Alma Mater Studiorum · Università di Bologna

**SCUOLA DI SCIENZE**

**Corso di Laurea in Informatica per il Management**

# Riconoscimento automatico di attività attraverso i Google Play Services: una valutazione sperimentale

**Relatore:**
**Chiar.mo Prof.**
**Marco Di Felice**

**Presentata da:**
**Raffaele Grezzi**

**Sessione I**
**Anno Accademico 2013/2014**

*Ai miei nonni Michele, Raffaele, Anna e Carmela, a tutta la mia famiglia e a tutti coloro che hanno creduto in me.*

# Introduzione

Il riconoscimento automatico di attività, in inglese activity recognition, è un ambito di ricerca della scienza informatica che ha lo scopo di riuscire a determinare l'attività o il tipo di spostamento di un utente sulla base di dispositivi in grado di fornire dati di sensori da analizzare in modo computazionale. Questi sensori possono essere di diversi tipi come giroscopi, accelerometri, ricevitori GPS, cardiofrequenzimetri, microfoni, ecc. Le finalità del riconoscimento automatico di attività possono andare dall'assistenza a pazienti disabili o anziani all'ausilio alla guida di un mezzo di trasporto, dal fornire informazioni in tempo reale quando si è in viaggio alla gestione di servizi basati sulla posizione degli utenti, dalla promozione di prodotti personalizzata in base alle abitudini degli utenti alla gestione automatizzata di dispositivi elettronici come televisori, computer, elettrodomestici o semplici lampadine. Il generale, qualcunque cosa abbia a che fare con l'interazione uomo-macchina può essere un buon campo di applicazione per questo ambito. Il recente avvento degli smartphones nel mercato globale e la prepotente crescita di questo mercato ha favorito in qualche modo l'utilizzo di questi dispositivi come mezzi di ricerca per il riconoscimento automatico di attività. La causa di ciò si può ritrovare nel fatto che questi dispositivi, per motivi spesso lontani dal concetto di telefono cellulare, sono dotati di ogni tipo di sensori elettronici in grado di misurare caratteristiche ambientali intorno a loro, ma anche caratteristiche soggettive relative agli utenti. Inoltre essi, a differenza di molti altri dispositivi elettronici, sono dotati di capacità di calcolo dovuta alla loro architettura disegnata per essere piccoli computer in

miniatura. Gli smartphones infatti, sono dotati di processore, RAM e memoria secondaria, oltre che di tutti i sensori elencati sopra, il che rende possibile l'esecuzione di sistemi operativi scritti ad hoc per gestire dispositivi simili. Inoltre con il passare del tempo essi sono diventati anche molto potenti dal punto di vista computazionale, in modo da renderli multitasking come un personal computer ma con la comodità di un telefono cellulare. Quindi la nascita delle app, programmi per dispositivi mobili in grado ormai di competere con programmi per computer desktop, ha permeso che si accentuasse l'interesse per l'interazione uomo-macchina da parte della scienza. Infatti questi dispositivi vengono portati dagli utenti dappertutto, in pratica sono sempre con noi, quindi possono diventare una sorta di assistente artificiale, in grado di svolgere miriadi di funzioni per noi. Oltre a queste caratteristiche, gli smartphone possono offrire anche una connessione ad internet mobile, in modo da essere sempre connessi in rete con qualsiasi servizio di nostro interesse. Proprio queste caratteristiche hanno portato i ricercatori di molti istituti di ricerca del mondo ad avviare ricerce in questo ambito utilizzando gli smartphones come piccoli catturatori di informazioni su cui basare poi tutta la ricerca. Ma oltre agli istituti di ricerca, anche varie aziende informatiche o di servizi informatici hanno iniziato a studiare il riconoscimento automatico di attività per essere in grado di fornire ai loro utenti servizi sempre più incentrati sull'automazione, che siano in grado di facilitare le dinamiche quotidiane di una persona attraverso l'intelligenza artificiale. È questo il caso di Google che ha recentemente ha pubblicato, all'interno della nuova versione del suo sistema operativo Android, la 4.4 (KitKat), una libreria software in grado di restituire ad un programmatore Android l'attività in corso da parte dell'utente sfruttando i dati catturati dai sensori di GPS e accelerometro di nome Activity Recognition, che sarà studiata, sulla base dei risultati forniti su un certo numero di prove, in questo elaborato.

# Indice

# Elenco delle figure

# Elenco delle tabelle

# Capitolo 1

# Stato dell'arte

Il riconoscimento automatico di attività umane è un ambito dell'informatica con molti campi di applicazione reali, per questo è al centro degli studi di molti istituti di ricerca in tutto il mondo. Gli studi [1, 2, 3, 4, 5] cercano di risolvere il problema del riconoscimento di attività utilizzando un telefono cellulare.

## 1.1 Ricerche basate su dati GPS

In [1] i ricercatori della University of Illinois di Chicago hanno affrontato questo problema ipotizzando di affiancare all'utilizzo dei dati provenienti dai sensori di uno smartphone una serie di dati riguardanti la mobilità su trasporto pubblico del bacino di Chicago, luogo in cui sono stati effettuati gli esperimenti. I dati sul trasporto pubblico riguardano sia il trasporto su gomma che su rotaia. Questo approccio consente, tramite l'utilizzo dei dati GPS, di confrontare la posizione dell'utente con la posizione in tempo reale dei mezzi pubblici di Chicago, in modo da riuscire a distinguere, senza il supporto di dati di altri sensori come l'accelerometro e il giroscopio, se l'utente si sta spostando con un mezzo pubblico oppure in un altro modo. Nel caso che l'utente si sposti effettivamente con un mezzo pubblico, l'applicazione è in grado, conoscendo i dati dell'azienda di trasporti, di indicare il numero di

autobus o di treno con il quale si sta viaggiando, oltre ad altre informazioni correlate con il viaggio/spostamento. Questo tipo di integrazione di dati prevede che vi siano implementazioni di Open Data[6] da parte di aziende o enti, soprattutto pubblici, che permettano di condividere i dati in tempo reale con l'utenza che li richiede. In questa ricerca si utilizza un'architettura implementativa centralizzata per arrivare ad essere in grado di riconoscere, attraverso l'analisi e l'integrazione dei dati, l'attività dell'utente. L'architettura centralizzata prevede un server centrale che, conoscendo i dati da integrare e ricevendo ad intervalli regolari le informazioni relative al GPS di un utente, calcola a monte (sul server stesso) l'attività umana più probabile, per poi inviare i risultati indietro allo smartphone che aveva effettuato la richiesta. Questa metodologia centralizzata ha fatto nascere interrogativi sulla protezione della privacy degli utenti, in quanto i loro telefoni cellulari inviano informazioni sulla posizione corrente ad un servizio web che deve essere in grado di proteggere tali informazioni da attacchi esterni o intrusioni. Ad ogni modo, questo studio dimostra che è possibile migliorare la precisione del riconoscimento automatico di attività umane anche diminuendo il numero di sensori interrogati su uno smartphone, affiancando però i dati del solo GPS ad una grossa mole di dati esterni riguardanti il trasporto pubblico. L'applicazione di prova implementata in questo studio è in grado di riconoscere attività umane tra le seguenti categorie: camminare, in autobus, guidare, in treno, fermo, in bicicletta.

## 1.2 Ricerche basate sui dati di accelerometro e giroscopio

In [2], invece, i ricercatori del dipartimento di Informatica dell'università di Bologna propongono un approccio alternativo al problema. In questo studio, la ricerca è focalizzata sul riconoscimento del tipo di movimento che l'utente effettua in un dato momento, ovvero come o con che mezzo si muove e non sul riconoscimento dell'attività in corso dell'utente. Le due cose so-

no molto simili dal punto di vista matematico/fisico, in quanto entrambi i riconoscimenti si basano sui dati dati provenienti dai sensori di uno smartphone. Come problema, invece, si può dire che il riconoscimento dell'attività dell'utente inglobi anche il riconoscimento del tipo di movimento dell'utente, mentre non è vero il contrario. Ad esempio, il riconoscimento dell'attività "viaggio verso casa" effettuata in automobile racchiude anche il riconoscimento del tipo di movimento "in automobile". La tesi che si vuole dimostrare è che l'utilizzo di sensori in più migliori la precisione del riconoscimento rispetto al caso in cui si considerino meno dati di sensori. Il primo passo per affrontare il problema è stato quello di raccogliere dati provenienti dai sensori durante lo svolgimento dal parte dell'utente di una attività tra camminare, muoversi in automobile, muoversi in treno. Questo tipo di raccolta dati è molto simile a quello utilizzato in questo studio per testare le capacità della libreria di Activity Recognition di Google. Infatti essi hanno realizzato una piccola applicazione per la piattaforma Android che semplicemente raccoglie dati ad un tasso di 10 Mhz e quindi salva questi dati su un file di log. I sensori di cui sono stati raccolti i dati sono due: accelerometro e giroscopio anche se i ricercatori affermano di aver pianificato di estendere l'applicazione con l'uso di altri sensori in futuro. Per ogni tipo di attività e di sensore, è stato creato un data set di circa 72000 elementi contenti ognuno: il timestamp di riferimento, i dati dei 3 assi (x, y e z) del sensore. I data set sono stati creati in modo eterogeneo, cioè i dati sono stati raccolti da diverse persone su mezzi diversi. Inoltre la raccolta dei dati è stata effettuata senza imporre all'utente di non usare il dispositivo o di lasciarlo in una posizione fissa per facilitare l'ottenimento dei dati. Questo rende più reale l'esperimento ed è reso possibile dall'utilizzo di una metrica chiamata magnitude, la quale riflette il modulo del vettore di 3 dati senza rispecchiare l'orientamento del sensore. Grazie a questa metrica, si ottiene un unico valore da un vettore di 3 elementi generato dal sensore. Una volta raccolti i data set, vengono estratte le features utili per creare i pattern di riconoscimento. Per ogni data set quindi, vengono create sequenze di dati frutto della divisione dei data

set in sessioni di 10 secondi. Per ogni sessione vengono estratti 4 valori per le features che sono: min(s,k), max(s,k), avg(s,k)e std(s,k) che rappresentano rispettivamente il minimi, il massimo, il valore medio e la deviazione standard delle magnitude dei sensori nella sessione k. Una volta estratte le features, i nuovi dati vengono dati in pasto ad un classificatore che, seguendo algoritmi di machine learning dati, cerca di riconoscere all'interno dei dati i pattern necessari al riconoscimento automatico di attività. Gli algoritmi di machine learning utilizzati in questo studio sono: Random Forest, SVM e NaiveBayes. I risultati mostrano che l'algoritmo più preciso in grado di capire l'attività dell'utente partendo dai dati, per ogni classificatore è il Random Forest con il 98,08% di precisione media. Seguono NaiveBayes (84,11%) e il SVM (82,50%). I test sui classificatori sono stati poi ripetuti tenendo conto non più di entrambi i sensori insieme, ma di uno solo per volta, il che ha prodotto risultati simili a quelli sperati in quanto si vede che l'utilizzo di un solo sensore abbassa la precisione media di tutti e 3 gli algoritmi: considerando solo l'accelerometro si ottiene una precisione del 93,49%, solo il giroscopio del 91,94% a fronte di una precisione media combinando i 2 sensori del 97,71%. Questi risultati dimostrano la tesi iniziale che affermava che l'utilizzo di più dati dei sensori in combinazione tra loro aumenta la precisione dei risultati dei test sui classificatori utilizzando algoritmi di machine learning.

