuate sulla base della aritmetica dei campi finiti $GF(2^8)$. In pratica, un campo è un insieme in cui possiamo effettuare le operazioni di somma, sottrazione, prodotto e divisione senza lasciare l'insieme. Essendo che AES lavora sull'aritmetica dei campi finiti $GF(2^8)$, il polinomio irriducibile utilizzato dall'algoritmo è il seguente:

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

AES consente di utilizzare tre differenti lunghezze per le chiavi: 128, 192 o 256-bit. Sulla base di ciò, i round possono essere 10, 12 o 14. La chiave viene fornita in input e viene espansa in un array di 44 parole a 32-bit. Di queste vengono prelevate 4 parole da 128-bit ciascuna. Ogni round è costituito da quattro differenti operazioni:

- **substitution bytes**: utilizza una S-box per effettuare una sostituzione byte per byte;
- **shift Rows**: una permutazione;
- **mixColumns**: una sostituzione che fa uso dell'aritmetica basata sui campi finiti;
- **addRoundKey**: un semplice XOR del blocco con la porzione corrispondente della chiave espansa.

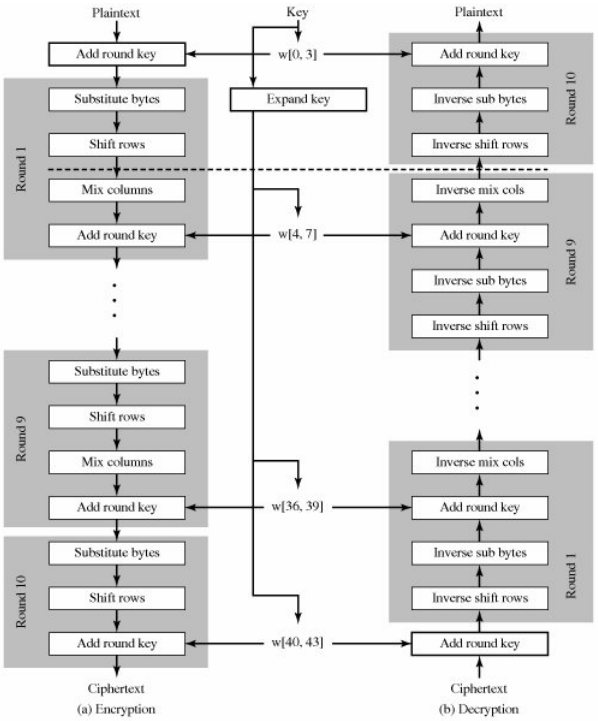


Figura 3.11: Schema AES.

Supponendo di utilizzare l'algoritmo con una chiave da 128 bit, la Fig.3.11 mostra le fasi di cifratura e di decifratura dell'AES, indicando la sequenza delle trasformazioni effettuate in ogni round. La struttura risulta essere molto semplice.

L'AddRoundKey è l'unica delle quattro operazioni che fa uso della chiave. Per questo motivo sia il processo di cifratura sia quello di decifratura iniziano con essa. Le altre tre operazioni, invece, non facendo uso della chiave, sono tutte reversibili e non aggiungono sicurezza aggiuntiva. Infatti vengono utilizzate solamente per introdurre le proprietà di confusione, diffusione e non-linearità.

RSA

Lo schema **Rivest-Shamir-Adleman** (RSA) appartiene alla famiglia degli algoritmi a chiave asimmetrica ed effettua una cifratura a blocchi in cui il testo in chiaro e quello cifrato sono interi compresi nell'intervallo $[0, n-1]$. Nelle applicazioni pratiche, generalmente, n assume il valore di 1024-bit e deve essere noto sia al mittente che al destinatario. Questo significa che ogni blocco deve avere dimensione minore o uguale a $log_2(n)+1$, dove $2^i < n \leq 2^{i+1}$. La cifratura e la decifratura di un testo in chiaro M e del testo cifrato C possono essere espressi nel modo seguente:

$$C = M^e mod(n)$$

$$M = C^d mod(n) = (M^e)^d mod(n) = M^{ed} mod(n)$$

Queste formule spiegano perché RSA è un algoritmo a chiave pubblica. Infatti il mittente conosce il valore di e mentre il destinatario il valore d. Ciò implica la generazione di due chiavi: quella pubblica $PU = \{e,n\}$ e quella privata $PR = \{d,n\}$.

Per poter comprendere come funziona l'RSA abbiamo bisogno di tre elementi fondamentali forniti da Eulero e Fermat, elencati di seguito.

Definizione 2. *Funzione toziente di Eulero* Φ: associa a un intero N il numero degli interi primi con N e minori di N.

$$\phi(N) = |\{M \in N | \gcd(1,M) = 1, 1 \leq M < N\}|$$

Definizione 3. *Teorema di Fermat-Eulero*: dati due interi qualsiasi N e M, con $\gcd(N,M) = 1$:

$$M^N \equiv 1 mod(N)$$

Definizione 4. Se N è primo ed m è coprimo con N, allora:

$$M^{N-1} \equiv 1 mod(N)$$

Definiti questi concetti è possibile descrivere i passaggi dell'algoritmo. Supponiamo che l'utente A voglia trasmettere il messaggio M a B. In primis B deve creare una chiave pubblica ed una privata stando attento a non divulgare d. Per fare ciò B sceglie due numeri primi p e q, molto grandi e li moltiplica nel seguente modo:

$$N = pq$$

Successivamente B sceglie un numero e, coprimo con N, più piccolo della funzione toziente di Eulero. Si ricordi che se N è dato dal prodotto di due numeri primi, allora:

$$\phi(N) = (p-1) \cdot (q-1)$$

Arrivati a questo punto B può calcolare il valore di d tale che:

$$\begin{cases} ed \equiv 1 mod(p-1)(q-1) \\ d \; inverso \; moltiplicativo \; di \; e \end{cases}$$

Infine possono essere create le due chiavi:

$$PU_B = (N, e)$$

$$PR_B = (N, d)$$

Fatto questo B può trasmettere la chiave pubblica ad A.

Per quanto riguarda la decifratura, A calcola il messaggio cifrato e lo manda a B:

$$C = M^e mod N$$

B riceve C e lo decifra eseguendo il seguente calcolo:

$$C^d \equiv M mod N$$

poiché:

$$M^d \equiv M^{ed} \equiv M^{1+\phi(N)} \equiv M^1 M^{\phi(N)} \equiv M mod N$$

La forza di questo algoritmo sta nel fatto che calcolare d, pur conoscendo la chiave, è un problema computazionalmente difficile. Questo non esclude che ci sia un modo semplice di calcolarlo ma al momento non si conosce.

3.4.2 Il sistema proposto basato sul TLS

Dopo aver fornito una breve descrizione dei principali algoritmi di crittografia utilizzati in letteratura è possibile proseguire la trattazione analizzando il protocollo TLS e di come questo sia stato utilizzato nel sistema.

Il TLS permette a due entità, come un client e un server, di comunicare tra loro in modo sicuro. Lo stack di tale protocollo è presentato in Fig.3.12. Nonostante vi siano diversi protocolli che compongono il TLS, due sono quelli principali:

- protocollo *Handshake*;
- protocollo *Record*.

Al fine di comprendere a fondo come avviene lo scambio della chiave e come vengono cifrati i messaggi, risulta utile descrivere il funzionamento di questi protocolli.

Per iniziare si prenda in considerazione il protocollo di Handshake di cui il flusso di messaggi è presentato in Fig.3.13.

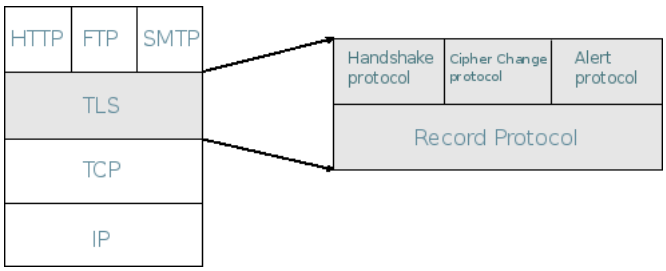
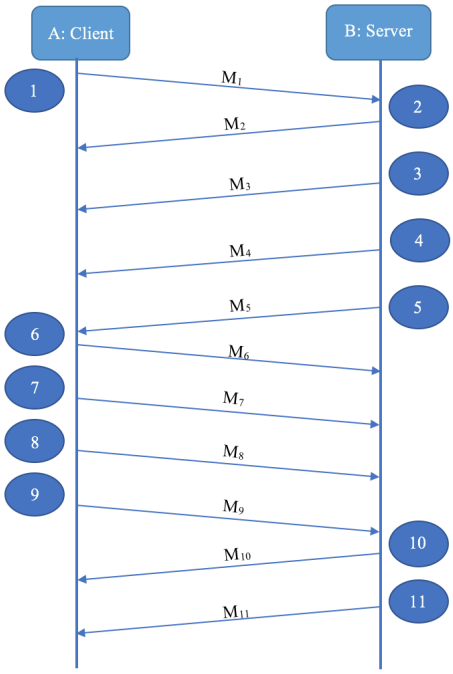


Figura 3.12: Stack TLS.



$M_1)$ $A \rightarrow B : A, clientHello$

$M_2)$ $B \rightarrow A : B, serverHello$

$M_3)$ $B \rightarrow A : B, cert_B$

$M_4)$ $B \rightarrow A : B, certificateRequest$

$M_5)$ $B \rightarrow A : B, serverDone$

$M_6)$ $A \rightarrow B : A, cert_A$

$M_7)$ $A \rightarrow B : A, clientKeyExchange$

$M_8)$ $A \rightarrow B : A, changeChiperSpec$

$M_9)$ $A \rightarrow B : A, finish$

$M_{10})$ $B \rightarrow A : B, changeChiperSpec$

$M_{11})$ $B \rightarrow A : B, finish$

Figura 3.13: Flusso messaggi Handshake.

I passaggi sono i seguenti:

1. il client invia il messaggio M_1 al server denominato *ClientHello*. Questo è composto da cinque campi:

   - *versione*: indica la versione più recente di TLS implementabile dal client;
   - *random*: è una struttura contenente un timestamp da 32-bit e 28-byte creati da un generatore di numeri casuali;
   - *sessionId*: può assumere due valori {0, 1}. Nel primo caso indica che non esiste una sessione esistente, viceversa nel secondo;
   - *chiperSuite*: rappresenta una lista di algoritmi di crittografia che sono supportati dal client (ordinata secondo la preferenza del client stesso);
   - *compression method*: solitamente il suo valore sarà null.

Infine il client invierà il messaggio al server e aspetterà il *ServerDone*.

2. Il server riceve M_1 da parte del client e crea un nuovo messaggio M_2, denominato *ServerHello*, con gli stessi campi del precedente:

   - *versione*: contiene la versione più bassa ti TLS implementabile dal client e la versione più alta supportata dal server;
   - *random*: contiene gli stessi campi del client. I valori di questo campo sono indipendenti da quelli del client;
   - *sessionId*: se l'id del client è diverso da zero, allora questo campo assume lo stesso valore, viceversa viene creato un nuovo id di sessione dal server;
   - *chiperSuite*: prende uno degli algoritmi tra quelli inviati dal client;
   - *compression method*: solitamente il suo valore sarà null.

