SAPIENZA
UNIVERSITÀ DI ROMA

Riconoscimento del gait attraverso caratteristiche di segnali prodotti da wearable sensors

Laureando
Andrea Palermo

Relatore
Maria De Marsico

SAPIENZA
UNIVERSITÀ DI ROMA

Riconoscimento del gait attraverso caratteristiche di segnali prodotti da wereable sensors

Facoltà di informatica
Dipartimento di ingegneria informatica, informatica e statistica
Corso di laurea in informatica

Andrea Palermo
Matricola 1810218

Relatore
Maria De Marsico

A.A. 2019-2020

Sommario

1. Introduzione

1.a. Uno sguardo ai sistemi biometrici

Ai fini di comprendere appieno il lavoro illustrato nel seguito della relazione, è necessario ricordare alcune nozioni basilari di biometria. Verrano quindi descritte in breve le principali.

Il riconoscimento biometrico è alla base di un sistema informatico che consente di identificare una persona in base ad alcune sue caratteristiche principali, fisiologiche e comportamentali. Si basa su sistemi hardware per l'acquisizione dei dati cui si integrano le componenti software che consentono, attraverso diversi algoritmi, di effettuare l'analisi dei dati e ricostruire l'identità di una persona e riconoscerla.

Da questa prima definizione appare chiara la differenza con i sistemi di riconoscimento basati su ID utente e password: quelli basati sulla biometria sono sistemi con i quali si può effettivamente stabilire l'identità e l'unicità di una persona, con ID e password in realtà si sa solo che una persona possiede le informazioni per accedere ad un servizio o un'app, senza avere certezze su chi in realtà sia la persona che possiede tali informazioni di accesso.

Caratteristiche sfruttate

In generale, le caratteristiche fisiologiche sono quelle più statiche, poco variabili nel tempo, mentre quelle comportamentali subiscono variazioni e possono anche essere influenzate da fattori esterni o da particolari condizioni emotive come stress o forti impatti psicologici. Tutte queste caratteristiche devono ovviamente essere il più possibile univoche per ogni individuo per poter essere efficacemente sfruttate per un riconoscimento biometrico. In particolare:

-le caratteristiche fisiologiche più comunemente usate sono le impronte digitali, il colore e dimensione dell'iride, la retina, la sagoma della mano, la forma dell'orecchio, la fisionomia del volto;

-tra quelle comportamentali, invece, troviamo l'impronta vocale, la scrittura, lo stile di battitura sulla tastiera, i movimenti del corpo, ed infine l'andamento della camminata (anche se va detto che alcuni approcci basati sulla computer vision utilizzano anche elementi fisici), di cui recenti studi sembrano confermare l'univocità. Quest'ultima è la caratteristica sfruttata dal prototipo di sistema biometrico realizzato durante il tirocinio.

Casi d'uso

Il funzionamento dei sistemi di riconoscimento biometrico varia in funzione dell'obiettivo che essi si pongono, che può essere la verifica dell'identità oppure l'identificazione (che può essere di tipo *open set* o *closed set*) di una persona. In breve, questi tre tipi di possibile obiettivo del sistema di riconoscimento biometrico consistono in:

- Verifica: la persona che richiede l'accesso al sistema dichiara la propria identità, quindi il suo riconoscimento diventa un processo di verifica (uno a uno) che richiede un match tra i dati acquisiti in tempo reale dai sensori e quelli presenti in un archivio (d'ora in poi *gallery*);

- Identificazione *closed set*: il riconoscimento biometrico avviene facendo il match tra l'immagine, i dati, o le informazioni acquisite in tempo reale con tutte le analoghe informazioni (template) presenti in un archivio (verifica uno a molti). Il sistema di riconoscimento biometrico associerà l'identità facendo il confronto e identificando le caratteristiche fisiologiche e comportamentali più simili e coerenti tra quelle in archivio e quelle raccolte in tempo reale. *Closed set* sta ad indicare che si

assume che tutti gli individui che richiedono il riconoscimento biometrico siano registrati, ovvero presenti nella gallery. Questo significa che il sistema non rifiuterà mai un utente, a meno che i dati acquisiti non siano di qualità troppo bassa.

- Identificazione *open set*: simile all'identificazione *closed set*, ma in questo caso non si assume che tutti gli individui che richiedono il riconoscimento biometrico siano presenti nella gallery. Per questo, il rifiuto dei soggetti è ovviamente possibile.

1.b. Obiettivi del tirocinio

Il lavoro svolto durante il tirocinio ha avuto come scopo lo sviluppo e la valutazione di un sistema di riconoscimento biometrico della camminata, in particolare in modalità di verifica dell'identità, per poi poterlo confrontare con altre soluzioni già esistenti e studiarne l'usabilità.

Possibili approcci

Dato che lo scopo principale per cui sono pensati ed utilizzati i sistemi di riconoscimento della camminata è l'autenticazione dei possessori di dispositivi mobili, la maggior parte degli approcci esistenti si concentra sulla verifica dell'identità; alcuni affrontano anche il problema dell'identificazione *closed set*, mentre quasi nessuno si occupa anche dell'identificazione *open set*.

Gli approcci adottati finora si dividono principalmente in due categorie: quelli basati su computer vision (analisi dei video ripresi mentre i soggetti camminano) e quelli che sfruttano invece dei sensori indossabili, come ad esempio accelerometro e giroscopio, e vanno ad analizzare i dati da essi prodotti.

In questo caso si è scelto di utilizzare una tecnica di riconoscimento basata su sensori indossabili (in particolare l'accelerometro), poiché allo stato attuale della ricerca sembra avere svariati vantaggi rispetto all'utilizzo di

computer vision. Tra questi, l'analisi delle sequenze video relative alle camminate presenta innanzitutto molti più elementi di disturbo rispetto a quella dei dati dei sensori, ad esempio l'illuminazione o qualsiasi elemento che possa variare la silhouette o i movimenti dell'individuo, come una posa particolare, dei vestiti larghi o un oggetto trasportato. Altro svantaggio di questo tipo di soluzione è che non rende possibile l'autenticazione di un individuo direttamente dal suo dispositivo mobile personale, poiché c'è la necessità di un altro dispositivo esterno che riprenda la camminata.

Questo non significa che gli approcci basati su sensori indossabili siano privi di inconvenienti: anche in questo caso troviamo molti elementi di disturbo, come il tipo di scarpe utilizzate, l'inclinazione del terreno o alcune condizioni di salute del soggetto; il vantaggio di questa scelta, tuttavia, è dato dal fatto che tutti questi elementi creano difficoltà ad entrambi i tipi di approccio, mentre quelli menzionati sopra sono relativi esclusivamente alla computer vision.

L'approccio basato su sensori indossabili si suddivide ulteriormente in due branche di ricerca: in breve, la prima tenta di calcolare direttamente un valore di somiglianza o distanza tra la serie temporale da verificare (d'ora in poi "*probe*") e quelle registrate nel sistema attraverso algoritmi di confronto tra vettori; la seconda invece utilizza una tecnica basata sul machine learning, in cui viene allenato un classificatore per ogni soggetto che si occuperà poi di autenticarlo o rifiutarlo. Spesso l'allenamento avviene utilizzando direttamente le serie temporali, ma in (De Marsico,

Fartade, Mecca, 2018) viene studiata un'altra soluzione, la quale estrae un insieme di caratteristiche (anche dette *feature*) dalle serie (e seleziona le più rilevanti) prima di usarle per l'allenamento dei classificatori. In questo caso si è scelto di proseguire su questa strada, provando ad utilizzare diverse tecniche di estrazione delle caratteristiche e algoritmi di machine learning.

1.c. Strumenti utilizzati

Dataset

I dati necessari al funzionamento del sistema provengono dal dataset *ZJU-GaitAcc* (Zhang *et al.*, 2015), contenente le serie temporali relative alle camminate di 175 soggetti. Dei 175, 153 hanno partecipato a due sessioni di raccolta dei dati, mentre gli altri 22 solamente ad una.

In ogni sessione relativa ad ogni soggetto sono stati raccolti 6 record di serie temporali, ognuno di essi contenente 5 serie, composte da valori di accelerazione catturati sui tre assi x, y e z da 5 diversi accelerometri posizionati su polso sinistro, spalla sinistra, lato destro del bacino, coscia sinistra e caviglia destra.

In questo caso si è deciso di utilizzare solo le serie catturate dall'accelerometro posto sul bacino, poiché dai risultati di altri studi (De Marsico, Fartade, Mecca, 2018) effettuati precedentemente erano risultati i più efficaci.