## 1.3 Ricerche basate sui dati di GPS e accelerometro

In [3] possiamo osservare una interessante ricerca internazionale. I ricercatori di questa alleanza di ricerca puntano a studiare il problema del riconoscimento automatico di attività utilizzando, come altri visti precedentemente, i dati catturati dai sensori di uno smartphone con il fine di studiare la rete di trasporti di Singapore per conto della Land Transportation Authority (LTA). Questa azienda aveva tentato un approccio basato su sondaggi per testare il sistema dei trasporti che gestisce, in modo da ricevere feed-

backs in base alle risposte ai sondaggi. Il team di ricerca, per ovviare agli errori dovuti a questo tipo di approccio "tradizionale" ha deciso di creare un sistema di survey basati sui dati di smartphone appositamente programmati, per fare in modo che la LTA possa utilizzarlo in futuro per capire se il funzionamento dei trasporti funziona a dovere o meno. Anche il sistema messo a punto da questa equipe di ricerca si basa sui dati di due sensori che sono il GPS (oppure la posizione derivata dalla rete cellulare ) e l'accelerometro. Il loro algoritmo è in grado di distinguere tra diversi tipi di trasporto, in particolare è in grado di distinguere la 3 diverse categorie di trasporto motorizzato come bus, MRT (Mass Rapid Transport), taxi. Queste categorie sono state scelte proprio perché lo studio è finalizzato alla ricezione di feedback da parte dell'azienda di trasporti pubblici. L'unica differenza con [1] nel metodo di raccolta dei dati consiste nel fatto che in questo studio si è scelto di utilizzare, in situazioni in cui non sia possibile ricevere informazioni dal GPS per disturbi del segnale dovuti a palazzi alti e luoghi chiusi, le caratteristiche della rete cellulare per determinare la posizione corrente dell'utente ed atri parametri anziché utilizzare il GPS. Questo causa in genere una minore precisione nel calcolo della posizione rispetto all'approccio basato su GPS oltre ad una quasi impossibilità nel determinare la posizione in zone in cui la rete cellulare è in roaming. Inoltre questo algoritmo è in grado di distinguere i momenti e i punti di stop nel trasporto e capire se lo stop sia dovuto alle caratteristiche del trasporto pubblico (es fermate di autobus o treni) oppure no. In pratica questo algoritmo raccoglie i tracciati della posizione dell'utente mentre si muove, calcolando i punti di fermata analizzando il tempo che l'utente impiega ad arivare da un punto ad un altro. Ogni volta che il tragitto tra un punto i ed uno j è maggiore di un certo tempo T medio, l'algoritmo considera che nell'andare da i a j vi è stato un punto di stop. Una volta calcolati i punti di stop, l'algoritmo procede con il riconoscimento di tipo di trasporto tra i vari punti di stop. In questo modo è in grado di capire se le fermate sono dovute al trasporto pubblico oppure ad altri fattori ambientali (es. code o parcheggio). Il riconoscimento del tipo di trasporto avviene quin-

di combinando i dati GPS precedentemente analizzati per lo stop con i dati dell'accelerometro. Per il calcolo del tipo di trasporto vengono usate le features stdev, maxV e avgV che rappresentano, rispettivamente, la deviazione standard della magnitude calcolata sui dati dell'accelerometro come in [2], la velocità massima calcolata nello spostamento e la velocità media dello stesso spostamento. Ogni finestra di tempo nella quale si calcolano questi valori si riferisce ad uno spostamento tra un punto di stop e il successivo. Una volta trovato il tipo di trasporto tra tutti i punti, vengono scartati quelli non interessanti, ovvero quelli che non riguardano il trasporto pubblico secondo il calcolo dell'algoritmo. Ottenuti tutti i punti di stop interessanti, questi vengono mostrati in una mappa online per illustra i risultati del calcolo del tipo di trasporto da parte dei dispositivi (smartphone Android ed iPhone) che sono impegnati nel test. L'analisi dei dati per l'azienda di trasporti e i risultati, sia del calcolo sia dell'analisi, possono essere visualizzati e modificati apertamente sul portale online creato al team di ricerca per questi scopi. In [4] l'approccio utilizzato alla risoluzione di questo problema è molto simile ai precedenti in quanto si basa sul"utilizzo di dati provenienti da due sensori principali (accelerometro e GPS) combinati con altri dati di supporto relativi a WiFi e GSM. Anche in questo caso è stata utilizzata la magnitude calcolata sui valori dei 3 assi dell'accelerometro per eliminare il problema dell'orientamento del dispositivo a momento della raccolta dati. In questo studio viene enfatizzata l'importanza dell'intervallo di aggiornamento dei dati durante la raccolta. La progettazione di questa ricerca ha previsto intervalli di aggiornamento di 1 secondo indicati come scelta migliore basandosi su dati sperimentali. La scelta migliore dell'intervallo di aggiornamento è l'intervallo con la maggiore accuracy del classificatore che si intende utilizzare. Per testare il classificatore una volta raccolti i dati è stato utilizzato l'ambiente di simulazione Weka[8], software usato per i test anche per questo elaborato. La raccolta dei dati è stata effettuata da 6 volontari su smartphone Nokia n95 equipaggiati con sistema operativo Symbian S60. La posizione e l'orientamento dei dispositivi di ogni utente sono stati volutamente resi diversi da

un utente all'altro in modo da studiare il problema del riconoscimento di attività in un ambiente quanto più reale possibile, ovvero non forzando la precisione dei dati utilizzando ad esempio un orientamento del dispositivo noto per la precisione totale del classificatore. I risultati, basati su 6 classificatori diversi (Decision Tree, K-Means Clustering, Naive Bayes, Nearest Neighbor, Support Vector System, modello nascosto di Markov continuo), tutti comparati con un modello nascosto di Markov discreto, indicano come, a seconda del classificatore, si abbiano risultati abbastanza contrastanti per quel che riguarda la precisione del riconoscimento. La maggiore precisione viene espressa per quasi tutte le classi dal classificatore Decision Tree, i cui valori di precisione più si avvicinano al classificatore scelto per la comparazione, il modello nascosto di Markov discreto. Inoltre in questo studio è emerso come la posizione del dispositivo sul corpo dell'utente al momento della raccolta dei dati non influisca in modo significativo sulla precisione del riconoscimento di attività, anzi in questo esperimento si è ottenuta un precisione media, utilizzando svariate posizioni contemporaneamente, superiore dell'1,4% rispetto alla precisione media ottenuta considerando una sola posizione fissa per tutti gli utenti. Questo risultato aumenta ulteriormente i margini di applicabilità di algoritmi di riconoscimento automatico di attività in applicazioni reali utili a qualche scopo collegato con questo ambito. In [5], infine, è stata realizzata una ricerca presso l'Austrian Institute of Technology riguardante il riconoscimento automatico di attività focalizzato sulla precisione e la varietà di attività da riconoscere. Lo scopo di questo esperimento è quello di riuscire a riconoscere le modalità di trasporto dell'utente non solo per macro aree (es. generico veicolo motorizzato), bensì distinguendo tra molte tipologie di trasporto realmente utilizzate tutti i giorni dagli utenti. Per ottenere questo risultato si è utilizzato un approccio basato sui sensori di accelerometro e GPS, includendo, in momenti di perdita di segnale da parte del GPS, i dati relativi alla posizione ottenuti dalle celle della rete mobile. In questo modo si è creato un robusto sistema in grado di prescindere da un unico sensore e aumentare la precisione in situazioni in cui altri metodi sareb-

bero meno precisi. Per i dati relativi all'accelerometro è stata estratta anche in questo caso la magnitude in modo da non considerare l'orientamento del dispositivo al momento della raccolta dei dati. Inotre sono state utilizzate altre 7 features estratte dai dati grezzi catturati dai sensori: indici statistici su velocità, accelerazione, decelerazione e velocità angolare, considerando 3 percentili (5°, 50° e 95°), deviazione standard del segnale dell'accelerometro, spettro di potenza per le frequenze di aggiornamento. L'utilizzo di questo elevato numero di features ha permesso il riconoscimento delle attività non più per macro aree ma entrando nello specifico per ogni categoria. Le attività che si è in grado di riconoscere utilizzando questo approccio sono: mezzi di trasporto tra bus, automobile, bicicletta, tram, treno, metropolitana e motocicletta, camminare. I risultati dei test sono stati ottenuti considerando un insieme di 100 classificatori applicati su sottospazi ottenuti randomicamente dagli spazi delle features utilizzate. I risultati mostrano una precisione media per tutte le attività del 76,375% la qual si dimostra superiore ad altri approcci preceenti di circa l'8%. Questi risultati sono utili soprattutto in virtù del fatto che questa ricerca riesce a distinguere molto bene tra i diversi tipi di trasporto motorizzato a differenza di molti altri approcci che raggruppano tutte queste attività in un'unica categoria come "mezzo di trasporto motorizzato" o simili.

# Capitolo 2

# Metodologia

La metodologia utilizzata per creare un set di dati sperimentali e studiarli confrontando i risultati ottenuti con altre implementazioni di algoritmi di machine learning prevede di utilizzare una serie di strumenti, hardware e software, che hanno permesso di seguire tutte le fasi dello studio, dalla raccolta dei dati, passando per il raffinamento dei dati stessi, fino alla comparazione finale dell'algoritmo che i Google Play Services utilizzano per l'activity recognition con altri algoritmi, simili ma aperti, contenuti anche all'interno del simulatore Weka. L'implementazione dell'algoritmo da parte di Google non è aperta alla visione e alla modifica da parte degli utenti, quindi non è stato possibile capire come questo algoritmo fosse fatto solo guardando il suo pseudo codice o il codice sorgente di una sua implementazione, bensì è stato necessario utilizzare la libreria che implementa l'algoritmo, distribuita per la prima volta da Google con la versione 4.4 (KitKat) di Android, per estrarne i dati risultanti ed altri dati provenienti dai sensori del dispositivo utilizzato, per poi effettuare uno studio su tali dati applicati ad algoritmi diversi e alla fine denotare le differenze tra i risultati. La prima fase ad essere eseguita è stata quella della raccolta dei dati, processo che ha permesso di creare un data set abbastanza grande da poter effettuare prove di comparazione per diverse attività umane riconoscibili da una macchina.

## 2.1 Raccolta dei dati

Per effettuare la raccolta dei dati ci si è avvalsi di un dispositivo Android e di un'applicazione per questo sistema operativo che svolgesse il lavoro necessario alle richieste dei diversi tipi di dati al dispositivo e al sistema operativo stesso. Si può vedere questa applicazione come un programma diviso in vari thread che operano ognuno per ottenere un singolo dato o, più precisamente, un singolo tipo di dato. Questa divisione è dovuta all'organizzazione delle API di Android che permettono di comunicare con i sensori o con altre librerie attraverso un paradigma di programmazione ad eventi nel quale, per ogni tipo di dato ritornato dal sensore che si vuole interrogare, bisogna creare un ascoltatore di eventi per quel dato sensore. Ad ogni ascoltatore è possibile affidare un quantitativo di tempo entro il quale si desidera ricevere aggiornamenti dei dati ciclicamente, in modo da permettere al sistema operativo di gestire le risorse del dispositivo in modo efficiente. Il funzionamento dei sensori in Android, infatti, prevede un meccanismo di richieste di dati da parte delle applicazioni al sistema operativo il quale, anche utilizzando il tempo di aggiornamento di ogni richiesta, gestisce il sensore in modo da ottenere dati da restituire a tutte le applicazioni che ne avevano fatto richiesta, il tutto gestendo il sensore in modo da evitare carichi eccessivi di utilizzo che richiederebbero troppa energia causando tempi di durata della batteria troppo bassi. I sensori interrogati dall'applicazione sono l'accelerometro e il GPS, o anche, a GPS spento, il gestore della posizione che si avvale delle reti senza fili, wifi o connessione dati, per ottenere, attraverso l'indirizzo IP, le coordinate geosatellitari che rappresentano la posizione attuale del dispositivo. I dati provenienti dai sensori, insieme ai dati riguardanti la libreria che si intende studiare, formano il data set totale risultante dall'esecuzione dell'applicazione durante i test effettuati per raccogliere dati. Il data set in sé è un file di testo nel quale, per ogni riga, vengono salvate 9 informazioni provenienti sia dai sensori, sia dalla libreria per l'Activity Recognition di Google, sia anche dal sistema (ad esempio il timestamp unix). Queste 9 informazioni sono dunque, nell'ordine in cui vengono salvate:

- unix timestamp del momento esatto del rilevamento
- risultato dell'elaborazione della libreria (attività in corso secondo l'algoritmo di Google)
- valore della probabilità del risultato
- vera attività in corso di svolgimento
- valore estratto dal sensore dell'accelerometro sull'asse x
- valore estratto dal sensore dell'accelerometro sull'asse y
- valore estratto dal sensore dell'accelerometro sull'asse z
- valore decimale della latitudine attuale del dispositivo
- valore decimale della longitudine attuale del dispositivo

