Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Analisi di Tecniche di Machine Learning per il Keylogging su Smartphone tramite i Sensori Inerziali

Relatore:
Dott.
FEDERICO MONTORI

Presentata da:
GIULIO AUGELLO

Correlatore:
Dott.
LUCA BEDOGNI

Sessione II
Anno Accademico 2020/2021

Sommario

I dispositivi mobili al giorno d'oggi sono tra gli elementi più diffusi e indispensabili per una persona. Questa enorme diffusione può portare a vantaggi e svantaggi. Se è vero che alcune applicazioni possono migliorare la vita quotidiana di un individuo, è anche vero che ci sono applicazioni dannose che possono causare disagi ad un utente, rubando informazioni sensibili, senza che vengano dati particolari permessi. Tramite un *keylogging*, ad esempio, è possibile registrare e leggere qualsiasi cosa l'utente attaccato scriva sulla tastiera. In alcuni articoli viene studiato l'uso dei sensori per il riconoscimento del testo, però vengono usati pochi sensori, ad esempio solo l'accelerometro, e in alcuni casi vengono riconosciuti solo i numeri e non le lettere. In questo elaborato viene proposta:

- la creazione di un *dataset*, partendo dall'acquisizione di dati tramite un'applicazione Android appositamente sviluppata;
- l'analisi e il confronto di sette algoritmi di *machine learning* per la classificazione tramite lettere;
- la creazione di *clusters*, tramite l'algoritmo del *K-Means* con $K = 1...27$, per il raggruppamento delle lettere sulla base dei valori delle *features*;
- la descrizione di un algoritmo per la classificazione delle parole, a partire dalla costruzione di grafi, e la rimozione dalla classifica di quelle non esistenti all'interno di un dizionario inglese.

Introduzione

Al giorno d'oggi i dispositivi mobili sono un elemento sempre più indispensabile e presente nella vita quotidiana. Questi hanno subito una notevole evoluzione nel tempo, sia hardware che software, così da permettere il loro utilizzo per moltissimi scopi. Infatti, tramite le applicazioni è ormai possibile svolgere sia attività di tutti i giorni che azioni mirate a specifici campi, soprattutto con l'utilizzo di dispositivi che hanno una notevole potenza di calcolo, ma anche con quelli meno potenti. Ad esempio, nel campo dell'*Activity Recognition* viene studiato proprio come usare i dati restituiti dagli smartphone per riconoscere le attività di un utente, come la corsa, il jogging o anche l'azione di salire e scendere le scale. La creazione di queste applicazioni, oltre a semplificare molte azioni quotidiane, potrebbero però mettere a rischio dati personali sensibili, come *password* o numero di carte di credito. Ad esempio, potremmo ottenere questi dati tramite un attacco *spoofing* oppure tramite un *keylogging*, dove nel primo caso chi effettua l'attacco simula l'identità di un mittente per poter intercettare informazioni sensibili, nel secondo caso, invece, viene letto tutto ciò che l'utente attaccato scrive sulla tastiera senza che quest'ultimo se ne accorga.
Tuttavia, in questo elaborato verrà considerato un tipo di attacco differente. Si basa sul fatto che tutti i dispositivi mobili hanno dei sensori al loro interno, che possono essere ad esempio accelerometro, giroscopio, magnetometro. Tramite questi sensori è possibile registrare gli spostamenti del dispositivo nel tempo, anche senza la necessità di chiedere determinati permessi all'utente. Per questo scopo è stato richiesto l'utilizzo del dispositivo con una

sola mano, così che l'utente debba muovere maggiormente il dispositivo per arrivare da un'estremità all'altra. Inoltre, questo è stato necessario per una duplice analisi, cioè per differenziare i risultati ottenuti dagli utenti che usano la mano destra da quelli che usano la mano sinistra. Lo scopo di questo elaborato è stato dunque quello di controllare la possibilità di riconoscimento di ciò che un utente scrive sullo smartphone partendo proprio dai dati acquisiti dai sensori del dispositivo. Questo è possibili dall'analisi dei dati da parte di un modello di *machine learning*, addestrato esclusivamente per tale scopo.
Lo sviluppo del progetto di tesi è stato suddiviso in cinque parti. La prima parte si occuperà della descrizione dell'implementazione di un applicazione *Android* per l'acquisizione dei dati restituiti dai sensori del dispositivo. Nella seconda parte verrà analizzato come sono state calcolate le *features* che verranno usate dai vari modelli di *machine learning*, e come sono stati costruiti i due *dataset*, il primo per la mano destra e il secondo per la mano sinistra. La terza parte, invece, tratterà l'analisi delle *features* e dei sette modelli di *machine learning*, che verranno messi a confronto tramite il calcolo delle *accuracy*, la creazione di *confusion matrix* e il calcolo di *Precision, Recall e F1 Score*. La quarta parte, simile alla terza, tratterà l'utilizzo dell'*unsupervised learning* per il raggruppamento di lettere simili sulla base dei valori delle *features*, così da poter verificare il cambiamento di accuratezza dei modelli in base al numero di classi utilizzate. Infine, la quinta ed ultima parte si occuperà di costruire una classifica di possibili parole tramite la creazione di grafi, ottenuti dalle *clusterizzazioni* calcolate nella quarta parte. Tornerà utile per vedere dove si posiziona la parola reale rispetto alla parola predetta dal classificatore.
Infine, visitando il seguente indirizzo:

`https://github.com/giulioaugello/Dataset-Code-Text_Recognition`

è possibile scaricare l'intero progetto discusso in questo elaborato, i due *dataset* e tutti i grafici generati dall'inizio dello sviluppo del progetto e discussi in questo elaborato.

Indice

Elenco delle figure

Elenco delle tabelle

Listings

Capitolo 1

Stato dell'arte

1.1 Machine Learning

Il *Machine Learning*, o *Apprendimento Automatico*, è una branca dell'informatica che comprende metodologie di analisi di dati in grado di fornire come risultato l'identificazione di modelli, schemi o tendenze, insegnando a uno strumento software come potrebbe risolvere un determinato problema senza che questo venga programmato completamente. Oggigiorno, viene utilizzato in molti ambiti tra cui quello della medicina, dove, ad esempio, in [17] si è vista l'accuratezza di tecniche di machine learning per la diagnosi del cancro al seno, nella sicurezza informatica, come in [21] dove viene spiegato come è possibile utilizzare il machine learning per l'analisi di rete e il rilevamento di intrusioni nel sistema, può essere utilizzato per riconoscere e limitare il cyberbullismo, come spiegato in [14] dove abbiamo un'accuratezza del 78.5% sul riconoscimento di post che contengono cyberbullismo, o anche nell'ambito dell'automazione per la guida autonoma, ad esempio in [7] viene dimostrato come la sterzata e la velocità del veicolo danno importanti informazioni per le previsioni di comportamenti di guida in alcuni segmenti di rotatoria.

Alla base di ogni problema di machine learning troviamo sempre una grande quantità di dati, che deve essere raccolta per riuscire ad allenare

l'algoritmo utilizzato. Questi dati verranno utilizzati per creare un *dataset*, o *training set*, cioè una matrice dove ogni colonna contiene le variabili di input (*features*), che possono essere sia numeri che valori predefiniti come 'uomo' e 'donna', e ogni riga contiene un esempio di training (*training examples*), dove ognuno di questi rappresenta un'entità che vogliamo analizzare [2]. In base al problema che dobbiamo risolvere possiamo applicare tre tecniche differenti di machine learning:

- *Supervised Learning*
- *Unsupervised Learning*
- *Reinforcement Learning*

In base alla tecnica utilizzata il *dataset* può essere suddiviso in diversi modi. Se utilizziamo il *supervised learning* il dataset viene suddiviso in due, la prima parte contiene le colonne che hanno le variabili di input, dette anche *features*, e la seconda parte contiene una colonna con l'output reale, detti *labels*. Ad esempio, la colonna di output non serve se utilizziamo l'*unsupervised learning*.

In questo elaborato verranno spiegati solamente *Supervised Learning* e *Unsupervised Learning*, in quanto argomenti del progetto sviluppato.

1.1.1 Supervised Learning

Gli algoritmi di *Supervised Learning* consistono nella definizione di una funzione in grado di associare ad ogni input un output corretto, questa previsione viene fatta sulla base di coppie *(input,output)* che vengono fornite al sistema per l'apprendimento del problema [20]. Il *dataset* iniziale viene suddiviso in due insiemi di dati, il *training set* ed il *test set*. L'idea è quella che l'algoritmo "impari" dal training set, che contiene sia le *features* che le *labels* (cioè l'output corrispondente), così da poter identificare gli esempi all'interno del test set, privi di *labels*, nel miglior modo possibile. Quindi nel *supervised learning* il *training set* è composto da n coppie (x_1,y_1), (x_2,y_2), ... , (x_n,y_n),

dove x_i è una misura o un insieme di misure di un singolo esempio, e y_i è l'etichetta corrispondente [9].

I problemi di *supervised learning* sono suddivisi in:

- *Classification Problem*
- *Regression Problem*

1.1.2 Classification Problem

Nei problemi di classificazione la funzione cerca di mappare le variabili di input in variabili di output discrete. Le variabili di output vengono spesso chiamate *labels*, *categories*, *etichette* o anche *classi* e la funzione le predice in base a input dati. Un esempio di problema di classificazione è classificare una email come "spam" o "non spam" [1]. Gli algoritmi di *supervised learning* che si occupano maggiormente di problemi di classificazione, nonché quelli usati nel progetto di tesi, sono [18]:

Random Forest (RF): è un metodo d'insieme che funziona addestrando un numero di alberi decisionali (k) e restituendo la classe con la maggioranza su tutti gli alberi nell'insieme, cioè ogni nodo dell'albero rappresenta una congiunzione di features e le foglie sono le etichette di output della classificazione. Viene molto usato nei problemi di classificazione in quanto molto veloce, scalabile, non produce over fit ed è semplice da interpretare e visualizzare non avendo molti parametri da maneggiare.

Gaussian Naive Bayes (GaussianNB): è un Bayesian Network con un solo genitore e molti figli con una forte assunzione di indipendenza tra i nodi figli. Se questa assunzione è valida, questo tipo di classificatore converge più velocemente dei modelli discriminativi (come Logistic Regression).

Support Vector Machine (SVC): si basa sul concetto di massimizzare la distanza minima dall'iperpiano, cioè un sottospazio la cui dimensione è

inferiore di uno allo spazio che circonda l'oggetto, al training example più vicino di una classe. È robusto a grandi quantità di dati ma la velocità di training è più lenta e le performance dipendono dalla scelta dei parametri.

Decision Tree (CART): questi algoritmi usano l'approccio *divide et impera* che funziona bene se esistono pochi attributi altamente rilevanti ma non molto bene se sono presenti molte interazioni complesse. Man mano che l'albero cresce, il numero di record nei nodi foglia potrebbe essere troppo piccolo per prendere decisioni statisticamente significative sulla rappresentazione della classe.

Ada Boost Classifier (ADA): crea una raccolta di classificatori di componenti mantenendo una serie di pesi sui training example e regolando questi pesi dopo ogni iterazione di Boosting: i pesi degli esempi che sono classificati erroneamente dal classificatore verranno aumentati mentre i pesi degli esempi classificati correttamente saranno diminuiti [10].

Logistic Regression (LR): sono modelli statistici in cui una curva logistica è adattata al dataset. Questa tecnica viene applicata quando la variabile dipendente o quella target è dicotomica, bipartita.

K Nearest Neighbor (KNN): assegna ad un training example senza etichetta, la più vicina classe di un insieme di punti precedentemente etichettati. È adatto per applicazioni in cui l'oggetto può avere molte etichette. Le performance dipendono dalla scelta del parametro k, ma non esiste un modo certo per scegliere quest'ultimo, se non tramite tecniche computazionalmente costose come il *cross validation*.

1.1.3 Regression Problem

Nei problemi di regressione la funzione cerca di mappare le variabili di input in variabili di output continue. Una variabile di output continua è un valore reale, queste sono spesso quantità o dimensioni. Un esempio di

problema di regressione è la predizione del valore di una casa in vendita, ad esempio in un range tra i 100,000 dollari e i 200000 dollari [19].

1.1.4 Unsupervised Learning

Gli algoritmi di *Unsupervised Learning* affrontano i problemi in modo differente. Infatti, alcune volte non sappiamo le *labels* reali del problema, oppure vogliamo semplicemente esaminare i modelli che emergono naturalmente dai dati. In altre parole, non abbiamo il vettore delle *labels* ma solo un *dataset* di *features* dove possiamo estrarre una struttura comune. Uno dei metodi più utilizzati per questo scopo è il *Clustering*.

1.1.5 Clustering

L'obiettivo del *Clustering* è di identificare sottogruppi rilevanti in un *dataset* senza avere un'ipotesi sulle proprietà che potrebbero avere i sottogruppi. Un *cluster* è un sottoinsieme di dati che sono simili tra di loro. Uno dei possibili approcci al *clustering* è il *K-means* [2].

K-means

Nel *K-means* Il numero di *cluster* da identificare viene specificato dal programmatore inserendo il parametro K. Ogni cluster è rappresentato da un *cluster center* (o *centroid*), che identifica un *training example* "artificiale" e che rappresenta la media (o la mediana) di tutti gli elementi all'interno del suo *cluster*. All'inizio i *centroid* vengono inseriti in modo casuale tra tutti gli elementi, successivamente l'algoritmo itera su due passaggi:

1. **Assegnamento**: ogni elemento viene assegnato al *centroid* più vicino
2. **Spostamento dei *centroid***: la posizione di ogni *centroid* viene aggiornata dopo aver fatto il primo passaggio.

Di solito riesce a convergere all'ottimo locale, dove l'assegnamento dei *centroid* non cambia o cambia marginalmente.

1.2 Applicazioni Mobili

Le *Applicazioni Mobili*, o semplicemente *app*, sono applicazioni software dedicate a dispositivi come smartphone o tablet. Avendo risorse hardware limitate, vengono progettate in modo più leggero rispetto ad un'applicazione per computer ed in linea con le restrizioni imposte dai vari produttori dei dispositivi. Al contrario di un normale software applicativo le applicazioni sono limitate, in quanto mirate ad una determinata funzione. Le app, infatti, vanno ad ampliare le capacità native del dispositivo incluse all'interno del sistema operativo. Le applicazioni vengono classificate in tre tipi differenti:

- **App Native**: consiste in un software che si installa e si utilizza interamente sul proprio dispositivo mobile e viene creata appositamente per uno specifico sistema operativo.
- **Wep App**: è sostanzialmente un collegamento verso un applicativo remoto, scritta in un linguaggio cross-platform. Il codice dell'interfaccia utente può risiedere sia sul dispositivo mobile che, anch'esso, sull'applicativo remoto. Il vantaggio di una web app è che non incide in alcun modo sulle capacità di memoria e di calcolo del dispositivo, in quanto alcune parti possiamo averle su un server remoto; lo svantaggio è che richiede il costante accesso ad internet, quindi le prestazioni dipenderanno dalla velocità di connessione.
- **App Ibride**: sono applicativi mobili che sfruttano sia componenti nativi sia tecnologie web. L'utilizzo delle tecnologie web avviene tramite il componente WebView.

1.2.1 Sensori negli smartphone

I *Sensori* sono, per definizione, dei dispositivi che si trovano in diretta interazione con il sistema misurato. Al giorno d'oggi la maggior parte dei dispositivi mobili sono dotati di sensori che riescono a misurare il movimento, l'orientamento e varie condizioni ambientali. Questi sono capaci di fornire

dati grezzi (*raw data*) con alta precisione e accuratezza, e sono molto utili se si vuole monitorare il movimento o la posizione tridimensionale del dispositivo. Possiamo suddividere i sensori in tre grandi categorie:

- **Sensori di Movimento**: questi misurano le forze di accelerazione e le forze di rotazione lungo tre assi. Questa categoria include quasi tutti i sensori che verranno successivamente citati in questo elaborato, cioè l'accelerometro, il giroscopio e i sensori di gravità.

- **Sensori Ambientali**: questa categoria include quei sensori che misurano vari parametri ambientali, come la temperatura e la pressione dell'aria nell'ambiente, l'illuminazione e l'umidità. Troviamo sensori come il barometro o il termometro.

- **Sensori di Posizione**: questi sensori, infine, misurano la posizione fisica di un dispositivo. Questa categoria include i sensori di orientamento e il magnetometro, anch'essi oggetto di studio all'interno di questo elaborato.

I sensori che vengono maggiormente utilizzati nelle applicazioni sono:

- **Accelerometro:** misura la forza di accelerazione in m/s^2 che viene applicata al dispositivo.

- **Giroscopio:** misura la velocità di rotazione di un dispositivo in rad/s attorno a ciascuno dei tre assi fisici.

- **Gravità:** misura la forza di gravità in m/s^2 applicata ad un dispositivo.

- **Magnetometro:** misura il campo magnetico dell'ambiente.

Applicazioni dei sensori

Grazie ai sensori sono state sviluppate varie applicazioni nel tempo. Possiamo vedere ad esempio in [11] come si è studiata la fattibilità dell'utilizzo

di vecchi smartphone e dei relativi sensori per costruire un sistema di sicurezza domestica per case non troppo grandi. Infatti, viene spiegato come gli eventi relativi alle porte, come l'apertura e la chiusura, producono delle vibrazioni uniche rispetto alle vibrazioni dell'ambiente e possono essere catturate tramite l'accelerometro di un telefono posto su un muro vicino alla porta; è possibile catturare anche la rotazione della porta utilizzando il magnetometro di uno smartphone posto su di essa. Per rilevare questi eventi sono stati "allenati" due modelli di machine learning in base ai dati registrati dai sensori citati precedentemente, un modello per ogni sensore. In seguito è stato creato un prototipo di sistema di sicurezza domestica che rileva l'apertura della porta e invia una notifica al proprietario di casa via email, SMS e chiamata al cellulare. I risultati mostrano che l'apertura di una porta può essere rilevata sia dai dati dell'accelerometro sia da quelli del magnetometro, con una accuratezza del 98%, o superiore, e nel caso dell'accelerometro riesce anche a rilevare l'apertura di una finestra con un elevata accuratezza.
Nell'articolo [15] si è cercato un modo alternativo all'utilizzo del convenzionale GPS, che consuma, in termini di batteria, molto più dei sensori presenti all'interno di uno smartphone e non è efficace in ambienti chiusi. Per ovviare a questi problemi, hanno creato un sistema di *location tracking* rapido e a bassa potenza per smartphone che sono in grado di tracciare posizioni continue con una buona precisione. La distanza coperta da due istanti temporali viene calcolata leggendo i dati dell'accelerometro, mentre con il magnetometro viene letta la direzione di movimento dello smartphone. Questi sensori vengono anche utilizzati per prendere una stima iniziale del numero di passi effettuati e della media della falcata della persona che sta usando il dispositivo. Il modello di base è stato calcolato usando il cambiamento di accelerazione dovuto alla gravità quando viene fatto un passo. Questi valori vengono quindi utilizzati per trovare continuamente la posizione di una persona che trasporta il dispositivo. Come risultati hanno ottenuto un'accuratezza di posizione tra i 2 e i 5 metri, sia in luoghi chiusi che in quelli aperti, e c'è stato un grande risparmio di batteria, più del 20% su una corsa di tre ore.