Linguaggi e librerie

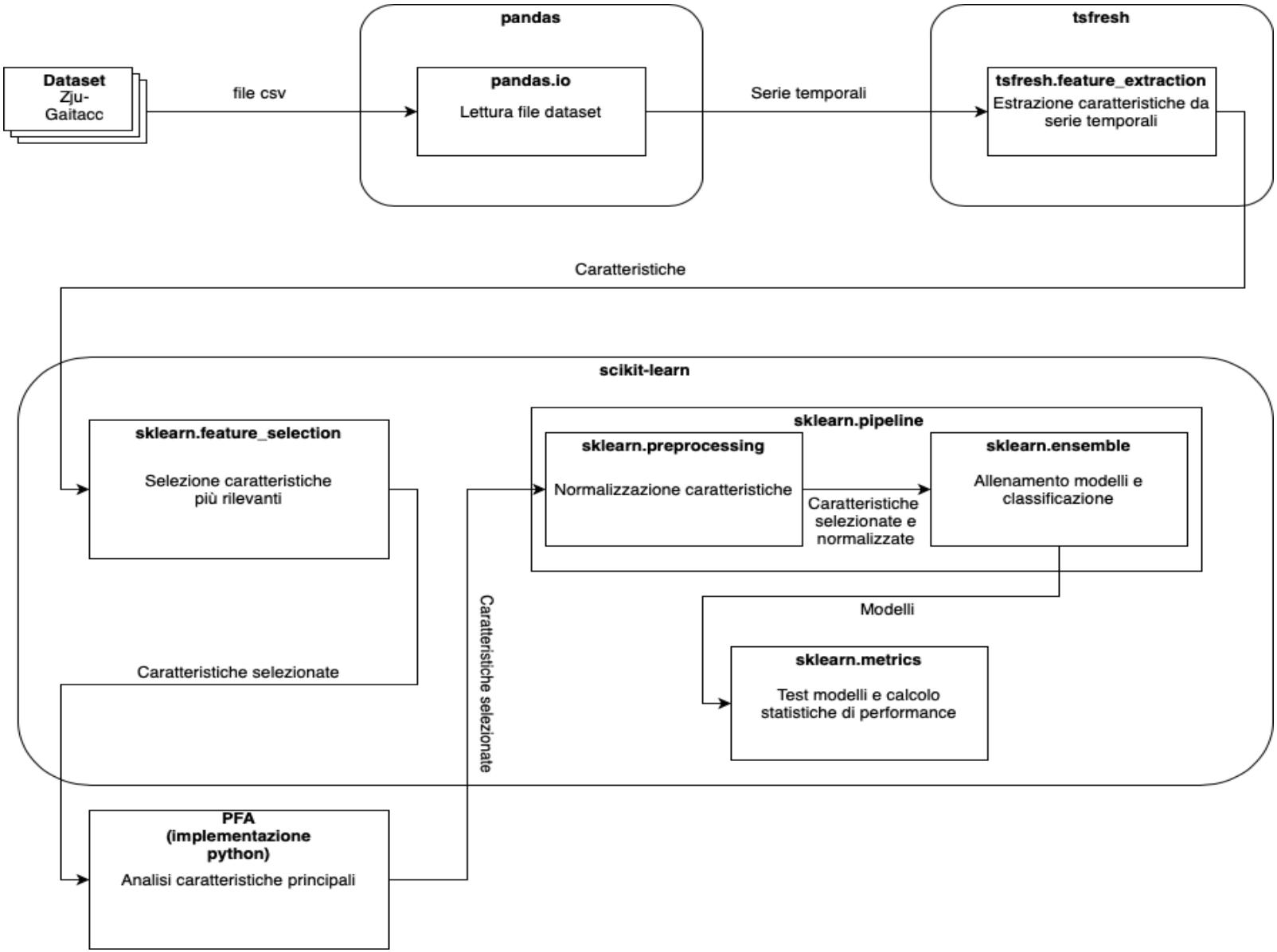
Il sistema è stato sviluppato quasi interamente utilizzando Python 3.8, eccetto alcuni brevi task iniziali di manipolazione dei file contenenti i dati relativi alle camminate effettuati in *bash*.

In particolare, la parte di estrazione delle caratteristiche dalle serie temporali è stata realizzata con l'ausilio della libreria *tsfresh*[1], mentre la selezione delle più rilevanti, l'allenamento e il test dei classificatori sfruttano la libreria *scikit-Learn*[2] e, per alcuni task specifici, *imbalanced-learn*[3] (tutte scritte in Python).

La libreria *Pandas*[4] è inoltre stata ampiamente utilizzata per la lettura e la manipolazione dei dati memorizzati su file.

Il seguente schema illustra brevemente l'interazione del sistema con tutti gli strumenti menzionati:

[1] https://tsfresh.readthedocs.io/en/latest/
[2] https://scikit-learn.org/stable/
[3] https://imbalanced-learn.readthedocs.io/en/stable/api.html
[4] https://pandas.pydata.org


pandas
tsfresh
Dataset
Zju-
Gaitacc
file csv
pandas.io
Lettura file dataset
Serie temporali
tsfresh.feature_extraction
Estrazione caratteristiche da
serie temporali
Caratteristiche
scikit-learn
sklearn.feature_selection
Selezione caratteristiche
più rilevanti
sklearn.pipeline
sklearn.preprocessing
Normalizzazione caratteristiche
Caratteristiche
selezionate e
normalizzate
sklearn.ensemble
Allenamento modelli e
classificazione
Modelli
sklearn.metrics
Test modelli e calcolo
statistiche di performance
Caratteristiche selezionate
Caratteristiche selezionate
PFA
(implementazione
python)
Analisi caratteristiche principali

2. Lavoro svolto

Passiamo ora alla descrizione dettagliata del lavoro svolto per lo sviluppo del sistema durante il tirocinio.

2.a Manipolazione del dataset

All'inizio del progetto è stato necessario un breve lavoro di manipolazione del dataset *ZJU-GaitAcc*, allo scopo di renderlo un input accettabile per la libreria tsfresh, che si sarebbe poi occupata dell'estrazione delle caratteristiche. Il lavoro è stato svolto realizzando brevi script *bash* ad hoc per le funzioni di manipolazione dei file csv contenuti nel dataset necessarie. Esso è consistito in:

- riunire tutte le serie temporali in un unico file csv ponendole semplicemente una dopo l'altra, ma assegnando ad ognuna un identificatore univoco per poterle poi distinguere;
- mantenere, in un altro file, una corrispondenza tra l'identificatore delle serie e i soggetti da cui esse sono state catturate (nel dataset originale, l'informazione sull'appartenenza delle serie era contenuta esclusivamente nella struttura delle cartelle), inserendo nel file i numeri identificativi dei soggetti nello stesso ordine in cui appaiono le serie temporali nel file ottenuto al passo precedente.

2.b. Estrazione delle caratteristiche

Una volta riadattato il dataset come descritto, si è proseguito con una fase di estrazione delle caratteristiche dalle serie temporali.

Esse consistono in dei valori aggregati ottenuti dalle serie per mezzo di funzioni messe a disposizione dalla libreria *tsfresh*[5]; come già specificato, le serie contengono i valori di accelerazione catturati in ogni istante su tutti e tre gli assi x, y e z, quindi ogni possibile caratteristica viene calcolata relativamente ad ogni asse.

Tramite questo processo si ottengono complessivamente 2289 caratteristiche da ogni serie temporale (763 per asse). Non è ovviamente possibile elencarle tutte: per questo verranno in seguito descritte solo alcune di quelle selezionate come più rilevanti, in ogni caso una lista completa è disponibile sulla documentazione di *tsfresh*[6].

Tutte le caratteristiche ottenute in questo modo sono state salvate in un ulteriore file csv: ricapitolando, ogni serie temporale è stata convertita in una lista di caratteristiche, che rappresenterà una riga in quest'ultimo file. Quindi, ai fini dell'allenamento e test del sistema, la *gallery* e le *probe* non saranno rappresentate dalle serie di valori di accelerazione catturati sui tre assi, ma piuttosto da un insieme di caratteristiche estratte da esse.

[5] https://tsfresh.readthedocs.io/en/latest/api/tsfresh.feature_extraction.html
[6] https://tsfresh.readthedocs.io/en/latest/text/list_of_features.html

2.c Selezione delle caratteristiche rilevanti

Una volta ottenuto un vettore di caratteristiche per rappresentare ognuna delle camminate registrate, si è presentata la necessità di una fase di filtraggio e selezione delle più rilevanti ai fini della verifica dell'identità tra di esse.

Infatti, provando ad allenare e testare dei classificatori con i vettori costituiti da tutte e 2289 le caratteristiche ottenute, si ottenevano risultati assolutamente insoddisfacenti.

A questo scopo sono stati fatti due tentativi, per poter scegliere quello che sarebbe risultato nelle performance migliori. Entrambi iniziano ovviamente con la lettura del file csv ottenuto al passo precedente, effettuata grazie alla libreria pandas.

Il primo sfrutta ulteriormente la libreria *tsfresh*, la quale mette a disposizione funzioni per individuare le caratteristiche più rilevanti all'interno di un insieme (date le identità relative ad ognuno dei vettori, necessarie appunto per stabilire in base a cosa calcolare la priorità delle caratteristiche). In realtà la libreria dà la possibilità di unificare questa fase e la precedente, quindi di estrarre dall'inizio solo le caratteristiche ritenute più rilevanti. In questo modo vengono mantenute 1421 delle 2289 caratteristiche iniziali.

Il secondo tentativo sfrutta invece funzionalità simili messe però stavolta a disposizione dalla libreria *scikit-learn*, la quale permette di scendere più in profondità nella scelta delle tecniche di selezione, a differenza di *tsfresh*

che mette invece a disposizione solamente la funzione vista in precedenza, modificabile in maniera lieve unicamente utilizzando i parametri che prende in input. Si è deciso di preferire questo secondo approccio al primo descritto in precedenza, in quanto riduce maggiormente il numero di caratteristiche e risulta in performance di classificazione migliori.
La selezione si suddivide in diverse fasi:

- La prima consiste nell'eliminazione di tutte le caratteristiche che presentano, su tutti i vettori relativi alle camminate, una varianza pari a 0, ovvero quelle costanti. Questo viene realizzato comodamente tramite la funzione *VarianceTreshold*[7], la quale permette di eliminare tutte le caratteristiche con varianza al di sotto di quella specificata direttamente all'interno di un dataframe pandas (il risultato della lettura del file csv con le camminate). Questo passaggio è ovviamente fondamentale perché se una caratteristica è costante su tutte le camminate, non conterrà sicuramente informazione di valore per la fase di apprendimento dei classificatori.

