Biometric project report: BioPhone

Dario Aragona

Luca Podo

Sommario

Introduzione

Questo report propone un metodo per l'autenticazione ad uno smartphone tramite l'analisi di dati biometrici, diverso dai sistemi attualmente più utilizzati, come ad esempio impronta digitale e riconoscimento facciale.

Mentre queste due tecnologie prevedono l'utilizzo di sensori dedicati, la nostra idea consiste nell'utilizzo dei dati biometrici rilevati da sensori più comuni, presenti in tutti gli smartphone. L'obiettivo del nostro progetto è quello di offrire un sistema di sicurezza che non richieda sensori dedicati oltre a quelli già presenti, in modo da garantire un maggiore livello di sicurezza di quello offerto dal solo PIN, ma che contemporaneamente non incida sul costo del dispositivo stesso.

Un dispositivo la cui sicurezza è basata esclusivamente sul codice PIN (e sprovvisto di verifica dell'impronta digitale o riconoscimento facciale) risulta vulnerabile nel momento in cui il codice d'accesso venga scoperto.

Per questo, la nostra soluzione prevede l'estrazione e l'analisi di dati biometrici che descrivono il comportamento dell'utente durante la fase di digitazione del PIN per accedere al proprio dispositivo. Per fare questo abbiamo sfruttato principalmente due tipi di dati:

- Dati provenienti dall'accelerometro del dispositivo durante la digitazione.
- Dati riguardanti la velocità di inserimento del PIN da parte dell'utente.

I dati raccolti sono stati analizzati attraverso una tecnica di Machine Learning: *Novelty Detection*.

Il progetto propone lo sviluppo di un'applicazione Android che, sulla base di quanto descritto finora, determina se l'inserimento del codice d'accesso al dispositivo è opera dell'effettivo proprietario o meno, notificando la risposta a livello grafico.

Problema

Attualmente i dispositivi mobile hanno affiancato al classico PIN per accedere al sistema, anche livelli di sicurezza basati su dati biometrici.

Alcuni tra i più diffusi sono:

- Sblocco con impronta digitale, il quale sfrutta un sensore dedicato per valutare se l'impronta coincide con quella del proprietario. Attualmente l'evoluzione di questa tecnologia prevede la sostituzione del sensore con una fotocamera ad alta definizione che cattura l'immagine dell'impronta. I vantaggi di questa evoluzione sono una maggiore precisione dell'impronta digitale, ma introduce nuove complicazioni dovute all'analisi di immagini.

- Sblocco tramite riconoscimento facciale; questa tecnologia sfrutta la combinazione tra più sensori del dispositivo come telecamera ed infrarossi per riconoscere o meno l'utente, attraverso la sua faccia. Gli svantaggi di questa tecnologia sono il costo dei sensori (maggiori rispetto all'impronta digitale) e la sensibilità alle variazioni di PIE.

Non tutti i dispositivi attualmente utilizzati sono in possesso dei suddetti sensori, ed in caso lo fossero, spesso utilizzano quelli di bassa precisione in modo tale da mantenere il costo del dispositivo entro un certo limite.
Questo crea una disparità in termini di sicurezza tra dispositivi di fascia alta da quelli di bassa fascia.

La nostra idea consiste nello sfruttare un sensore basilare, l'accelerometro, presente in tutti i dispositivi, per cercare un'alternativa ai metodi sopra citati e rendere più sicuri anche i dispositivi meno all'avanguardia.

Scelte progettuali

Per venire incontro ai limiti tecnologici offerti dai dispositivi mobile abbiamo deciso di delegare ad un Server in cloud la fase ML di predizione; in questo modo il progetto è stato sviluppato tenendo conto di tre moduli principali:

1. Server
2. Machine Learning Model
3. Android Application

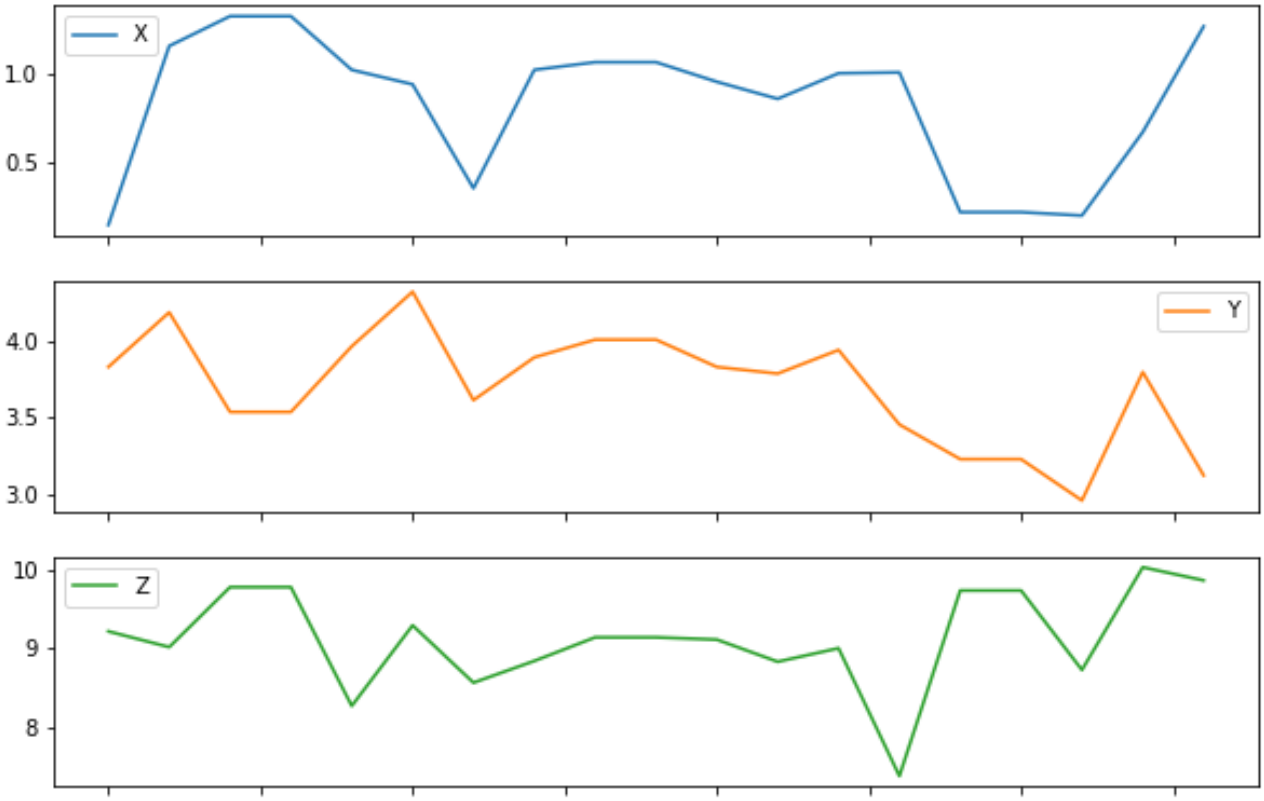
Un'importante considerazione che è stata fatta prima della definizione dei moduli precedenti è quella relativa alla struttura dei dati da analizzare: è stato necessario trovare un punto di incontro tra i dati generati dai sensori del dispositivo e le strutture dati offerte dal back-end.

La nostra scelta è quindi ricaduta sull'uso del formato JSON, in quanto soddisfa i requisiti richiesti dal lato client e server.
La struttura dati comprende due tipi di informazioni, ovvero i dati relativi all'accelerometro e le misurazioni degli intervalli di tempo tra la pressione di un tasto e il successivo.