1.3 Activity Recognition

L'*Acitivity Recognition*, o riconoscimento dell'attività, mira a riconoscere le azioni e gli obiettivi di uno o più agenti da una serie di osservazioni sulle azioni di questi ultimi e sulle condizioni ambientali. Diversi campi possono riferirsi all'activity recognition come: riconoscimento del comportamento, stima della posizione, o servizi basati sulla posizione. Esistono vari tipi di activity recognition:

Sensor-based, single-user activity recognition: integra nel campo *sensor networks* nuove tecniche di data mining e machine learning per modellare un'ampia gamma di attività umane.

Sensor-based, multi-user activity recognition: le attività di più utenti in "ambienti intelligenti" sono state affrontate in [8] dove hanno investigato sui problemi fondamentali per il riconoscimento delle attività per più utenti tramite la lettura di sensori in ambiente domestico e hanno proposto un pattern per riconoscere le attività sia di un singolo utente che di più utenti.

Sensor-based group activity recognition: in questa categoria viene data importanza al riconoscimento del comportamento del gruppo come entità, piuttosto che il comportamento dei singoli membri al suo interno. Infatti, le proprietà del comportamento del gruppo sono diverse da quelle dei singoli individui o dalla somma di essi.

Come già detto in precedenza, molti campi fanno riferimento all'activity recognition tra cui il *context recognition*. Come è possibile vedere in [12] vengono usati i sensori dello smartphone e algoritmi di classificazione per identificare attività svolte da un utente e il contesto. Gli algoritmi utilizzati per questo scopo sono stati il k-NN, RandomForest e MLP che hanno dato degli ottimi risultati, rispettivamente 98.77%, 98.67% e 98.1%. Si è visto come l'uso di dati acquisiti da più sensori, in particolare combinando i dati del sensore di orientamento e di rotazione con l'accelerometro, può migliorare le

performance del classificatore dall'1.5% al 5%. Inoltre, si afferma che alcuni classificatori non riescano effettivamente a riconoscere attività specifiche che hanno pattern di segnali periodici simili. Ad esempio alcuni algoritmi confondono attività come jogging, corsa e camminata, probabilmente perché i pattern di questi segnali cambiano leggermente per utenti differenti.
In [16], invece, si è studiato il ruolo dell'accelerometro, del giroscopio e del magnetometro all'interno dell'activity recognition. Sono stati valutati i loro ruoli su quattro posizioni del corpo (tasca destra dei jeans, cintura, braccio destro e polso destro) utilizzando sette classificatori e riconoscendo sei attività (camminare, correre, sedersi, stare in piedi, salire e scendere le scale). Viene mostrato che, in generale, l'accelerometro e il giroscopio si completano a vicenda, quindi rendendo il riconoscimento più affidabile, e che il giroscopio ha ottime performance anche quando utilizzato da solo. Al contrario, per quanto riguarda il magnetometro non si hanno avuti buoni risultati. Infatti, il magnetometro è causa di over-fitting nei classificatori di training a causa della sua dipendenza dalle direzioni. Gli autori hanno concluso che l'attività di *salire le scale* viene riconosciuta meglio dal giroscopio, ad eccezione della posizione sulla cintura, mentre l'attività di *stare in piedi* viene riconosciuta meglio dall'accelerometro, in tutte le posizioni del corpo. In generale, però, è difficile fare un'affermazione esatta sul ruolo dei sensori per il riconoscimento delle attività citate perché molto dipende dalla posizione dello smartphone, dal l'attività da riconoscere e dalla scelta del classificatore.

1.3.1 Keylogging

Con *Keylogging*, o *keystroke logging*, si intende l'attività di intercettare e catturare segretamente tutto ciò che viene digitato sulla tastiera senza che l'utente se ne accorga. Possiamo effettuare un keylogging tramite un **keylogger**, cioè uno strumento che riesce ad effettuare la registrazione della tastiera di un computer. Spesso i keylogger sono trasportati e installati nel computer tramite *worm* o *trojan* ricevuti tramite internet e hanno lo scopo

di intercettare password o numeri di carte di credito e inviarle tramite posta elettronica al creatore degli stessi. In generale esistono due tipi di keylogger:

Hardware: è un dispositivo che viene collegato al cavo di comunicazione tra la tastiera ed il computer o all'interno della tastiera.

Software: è un programma che controlla e salva la sequenza di tasti che viene digitata da un utente su un computer.

A differenza di altri programmi malevoli, i keyloggers non causano alcuna minaccia al sistema, ma possono essere usati per intercettare password ed altre informazioni confidenziali immesse tramite la tastiera. In questo modo è possibile impersonare un utente durante, ad esempio, una transazione bancaria. Per prevenire il keylogging bisogna dare autenticazione in modo rigoroso ai programmi o alle applicazioni.
Ad esempio in [6] viene spiegato come un QR code può essere usato per progettare i protocolli di autenticazione visiva, cioè il protocollo One-Time-Password (OTP) e il protocollo di autenticazione basato su Password, per ottenere un'elevata usabilità e sicurezza. Da accurate analisi, si è visto che i protocolli risultano essere robusti a molti attacchi, come keylogging o altri attacchi malware. Implementando questi protocolli in un'applicazione, specialmente in quelle con transazioni online, è possibile ottenere dei requisiti di sicurezza soddisfacenti.
La minaccia del keylogging è stata ampiamente studiata per i sistemi dei computer ma non così tanto per quelli dei dispositivi mobili. Infatti, al giorno d'oggi sono oltre tre miliardi i dispositivi attivi che hanno Android come sistema operativo (dati riportati nel *Google I/O 2021*). Android è conosciuto per la sua flessibilità e personalizzazione, ma è anche vero che questo potrebbe causare problemi. Infatti una semplice tastiera sviluppata da terze parti potrebbe contenere parti di codice malevolo, come appunto un keylogger. Nell'articolo [5] sono stati analizzati i comportamenti di sicurezza di 139 applicazioni per tastiera disponibili su Google Play e i risultati dimostrano che la maggior parte delle tastiere prese in esame, 84 su 139, potrebbe esse-

re utilizzato come un keylogger. Per utilizzare questi keyloggers basterebbe anche solo il permesso per INTERNET.

1.4 Motivazioni

Come abbiamo visto, tramite l'*Activity Recognition* è possibile riconoscere ciò che un individuo sta facendo tramite il suo smartphone. Utilizzando assieme activity recognition e algoritmi di machine learning è possibile avere grandi problemi per quanto riguarda la privacy e la segretezza dei dati, senza che l'utente dia effettivamente dei permessi specifici. Basterebbe dare anche solo i permessi di Internet per poter installare un keylogger perfettamente funzionante. Negli articoli [13] e [4] possiamo vedere come vengono utilizzati i sensori dello smartphone per dedurre cosa si sta scrivendo, nel primo articolo viene utilizzato l'accelerometro per capire la password di un utente, nel secondo vengono utilizzati accelerometro e sensore di orientamento per capire quali numeri vengono premuti.
Scopo del progetto di tesi è stato quello di studiare la possibilità di riconoscere il testo scritto da un individuo partendo dai dati acquisiti dai sensori dello smartophone, non solo dall'accelerometro ma anche, ad esempio, dal giroscopio e dal magnetometro, applicando algoritmi di *supervised learning*. Per questo scopo è stato creato un dataset usando un'applicazione Android, appositamente creata, dove l'utente riscrive delle frasi che appaiono sullo schermo mentre vengono registrati i sensori. Successivamente è stata calcolata l'accuratezza di 7 algoritmi di machine learning differenti, dove *Random Forest* e *SVC* hanno ottenuto una percentuale superiore rispetto agli altri.
Inoltre, è stato utilizzato anche l'*unsupervised learning*, con l'algoritmo *K-means* con $K = 1...27$, per raggruppare le lettere in *clusters* sulla base dei valori di ogni *features*. Ogni *clusterizzazione* verrà usata, in primis, come *labels* per gli algoritmi di *supervised learning* per vedere quanto aumentasse l'accuratezza di ciascun algoritmo al diminuire delle *labels* e successivamente per valutare la realizzazione di un attacco dizionario tramite un algoritmo

basato su grafi di probabilità, dove ogni cammino radice-foglia corrisponderà ad una parola, a cui sarà attribuita una probabilità. La probabilità della parola predetta verrà messa a confronto con la probabilità della parola reale, per capire la lontananza di quest'ultima dalla previsione del modello.

1.4.1 Architettura del progetto

Nella prossima pagina verrà mostrato uno schema dell'architettura del progetto di tesi, partendo dallo sviluppo dell'applicazione Android con la seguente creazione dei file csv, passando per il calcolo di Magnitude, Roll e Pitch con la successiva creazione delle *features* che formano il *Dataset* finale, suddividendo quest'ultimo in:

- ***Supervised Learning***: con la classificazione per lettera.
- ***Unsupervised Learning***: che è composto da:
  - *Supervised Learning* con la classificazione per *clusters*.
  - Grafi delle parole con le probabilità di ogni lettera

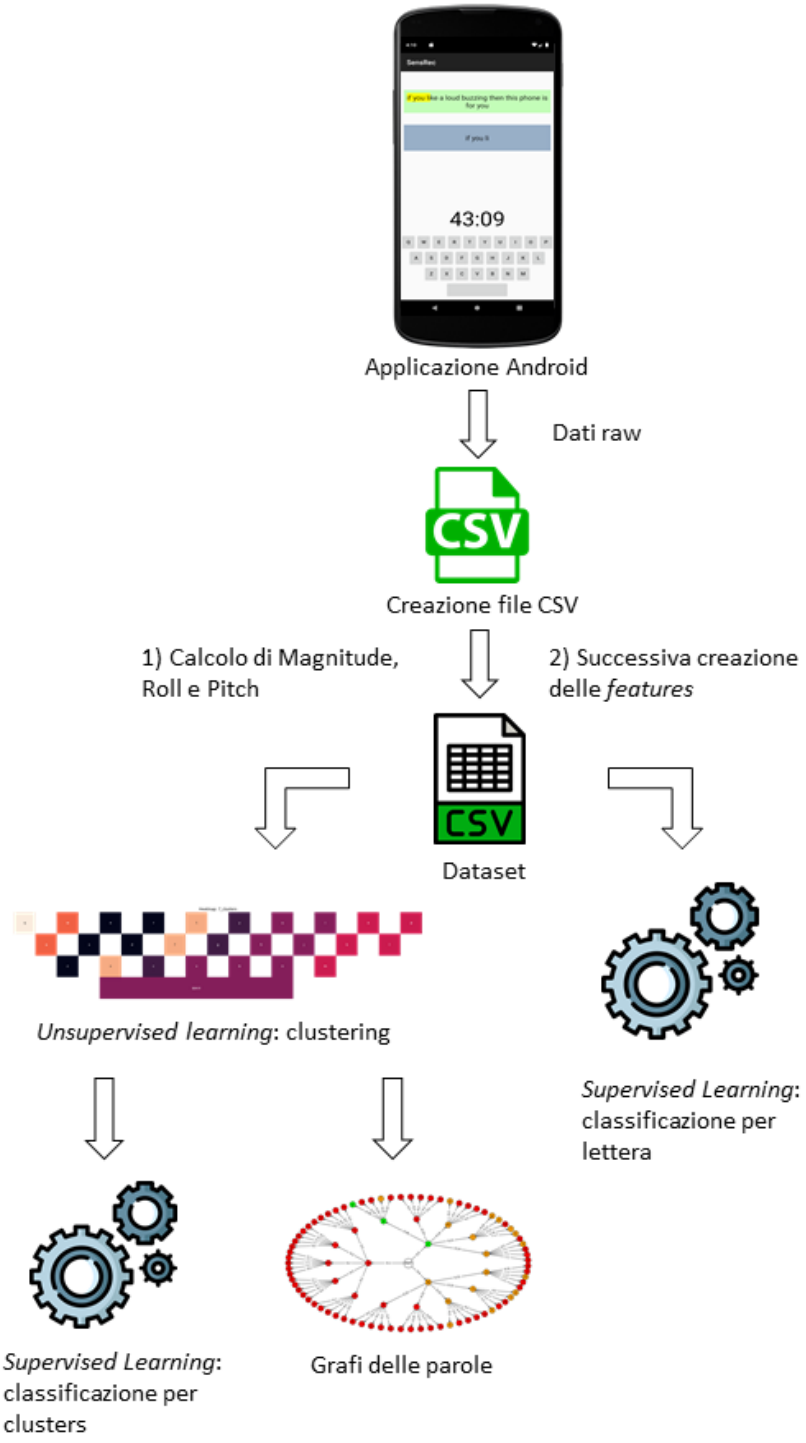


Figura 1.1: Architettura del progetto

Capitolo 2

Applicazione

2.1 Descrizione

L'applicazione è stata sviluppata per dispositivi Android con versione dalla 6.0 (*API level 23*) alla 11 (*API level 30*). Si occupa della raccolta dei dati riguardanti i sensori installati sugli smartphone. I sensori presi in considerazione sono: Accelerometro, Giroscopio, Magnetometro, Gravità e Orientamento.
Lo scopo è quello di raccogliere i dati dei sensori durante eventi *touch*. Per fare ciò, l'utente dovrà scrivere delle frasi che appariranno sulla parte alta dello schermo, tramite una tastiera creata appositamente, in un determinato tempo, stabilito da un *countdown* e calcolato in relazione alla lunghezza della frase da immettere, e che partirà una volta premuta la prima lettera. Se la frase non viene scritta entro il tempo indicato il test verrà rifiutato e non preso in considerazione. Durante la digitazione, se l'utente sbaglia nella scrittura di una lettera gli verrà chiesto di reinserire la parola che stava scrivendo dalla prima lettera.
Dal momento che l'utente tocca la prima lettera parte il *countdown* e inizia la registrazione dei sensori. Quest'ultima viene effettuata su un file *CSV* che viene creato nella seconda Activity, sul dispositivo. Una volta finito il test, il file viene salvato sullo *storage* di *Firebase*, dove è possibile salvare molti file

(5GB in totale) e gestirli tramite la *Google Cloud SDK Shell*.
Se l'applicazione viene definitivamente chiusa durante il test, il file *csv* sarà cancellato dal dispositivo, se invece viene messa solo in background il file sarà cancellato e, una volta riaperta l'applicazione, verrà ricreato per permettere all'utente di scrivere la frase dall'inizio.
Ogni utente che si è offerto per eseguire i test ha dovuto totalizzare almeno cento eventi *touch* per ogni lettera. Per controllare l'avanzamento di questi eventi è stato creato in un *OptionMenu* un *Dialog* dove sono riportati i contatori di ogni lettera. I test verranno svolti utilizzando una singola mano per fare in modo che il dispositivo si muova maggiormente quando bisogna premere lettere lontane.
Per quanto riguarda la struttura dell'applicazione, questa è composta da due *activity*:

Activity Principale: qui vengono raccolte informazioni sull'utente che saranno utili per la creazione dei file e per la successiva analisi;

Activity SensorEventListener: in questa activity, invece, avviene il vero e proprio test.

Entrambe verranno approfondite nelle sezioni successive.

2.2 Activity Principale

L'activity principale è la *MainActivity* che, per prima cosa, controlla se è stato già dato il permesso *WRITE_EXTERNAL_STORAGE*, indispensabile per poter creare il file *csv* sul dispositivo dove verranno scritti i dati grezzi dei sensori e che verrà successivamente inviato allo *storage* di *Firebase*.
Una volta dato il permesso, troviamo una schermata formata da:

***EditText* Username:** dove si inserisce il nome della persona che sta facendo i test. Se un Username era già presente e si inizia un nuovo test con un altro nome allora tutti i contatori delle lettere vengono riportati a 0 automaticamente.

***EditText* Smartphone Name:** dove si inserisce il nome del dispositivo che si sta utilizzando.

***RadioGroup*:** formato da due *RadioButton* per indicare la mano che si sta utilizzando.

Basterà completare questi campi una sola volta in quanto le informazioni verranno salvate tramite le *SharedPreferences*. La schermata sarà come quella che segue:

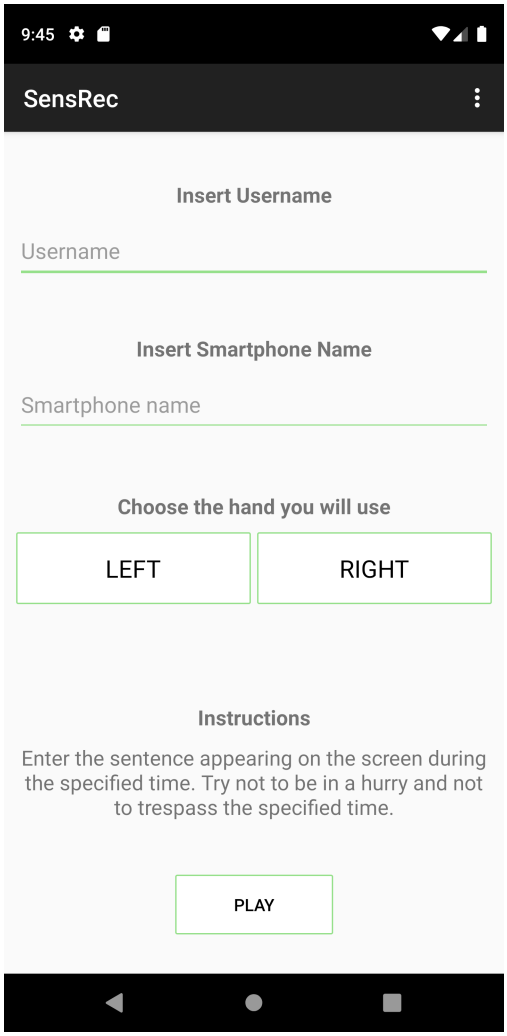


Figura 2.1: MainActivity

Ogni volta che viene premuto un *RadioButton* viene chiamata la funzione *randomSentence* che ritorna la frase che verrà utilizzata per fare il test. Questa viene presa da un file (*TextSentences.txt*), contenente un totale di 45 frasi, nel modo seguente:

```
br = new BufferedReader(new InputStreamReader(getAssets().
    open("TextSentences.txt")));
line = br.readLine();
```

Listing 2.1: BufferReader del file TextSentences.txt

Le frasi sono state scelte in modo da avere un minimo di occorrenze per ogni lettera. Alcune di queste, 30 su 45, sono state scelte da un database consultabile all'url `https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences` in particolare dal file *amazon cells labelled.txt*, mentre le restanti quindici sono pangrammi, cioè frasi di senso compiuto in cui vengono utilizzate tutte le lettere dell'alfabeto (uno dei più famosi pangrammi è *The quick brown fox jumps over the lazy dog*). Una volta completati gli *EditText* e selezionato un *RadioButton*, il *Button Play* si sbloccherà. Premendo quest'ultimo compare un *Dialog* che chiede se iniziare il test oppure tornare indietro per modificare le informazioni. Se si tocca *OK* per iniziare, viene creato un *Intent*:

```
Intent data = new Intent("com.tesi.sensrec");
data.putExtra("User", User);
data.putExtra("hand", Hand);
data.putExtra("sentence", line);
startActivity(data);
```

Listing 2.2: Intent in MainActivity

dove andiamo ad inserire le informazioni raccolte come l'username, la mano da utilizzare e la frase da scrivere, che serviranno per la creazione del file *csv*, e che ci porterà alla seconda *activity*.