  Al termine di questa fase vengono mantenute 2158 delle 2289 caratteristiche inizialmente presenti.

[7] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html

- Il passaggio successivo tenta di nuovo di determinare l'importanza delle caratteristiche basandosi sullo studio della varianza, ma con un approccio ben diverso.

  In particolare, utilizzando la funzione *SelectPercentile*[8] di *scikit-learn*, vengono mantenute solo il 45% delle caratteristiche analizzate, basandosi su un punteggio assegnato da una funzione messa a disposizione dalla libreria, chiamato *ANOVA F value*[9]. Questa funzione assegna il punteggio *F* ad ogni caratteristica effettuando il seguente calcolo:

  $$F = (\text{varianza inter-gruppi})/(\text{varianza intra-gruppo})$$

  dove "varianza inter-gruppi" sta ad indicare la media delle distanze dei valori medi delle singole caratteristiche dal valor medio globale di tutti i campioni di ogni caratteristica, elevata al quadrato; "varianza intra-gruppo" è la normale varianza delle caratteristiche vista in precedenza.

  Al termine di questo calcolo vengono quindi mantenute il 45% delle caratteristiche in input che hanno ottenuto il punteggio più alto.

  Questa tecnica è stata scelta ed utilizzata poiché sulla documentazione di *Scikit-Learn* era consigliata per il calcolo della rilevanza di caratteristiche quantitative (come ovviamente sono

[8] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectPercentile.html
[9] https://en.wikipedia.org/wiki/F-test

quelle relative alle camminate), ed ha effettivamente dato risultati migliori di altre funzioni di calcolo del punteggio che la libreria metteva a disposizione, come *chi*^{2}.

Le caratteristiche mantenute dopo questo passaggio passano dalle precedenti 2158 a 971.

- A questo punto ha luogo l'ultima fase di selezione, questa volta basata sulla tecnica chiamata *Principal Feature Analysis* (PFA). Essa si basa sulla simile *Principal Component Analysis* (PCA).

  Entrambi gli approcci hanno come scopo quello di ridurre in qualche modo la dimensione del dataset: nel seguito viene illustrata brevemente la differenza tra le due procedure.

  Innanzitutto, il dataset è interpretato come una matrice X di dimensione m∗n, ovvero costituita da n vettori $\vec{x_i}$ di dimensione m (i vettori di caratteristiche rappresentanti le camminate), dove m è il numero di caratteristiche e n il numero di camminate registrate.

  Segue quindi una trasformazione della matrice X in una nuova matrice Z, volta a portare a 0 la media dei valori di tutte le colonne di X: questo si ottiene sottraendo ad ogni valore la media dei valori di tutta la colonna.

  La PCA lavora trasformando lo spazio delle caratteristiche in uno di dimensione minore (che chiameremo q, input dell'algoritmo), i cui assi sono rappresentati dai q autovettori con associati autovalori più elevati della matrice di covarianza ottenuta a partire da Z: essi

non hanno correlazione con le caratteristiche inizialmente presenti nel dataset.

La PFA, invece, prende in input un numero p di caratteristiche che si vogliono mantenere, e ripete la procedura appena vista così com'è (utilizzando come q un valore di poco inferiore a p), per poi proseguire a partire dalla matrice A_q, avente per colonne gli autovettori individuati come assi.

Lo scopo è infatti quello di analizzare la struttura dei componenti principali appena ottenuti dall'insieme completo di caratteristiche per trovarne il sottoinsieme di dimensione p che sia maggiormente informativo.

Denominiamo ora le righe della matrice A_q $V_1, \dots, V_n$:

il vettore V_i consiste nella proiezione dell'i-esima caratteristica della matrice X nel nuovo spazio di dimensione minore: ciò significa che i q elementi di V_i rappresentano i pesi che l'i-esima caratteristica ha rispetto ad ogni asse del nuovo spazio.

A questo punto, gli stessi vettori $V_1, \dots, V_n$ vengono raccolti in p gruppi in base alla loro somiglianza per mezzo dell'algoritmo *K-means*: per ogni gruppo creato in questo modo, si sceglie quindi il vettore più vicino alla media del gruppo, e si mantiene la caratteristica ad esso associata, ottenendo in questo modo le p caratteristiche desiderate.

Una descrizione più dettagliata di ogni passaggio dell'algoritmo, che richiede alcune conoscenze di algebra non basilari, è fornita in (Lu *et al.*, 2007).

Per quanto riguarda l'implementazione utilizzata, essa consiste in una traduzione minuziosa dei passi descritti come pseudocodice in (Lu *et al.*, 2007) in Python, dato che nessuna libreria mette attualmente a disposizione funzioni che si occupino di *Principal Feature Analysis*.

Al termine di quest'ultima fase, le caratteristiche definitivamente mantenute sono in tutto 400.

A questo punto il lavoro di selezione è terminato, ma vale la pena di menzionare alcuni altri approcci che sono stati testati ma poi non utilizzati nella versione finale del sistema in quanto privi di efficacia o meno efficaci di quelli già visti:

- Un'altra tecnica studiata sfrutta un'idea del tutto diversa da quelle viste: si basa infatti sul fatto che alcuni algoritmi di machine learning assegnino già di per sé un coefficiente di rilevanza alle caratteristiche presenti nel dataset che viene loro sottoposto, che utilizzeranno poi ai fini della classificazione o regressione; si può quindi allenare uno di questi modelli, anche se poi non sarà effettivamente utilizzato per il riconoscimento

delle camminate, semplicemente per ottenere i coefficienti ed eliminare le caratteristiche che hanno riportato quelli minori.

In questo caso si è provato ad utilizzare un classificatore *LinearSVC* (*C-Support Vector* con kernel lineare), per mezzo dell'implementazione fornita da *scikit-learn*[10], mentre la selezione vera e propria delle caratteristiche con i coefficienti migliori viene effettuata tramite la funzione di comodità *SelectFromModel()* (della libreria stessa) che prende in input il classificatore e può essere inserita all'interno della pipeline.

Essa scarta tutte le tutte le caratteristiche con coefficiente inferiore ad una soglia (anch'essa input della funzione): in questo caso si è scelto di utilizzare quella scelta di default dalla libreria (0,00001), basandosi su delle euristiche che prendono in considerazione anche il tipo di classificatore utilizzato.

Altre opzioni erano la media o la mediana dei valori dei coefficienti.

- L'ultimo approccio che vale la pena citare è l'utilizzo della funzione *SelectPercentile* vista in precedenza con altre funzioni di calcolo del punteggio delle caratteristiche.

  Scikit-learn ne mette a disposizione numerose (la lista completa è disponibile sulla documentazione[11]).

[10] https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
[11] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectPercentile.html

In questo caso sono state provate *chi*^{2} e quella vista in precedenza basata su *ANOVA F-value,* rivelatasi migliore.

In questo modo si conclude l'illustrazione della fase di selezione delle caratteristiche rilevanti.

2.d Normalizzazione delle caratteristiche

Una volta selezionate le caratteristiche più rilevanti da utilizzare per la classificazione, è necessario un lavoro di normalizzazione sui loro valori. Questo è indispensabile in quanto, anche dopo il filtraggio, si ha a che fare con un gran numero di caratteristiche: se esse si presentano in scale molto diverse, i loro valori potrebbero non venire interpretati correttamente ai fini dell'allenamento del modello.

A questo scopo, la formula utilizzata per la normalizzazione di un valore x di una caratteristica è:

$$\frac{x-u}{s}$$

dove u è la media di tutti i valori assunti dalla caratteristica e s è la loro deviazione standard.

Per comodità questa procedura è stata di nuovo inserita nella pipeline già creata in precedenza per la selezione delle caratteristiche, ed è messa a disposizione dalla classe *StandardScaler*[12] di *scikit-learn*.

[12] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

2.e Partizione del dataset

Dopo aver selezionato e normalizzato le caratteristiche delle camminate, ai fini della classificazione è necessario dividere il dataset risultante in due insiemi: uno per l'allenamento dei modelli, l'altro per il test degli stessi.
La proporzione scelta per la divisione è 70:30 (il 70% delle camminate sarà usato per l'allenamento ed il restante 30% per i test).
Anche se, come già specificato, ci sarà bisogno di allenare un modello per ogni soggetto di cui si vuole verificare l'identità, gli insiemi creati in questo modo rimangono sempre costanti (non vengono creati nuovamente per ogni modello da allenare).
Inoltre, allo scopo di ottenere una maggiore efficacia dagli algoritmi di machine learning, si è avuto cura di far sì che le camminate dei diversi soggetti fossero equamente distribuite all'interno dei due insiemi. Quindi l'insieme di allenamento contiene circa il 70% delle camminate di ogni soggetto e quello di test contiene il rimanente 30%.
Il compito di divisione è svolto dalla funzione *train_test_split()*[13] di *scikit-learn*, la quale consente di provvedere a tutte le necessità appena viste: essa prende infatti in input il dataset, la proporzione desiderata per la divisione ed un parametro *stratify* che si occupa della distribuzione equa dei soggetti.

[13] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

2.f Allenamento e test dei modelli

Una volta diviso il dataset nei due insiemi, può iniziare la fase di allenamento dei modelli per la classificazione delle camminate.
Dato che è necessario allenare un modello per ogni individuo, inizia ora una procedura che si ripeterà invariata per ogni individuo X presente nel dataset.
Segue una descrizione dettagliata di ogni passo:

1. A questo punto del lavoro, ogni vettore relativo ad una camminata è accompagnato nel dataset da un intero compreso in [1,153] che serve ad identificare il soggetto in questione. Dovendo allenare un modello per individuo, tuttavia, c'è bisogno per ognuno di essi di andare a modificare queste etichette, binarizzandole rispetto all'identità di cui il modello si deve occupare. In particolare, per il soggetto X, tutte le etichette pari ad X vengono trasformate in 1, mentre tutte le altre in 0.