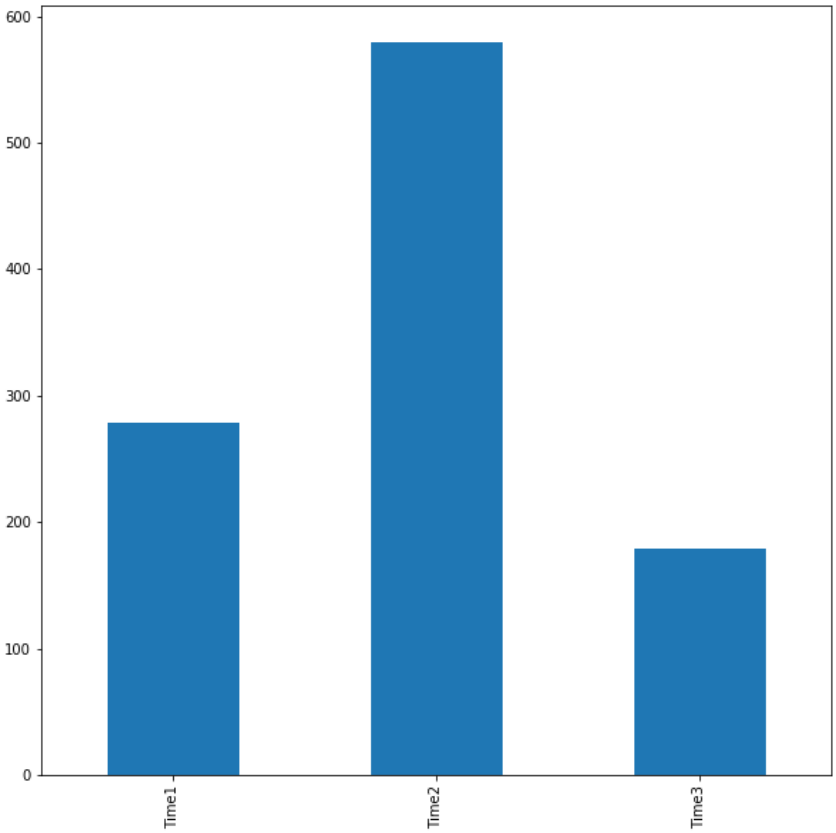
Di seguito è riportata la struttura dati che abbiamo definito:

- *Accelerometer data from X, Y and Z axis* sono i dati restituiti dall'accelerometro dello smartphone per gli assi x, y, z
- *Times series id* è il valore usato per identificare l'ordine temporale di ogni acquisizione in una sessione (sulla base della frequenza di acquisizione)
- *Session id* è usata per aggregare I dati appartenenti alla stessa sessione di inserimento
- *Time intervals* sono gli intervalli registrati tra l'inserimento di un numero e il suo successivo

Di seguito è riportata la visualizzazione dei dati ricevuti dal client, per l'accelerometro,



e per gli intervalli di tempo.



L’immagine che segue mostra un esempio di una sessione di scrittura del pin, cosi come registrata dal client e convertita in formato JSON, per essere poi inviata al server:


```
"prediction": [
  "X":0.5793967843055725 , "Y":  5.037400245666504 , "Z": 7.915230751037598 , "Timestamp":   0  ,  "Sessione" :   1  ,  "Time1":   300 ,  "Time2":   316 ,  "Time3":
  279 },
  "X":-0.4836287498474121, "Y":  4.7405195236206055 , "Z": 8.959102630615234 , "Timestamp":  2  ,  "Sessione" :   1  ,  "Time1":   300 ,  "Time2":   316 ,  "Time3":
  279 },
  "X":-0.4836287498474121, "Y":  4.7405195236206055 , "Z": 8.959102630615234 , "Timestamp":  3  ,  "Sessione" :   1  ,  "Time1":   300 ,  "Time2":   316 ,  "Time3":
  279 },
  "X":-0.5937620401382446, "Y":  4.381389141082764 , "Z": 9.495404243469238 , "Timestamp":   5  ,  "Sessione" :   1  ,  "Time1":   300 ,  "Time2":   316 ,  "Time3":
  279 },
  "X":0.11013327538967133, "Y":  3.825934410095215 , "Z": 9.457097053527832 , "Timestamp":   6  ,  "Sessione" :   1  ,  "Time1":   300 ,  "Time2":   316 ,  "Time3":
  279 },
  "X":0.751779317855835  , "Y": 3.9217023849487305 , "Z": 8.7484130859375 , "Timestamp":  7  ,  "Sessione" :   1  ,  "Time1":   300 ,  "Time2":   316 ,  "Time3":
  279 },
  "X":-0.0957680642604827, "Y": 4.716577529907227 , "Z": 7.876923561096191 , "Timestamp": 1  ,  "Sessione" :   1  ,  "Time1":   300 ,  "Time2":   316 ,  "Time3":
  279 },
  "X":2.078166961669922, "Y": 4.122815132141113 , "Z": 7.359776020050049 , "Timestamp":  14  ,  "Sessione" :   1  ,  "Time1":   300 ,  "Time2":   316 ,  "Time3":
  279 },
  "X":-1.0007762908935547, "Y": 4.036623954772949 , "Z": 9.346963882446289 , "Timestamp": 4  ,  "Sessione" :   1  ,  "Time1":   300 ,  "Time2":   316 ,  "Time3":
  279 },
  "X":1.1348515748977661, "Y": 3.8786067962646484 , "Z": 9.21288776397705 , "Timestamp":  13  ,  "Sessione" :   1  ,  "Time1":   300 ,  "Time2":   316 ,  "Time3":
  279 },
  "X":0.15322890877723694, "Y": 4.055777549743652 , "Z": 9.150638580322266 , "Timestamp": 11  ,  "Sessione" :   1  ,  "Time1":   300 ,  "Time2":   316 ,  "Time3":
  279 },
  "X":0.8523358106613159, "Y": 3.7397429943084717 , "Z": 9.222464561462402 , "Timestamp": 8  ,  "Sessione" :   1  ,  "Time1":   300 ,  "Time2":   316 ,  "Time3":
  279 },
  "X":0.19632454216480255, "Y": 3.8594532012939453 , "Z": 9.299078941345215 , "Timestamp":   9  ,  "Sessione" :   1  ,  "Time1":   300 ,  "Time2":   316 ,  "Time3":
  279 },
  "X":0.19632454216480255, "Y": 3.8594532012939453 , "Z": 9.299078941345215 , "Timestamp":   10  ,  "Sessione" :   1  ,  "Time1":   300 ,  "Time2":   316 ,  "Time3":
  279 },
  "X":2.078166961669922, "Y": 4.122815132141113 , "Z": 7.359776020050049 , "Timestamp":  15  ,  "Sessione" :   1  ,  "Time1":   300 ,  "Time2":   316 ,  "Time3":
  279 }

 ]
}
```