   Dopo che i vari campi sono stati settati correttamente il server invia il messaggio M_2 al client.

3. Subito dopo il *ServerHello*, il server manda un nuovo messaggio M_3, denominato *ServerCertificate*, contenente proprio certificato.

4. Nel caso in cui l'applicazione abbia necessità di mutua autenticazione tramite certificati, allora è proprio in questa fase che il server richiederà esplicitamente il certificato del client tramite il *certificateRequest*, indicato in figura come M_4. Questo contiene due parametri:

   - *certificate-type*: indica quale algoritmo a chiave pubblica verrà utilizzato;
   - *certificate-authorities*: lista di tutte le autorità riconosciute dal server.

   Nel sistema implementato M_4 non viene utilizzato. Infatti in un'applicazione user-friendly è impensabile che un'utente debba farsi rilasciare un certificato valido da un'autorità certificante prima di poter utilizzare il sistema. Pertanto l'autenticazione dell'utente presso il server avviene tramite utilizzo di password. Pertanto tale passaggio è stato omesso nell'implementazione finale.

5. Infine il server invia *ServerDone*, indicato in figura con M_5. Questo messaggio viene semplicemente utilizzato per segnalare che il server aspetta una risposta dal client.

6. Quindi entra in gioco il client, che dopo aver ricevuto il messaggio *ServerDone*, verifica se i parametri inviati dal server siano accettabili o meno: se tali controlli vanno a

buon fine, allora il client invierà un messaggio per informarlo sul successo dell'operazione. Inoltre se il server aveva richiesto nei messaggi precedenti il certificato del client, allora quest ultimo invia M_6 (*ClientCertificate*), contenente il proprio certificato.

7. A questo punto il client, manderà un altro messaggio M_7, denominato *clientKeyExchange*. In base al tipo di algoritmo utilizzato per lo scambio di chiavi, tale messaggio conterrà diverse informazioni. Considerando che l'algoritmo adoperato nel sistema finale è RSA, il messaggio conterrà un pre-master secret da 48-byte. Questo valore verrà utilizzato sia dal client che dal server per produrre il *master-secret* nel modo seguente:

$$\textit{master-secret}=$$
$$\textit{PRF(pre-master-secret,master-secret,Client.random||Server.random)}$$

Dopo aver generato il segreto il client cifrerà il messaggio con la chiave pubblica del server ottenuta dal messaggio M_3.

I restanti messaggi del client, ovvero M_8 e M_9, vengono utilizzati rispettivamente per memorizzare gli algoritmi da utilizzare nello strato applicativo e per avvertire il server che non ci saranno altri messaggi da parte dello stesso. Analogo discorso può essere fatto per i messaggi M_{10} e M_{11}.
Il contenuto dei messaggi *finish* è il seguente:

$$\textit{PRF(master-secret,label,MD5(handshake-msg)||SHA-1(handshake-msg))}$$

dove:

- *label*: è la stringa *client finished* o *server finished* a secondo di chi manda il messaggio;
- *handshake-msg*: si riferisce a tutti i messaggi inviati e ricevuto da ogni singola entità durante il protocollo di handshake.

Una volta terminato il protocollo di handshake, il client e il server possono iniziare lo scambio di dati sull'application-layer.

Arrivati a questo punto il sistema possiede due chiavi:

- K_1: una chiave segreta che viene utilizzata per la crittografia a chiave simmetrica;
- K_2: una chiave segreta per viene utilizzata per creare un *Message Authentication Code* (MAC).

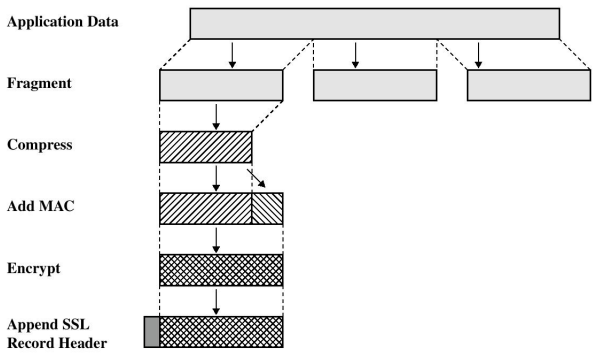


Figura 3.14: Record protocol del TLS.

Sulla base dei parametri concordati durante l'handshake, il Record Protocol garantirà due proprietà: confidenzialità e integrità dei messaggi. Al fine di comprendere al meglio il funzionamento si faccia riferimento alla Fig.3.14, la quale mostra l'insieme delle operazioni svolte:

- *frammentazione*: ogni messaggio dello strato superiore è diviso in blocchi di dimensione al più di 214-bit;
- *compressione*: nel caso di TLSv2 non viene effettuata alcuna compressione;
- *MAC*: viene aggiunto il message authentication code al frammento. Per la precisione viene utilizzato l'algoritmo HMAC, definito nel modo seguente:

$$HMAC_K = H(K^+ \oplus opad || H(K^+ \oplus opad) || M)$$

- *encryption*: il frammento con il MAC viene criptato secondo l'algoritmo scelto, che può essere uno tra l'AES (con chiave 128 o 256), 3DES (con chiave 168) o RC4-128 (con chiave 128). All'interno del progetto sono stati analizzati e presi in considerazione solo i cifrari a blocchi. In questo caso, il messaggio in input, prima di essere criptato, può essere soggetto ad un padding. Questo serve per evitare attacchi basati sulla lunghezza dei messaggi scambiati;
- *header*: il messaggio ottenuto viene appeso ad un header, contenente quattro campi differenti, quali il tipo di contenuto, le versioni più datata e recente di TLS e la lunghezza in byte del frammento in chiaro.

Durante lo scambio dei messaggi si è accennato al fatto che nell'implementazione proposta il server non fa alcuna richiesta al client riguardo l'invio del suo certificato. Per-

tanto è necessario introdurre l'altro elemento fondamentale dell'implementazione, ovvero l'autenticazione dell'utente presso il server.

Prima di poter usufruire del servizio è indispensabile una fase di registrazione da parte dell'utente tramite l'utilizzo di un form. A tal fine viene chiesto all'utente di inserire username e password. Ovviamente quest'ultima non può essere memorizzata in chiaro nel database del server poichè sarebbe vulnerabile. Per poter ovviare al problema l'idea è quella di salvare il valore hash della password (il *digest*). Il maggior vantaggio fornito dall'uso delle funzioni hash è rappresentato dall'impossibilità di poter risalire al valore originale una volta generato il digest.

Tra le più famose funzioni hash vi è quella che prende il nome di **Message-Digest Algorithm**, che genera un valore hash da 128-bit. Essa mappa un insieme di dati di lunghezza variabile in un altro di lunghezza fissa. Purtroppo, anche se questo algoritmo è veloce e molto facile da implementare, è lontano dall'essere sicuro. Infatti risulta essere debole nel caso di attacchi come il *brute-force* o quello del *dizionario*. Inoltre non è *collision-resistant*, ovvero più password possono avere lo stesso valore hash. Un miglioramento si ottiene se si aggiunge il *salt*, ovvero un valore random che può essere utilizzato, in input, come informazione aggiuntiva. Ogni volta che viene utilizzato il sale per una data password, questo deve essere memorizzato nel database poiché nel momento in cui l'utente riaccede al sistema se viene utilizzato un valore di sale differente, allora il digest sarà differente. Una funzione hash più potente è la **Secure Hash Algorithm** (SHA). Anche se non è collision-resistant, con questa funzione le collisioni avvengono molto raramente.

Il problema di fondo è che oggigiorno l'hardware dei dispositivi elettronici è talmente potente e veloce che è possibile effettuare un attacco brute-force senza troppi sforzi. L'obiettivo, allora, consiste nel rendere l'attacco a forza bruta più lento possibile in modo da minimizzare l'eventuale danno. Pertanto è necessario utilizzare algoritmi che da un lato garantiscano la condizione appena enunciata e dall'altro siano abbastanza veloce per non creare problemi di ritardo. Queste due caratteristiche possono essere ritrovate in algoritmi CPU intensive, come *PBKDF2*. La funzione può essere descritta nel modo seguente:

$$P = PBKDF2(PRF, Password, Salt, C, kLen)$$

dove:

- *PRF*: una funzione pseudo-random;

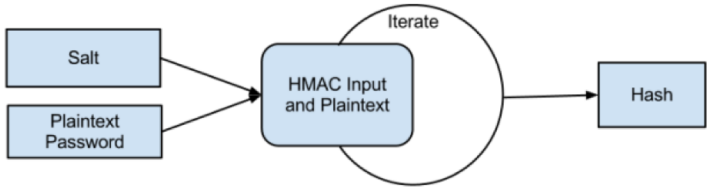


Figura 3.15: Schema logico PBKDF2.

- *password*: la password scelta dall'utente;
- *salt*: è una sequenza di bit;
- *C*: è il numero di iterazioni desiderate;
- *kLen*: è la lunghezza desiderata, al più $(2^{32}-1)*hLen$.

Il funzionamento è molto semplice e può essere generalizzato col diagramma di Fig.3.15. La peculiarità di questo algoritmo risiede nell'implementazione di un HMAC iterativo che continua ad incrementare la sicurezza e il cracking time del valore hash. Questa peculiarità permette di aumentare il livello di resistenza al crack della password semplicemente aumentando le iterazioni dell'algoritmo. Per capire meglio il funzionamento si può far riferimento allo pseudo-codice illustrato di seguito.

Algorithm 3: PBKDF2

Input : $P, S, C, kLen$
Output: mk (Master key)

if $(kLen > (2^{32}-1)*hLen)$ **then**
  return error and stop;
end
$len = \lceil kLen/hLen \rceil$; $r = kLen - (len-1)*hLen$;
for $i = 1$ *to* len **do**
  $T_i = 0$;
  $U_0 = S||Int(i)$;
  for $j = 1$ *to* C **do**
    $U_j = PRF(P, U_j - 1)$;
    $T_i = T_i \oplus U_j$;
  end
end
return $mk = T_1||\ldots||T_{len}$

L'algoritmo inizia effettuando un controllo sulla lunghezza desiderata in uscita e se questa dovesse superare quella massima allora verrà restituito errore. Successivamente viene calcolato il numero di blocchi della chiave in output. Si ricordi che l'operazione indicata con

$$\lceil a \rceil = ceiling(a);$$

si riferisce al più piccolo numero intero è minore o uguale ad *a*. Inoltre la quantità *r* si riferisce al numero di ottetti contenuti nell'ultimo blocco. Dopo ciò, per ognuno di essi,

viene applicata una funzione F a cui verranno passati come parametri P, S, C e l'indice del blocco analizzato. Si avrà qualcosa del tipo:

$$T_1 = F(P,S,C,1)$$
$$\vdots$$
$$T_{len} = F(P,S,C,len)$$

La funzione F viene definita come uno XOR esclusivo tra la T_i e le C iterazioni della PRF applicata alla password e al contenuto della U al passo precedente. Per maggiore chiarezza, si può affermare che le U assumono la seguente forma:

$$U_1 = PRF(P,S||Int(i))$$
$$\vdots$$
$$U_C = PRF(P,U_{c-1})$$

dove $Int(i)$ rappresenta la codifica a 32-bit di i con il bit più significativo a sinistra. Infine concatenando le varie T_i, otteniamo la chiave in uscita.