   Ciò avviene sia per l'insieme di allenamento che per quello di test, creandone ovviamente delle copie poiché essi, come già visto, dovranno essere utilizzati per tutti i modelli.

2. Fatto ciò, si presenta subito un'ulteriore problematica, sempre dovuta alla necessità di avere un modello per ogni soggetto: dato che l'insieme di allenamento è costituito da camminate relative a

tutte le identità equamente distribuite, quando si va a binarizzare il problema (le etichette) rispetto ad un'identità X, nel set di training rimarranno molte più camminate con identità=0 (diversa da X) che con identità=1 (X).

Tecnicamente si può dire di essere di fronte ad un task di apprendimento sbilanciato, ovvero si sta cercando di classificare istanze le cui classi compaiono nel dataset in numero molto diverso tra loro.

La soluzione adottata più spesso in questi casi è l'attuazione di un cosiddetto *resampling* del dataset. Questa tecnica può essere utilizzata in due diverse varianti:

- la prima è chiamata *downsampling* e consiste semplicemente nell'eliminazione di alcune istanze della classe maggioritaria fino al raggiungimento di una proporzione desiderata.

  Non è tuttavia utilizzata molto spesso in quanto, in applicazioni di machine learning, la quantità di dati a disposizione è quasi sempre fondamentale, ed è quindi difficile che convenga rinunciare a parte di essi.

  Questo è il motivo per cui non è stata scelta per lo sviluppo del sistema in questione.
- la seconda variante è invece chiamata *upsampling* e, come dice il nome, consiste nell'aumento delle istanze della classe minoritaria per mezzo di diverse strategie. Questo risolve il

problema della perdita di dati preziosi visto per il *downsampling*.

Le strategie normalmente utilizzate a questo scopo sono svariate: la meno elaborata e più conosciuta è quella che va semplicemente a duplicare alcune istanze scelte all'interno della classe minoritaria; il suo unico vantaggio, tuttavia, è quello di bilanciare la distribuzione delle istanze, senza però portare informazioni aggiuntive all'insieme di allenamento.

Per questo si è deciso di adottare un'altra strategia ai fini dell'*oversampling* del dataset: essa consiste nell'applicazione di un algoritmo chiamato SMOTE, che è in grado di creare istanze sintetiche della classe minoritaria a partire da quelle già presenti.

L'implementazione utilizzata[14] è fornita dalla libreria *imbalanced-learn*.

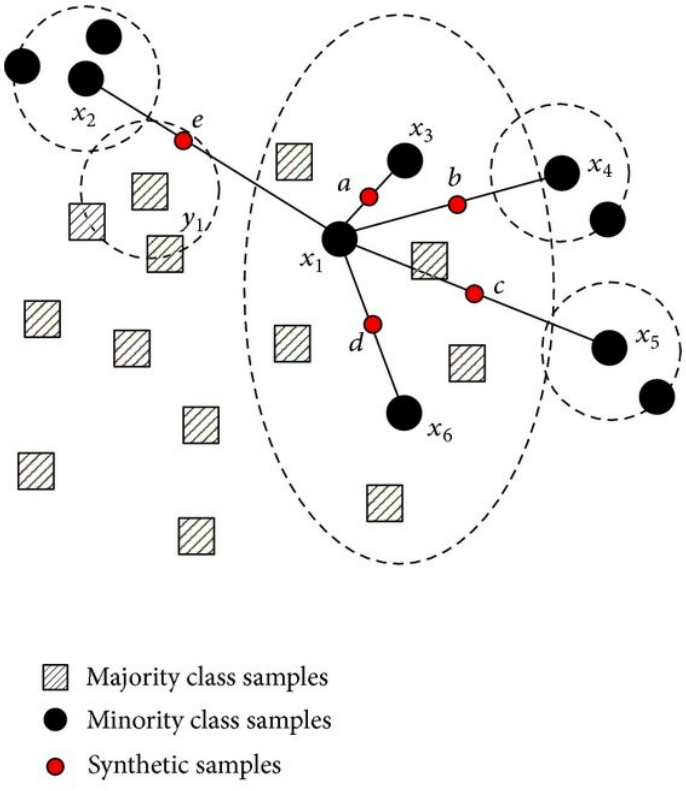
Effettivamente, andando a testare i modelli con e senza l'applicazione dell'*oversampling*, si è potuto notare che le performance di classificazione giovassero moderatamente dell'applicazione di questa tecnica.

L'algoritmo si basa su un'idea molto semplice: inizia selezionando randomicamente un'istanza I nella classe minoritaria e cercando le k istanze più vicine nello spazio

[14] https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html

delle caratteristiche (k è un parametro dell'algoritmo, in questo caso si è utilizzato 5), sempre all'interno della classe minoritaria. A questo punto sceglie randomicamente uno dei k vicini trovati e lo collega all'istanza I tramite un segmento; la nuova istanza sintetica viene ottenuta selezionando, di nuovo randomicamente, un punto appartenente al segmento (diverso ovviamente dagli estremi).

L'immagine seguente, pubblicata in (Hu *et al*, 2013), visualizza in maniera molto chiara il funzionamento del procedimento di creazione delle nuove istanze:



3. Una volta riorganizzata la distribuzione delle istanze come visto, si prosegue con l'allenamento vero e proprio del modello che si occuperà della verifica dell'identità X.

   L'algoritmo scelto per la classificazione è noto come AdaBoost-SAMME, presentato in (Hastie *et al*, 2009).

   Esso lavora nel seguente modo: inizia assegnando ad ogni istanza un peso pari a 1/n, se n è il numero totale delle istanze. Sempre il peso andrà a calibrare la funzione di decisione (maggiore è il suo valore, più un'istanza dovrà essere considerata rilevante dalla funzione); prosegue quindi allenando un classificatore con i pesi originali e testandolo. Grazie ai risultati dei test, i pesi vengono ricalibrati in base alle istanze classificate erroneamente, aumentandone i pesi. I pesi modificati vengono a questo punto utilizzati per allenare varie altre copie del modello che si concentreranno maggiormente sui casi difficili, ed infine verrà scelto quello che mostra performance migliori.

   L'implementazione utilizzata è messa a disposizione da *scikit-learn* tramite la classe *AdaBoostClassifier*[15].

   Questa soluzione è stata ovviamente scelta confrontandola con altri tipi di algoritmi di classificazione, che vale la pena menzionare anche se poi non effettivamente utilizzati, come un semplice albero

[15] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

decisionale[16] creato con l'algoritmo C4.5 o un classificatore *Random Forest*[17].

4. Il prossimo passo riguarda un'ulteriore calibrazione della funzione di decisione; ai fini del test del sistema, non basta che i modelli predicano la classe di appartenenza, ma c'è bisogno di avere a disposizione le probabilità di appartenenza alla classe positiva e negativa.

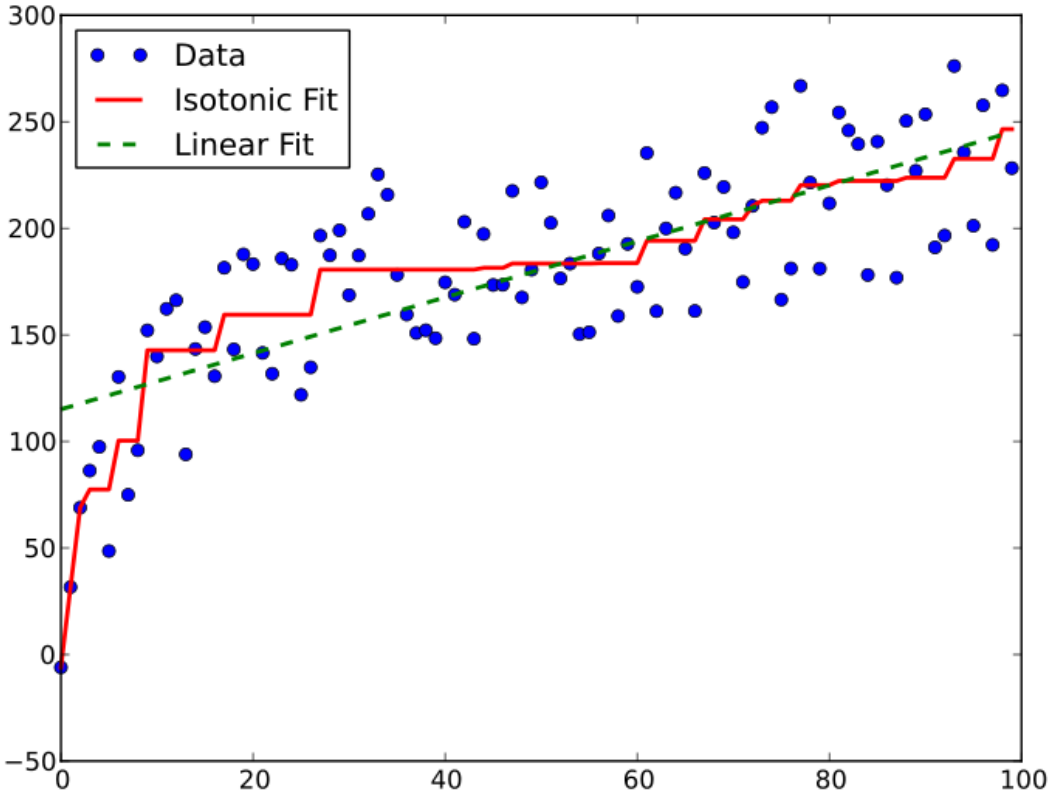
   In questo ambito, ogni algoritmo di machine learning produce valori calibrati in maniere differenti, spesso non compresi tra 0 ed 1 (come nel caso di *AdaBoost-SAMME*).