L'applicazione, in pratica, interroga la libreria ed i sensori per ottenere i dati che queste componenti esterne restituiscono e, una volta ottenute tutte e 9 le informazioni necessarie, le salva su un file di testo. Dall'elenco precedente si può notare come i risultati della libreria di Activity Recognition di Google vengano espressi secondo una coppia di valori: risultato letterale della richiesta di riconoscimento dell'attività e valore della probabilità calcolata del risultato stesso. Questi due valori vengono salvati in coppia poiché ci mostrano indirettamente, almeno a grandi linee, come funziona l'algoritmo che genera i risultati. Infatti, la presenza di un valore di probabilità nei risultati ci fa capire che l'algoritmo implementato da Google, sulla base dei valori ottenuti dai sensori, elabora una lista di attività possibili, cioè che potrebbero essere in corso da parte dell'utente, ordinata sulla base della probabilità in ordine decrescente. Per poter interrogare la libreria di Activity Recognition è necessario creare una classe che estenda Activity[7] o una sua sottoclasse, nel nostro caso FragmentActivity, in quanto queste super classi dispongono del metodo onActivityResult(int requestCode, int resultCode, Intent data), il quale viene eseguito dal sistema in risposta ad una richiesta

di utilizzo dei Google Play Services, ritornando, nella variabile resultCode, anche il risultato della richiesta, in modo che sia possibile capire se il sistema operativo che esegue l'applicazione è dotato di Google Play Services o meno. È possibile che il sistema non abbia i Google Play Services in quanto Android, essendo un sistema open source, permette agli utenti di esaminare e modificare il codice del sistema operativo, mentre i Google Play Services, come altre app proprietarie di Google, possono essere distribuite a parte e quindi possono non essere presenti nel sistema. La classe creata, inoltre, deve implementare le interfacce ConnectionCallbacks ed OnConnectionFailedListener che sono utili per poter gestire il meccanismo di risposta della libreria di Activity Recognition sia nei casi in cui il sistema riesca a connettersi con i Google Play Services, sia nel caso in cui non ci riesca. Una volta appurato che il sistema operativo dispone dei Google Play Services, si deve istanziare un oggetto della classe ActivityRecognitionClient, contenuto nel package com.google.android.gms.location, disponibile dalla versione 4.4 di Android solo con i Google Play Services. In aggiunta a questo oggetto, è necessario creare un Intent, che punti al contesto dell'applicazione e ad una classe che estenda IntentService, una classe del package android.app che serve a dichiarare codice da eseguire come callback ma in background, ed un PendingIntent che serve per mostrare al sistema quale parte di codice eseguire, in risposta ad un evento, anche quando l'applicazione che lo contiene vie chiusa o uccisa dal sistema operativo per qualche motivo. Infatti la creazione del PendingIntent può prevedere un riferimento ad un IntentService che contiene la callback da eseguire per gestire i risultati in background. La callback in questione è rappresentata dal metodo onHandleIntent(Intent intent) il quale sarà chiamato tutte le volte che la libreria avrà elaborato un nuovo risultato di riconoscimento dell'attività, e che conterrà le istruzioni necessarie a comunicare i dati all'applicazione per poterli salvare nel data set. In questo modo, quando viene eseguito questo metodo, l'intent ricevuto in ingresso conterrà o meno i risultati di ActivityRecognition a seconda che l'elaborazione sia andata a buon fine o meno. La ricezione di un intent senza risultati può avvenire

ad esempio se per qualche ragione il sistema operativo spegne tutti i sensori come può avvenire nel caso in cui impostazioni di risparmio energetico prevendano tale azione oppure quando si spegne lo schermo per mettere in standby il dispositivo. Per far partire le richieste di aggiornamento dati verso la libreria, si deve implementare il metodo onConnected(Bundle bundle) dell'interfaccia ConnectionCallbacks, il quale sarà eseguito a seguito del controllo della presenza dei Google Play Services illustrato sopra. All'interno di questo metodo quindi, sarà necessario chiamare il metodo requestActivityUpdates(long detectionIntervalMillis, PendingIntent callbackIntent) che, come nel caso della richiesta ad un sensore, richiede in ingresso un parametro che rappresenta l'intervallo di aggiornamento in corrispondenza del quale inviare i risultati al PendingIntent, passato come secondo parametro, il quale, come visto sopra, contiene un riferimento alla classe che implementa la callback da chiamare per gestire la ricezione dei dati. Questo meccanismo di scatole cinesi permette la ricezione dei risultati anche in background, che altrimenti sarebbe impossibile effettuare una volta chiusa l'applicazione oppure senza che l'applicazione sia stata lanciata, come potrebbe essere nel caso di un servizio in background che serve appunto a lanciare l'applicazione se si verifica qualche evento particolare. Il funzionamento della libreria, ad ogni modo, permette di ricevere i dati con la risposta alle richieste di riconoscimento dell'attività corrente dell'utente. L'applicazione utilizzata per raccogliere i dati implementa sia il meccanismo di utilizzo appena illustrato della libreria di Activity Recognition, sia quello illustrato prima relativo al funzionamento dei sensori. Le raccolta dei dati è stata svolta eseguendo l'applicazione nel mentre che l'utente effettuava una attività tra quelle che la libreria è in grado di riconoscere; quindi, grazie all'interfaccia utente dell'applicazione, è stata indicata all'applicazione la vera attività in corso di svolgimento, in modo che la si potesse salvare nel data set per confrontare i risultati. Oltre a questo dato, dall'interfaccia utente si deve scegliere anche l'intervallo di aggiornamento dei dati per i sensori e per la libreria di Activity Recognition. La scelta dell'intervallo di aggiornamento avviene tra 4 valori (5, 10, 15 e 20

secondi), ed influisce sulla scelta del file che raccoglie il data set, in quanto i risultati vengono raggruppati a seconda dell'intervallo di aggiornamento con cui sono stati generati. Una volta indicata la vera attività e l'intervallo, si preme il tasto "on" nell'interfaccia grafica dell'applicazione per far partire le richieste ai vari sensori e alla libreria di Activity Recognition. Da questo momento l'applicazione inizia ad eseguire in parallelo i vari compiti che deve svolgere e, una volta ottenuto un insieme completo di dati, salva su file la riga che corrisponde a questo insieme. Nel mentre, l'utente continua a svolgere l'attività per la quale si desidera raccogliere i dati. Se, durante la raccolta, l'attività dell'utente cambia, ad esempio passando da in veicolo a a piedi o da in veicolo a fermo, ecc, l'utente può cambiare in corso d'opera, tramite lo stesso menu di selezione dell'interfaccia grafica, la vera attività che bisogna salvare come punto di riferimento, senza dover fermare l'applicazione e poi ripartire. Lo stesso non vale invece per l'intervallo di aggiornamento, per cambiare il quale bisogna interrompere la raccolta in corso e ricominciare dopo aver selezionato il nuovo intervallo che si desidera utilizzare. Questo meccanismo di funzionamento è stato utilizzato svariate volte per riuscire a raccogliere un data set totale, diviso in 4 files, di circa 1420 rilevamenti, ognuno strutturato come indicato sopra. Sono state effettuate raccolte dati per tutti i 4 tipi di attività che la libreria di Google è in grado di riconoscere, ovvero still (fermo), in vehicle (in veicolo), on bicycle (in bicicletta), on foot (a piedi). Vi è anche un quinto tipo di risultato che la libreria è in grado di ritornare come risultato, tilting, letteralemente scuotimento, ma non si è ritenuto necessario raccogliere dati per questa categoria in quanto più che una attività vera e propria è piuttosto un risultato frutto di movimenti maldestri del dispositivo nello spazio che rendono difficile, se non impossibile , il parsing dei dati provenienti dai sensori, su tutti l'accelerometro. Una volta ottenuto il data set, è stato necessario plasmare i dati contenuti in esso per poter estrarre grandezze diverse da quelle contenute nei dati. Infatti, dati i valori dei 3 assi provenienti dall'accelerometro, si è calcolata la magnitude, espressa come radice quadrata della somma dei quadrati dei 3 valori. Inoltre,

dato il timestamp unix presente in ogni rilevamento e date le due coordinate geosatellitari, si è calcolata la velocità di spostamento tra un rilevamento ed un altro, espressa come spazio fratto tempo (in km/h). Questi calcoli aggiuntivi sono stati effettuati con un programma scritto in Java ad hoc per questo compito e non all'interno dell'applicazione, in quanto si è ritenuto di generare altri files con le nuove grandezze calcolate e conservando i dati originali piuttosto che modificare la struttura del data set e perdere così una parte dei dati raccolti. Il trattamento del data set originale ha generato un altro data set identico ma con grandezze diverse contenute all'interno ed un formato di file diverso: non più file di testo (txt) bensì comma separeted values (csv), questo per facilitare il caricamento del data set all'interno dell'ambiente di simulazione Weka. Weka è un programma interamente scritto in Java che consiste di una collezione di algoritmi di machine learning per attività di data mining[1]. Questo ambiente è in grado di applicare a data set esistenti gli algoritmi di data mining che si vogliono testare. I dati possono essere caricati da un file esterno o da una base di dati, nel nostro caso, come detto, il caricamento è avvenuto tramite file csv, formato supportato da Weka insieme ad altri come arff o binary serialized format. Gli algoritmi di machine learning ai quali sono stati applicati i dati del data set sono 4:

- Random Forest
- Bayesian Network
- Naive Bayes
- SMO

L'output dell'elaborazione di Weka prevede una analisi dei dati attraverso Confusion Matrix e indici di precisione come precision e accuracy che saranno discusse nella sezione 2.3. La seguente Tabella 2.1 riassume le informazioni relative alla raccolta dei dati effettuata:

[1]Descrizione dal sito ufficiale

| | |
|---|---|
| Modello smartphone | Asus Phonepad 7 (K00E) |
| Versione S.O. | Android 4.3 “Jelly Bean” |
| Posizioni smartphone | Borsa, tasca, zaino, in mano |
| # Campionamenti | 1420 |
| # transportation mode riconosciuti | $\sim$ 747 |
| Mezzi di trasporto | Auto, moto, bicicletta, autobus, treno |

Tabella 2.1: Tabella riassuntiva della raccolta dati

## 2.2 Features

La scelta delle features da utilizzare per effettuare questo studio dipende unicamente dai sensori contenuti all'interno di dispositivi come smartphone e tablet. Si è scelto di considerare i dati dei sensori di accelerometro e GPS perché sono i più precisi tra i vari sensori che può contenere un dispositivo. Inizialmente era stato previsto anche l'uso di una terza feature, il giroscopio, salvo poi scoprire, al momento dell'impementazione dell'applicazione, che il dispositivo sul quale sono state effettuate le prove ne era sprovvisto. I dati provenienti dai sensori del dispositivo sono sempre espressi in termini di coordinate relative o assolute in base al tipo di sensore interrogato. Le coordinate relative esprimono numeri relativi al dispositivo stesso, come ad esempio l'accelerazione lungo i tre assi di un grafico tridimensionale, x, y e z, ritornata dal sensore accelerometro:

Come si vede dalla figura, l'asse x rappresenta l'accelerazione orizzontale del dispositivo in un dato momento di tempo, l'asse y quella verticale, mentre l'asse z rappresenta l'accelerazione di tutto il dispositivo nello spazio che lo circonda. I valori di x, y e z al momento della risposta del sistema operativo alla richiesta da parte dell'applicazione sono i dati che vengono inviati all'applicazione e quindi sono i dati relativi all'accelerazione che sono stati usati dalla nostra applicazione. Le coordinate assolute non prendono come riferimento il dispositivo, ma riferimenti assoluti nell'ambiente in cui esse esistono;

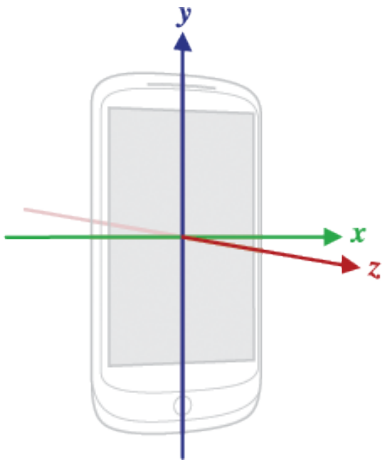


Figura 2.1: I 3 assi relativi ai dati dei sensori

ad esempio il GPS ritorna una coppia di numeri decimali che rappresentano la posizione attuale del dispositivo in un dato momento di tempo rispetto alla superficie terrestre.