1. Server

Il server rappresenta il modulo delegato alla ricezione e gestione dei dati da parte dell'applicazione.

Per l'hosting e la gestione del nostro script del *back-end* abbiamo deciso di utilizzare la piattaforma *Heroku*.

Questa piattaforma, un *cloud platform as a service*, supporta vari linguaggi di programmazione tra cui Python; il vantaggio offerto da *Heroku* consiste nella possibilità di poter rilasciare rapidamente un'applicazione senza dover configurare tutta l'infrastruttura *server in cloud* normalmente necessaria.

Questo ci ha aiutato nello sviluppo del *back-end* senza doverci soffermare eccessivamente sulla configurazione *in cloud* non di pertinenza del progetto.

Per lo scripting abbiamo utilizzato il linguaggio di programmazione Python: questo linguaggio permette di avere una forte integrazione con la gestione e l'analisi dei dati usati in Machine Learning.

Python mette a disposizione, tramite diversi packages, molte funzionalità di interazione con un modello di Machine Learning ed inoltre permette di estendere il codice con packages terzi.

Lo script, ospitato su Heroku, si occupa della:

- **Ricezione in ingresso di richieste di analisi dati tramite POST**:
  Questa operazione è stata gestita attraverso la libreria Flask, tramite la quale abbiamo creato una route */api/* sul server per poter ricevere le richieste in POST fatte dal client. Di seguito è riportato come è stata strutturata la route:

```
@app.route('/api/', methods=['POST'])
@cross_origin(origin='*')
def post(): #definiamo una chiamata come post
```

- **Analisi e gestione dei dati ricevuti:**
  Per la gestione dei dati inviati dal client abbiamo deciso di utilizzare il formato JSON, che ha facilitato la creazione di un formato dati compatibile sia con il client che con il server, così da poter essere processato dallo script Python come un normale DataFrame, struttura dati messa a disposizione dalla libreria Python Pandas.
  In questa fase, inoltre, i dati sono soggetti ad una fase di pre-processing che

sarà descritta nella sezione riguardante il modello ML. Di seguito è riportato il codice per la gestione dei dati ricevuti con la richiesta:

```
@app.route('/api/', methods=['POST'])
@cross_origin(origin='*')
def post(): #definiamo una chiamata come post
    json_data = request.json # lettura del json in arrivo da
    if not json_data:
            return {'message': 'No input data provided'}, 400
    print(json_data)
    df = pd.DataFrame(json_data["prediction"])
```

- **Gestione del modello ML:**
  I dati raccolti, convertiti in DataFrame, sono presi come input dal modello ML che li confronta con i dati dell'utente proprietario; in base alla distanza tra di essi, il modello deciderà se l'utente è il proprietario o meno.