Dopo che l'account del Client viene creato correttamente, il server la quadrupla:

$$[I,U,H,S]$$

dove:

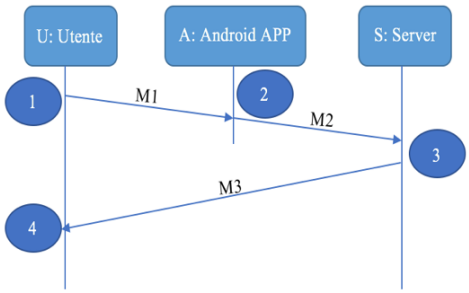
- I è un identificatore univoco;
- U è l'username; e due valori;
- H ed S, i quali si riferiscono rispettivamente al valore hash e al sale.

In questo modo non viene lasciata traccia in memoria della password del client. Questo fatto è cruciale poiché facilita la gestione delle politiche di sicurezza del server e semplifica il protocollo.

Prima di proseguire con la trattazione è necessario descrivere dove verrà calcolato il digest e perchè. A tal proposito si consideri un caso di studio in cui è necessario che l'utente si autentichi secondo i meccanismi descritti fino a questo momento. Si supponga che il valore hash della password venga calcolato dal client e inviato al server. Quest'ultimo accetterà il valore e controllerà se esiste una corrispondenza sul database. Questo approccio sembrerebbe più sicuro poiché la password dell'utente non viene mai inviata al Server. Purtroppo non è così. Infatti, secondo questo approccio, il digest calcolato lato client, e successivamente

inviato, rappresenta il testo in chiaro della password. In questo modo vengono persi tutti i benefici portati dall'utilizzo delle funzioni hash. Se un attaccante riuscisse ad ottenere il valore hash, allora si potrebbe autenticare col server senza conoscere la password originale e, quindi, avere accesso al database col minimo sforzo. Questo significa che un buon approccio, potrebbe essere quello di inviare al server la password cifrata e dopo calcolare l'hash per vedere se esiste una corrispondenza nel database.

Dopo che la fase di registrazione da parte dell'utente è completata si può descrivere come avviene il login al sistema. A tal fine si consideri lo schema di Fig.3.16.



$M_1)$	$U \rightarrow A : U, password$
$M_2)$	$A \rightarrow S : A, E(K, password)$
$M_3)$	$S \rightarrow U : S, E(K, response)$

Figura 3.16: Protocollo di autenticazione.

Al fine di comprendere al meglio i meccanismi di questo protocollo, si consideri la Tabella che riporta lo pseudo-codice dello schema di Fig.3.16. In particolare, nella *fase 1*, l'utente immetterà la sua password sul dispositivo Android. Successivamente, nella *fase 2*, tale password viene inviata al server. Si ricordi che tale scambio di informazioni verrà effettuato dopo il protocollo TLS e, di conseguenza la password non viaggerà in chiaro. Nella *fase 3* il server riceverà la password, calcolerà il valore hash corretto ed infine restituirà un messaggio di errore o conferma a seconda se esiste un match nel DB.

1	$U = getMyID();$ $plainPassword = getPassByForm();$ $M_1 = [U, plainPassword];$ $send(M_1, A);$
2	$M_1 = receive();$ $[U, plainPassword] = M_1;$ $A = getMyID();$ $chiperText = Enc(K, plainPassword);$ $M_2 = [A, chiperText];$ $send(M_2, S);$
3	$M_2 = receive();$ $S = getMyID();$ $[A, chiperText] = M_2;$ $[plainPassword'] = Dec(K, chipherText);$ $\pi = PBKDF2(PRF, plainPassword', S, C, kLen);$ $bool = verifyPassword(\pi, storedPassword);$ $response = Enc(K, bool);$ $M_3 = [S, response];$ $send(M_3, U);$
4	$M_3 = receive();$ $[S, response] = M_3;$ $[bool] = Dec(K, response);$ $if(bool == true)ACCEPTelseABORT;$

Tabella 3.1: Pseudo-codice protocollo autenticazione.

Un concetto che non deve essere sottovalutato è quello di come scegliere una password appropriata per ogni utente. Infatti, quando i sistemi informatici fanno uso di password, è necessario tenere a mente che un attaccante è in grado di effettuare attacchi offline. Di conseguenza le password devono avere:

- *alta entropia*: permette di rallentare notevolmente la velocità degli attacchi offline. Generalmente viene lasciato al sistema il compito di generazione della password random, poichè a parità di lunghezza, con quelle scelte dall'utente, presentano un valore di entropia molto più elevato;
- *lunghezza*;
- *casualità*.

A prescindere da tali fattori, per rendere l'applicazione Android *user-friendly*, è stato previsto la possibilità da parte dell'utente di poter scegliere autonomamente la password, purchè soddisfi alcuni parametri che è possibile descrivere riferendosi alla Tabella 3.2 [34]. Infatti

una password composta da 8 lettere, tra maiuscole e minuscole, e un numero di iterazioni pari a 1000 dovrebbe risultare sufficiente per scoraggiare un attaccante.

Complessità Password	**Entropia** (bits)	**Iterazioni** 1000	**Iterazioni** 10000
Comprehensive 8	33	4:36 ore	47 ore
8 lettere random minuscole	37	12 ore	5 giorni
8 lettere random	55	123 giorni	3 anni e 5 mesi
8 lettere random + numeri + punteggiatura	52	325 giorni	3250 anni

Tabella 3.2: Analisi *PBKDF2*.

Capitolo 4

Valutazione sperimentale

Per verificare l'efficacia del sistema individuato quale caso di studio nel Capitolo 3, sono state condotte un insieme di prove sperimentali.

Il seguente capitolo è suddiviso nel modo seguente: nella sezione 4.1 viene mostrata l'interfaccia sia lato client sia lato server dell'intero sistema. In particolare, per il client, verrà mostrata l'applicazione Android e come è possibile avviare e fermare il processo di classificazione; per il server, invece, verranno brevemente introdotte quali siano le funzioni a cui è possibile accedere. Nelle sezione 4.2 verranno descritti tutti gli esperimenti e, per ognuno di essi, saranno analizzate gli aspetti positivi e/o negativi. In primis, in 4.6, verranno analizzate le due tecniche di machine learning, prese in esame nel paragrafo 3.3, al fine di capire quale sia migliore per essere implementata nel sistema finale. Successivamente, in 4.2.2, si passerà al confronto con alcune tecniche allo stato dell'arte, descrivendo i punti di forza del framework. Infine in 4.7 e 4.2.4 si entrerà nel dettaglio delle prestazioni dell'architettura Client/Server. In particolare verrà stabilito quale algoritmo di crittografia, tra AES e 3DES, sia più adatto per l'implementazione sugli smartphone, per arrivare poi all'analisi delle tempistiche necessarie all'architettura per svolgere le varie funzioni.

4.1 L'applicazione

Per il confronto con gli altri sistemi, che verrà descritto nella successiva sezione, è stato necessario implementare un'applicazione Android ad-hoc al fine di poter raccogliere ed elaborare i dati. La seguente sezione descrive l'interfaccia dell'intero sistema e di come un utente può interagire con essa.

In principio si prenda in considerazione la Fig.4.1 che mostra l'interfaccia dell'applica-

zione Android. In particolare gli screen fanno riferimento alla schermata di login o autenticazione (a), di presentazione dell'applicazione (b), quella relativa alla fase di training (c) e quella della fase di test (d).

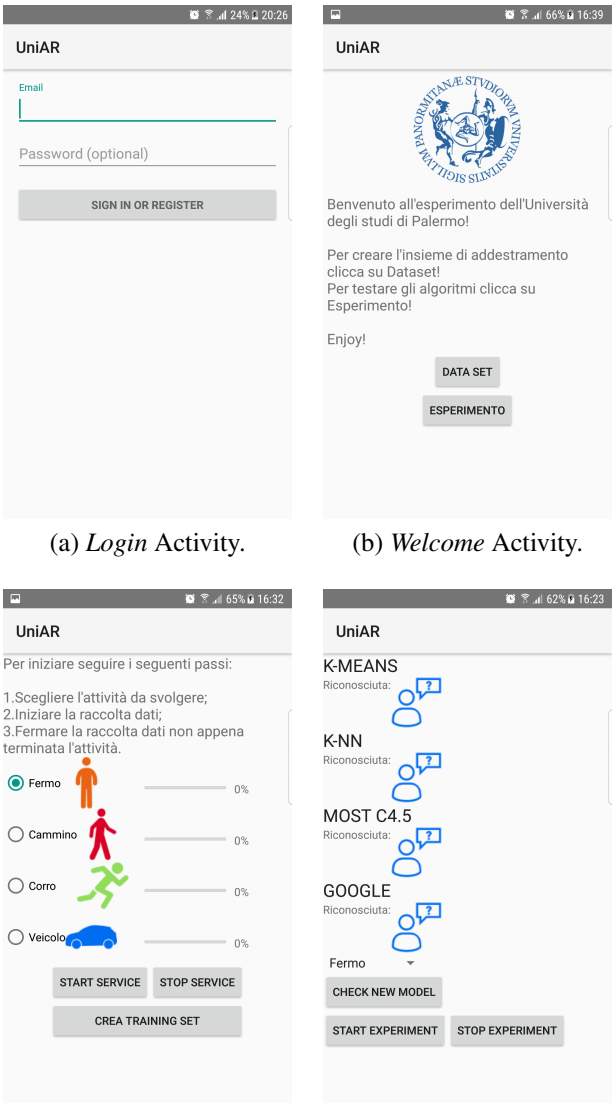
Quando l'applicazione viene avviata per la prima volta, l'utente si ritroverà la Fig.4.1 (a) che richiede di inserire username e password secondo quanto descritto nel paragrafo 3.4.2. Successivamente verrà indirizzato alla pagina di benvenuto dell'applicazione che permetterà, tramite i due *button*, di passare o all'interfaccia per la creazione dell'insieme di addestramento o per la fase di testing. La prima delle due Android activity mostra diversi elementi, quali:

- *progress bar*: per ognuna delle attività devono essere raccolti 5 minuti. Gli spinner vengono utilizzati per tener traccia della percentuale di completamento per ogni attività da svolgere;
- *start service*: serve per iniziare in background il processo di raccolta dei dati grezzi;
- *stop service*: serve per terminare il processo di raccolta dati;
- *crea training set*: una volta che, per ognuna delle attività, sono stati raccolti sufficienti dati, viene creato l'insieme di addestramento e inviato al server.

Una volta che l'insieme di addestramento è completo è possibile passare alla successiva fase di testing, la cui interfaccia è mostrata in Fig.4.1 (d). Anche qui vi sono diversi elementi:

- *lista algoritmi*: per ognuno dei algoritmi, presi in considerazione nel seguente lavoro di tesi, viene assegnata un'immagine che mostrerà quale attviità sia stata riconosciuta alla fine del processo di analisi dei dati;
- *spinner*: viene utilizzato per dare all'utente la possibilità di esprimere un feedback sull'attività riconosciuta;
- *start experiment*: serve per iniziare in background il processo di raccolta dei dati grezzi;
- *stop experiment*: serve per terminare il processo di raccolta dati, dare il via a quello di estrazione delle feature ed, infine, il processo di classificazione.