Le coordinate di cui ci si è avvalsi per ottenere la posizione sono espresse

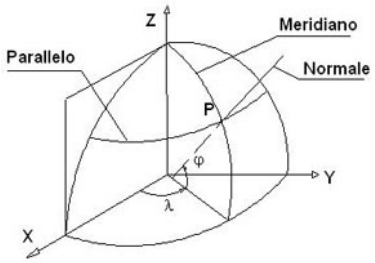


Figura 2.2: Coordinate polari rappresentate dai valori estreatti dal GPS

in gradi decimali dalle due grandezze di latitudine e longitudine. Come mostrato dalla figura, il sistema di tracciamento GPS è in grado di restituire anche un'altra grandezza, l'altitudine, che però non è stata necessaria ai fini dello studio. Questo perché le coppie latitudine-longitudine, nell'insieme di dati, combinate con una grandezza temporale, sono sufficienti a calcolare la velocità di uno spostamento del dispositivo, o di chi lo utilizza, nello spazio.

In Android, l'interrogazione di un sensore particolare avviene attraverso l'utilizzo delle classi Sensor e SensorManager, del package android.hardware, e dell'interfaccia SensorEventListener, contenuta nello stesso package. Il funzionamento di questi due oggetti è molto semplice e rispecchia quanto illustrato precedentemente sul funzionamento generale dei sensori in un sistema operativo Android. Per poter ricevere informazioni dai sensori, infatti, bisogna eseguire una classe, ad esempio una classe che estenda Activity, la quale implementi l'interfaccia SensorEventListener e che quindi dichiari e implementi il metodo onSensorChanged(SensorEvent event), il quale verrà eseguito, come callback[2] e che fornirà una sezione di codice, scritta da chi implementa l'interfaccia, nella quale è possibile accedere ai dati provenienti dal sensore che sono contenuti dall'oggetto della classe SensorEvent che il sistema passerà in ingresso quando eseguirà questo metodo. Per richiedere invece gli aggiornamenti, quindi per inizializzare l'ascoltatore degli eventi dei sensori, si usano un oggetto della classe SensorManager ed uno della classe Sensor, inizializzati all'interno della classe in esecuzione, chiamando il metodo booleano registerListener(SensorEventListener listener, Sensor sensor, int rateUS) che prende in ingresso un oggetto di SensorEventListener come riferimento per eseguire in futuro le callback implementate, un oggetto di Sensor per capire di quale sensore si desidera ricevere i dati ed un intero che rappresenta l'intervallo di aggiornamento per la ricezione dei dati, in pratica ogni quanto tempo il sistema deve effettuare una chiamata alla callback per inviare i dati del sensore all'applicazione. Quest'ultimo valore può essere scelto a discrezione dell'implementatore come un numero intero espresso in millisecondi oppure uno a scelta tra le costanti della classe SensorManager della serie SENSOR_DELAY_ che sono valori predefiniti per categorie conosciute di utilizzo dei sensori come SENSOR_DELAY_NORMAL, SENSOR_DELAY_UI, SENSOR_DELAY_GAME o SENSOR_DELAY_FASTER. Quindi, per interrogare più di un sensore è necessario avere due oggetti della

[2]Funzione o metodo eseguito nel paradigma ad eventi in risposta al verificarsi di un evento per il quale era stato creato un ascoltatore

classe Sensor, e chiamare due volte il metodo registerListener di SensorManager, controllando nell'implementazione del metodo onSensorChanged di SensorEventListener per quale dei due sensori si è ottenuto un risultato ed agire di conseguenza a seconda del caso. Lo stesso vale per un numero di sensori in generale maggiore di 1.

## 2.3 Metriche

Ad un certo punto dello studio, esattamente una volta raccolto tutto il data set, si è manifestata la necessità di trattare i dati grezzi provenienti dai sensori per poterne ricavare, attraverso calcoli matematici, dei dati migliori per poter affrontare il problema del riconoscimento automatico di attività. Inoltre, in un secondo momento, è stato necessario studiare i dati ottenuti dopo il trattamento di cui sopra, e quindi ci si è avvalsi di metriche di risultati differenti che fossero in grado di mostrarci le differenze tra i risultati di elaborazione della libreria di Activity Recognition di Google e gli algoritmi di machine learning scelti dal set contenuto in Weka. In questa sessione si presentano tali metriche.

### 2.3.1 Magnitude

La magnitude è una grandezza che serve a condensare più dati di un sensore, nel nostro caso provenienti dall'accelerometro, in un unico numero dato dalla formula $M = \sqrt{x^2 + y^2 + z^2}$ dove x, y e z sono i dati dell'accelerometro sui 3 assi come illustrato nella sezione precedente. Il calcolo di questa metrica all'interno della fase di trattamento dei dati grezzi è stato effettuato in quanto essa, a differenza del set di 3 valori, non dipende dall'orientamento del dispositivo al momento del rilevamento. Questo ci permette di avere una stima dell'accelerazione del dispositivo in termini assoluti, quindi ci garantisce una maggiore precisione nell'elaborazione dei dati attraverso gli algoritmi di machine learning.

### 2.3.2 Confusion Matrix

Nell'ambito dell'intelligenza artificiale, la Confusion Matrix restituisce una rappresentazione dell'accuratezza di classificazione statistica[7]. Questa matrice contiene sulle colonne i valori dei classificatori predetti, mentre sulle righe i valori reali, in modo da tastare la precisione della previsione. Nel caso di questo studio la precisione misurata è quella dei risultati di elaborazione della libreria di Activity Recognition di Google rispetto ai valori reali indicati direttamente dall'utente al momento della raccolta dei dati. Inoltre, seguendo lo stesso principio, la Confusion Matrix è stata utilizzata per calcolare anche la precisione degli algoritmi di machine learning di Weka applicati ai dati raccolti (magnitude e velocità). La matrice di confusione relativa ai risultati della libreria è stata generata all'interno del programma Java creato per trattare i dati ed estrarne un data set più efficiente, mentre quelle relative agli algoritmi di Weka sono state generate come parte dei risultati dell'elaborazione di Weka stesso.

### 2.3.3 Accuracy

L'accuracy è definita come il numero di volte in cui la predizione risulta corretta diviso per il numero di campioni totali testati ($\frac{\#CampioniCorretti}{\#CampioniTotali}$). I numeri dei campioni corretti e totali vengono presi direttamente dalla Confusion Matrix relativa ai dati di cui si vuole calcolare l'accuracy. Come nel caso delle Confusion Matrix, anche l'accuracy relativa ai risultati della libreria di Activity Recognition è stata calcolata all'interno del programma Java utilizzato per trattare i dati, mentre quelle relative ai risultati degli algoritmi di machine learning di Weka sono state generate da Weka stesso.

## 2.4 Algoritmi

### 2.4.1 Random Forest

Il Random Forest è un classificatore basato su un approccio insiemistico che può essere pensato come una forma di predittore di vicino più prossimo. Gli insiemi sfruttano un approccio divide-et-impera per migliorare le prestazioni complessive dell'algoritmo. Il principio portante dei metodi insiemistici è consiste nell'utilizzare un insieme di learners "deboli" per formare un learner "forte". In pratica ogni classificatore preso singolarmente è un learner debole, ma tutto l'insieme ne forma uno forte. In questo meccanismo il Random Forest utilizza una classica tecnica di machine learning chiamata decision tree (albero di decisione) che, in termini insiemistici rappresenta il learner "debole". In un decision tree i dati vengono inseriti in testa all'albero e man mano lo attraversano vengono divisi in set di dati sempre più piccoli. Quindi in termini insiemistici i singoli alberi di decisione sono learners "deboli" mentre tutto il Random Forest rappresenta il learner "forte". L'algoritmo quindi capiona un numero $N$ di casi casualmente con sostituzione, per creare un sotto-insieme di dati, il quale risulta essere circa il 66% del set di dati iniziale. Dopodichè, per ogni nodo degli alberi di decisione, dato un numero $m$, vengono selezionate $m$ variabili di previsione dall'insieme di tutte le variabili di previsione. La variabile di previsione tra le $m$ scelte che fornisce la migliore divisione ammissibile, secondo alcune funzioni obiettivo, viene usata per effettuare una divisione binaria del nodo in questione, creando due nodi figli del nodo diviso. Per ogni nodo dell'albero vengono scelte altre $m$ variabili di previsione e con la migliore di esse viene di nuovo effettuata la divisione binaria. Il Random Forest prevede valori di $m$ a scelta tra $\frac{1}{2}\sqrt{m}$, $\sqrt{m}$ e $2\sqrt{m}$. In questo modo viene effettuata la divisione tra i dati in modo da classificare, sulla base di tutti i decision tree i dati in una classe, tra quelle possibili, che rappresenta l'output dell'algoritmo di classificazione.

### 2.4.2 Naive Bayes

L'algoritmo Naive Bayes è un classificatore basato sull'applicazione del Teorema di Bayes. Quest'ultimo è un teorema matematico basato sulla probabilità condizionata che serve a calcolare la causa che ha scatenato un dato evento. Si può vedere il classificatore bayesiano come un algoritmo probabilistico che sfrutta la probabilità che ha un set di dati di appartenere ad una data classe (output) per classificare i dati man mano che l'algoritmo viene eseguito e che diventa sempre più preciso al'aumentare della mole di dati classificati. Questo algoritmo si basa sull'ipotesi che una feature di una classe sia presente (o assente) nei dati indipendentemente dalla presenza (o assenza) di altre features. Per applicare questo classificatore al problema del riconoscimento automatico di attività si possono pensare le features come i dati catturati dai sensori, quindi il classificatore basa la decisione della classe considerando la frequenza con cui i valori dei dati appaiono nel data set; in modo analogo il Naive Bayes è in grado di classificare i testi (applicazione più appropriata di questo algoritmo) considerando la frequenza con cui determinate parole appaiono nel testo da classificare; le classi di classificazione di un testo possono essere ad esempio spam o legittimo, sport o politica ecc.

### 2.4.3 Bayesian Network

L'algoritmo Bayesian Network si basa sulle reti di Bayes, dalle quali eredita il nome, che sono strutture dati formate da grafi aciclici diretti in cui ogni nodo rappresenta una variabile ed ogni arco pesato rappresenta la relazione di dipendenza statistica tra le variabili e le distrubuzioni locali di probabilità dei nodi foglia rispetto ai nodi padre. Quindi tutta la rete rappresenta la distribuzione di probabilità congiunta dell'insieme delle variabili. Nel caso del riconoscimento automatico di attività i nodi padre sono i dati che si intende classificare, i nodi foglia sono le classi di attività, mentre i valori degli archi corrispondono alla probabilità di un insieme di dati di appartenere ad una data classe. L'algoritmo costruisce il grafo con i dati da classificare ed assegna

probabilità (valori degli archi) maggiori man mano che il numero di dati classificati aumenta, applicando il concetto di apprendimento dall'esperienza.

### 2.4.4 SMO

L'algoritmo SMO (acronimo di Sequential Minimal Optimization è un algoritmo utile per risolvere problemi di Programmazione Quadratica che sorgono durante test di Support Vector Machines (SVM). Gli algoritmi di Support Vector Machines, applicati a modelli di learning supervisionati, sono in grado di riconoscere pattern dall'analisi di dati utilizzabili per la classificazione dei dati stessi. Dato un insime di dati di esempio, ognuno dei quali poò appartenere ad una o due categorie, un algoritmo di addestramento SVM costruice un modello che assegna un nuovo dato (o un nuovo set di dati) ad una categoria o all'altra a seconda del pattern riconosciuto. In relazione al riconoscimento automatico di attività, a seconda del patter riconosciuto in un set di dati, questo algoritmo assegna la classe di appartenenza relativa allo stesso pattern. Questo tipo di algoritmo viene definito *classificatore binario non probabilistico*.