2.3 Activity SensorEventListener

La seconda *activity* è la parte fondamentale dell'applicazione. Infatti, implementa l'interfaccia *SensorEventListener* e si occupa: della registrazione dei valori dei sensori acquisiti durante la scrittura del testo, della creazione del file *csv* e del suo invio sullo *storage* di *Firebase*.

2.3.1 Metodi di Inizializzazione

All'interno dell'*onCreate* vengono inizializzate alcune variabili che verranno utilizzate all'interno dell'activity attuale e chiamati i metodi fondamentali per l'inizializzazione delle sue funzionalità. In particolare ci sono quattro metodi che svolgono questa funzione:

setData(): da questo metodo vengono recuperati i dati dell'Intent tramite il metodo *getIntent().getStringExtra(String name)*

```
1 private void setData() {
2     User = getIntent().getStringExtra("User").trim();
3     testSentence= getIntent().getStringExtra("sentence");
4     Hand = getIntent().getStringExtra("hand");
5 }
```

Listing 2.3: Metodo setData()

setView(): viene caricato il layout dell'activity dal file *test.xml* e viene anche creato il *countdown* in base alla lunghezza della frase recuperata dal metodo *setData()*.

createNewFileCSV(): in questo metodo viene creato il file *csv* all'interno del dispositivo. Il nome del file sarà formato così:

```
1 filename = User +  "-" + upperSmartphone +  "-" + Hand +
     "-" + time + "-" + testWord + ".csv";
```

Listing 2.4: Struttura del nome del file *csv*

che identificano rispettivamente: username, nome del dispositivo, mano che si utilizza per fare il test, il tempo corrente (tramite la funzione *System.currentTimeMillis()*) e la frase da scrivere.
Sempre in questo metodo viene scritto l'*header* del file csv che verrà spiegato con maggiore precisione nella sezione 2.3.3.

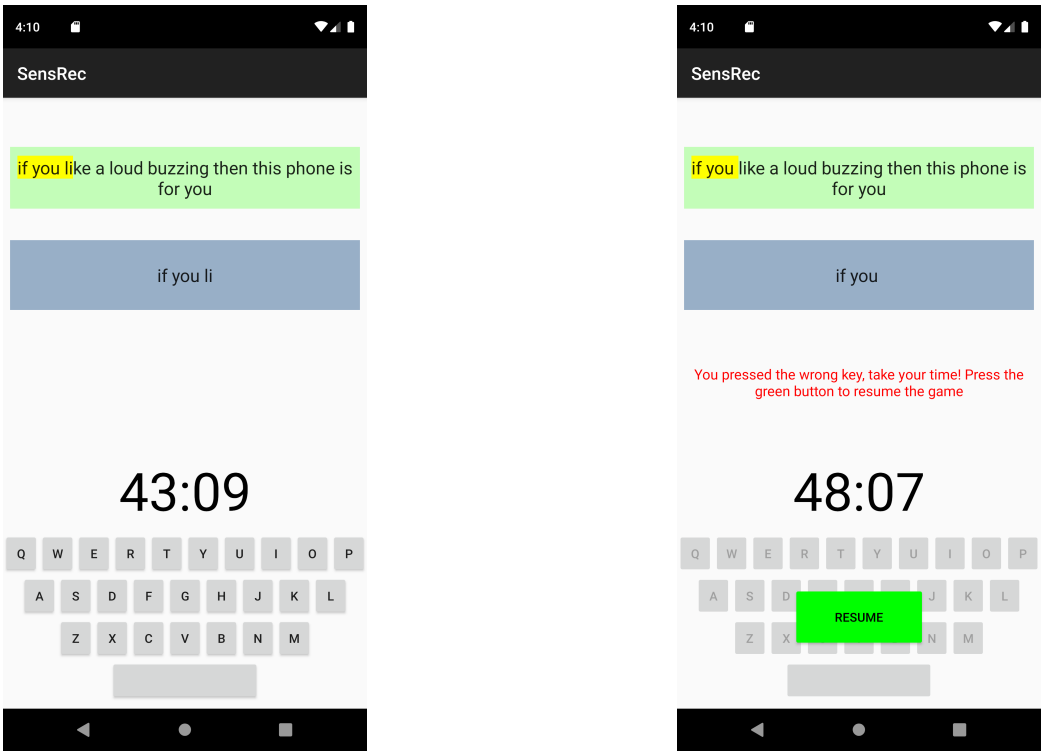
setSensor(): qui vengono inizializzati il *SensorManager* e le variabili per i sensori.

```
1 private void setSensor() {
2     mSensorManager = (SensorManager)getSystemService(
    Context.SENSOR_SERVICE);
3     mAcc = mSensorManager.getDefaultSensor(Sensor.
    TYPE_ACCELEROMETER);
4     mGyr = mSensorManager.getDefaultSensor(Sensor.
    TYPE_GYROSCOPE);
5     mMag = mSensorManager.getDefaultSensor(Sensor.
    TYPE_MAGNETIC_FIELD);
6     mGrav = mSensorManager.getDefaultSensor(Sensor.
    TYPE_GRAVITY);
7 }
```

Listing 2.5: Metodo setSensor()

2.3.2 Digitazione del testo

La digitazione del testo avviene tramite una tastiera creata appositamente per questa applicazione, all'interno del file *keyboard.xml*. Ha lo stesso stile di una normale tastiera *QWERTY* per smartphone ed è stata sviluppata in modo che i vari *Button* abbiano come attributo *tag* la lettera che rappresentano.
I vari eventi *touch* saranno catturati dal metodo ***onTouch(View v, MotionEvent event)***, che tramite i due *MotionEvent ACTION_DOWN* e *ACTION_UP* permette di riconoscere, rispettivamente, quando viene premuto lo schermo e quando viene rilasciato.



(a) Seconda activity (b) Schermata di errore

Figura 2.2: Seconda *activity* e schermata di errore

Ogni volta che viene rilevato un evento di tocco si calcola la pressione con cui avviene questo evento, tramite il metodo *event.getPressure()*, e viene chiamato il metodo *SentenceMode* che è un supporto per uno dei metodi più importi, cioè *checkAllCharacter(boolean isLastWord)*. *SentenceMode* semplicemente separa la parola appena scritta dal resto della frase ogni volta che viene premuto lo "spazio" tra le parole e imposta la variabile *TouchedView* uguale alla lettera premuta, mentre *checkAllCharacter(boolean isLastWord)* controlla che gli *StringBuilder*, all'interno di un *ArrayList⟨StringBuilder⟩*, contengano tutte le lettere della parola appena scritta, una per ogni *StringBuilder*. Questa cosa è molto importante in quanto dobbiamo avere per ogni file tutte le lettere della frase, cosa non scontata per le chiamate della funzione *onSensorChanged()* che verrà spiegata successivamente.
Quando, invece, viene invocato il *MotionEvent ACTION_UP* la variabile

TouchedView viene impostata a *-1*, la pressione uguale a *0* e viene chiamato il metodo *checkString()*. Quest'ultimo controlla la correttezza delle lettere che vengono premute e se qualche lettera non viene registrata negli *StringBuilder*, esempio in Figura 2.2b. Quando questo accade viene fermato il *countdown*, riportandolo al tempo che c'era quando è stato premuto l'ultimo spazio, e la registrazione dei sensori, per poi essere riattivati quando si preme il tasto *Resume* che appare sulla tastiera (Figura 2.2b).

2.3.3 Salvataggio dei dati su file CSV

Come accennato nella sezione *2.3.1*, tramite il metodo *createNewFileCSV()* viene creato un nuovo file *CSV* e scritto il relativo *header*. Questo sarà composto da 21 valori: il primo è *Timestamp*, poi ogni sensore (accelerometro, giroscopio, magnetometro, orientamento, gravità) avrà tre valori che corrispondono ai tre assi (*x,y,z*) di ognuno, infine ci sarà la *View* che corrisponde alla lettera che viene premuta (o -1 quando non viene premuto nulla), la *Pressure* che corrisponde all'event.getPressure(), l'*Username*, la mano utilizzata e il tempo corrente.
Quindi ogni riga del file sarà composta nel modo seguente:

$$\Theta(t) =< t, A_{\mathrm{x,y,z}}, G_{\mathrm{x,y,z}}, M_{\mathrm{x,y,z}}, O_{\mathrm{x,y,z}}, Gr_{\mathrm{x,y,z}}, \omega, Pre >$$

dove t corrisponde all'istante in cui i sensori sono stati modificati; *A, G, M, O, e Gr* corrispondono ai valori dei sensori; ω corrisponde alla lettera (o -1) e *Pre* corrisponde alla pressione.
Per ottenere i dati dai sensori viene implementata una funzione dell'interfaccia *SensorEventListener*, cioè ***onSensorChanged(SensorEvent event)*** che viene chiamata ogni qual volta rileva una cambiamento nei valori dei sensori. Per scrivere i valori per ogni riga del file è stato creato un *array* che conterrà i valori di tutti i sensori in un determinato istante di tempo, chiamato *SensorVal*. Per ogni lettera è molto probabile che ci siano righe successive con la stessa *View*, dovuto al fatto che cambiano più sensori. Ad

esempio per quanto riguarda il Giroscopio e il sensore di Orientamento la funzione risulta essere come segue:

```
public void onSensorChanged(SensorEvent event) {
    \\Giroscopio
    if (event.sensor.getType() == Sensor.TYPE_GYROSCOPE) {
        System.arraycopy(event.values, 0, SensorVal, 3, event
    .values.length);
    }
    \\Orientamento
    if (mLastAccSet && mLastMagSet){
        SensorManager.getRotationMatrix(mR, null, mLastAcc,
    mLastMag);
        SensorManager.getOrientation(mR, mOrientation);
        System.arraycopy(mOrientation, 0, SensorVal, 9, event
    .values.length);
    }

    StringBuilder csvRow = CSV.generateCSVRow(SensorVal,
    TouchedView, pressure, time);
    rowsToLoad.add(csvRow);
}
```

Listing 2.6: Metodo onSensorChanged(SensorEvent event) per Giroscopio e sensore di Orientamento

Come è possibile vedere dal codice in alto, il sensore di Orientamento viene calcolato in modo diverso rispetto al Giroscopio. Infatti, per trovare il suo valore bisogna partire dai valori dell'Accelerometro e del Magnetometro per poter utilizzare la funzione *getRotationMatrix()* che a sua volta servirà per la funzione *getOrientation()*.
Infine, per creare la vera e propria riga del file viene utilizzato il metodo *generateCSVRow()* che semplicemente mette i valori delle variabili in ordine secondo l'*header* all'interno di uno *StringBuilder*. Lo *StringBuilder* risultante viene aggiunto in un *ArrayList* che verrà utilizzato per fare i vari controlli di cui si è discusso nella sezione precedente.
Per stampare ogni riga nel file csv è stato utilizzato il metodo *print()* della

classe *PrintWriter* come mostrato dal seguente codice:

```
for (StringBuilder stringBuilder: rowsToLoad){
    printWriter.print(stringBuilder);
}
```

Listing 2.7: Scrittura sul file csv

Quando il test è terminato si ritorna alla *MainActivity* dove tramite un *OptionMenu*, posto in alto a destra della schermata, è possibile aprire un *Dialog* che contiene i contatori di ogni lettera, utile per capire l'avanzamento di ognuno:

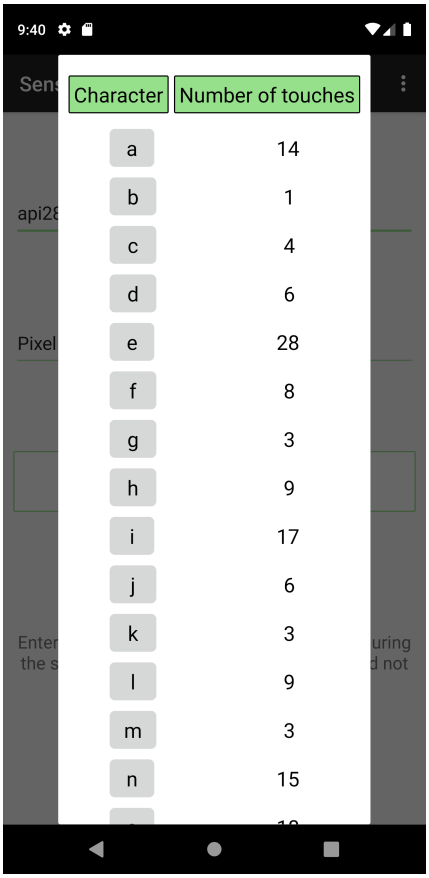


Figura 2.3: Contatori delle lettere

2.3.4 Salvataggio dei file CSV su Firebase

Una volta terminato il test con successo, il file *csv* creato all'interno del dispositivo verrà spedito allo *storage* di *Firebase*. *Firebase* è una piattaforma creata da *Google* che offre un ampio catalogo di servizi, utili per la creazione di applicazioni per dispositivi o anche per applicazioni web. Tra i vari servizi offerti da *Firebase* troviamo lo *storage* dove è possibile memorizzare 5GB di file gratuitamente. Quindi, per inviare i file allo *storage* è stata scritta una funzione *sendFilesToStorage()* dove prima di tutto viene disattivata la registrazione dei sensori, chiuso il *printWriter* e aggiornati i contatori delle lettere appena inserite, successivamente viene inviato il file allo *storage* e cancellato definitivamente dal dispositivo.

```
1  private void sendFilesToStorage() {
2      for(final File elem : Files){
3          Uri file = Uri.fromFile(elem);
4          StorageReference riversRef = storageRef.child("rawCsv
       /" + file.getLastPathSegment());
5          riversRef.putFile(file).addOnSuccessListener(new
       OnSuccessListener<UploadTask.TaskSnapshot>() {
6              @Override
7              public void onSuccess(UploadTask.TaskSnapshot
       taskSnapshot) {
8                  elem.delete();
9              }
10         })
11         .addOnFailureListener(new OnFailureListener() {
12             @Override
13                     public void onFailure(@NonNull Exception
       e) {
14                         Toast.makeText(RecSensor.this, "
       Failed " + e.getMessage(), Toast.LENGTH_LONG).show();
15                     }
16                 });
17     }
18 }
```

Listing 2.8: Invio file csv allo *storage*

Capitolo 3

Creazione del Dataset

3.1 Descrizione Generale

Una volta terminata l'applicazione, è stato chiesto a diverse persone di effettuare i test. Ogni persona ha dovuto fare almeno 100 eventi *touch* per ogni lettera, per poter avere un numero sostenuto di lettere e di conseguenza un *dataset* più grande. Dallo *storage* di *Firebase*, tramite la *Google Cloud SDK Shell*, sono stati raccolti i test da tre persone che hanno utilizzato la mano destra e da una che ha usato la mano sinistra. Per ogni persona che ha usato la mano destra, sono stati raccolti circa 200 file a testa, quindi un totale di circa 600 file, mentre per la persona mancina sono stati raccolti circa 400 file.
Per arrivare al *dataset* finale, il lavoro da effettuare è stato suddiviso in tre fasi principali:

- *Prima fase:* dove vengono aggiunti ai vari file i valori di *Magnitude*, *Roll* e *Pitch*;
- *Seconda fase:* dove vengono calcolate le *Features* del *dataset*;
- *Terza fase:* dove si effettua l'unione di tutti i file.

Queste tre fasi verranno approfondite nelle sezioni successive fino a mostrare il *dataset* risultante.

Da questa fase fino alla fine del progetto di tesi è stato usato il linguaggio di programmazione *python*, per la sua maggiore flessibilità nella gestione e modifica dei file e per il modulo *sklearn* per la parte di *machine learning*.

3.2 Calcolo di Magnitude, Roll e Pitch

La prima fase per arrivare al *dataset* finale, è l'aggiunta di nuove colonne all'interno dei file restituiti dai test dell'applicazione. Infatti, sono stati aggiunti ai file iniziali *Magnitude*, *Roll* e *Pitch* per ogni sensore preso in considerazione e questo ha comportato anche un iniziale modifica dell'*header*. In particolare la *Magnitude* rappresenta la forza registrata dal singolo sensore. Può essere calcolata come la radice quadrata della somma dei quadrati degli assi. In questo elaborato, è stata calcolata anche la *magnitude* delle coppie di assi. Ad esempio le *magnitude* calcolate per l'accelerometro sono:

$$Mag(A_{\mathrm{x,y,z}}) = \sqrt{x^2 + y^2 + z^2}$$
$$Mag(A_{\mathrm{x,y}}) = \sqrt{x^2 + y^2}$$
$$Mag(A_{\mathrm{y,z}}) = \sqrt{y^2 + z^2}$$
$$Mag(A_{\mathrm{x,z}}) = \sqrt{x^2 + z^2}$$

Queste verranno calcolate per ogni sensore citato precedentemente.
In ambiente *python* un modo semplice per calcolare la *magnitude* è quello di usare funzioni del *package Numpy* come mostra il codice seguente:

```
1 def calcMagnitude(sensorAxes):
2     xyz = [float(sensorAxes[0]), float(sensorAxes[1]), float(
    sensorAxes[2])]
3     xy = [float(sensorAxes[0]), float(sensorAxes[1])]
4     xz = [float(sensorAxes[0]), float(sensorAxes[2])]
5     yx = [float(sensorAxes[1]), float(sensorAxes[2])]
6     mag_XYZ = np.linalg.norm(xyz)
7     mag_XY = np.linalg.norm(xy)
8     mag_XZ = np.linalg.norm(xz)
9     mag_YZ = np.linalg.norm(yx)
```

```
10      return [mag_XYZ, mag_XY, mag_XZ, mag_YZ]
```

Listing 3.1: Funzione per calcolare le *magnitude*

Oltre alla *magnitude*, sono stati aggiunti anche *Roll* e *Pitch* per ogni sensore. Questi valori calcolano l'angolo di rotazione del dispositivo attorno ai suoi tre assi, come mostra la seguente figura:

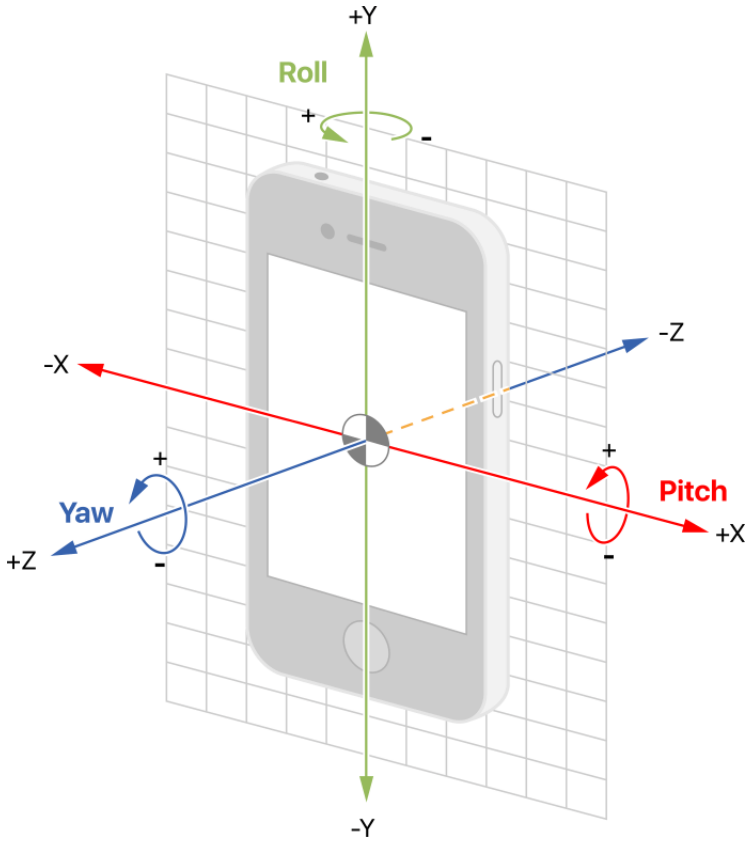


Figura 3.1: *Roll* e *Pitch* di un dispositivo mobile

Le formule per calcolare questi due valori, ad esempio per l'accelerometro, sono:

$$Roll(A_{x,y,z}) = \frac{A_y}{A_z}$$

$$Pitch(A_{x,y,z}) = \frac{-A_x}{\sqrt{A_y^2 + A_z^2}}$$

In *python* si possono calcolare tramite la funzione *atan2()* in questo modo:

```
# Roll
def calcRoll(sensorAxes):
    return math.atan2(float(sensorAxes[1]), float(sensorAxes
    [2]))
# Pitch
def calcPitch(sensorAxes):
    return math.atan2(-float(sensorAxes[0]), math.sqrt(pow(
    float(sensorAxes[1]), 2) + pow(float(sensorAxes[2]), 2)
```

Listing 3.2: Funzioni per calcolare *Roll* e *Pitch*

I file *csv* risultanti saranno formati da nove valori per ogni sensore. Per l'Accelerometro i valori sono:

$$< A_{\mathrm{x}}, A_{\mathrm{y}}, A_{\mathrm{z}}, AMag_{\mathrm{x,y}}, AMag_{\mathrm{x,z}}, AMag_{\mathrm{y,z}}, AMag_{\mathrm{x,y,z}}, ARoll, APitch >$$

Una volta calcolati questi valori per ogni sensore sarà possibile determinare le *Features* da utilizzate per la successiva analisi dei modelli di *machine learning*. A seguire vengono spiegate in modo specifico.