   L'obiettivo della calibrazione è di far si che le probabilità predette possano essere direttamente utilizzate come valori di confidenza: ciò significa ad esempio che, tra le istanze la cui probabilità predetta è vicina a 0.6, all'incirca il 60% di esse dovranno effettivamente appartenere alla classe positiva.

   La tecnica utilizzata per la calibrazione è la regressione isotonica, fornita dalla classe *CalibratedClassifierCV*[18] di *scikit-learn*; essa traccia una linea sul grafico delle osservazioni (i valori ottenuti senza calibrazione) rispettando due condizioni: la linea deve essere non decrescente e il più vicina possibile alle osservazioni.

[16] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
[17] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[18] https://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html

La seguente figura[19] illustra il funzionamento della procedura di calibrazione (e la confronta con una regressione di tipo lineare, mostrando come quella isotonica si possa adattare di più ai tratti più "pendenti" della distribuzione delle osservazioni):



[19] Fonte: https://en.wikipedia.org/wiki/Isotonic_regression

5. Terminata anche la calibrazione dell'output del classificatore, può avere inizio la fase di test.

   Tutte le istanze presenti nell'insieme di test vengono sottoposte alla classificazione, simulando che esse dichiarino tutte identità X (alcune correttamente e altre no) e debbano quindi essere accettate o rifiutate.

   La funzione di decisione dei modelli restituisce quindi le probabilità (già calibrate come visto) che ogni istanza sia relativa all'identità X o meno.

   A questo punto le fasi di allenamento e test per l'identità X possono considerarsi concluse: per ora ci si limita a memorizzare le probabilità ottenute, che saranno poi utilizzate nel calcolo delle statistiche di performance complessive (per tutte le identità).

2.g Tecniche di valutazione

Una volta completata la procedura appena illustrata per tutti i 153 soggetti presenti nel dataset, si hanno a disposizione le probabilità predette per ogni camminata presente nell'insieme di test da tutti i modelli (quindi, di ogni camminata si conoscono la probabilità che appartenga o meno ad ogni soggetto, dato che esiste un modello per ognuno di essi).

Questo è il punto di partenza per la fase di valutazione del sistema, indispensabile per lo studio delle performance ed il confronto con altre soluzioni.

Normalmente, l'accettazione o il rifiuto di un'istanza in un sistema biometrico seguono la seguente logica: si deve avere a disposizione una misura di somiglianza o distanza tra l'istanza in questione e tutte le altre istanze relative allo stesso individuo registrate nel sistema (solitamente viene usato ripetutamente lo stesso algoritmo di confronto tra vettori ed il risultato migliore viene scelto come punteggio di distanza o somiglianza). Viene inoltre scelta una soglia compresa nel possibile range di valori delle somiglianze/distanze: se il punteggio ottenuto è migliore, l'istanza viene accettata, altrimenti viene rifiutata.

In questo caso la tecnica utilizzata è molto simile, poiché si utilizzano le probabilità predette dai modelli come i valori di somiglianza appena descritti, quindi il resto del procedimento rimane invariato.

La scelta della soglia da utilizzare è ovviamente cruciale ed esistono delle tecniche standard che consentono di sceglierne una efficace, provandone

diverse ed andando ad osservare il comportamento del sistema al variare di esse: questa fase del lavoro sarà spiegata dettagliatamente nel seguito dell'illustrazione della fase di valutazione.

Per poter comprendere i risultati ottenuti, è tuttavia necessario fare riferimento ad alcune nozioni di base riguardo alle metriche di performance più usate nel campo dei sistemi biometrici.

Nel seguito sono elencate le principali, con una breve spiegazione del loro significato:

- FAR – *False acceptance rate*: è definito come la percentuale di istanze testate che si sono concluse con una falsa accettazione dell'individuo (la camminata non appartiene al soggetto di cui si occupa il modello, ma viene comunque verificata come genuina).

  Questo significa che ad esempio, se il FAR è pari all'1%, all'incirca un impostore su 100 tra quelli che provano ad ottenere l'accesso al sistema riuscirà effettivamente ad ottenerlo. Dipende ovviamente dalla soglia scelta.
- FRR – *False rejection rate:* è definito come la percentuale di istanze testate che si sono concluse con un falso rifiuto dell'individuo (la camminata appartiene al soggetto di cui si occupa il modello, ma viene comunque rifiutata).

  Questo significa che ad esempio, se il FRR è pari all'1%, all'incirca un soggetto autorizzato su 100 che provano ad

ottenere l'accesso al sistema verrà rifiutato. Dipende anch'esso ovviamente dalla soglia scelta.

- GAR – *Genuine acceptance rate*: è definito come la percentuale di istanze testate che si sono concluse con una corretta accettazione dell'individuo (la camminata appartiene al soggetto di cui si occupa il modello e viene riconosciuta come genuina).

  Questo significa che ad esempio, se il GAR è pari al 90%, all'incirca 90 soggetti autorizzati su 100 che provano ad ottenere l'accesso al sistema vengono effettivamente riconosciuti come genuini. Per come è definito, è ovvio che si possa ottenere calcolando 1 - FAR. Dipende anch'esso ovviamente dalla soglia scelta.
- HTER – *Half total error rate*: è definito come la media tra FAR e FRR ed è utile per avere una visione cumulativa dei due tipi di errore. Dipende anch'esso ovviamente dalla soglia scelta.
- EER – *Equal error rate*: come già visto, è utile provare ad utilizzare diverse soglie per poter determinare la migliore: a questo scopo è molto utile il calcolo dell'EER.

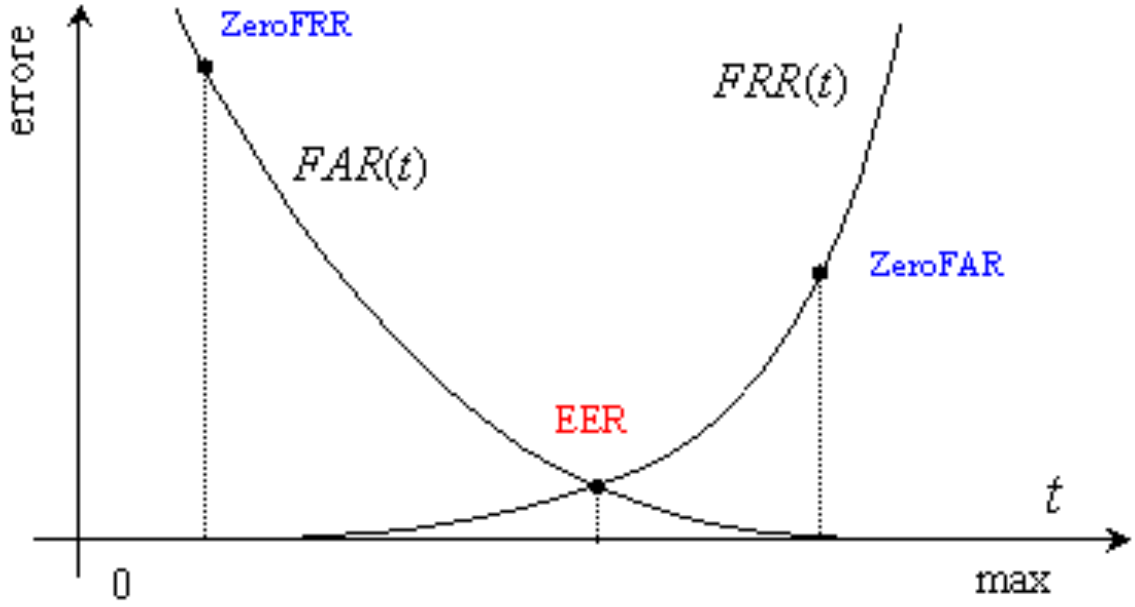
  Esso è ottenuto graficando le curve relative a FAR e FRR ponendo le soglie utilizzate sull'asse x e l'*error rate* sull'asse y: l'EER corrisonde all'*error rate* nel punto di incontro delle due curve. La soglia corrispondente all'EER è solitamente la più

conveniente poiché è il punto in i due *error rate* raggiungono complessivamente il loro minimo.

- *ZeroFAR*: considerando lo stesso grafico illustrato per il calcolo dell'EER, è definito come il valore del FRR nel punto in cui il FAR tocca lo 0. In applicazioni in cui la sicurezza è importante è ovviamente cruciale che questo valore sia il più basso possibile (non a caso spesso viene calcolato solo esso e non il corrispondente *ZeroFRR*), poiché sta ad indicare la probabilità di *False Rejection* che si dovrà accettare se si vuole che il sistema non accetti mai impostori. In realtà, spesso si tende ad essere meno rigidi e calcolare il valore del FRR in corrispondenza di valori di FAR molto bassi, ma non esattamente pari a 0.