### 2.4.5 Metaclassificatori

I metaclassificatori sono algoritmi di machine learning basati su meta dati che in genere applicano una serie di algoritmi differenti per arrivare alla classificazione di un set di dati. Gli algoritmi applicati ai metaclassificatori utilizzati in questo studio sono gli stessi esposti fin qui in questo capitolo: Random Forest, Naive Bayes, Bayesian Network ed SMO. Nei test relativi a questo studio sono stati utilizzati due metaclassificatori: multi stage cascading e voting.

#### Multi-Stage Cascading

Il multi stage cascading è un metaclassificatore che sfrutta l'esecuzione di un gruppo di algoritmi di machine learning per arrivare ad una classificazione

più precisa, basata appunto sui risultati dei singoli algoritmi. In pratica, per ogni set di dati da classificare, vengono eseguiti uno dopo l'altro tutti gli algoritmi di machine learning indicati e, sulla base dei risultati dei singoli, il metaclassificatore decide quale sia la classe più opportuna da ritornare come output.

### Voting

Il voting è un tipo di metaclassificatore basato anch'esso su metadati derivanti dall'esecuzione di differenti algoritmi di machine learning. In questo caso l'algoritmo esegue, per ogni set di dati, il primo degli algoritmi indicati dai risultati del quale vengono estratti dei metadati che saranno dati in pasto al secondo algoritmo della lista scelta, e così man mano che vi sono ancora algoritmi di machine learning a eseguire. Quando l'ultimo algoritmo di machine tra quelli scelti viene eseguito sui metadati estratti dai risultati dell'algoritmo precedente, viene estratta la classe di appartenenza del set di dati e si ricomincia con il set successivo riapplicando tutta la catena. La differenza tra voting e cascading consiste nell'utilizzo dei metadati per arrivare alla classificazione: il voting applica direttamente ai metadati gli algoritmi successivi nella lista, mentre il cascading raccoglie i metadati di tutti gli algoritmi nella lista e in base ad essi decide la classe da applicare al set di dati.

# Capitolo 3

# Implementazione

L'implementazione di questo esperimento è avvenuta tramite la creazione di un'applicazione per la piattaforma Android che racchiude tutti i gestori di dati necessari all'ottenimento delle informazioni utili a creare il data set. L'applicazione consiste di una sola activity (Figura 3.1), considerato anche che essa è stata scritta con l'obiettivo di raccogliere dati e non per creare strumenti grafici di supporto al riconoscimento automatico di attività. L'unica activity contiene al suo interno i widget grafici per: permettere all'utente di selezionare la vera attività corrente, permettere all'utente di selezionare l'intervallo di aggiornamento desiderato, far partire il meccanismo di raccolta dati, esportare i dati su file una volta raccolti.
Oltre all'activity, l'applicazione è stata progettata per essere in grado di raccogliere, contemporaneamente, i dati provenienti dai due sensori e quelli provenienti dalla libreria di Activity Recognition di Google. Per poter essere in grado di assolvere a questi compiti in parallelo, l'applicazione, al momento del lancio, registra degli ascoltatori di eventi per indicare al sistema operativo di voler ricevere i dati dell'accelerometro, del GPS e della libreria. Questi ascoltatori prevedono che l'activity implementi le interfacce ConnectionCallbacks, OnConnectionFailedListener e LocationListener. Le prime due sono necessarie per poter richiedere aggiornamenti di dati alla libreria di Google e, implementandole, si creano anche le callback all'interno delle quali vengono

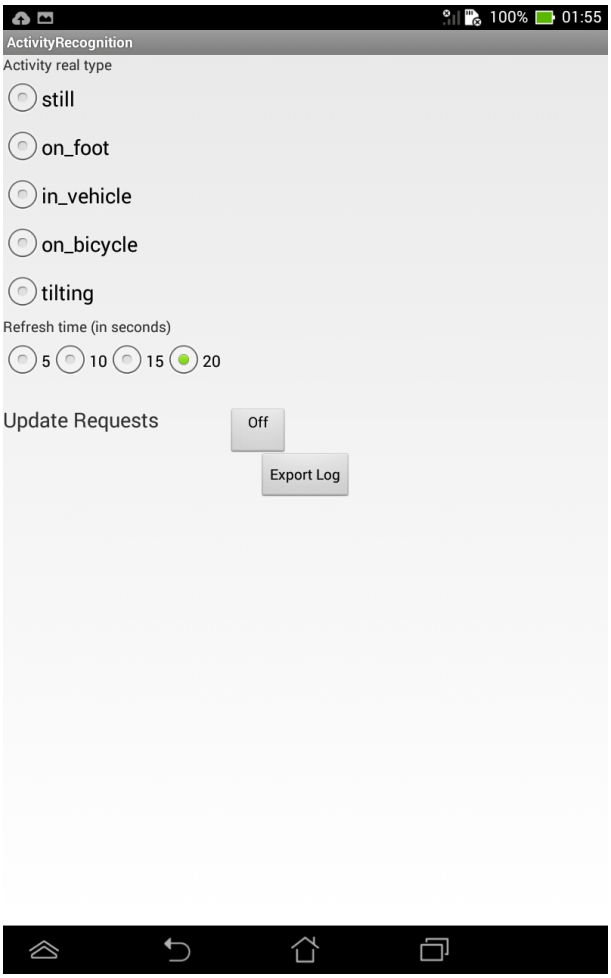


Figura 3.1: Screenshot dell'activity dell'applicazione

gestite le richieste di inizio di raccolta dei dati della libreria. LocationListener, invece, permette di registrare callback che saranno chiamate dal sistema tutte le volte che sarà disponibile un aggiornamento dei dati relativi alla posizione. Il paradigma di programmazione ad eventi illustrato nel capitolo 3 relativamente alla raccolta dei dati dei sensori è frutto del modo in cui Android (e Java più in generale) gestisce i meccanismi di comunicazione tra le diverse parti di un'applicazione e i servizi essenziali messi a disposizione dal sistema operativo. Il codice seguente mostra come è possibile richiedere i dati relativi alla posizione mantenendo in esecuzione l'applicazione e i suoi

componenti grafici in parallelo:

```
import com.google.android.gms.location.LocationClient;
import com.google.android.gms.location.LocationListener;
import com.google.android.gms.location.LocationRequest;

private LocationClient locationClient;
private LocationRequest locationRequest;

public class Main extends FragmentActivity implements
    ConnectionCallbacks, OnConnectionFailedListener,
    LocationListener {

  @Override
  protected void onCreate(Bundle savedInstanceState) {

    ...

    locationRequest = LocationRequest.create();
    locationClient = new LocationClient(this, this, this);

    ...
  }

  ...

  @Override
  public void onLocationChanged(Location arg0) {
    lastLocationLat = arg0.getLatitude();
    lastLocationLng = arg0.getLongitude();
  }

  ...
}
```

Il metodo onLocationChanged verrà chiamato ogni volta che il sistema operativo dovrà restituire all'applicazione la nuova posizione corrente del dispositivo. Per poter effettuare la richiesta di aggiornamento della posizione è

necessario registrare l'ascoltatore per dire al sistema operativo che si vuole ricevere la posizione del dispositivo. Tale registrazione si effettua con una chiamata al metodo requestLocationUpdates della classe LocationClient:

```
SharedPreferences settings = getSharedPreferences(SP_NAME,
    MODE_APPEND);
String settingsInterval = settings.getString(
    TIME_INTERVAL_SETTINGS_KEY, "19");
int interval = Integer.parseInt(settingsInterval);
long millis = (long) (interval - 1) * 1000;
locationRequest.setInterval(millis);
locationClient.requestLocationUpdates(locationRequest, this);
```

In questo blocco di codice si può vedere come il valore dell'intervallo di aggiornamento di tutti i valori dei sensori e della libreria sia preso dal valore impostato dall'utente tramite l'interfaccia grafica e memorizzato all'interno delle preferenze dell'applicazione con chiave TIME_INTERVAL_SETTINGS_KEY. Il codice che permette di impostare il valore dell'intervallo di aggiornamento scelto dall'utente è il seguente:

```
final RadioButton timeFifteen = (RadioButton) findViewById(R.
    id.time_fifteen);
timeFifteen.setOnClickListener(new OnClickListener() {

  @Override
  public void onClick(View v) {
    SharedPreferences.Editor editor = settings.edit();
    String value = res.getString(R.string.time_fifteen);
    editor.putString(TIME_INTERVAL_SETTINGS_KEY, value);
    boolean esito = editor.commit();
    if (!esito) {
      Toast.makeText(getApplicationContext(), "Error setting
          new time interval to
           15", Toast.LENGTH_LONG).show();
    }
  }
});
```

In questo estratto è mostrata l'impostazione dell'intervallo di aggiornamento in risposta ad un tap dell'utente sul radio button che indica il numero 15 nell'activity dell'applicazione. Questo valore viene poi impostato come intervallo dell'oggetto locationRequest della classe LocationRequest con il metodo setInterval. L'oggetto viene poi passato al client della posizione e al suo metodo requestLocationUpdates per poter iniziare a ricevere aggiornamenti della posizione. Si può notare come il meccanismo di callback in risposta ad eventi sia utilizzato, in Android, per registrare tutti i tipi di eventi, sia che essi provengano da servizi o features del sistema operativo, sia che essi appartengano solo all'applicazione. Lo stesso meccanismo viene usato per poter ricevere aggiornamenti dal sensore dell'accelerometro e dalla libreria per l'Activity Recognition di Google. Le classi necessarie per istanziare oggetti che ci permettano di effettuare richieste alla libreria di Activity Recognition di Google sono diverse: la classe ActivityRecognitionClient è una classe della libreria (o package) com.google.android.gms.location che permette di avere all'interno di una applicazione un oggetto in grado di comunicare con i Google Play Services per ottenere le informazioni necessarie al riconoscimento di attività. Attraverso questo client, infatti, si può iniziare a richiedere al sistema aggiornamenti dei dati calcolati dalla libreria, sulla base dei dati ricevuti dai Google Play Services, relativi all'attività in corso da parte dell'utente. Per istanziare un oggetto della classe ActivityRecognitionClient è necessario avere a disposizione una classe o un oggetto che implementi le interfacce ConnectionCallbacks ed OnConnectionFailedListener; questo per permettere al client, sia in caso di successo, sia in caso di errore, di poter eseguire una callback, definita da chi implementa le interfacce, in grado di gestire i risultati della connessione con i Google Play Services necessaria per ottenere le informazioni sulla attività corrente. Come mostrato precedentemente, la classe che implementa queste due interfacce nell'applicazione creata per questo studio è la classe Main, utilizzata anche come sottoclasse di Activity per mostrare a schermo l'interfaccia grafica utente che contiene i widget per impostare la racoolta dei dati e farla partire o fermarla. Implementando

le due interfacce suddette, la classe Main dichiara i metodi necessari per il funzionamento dei Google Play Services e degli errori:

- onConnectionFailed(ConnectionResult connection)
- onConnected(Bundle bundle)
- onDisconnected()

Il metodo onConnectionFailed viene chiamato nel caso in cui la richiesta di connessione ai Google Play Services non vada a buon fine. Spesso ciò accade semplicemente per la mancanza di questi servizi sul sistema operativo. Google consiglia [7] di mostrare all'utente una maschera di dialogo nella quale è suggerito di installare i Google Play Services per far funzionare l'applicazione. Questo è esattamente quello che la nostra implementazione del metodo onConnectionFailed fa.

```
@Override
public void onConnectionFailed(ConnectionResult connection) {
  inProgress = false;

  if(connection.hasResolution()) {
    try {
      connection.startResolutionForResult(this,
          CONNECTION_FAILURE_RESOLUTION_REQUEST);
    } catch (SendIntentException e) {
      e.printStackTrace();
    }
  } else {
    int errorCode = connection.getErrorCode();
    Dialog errorDialog = GooglePlayServicesUtil.
       getErrorDialog(errorCode, this,
       CONNECTION_FAILURE_RESOLUTION_REQUEST);

    if(errorDialog != null) {
      ErrorDialogFragment errorFragment = new
          ErrorDialogFragment();
      errorFragment.setDialog(errorDialog);
```

```
        errorFragment.show(getSupportFragmentManager(), "
            Activity Recognition");
      }
    }
}
```