L'architettura finale permette all'*amministratore*, correttamente autenticato, di poter accedere alla informazioni contenute nel server. In particolare la Fig.4.2 mostra l'interfaccia sul browser. A sinistra vi sarà un menu a tendina con cui è possibile selezionare varie opzioni:



(a) *Login* Activity. (b) *Welcome* Activity.

(c) *Training* Activity. (d) *Test* Activity.

Figura 4.1: Interfaccia applicazione Android.

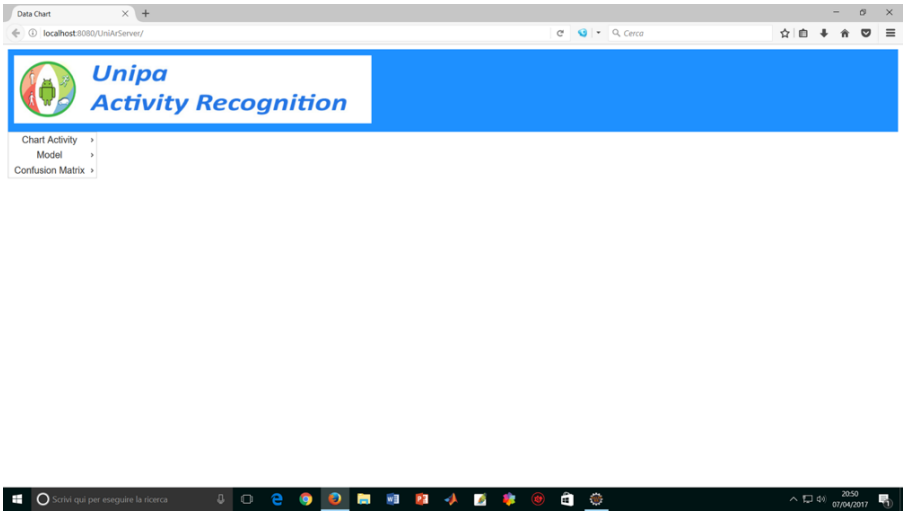


Figura 4.2: Interfaccia per l'end-user.

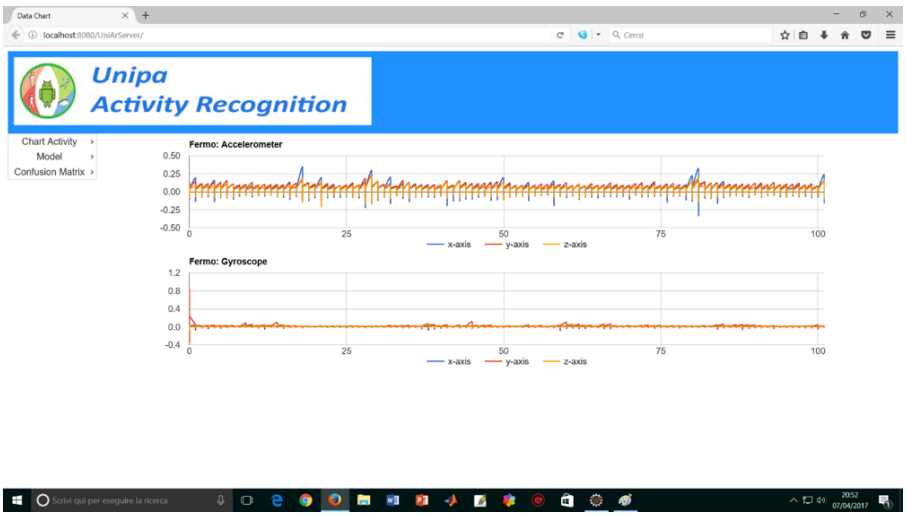


Figura 4.3: Interfaccia per l'end-user: pattern dell'attività *Fermo*.

- *Chart Activity*: permette di stampare il pattern delle attività sia per l'accelerometro che per il giroscopio. Un esempio è mostrato in Fig.4.3 che descrive l'attività "Fermo".
- *Model*: permette di visualizzare informazioni relative ai modelli degli algoritmi di classificazione utilizzati per il riconoscimento delle attività. Per esempio è possibile stampare i centroidi.
- *Confusion Matrix*: permette di mostrare le confusion matrix relative ad ogni algoritmo e le percentuali di Accuracy, Precision e Recall. Questo rappresenta uno strumento fondamentale per l'analisi del sistema.

4.2 Esperimenti

Al fine di valutare l'efficienza del framework, sono stati effettuali diversi esperimenti. Per prima cosa viene presentato un confronto tra i risultati ottenuti utilizzando le due tecniche di K-Means e K-NN. La scelta dell'algoritmo di classificazione da adottare si vedrà essere dipendente da due parametri: accuracy e l'efficienza in termini di complessità e consumo della memoria. Dopo aver appurato quale sia l'algoritmo migliore, le performance del sistema HAR proposto vengono confrontate con due tecniche allo stato dell'arte, ovvero le Google API e il sistema di activity recognition fornito da MoST. Oltre a ciò, nella fase finale vengono valutate le performance dell'architettura Client/Server, ponendo particolare attenzione al tempo richiesto per completare correttamente un ciclo di request/response.

4.2.1 K-Means vs K-NN

Prima di procedere con il confronto diretto tra i due algoritmi, è stata necessaria una fase di prototipazione, sia per il K-Means che per il K-NN, al fine di capire quali siano i parametri migliori per la loro implementazione nel sistema finale. Dopo che verranno forniti questi parametri, la sezione proseguirà con la comparazione diretta tra i due algoritmi in termini di performance e accuracy, precision e recall.

Il K-Means è un algoritmo di clustering partizionante e, di conseguenza, non viene utilizzato per il processo di classificazione. Tuttavia nel paragrafo 3.3 è stata presentata una logica con cui è possibile adattare tale algoritmo a scopi di classificazione. In particolare sono state definite due configurazioni, sulla base delle quali, gli esperimenti sono stati effettuati analizzando quattro casi di studio differenti. Nel dettaglio tali studi sono serviti per due scopi: in primis si è voluto analizzare se la classificazione di un nuovo punto debba avvenire, a partire dalla funzione di stima, o direttamente dai dati grezzi o dai valori statistici; in secondo luogo stabilire quali siano i parametri migliori per una corretta implementazione del K-Means. I quattro casi vengono presentati di seguito:

- *caso 1*: implementazione con la *configurazione 2* utilizzando la distanza euclidea tra valori statistici;
- *caso 2*: implementazione con la *configurazione 2* utilizzando la distanza euclidea con i dati grezzi;
- *caso 3*: implementazione con la *configurazione 1* utilizzando la distanza euclidea con i dati grezzi;
- *caso 4*: implementazione con la *configurazione 1* utilizzando la distanza euclidea tra valori statistici.

Un parametro chiave per il K-Means è rappresentato dal numero di centroidi che caratterizzano una singola classe. Soprattutto nella configurazione 1, in cui vengono generati c centroidi per ogni classe, la scelta di tale valore rappresenta un fattore cruciale poichè può incidere negativamente o positivamente sull'efficacia e complessità dell'algoritmo stesso.

Al fine di comprendere i risultati ottenuti negli esperimenti si faccia riferimento alle Tabelle 4.1, 4.2, 4.3 e 4.4. Le prime considerazioni possono essere fatte in merito alle Tabelle 4.2 e 4.3. Infatti è possibile notare come al variare dei seguenti parametri:

- r: finestra di registrazione;

r	w	n	Accuracy	Precision	Recall
240	2	200	84.4436	72.9880	76.9870
			85.8308	78.5197	79.2367
			86.0336	79.4820	79.6831
	3	300	84.0306	72.9992	75.9780
			85.9591	76.4058	79.1337
			86.8767	80.6228	80.7894
			86.4081	80.2429	80.1565
300	2	200	84.4478	77.5717	76.9559
		100	82.7757	71.6638	74.1907
		200	86.1150	77.4464	79.4818
	3	200	87.0173	80.0397	80.7423
		300	82.8773	70.4096	74.6802
		300	86.7973	80.2840	80.5662
		150	87.5799	80.3285	81.4420
		300	86.7770	80.6802	80.6312

Tabella 4.1: Risultati per caso 1.

r	w	n	Accuracy	Precision	Recall
240	2	200	60.41	58.99	61.97
			58.42	59.55	60.67
			57.92	57.42	58.62
	3	300	58.03	60.96	59.97
			62.71	60.05	62.33
			57.86	59.62	58.74
			59.41	60.29	59.65
300	2	200	62.48	62.57	59.99
		100	62.77	61.66	62.19
		200	57.1	61.64	58.88
	3	200	58.01	60.07	60.23
		300	63.87	60.49	60.68
		300	61.79	61.84	60.01
		150	59.99	59.34	61.4
		300	60.79	60.32	61.31

Tabella 4.2: Risultati per caso 2.

r	w	n	c	Accuracy	Precision	Recall
240	2	200	3	71.3	61.95	60.07
			4	72.56	63.24	62.03
			5	73.56	69.79	63.69
	3	300	3	67.77	65.14	64.55
			4	68.84	60.26	66.2
			5	68.96	69.16	66.47
300	2	100	3	70.72	70.75	69.07
			3	71.06	70.17	69.9
		200	4	71.42	69.29	70.16
			5	74.24	68.36	64.79
	3	75	4	69.46	64.46	66.97
		150	4	71.41	68.31	70.09
		300	5	72.2	68.06	71.45

Tabella 4.3: Risultati per caso 3.

r	w	n	c	Accuracy	Precision	Recall
240	2	100	1	96.88	94.14	95.04
		200	1	97.45	95.16	95.95
		200	2	96.81	93.88	93.99
	3	150	2	98.86	97.78	97.93
		150	3	99.16	98.26	98.6
		150	4	99.01	97.96	98.36
		300	4	99.21	98.36	98.67
300	2	100	3	97.39	94.98	95.66
		200	3	98.75	97.58	97.7
		100	4	98.29	96.71	96.7
	3	300	2	98.41	96.9	97.87
		150	3	98.76	97.63	97.66
		300	4	99.31	98.59	98.75

Tabella 4.4: Risultati per caso 4.