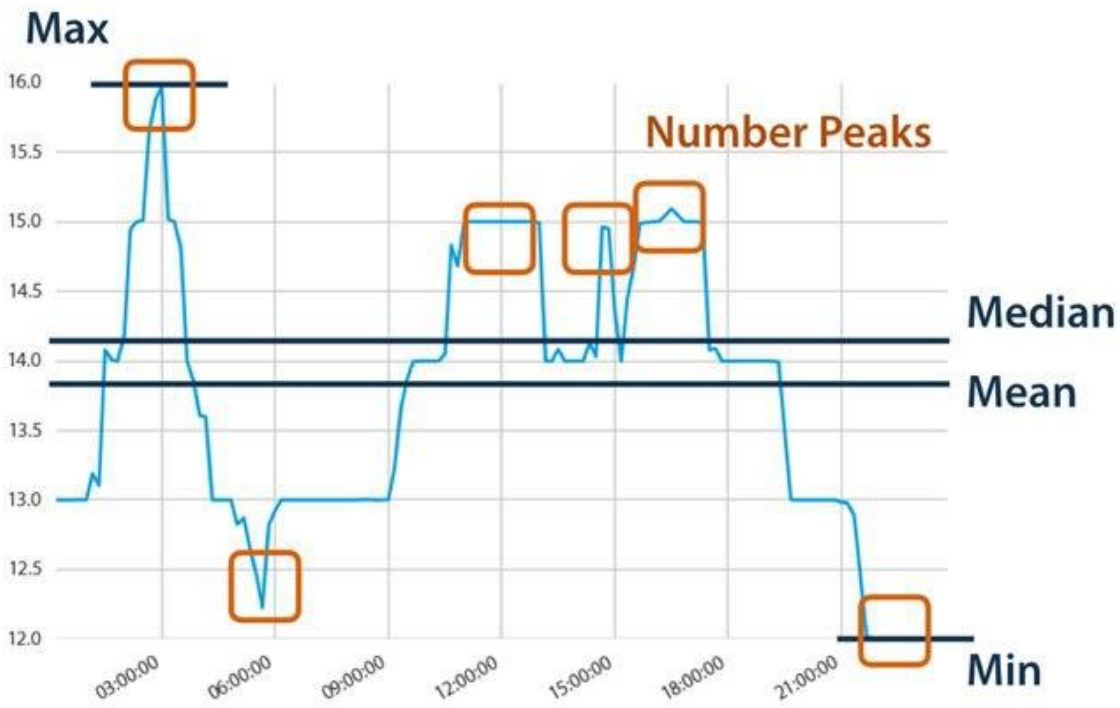
- **Gestione dell'invio dell'esito da parte del modello ML:**
  Nella stessa sessione della richiesta POST, il risultato restituito dal modello ML viene inviato al client.

2. Machine Learning Model

In questa sezione discuteremo di tutti gli aspetti legati alla fase di Machine Learning, che prevede:

- **Pre-processing dei dati:**
  Una volta effettuata la conversione da JSON a DataFrame è necessario ordinare cronologicamente i dati ricevuti dal sensore del client, in quanto potrebbero non essere in ordine. In seguito, il DataFrame è diviso in due parti che rappresentano le due diverse tipologie di dati (accelerometro e time intervals): questa divisione è necessaria perché la fase di feature extraction sarà diversa per i due tipi di dati.

- **Feature extraction:**
  I dati pre-processati relativi all'accelerometro sono definiti come Time Series, e necessitano l'estrazione di alcuni valori che caratterizzano l'andamento temporale del segnale. Per effettuare questa operazione abbiamo utilizzato le funzioni del package *Tsfresh*, che permette di calcolare più di un centinaio di diverse features a partire da una singola Time Series. Di seguito è riportato un esempio di quali valori vengono estratti.