3.3 Calcolo delle Features

Il secondo passo per costruire il *dataset* finale è il calcolo delle *Features*. Queste nell'ambito del *machine learning* sono, per definizione, delle proprietà individuali e misurabili di un fenomeno osservato, solitamente espresse in forma numerica.
Grazie alla creazione delle *features* abbiamo un ricalcolo dei dati grezzi. Questo ricalcolo viene effettuato in modo da studiare i dati accorpando tutti quelli relativi ad un singolo tocco su una singola lettera durante la digitazione, quindi quelli relativi ad un singolo *touch event*. Per ogni riga $\Theta(t)$ consecutiva che ha lo stesso ω vengono accorpati ognuno dei 9 valori di ciascun sensore tramite funzioni di aggregazione, i valori risultanti saranno le *features* che

vengono usati per creare il *dataset*.
Le funzioni di aggregazione prese in considerazione sono:

- *Mean:* corrisponde alla media tra i valori dell'intervallo considerato.
- *Median:* corrisponde alla mediana tra i valori dell'intervallo considerato
- *Stdev:* corrispondente alla deviazione standard tra i valori dell'intervallo considerato
- *Var:* corrisponde alla varianza tra i valori dell'intervallo considerato
- *Max:* corrisponde al massimo tra i valori dell'intervallo considerato
- *Min:* corrisponde al minimo tra i valori dell'intervallo considerato

In *python* è molto semplice calcolare questi valori tramite il modulo *statistics*:

```
def calcFeatures(sensorValues):
    mean = statistics.mean(sensorValues)
    median = statistics.median(sensorValues)
    stdev = statistics.stdev(sensorValues)
    variance = statistics.variance(sensorValues)
    max_val = max(sensorValues)
    min_val = min(sensorValues)
```

Listing 3.3: Funzione per calcolare le *features*

Dunque per ogni sensore ci sarà un totale di 54 *features*. Avendo preso in considerazione 5 sensori differenti il numero totale di *features* che saranno utilizzate è di 270.
Successivamente da ogni file sono state eliminate tutte le righe che hanno -1 come ω, quindi il numero di righe all'interno di un file corrisponderà alla lunghezza della frase di quel file.

3.4 Creazione del Dataset finale

Una volta calcolate le *features*, per costruire il *Dataset* è stato creato un ultimo file *csv* dove all'interno sono state inserite tutte le righe dei file creati

nella sezione precedente, in ordine e senza *header*. Così facendo il *dataset* sarà formato da tutte le frasi scritte dai vari *tester* con i valori delle relative *features* per ogni lettera. Quindi le colonne del *dataset* sono:

$$< features(A), features(G), features(M), features(O), features(Gr),$$
$$\omega, Username, Hand, Smartphone >$$

In *python* il *dataset* è stato importato tramite il metodo *read_csv()* del *package Pandas* che permette di inserire il *dataset* all'interno di un *Dataframe*, una struttura molto comoda che consente di svolgere varie operazioni sulle righe, sulle colonne o anche sulle singole celle.
Ciascun passaggio spiegato in questo capitolo è stato applicato sia ai file creati utilizzando la mano destra che a quelli creati usando la mano sinistra. Quindi, sono stati creati due *dataset* differenti, uno per la mano destra e l'altro per la mano sinistra. Per quanto riguarda il primo si è ottenuto un *dataset* di circa 25600 righe mentre per il secondo circa 16700 righe.

Capitolo 4

Modelli di Supervised e Unsupervised Learning

4.1 Data scaling e analisi dei sensori

Prima di passare ai modelli di *machine learning* presi in considerazione è stata necessaria una normalizzazione dei dati, cioè i dati sono stati ridimensionati su un intervallo che va da -1 a 1. Questo perché la normalizzazione rende i dati più adatti ad una convergenza e ad una comparazione, e soprattutto ad una successiva rappresentazione grafica. Per effettuare la normalizzazione in *python* è stato utilizzato il *package sklearn*, tramite la funzione *preprocessing.MinMaxScaler()*:

```
scaler = preprocessing.MinMaxScaler(feature_range=(-1,1))
# x = valore di ogni features
x_scaled = scaler.fit_transform(x)
```

Listing 4.1: Codice per la normalizzazione dei dati

Una volta normalizzati i dati, si è passati alla creazione delle *heatmap* per ognuna delle *features* dei sensori. La *heatmap*, o mappa termica, è una rappresentazione grafica dei dati dove i singoli valori contenuti in una matrice sono rappresentati da colori che possono variare per intensità o tonalità. È molto comoda per dare spunti visivi su come un fenomeno varia nello

spazio. In *python* è possibile creare una *heatmap* tramite l'omonima funzione all'interno del *package Seaborn*. Nel nostro caso è stata creata una *heatmap* a forma di tastiera per simulare al meglio il comportamento di ciascun sensore. Per creare questi grafici è stato necessario raggruppare le righe con lo stesso ω, facendo la media di tutti i valori. Inoltre, sono stati aggiunti i seguenti due valori a ciascuna riga:

- X: è il valore sull'asse delle ascisse per ogni lettera all'interno della tastiera;
- Y: è il valore sull'asse delle ordinate per ogni lettera all'interno della tastiera.

Quindi le *heatmap* sono state create nel modo seguente:

```
# labz = lista contenente 4 liste, ognuna delle quali
    contiene una riga di lettere della tastiera.
# 'dist' = il valore di ogni features
_ = df_heat.pivot('y', 'x', 'dist')
sns.heatmap(_, annot=labz, fmt='', square=True, cbar_kws =
    dict(use_gridspec=False,location="bottom"))
```

Listing 4.2: Codice per la creazione di *heatmap* a forma di tastiera

Nella sezione successiva verranno spiegati solamente alcuni dei grafici relativi ai valori dei sensori.
Come detto nel capitolo precedente, sono tre le persone che hanno partecipato ai test utilizzando la mano destra, con tre *smartphone* differenti, mentre solo una persona ha svolto i test con la mano sinistra facendo, però, almeno 200 eventi *touch* per ogni lettera. Sarà interessante vedere come si comportano i sensori usando dati aggregati da più *smartphone* rispetto a quelli di un singolo dispositivo.

4.1.1 Heatmap dei sensori

Accelerazione media asse x

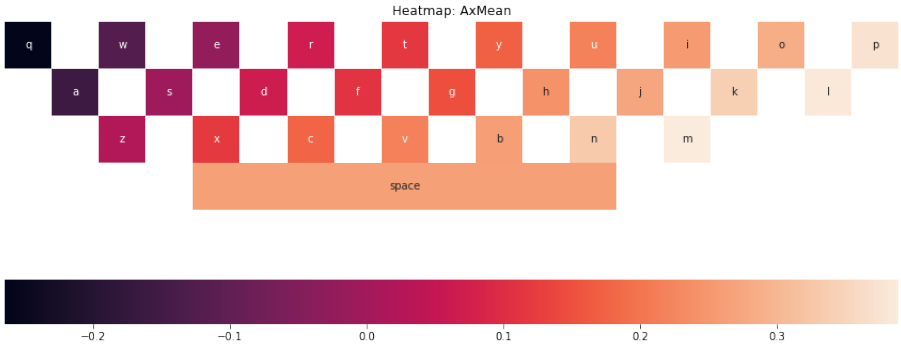


Figura 4.1: *Heatmap* dell'accelerazione media sull'asse x con mano destra

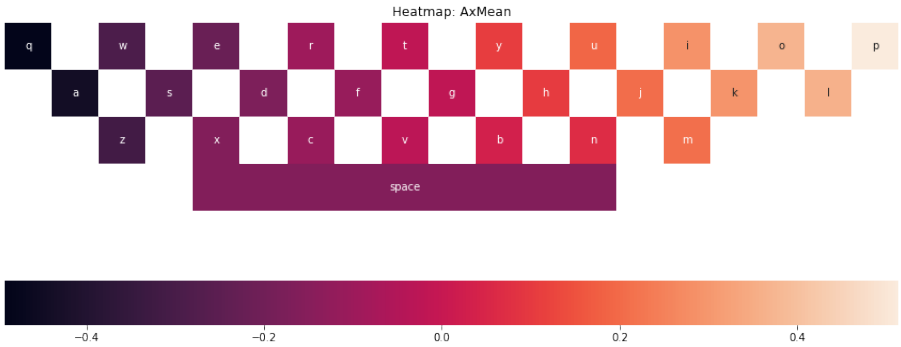


Figura 4.2: *Heatmap* dell'accelerazione media sull'asse x con mano sinistra

Come possiamo vedere da entrambi i grafici, l'accelerazione media sull'asse x è un ottima *feature* per lo studio relativo a questo elaborato. Infatti possiamo notare come, sia per la mano destra che per quella sinistra, la variazione di tonalità è similare con i valori che gradualmente aumentano se ci si sposta dal lato sinistro a quello destro. Una differenza tra i due grafici è che la Figura 4.1 identifica come valore massimo la lettera 'M', mentre la Figura 4.2 identifica la 'P'. Per entrambe il valore minimo viene identificato dalla lettera 'Q'. In generale però entrambe hanno un andamento costante.

Roll medio del sensore di Gravità

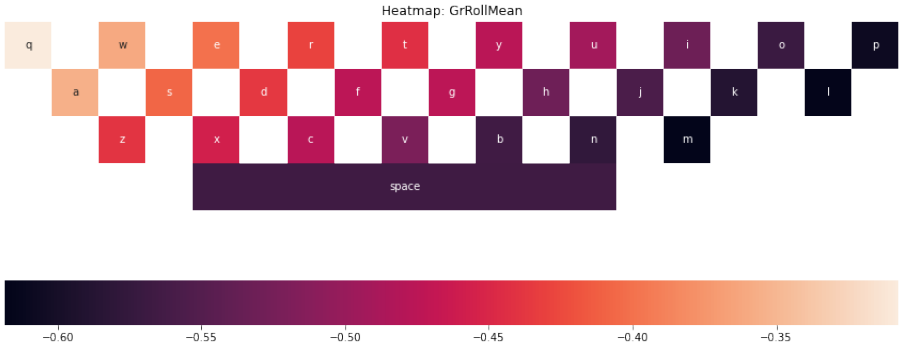


Figura 4.3: *Roll* medio del sensore di Gravità con mano destra

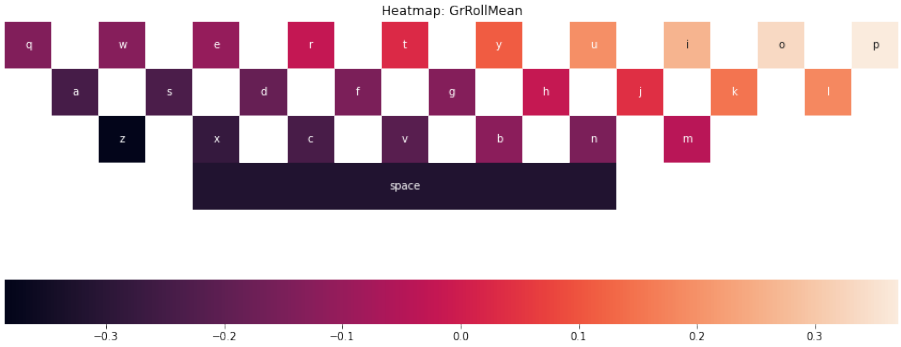


Figura 4.4: *Roll* medio del sensore di Gravità con mano sinistra

Un risultato interessante da vedere è la *feature Roll* applicata al sensore di Gravità. Infatti, come è possibile osservare dalle Figure 4.3 e 4.4, otteniamo che il valore minimo e massimo si trovano sostanzialmente all'opposto nelle due tastiere. In Figura 4.3 c'è un incremento costante del colore a partire da in basso a destra della tastiera fino ad arrivare in alto a sinistra, e in Figura 4.4 a partire da in basso a sinistra fino ad arrivare in alto a destra. Questo ci fa capire che l'angolo di rotazione, sull'asse y della Figura 3.1, è praticamente opposto in base alla mano che viene usata.

Differenze con Giroscopio e Magnetometro

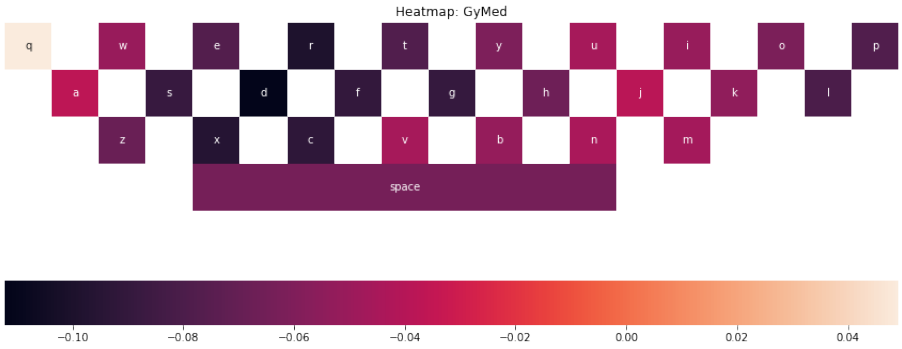


Figura 4.5: Mediana su asse y del Giroscopio con mano destra

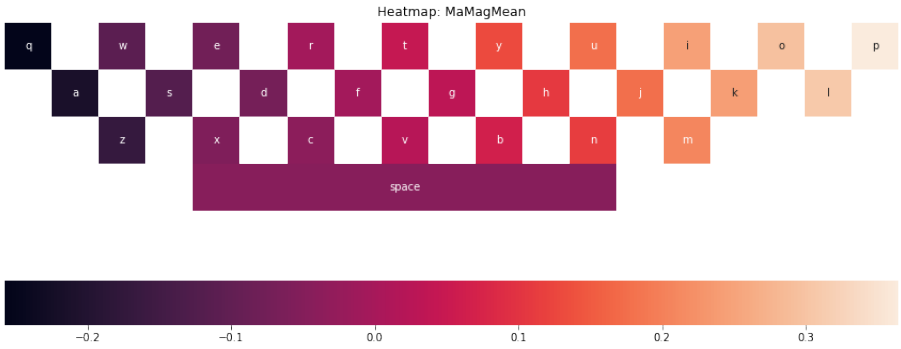


Figura 4.6: *Magnitude* del Magnetometro con mano sinistra

Non si hanno avuti risultati simili per quanto riguarda sensori come Giroscopio e Magnetometro, dove il primo non è ben riconosciuto dalla mano sinistra mentre il secondo non è ben riconosciuto dalla mano destra.
Ad esempio la mediana sull'asse y del Giroscopio con mano destra, come mostrato in Figura 4.5, potrebbe essere interessante perché viene isolata la lettera 'Q', che si trova in una posizione particolare rispetto a tutte le altre lettere, infatti ha un colore molto distante rispetto a quello di tutti gli altri.
Abbiamo dei buoni risultati, ad esempio, con la *Magnitude* del Magnetome-

tro con mano sinistra, dove la forza viene distribuita in modo costante su tutte le lettere.

4.2 Modelli di Supervised Learning

Per capire se è effettivamente possibile riconoscere parole e frasi scritte usando i valori restituiti dai sensori dello *smartphone*, sono stati studiati e confrontati diversi modelli di *machine learning*, utilizzando inizialmente algoritmi di *supervised learning*. Infatti, sono stati confrontati sette modelli, tra quelli più famosi, e di cui si è discusso con maggiore attenzione nel *capitolo 1*. È possibile utilizzarli importandoli dal *package sklearn*.
I modelli presi in considerazione sono:

- *RandomForestClassifier(n_estimators = 100, random_state = 42)*
- *GaussianNB()*
- *SVC(C=1.0, kernel='linear', gamma='auto', random_state = 42)*
- *DecisionTreeClassifier(random_state=42)*
- *AdaBoostClassifier(n_estimators=100, learning_rate = 0.1)*
- *LogisticRegression(solver='liblinear', random_state = 42)*
- *KNeighborsClassifier(weights='distance', n_neighbors=30, n_jobs = -1)*

Per poter applicare questi algoritmi è necessario dividere il *dataset* in due parti:

Training set: composto da $\frac{2}{3}$ del *dataset* originale. A sua volta suddiviso in *X_train* e *y_train*, dove il primo contiene tutte le colonne delle *features* mentre il secondo contiene la colonna delle *View* (ω). Viene utilizzato per "allenare" il modello che si vuole utilizzare per predire le lettere.