  L'immagine seguente[20] dà una visione complessiva di FAR, FRR, EER, *ZeroFar* e *ZeroFRR*:

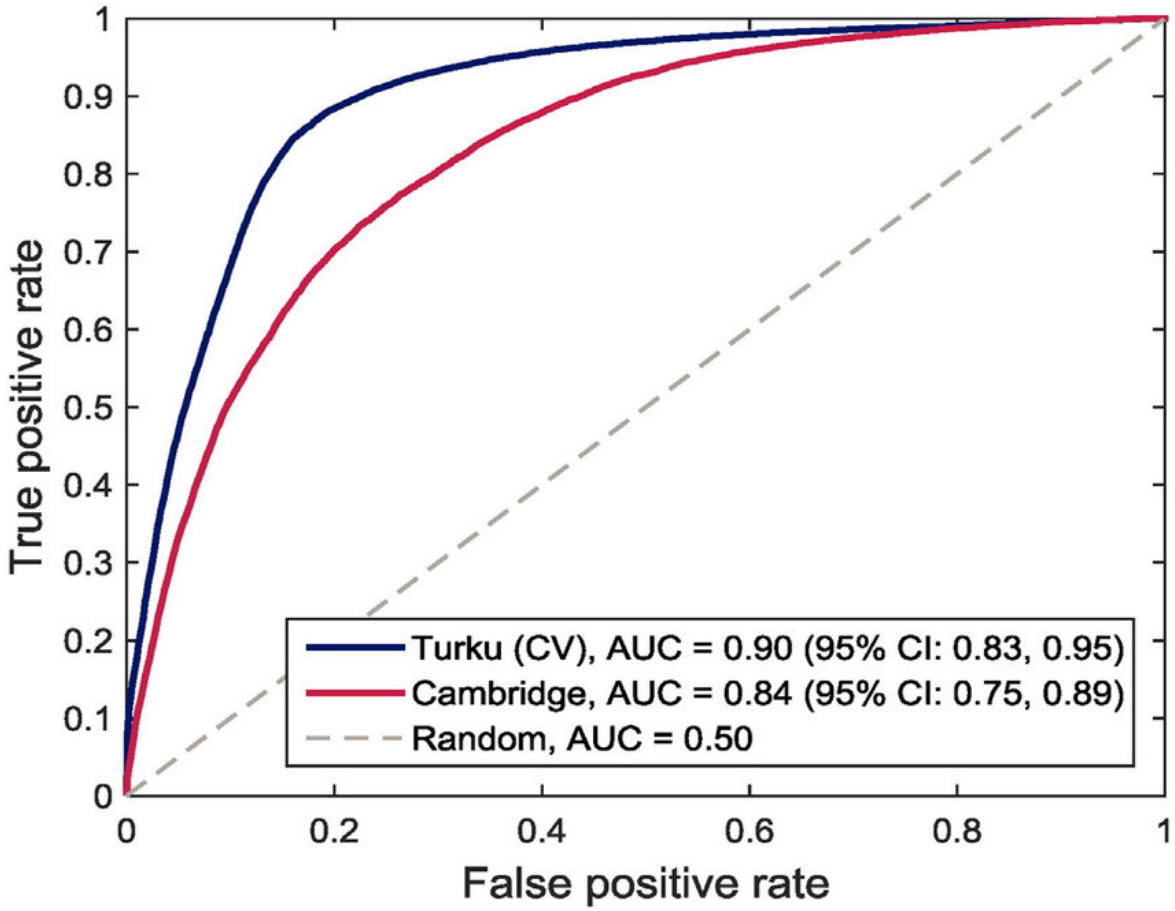
[20] Fonte: http://www.biometrika.it/wp_biointro.html



- ROC – *Receiver Operating Characteristic*: è una curva che confronta i FAR (posti sull'asse x) con i GAR (posti sull'asse y).
- ROC AUC – *Area Under Curve*: definito come l'integrale (area sottostante) della curva ROC. Maggiore è questo valore e migliore sarà il sistema, in quanto dà indicazioni sulla probabilità di *Genuine Acceptance* rispetto a quella di *False Acceptance*.

  Viene riportato un esempio[21] di due diverse curve ROC per capirne l'interpretazione corretta:

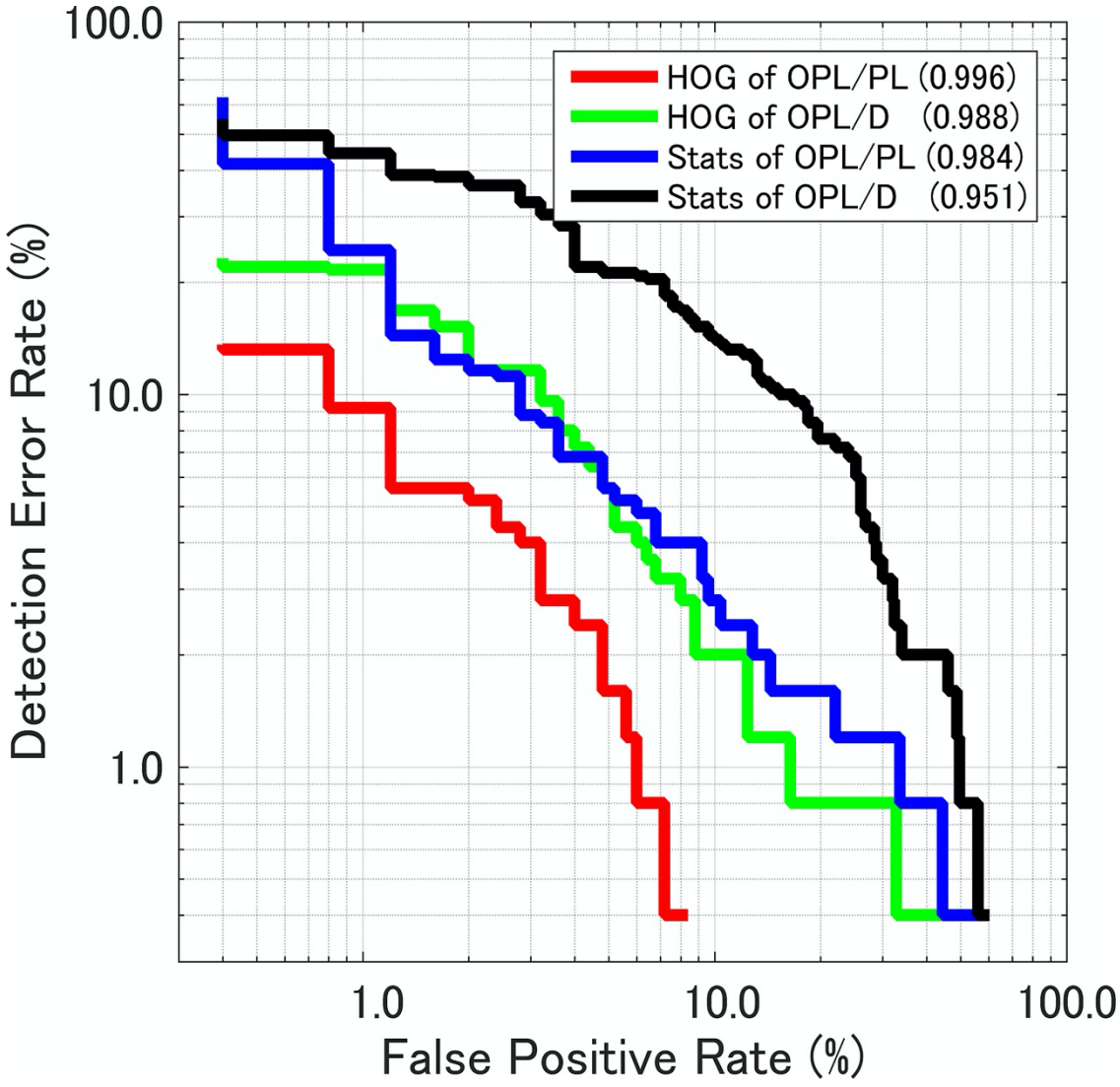
[21] Fonte: (Oresic *et al,* 2016)



Si nota subito come la curva che può essere considerata migliore è quella riportata in blu, poiché delinea una probabilità di *Genuine Acceptance* rispetto a quella di *False acceptance* più elevata, ed infatti presenta un'area sottostante AUC maggiore.

- DET – *Detection Error Tradeoff*: è una curva che confronta i FAR (posti sull'asse x) con i FRR (posti sull'asse y). Spesso viene rappresentata graficamente in forma logaritmica.

Anche in questo caso riportiamo un confronto di diverse curve DET:



Qui la curva che esprime le performance migliori è quella riportata in rosso, poiché rispetto alle altre mostra valori degli *error rate* complessivamente meno elevati.