I metodi onConnected ed onDisconnected invece, fanno parte dell'interfaccia ConnectionCallbacks e servono a gestire la connessione e la disconnessione dai Google Play Services in modo appropriato. In pratica i metodi dichiarati dalle due interfacce sono alternativi tra loro. Infatti essi vengono chiamati nel caso in cui i Google Play Services siano già presenti nel sistema operativo e già pronti per l'uso. Il metodo onConnected viene chiamato a connessione eseguita, come suggerisce anche il nome, e serve a mettere a disposizione degli sviluppatori una callback per gestire l'avvenuta connessione con i servizi di Google. Questo approccio risulta molto utile in quanto i Google Play Services permettono di usufruire di una vasta gamma di servizi diversi, quindi avere un entry point che comunichi l'avvenuta connessione all'interno della quale gestire l'inizio delle richieste verso il singolo servizio che si desidera usare risulta molto conveniente, e lo è ancora di più nel caso che si vogliano utilizzare più servizi contemporaneamente. Quindi all'interno della callback onConnected non facciamo altro che richiedere alla libreria di Activity Recognition di iniziare a cercare di capire che tipo di attività l'utente stia svolgendo. Per fare ciò utilizziamo il seguente codice che implementa il metodo onConnected:

```
@Override
public void onConnected(Bundle arg0) {

  switch(requestType) {
    case START :
      try {
 SharedPreferences settings = getSharedPreferences(SP_NAME,
    MODE_APPEND);
 String settingsInterval = settings.getString(
    TIME_INTERVAL_SETTINGS_KEY, "20");
```

```
int interval = Integer.parseInt(settingsInterval);
client.requestActivityUpdates(interval * 1000, pIntent);
saveUpdateState(true);
     } catch(Exception e) {
  e.printStackTrace();
     }

     inProgress = false;
     client.disconnect();

     break;

     case STOP :
client.removeActivityUpdates(pIntent);

try {
  saveUpdateState(false);
} catch (Exception e1) {
  e1.printStackTrace();
}

break;

     default :
try {
  throw new Exception("Request type not valid");
} catch (Exception e) {
  e.printStackTrace();
}

break;
 }
}
```

Con questo codice si richiede alla libreria di iniziare ad inviare all'applicazione i risultati delle sue elaborazioni riguardanti i dati dei sensori per capire l'attività in corso da parte dell'utente. Per inviare le informazioni con i risultati si passa al metodo requestActivityUpdates un oggetto della classe

PendingIntent creato in fase di inizializzazione dell'applicazione. Tale oggetto, come descritto precedentemente, serve a garantire al sistema operativo una callback per gestire i risultati in background anche quando l'applicazione è stoppata o chiusa. Questo meccanismo è premesso dalla struttura della libreria di Google per l'Activity Recognition e da quella del sistema operativo stesso, in quanto è lui che si occupa, avvisato dalla libreria nel momento in cui può fornire un nuovo risultato, di ricordare qual'era il PendingIntent di riferimento registrato da ogni applicazione interessata a ricevere determinate informazioni da un servizio. Questo metodo gestisce sia la fase di start che quella di stop relative alle richieste verso la libreria. Quando l'utente, attraverso l'interfaccia grafica, indica all'applicazione che vuole terminare la ricezione di aggiornamenti viene chiamato il metodo onConnected con valore di requestType uguale a STOP, il che comporta la chiamata del metodo removeActivityUpdates dell'oggetto client che indicherà al sistema operativo e alla libreria la volontà dell'utente di terminare la ricezione dei dati dal servizio. In seguito a questa chiamata, la libreria smette di funzionare e il sistema chiama forzatamente l'altra callback utilizzata, il metodo onDisconnected, per gestire internamente all'applicazione la fase post-stop della libreria. Nel nostro caso l'implementazione di questo metodo prevede il settaggio di alcuni parametri booleani che indicano in tempo reale se si sta utilizzando la libreria di Google ed i sensori a FALSE, per poi deinizializzare gli oggetti client e locationClient per liberare memoria:

```
@Override
public void onDisconnected() {
  inProgress = false;
  locationInProgress = false;
  client = null;
  locationClient = null;
}
```

Per quanto riguarda l'accelerometro, l'interfaccia da implementare nella classe di interesse è SensorEventListener: grazie a questa interfaccia, si può implementare il metodo onSensorChanged all'interno del quale è possibile

indicare il codice che sarà in grado di gestire e salvare i dati provenienti dal sensore (in questo caso i 3 valori degli assi dell'accelerometro):

```
public class ActivityRecognitionPendingIntent
  extends IntentService
  implements SensorEventListener {

  private SensorManager sensorManager;
  private Sensor accelerometer;

  ...

  @Override
  public void onSensorChanged(SensorEvent event) {


    Sensor sensor = event.sensor;
    if (sensor.getType() == Sensor.TYPE_ACCELEROMETER) {

      // Isolate the force of gravity with the low-pass
         filter.
      gravity[0] = alpha * gravity[0] + (1 - alpha) * event.
         values[0];
      gravity[1] = alpha * gravity[1] + (1 - alpha) * event.
         values[1];
      gravity[2] = alpha * gravity[2] + (1 - alpha) * event.
         values[2];

      // Remove the gravity contribution with the high-pass
         filter.
      linear_acceleration[0] = event.values[0] - gravity[0];
      linear_acceleration[1] = event.values[1] - gravity[1];
      linear_acceleration[2] = event.values[2] - gravity[2];

      Intent bIntent = new Intent();
      bIntent.setAction(UpdateBroadcastReceiver.
         REQUEST_MESSAGE);
      bIntent.addCategory(Intent.CATEGORY_DEFAULT);
```

```
        bIntent.putExtra(UpdateBroadcastReceiver.RESPONSE_MODE,
            1);
        bIntent.putExtra(UpdateBroadcastReceiver.RESPONSE_X,
           linear_acceleration[0]);
        bIntent.putExtra(UpdateBroadcastReceiver.RESPONSE_Y,
           linear_acceleration[1]);
        bIntent.putExtra(UpdateBroadcastReceiver.RESPONSE_Z,
           linear_acceleration[2]);
        sendBroadcast(bIntent);
    }
    sensorManager.unregisterListener(this, accelerometer);
}
```

Quando scade l'intervallo di aggiornamento della libreria di Activity Recognition di Google, insieme ai risultati ottenuti dall'elaborazione della libreria, vengono richiesti anche i dati relativi all'accelerometro e, una volta ottenuti, il sistema operativo chiama il metodo di callback onSensorChanged registrato precedentemente all'interno del quale vengono inviati all'activity dell'applicazione, per mezzo di un Broadcast, i dati ottenuti dal sensore per poter essere salvati, insieme ai dati relativi alla posizione del dispositivo contenuti in due variabili della classe dell'activity (Main). I dati inviati con il Broadcast vengono ricevuti dall'activity principale dell'applicazione grazie alla precedentemente registrazione di un BroadcastReceiver all'interno di una delle fasi di inizializzazione dell'activity. Il funzionamento dei BroadcastReceiver in Android è molto semplice e soprattutto molto utile per far comunicare pezzi diversi dell'applicazione che altrimenti non potrebbero scambiarsi dati tra loro. Il concetto alla base dello scambio di messaggi tra parti diverse di un'applicazione è lo stesso utilizzato per le richieste ai vari sensori del dispositivo, con la differenza che il BroadcastReceiver è completamente personalizzabile in entrabi i sensi (chi invia e chi riceve). Per creare un BroadcastReceiver personalizzato basta creare una clase che estenda BroadcastReceiver, all'interno della quale implementare il metodo onReceive con il proprio codice personalizzato. Dal lato di chi invia, quindi, basta creare un intent contenente la stessa action personalizzata utilizzata dal ricevitore

registrato presso il sistema operativo. Una volta infilati nell'intent tutti i dati che si vogliono inviare con le rispettive chiavi per estrarli, si lancia un intent implicito contenente i dati al sistema, il quale si occuperà di cercare le applicazioni e i servizi, o parti di essi, che aveva registrato un receiver, filtrando l'action tramite un IntentFilter, per l'intent implicito ricevuto. Trovato il receiver corrispondente, il sistema operativo esegue il suo metodo onReceive e lo scambio di dati è completato. Il codice relativo allo scambio di messaggi utilizzato nell'applicazione è il seguente:

```
@Override
protected void onResume() {
  super.onResume();
  IntentFilter filter = new IntentFilter(
     UpdateBroadcastReceiver.REQUEST_MESSAGE);
  filter.addCategory(Intent.CATEGORY_DEFAULT);
  receiver = new UpdateBroadcastReceiver();
  registerReceiver(receiver, filter);
}

...

public class UpdateBroadcastReceiver extends
    BroadcastReceiver {

  public static final String REQUEST_MESSAGE = "org.raffa.
     activityrecognition.update";
  public static final String RESPONSE_STRING = "org.raffa.
     activityrecognition.string";
  public static final String RESPONSE_TYPE = "org.raffa.
     activityrecognition.type";
  public static final String RESPONSE_REALTYPE = "org.raffa.
     activityrecognition.realtype";
  public static final String RESPONSE_CONFIDENCE = "org.raffa
     .activityrecognition.confidence";
  public static final String RESPONSE_LAT = "org.raffa.
     activityrecognition.lat";
  public static final String RESPONSE_LNG = "org.raffa.
```

```
    activityrecognition.lng";
 public static final String RESPONSE_MODE = "org.raffa.
    activityrecognition.mode";
 public static final String RESPONSE_X = "org.raffa.
    activityrecognition.x";
 public static final String RESPONSE_Y = "org.raffa.
    activityrecognition.y";
 public static final String RESPONSE_Z = "org.raffa.
    activityrecognition.z";

 @Override
 public void onReceive(Context context, Intent intent) {
   int mode = intent.getIntExtra(RESPONSE_MODE, -1);
   if (mode != -1) {
     switch (mode) {
     case 0:
String type = intent.getStringExtra(RESPONSE_TYPE);
int confidence = intent.getIntExtra(RESPONSE_CONFIDENCE, 0);
Main.responseActivityName = type;
Main.responseConfidence = confidence;
Main.responseRealType = intent.getStringExtra(
   RESPONSE_REALTYPE);
Main.responseCounter++;
break;
     case 1:
Main.responseX = intent.getFloatExtra(RESPONSE_X, 0);
Main.responseY = intent.getFloatExtra(RESPONSE_Y, 0);
Main.responseZ = intent.getFloatExtra(RESPONSE_Z, 0);
Main.responseCounter++;
break;
     case 2:
Main.responseLat = intent.getDoubleExtra(RESPONSE_LAT, 0);
Main.responseLng = intent.getDoubleExtra(RESPONSE_LNG, 0);
Main.responseCounter++;
     }
   }

   if (Main.responseCounter == 3) {
```

```
            FileLogger.log(getApplicationContext(), Main.
                responseActivityName, Main.responseConfidence, Main.
                responseRealType, Main.responseX,
             Main.responseY, Main.responseZ, Main.responseLat, Main
                .responseLng );
            Main.responseCounter = 0;
        }
    }
}
```

Il blocco `switch` all'interno del metodo onReceive discrimina il tipo di dati ricevuti col BroadcastReceiver per poter estrarre le informazioni dall'intent con le chiavi giuste. Questo perchè è possibile ricevere in questo blocco 3 tipi diversi di dati:

- i valori dei 3 assi dell'accelerometro
- i valori del GPS, che per questioni implementative fanno uno strano giro per poi ritornare all'activity
- i dati della libreria di Activity Recognition di Google