- w: finestra di osservazione;
- n: numero di campioni per finestra di osservazione;
- c: numero centroidi c;

i valori percentuali di accuracy, precision e recall risultano essere non molto significativi ai fini di una corretta classificazione. Questo risultato ha permesso di stabilire che un approccio migliore è quello che fa uso dei valori statistici descritti nel paragrafo 3.2 poichè le tabelle 4.1 e 4.4 mostrano valori di gran lunga migliori. Particolarmente significativi sono i valori riportati nel *caso di studio 4*, dove la configurazione 1 mostra risultati migliori. A partire da ciò, considerando che il sistema verrà implementato su un dispositivo Android e che il numero di centroidi può influenzarne le prestazioni, si è deciso di settare i seguenti parametri per l'implementazione del K-Means nel sistema finale:

- $r = 300\ sec = 5\ min$;
- $w = 3\ sec$;
- $n = 300$;

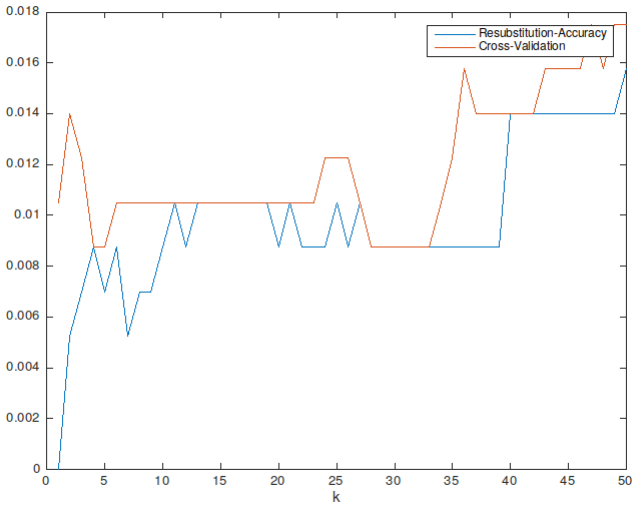


Figura 4.4: Resubstitution Accuracy e Cross Validation per K-nn.

- $c = 2$.

Terminata l'analisi del K-Means è possibile passare a quella del K-NN. Per quanto riguarda questo algoritmo sono stati fatti dei test per determinare, al variare di k, le due misure di *Re-Substitution Error Rate* e la *Cross Validation*. La figura 4.4 mostra i risultati ottenuti dalla prototipazione su MATLAB.

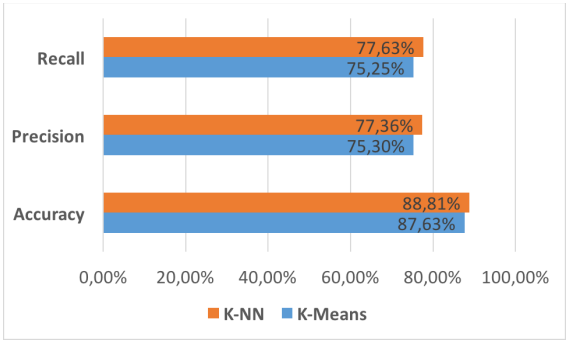
Considerando che il K-NN deve girare su un dispositivo con risorse limitate, i valori di k considerati non superano 50. Questo è giustificato dal fatto che all'aumentare di k la complessità computazionale dell'algoritmo cresce di molto. Ai fini dell'esperimento, sulla base del plot, è stato considerato:

$$k = 7$$

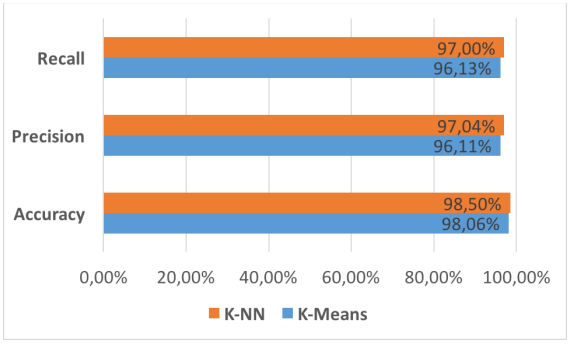
*** OMISSIS ***

Finalmente, dopo aver definito quali siano i parametri migliori per entrambi gli algoritmi, è arrivato il momento di verificare quale delle due tecniche risulti migliore per il framework finale. A tale scopo è stato chiesto a diversi utenti di svolgere le quattro attività per una intera settimana.

*** OMISSIS ***



(a) *Accelerometro.*



(b) *Accelerometro e giroscopio.*

Figura 4.5: Valori di accuracy, precision e recall ottenuti utilizzando solamente l'accelerometro (a) e accelerometro e giroscopio (b).

In un primo momento gli esperimenti sono stati effettuati utilizzando solamente i valori letti dall'accelerometro. Questi test consentono di mostrare l'utilità del giroscopio nel sistema. I risultati ottenuti, rappresentati in Fig.4.5(a), mostrano percentuali di accuracy, precision e recall non troppo elevate per entrambi gli algoritmi. Tali valori sono dovuti principalmente alla difficoltà da parte del K-Means e del K-NN di distinguere correttamente le attività di *Fermo* e *Veicolo*. Al fine di migliorare tali valori, oltre all'accelerometro, è stato preso in considerazione anche il giroscopio. Grazie alle informazioni aggiuntive introdotte da questo sensore (si veda paragrafo 3.2), il riconoscimento delle attività è nettamente migliorato per entrambi gli algoritmi di machine learning, come mostrato in Fig.4.5(b). Sulla base delle metriche analizzate risulta difficile dire quale algoritmo sia più adatto per il sistema descritto nel seguente documento, poichè i valori differiscono di poco.

Essendo gli smartphone Android dispositivi con risorse limitate, sono state effettuati ulteriori studi per valutare tali algoritmi in termini di tempo di elaborazione e carico della memoria.

*** OMISSIS ***

Sono stati condotti cinque differenti test (indicati in figura con le lettere maiuscole) sulla base di un certo intervallo temporale. In particolare per i casi A e B, tale intervallo è di circa 2 minuti, mentre per i restanti casi di circa 50 secondi. Le figure evidenziano due esiti importanti:

- il K-Means impiega meno tempo rispetto al K-NN per il processo di classificazione;
- considerando il carico di memoria risulta che il K-NN, al variare del tempo di raccolta

dati, ha un andamento quasi lineare a differenza del K-Means che, invece, è influenzato da tale parametro.

Conseguentemente ai risultati descritti, a parità dei valori di accuracy, precision e recall, il K-NN risulta essere un candidato migliore rispetto al K-Means. La motivazione risiede nel fatto che il minor carico della memoria risulta essere un fattore preponderante rispetto alla velocità di elaborazione. Risultati analoghi a quelli descritti nel seguente paragrafo sono presentati in [35].

*** OMISSIS ***

4.2.2 Il sistema finale vs Google vs MosT

Dopo aver ultimato l'implementazione del sistema sulla base dei risultati descritti nel paragrafo 4.2.1, sono stati effettuati esperimenti al fine di confrontarlo con il framework MosT e le API di Google. Essendo che entrambe le implementazioni riconoscono un differente numero e tipo attività, al fine di poter effettuare un diversi tipi di confronto col sistema descritto in questo lavoro, è stato necessario considerare due sottoinsiemi delle classi. In particolare è stata introdotta la classe *Other* che fa riferimento all'insieme delle attività che non devono essere considerate nel confronto. Sulla base di tale considerazione, i sottoinsiemi costruiti sono mostrati in Tabella 4.5.

Il primo set di esperimenti ha mostrato la necessità dell'introduzione del giroscopio nel nostro sistema. Per tali test sono stati messi a confronto il sistema finale e il framework MosT. In particolare sono stati considerati l'accelerometro, come unico sensore, e gli insiemi di activity mostrati nella Tabella 4.5. I risultati ottenuti (vedere Fig.4.6) evidenziano che, a differenza di MosT, i parametri di accuracy, precision e recall per il sistema finale assumono valori percentuale non ottimali. Questi dati vengono giustificati dal fatto che l'accelerometro non fornisce sufficienti informazioni per consentire la corretta classificazione delle attività di *Fermo* e *Veicolo*, come mostra la confusion matrix presentata nella Tabella 4.8.

A partire da tali risultati, l'idea è stata quella di aggiungere il giroscopio oltre che all'accelerometro. Inoltre, sulla base degli ottimi risultati ottenuti con il C4.5 di MosT e al fine di confrontare tra loro tutti i sistemi in modo diretto, l'algoritmo implementato dentro la libreria MosT è stato modificato in modo da poter riconoscere esattamente lo stesso numero di

Confronto	Insiemi per il confronto
Sistema Vs. MosT	*{Fermo, Corro, Cammino, Other}* *{Fermo, Cammino, Corro}*
Sistema Vs. Google	*{Fermo, Corro, Cammino, Veicolo}* *{Fermo, Corro, Cammino, Veicolo, Other}*

Tabella 4.5: Insiemi considerati per effettuare il confronto diretto tra il sistema finale e le implementazioni di Google e MosT.

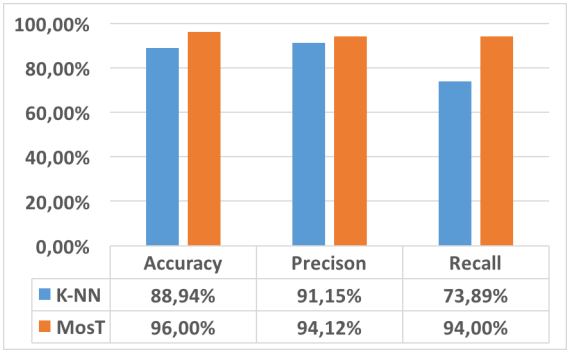


Figura 4.6: Confronto tra il sistema e MosT utilizzando solo l'accelerometro.

classi preso in esame nel nostro framework. Per quanto riguarda Google, come già discusso nel capitolo 2, esso non fornisce alcun dettaglio su come funziona internamente il processo di riconoscimento delle attività e di conseguenza, non potendone modificare il codice, è stato considerato il sottoinsieme mostrato nella Tabella 4.5. Sulla base di ciò i risultati possono essere descritti facendo riferimento a Fig.**??**.

*** OMISSIS ***

Questa evidenzia due importanti risultati:

- il framework MosT, rispetto ai valori presentati in Fig.4.6, è peggiorato notevolmente;
- Google presenta valori percentuali più bassi rispetto a quelli ottenuti dal nostro sistema. Il riconoscimento potrebbe migliorare se le Google API fossero impostate per riconoscere solamente le quattro attività prese in considerazione nel seguente documento. Purtroppo si ribadisce il fatto che Google utilizza un approccio di tipo black-box e quindi non è possibile effettuare tali modifiche.

	Fermo	*Cammino*	*Corro*	*Veicolo*
Fermo	.33	0	0	.67
Cammino	0	.96	.04	0
Corro	0	.02	.98	0
Veicolo	.43	0	0	.57

Tabella 4.6: Confusion matrix MoST.

	Fermo	*Cammino*	*Corro*	*Veicolo*	*Other*
Fermo	.92	0	0	0	.08
Cammino	0	.56	.44	0	0
Corro	0	.45	.55	0	0
Veicolo	0	0	0	.91	.09

Tabella 4.7: Confusion matrix Google.

	Fermo	*Cammino*	*Corro*	*Other*
Fermo	.59	0	.01	.41
Cammino	0	.99	0.01	0
Corro	0	0	1	0

Tabella 4.8: Confusion matrix K-NN.

In entrambi i casi il sistema descritto nel seguente lavoro risulta essere più performante rispetto alle altre due tecniche. Il motivo di tale affermazione può essere giustificato facendo riferimento alle Tabelle 4.6 e 4.7, che mostrano rispettivamente le *Confusion Matrix* di MosT e di Google. Innanzitutto dalla confusion matrix di MosT è possibile notare come, a causa dell'aggiunta dell'attività di *Veicolo*, l'algoritmo basato sugli alberi di decisione non riesca a discriminare correttamente tale attività con quella di *Fermo*, così come accadeva sia per il K-NN. Questo rafforza ulteriormente il concetto che un approccio corretto consiste nell'utilizzare la combinazione di accelerometro e giroscopio.