2.h Valutazione del sistema

La fase di valutazione ha inizio scegliendo una serie di soglie all'interno dell'intervallo dei possibili valori di somiglianza, in questo caso [0,1], con un campionamento di 0.01 (quindi le soglie utilizzate sono in tutto 100).
Per ogni soglia vengono quindi prodotte le predizioni relative alle camminate, andando a confrontare tutte le probabilità predette in precedenza con la soglia: se la probabilità è maggiore, la camminata viene considerata verificata, altrimenti rifiutata.
Una volta ottenute le predizioni, esse vengono confrontate con le etichette di classe delle camminate corrispondenti, che erano state precedentemente binarizzate come visto; si possono presentare quattro casi:

- *Genuine acceptance* (GA): la camminata è verificata e l'etichetta è pari a 1
- *False acceptance* (FA): la camminata è verificata e l'etichetta è pari a 0
- *Genuine rejection* (GR): la camminata è rifiutata e l'etichetta è pari a 0
- *False rejection* (FR): la camminata è rifiutata e l'etichetta è pari a 1

Andando a contare, per ogni soglia t, il numero di occorrenze di questi quattro casi, si ottengono i relativi rate illustrati in precedenza, tramite le formule:

$$FAR(t) = \frac{FA(t)}{FA(t) + GR(t)}$$

$$FRR(t) = \frac{FR(t)}{FR(t) + GA(t)}$$

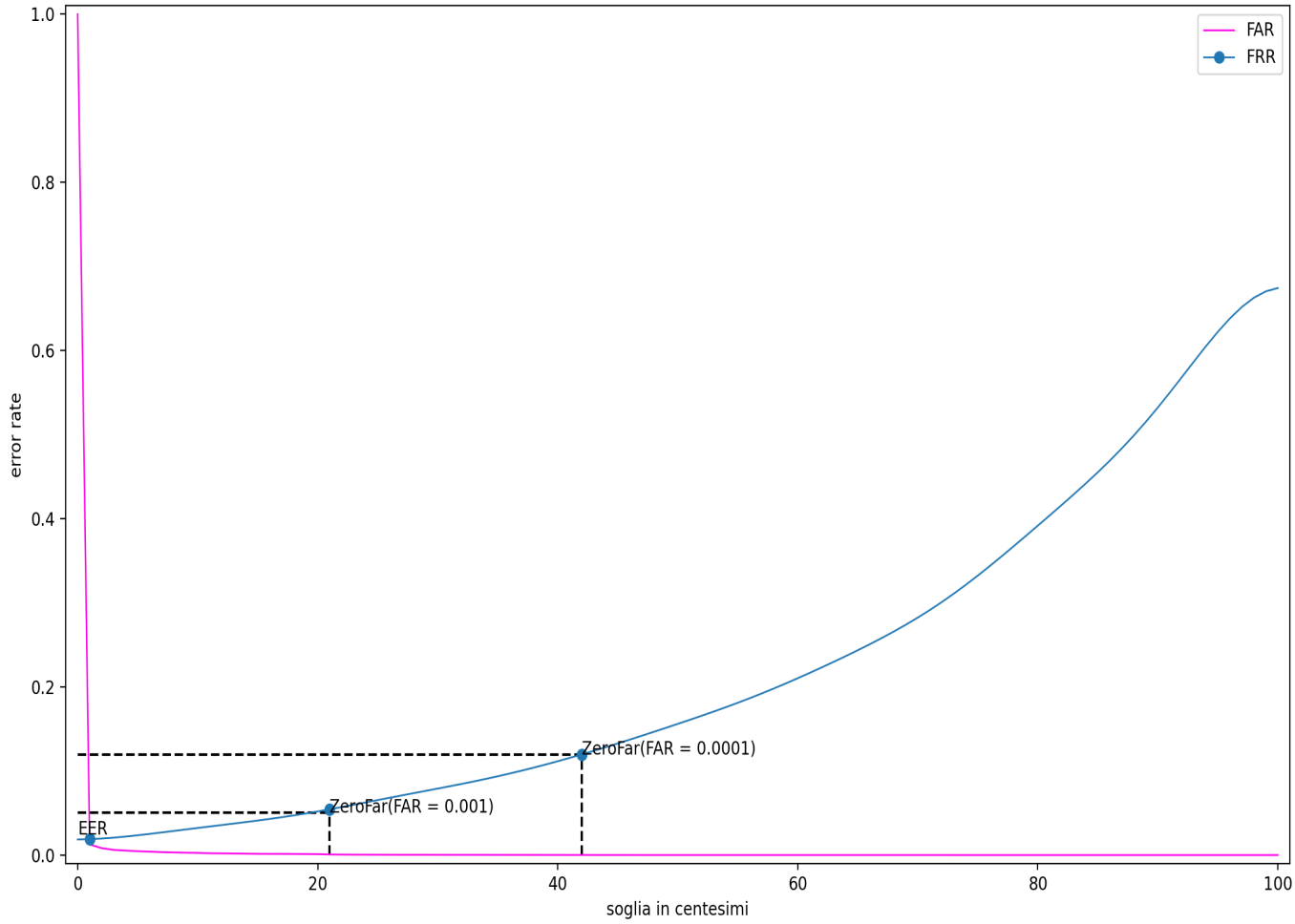
$$GAR(t) = 1 - FRR(t)$$

$$HTER(t) = \frac{FAR(t) + FRR(t)}{2}$$

A questo punto si può produrre un grafico dell'andamento di FAR e FRR al variare della soglia t, andando ad ordinare i FAR in maniera decrescente e i FRR in maniera crescente, poiché si ha a che fare con punteggi di similarità (se fossero state distanze, l'ordinamento corretto sarebbe stato il contrario).

Viene riportato il grafico ottenuto in questo modo: si può notare che la curva FAR decresce subito molto velocemente e rimane poi abbastanza stabile (decresce estremamente lentamente) per tutta la sua continuazione. Questo accade poiché le probabilità predette per la stragrande maggioranza degli impostori sono giustamente molto basse (minori di 0,01), quindi appena la soglia varia da 0 a 0,01 si verifica un'enorme diminuzione delle *False Acceptance* poiché tutti questi impostori non saranno più accettati.

Al contrario, la curva FRR cresce più gradualmente per tutta la sua lunghezza: questo fa si che le due curve si incontrino molto presto, quindi l'EER risultante sarà basso.



Si procede ora al calcolo dell'EER, rappresentato come si vede in figura dal valore della coordinata y nel punto di incontro delle due curve, facendo la

media dei due valori di FAR e FRR corrispondenti al punto (indice) dove i due vettori così ordinati si incrociano.

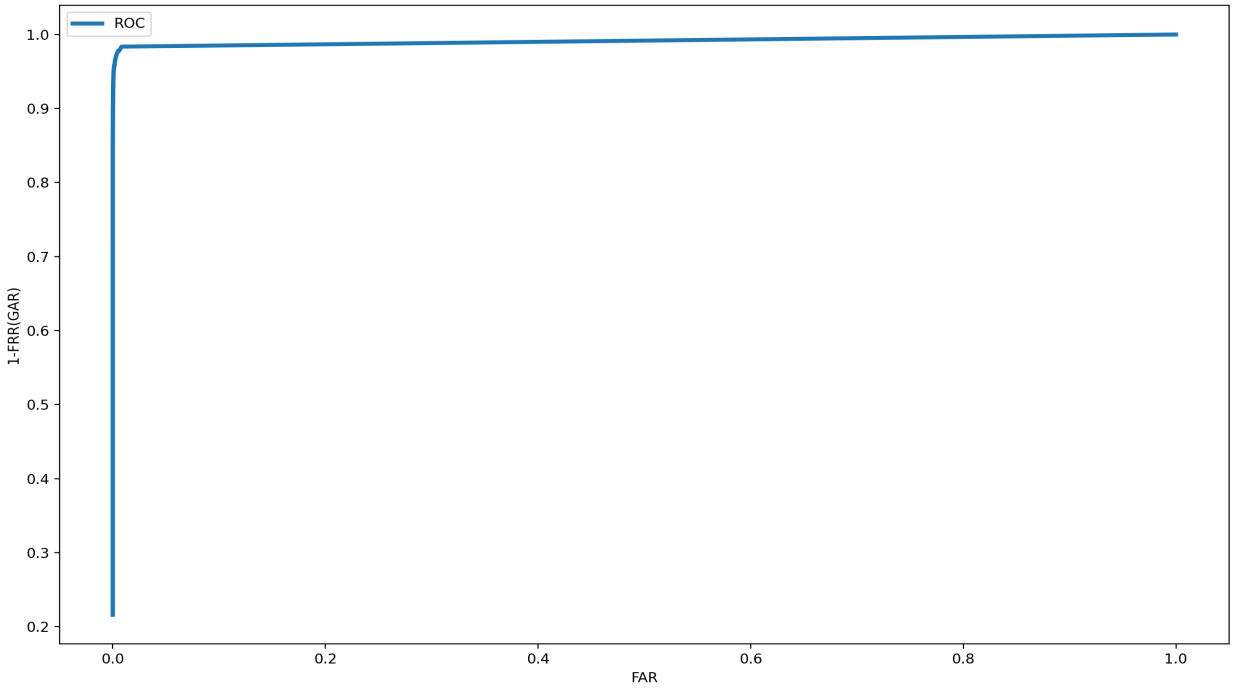
Tale valore è pari a 0.01447726621453816 (circa l'1,45%).

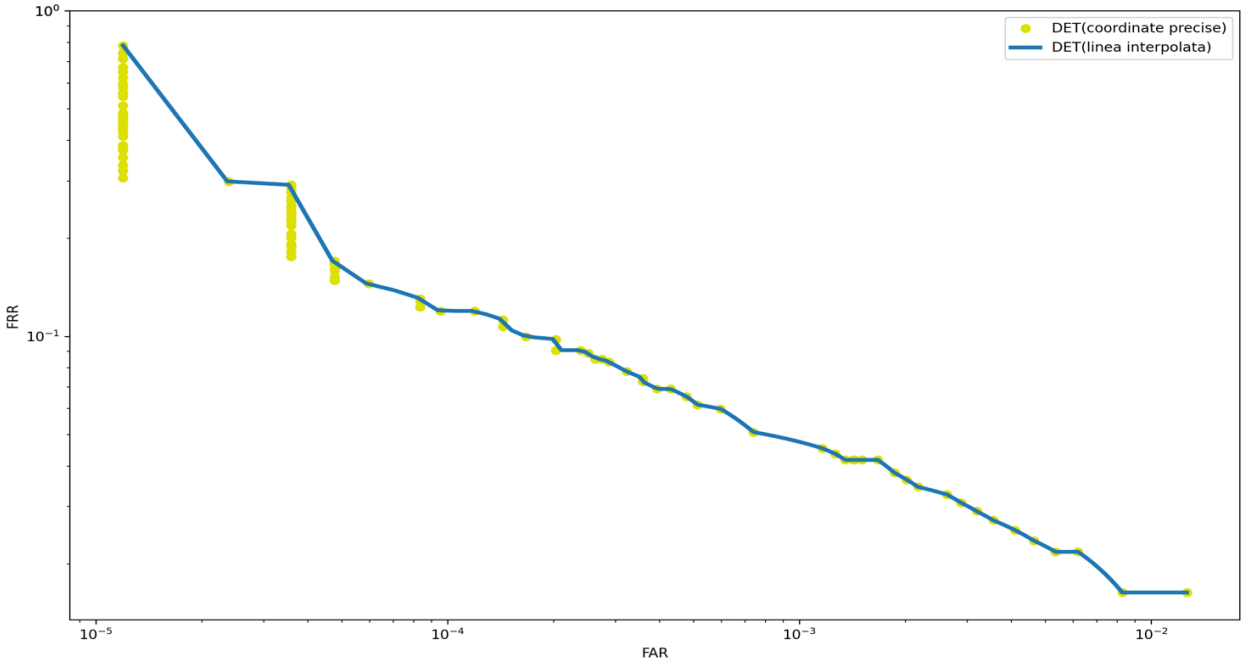
Si ottiene inoltre il valore *zeroFAR*, pari a quello del FRR nel punto in cui la curva FAR tocca lo 0.