La funzione della variabile responseCounter è quella di contare i tipi di dati ricevuti all'interno di questo codice. Quando tutti e 3 i tipi di dati sono stati ricevuti dall'activity, si è certi di aver composto una riga del data set, quindi il nuovo record viene salvato su file chiamando il metodo statico log della classe FileLogger, creata appositamente per questo scopo:

```
public class FileLogger {

  public static String FILENAME="ar_log.txt";
  public static final String ACCELEROMETER_FILENAME="
     ar_log_accelerometer.txt";
  public static final String GYROSCOPE_FILENAME="
     ar_log_gyroscope.txt";

  public static void log(Context context, String type, int
     confidence, String realType, float x, float y, float z,
     double lat, double lng) {
```

```
    SimpleDateFormat sdf = new SimpleDateFormat("
       ddMMyyyy_HHmmss", Locale.ITALIAN);
    String currentDateandTime = sdf.format(new Date());

    /*
      * Controllo per l'intervallo di aggiornamento in modo
        da modificare il file di output su cui loggare i
        risultati
      */
    SharedPreferences settings = context.getSharedPreferences
       (Main.SP_NAME, Context.MODE_APPEND);
    int timeInterval = Integer.parseInt(settings.getString(
       Main.TIME_INTERVAL_SETTINGS_KEY, "20"));
    if (timeInterval != 20) {
      FILENAME = "ar_log_"+timeInterval+"_seconds.txt";
    }

    String toWrite = currentDateandTime+" "+type+" "+
       confidence+" "+realType+" "+x+" "+y+" "+z+" "+lat+" "+
       lng+"\n";

    try {
      FileOutputStream stream = context.openFileOutput(
         FILENAME, Context.MODE_APPEND);
      stream.write(toWrite.getBytes());
      stream.flush();
      stream.close();
    } catch (FileNotFoundException e) {
 e.printStackTrace();
    } catch (IOException e) {
 e.printStackTrace();
    }
  }

  ...
}
```

Come denotato dal commento all'interno del metodo log, il nome del file su

cui salvare i dati sarà differente per i dati raccolti con differenti intervalli di aggiornamento. Questo metodo salva i record creati dall'activity con i dati ricevuti dai diversi pezzi dell'applicazione su un file che sarà localizzato, nel filesystem del dispositivo, in una cartella assegnata all'applicazione dal sistema operativo, quindi impossibile da trovare con i classici gestori di file di Android. Per questo motivo è stato predisposto un meccanismo di export, già citato precedentemente, che essenzialmente copia il file del data set interno in un file all'interno di una cartella situata sulla memoria esterna, in genere su scheda sd. Il file copiato diventa il data set che poi sarà trattato dal programma Java indicato nel capitolo 2 e dal quale infine sarà estratto il data set finale che viene dato in pasto a Weka. Il metodo che copia il data set su memoria esterna è il seguente:

```
public static void exportLog(Context context) {
  SharedPreferences settings = context.getSharedPreferences(
     Main.SP_NAME, Context.MODE_APPEND);
  int timeInterval = Integer.parseInt(settings.getString(Main
     .TIME_INTERVAL_SETTINGS_KEY, "20"));

  try {
    if (timeInterval != 20) {
      FILENAME = "ar_log_"+timeInterval+"_seconds.txt";
    }
    FileInputStream inputStream = context.openFileInput(
       FILENAME);
    File root = Environment.getExternalStorageDirectory();
    String rootPath = root.getAbsolutePath()+"/ar/
       AR_LogFile_Sensors.txt";
    if (timeInterval != 20) {
      rootPath = root.getAbsolutePath()+"/ar/
         AR_LogFile_Sensors_"+timeInterval+"_seconds.txt";
    }
    FileOutputStream outputStream = new FileOutputStream(
       rootPath, false);

    int read;
```

```
    while((read = inputStream.read()) != -1) {
      outputStream.write(read);
      outputStream.flush();
    }
    inputStream.close();
    outputStream.close();
  } catch (FileNotFoundException e) {
      e.printStackTrace();
  } catch (IOException e) {
      e.printStackTrace();
  }

  Toast.makeText(context, "Log files exported", Toast.
     LENGTH_LONG).show();

}
```

# Capitolo 4

# Risultati

I risultati dell'elaborazione degli algoritmi di machine learning utilizzati come classificatori all'interno di Weka sono tutti espressi in funzione delle Confusion Matrix e dei valori medi della accuracy calcolate da Weka stesso. Queste matrici, come spiegato nel capitolo 2, ci indicano la precisione degli algoritmi scelti in funzione del riconoscimento automatico di attività.Gli indici statistici utlizzati per quantificare la precisione sono invece:

- TP Rate: Tasso di veri positivi
- FP Rate: Tasso di falsi positivi
- Precision: precisione o fedeltà
- Recall: recupero o completezza
- F-Measure: media armonica pesata tra precision e recall
- ROC Area: area sottostante alla curva di funzione di TP Rate e FP Rate in un grafico

Inotre tutti i test sui dati sono stati effettuati utilizando classificatori singoli (Random Forest, SMO, Naive Bayes e Bayesian Network), e due metaclassificatori (Cascading e voting) i quali sono classificatori che utilizzano in prove consecutive tutti e 4 i singoli algoritmi estrando risultati come se fosse un

unico algoritmo generale. Le prove sono state effettuate considerando data set contenenti sia entrambe le features (accelerometro e GPS) sia i valori delle features presi singolarmente. Dall'analisi sul data set completo, ovvero quello che contiene sia i dati relativi all'accelerometro sia quelli relativi al GPS, si può notare dalle seguenti tabelle come variano i risultati di elaborazione in base al classificatore considerato:

| Attività | Tilting | Still | On foot | In Vehicle | On bycicle |
|---|---|---|---|---|---|
| Tilting | 37 | 0 | 2 | 2 | 0 |
| Still | 0 | 199 | 5 | 3 | 0 |
| On foot | 1 | 4 | 481 | 2 | 4 |
| In vehicle | 0 | 9 | 5 | 424 | 1 |
| On bycicle | 0 | 1 | 6 | 5 | 119 |

Tabella 4.1: Confusion Matrix relativa all'algoritmo Random Forest

I risultati ottenuti con l'algoritmo Random Forest hanno una notevole precisione; si può osservare infatti come i valori nella tabella risiedano quasi tutti sulla diagonale, la quale indica appunto i risultati corretti ottenuti. I valori medi relativi alla precisione calcolati dall'elaborazione del Random Forest sono i seguenti:

| TP Rate | FP Rate | Precision | Recall | F-Measure | ROC Area |
|---|---|---|---|---|---|
| 0.962 | 0.015 | 0.962 | 0.962 | 0.962 | 0.998 |

Tabella 4.2: Indici di precisione relativi all'algoritmo Random Forest

La precision (o accuracy) relativa al Random Forest sfiora il 100%, quindi questo algoritmo è il più preciso tra quelli utilizzati per le prove.
La Confusion Matrix relativa ai risultati dell'algoritmo Naive Bayes è la seguente:

| Attività | Tilting | Still | On foot | In Vehicle | On bycicle |
|---|---|---|---|---|---|
| Tilting | 0 | 19 | 2 | 2 | 18 |
| Still | 0 | 196 | 0 | 4 | 7 |
| On foot | 0 | 117 | 4 | 2 | 369 |
| In vehicle | 0 | 348 | 3 | 6 | 82 |
| On bycicle | 0 | 63 | 0 | 0 | 68 |

Tabella 4.3: Confusion Matrix relativa all'algoritmo Naive Bayes

Si nota sin da subito una differenza sostanziale con la Tabella 4.1 dettata dai valori residenti sulla diagonale delle due matrici; nella tabella 4.3 i dati sono molto più distribuiti rispetto alla situazione riportata in Tabella 4.1. Questo ci indica che la precisione di questo classificatore sarà molto minore di quella del Random Forest. Infatti i valori di precisione medi calcolati per il Naive Bayes sono i seguenti:

| TP Rate | FP Rate | Precision | Recall | F-Measure | ROC Area |
|---|---|---|---|---|---|
| 0.209 | 0.124 | 0.365 | 0.209 | 0.1 | 0.718 |

Tabella 4.4: Indici di precisione relativi all'algoritmo Naive Bayes

Come si poteva dedurre anche dalla Confusion Matrix, questo classificatore risulta molto meno preciso del Random Forest, precisamente è circa un terzo meno preciso del precedente algoritmo, con un valore di precision pari a 0.365 (36,5%). Vediamo ora la Confusion Matrix relativa all'algoritmo Bayesian Network:

| Attività | Tilting | Still | On foot | In Vehicle | On bycicle |
|---|---|---|---|---|---|
| Tilting | 0 | 13 | 23 | 5 | 0 |
| Still | 0 | 130 | 7 | 70 | 0 |
| On foot | 0 | 29 | 380 | 83 | 0 |
| In vehicle | 0 | 110 | 62 | 267 | 0 |
| On bycicle | 0 | 16 | 57 | 58 | 0 |

Tabella 4.5: Confusion Matrix relativa all'algoritmo Bayesian Network

Questa matrice ci mostra come l'algoritmo Bayesian Network sia molto preciso solo per alcune delle attività testate (still, on foot ed in vehicle). Gli indici medi di precisione relativi al Bayesian Network sono invece mostrati nella tabella seguente:

| TP Rate | FP Rate | Precision | Recall | F-Measure | ROC Area |
|---|---|---|---|---|---|
| 0.593 | 0.176 | 0.524 | 0.593 | 0.555 | 0.797 |

Tabella 4.6: Indici di precisione relativi all'algoritmo Bayesian Network

Con una precision di 0.524 (52,4%) l'algoritmo Bayesian Network risulta sicuramente più preciso del Naive Bayes ma comunque molto lontano dal Random Forest. La tabella seguente riporta invece i valori della Confusion Matrix relativa all'algoritmo SMO:

| Attività | Tilting | Still | On foot | In Vehicle | On bycicle |
|---|---|---|---|---|---|
| Tilting | 0 | 0 | 41 | 0 | 0 |
| Still | 0 | 0 | 207 | 0 | 0 |
| On foot | 0 | 0 | 492 | 0 | 0 |
| In vehicle | 0 | 0 | 439 | 0 | 0 |
| On bycicle | 0 | 0 | 131 | 0 | 0 |

Tabella 4.7: Confusion Matrix relativa all'algoritmo SMO

Si intuisce immediatamente da questa matrice che l'algoritmo SMO non riesce a distinguere tra le attività considerate, in quanto segnala tutti i test come appartenenti alla classe On foot. Questa Confusion Matrix genera valori medi relativi alla precisione molto bassi:

| TP Rate | FP Rate | Precision | Recall | F-Measure | ROC Area |
|---|---|---|---|---|---|
| 0.376 | 0.376 | 0.141 | 0.376 | 0.205 | 0.5 |

Tabella 4.8: Indici di precisione relativi all'algoritmo SMO

La precision par a 0.141 (14,1%) mostra la poca applicabilità di questo algoritmo come classificatore nel riconoscimento automatico di attività. Di

seguito sono riportati i risultati dei test effettuati con i 2 metaclassificatori Cascading e voting:

| Attività | Tilting | Still | On foot | In Vehicle | On bycicle |
|---|---|---|---|---|---|
| Tilting | 1 | 1 | 29 | 10 | 0 |
| Still | 0 | 57 | 66 | 81 | 3 |
| On foot | 1 | 4 | 422 | 39 | 26 |
| In vehicle | 3 | 30 | 149 | 253 | 4 |
| On bycicle | 0 | 4 | 66 | 47 | 14 |

Tabella 4.9: Confusion Matrix relativa all'algoritmo Cascading

In teoria i metaclassificatori dovrebbero avere una precisione maggiore rispetto a singoli algoritmi per via del funzionamento descritto precedentemente. Nel nostro caso però avremo una eccezione nei confronti del Random Forest, infatti i valori medi relativi alla precisione per l'algoritmo Cascading sono:

| TP Rate | FP Rate | Precision | Recall | F-Measure | ROC Area |
|---|---|---|---|---|---|
| 0,570 | 0,219 | 0,544 | 0,570 | 0,531 | 0,724 |