  I dati restituiti da Tsfresh subiscono un ulteriore processing tramite la funzione *impute*, che si occupa di sostituire i valori *nan* restituiti dall'operazione di estrazione delle features.

```
def extract_features_(raw_data):
    data = raw_data.sort_values(by=['Timestamp'])
    data = data.reset_index(drop=True)
    data_accel = data[["X","Y","Z","Sessione"]]
    data_time = data[["Time1","Time2","Time3","Sessione"]]
    data_time = data_time.groupby(['Sessione']).mean()
    #tsfresh function
    extracted_accel_features = extract_features(data_accel, column_id="Sessione")
    impute(extracted_accel_features)
    finaldata = pd.concat([extracted_accel_features, data_time], axis=1)
    return finaldata
```

- **Scelta del modello:**
  Nel contesto in cui il nostro progetto lavora, l'obiettivo è quello di identificare

samples che differiscono significativamente dalla maggior parte dei samples registrati: questa operazione può essere definita Anomaly Detection.
Esistono tre diverse tipologie di Anomaly Detection:

1. Unsupervised anomaly detection individua anomalie in un dataset senza annotazioni cercando le istanze che differiscono dalla maggioranza dei dati nel dataset.
2. Supervised anomaly detection, dove i dati sono annotati come 'normali' o 'anomalie'.
3. Semi-supervised anomaly detection.

Il modello su cui è ricaduta la nostra scelta è il ***LocalOutlierFactor*** della libreria *sklearn*, che misura la variazione della densità di un sample rispetto ai suoi vicini: la densità dipende da quanto il sample è isolato rispetto ai suoi vicini. Più precisamente, la densità è definita dai *k-nearest neighbors*, usati per stimare la densità di un sample. Confrontando la densità di un sample con le densità dei suoi vicini, si possono identificare quelli a cui corrispondono dei valori sostanzialmente inferiori: questi sono considerati *outliers*.

Durante la fase di scelta del modello è stato anche valutato l'*SVMOneClass*, un unsupervised anomaly detection di *sklearn,* ma che è stato scartato a causa delle non soddisfacenti performances e della minor pertinenza con il nostro progetto durante una fase preliminare di studio dei modelli.

3. Android Application

Per lo sviluppo del front-end la nostra scelta è ricaduta sull'utilizzo di un framework per la progettazione di app ibride. Questi tipi di framework permettono di poter scrivere un'applicazione compatibile sia con Android, sia con IOS scrivendo il codice una sola volta.

Il framework che abbiamo utilizzato si chiama Ionic, il quale sfrutta principalmente due diverse tecnologie:

- Linguaggi web per la stesura del codice
- Cordova per l'interazione con il dispositivo nativo su cui si vuole far girare l'app

Quando si scrive una app con l'utilizzo di Ionic, questa sfrutta una webView messa a disposizione da Cordova per eseguire le operazioni definite. Cordova attraverso l'utilizzo plugin, permette di poter interagire con elementi nativi del dispositivo.

Il vantaggio principale di questi framework è la possibilità di poter astrarre dalla piattaforma su cui si vuole esportare, senza dover conoscere in maniera approfondita quest'ultima, a meno che non si vogliano definire delle estensioni custom per poter eseguire delle operazioni particolari. Inoltre, permettono anche di poter prototipare in maniera veloce.

Per lo sviluppo del nostro progetto, abbiamo utilizzato questo framework per sviluppare il client.

Le funzioni principali di questo modulo sono:

- Acquisizione dei dati durante una sessione di inserimento del pin (accelerometro e intervalli di tempo)
- Invio dei dati al server
- Fase decisionale una volta ricevuta la risposta dal server

Architettura di funzionamento

Il nostro progetto si compone due blocchi di funzionamento:

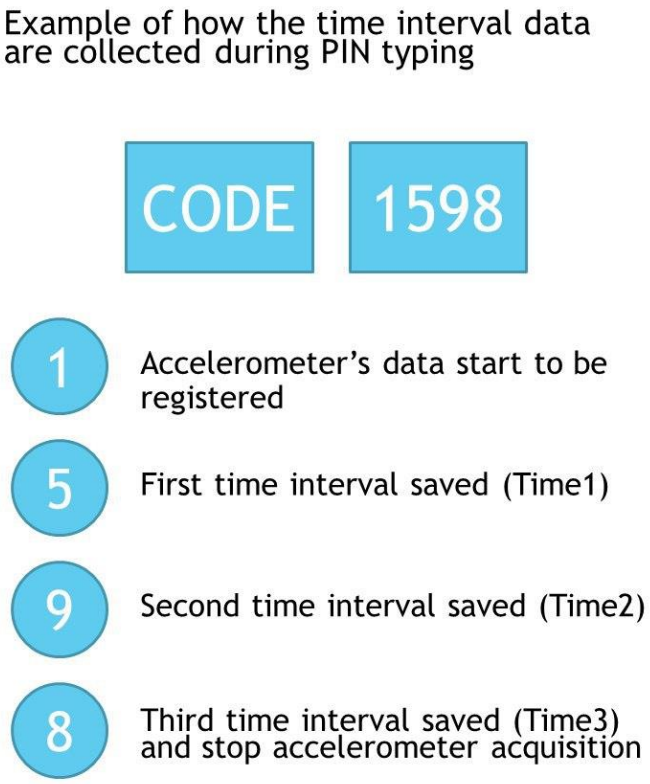
1. Blocco di acquisizione dei dati
2. Blocco di interazione con il server e il modello ML