Per quanto riguarda Google, la confusion matrix mostra come tale sistema abbia grosse difficoltà a distinguere le attività di *Camminare* e *Correre*. Risultati simili a quelli descritti sono stati ottenuti in [25], dove per ovviare al problema viene aggiunto un meccanismo di geolocalizzazione che raccoglie campioni ogni 5 minuti. Purtroppo un approccio di questo tipo può risultare invadente e di conseguenza esula gli scopi degli studi effettuati in questo lavoro di tesi, poichè l'intento è stato quello di progettare un sistema non invasivo per il riconoscimento delle attività.

4.2.3 AES vs 3DES in Android

Nel paragrafo 3.4.1 sono stati introdotti l'AES e il 3DES, due algoritmi simmetrici per cifrare/decifrare un testo in chiaro al fine di fornire un meccanismo di sicurezza che garantisca *confidenzialità*. In scenari specifici, come quello del Participatory Sensing, è necessario implementare tali meccanismi per evitare che un attaccante possa leggere in chiaro le informazioni scambiate tra un client e un server. Al fine di rendere l'architettura Client/Server più performante possibile, sono stati condotti degli esperimenti per verificare quale delle due tecniche risulti essere più adatta per essere utilizzata sui dispositivi Android.

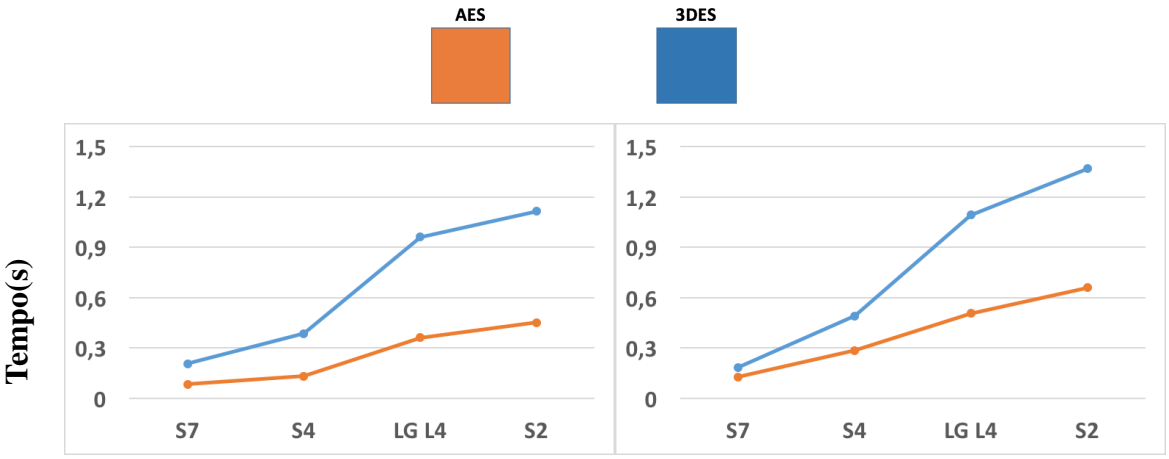
*** OMISSIS ***

Per poter verificare le prestazioni delle due tecniche è stata sviluppata un'applicazione ad-hoc per Android che permettesse di effettuare i confronti. Esistono differenti metriche per valutare tali tecniche [36]. Nel seguente lavoro sono state prese in considerazione:

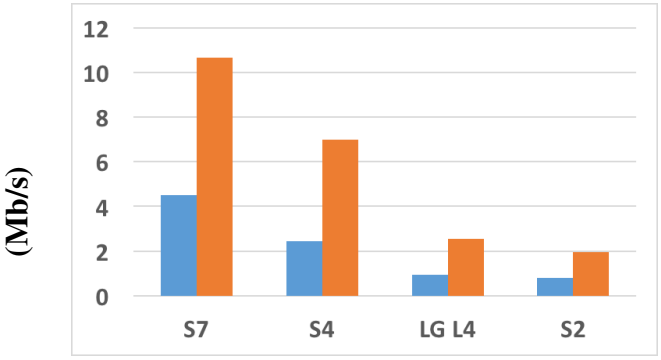
- **encryption time**: il tempo che il dispositivo impiega per produrre un testo cifrato da un testo in chiaro;
- **decryption time**: il tempo che il dispositivo impiega per produrre un testo in chiaro da un testo cifrato;
- **throughput**: il rapporto tra il numero totale di byte del testo criptato fratto l'encryption time.

Infine è importante sottolineare che i test sono stati effettuati variando la dimensione del file in input da cifrare/decifrare.

Al fine di riassumere i risultati ottenuti dai test si faccia riferimento alle Fig.4.7(a),(b) e (c), che mostrano rispettivamente i valori medi del tempo necessario per la cifratura, la decifratura e i throughput ottenuti per ogni dispositivo. I valori evidenziano come l'AES risulta essere meno dispendioso in termini di tempo e throughput rispetto al 3DES. Questo risparmio di tempo indica un minore uso delle risorse e di conseguenza l'AES è stato preferito rispetto al 3DES.



(a) *Encryption Time.* (b) *Decryption Time.*



(c) *Throughput.*

Figura 4.7: Confronto tra AES e 3DES in termini di encryption/decryption time e throughput.

4.2.4 Analisi temporale Client/Server

Le architetture Client/Server sono sistemi in cui:

- un client effettua una richiesta a un server;
- un server analizza la richiesta del client e risponde in base al servizio richiesto.

Tale scambio di informazioni prende il nome di *ciclo request/response*. Generalmente le due entità si scambiano dati di vario tipo sia in termini di contenuto informativo sia in termini di dimensione. In particolare quest ultimo può incidere negativamente sulle prestazioni dell'intera infrastruttura poichè all'aumentare della dimensione dei dati da inviare, crescerà il tempo richiesto per completare un ciclo di request/response.

Sulla base di quanto descritto è stato necessario effettuare un'analisi riguardo il tempo impiegato dal sistema per lo scambio di servizi e capire se sia possibile utilizzare un tipo di connessione rispetto che un'altra (es. WiFi, 3G, 4G).

Prima di procedere è necessario comprendere quali siano i dati scambiati tra il client Android e il server. Nel sistema finale possono essere distinti tre insiemi di dati:

- **Training Set Data**: dopo che ogni utente ha creato correttamente tale insieme, questo verrà inviato al server, il quale si occuperà di memorizzarlo nel suo database. Infine il server aggiornerà i modelli e li invierà come risposta al client Android, che li utilizzerà per le successive fasi di riconoscimento delle attività.
- **New Model Data**: il client può chiedere al server di controllare se esistono dei modelli aggiornati e, in caso affermativo, farseli inviare.
- **New Samples Data**: dopo che l'utente termina di raccogliere i dati sensoriali, questi insieme al feedback verranno inviati al server. Tramite queste informazioni, saranno

aggiornate le confusion matrix dei vari algoritmi e nuovi elementi saranno inseriti nell'insieme di addestramento solo se passeranno un controllo basato su un valore di threshold.

La Tabella 4.9 mostra i risultati ottenuti analizzando il tempo necessario all'architettura per processare correttamente le tre tipologie di dati.

*** OMISSIS ***

I test hanno evidenziato come non vi siano particolari problematiche per gli insiemi dei dati relativi *Training Set* e *New Model*. Risultati più critici riguardano l'insieme *New samples*. Infatti il tempo di elaborazione aumenta al variare dell'intervallo temporale in cui vengono raccolti i dati dai sensori. In particolare, facendo riferimento al grafico di Fig.**??**(b), è possibile notare come all'aumentare del tempo di raccolta, la dimensione dei dati trasmessi dal Client al Server cresce molto rapidamente e in modo non lineare. Come conseguenza, il tempo richiesto dall'intero sistema per completare il processo di request/response risulta essere troppo alto. Infatti se la raccolta dei dati dovesse durare più di 20 minuti il sistema richiederebbe un tempo di elaborazione superiore ai 15 secondi.

Dati	**Dimensione** (KB)	**Tempo** (s)
Training Set	110	2,46
	110	1,09
	111	2,01
	110	1,53
	112	2,25
New Model	2,62	0,38
	2,76	0,54
	2,69	0,54
	2,63	0,39
	2,62	0,38
New Samples	467	1,95
	773	5,03
	1213,76	9,35
	2211,84	15,2

Tabella 4.9: Dimensione dei dati scambiati e tempo necessario per completare un ciclo request/response.

*** OMISSIS ***

Per poter risolvere tale problematica sono state analizzate due tecniche di compressione

molto conosciute in letteratura: CJSON e GZip. Questi algoritmi sono di tipo *loseless* e vengono adoperati per i file o dati per cui non è accettabile una perdita di informazione e di conseguenza risultano particolarmente adatti alla problematica presa in esame.

Anche in questo caso sono stati condotti alcuni esperimenti per verificare quale tecnica sia il migliore. Sulla base di quanto descritto in [37] sono state considerate le seguenti metriche:

- **tempo compressione**;
- **tempo decompressione**;
- **percentuale di salvataggio**: indica in termini percentuali di quanto viene ridotto il file sorgente;
- **rapporto compressione**: il rapporto tra la dimensione del file dopo essere stato compresso e la dimensione del file prima della compressione.

Al fine di testare gli algoritmi è stata creata un'applicazione Android che permettesse di calcolare le metriche sopra elencate. Per i test sono stati considerati 7 file testo di varie dimensioni (da 500 Kb fino a 4 Mb).

*** OMISSIS ***

Capitolo 5

Conclusioni

In questo documento è stato presentato un framework per la Human Activity Recognition basato sui sensori montanti a bordo del disposivo: *accelerometro* e *giroscopio*. Le caratteristiche delle quattro attività analizzate, cioè *Fermo*, *Cammino*, *Corro* e *Veicolo*, sono rappresentate tramite vettori di feature ottenuti elaborando i dati grezzi in input in finestre temporali di lunghezza fissa. Un classificatore basato sul K-NN viene applicato per riconoscere le attività svolte da un utente. Inoltre questa architettura include un meccanismo per il Participatory Sensing che consente di sfruttare i feedback degli utenti riguardo l'attività svolta al fine di poter valutare gli algoritmi di machine learning. Oltre a ciò, grazie all'architettura Client/Server, è possibile aggiornare i modelli di tali algoritmi e raccogliere dati sensoriali da più utenti in modo da rendere l'intero sistema più completo possibile.

Nelle fasi finali sono stati effettuati degli esperimenti per confrontare il sistema, descritto nel seguente lavoro di tesi, con due tecniche molto conosciute in letteratura, quali MoST e Google API. I risultati ottenuti hanno mostrato l'effiacia del sistema finale sia in termini di accuratezza che di efficienza.

Come lavori futuri, un'idea potrebbe essere quella di progettare un algoritmo ad-hoc per poter gestire automaticamente il rumore che viene generato da movimenti fuorvianti durante le fasi di avvio e stop dell'applicazione. Questo rappresenta un punto cruciale per un buon riconoscimento delle attività e va gestito in modo adeguato, poiché ogni utente ha delle tempistiche diverse per lo svolgimento di queste fasi. L'algoritmo, per esempio, potrebbe far uso di un meccanismo dinamico che adopera finestre di lunghezza variabile per il riconoscimento del rumore.