Tabella 4.10: Indici di precisione relativi all'algoritmo Cascading

Come si vede dalla Tabella 4.10, la precision dell'algoritmo Cascading, con una precision pari a 0.544 (54,4%), è di poco più preciso rispetto al Bayesian Network, molto più preciso di SMO e Naive Bayes ma anche molto meno preciso del Random Forest, che conferma la nostra eccezione. Lo stesso vale per il metaclassificatore Voting, anche se in questo caso troviamo valori di precisione e Confusion Matrix più adatte a quelle che ci attende da un metaclassificatore:

| Attività | Tilting | Still | On foot | In Vehicle | On bycicle |
|---|---|---|---|---|---|
| Tilting | 0 | 14 | 24 | 3 | 0 |
| Still | 0 | 200 | 4 | 1 | 2 |
| On foot | 0 | 34 | 443 | 13 | 2 |
| In vehicle | 0 | 91 | 37 | 311 | 0 |
| On bycicle | 0 | 35 | 14 | 8 | 74 |

Tabella 4.11: Confusion Matrix relativa all'algoritmo Voting

Questa matrice ci mostra dei valori distribuiti in modo abbastanza denso lungo la diagonale, indice che avrà valori medi di precisione abbastanza elevati:

| TP Rate | FP Rate | Precision | Recall | F-Measure | ROC Area |
|---|---|---|---|---|---|
| 0.785 | 0.071 | 0.808 | 0.785 | 0.777 | 0.973 |

Tabella 4.12: Indici di precisione relativi all'algoritmo Voting

Con una precision pari a 0.808 (80,8%), il metaclassificatore Voting si avvicina molto per precisione al Random Forest, quindi si può considerare un buon classificatore utile per il riconoscimento automatico di attività. Nella differenza tra i metaclassificatori e il singolo classificatore Random Forest bisogna tenere conto che nell'esecuzione dei test u metaclassificatore, i valori bassi calcolati con SMO abbassano la precisione totale di tutto il metaclassificatore. Ciò vale sia per Cascading che per Voting. Infine visualizziamo i rusultati dell'elaborazione della libreria di Activity Recognition di Google; questi valori sono stati calcolati ed estratti dal programma Java creato per raffinare i dati di cui si è discusso nel capitolo 2. La Confusion Matrix per l'algoritmo di Google è la seguente:

| Attività | Tilting | Still | On foot | In Vehicle | On bycicle |
|---|---|---|---|---|---|
| Tilting | 0 | 3 | 20 | 11 | 9 |
| Still | 0 | 109 | 3 | 83 | 17 |
| On foot | 0 | 1 | 497 | 3 | 9 |
| In vehicle | 0 | 56 | 2 | 375 | 22 |
| On bycicle | 0 | 5 | 18 | 68 | 42 |

Tabella 4.13: Confusion Matrix relativa all'algoritmo di Activity Recognition di Google

Si può notare subito, che per l'attività *Tilting* non sono state effettuate prove (vedere capitolo 2). In più si nota anche che la precisione cambia sempre da

un'attività all'altra, quindi ci sono attività più semplici da riconoscere con questo algoritmo (on foot ed in vehicle) ed altre che saranno meno semplici da riconoscere, su tutte on bicycle per la quale è più alto il numero di errori di quello dei successi nel riconoscimento. All'interno del programma Java utilizzato per raffinare i dati si è scelto di non calcolare tutti i valori medi come fa Weka per i risultati degli algoritmi di machine learning, ma si è calcolata solo la precision media (o accuracy) il cui valore è 0.526 (52,6%). I risultati mostrati in questo capitolo sono relativi ai test su classificatori effettuati utilizzando il data set contenente tutte e due le features estratte dalla raccolta dei dati. Alcuni test sono stati eseguiti su data set contenenti dati relativi solo a una delle due features con risultati in linea con le aspettative, in quanto concordanti con l'ipotesi, descritta anche in [2], riguardante la precision in relazione al numero di sensori utilizzati: i valori medi relativi alla precisione tendono ad essere più elevati se si utilizzano dati catturati da più sensori rispetto a quelli calcolati tenendo conto di meno dati dei sensori stessi. Nel nostro caso i valori medi relativi alla precisione mostrati in questo capitolo sono maggiorni in tutti i casi rispetto a quelli calcolati utilizzando altri due data set che contengono o solo i dati relativi al sensore dell'accelerometro o solo i dati relativi a quello del GPS. Le Confusion Matrix e le relative tabelle contenenti i valori medi relativi alla precisione per i test effettuati sui data set composti dai dati dei singoli sensori si possono consultare in Appendice A.

# Conclusioni Conclusioni e sviluppi futuri

I risultati esposti nel capitolo 4 ci mostrano le differenze che vanno a crearsi tra l'utilizzo di un algoritmo di machine learning oppure un altro. Non essendo possibile consultare il codice sorgente o anche lo pseudo-codice dell'algoritmo creato da Google per il riconoscimento automatico di attività, questo studio si è focalizzato sulle differenze nei risultati derivanti dall'elaborazione e sulla comparazione dei valori medi che esprimono la precisione del riconoscimento di un dato algoritmo. Per effettuare la raccolta dei dati è stato neccessario sviluppare un'applicazione per la piattaforma Android che fosse in grado di gestire i servizi di connessione con i sensori e con la libreria di Activity Recognition di Google. Un dato importante ai fini dello studio è che l'intervallo di aggiornamento modifica la precisione del riconoscimento di attività, quindi ci dimostra che anche l'algoritmo di Google tende ad imparare dall'esperienza per migliorare la precisione nel riconoscimento. Inoltre si è potuto osservare come l'utilizzo di un numero maggiore di sensori da cui estrarre i dati da dare in pasto ai classificatori migliori anch'esso la precisione media totale del riconoscimento. Gli approcci descritti in questo elaborato si basano tutti su algoritmi di machine learning, grazie ai quali si potrebbe essere in grado di definire pattern di riconoscimento basati su esperienze precedenti in modo da ridurre il peso di elaborazione del sistema. L'approccio basato sull'esperienza si contrappone all'approccio probabilistico il quale utilizza i modelli nascosti di Markov discreti e continui per cercare

di stabilire, basandosi sulla probabilità che l'utente stia svolgendo una data attività, quale sia l'attività in corso nella realtà da parte dell'utente. Sicuramente un sistema solido di riconoscimento automatico di attività potrebbe sfruttare entrambi questi tipi di algoritmi per avere una precisione ancora maggiore di quelle indicate in questo studio. Nel capitolo 4 si è visto come esistano algoritmi di machine learning che garantiscono una precisione che si avvicina alla perfezione basandosi sui circa 1420 rilevamenti di dati effettuati in questo studio. Questi algoritmi, il Random Forest e il metaclassificatore Voting, riportano una precisione media calcolata su tutte le classi (attività da riconoscere) maggiore, rispettivamente, del 41,6% e del 28,2% rispetto alla precisione riportata dall'algoritmo di Google. Questi dati ci indicano senz'altro che il classificatore, scelto dagli sviluppatori di Google e implementato nella libreria per la piattaforma Android utilizzata in questo studio per la raccolta dei dati, non sia sicuramente il classificatore più efficiente da utilizzare per il riconoscimento automatico di attività umane. Questa tesi potrebbe destare obiezioni, ma si basa sul fatto che, in media, l'attività riconosciuta dalla libreria è poco pià della metà delle volte la vera attività che l'utente sta svolgendo in un determinato momento. Questo valore, per alcuni campi di applicazione non sarebbe sufficiente per garantire gli scopi che l'applicazione in particolare dovrebbe raggiungere, mentre una media come quella ad esempio del Random Forest che si avvicina al 97% di successi garantirebbe molto più la stabilità nell'utilizzo del riconoscimento automatico di attività per scopi secondari. Infatti i campi di applicabilità di questo ambito sono molteplici, ad esempio l'impostazione automatica di funzioni sullo smartphone date determinate condizioni, l'automatizzazione di funzioni di computer o elettrodomestici in base all'attività in corso del proprietario, o ancora automatizzazione del controllo del traffico in base ai dati provenienti da sensori di smartphone coadiuvati dal riconoscimento automatico di attività per partire a trasmettere i dati. Per un esempio concreto si pensi al caso del controllo del traffico; se un'applicazione per smartphone inizia a trasmettere i dati relativi allo spostamento solo dal momento che si

accorge che l'utente sta spostando, e se il riconoscimento dello spostamento è corretto solo la metà delle volte, allora tutto il sistema di controllo del traffico avrà una minore credibilità o comunque un minore flusso di dati su cui basare le stime di traffico. Questo esempio mostra come i campi di applicabilità del riconoscimento automatico di attività possano essere utili agli utenti nella loro vita reale, ma al contempo mostra quanto sia importante la precisione nel riconoscimento per prevenire malfunzionamenti o disagi per gli utenti derivanti dall'utilizzo di un sistema. In ogni caso i parametri di precisione dell'algoritmo di Google studiato sono nella media rispetto ad altri algoritmi basati su classificatori come il Bayesian Network o il metaclassificatore Cascading. Questo dato sicuramente ci consente di vedere i margini di miglioramento possibili relativi all'ambito del riconoscimento automatico di attività, miglioramento che porterebbe a maggiori applicazioni implementative e a maggiori campi di utilizzo e di interazione automatica di un sistema con gli utenti. Uno sviluppo futuro per questo studio potrebbe essere il miglioramento dell'applicazione creata per la raccolta dei dati per renderla in grado di essere utile agli utenti in situazioni di mobilità, ad esempio partire dal riconoscimento di attività per mostrare all'utente alternative più efficienti per il viaggio o tempi di percorrenza e servizi simili. Inoltre si potrebbero rendere liberi i dati raccolti durante l'esecuzione dell'applicazione i quali potrebbero servire in altre ricerche ad esempio per l'individuazione di pattern concernenti il riconoscimento automatico di attività.

# Appendice A

# Risultati di elaborazione di algoritmi di machine learning

| Algoritmo | Precision media |
|---|---|
| Bayesian Network | 0.524 |
| Naive Bayes | 0.365 |
| SMO | 0.141 |
| Random Forest | 0.962 |
| Google | 0.526 |
| Voting | 0.808 |
| Cascading | 0,544 |

Tabella A.1: Tabella delle medie della precision degli algoritmi considerati

| Algoritmo | Precision media |
|---|---|
| Bayesian Network | 0.408 |
| Naive Bayes | 0.503 |
| SMO | 0.141 |
| Random Forest | 0.943 |
| Voting | 0.784 |
| Cascading | 0,398 |

Tabella A.2: Tabella delle medie della precision degli algoritmi considerando solo l'accelerometro

| Algoritmo | Precision media |
|---|---|
| Bayesian Network | 0.347 |
| Naive Bayes | 0.286 |
| SMO | 0.141 |
| Random Forest | 0.805 |
| Voting | 0.765 |
| Cascading | 0,482 |

Tabella A.3: Tabella delle medie della precision degli algoritmi considerando solo il GPS

# Bibliografia


[1] Leon Stenneth, Ouri Wolfson, Philip S. Yu, Bo Xu - Transportation Mode Detection using Mobil Phones and GIS information

[2] Luca Bedogni, Marco Di Felice, Luciano Bononi - By Train or By Car? Detecting the User's Motion Type through Smartphone Sensor Data

[3] Singapore-MIT Alliance for Research and Technology - Transportation Activity Analysis Using Smartphones

[4] Sasank Reddy, Min Mun, Jeff Burke, Deborah Estrin, Mark Hansen, Mani Srivastava - Using Mobile Phones to Determine Transportation Modes

[5] Peter Widhalm. Philippe Nitsche, Norbert Brandle - Transportation Mode Detection with Realistic Smartphone Sensor Data

[6] http://it.wikipedia.org/wiki/Dati_aperti

[7] http://it.wikipedia.org/wiki/Matrice_di_confusione

[8] http://www.cs.waikato.ac.nz/ml/weka

# Ringraziamenti

Desidero ringraziare il Dott. Di Felice per il sostegno che mi ha fornito durante tutto il lavoro svolto per il completamento di questa tesi. Un ringraziamento particolare a Silvio Cambiaso, Mauro Melis e Harris Lygidakis di Lumos! per aver messo a disposizione il dispositivo utilizzato per la raccolta dati. Un grazie anche a tutti gli amici che mi hanno supportato durante questi anni. Un ringraziamento speciale alla mia famiglia per i tanti sforzi sostenuti per me.