In realtà, come si fa spesso e già anticipato, il valore è stato calcolato in corrispondenza di FAR molto bassi, pari a 0,001 e 0,0001: i valori ottenuti sono rispettivamente 0.0508167 e 0.11978221.

La procedura continua andando a calcolare e produrre grafici per le due curve ROC e DET, confrontando rispettivamente FAR con GAR e FAR con FRR. Per la curva ROC viene anche calcolato il punteggio AUC integrando la curva ottenuta (con il metodo dei trapezi). Esso è pari a 0.9915839799643146.

I grafici ottenuti sono i seguenti:


ROC
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
1-FRR(GAR)
0.0
0.2
0.4
0.6
0.8
1.0
FAR


DET(coordinate precise)
DET(linea interpolata)
10^0
10^{-1}
FRR
10^{-5}
10^{-4}
10^{-3}
10^{-2}
FAR

Quest'ultima curva (DET) è un'approssimazione ottenuta a partire dai valori reali delle coppie (FAR, FRR), mostrati per mezzo dei punti in giallo, per far si che avesse un andamento non esageratamente "spigoloso".

L'ultima fase della valutazione è di nuovo relativa all'analisi dell'usabilità del sistema in scenari reali: essa va infatti a studiare il costo computazionale dei vari componenti del sistema.
Viene innanzitutto calcolato il tempo necessario alla selezione delle caratteristiche rilevanti ed al conseguente allenamento dei modelli, operazioni che possono essere effettuate una volta per tutte senza doverle più ripetere in seguito. Il tempo impiegato per questo prima fase è di circa 3 ore e 43 minuti.
Successivamente viene invece calcolato il cosiddetto *throughput rate,* ovvero il tempo medio per una singola operazione di riconoscimento avendo i modelli già allenati.
Esso viene ottenuto dividendo il tempo medio impiegato per la predizione delle probabilità delle istanze dell'intero insieme di test per la sua cardinalità, ovvero il 30% dell'intero dataset, cioè 550.
In questo modo si ottiene un tempo medio per la verifica di un'istanza pari a circa $0{,}00\overline{36}$ secondi.
Si è registrato anche il tempo medio necessario al successivo confronto con la soglia, ma esso si è rivelato irrilevante in quanto su un ordine di

grandezza molto minore di quello del tempo impiegato per la predizione (10^{-5}).

Tutti i test sono stati eseguiti su un *macbook pro* del 2017 con processore *Intel Core i5 dual-core (2,3 GHz)* e 16 GB di RAM.

L'idea alla base della divisione illustrata è volta a massimizzare l'usabilità del sistema in un possibile scenario reale, ad esempio per la verifica dell'identità tramite dispositivi mobili: non sarebbe pensabile eseguire per ogni istanza di verifica entrambe le procedure su smartphone, quindi la prima dev'essere anticipata e svolta una volta per tutte a priori, in modo da avere poi i modelli pronti per la classificazione.

L'analisi delle tempistiche viene comodamente realizzata per mezzo della libreria *time* di Python, che consente di calcolare i tempi di esecuzione inserendo *breakpoint* all'interno del codice.

E' interessante a questo punto andare a fare un confronto delle performance ottenute con quelle del sistema illustrato in (De Marsico, Fartade, Mecca, 2018), che è stato il punto di partenza ed ha fornito svariati spunti per lo sviluppo della procedura di verifica dell'identità.

Il sistema descritto in (De Marsico, Fartade, Mecca, 2018) utilizza lo stesso dataset *Zju-Gaitacc* e si basa anch'esso sull'estrazione e selezione delle caratteristiche più rilevanti dalle serie temporali, usando tuttavia tecniche diverse di estrazione, selezione e confronto delle camminate (ad esempio,

dopo la selezione si mantengono in generale molte meno caratteristiche delle 400 mantenute in questo caso).

Esso è stato testato in quattro scenari differenti, di cui tuttavia il primo (chiamato T1) non è realistico poiché non attua alcuna selezione delle caratteristiche e serve unicamente ad ottenere performance di base con cui confrontare le altre; gli altri tre, invece, sono i seguenti (per ognuno, sono stati riportati gli EER ottenuti confrontando i vettori di caratteristiche utilizzando rispettivamente distanza Manhattan[22] ed Euclidea[23]: per T1 essi valgono 22.4% e 24.6%):

- T2: seleziona le caratteristiche che presentano varianza maggiore per tutti e 3 gli assi (in tutto 165). Gli EER ottenuti sono pari a 20,2% e 22,5%
- T3: similmente a T2 seleziona le caratteristiche con varianza maggiore, ma invece di mantenere le caratteristiche abbastanza informative (quindi con varianza elevata) per tutti gli assi, vengono mantenute quelle che lo sono per un sottoinsieme stretto di essi (1 o 2). Gli EER ottenuti sono pari a 30,6% e 31,5%
- T4: utilizza solamente la *Principal Feature Analysis* vista in precedenza, fino a mantenere un numero di circa 60 caratteristiche. Gli EER ottenuti sono pari a 18,7% e 19,6%

[22] https://xlinux.nist.gov/dads/HTML/manhattanDistance.html
[23] https://en.wikipedia.org/wiki/Euclidean_distance

Si può quindi affermare che le performance siano sensibilmente migliorate nel sistema in questione, probabilmente grazie all'utilizzo più massiccio della PFA (che gli autori avevano già notato avere effetti positivi) ed alla diversa tecnica di confronto delle camminate illustrata in precedenza.

3. Conclusioni

Lo scopo principale del lavoro è stato quello di proseguire la ricerca svolta in (De Marsico, Fartade, Mecca, 2018): essa aveva mostrato come un approccio basato sull'estrazione di caratteristiche dalle serie temporali potesse ridurre il costo computazionale delle operazioni di verifica dell'identità rispetto all'utilizzo di algoritmi di confronto dei segnali puri, aprendo la strada per una possibile esecuzione della procedura su dispositivi mobili.

Gli EER ottenuti, tuttavia, erano più elevati rispetto a quelli registrati per gli altri algoritmi citati: il sistema sviluppato riesce quindi a migliorare l'accuratezza della verifica, rendendola superiore a quella di entrambi gli approcci appena visti, senza rinunciare al costo computazionale contenuto della singola operazione di verifica. Questo è garantito anche dalla divisione della fase di selezione delle caratteristiche ed allenamento dei modelli da quella di verifica vera e propria.

Come tentativi per il futuro, è possibile studiare l'efficacia di altri tipi di algoritmi di machine learning, possibilmente con o senza *resampling* del dataset.

Riferimenti

Yuting Zhang, Gang Pan, Kui Jia, Minlong Lu, Yueming Wang, Zhaohui Wu (2015). "Accelerometer-based Gait Recognition by Sparse Representation of Signature Points with Clusters", IEEE Transactions on Cybernetics, vol. 45, no. 9, pp. 1864-1875.

De Marsico M., Fartade E., Mecca A. (2018). "Feature-based Analysis of Gait Signals for Biometric Recognition - Automatic Extraction and Selection of Features from Accelerometer Signals". In Proceedings of the 7th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2018), pp. 630-637.

Lu, Y., Cohen, I., Zhou, X. S., & Tian, Q. (2007). "Feature selection using principal feature analysis". In Proceedings of the 15th ACM international conference on Multimedia, pp. 301-304.

Hu, Feng & Li, Hang. (2013). "A Novel Boundary Oversampling Algorithm Based on Neighborhood Rough Set Model: NRSBoundary-SMOTE". Mathematical Problems in Engineering. 2013.

Hastie, T., Rosset, S., Zhu, J., & Zou, H. (2009). "Multi-class adaboost. Statistics and its Interface", 2(3), pp. 349-360.

Oresic, Matej & Posti, Jussi & Kamstrup-Nielsen, Maja & Takala, Riikka & Lingsma, Hester & Mattila, Ismo & Jäntti, Sirkku & Katila, Ari & Carpenter, Keri & Ala-Seppälä, Henna & Kyllönen, Anna & Maanpää, Henna-Riikka & Tallus, Jussi & Coles, Jonathan & Heino, Iiro & Frantzén, Janek & Hutchinson, Peter & Menon, David & Tenovuo, Olli & Hyotylainen, Tuulia. (2016). "Human Serum Metabolites Associate With Severity and Patient Outcomes in Traumatic Brain Injury". EBioMedicine. 12.