Test set: composto dalle rimanenti righe del *dataset*. Anche'esso si suddivide in *X_test* e *y_test* allo stesso modo della parte di *training*. Il *test set* viene utilizzato per calcolare la parola predetta (chiamata *y_pred*) e quanto ci si avvicina rispetto a quella originale, calcolando l'*accuracy* tra *y_pred* e *y_test*.

In *python* per fare la suddivisione è stata utilizzata la funzione *StratifiedShuffleSplit()* come mostrato in Listing 4.3. Questa funzione restituisce una suddivisione randomizzata in base ai parametri *n_split*, per dire quante volte suddividere, e *test_size*, per dire in che dimensione si vuole suddividere il *dataset*. La suddivisione viene realizzata conservando la percentuale di campioni per ogni *labels*.

```
features = df.drop(['View','User','Hand', 'Smartphone', 'x',
    'y','dist'], axis=1)
# Convert to numpy array
labels = np.array(df['View'])
features = np.array(features)
sss = StratifiedShuffleSplit(n_splits=1, test_size=0.3,
    random_state=0)
for train_index, test_index in sss.split(features, labels):
    X_train, X_test = features[train_index], features[
    test_index]
    y_train, y_test = labels[train_index], labels[test_index]
```

Listing 4.3: Codice per la suddivisione del *dataset* in *training set* e *test set*

Successivamente, è stata usata la classe *GridSearchCV* per capire quali parametri applicare ad ogni modello citato precedentemente. Infatti, tramite le funzioni *grid_result.best_score_* e *grid_result.best_params_* è possibile calcolare con quale dei parametri, che vengono passati alla *GridSearchCV*, si ottiene uno *score* più elevato. Ogni modello è stato aggiunto all'interno di una lista, chiamata *models*, per poterli confrontare comodamente.

Per allenare il modello si usa la funzione *fit(X_train, y_train)*, mentre per calcolare la lista dei valori predetti viene chiamata la funzione *predict(X_test)*.

Per calcolare l'*accuracy* del modello, invece, viene usata la funzione *accuracy_score(y_test, y_pred)*. Quindi il confronto viene effettuato come mostra l'immagine seguente:

```
for model in models:
    # allena il modello
    model.fit(X_train, y_train)
    # predice i valori
    y_pred = model.predict(X_test)
    # calcola l'accuracy
    accuracy = metrics.accuracy_score(y_test, y_pred)
```

Listing 4.4: Confronto dei modelli di *machine learning*

I risultati delle *accuracy* verranno discussi nel capitolo 6.

4.2.1 Confusion Matrix

La *Confusion Matrix* è uno dei metodi per mostrare graficamente come sono stati predetti i valori dall'algoritmo di *machine learning*, cioè se vi è "confusione" nella classificazione delle diverse classi. Infatti, restituisce una rappresentazione dell'accuratezza della classificazione. Per definizione, una *Confusion Matrix* C è una matrice quadrata dove $C_{i,j}$ rappresentano il numero di istanze dei dati che si conosce essere nella classe i, detta *true label*, e predetta nella classe j, detta *predicted label*. Quindi ogni colonna della matrice rappresenta i valori predetti, mentre ogni riga rappresenta i valori reali. L'elemento sulla riga i e sulla colonna j è il numero di casi in cui il classificatore ha classificato la classe "vera" i come classe j.
Di seguito verranno mostrate e spiegate le *confusion matrix* di alcuni dei modelli visti in precedenza sia per la mano destra che per quella sinistra. Queste ci permetteranno di capire con quali lettere vengono "confuse" le vere *View*.

Confusion Matrix RandomForest-mano destra

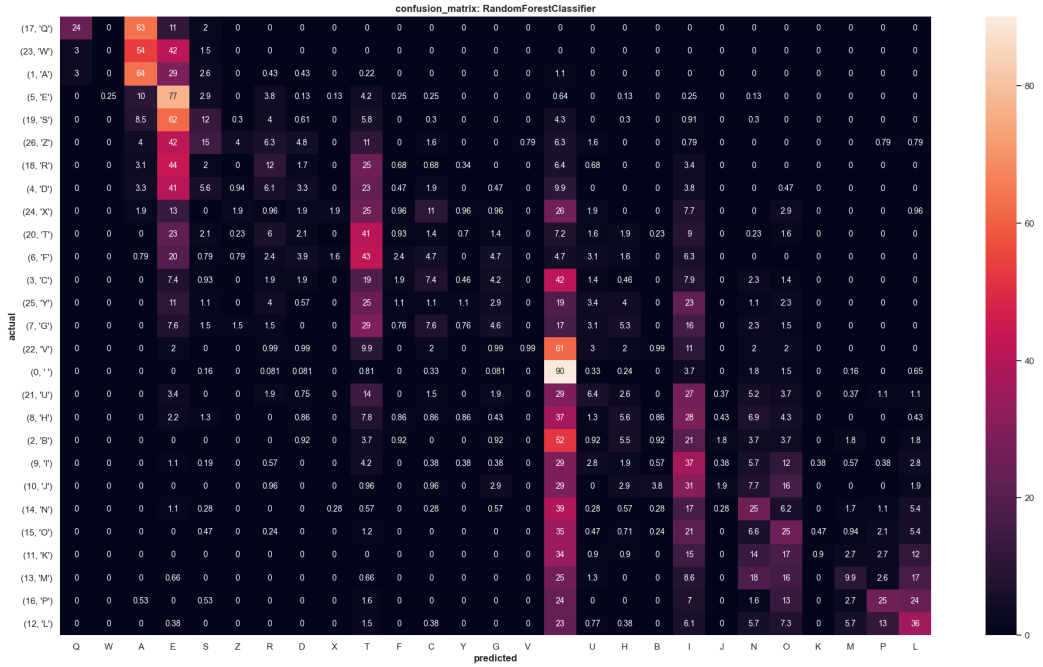


Figura 4.7: *Confusion Matrix* con valori scalati dell'algoritmo *RandomForest* - mano destra

Come possiamo vedere dalla *heatmap* della *confusion matrix* in Figura 4.7, similmente anche in Figura 4.9, ci sono alcune lettere che vengono predette maggiormente rispetto ad altre, ad esempio le lettere 'A', 'E', 'T' e soprattutto 'SPAZIO'. È pur vero che i valori massimi di queste lettere sono nelle celle corrette o comunque, in alcuni casi, nelle celle in corrispondenza di lettere molto vicine, come ad esempio nella colonna della 'T' dove il valore massimo (43) si ha in corrispondenza della lettera reale 'F'.
Inoltre, ci sono lettere che vengono predette poche volte come ad esempio la 'W' o la 'V'. Questo problema può derivare dal fatto che alcune lettere abbiano più *training examples* di altre. Infatti, analizzando i contatori delle lettere degli eventi *touch* di una singola persona si è visto come ci sia una differenza consistente tra il numero di tocchi di alcune lettere vicine. Ad

esempio la lettera 'T', da un singolo utente, è stata premuta un totale di 451 volte mentre lettere vicine come la 'F' o la 'Y' rispettivamente 150 e 200 volte. Lo stesso ragionamento è applicabile ad altre lettere come la 'E' e lo 'SPAZIO' dove si raggiungono quasi mille tocchi per il primo e più di mille tocchi per il secondo, per singolo utente. Quindi si è pensato di creare una *heatmap* delle distanze tra le varie lettere con i valori presi dalla *confusion matrix*. Se D è la nuova *heatmap*, con le lettere sull'asse y e le distanze sull'assex, C è la *confusion matrix* e ω è la lettera, allora sarà che:

$$D_{\omega,0} = C_{\omega,\omega}$$

$$D_{\omega,\mathrm{i}} = \sum_{\gamma_\mathrm{i}} C_{\omega,\gamma_\mathrm{i}} \ \forall i = 1...max\_distance(\omega)$$

dove γ_i saranno tutte le lettere a distanza i da ω.

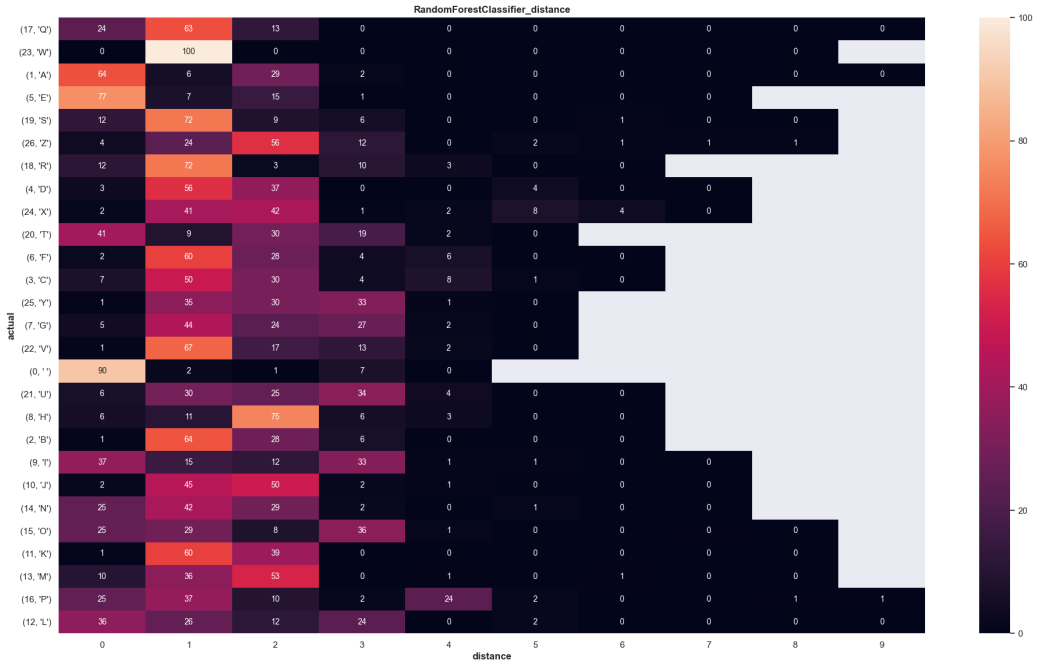


Figura 4.8: Distanze tra le lettere per RandomForest - mano destra

Come mostra la Figura 4.8, tralasciando le lettere più toccate come 'SPAZIO', 'T' o 'E', la maggior parte delle lettere hanno un valore alto a distanza 1, ciò vuol dire che se si sbaglia nel predire una lettera si sbaglia di poco.

Confusion Matrix RandomForest-mano sinistra

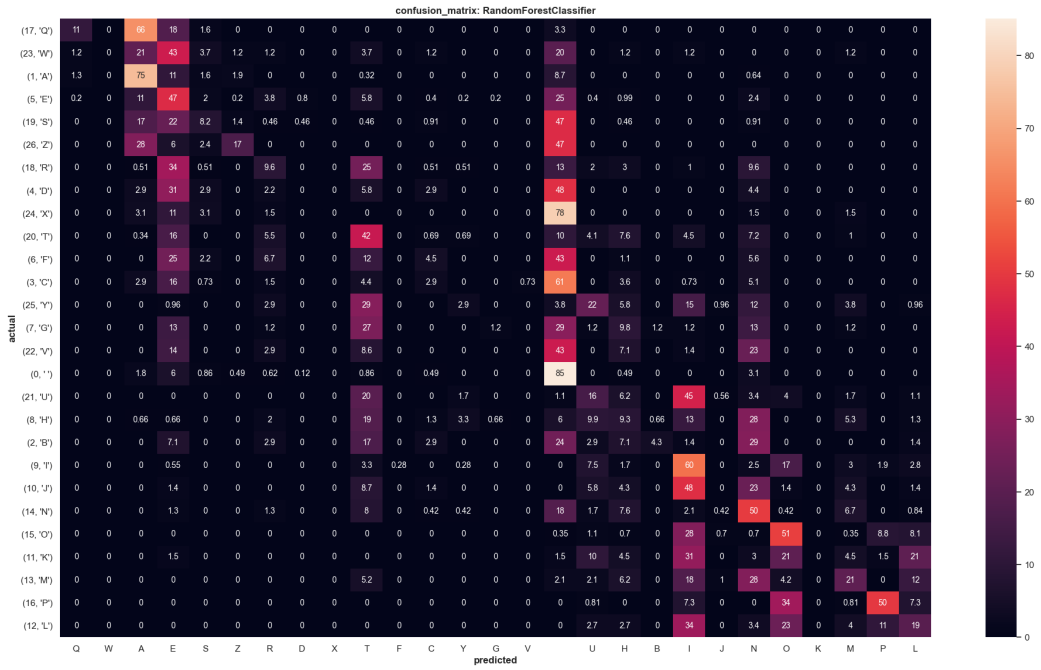


Figura 4.9: *Confusion Matrix* con valori scalati dell'algoritmo *RandomForest* - mano sinistra

Simile alla Figura 4.7 è ciò che si ottiene dai dati dove è stata usata la mano sinistra. Infatti possiamo vedere anche in Figura 4.9 come i valori più comuni vengono predetti maggiormente rispetto ad altre lettere. Al contrario della Figura 4.7, però, è possibile osservare come i valori predetti, ad esempio, come lo 'SPAZIO' o la 'I' questa volta siano nella parte alta della matrice, che corrisponde alle lettere nella parte sinistra della tastiera.
Anche qui è stato creato un grafico come quello di Figura 4.8 che ha dato risultati uguali a quelli descritti in precedenza.

4.2.2 Precision, Recall e F-score

Prendere l'*accuracy* come unica metrica di giudizio di un modello non è molto utile, soprattutto quando il *il dataset* non è bilanciato, cioè quando

c'è un numero diverso di istanze per ogni classe. Questo è dovuto dal fatto che l'*accuracy* viene calcolata come:

$$Accuracy = \frac{TN + TP}{TN + FP + TP + FN}$$

dove, se consideriamo la seguente matrice di confusione:

		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

Figura 4.10: *Confusion Matrix* per spiegazione

i valori saranno:

- *TN*: rappresenta i *True Negative*, cioè quando il valore attuale è negativo e viene predetto come negativo;
- *TP*: rappresenta i *True Positive*, cioè quando il valore attuale è positivo e viene predetto come positivo;
- *FN*: rappresenta i *False Negative*, cioè quando il valore attuale è positivo e viene predetto come negativo;
- *FP*: rappresenta i *False Positive*, cioè quando il valore attuale è negativo e viene predetto come positivo.

L'*accuracy* è una buona metrica quando abbiamo un *dataset* simmetrico (il numero di FN e FP è vicino) e quando i FN e i FP hanno un costo simile. In alternativa ci sono altre tre metriche: Precision, Recall e F-Score.

Precision

La *Precision* è definita come:

$$Precision = \frac{TP}{TP + FP}$$

La *Precision* dice quanti esempi sono effettivamente positivi. È una buona metrica quando il costo di FP è alto. Questa metrica risponde alla domanda: quanti di quelli che abbiamo etichettato come lettera X sono veramente una lettera X?

Recall

Il *Recall*, invece, è definito come:

$$Recall \ = \ \frac{TP}{TP + FN}$$

Quindi il *Recall* calcola quanti degli *Actual Positive* il modello allenato acquisisce etichettandoli come positivi (True Positive). È una buona metrica quando il costo dei FN è alto, ci permette di trovare il modello migliore. Idealmente un buon classificatore deve avere un valore alto sia per la *Precision* che per il *Recall*. Si sceglie il *Recall* quando le occorrenze di FN sono inaccettabili. Questa metrica risponde alla domanda: di tutte le lettere X, quante di quelle che prevediamo sono corrette?

F1 Score

L'ultima metrica è l'*F1 Score*. È una metrica che tiene conto sia della *Precision* che del *Recall* ed è definita come:

$$F1 \ Score \ = \ 2 \ * \ \frac{Precision * Recall}{Precision + Recall}$$

Quindi l'*F1 Score* è la media armonica di *Precision* e *Recall* ed è una metrica migliore dell'*accuracy*, infatti è il migliore quando si ha un *dataset* irregolare. L'*F1 Score* è alto solo quando anche *Precision* e *Recall* sono alti.

Dopo aver calcolato le *confusion matrix*, sono state calcolate tutte e tre le metriche discusse precedentemente. In *python* è possibile ottenerle tramite la funzione *classification_report()*, del *package sklearn.metrics*, passando come parametri *y_test* e i valori predetti. Di seguito verranno mostrati i grafici contenente le tre nuove metriche dell'algoritmo RandomForest, sia per la mano destra che per quella sinistra.

Precision, Recall e F1 Score per RandomForest

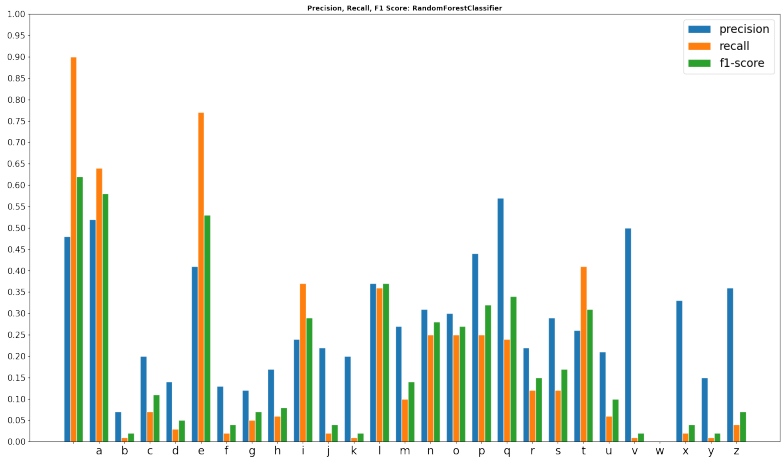


Figura 4.11: *Precision*, *Recall* e *F1 Score* - mano destra

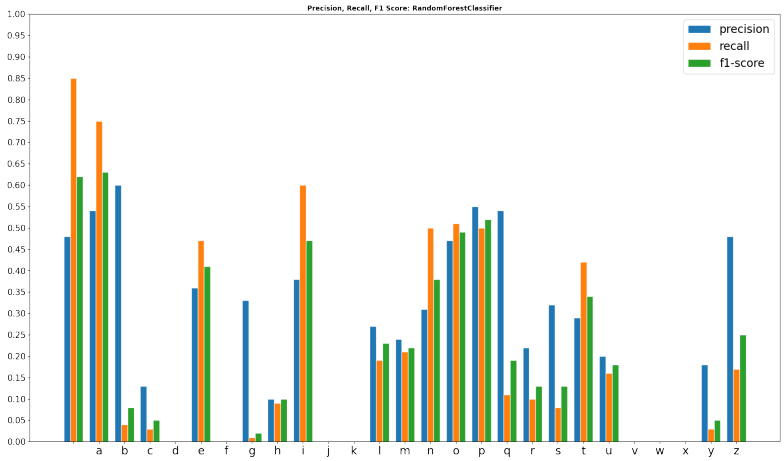


Figura 4.12: *Precision*, *Recall* e *F1 Score* - mano sinistra

Possiamo vedere come in entrambi i casi le lettere con più *training examples* hanno una percentuale di *recall* maggiore rispetto alla *precision*, viceversa per le lettere con meno *training examples* hanno una percentuale di *precision* superiore al *recall*.
Alcune lettere, soprattutto per la mano sinistra, non sono predette. Questo

è dovuto alla dimensione del *dataset* che non basta a ricoprirle tutte, infatti già con un *dataset* più grande, come in Figura 4.11, c'è solo una lettera che non viene rappresentata nel grafico.