Seguendo la logica presentata nel documento, un altro lavoro futuro potrebbe essere quello di riconoscere attività più complesse come andare in ufficio, giocare ecc. In questo nuovo

scenario i comportamenti semplici (*Fermo*, *Cammino*, *Corro* e *Veicolo*) indicheranno le azioni, mentre quelli più complessi verranno considerati activity. Intuitivamente queste ultime sono molto più complesse, ma allo stesso tempo sono più rappresentative della reale vita di una persona e ciò è giustificato dal fatto che le persone compiono simultaneamente diversi tipi di azioni durante il giorno. L'idea base per il riconoscimento di queste attività complesse consiste nel catturare le relazioni temporali che sussistono tra le azioni in modo sequenziale o concorrente. Sulla base di questo, pertanto, l'intenzione è quella di sviluppare un algoritmo che permetta di descrivere in modo generale le relazioni temporali tra le semplici azioni atomiche.

Infine, verranno prese in considerazione algoritmi di gestione della reputazione, come quelli descritti in [38, 39], che possano stimare l'affidabilità degli utenti del sistema al fine di dare maggior peso ai feedback rilasciati dagli utenti più affidabili.

Appendice A

Certificati Self-signed

A.1 Installazione strumenti

I passi mostrati di seguito sono stati effettuati su una versione di *Windows 10 64-bit*. In ogni modo, le seguenti istruzioni possono essere eseguito su altri sistemi operativi a patto che i tool utilizzati vengano configurati in modo corretto.

- **Installare OpenSSL**: per fare ciò è possibile seguire diversi modi. Nel progetto si è scelto di installare il tool tramite GnuWin, scaricabile al seguente link:

  https://sourceforge.net/projects/gnuwin32/files/openssl/0.9.8h-1/openssl-0.9.8h-1-setup.exe/download?use_mirror=vorboss
- **Keytool**: questo è uno strumento messo a disposizione da Java. Questo vuol dire che se Java è configurato correttamente nella macchina su cui si sta lavorando, dovremmo trovare l'eseguibile in un path di questo tipo:

  ```
  C:\Program Files (x86)\Java\jre1.8.0_111\bin\keytool.exe
  ```

  Se l'eseguibile esiste possiamo procedere con le altre fasi.

- **Bouncy Castle Provider**: prima di poter creare i certificati dobbiamo configurare il provider. Tra tutti quelli esistenti, è stato scelto il seguente provider, poiché risulta essere compatibile con Android. Per configurare correttamente il provider, per prima cosa abbiamo scaricato il relativo file .jar reperibile al seguente link:

  http://www.bouncycastle.org/latest_releases.html

Una volta scaricato deve essere configurato in modo tale che Java possa utilizzarlo. A tal fine, dalla documentazione ufficiale Oracle, abbiamo eseguito i seguenti passi:

1. Il provider verrà considerato come un'estensione se viene posizionato nella standard extension directory, ovvero la cartella ext. Abbiamo inserito il file .jar del provider al seguente path:

```
C:\Program Files (x86)\Java\jre1.8.0_111\lib\ext\<provider>.jar
```

2. Il secondo passo è quello di registrare tale provider. Può essere fatto in due differenti modi: statico e dinamico. Ai fini del progetto abbiamo effettuato una registrazione statica del provider. Per fare ciò, è stato sufficiente modificare il file java.security, situato nel seguente path:

```
C:\Program Files (x86)\Java\jre1.8.0_111\lib\security\java.security
```

   Ed inserire la seguente linea nella lista dei provider:

```
security.provider.N=org.bouncycastle.jce.provider.BouncyCastleProvider
```

   La N dovrà essere sostituita col numero di priorità che interessa.

A.2 Truststore e keystore

- **Chiavi provate**: per generare una chiave RSA, si utilizza il comando openssl genrsa, la cui sintassi è la seguente:

```
openssl genrsa [-out nome_file] [-passout arg] [-des] [-des3] [-idea]
[-f4] [-3] [-rand file(s)] [numbits]
```

È importante sottolineare che, per default, il comando genrsa crea una chiave RSA utilizzando come esponente pubblico il valore $e = 655537$. È possibile scegliere il valore dell'esponente $e = 3$ utilizzando l'opzione -3, poiché l'opzione $-f4$ indica sempre l'esponente di default. È possibile, inoltre, creare una chiave privata con un algoritmo di cifratura a blocchi che sia DES, 3DES oppure IDEA. Se non viene specificata nessuna di queste 3 opzioni allora non viene utilizzata nessuna cifratura. Per generare le chiavi di interesse, ai fini del seguente documento, sono stati utilizzati i seguenti comandi:

```
openssl genrsa -des3 -out client_android_key.pem 2048
openssl genrsa -des3 -out end_user_key.pem 2048
openssl genrsa -des3 -out server_key.pem 2048
```

Se si vogliono processare le chiavi RSA create, può essere utilizzato il comando openssl rsa, la cui sintassi è la seguente:

```
openssl rsa [-inform] [-outform |PEM|NET|DER] [-in filename]
[-passin arg] [-out filename] [passout arg] [-sgckey] [-des] [-des3]
[-idea] [-text] [-noout] [-modulus] [-check] [-pubin] [-pubout]
```

Per esempio per mostrate in chiaro le chiavi cifrate, create precedentemente, mostrando i valori della chiave pubblica e privata, nonché i fattori ed altri coefficienti, è possibile eseguire il seguente comando per ognuna di esse:

```
openssl rsa -in chiave_privata.pem -text
```

- **Certificati self-signed**: generalmente, il certificato che ha il compito di garantire l'identità di qualcuno, deve essere rilasciato da un CA. Questa, per rilasciare il certificato, deve ricevere un documento intermedio, definito come una certificate request. A tal fine, per poter ottenere un certificato o una richiesta di certificato, può essere utilizzato il comando openssl req, la cui sintassi è la seguente:

```
openssl req [-inform PEM|DER] [-outform PEM|DER] [-in filename]
[-passin arg] [-out filename] [-passout arg] [-text] [-pubkey] [-noout]
[-verify] [-modulus] [-new] [-rand file(s)] [-newkey rsa:bits]
[-newkey alg:file] [-nodes] [-key filename] [-keyform PEM|DER]
[-keyout filename] [-keygen_engine id] [-[digest]] [-config filename]
[-subj arg] [-multivalue-rdn] [-x509] [-days n] [-set_serial n]
[-asn1-kludge] [-no-asn1-kludge] [-newhdr] [-extensions section]
[-reqexts section] [-utf8] [-nameopt] [-reqopt] [-subject] [-subj arg]
[-batch] [-verbose] [-engine id]
```

La chiave pubblica che viene inserita, si ottiene dalla chiave privata generata col comando *openssl genrsa*, mentre gli altri campi vengono inseriti dinamicamente o dal software o dall'utente. Per generare i certificati self-signed, a partire dalle chiave private generate al passo precedente, è possibile utilizzare i seguenti comandi:

```
openssl req -new -x509 -key client_android_key.pem -out client_android.pem -days
openssl req -new -x509 -key end_user_key.pem -out end_user.pem -days 365
openssl req -new -x509 -key server_key.pem -out server.pem -days 365
```

È importante sottolineare che queste chiavi siano state generate tramite i certificati dello standard x509 e siano valide per 365 giorni. I certificati con estensione .PEM non sono leggibili in modo diretto.

- **TrustStore**: per poter utilizzare le chiavi e i certificati generati nei passi precedenti, si devono importare entrambi nei keystore. Per fare ciò, può essere utilizzato il seguente comando:

```
-importcert {-alias alias} {-file cert_file} [-keypass keypass]
{-noprompt} {-trustcacerts} {-storetype storetype} {-keystore keystore}
[-storepass storepass] {-providerName provider_name}
{-providerClass provider_class_name {-providerArg provider_arg}} {-v}
{-protected} {-Jjavaoption}
```

Per poter proseguire, il primo passo da effettuare è quello di spostarsi nella cartella in cui è presente l'eseguibile keytool.exe. Fatto questo, per fare in modo che il client si fidi del certificato del server, può essere eseguito il comando:

```
keytool -importcert -trustcacerts
-keystore client_android_truststore.bks -storetype bks -storepass
<truststore_password> -file <path-to-file>\server.pem
-provider org.bouncycastle.jce.provider.BouncyCastleProvider
-providerpath <path_to_bcprov_jar>
```

Dove:

1. *truststore-password*: è la password.
2. *path-to-file*: è il path fino alla cartella in cui si trova il file desiderato.
3. *path-to-bcprov-jar*: il path in cui si trova il .jar del provider.

Analoghe operazione vanno fatte sia per l'entità end user che per il server. Nel primo caso basti eseguire il comando:

```
keytool -importcert -trustcacerts -keystore end_user_truststore.jks
-storetype jks -storepass <end_user_truststore_password>
-file <path-to-file>\ server.pem
```

Mentre, per il server:

```
keytool -importcert -trustcacerts -keystore servertruststore.jks
-storetype jks -storepass <server_truststore_password>
-file <path-to-file>\client_android.pem keytool
-importcert -trustcacerts -keystore servertruststore.jks
-storetype jks -storepass <server_truststore_password>
-file <path-to-file>\end_user.pem
```

- **Combinare chiavi private e certificati**: l'applicazione keytool di Java presenta un problema. Infatti, non permette di importare una chiave privata esistente in un keystore.Per poter ovviare a questo problema è possibile combinare la chiave privata con il certificato in un file .PKCS12 e importare il tutto nel keystore. Un comando che può essere utilizzato per convertire certificati o chiavi dal formato .PEM ad un formato .PKCS12 è il seguente:

```
openssl pkcs12 [-export] [-chain] [-inkey filename][-certfile filename]
[-name name] [-caname name] [-in filename] [-out filename] [-noout]
[-nomacver] [-nocerts] [-clcerts] [-cacerts] [-nokeys] [-info]
[-des | -des3 | -idea | -aes128 | -aes192 | -aes256 | -camellia128 |
-camellia192 | -camellia256 | -nodes] [-noiter] [-maciter | -nomaciter|
-nomac] [-twopass] [-descert] [-certpbe cipher] [-keypbe cipher]
[-macalg digest] [-keyex] [-keysig] [-password arg] [-passin arg]
[-passout arg] [-rand file(s)] [-CAfile file] [-CApath dir][-CSP name]}
```

Generalmente, il comando pkcs12 ha una funzione più ampia di quella di una semplice conversione di formato. Di fatti, fornisce la possibilità di trattare in generale i certificati in formato PKCS-12. Comunque, ai fini del seguente documento, ciò che interessa è eseguire i seguenti comandi:

```
openssl pkcs12 -export -inkey client_android_key.pem -in client_android.pem -out
openssl pkcs12 -export -inkey end_user_key.pem -in end_user.pem -out end_user.p12
openssl pkcs12 -export -inkey server_key.pem -in server.pem -out server.p12
```

L'ultimo passo consiste nell'importare i file appena generati, all'interno del keystore. Il comando che permette di effettuare un'operazione del genere è il seguente:

```
-importkeystore -srckeystore srckeystore -destkeystore destkeystore
{-srcstoretype srcstoretype} {-deststoretype deststoretype}
[-srcstorepass srcstorepass] [-deststorepass deststorepass]
{-srcprotected} {-destprotected} {-srcalias srcalias {-destalias alias}
[-srckeypass srckeypass] [-destkeypass destkeypass] } {-noprompt}
{-srcProviderName src_provider_name} {-destProviderName dest_provider}
{-providerClass provider_class_name {-providerArg provider_arg}} {-v}
{-protected} {-Jjavaoption}
```

Quindi, basta eseguire:

```
keytool -importkeystore -srckeystore <path-to-file>\client_android.p12
-srcstoretype pkcs12 -destkeystore client_android_truststore.bks
-deststoretype bks
-provider org.bouncycastle.jce.provider.BouncyCastleProvider
-providerpath <path_to_bcprov_jar>

keytool -importkeystore -srckeystore <path-to-file>\end_user.p12
-srcstoretype pkcs12 -destkeystore end_user_truststore.jks
-deststoretype jks

keytool -importkeystore -srckeystore <path-to-file>\server.p12
-srcstoretype pkcs12 -destkeystore server.jks -deststoretype jks
```

Arrivati a questo punto, sono stati generati tutti i file necessari per la mutua autenticazione per il protocollo SSL/TSL. Ogni file dovrà essere posizionato nel path corretto:

- Smartphone: client-android-truststore.bks e client-android.bks.
- End User: end-user-truststore.jks e end-user.jks.
- Server: server-truststore.jks e server.jks.

A.3 Configurazione Tomcat

Per configurare Tomcat correttamente, basta andare nel file "path/to/server.xml" e modificare l'oggetto Connector nel seguente modo:

```
<Connector
clientAuth="false" port="8443"
enableLookups="true" disableUploadTimeout="true"
acceptCount="100" maxThreads="200"
scheme="https" secure="true" SSLEnabled="true"
keystoreFile="path/to/server.jks"
keystoreType="JKS" keystorePass="********"
truststoreFile="path/to/server_truststore.jks"
truststoreType="JKS" truststorePass="********"
SSLVerifyClient="require" SSLEngine="on" SSLVerifyDepth="2"
sslProtocol="TLS" />
```

Particolare attenzione deve essere posta all'attributo clientAuth. Esso può assumere due distinti valori true e false. Nel primo caso forza il server affinché richieda un certificato da parte del client, effettuando così una mutua autenticazione. Nel secondo caso il server non chiederà nulla. Il secondo valore solitamente viene usato a scopo di debug.

Elenco delle figure

Elenco delle tabelle

Bibliografia

[1] Gastone Ciuti, Leonardo Ricotti, Arianna Menciassi, and Paolo Dario. Mems sensor technologies for human centred applications in healthcare, physical activities, safety and environmental sensing: a review on research activities in italy. *Sensors*, 15(3):6441–6468, 2015.

[2] Activity recognition api. https://developers.google.com/android/reference/com/google/android/gms/location/ActivityRecognitionApi/, November 2016.

[3] Wikibooks. Intelligenza artificiale/apprendimento automatico — wikibooks, pensa liberamente, impara liberamente, 2007.

[4] Qinghua Li and Guohong Cao. Privacy-preserving participatory sensing. *IEEE Communications Magazine*, 53(8):68–74, 2015.

[5] Raul Montoliu, Jan Blom, and Daniel Gatica-Perez. Discovering places of interest in everyday life from smartphone data. *Multimedia tools and applications*, pages 1–29, 2013.

[6] Nidhi Kalra, Gunjan Chugh, and Divya Bansal. Analyzing driving and road events via smartphone. *International Journal of Computer Applications*, 98(12):5–9, 2014.

[7] Matteo Ciman. Smartphones as ubiquitous devices for behavior analysis and better lifestyle promotion. 2016.

[8] Dror Ben-Zeev, Emily A Scherer, Rui Wang, Haiyi Xie, and Andrew T Campbell. Next-generation psychiatric assessment: Using smartphone sensors to monitor behavior and mental health. *Psychiatric rehabilitation journal*, 38(3):218, 2015.

[9] Oliver Meynberg, Florian Hillen, and Bernhard Höfle. Navigation in dense human crowds using smartphone trajectories and optical aerial imagery. *The ISPRS Tracking and Imaging Challenge 2014*, pages 1–4, 2014.

[10] Martin Wirz, Tobias Franke, Daniel Roggen, Eve Mitleton-Kelly, Paul Lukowicz, and Gerhard Tröster. Probing crowd density through smartphones in city-scale mass gatherings. *EPJ Data Science*, 2(1):5, 2013.

[11] Jake K Aggarwal and Lu Xia. Human activity recognition from 3d data: A review. *Pattern Recognition Letters*, 48:70–80, 2014.

[12] Salvatore Gaglio, Giuseppe Lo Re, and Marco Morana. Human activity recognition process using 3-d posture data. *IEEE Transactions on Human-Machine Systems*, 45(5):586–597, 2015.

[13] Pietro Cottone, Gabriele Maida, and Marco Morana. User activity recognition via kinect in an ambient intelligence scenario. *IERI Procedia*, 7:49–54, 2014.

[14] Alessandro Manzi, Paolo Dario, and Filippo Cavallo. A human activity recognition system based on dynamic clustering of skeleton data. *Sensors*, 17(5):1100, 2017.

[15] Nishkam Ravi, Nikhil Dandekar, Preetham Mysore, and Michael L Littman. Activity recognition from accelerometer data. In *Aaai*, volume 5, pages 1541–1546, 2005.

[16] Ling Bao and Stephen S Intille. Activity recognition from user-annotated acceleration data. In *International Conference on Pervasive Computing*, pages 1–17. Springer, 2004.

[17] Oresti Banos, Miguel Damas, Hector Pomares, Alberto Prieto, and Ignacio Rojas. Daily living activity recognition based on statistical feature quality group selection. *Expert Systems with Applications*, 39(9):8013–8021, 2012.

[18] Amarnag Subramanya, Alvin Raj, Jeff A Bilmes, and Dieter Fox. Recognizing activities and spatial context using wearable sensors. *arXiv preprint arXiv:1206.6869*, 2012.

[19] Jonathan Lester, Tanzeem Choudhury, and Gaetano Borriello. A practical approach to recognizing physical activities. In *International Conference on Pervasive Computing*, pages 1–16. Springer, 2006.

[20] Božidara Cvetković, Vito Janko, Alfonso E Romero, Özgür Kafalı, Kostas Stathis, and Mitja Luštrek. Activity recognition for diabetic patients using a smartphone. *Journal of medical systems*, 40(12):256, 2016.

[21] Cesar Torres-Huitzil and Andres Alvarez-Landero. Accelerometer-based human activity recognition in smartphones for healthcare services. In *Mobile Health*, pages 147–169. Springer, 2015.

[22] S Hwang, M Ryu, Y Yang, and N Lee. Fall detection with three-axis accelerometer and magnetometer in a smartphone. In *Proceedings of the International Conference on Computer Science and Technology, Jeju, Korea*, pages 25–27, 2012.

[23] Yongjin Kwon, Kyuchang Kang, and Changseok Bae. Unsupervised learning for human activity recognition using smartphone sensors. *Expert Systems with Applications*, 41(14):6067–6074, 2014.

[24] Giuseppe Cardone, Andrea Cirri, Antonio Corradi, Luca Foschini, and Dario Maio. Msf: An efficient mobile phone sensing framework. *International Journal of Distributed Sensor Networks*, 2013, 2013.

[25] Giuseppe Cardone, Andrea Cirri, Antonio Corradi, Luca Foschini, Raffaele Ianniello, and Rebecca Montanari. Crowdsensing in urban areas for city-scale mass gathering management: Geofencing and activity recognition. *IEEE Sensors Journal*, 14(12):4185–4195, 2014.

[26] Giuseppe Cardone, Antonio Corradi, Luca Foschini, and Raffaele Ianniello. Participact: A large-scale crowdsensing platform. *IEEE Transactions on Emerging Topics in Computing*, 4(1):21–32, 2016.

[27] Xiping Hu, Terry HS Chu, Henry CB Chan, and Victor CM Leung. Vita: A crowdsensing-oriented mobile cyber-physical system. *IEEE Transactions on Emerging Topics in Computing*, 1(1):148–165, 2013.

[28] Zheng Xu, Hui Zhang, Vijayan Sugumaran, Kim-Kwang Raymond Choo, Lin Mei, and Yiwei Zhu. Participatory sensing-based semantic and spatial analysis of urban emergency events using mobile social media. *EURASIP Journal on Wireless Communications and Networking*, 2016(1):44, 2016.

[29] Delphine Christin. Privacy in mobile participatory sensing: Current trends and future challenges. *Journal of Systems and Software*, 116:57–68, 2016.

[30] Alessandra De Paola, Pierluca Ferraro, Salvatore Gaglio, and Giuseppe Lo Re. *Context-Awareness for Multi-sensor Data Fusion in Smart Environments*, pages 377–391. Springer International Publishing, Cham, 2016.

[31] Alessandra De Paola and Marco Morana. *Bio-inspired Sensory Data Aggregation*, pages 367–368. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[32] Alessandra De Paola, Marco La Cascia, Giuseppe Lo Re, Marco Morana, and Marco Ortolani. User detection through multi-sensor fusion in an ami scenario. In *2012 15th International Conference on Information Fusion*, pages 2502–2509, July 2012.

[33] Peter Hall, Byeong U Park, and Richard J Samworth. Choice of neighbor order in nearest-neighbor classification. *The Annals of Statistics*, pages 2135–2152, 2008.

[34] CryptoSense. Parameter choice for pbkdf2. https://cryptosense.com/parameter-choice-for-pbkdf2/.

[35] Alhamza Alalousi, Rozmie Razif, Mosleh AbuAlhaj, Mohammed Anbar, and Shahrul Nizam. A preliminary performance evaluation of k-means, knn and em unsupervised machine learning methods for network flow classification. *International Journal of Electrical and Computer Engineering*, 6(2):778, 2016.

[36] Diaa Salama Abd Elminaam, Hatem Mohamed Abdual-Kader, and Mohiy Mohamed Hadhoud. Evaluating the performance of symmetric encryption algorithms. *IJ Network Security*, 10(3):216–222, 2010.

[37] SR Kodituwakku and US Amarasinghe. Comparison of lossless data compression algorithms for text data. *Indian journal of computer science and engineering*, 1(4):416–425, 2010.

[38] Vincenzo Agate, Alessandra De Paola, Giuseppe Lo Re, and Marco Morana. *A Simulation Framework for Evaluating Distributed Reputation Management Systems*, pages 247–254. Springer International Publishing, Cham, 2016.

[39] Vincenzo Agate, Alessandra De Paola, Salvatore Gaglio, Giuseppe Lo Re, and Marco Morana. A framework for parallel assessment of reputation management systems. In *Proceedings of the 17th International Conference on Computer Systems and Technologies 2016*, CompSysTech '16, pages 121–128, New York, NY, USA, 2016. ACM.