4.3 Clustering

È stato interessante studiare come i modelli si comportino utilizzando come classi i cluster, restituiti da un algoritmo di *clustering*, al posto delle *View*. Infatti, tramite il *clustering* è possibile raggruppare le lettere per similitudine, vedendo la somiglianza delle *features*. L'algoritmo di *clustering* che è stato applicato è il ***K-Means***, approfondito nel Capitolo 1.
L'algoritmo è stato usato con K da 1 a 27 su un *Dataframe* diverso, che ha un solo campione per ogni lettera e con i valori delle *features* uguali alla media di ogni *features* delle righe appartenenti alla stessa lettera nel *Dataframe* iniziale. A differenza del *supervised learning* con l'*unsupervised* non c'è bisogno di suddividere il *dataset* in *training* e *test set*.

```
for k in range(1, 28):
    model_kmeans = KMeans(n_clusters=k)
    model_kmeans.fit(df_kmeans)
    test_labels = model_kmeans.predict(df_kmeans)
```

Listing 4.5: Applicazione dell'algoritmo *K-Means*

Quando $K = 27$, come in Figura 4.13, ogni lettera fa parte di un cluster a sé quindi è come non aver il *clustering*. Invece, quando $K = 1$ le lettere fanno parte dello stesso cluster e quindi hanno la stessa *label*.

4.3.1 Calcolo dell'accuracy dei modelli

Ogni lettera del nuovo *Dataframe*, avrà un cluster assegnato partendo dalla *clusterizzazione* ad 1 fino ad arrivare alla *clusterizzazione* a 27. Successivamente, tramite la funzione *merge* del *package Pandas* sono state create 27 nuove colonne nel *dataset* iniziale che contengono i valori di ogni *clusterizzazione* in base alla lettera di ciascuna riga.

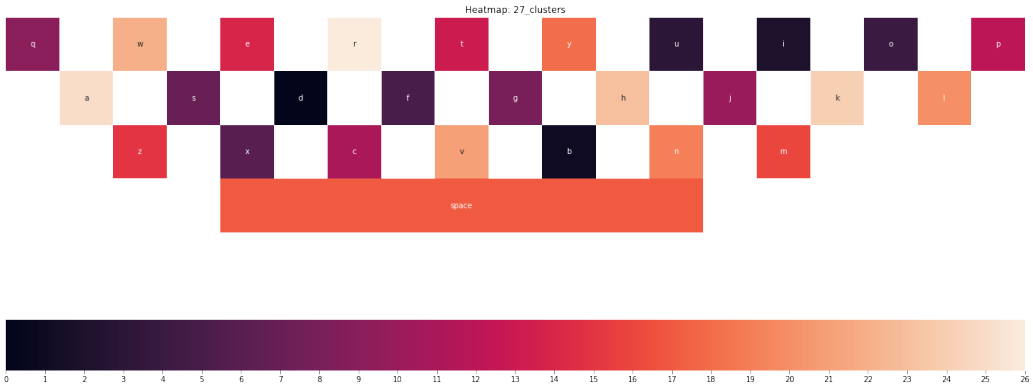


Figura 4.13: Esempio dei risultati con $K = 27$

Le nuove colonne verranno utilizzate come colonna di *labels*, una alla volta, tranne quella che identifica la *clusterizzazione* a 1 perché vorrebbe dire che tutte le lettere appartengono alla stessa classe e alcuni algoritmi necessitano di almeno due classi per calcolare l'*accuracy*. Quindi, come in *Listing* 4.3, è stato suddiviso il *dataset* in *train* e *test set*, questa volta però utilizzando come *labels* ciascuna colonna delle *clusterizzazioni*. Successivamente sono stati confrontati i modelli esattamente come in *Listing* 4.4.
Usare la *clusterizzazione* a 27 come *labels* equivale ad usare le *View*, quindi l'*accuracy* dei modelli non varia molto se si usa uno o l'altro. Al contrario quando viene usata la *clusterizzazione* a 1 l'*accuracy* dei modelli è 100% perché tutte le lettere appartengono allo stesso cluster.
Nella sezione successiva è possibile vedere degli esempi di clusterizzazioni e quali hanno causato un importante decremento dell'*accuracy* nei sette modelli presi in considerazione.

4.3.2 Analisi di alcune clusterizzazioni

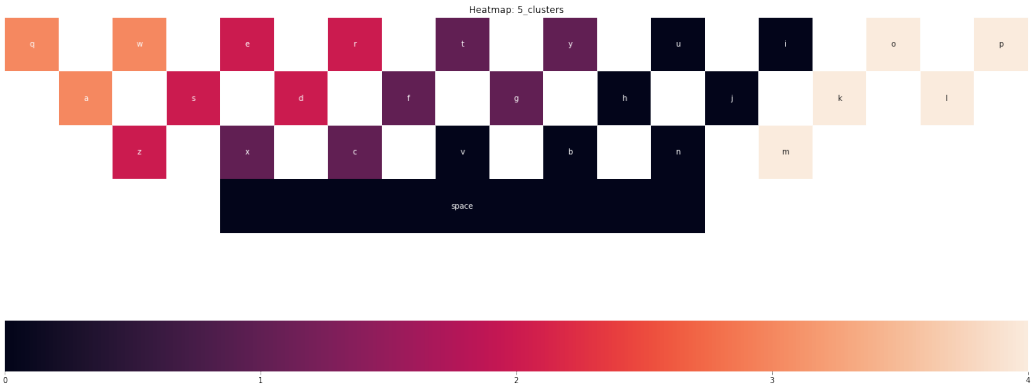


Figura 4.14: Clusterizzazione a 5 - mano destra

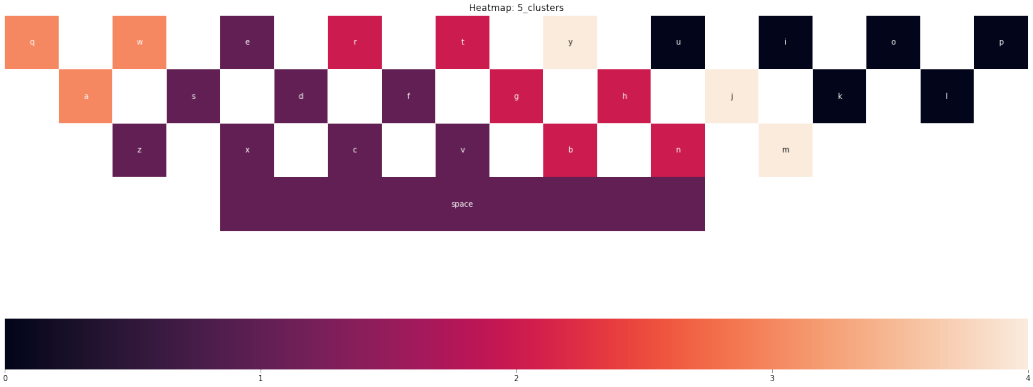


Figura 4.15: Clusterizzazione a 5 - mano sinistra

Un elemento interessante comune ad entrambe le Figure è l'appartenenza allo stesso cluster delle lettere 'Q', 'W' e 'A', che risultano essere le lettere più lontane per la mano destra mentre le più vicine per la mano sinistra.
Ci sono anche differenze evidenti, come ad esempio nella Figura 4.14 le lettere di ogni cluster nella prima riga si trovano sempre più a destra rispetto alle lettere nella terza riga, viceversa in Figura 4.15 avviene esattamente l'opposto.

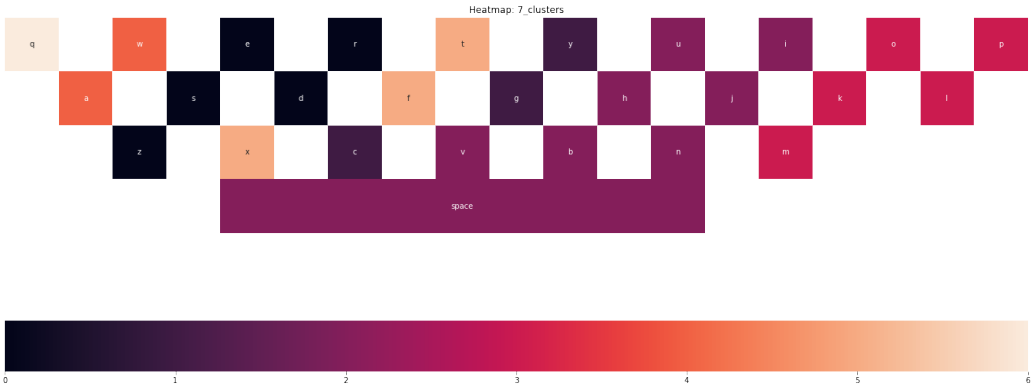


Figura 4.16: Clusterizzazione a 7 - mano destra

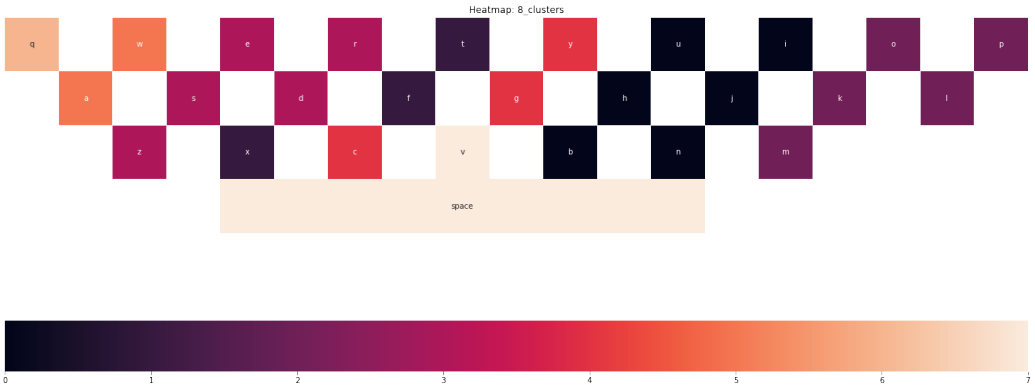


Figura 4.17: Clusterizzazione a 8 - mano destra

Le Figure 4.16 e 4.17, invece, rappresentano le *clusterizzazioni* che hanno causato un picco di prestazioni per i modelli della mano destra. Ad esempio, per l'algoritmo RandomForest passare dalla *clusterizzazione* a 7 alla *clusterizzazione* ad 8 causa una riduzione di *accuracy* di circa 7%, oppure nell' algoritmo AdaBoostClassifier la differenza di *accuracy* tra il primo ed il secondo è del 17%. La stessa cosa accade per la mano sinistra però passando dalla *clusterizzazione* ad 8 alla *clusterizzazione* a 9.
I risultati verranno approfonditi maggiormente nel capitolo 6.

Capitolo 5

Grafo delle parole

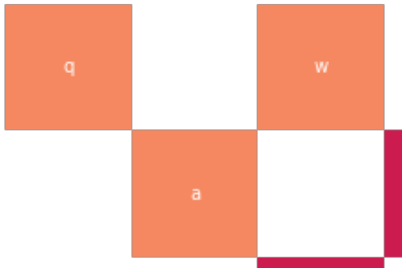
5.1 Riconoscimento del testo

Al fine di rendere possibile il riconoscimento del testo, è necessario correlare tutte le sequenze di caratteri digitati dall'utente e costruire un albero dei possibili caratteri digitati. In questo modo, si può ottenere la possibilità di classificare la parola in base alla probabilità associata ad ogni lettera. Per fare ciò sono stati utilizzati i cluster trovati nel capitolo precedente, così da poter costruire non un unico albero ma un *grafo di possibili parole*.
Il grafo viene definito partendo da una **configurazione** che specifica le *clusterizzazioni* da cui partire per poter calcolare le probabilità di ogni lettera. Prima di arrivare alla creazione del grafo finale è necessario spiegare la creazione dei grafi intermedi associati alle *clusterizzazioni* nella configurazione.
In ambiente *python* la creazione dei grafi è stata effettuata tramite la funzione *DiGraph()* del *package NetworkX*.

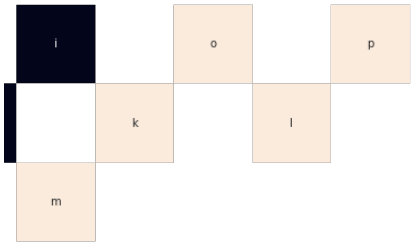
5.2 Grafi Intermedi

Per ogni *clusterizzazione* nella configurazione viene creato un grafo intermedio. La costruzione viene effettuata nel seguente modo:

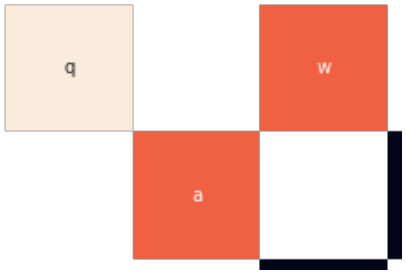
Sia C_i l'i-esima *clusterizzazione* nella configurazione, per ogni carattere digitato L_j alla j-esima posizione viene creata una lista $Cl(L_j)$ contenente L_j e le lettere all'interno del suo cluster nella C_i *clusterizzazione*. Ad ogni livello j viene inserito nel grafo il nodo L_j e tutti i nodi all'interno di $Cl(L_j)$.
Supponiamo che la parola predetta sia *'app'* e la configurazione è composta dalle *clusterizzazioni* [5, 7, 27] allora *Cl(a)* e *Cl(p)*, delle prime due *clusterizzazioni*, contengono le lettere dei seguenti cluster:



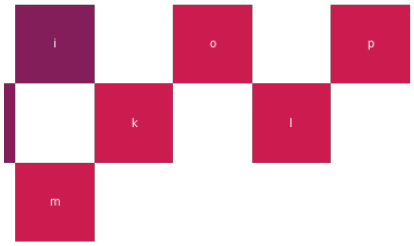
(a) *Cl(a)* in *clusterizzazione* a 5



(b) *Cl(p)* in *clusterizzazione* a 5



(c) *Cl(a)* in *clusterizzazione* a 7



(d) *Cl(p)* in *clusterizzazione* a 7

Figura 5.1: *Cl(a)* e *Cl(p)* in *clusterizzazione* a 5 e a 7

La *clusterizzazione* a 27, invece, identifica un grafo con solo i nodi della parola *'app'*. Quindi le liste di nodi saranno formate in questo modo:

- In *clusterizzazione* a 5: $Cl(a) = \{q, a, w\}$
- In *clusterizzazione* a 5: $Cl(p) = \{p, o, m, l, k\}$
- In *clusterizzazione* a 7: $Cl(a) = \{a, w\}$
- In *clusterizzazione* a 7: $Cl(a) = \{p, o, m, l, k\}$

- In *clusterizzazione* a 27: $Cl(a) = \{a\}$
- In *clusterizzazione* a 27: $Cl(p) = \{p\}$

allora i grafi intermedi risultano essere come quelli mostrati in Figura 5.2 e in Figura 5.3.

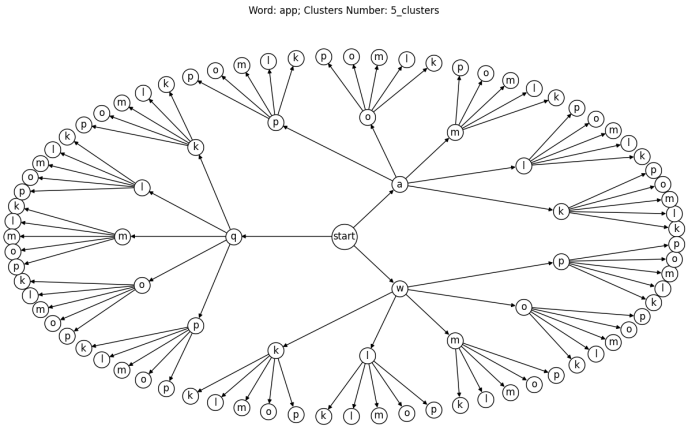


Figura 5.2: Grafo intermedio della clusterizzazione a 5 per la parola 'app'

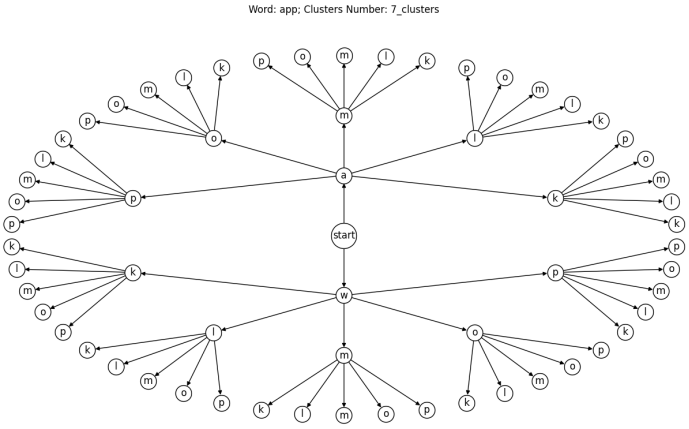


Figura 5.3: Grafo intermedio della clusterizzazione a 7 per la parola 'app'

Dopo aver costruito tutti i grafi intermedi della configurazione, è possibile

costruire il grafo finale ed assegnare le probabilità ad ogni lettera. Nella sezione successiva verrà spiegato come costruire il grafo finale e come vengono assegnate le probabilità alle lettere.

5.3 Grafo Finale

La struttura del grafo finale sarà la stessa di quella del primo grafo intermedio della configurazione, con stessi nodi e stessi archi. Quindi se consideriamo l'esempio della sezione precedente, il grafo finale avrà la stessa struttura di quello in Figura 5.2.
Il metodo per calcolare la probabilità di ogni nodo è il seguente:
Per ogni successore S di ogni nodo L_j al j-esimo livello, escluso quello delle foglie, si conta quante volte S compare in ogni grafo intermedio C_i, supponiamo Sum_S volte. Allora la probabilità che il nodo S esca è:

$$p_{\mathrm{S}} = \frac{Sum_{\mathrm{S}}}{Sum_{\mathrm{S}} + \sum_{k} Sum_{\mathrm{k}}} \qquad con\ k \in successori(L_{\mathrm{j}}) \setminus S$$

Quindi il grafo risultante dall'esempio precedente è:

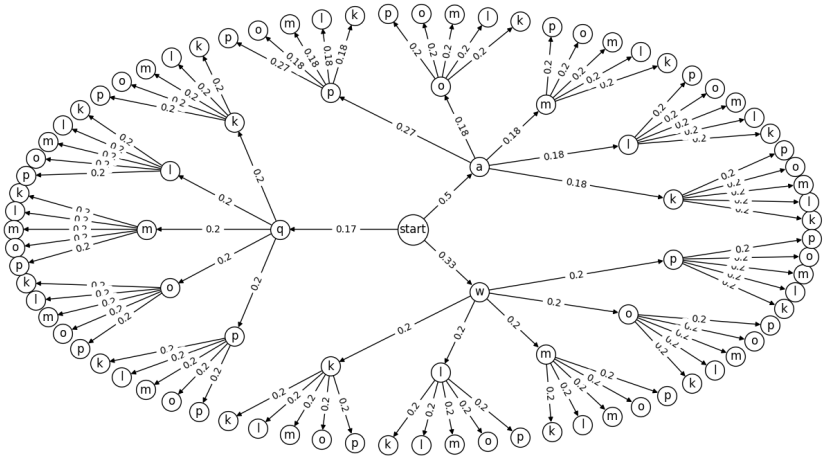


Figura 5.4: Grafo finale per la parola 'app'

Ad esempio per calcolare la probabilità dei primi 3 nodi 'A', 'Q' e 'W' bi-

sogna vedere quante volte appaiono nei grafi intermedi della configurazione [5, 7, 27], in quella specifica profondità. La lettera 'A' compare in tutte le *clusterizzazioni*, quindi $Sum_{\mathrm{A}} = 3$; la lettera 'W' appare due volte e la 'Q' una sola volta. Quindi le probabilità delle prime tre lettere sono:

$$p_{\mathrm{A}} = \frac{Sum_{\mathrm{A}}}{Sum_{\mathrm{A}} + Sum_{\mathrm{W}} + Sum_{\mathrm{Q}}} = \frac{3}{3 + 2 + 1} = \frac{3}{6} = 0.5$$

$$p_{\mathrm{W}} = \frac{Sum_{\mathrm{W}}}{Sum_{\mathrm{W}} + Sum_{\mathrm{A}} + Sum_{\mathrm{Q}}} = \frac{2}{2 + 3 + 1} = \frac{2}{6} = 0.33$$

$$p_{\mathrm{Q}} = \frac{Sum_{\mathrm{Q}}}{Sum_{\mathrm{Q}} + Sum_{\mathrm{W}} + Sum_{\mathrm{A}}} = \frac{3}{1 + 2 + 3} = \frac{1}{6} = 0.17$$

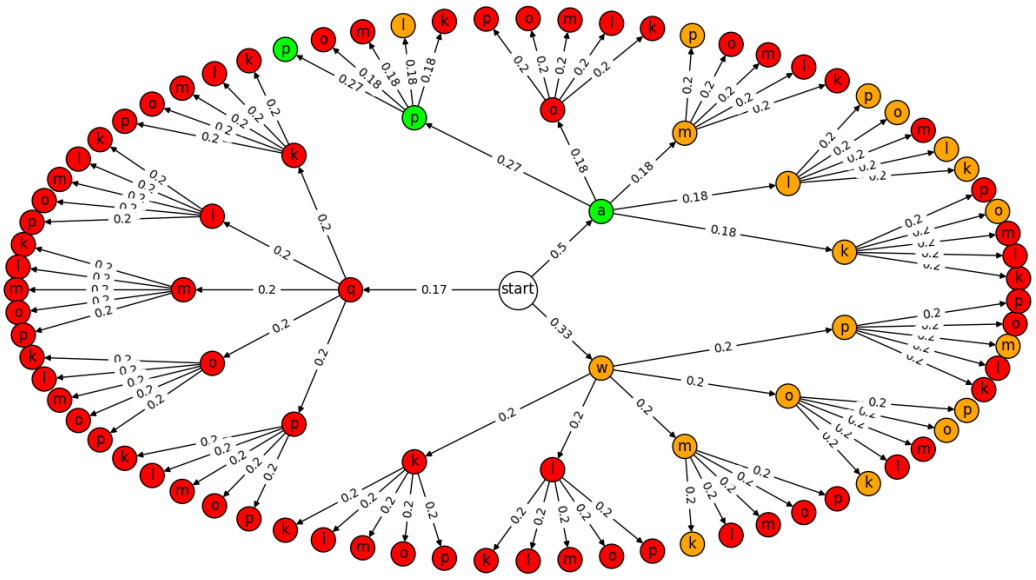


Figura 5.5: Grafo finale in base alle parole esistenti

Una volta costruito il grafo finale con le probabilità si è passati al filtraggio delle parole, formate dalle sequenze di lettere all'interno dei cammini *radice-foglia* del grafo. Questo è possibile grazie all'utilizzo di un dizionario inglese composto da oltre 470.000 parole e consultabile all'indirizzo `https://github.com/dwyl/english-words`. Nel grafo in Figura 5.5, ogni

cammino *radice-foglia* che ha la foglia di colore rosso identifica una parola che non si trova all'interno del dizionario. Se i cammini *radice-foglia* sono di colore arancione vuol dire che sono possibili parole riconosciute, mentre il cammino di colore verde rappresenta la parola che è stata digitata e da cui sono stati costruiti tutti i grafi. Quindi nell'esempio in Figura 5.5 ci sono 13 parole possibili

5.4 Classifica delle parole

Infine, viene costruita una **classifica** delle parole esistenti all'interno del grafo finale. Questa viene calcolata facendo il prodotto della probabilità di ogni lettera del cammino, ad esempio, considerando il grafo in Figura 5.5, lo *score* della parola *'app'* è:

$$score(\ app\ )\ =\ 0.5\ *\ 0.27\ *\ 0.27\ =\ 0.0365$$

Per questo esempio è stato considerato il caso ottimo, cioè quello in cui la parola predetta dal classificatore sia proprio quella che abbiamo digitato, quindi *'app'*. In realtà ciò che si vuole analizzare utilizzando i grafi è quanto in alto nella classifica si trova la parola reale rispetto alla parola predetta, che avrà lo *score* più alto, nella configurazione che viene considerata. Quindi la parola da cui costruire i grafi intermedi sarà quella data in *output* dal classificatore.

Nel prossimo capitolo verrà mostrato l'output del classificatore considerando *app* come parola digitata. Inoltre, verrà calcolata la posizione in classifica della parola reale rispetto alla parola predetta, utilizzando, per la mano destra, [4, 7, 14, 27] come configurazione dell'algoritmo RandomForest e [2, 7, 14, 27] per l'algoritmo SVC, mentre per la mano sinistra verrà usata la configurazione [13, 20, 27] per entrambi gli algoritmi.
Questo lavoro però è ancora al suo stadio preliminare, infatti l'algoritmo dovrebbe essere testato su tutte le parole predette da quelle all'interno del *test*

set, definito precedentemente, e con una configurazione tale da poter contenere sempre tutte le lettere della parola reale, così da poter fare un confronto con quella predetta. Quindi, ad esempio, utilizzando la *clusterizzazione* ad 1 come primo elemento avremmo ciò che desideriamo. Non sono stati forniti risultati per questo studio a causa di alcune complicazioni, come la costruzione dei grafi a partire dalla *clusterizzazione* ad 1, in quanto per costruire il grafo di una parola con una lunghezza pari a tre deve creare più di 17500 nodi, oppure per il tempo di run dell'algoritmo, che per costruire la classifica di una parola lunga quattro deve visitare più di 21000 archi. Quindi questa è stata più un'analisi per determinare quali sono le insidie di questo algoritmo e come poterle risolvere, ad esempio per la costruzione del grafo si potrebbe creare un algoritmo per determinare la configurazione migliore senza doverla inserire a mano, oppure eliminare quei cammini che non servono man mano che viene costruito.

Capitolo 6

Risultati

6.1 Introduzione ai risultati

In questo capitolo verranno elencati i risultati ottenuti dalle varie analisi spiegate nei capitoli precedenti. In particolare:

- nella sezione 6.2 verrà mostrata l'*accuracy* della classificazione per lettera con i sette algoritmi di *machine learning* tramite un grafico a barre (*seaborn.barplot*), che è un modo comodo per riassumere una serie di dati relativi alla categoria;
- successivamente, nella sezione 6.3, sarà possibile osservare l'*accuracy* della classificazione per n cluster, tramite un grafico a linee che è ideale per mostrare la riduzione dell'*accuracy* in base alla clusterizzazione usata;
- infine, nella sezione 6.4, sarà chiarito il procedimento utilizzato per generare la classifica delle parole, con la configurazione che cambierà in base all'algoritmo e alla mano utilizzata e supponendo che la parola digitata sia *app*.

6.2 Classificazione per lettera

In questa sezione verranno mostrate le *accuracy* dei vari algoritmi di *machine learning*, nel caso in cui sono state utilizzate le lettere come *labels* del problema. Ricordo che i sette algoritmi usati, con i relativi colori nei grafici, sono: RandomForest (in blu), GaussianNB (in arancione), SVC (in verde), DecisionTree (in rosso), AdaBoost (in viola), LogisticRegression (in marrone) e KNeighbors (in rosa). Saranno descritti prima i risultati per la mano destra e successivamente quelli per la mano sinistra.

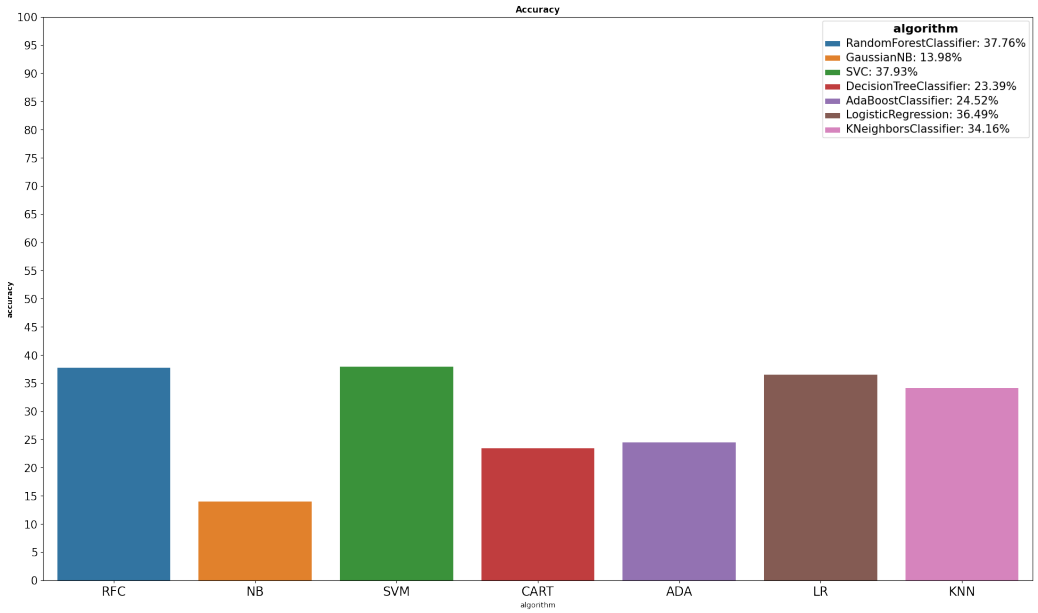


Figura 6.1: *Accuracy* della classificazione per lettera - mano destra

Per la mano destra, come è possibile vedere nella Figura 6.1, l'*accuracy* più elevata viene riportata dall'algoritmo SVC con una percentuale del 37.93%, seguito da RandomForest con 37.76%; mentre per la mano sinistra, in Figura 6.2, la percentuale più elevata deriva sempre da SVC ma questa volta con una percentuale del 41.36%, seguito da LogisticRegression e da RandomForest con rispettivamente 40.37% e 39.43%.

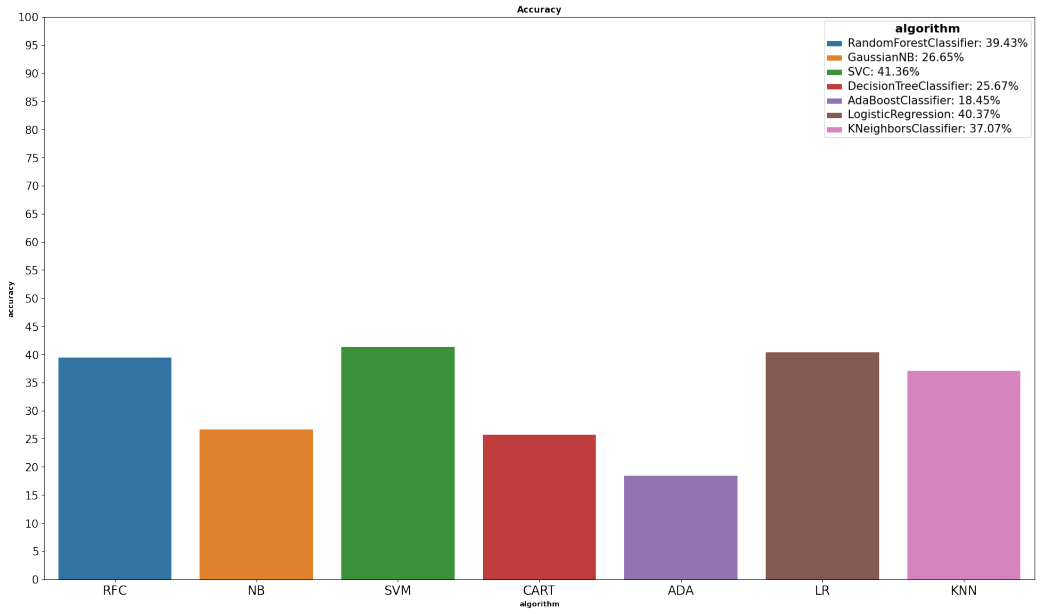


Figura 6.2: *Accuracy* della classificazione per lettera - mano sinistra

È interessante notare come nel caso della mano destra l'utilizzo di più dispositivi causa una diminuzione dell'*accuracy* tra il 2% e il 4% per gli algoritmi citati precedentemente, e di circa 13% per l'algoritmo GaussianNB. Invece, si può notare come l'algoritmo AdaBoostClassifier si comporti meglio con dati derivanti da più dispositivi, con un *accuracy* superiore di circa 6%. Quindi in generale gli algoritmi migliori risultano essere:

- **RandomForest:** con un *accuracy* del 37.76% per la mano destra e 39.43% per la mano sinistra;
- **SVC:** con un *accuracy* del 37.93% per la mano destra e 41.36% per la mano sinistra;
- **LogisticRegression:** con un *accuracy* del 36.49% per la mano destra e 40.37% per la mano sinistra;
- **KNeighbors:** con un *accuracy* del 34.16% per la mano destra e 37.07% per la mano sinistra.

6.3 Classificazione per cluster

Mentre nella sezione precedente sono stati applicati gli algoritmi utilizzando le lettere come *labels*, qui, invece, verranno discussi i risultati ottenuti dall'utilizzo dei cluster, prodotti dall'algoritmo K-Means, come *labels*, partendo dalla clusterizzazione a 2 fino alla clusterizzazione a 27.

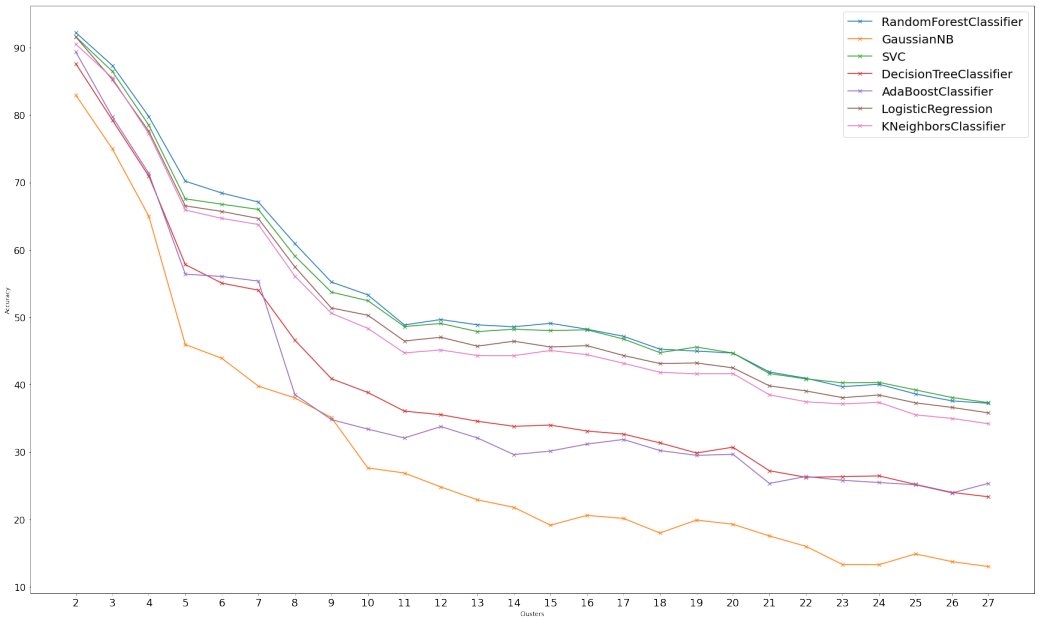


Figura 6.3: *Accuracy* della classificazione per cluster - mano destra

N_CLUSTER	*RFC*	*NB*	*SVC*	*CART*	*ADA*	*LR*	*KNN*
6	68.43	43.91	66.77	55.06	56.05	65.7	64.67
7	67.1	39.79	66.0	54.02	55.37	64.66	63.76
8	60.94	38.03	59.09	46.61	38.54	57.49	56.1

Tabella 6.1: *Accuracy* con *clusterizzazione* da 6 a 8

Come mostra la Figura 6.3, per ogni modello c'è una netto decremento dell'*accuracy* dalla *clusterizzazione* a 2 fino alla 5. Ad esempio per Random-

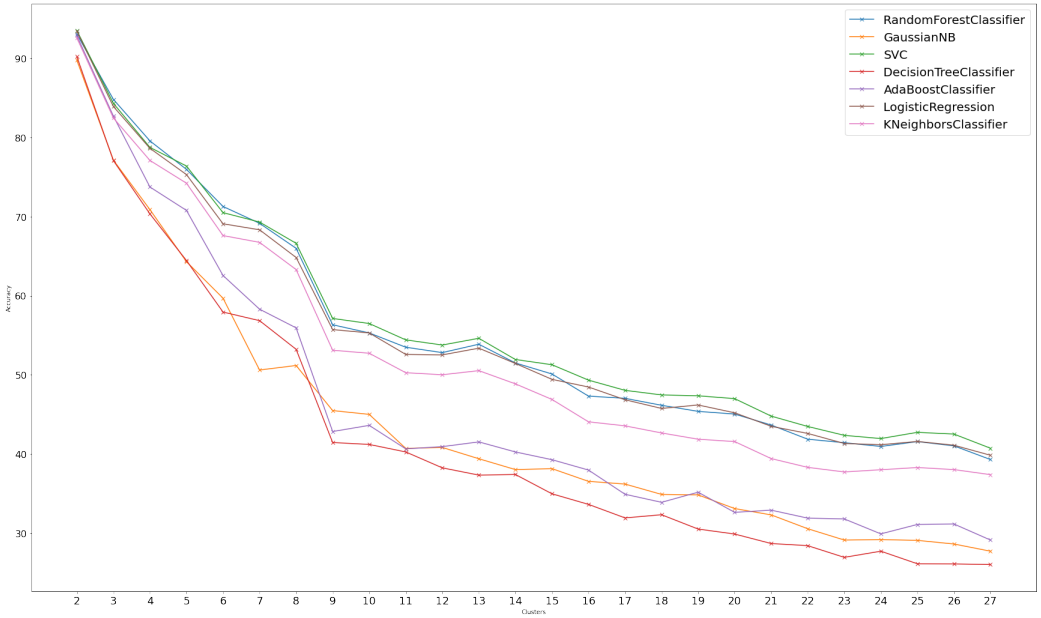


Figura 6.4: *Accuracy* della classificazione per cluster - mano sinistra

Forest si passa da circa 92% a circa 70%, quindi abbiamo un decremento del 22%; oppure per AdaBoost che dall'89% arriva a 56%. Dopo questo picco, l'*accuracy* di quasi tutti gli algoritmi tra la *clusterizzazione* a 5 e quella a 7 non decrementa troppo, per poi abbassarsi dall'8 in poi. Infatti, nella Tabella 6.1 è possibile leggere nel dettaglio le *accuracy* ottenuto con 6, 7 e 8 clusters e come questa diminuisce di circa il 6-7% per la maggior parte degli algoritmi nel passaggio tra la *clusterizzazione* a 7 e quella ad 8. Dalla *clusterizzazione* a 11 non abbiamo più picchi di *accuracy*, c'è una diminuzione dell'1-2% e anche qualche risalita in alcuni casi, come ad esempio con la *clusterizzazione* a 15 in RandomForest o quella a 20 per DecisionTree.

Dalla Figura 6.4 possiamo vedere come per la mano sinistra c'è un decremento di *accuracy* di 1-2% per ogni nuova *clusterizzazione* dalla 3 fino alla 8 per l'algoritmo RandomForest. Similmente alla mano destra, per tutti gli algoritmi c'è un picco dell'*accuracy* tra il 9% e il 12% dalla *clusterizzazione* ad 8 alla *clusterizzazione* a 9. Anche per la mano sinistra c'è una leggera risalita ma quando vengono usati 13 clusters.

6.4 Classifica della parola app

Nel capitolo precedente è stato spiegato il metodo del grafo per costruire una classifica di possibili parole in cui possiamo vedere dove si posiziona la parola reale rispetto a quella predetta. Di seguito verrà mostrato un esempio di ciò che si ottiene supponendo che la parola digitata sia *app* e venga usata per la mano destra la configurazione [4, 7, 14, 27] per RandomForest e [2, 7, 14, 27] per SVC, mentre per la mano sinistra, avendo dei risultati migliori, possiamo partire per entrambi gli algoritmi dalla *clusterizzazione* a 13, ad esempio possiamo usare la configurazione [13, 20, 27].

Per prima cosa si è visto cosa hanno dato in *output* l'algoritmo RandomForest e l'algoritmo SVC. Quindi è stato suddiviso il *dataset* in $\frac{2}{3}$ per il *training* e $\frac{1}{3}$ per il test, successivamente dal *test set* si è presa un occorrenza della parola *app*. A seguire, dal *training set* sono state rimosse le righe che contengono lo 'SPAZIO' come *View* (ω), perché non utili all'analisi che stiamo facendo. Per arrivare alla previsione della parola prima di tutto è stato allenato il modello con il *training set* appena creato, infine per la previsione è stata usata la funzione *predict* sui valori delle *features* dell'occorrenza di *app* che è stata presa dal *test set*.
Per la mano destra:

- *RandomForest* ha predetto la parola ***auu***;
- *SVC* ha predetto la parola ***apt***.

Mentre per la mano sinistra:

- *RandomForest* ha predetto la parola ***aoo***;
- *SVC* ha predetto la parola ***apo***.

Verranno discussi prima i risultati per la mano destra e successivamente quelli per la mano sinistra.

Una volta predetta la parola digitata, possiamo costruire il grafo partendo da essa e usando come configurazione: [4, 7, 14, 27] per RandomForest

e [2, 7, 14, 27] per SVC. Non viene riportato il grafo risultante per motivi di leggibilità, in quanto per RandomForest, partendo dalla *clusterrizzazione* a 4, si ottiene un grafo con più di 400 nodi, ma viene riportata la classifica finale.

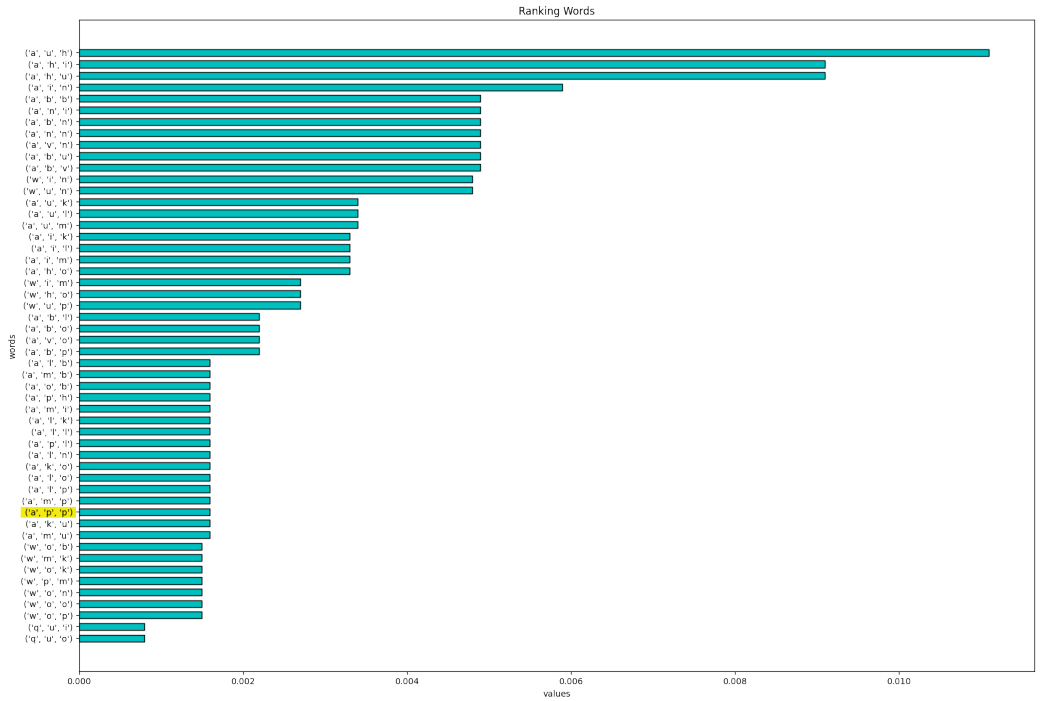


Figura 6.5: Classifica parola app con RandomForest - mano destra

Nella Figura 6.5 possiamo vedere come partendo dalla parola predetta da RandomForest *auu*, la parola *app* risulta essere tra la metà e la parte bassa della classifica, con lo stesso *score* di altre 15 parole (0.0016). Il dizionario torna molto utile in quanto sono state eliminate 380 parole da un totale di 432, tra cui anche la parola predetta dal classificatore.
Per SVC, invece, pur avendo un risultato migliore non possiamo usare la stessa configurazione di RandomForest perché con la *clusterizzazione* a 4 la lettera 'P' non è nello stesso cluster della lettera 'T', quindi la parola *app* non sarebbe presente all'interno della classifica. Per questo motivo si è scelto di usare una configurazione simile ma non uguale, cioè [2, 7, 14, 27]. Per

quanto riguarda la classifica, invece, abbiamo dei buoni risultati in quanto la parola *apt* compare in ottava posizione con lo stesso punteggio (0.0019) di *aph* e *apl*, rispettivamente in sesta e settima posizione, su un totale di 277 parole.

Come detto in precedenza, per la mano sinistra gli algoritmi hanno dato risultati migliori. Per questo motivo possiamo anche usare una configurazione composta da una *clusterizzazione* iniziale più grande. Infatti, la configurazione che è stata presa in considerazione è [13, 20, 27] per entrambi gli algoritmi. Il grafo finale risulta essere come il seguente:

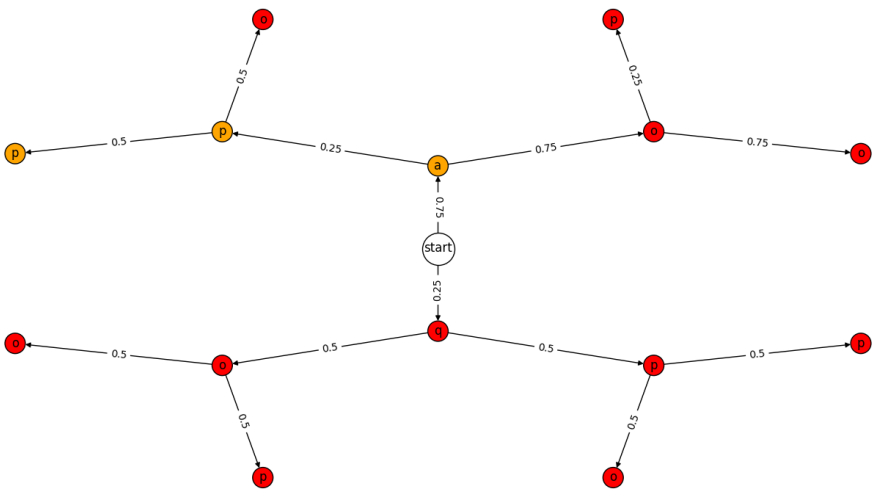


Figura 6.6: Classifica parola app con RandomForest e SVC - mano sinistra

Dalla Figura 6.6 possiamo vedere che, avendo preso in considerazione una struttura iniziale più piccola, risultano esserci molte meno parole. Tramite la cancellazione delle parole da parte del dizionario, l'unica parola che risulta esistere tra quelle del grafo è proprio la parola *app* con una probabilità dello 0.0938. Inoltre, questo risultato è valido anche per la parola predetta da SVC, cioè *apo*, perché, come si può vedere sempre dalla Figura 6.6, il cammino corrispondente porta ad una parola inesistente.

Conclusioni

In definitiva possiamo concludere che, in questo elaborato è stato mostrato come sia possibile il riconoscimento delle parole digitate da un utente acquisendo i dati generati dai sensori inerziali di un dispositivo mobile e utilizzando algoritmi di *machine learning*. I sensori che sono stati usati per l'acquisizione dei dati sono l'accelerometro, il giroscopio, il magnetometro, il sensore di orientamento e il sensore di gravità, che hanno stabilito il punto di partenza per la creazione del *dataset* e per il calcolo delle *features* utilizzate per allenare i vari modelli. Infatti, sono stati creati due *dataset*, uno per i test effettuati con l'uso della mano destra, dove sono stati acquisiti dati da tre dispositivi differenti, e l'altro per i test effettuati con la mano sinistra, con i dati acquisiti da un unico dispositivo.
Per prima cosa è stata calcolata l'accuratezza dei sette modelli di *supervised learning* utilizzando le lettere come classi. I risultati migliori, sia per la mano destra che per la mano sinistra, sono stati ottenuti dagli algoritmi RandomForest e SVC con una percentuale di riconoscimento per la mano destra pari a 37.76% e 37.93%, mentre per la mano sinistra pari a 39.43% e 41.36%.
Successivamente, è stato utilizzato anche l'*unsupervised learning* per raggruppare le lettere per similitudine, studiando quali si assomigliano in termini di *features*, partendo da un raggruppamento con 2 clusters fino ad arriva a 27 clusters. Ogni *clusterizzazione* è stata usata come *label* per i modelli di *machine learning* così che si potesse studiare di quanto diminuisse l'accuratezza dei vari algoritmi all'aumentare del numero di clusters.
L'ultima parte del progetto di tesi riprende e modifica una parte dell'articolo

[3], dove viene costruito un albero dei possibili caratteri digitati. In modo differente dell'articolo, dove per calcolare la probabilità che ogni lettera esca venivano considerati i tasti adiacenti alla lettera digitata, in questo elaborato sono stati utilizzati i cluster, calcolati precedentemente, per costruire dei grafi da cui poter calcolare la probabilità. La costruzione dei grafi parte effettivamente dal riconoscimento del testo usando gli algoritmi RandomForest e SVC, supponendo che la parola digitata sia *app*. Dalla costruzione del grafo finale viene creata una classifica delle possibili parole e da questa si può vedere in che posizione si trova la parola *app* rispetto a quella predetta dai vari classificatori.
Nonostante i risultati ottenuti non siano ottimi, costituiscono una base solida per i molteplici lavori futuri.

Lavori Futuri

Oltre all'aggiunta di nuove funzionalità, ci sono ancora molte modifiche che possono essere apportate al progetto tra cui:

- ingrandire il *dataset* per la mano sinistra, soprattutto usando diversi modelli di *smartphone*;
- bilanciare i *dataset*, perché al momento ci sono alcune lettere che vengono predette maggiormente rispetto ad altre (come la 'E' o la 'T');
- provare una classificazione con e senza spazio, per vedere se migliora l'*accuracy* dell'algoritmo;
- la modifica della scelta dei vari *clusters* per la costruzione del grafo;
- modificare il modo in cui viene costruito il grafo togliendo man mano i cammini che non servono, per poter ottimizzare il tempo di *running* dell'algoritmo;
- fare il *testing* dell'algoritmo del grafo su tutte le parole del *test set*, così da poter costruire un grafico sul rapporto tra *lunghezza della parola-quantile* (calcolato dalle varie classifiche delle parole predette).

Bibliografia

[1] WA Awad and SM ELseuofi. Machine learning methods for spam e-mail classification. *International Journal of Computer Science & Information Technology (IJCSIT)*, 3(1):173–184, 2011.

[2] Solveig Badillo, Balazs Banfai, Fabian Birzele, Iakov I Davydov, Lucy Hutchinson, Tony Kam-Thong, Juliane Siebourg-Polster, Bernhard Steiert, and Jitao David Zhang. An introduction to machine learning. *Clinical Pharmacology & Therapeutics*, 107(4):871–885, 2020.

[3] Luca Bedogni, Andrea Alcaras, and Luciano Bononi. Permission-free keylogging through touch events eavesdropping on mobile devices. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 28–33. IEEE, 2019.

[4] Liang Cai and Hao Chen. Touchlogger: Inferring keystrokes on touch screen from smartphone motion. *HotSec*, 11(2011):9, 2011.

[5] Junsung Cho, Geumhwan Cho, and Hyoungshick Kim. Keyboard or keylogger?: A security analysis of third-party keyboards on android. In *2015 13th Annual Conference on Privacy, Security and Trust (PST)*, pages 173–176. IEEE, 2015.

[6] R Divya and S Muthukumarasamy. Visual authentication using qr code to prevent keylogging. *International Journal of Engineering Trends and Technology*, 20(3):149–154, 2015.

[7] Laura Garcia Cuenca, Javier Sanchez-Soriano, Enrique Puertas, Javier Fernandez Andres, and Nourdine Aliane. Machine learning techniques for undertaking roundabouts in autonomous driving. *Sensors*, 19(10):2386, 2019.

[8] Tao Gu, Zhanqing Wu, Liang Wang, Xianping Tao, and Jian Lu. Mining emerging patterns for recognizing activities of multiple users in pervasive computing. In *2009 6th Annual International Mobile and Ubiquitous Systems: Networking & Services, MobiQuitous*, pages 1–10. IEEE, 2009.

[9] Erik G Learned-Miller. Introduction to supervised learning. *I: Department of Computer Science, University of Massachusetts*, 2014.

[10] Xuchun Li, Lei Wang, and Eric Sung. Adaboost with svm-based component classifiers. *Engineering Applications of Artificial Intelligence*, 21(5):785–795, 2008.

[11] Michael A Mahler, Qinghua Li, and Ang Li. Securehouse: A home security system based on smartphone sensors. In *2017 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 11–20. IEEE, 2017.

[12] Abayomi Moradeyo Otebolaku and Maria Teresa Andrade. User context recognition using smartphone sensors and classification models. *Journal of Network and computer applications*, 66:33–51, 2016.

[13] Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, and Joy Zhang. Accessory: password inference using accelerometers on smartphones. In *proceedings of the twelfth workshop on mobile computing systems & applications*, pages 1–6, 2012.

[14] Kelly Reynolds, April Kontostathis, and Lynne Edwards. Using machine learning to detect cyberbullying. In *2011 10th International Conference on Machine learning and applications and workshops*, volume 2, pages 241–244. IEEE, 2011.

[15] Subhanjan Saha, Samrat Chatterjee, Amit Kr Gupta, Indrajit Bhattacharya, and Tamal Mondal. Trackme-a low power location tracking system using smart phone sensors. In *2015 International Conference on Computing and Network Communications (CoCoNet)*, pages 457–464. IEEE, 2015.

[16] Muhammad Shoaib, Hans Scholten, and Paul JM Havinga. Towards physical activity recognition using smartphone sensors. In *2013 IEEE 10th international conference on ubiquitous intelligence and computing and 2013 IEEE 10th international conference on autonomic and trusted computing*, pages 80–87. IEEE, 2013.

[17] Jenni AM Sidey-Gibbons and Chris J Sidey-Gibbons. Machine learning in medicine: a practical introduction. *BMC medical research methodology*, 19(1):1–18, 2019.

[18] Amanpreet Singh, Narina Thakur, and Aakanksha Sharma. A review of supervised machine learning algorithms. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 1310–1315. Ieee, 2016.

[19] Alex Smola and SVN Vishwanathan. Introduction to machine learning. *Cambridge University, UK*, 32(34):2008, 2008.

[20] Francesco Sortino, Responsabile Innovazione RF Celada, and Competence Center Industria. Le tecniche di machine learning nella fabbrica 4.0.

[21] Yang Xin, Lingshuang Kong, Zhi Liu, Yuling Chen, Yanmiao Li, Hongliang Zhu, Mingcheng Gao, Haixia Hou, and Chunhua Wang. Machine learning and deep learning methods for cybersecurity. *Ieee access*, 6:35365–35381, 2018.

Ringraziamenti

Vorrei dedicare questo spazio a chi ha contribuito alla realizzazione di questo elaborato e a coloro che mi hanno supportato durante il mio percorso di laurea.

Innanzitutto, vorrei ringraziare il mio relatore, il Dott. Federico Montori, e correlatore, il Dott. Luca Bedogni, per la loro disponibilità e per avermi guidato durante tutto il percorso di realizzazione del progetto, dalle fasi iniziali fino alla sua stesura.

Desidero ringraziare i miei genitori, mio fratello e tutti i miei familiari che mi hanno aiutato a raggiungere anche questo obiettivo, incoraggiandomi e consolandomi nei momenti di difficoltà. Un ringraziamento speciale va ai miei *cugini-coinquilini* per i numerosi momenti di spensieratezza di questi anni.

Desidero inoltre ringraziare la famiglia Pulerà per avermi sostenuto all'inizio di questo percorso in questa bellissima città.

Infine, vorrei ringraziare tutti i miei amici e colleghi che oltre ad avermi sostenuto durante questo percorso di studi mi hanno anche offerto momenti di svago e divertimento.