Per il primo blocco, l'entità fondamentale è l'applicazione. In questa fase tutte le operazioni sono eseguite offline e non necessitano di una connessione ad Internet.

L'applicazione propone un tastierino numerico, dove l'utente può inserire il pin di 4 cifre. Non appena l'utente inizia a inserire il pin, il sistema inizia a rilevare i dati dall'accelerometro con una frequenza di un dato registrato ogni 50 millisecondi. Questi dati vengono, per ogni acquisizione, salvati all'interno della struttura dati predisposta per poi essere passati al server. Alla pressione del secondo tasto, il sistema rileva anche l'intervallo trascorso tra un tasto e il suo precedente, andando così a salvare le informazioni relative la velocità di typing dell'utente.

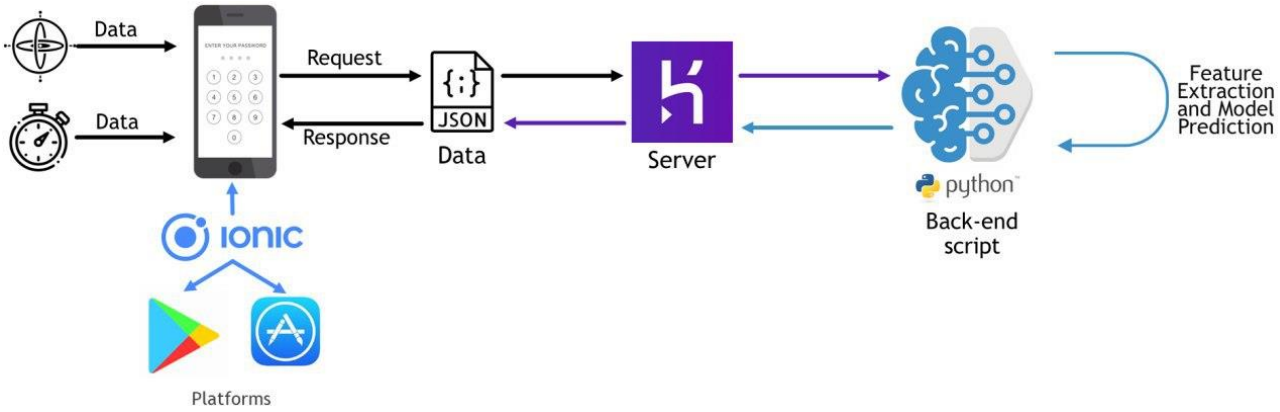
Le operazioni di acquisizione terminano con la pressione del quarto numero del pin, che abilita il sistema a mandare tutti i dati al server per il processing.

Di seguito è riportato l'esempio di come vengono acquisiti i dati durante una sessione:



Il secondo blocco si occupa invece dell'analisi e processing dei dati. Una volta che il client invia i dati al server in formato JSON, tramite richiesta POST, il server elabora i dati come definito nei blocchi precedenti, facendo il pre-processing dei dati e la relativa features extraction.

Finita questa fase, i dati vengono elaborati dal modello che cerca di determinare se risultano come anomalia o meno rispetto ai dati registrati. Una volta ottenuto il risultato, viene inviato al client che decise se abilitare o meno l'utente mostrando un messaggio di conferma o errore.



Valutazione delle prestazioni

Nel nostro caso, la fase di *enrollment* consiste nell'acquisizione dei dati del proprietario del telefono che digita il PIN (abbiamo utilizzato un pin standard 1598).

Il dataset usato nella fase di valutazione è composto dai dati di 30 sessioni di inserimento PIN da parte dell'utente proprietario del dispositivo (genuine) e i dati di un centinaio di sessioni di inserimento da parte di altri individui (impostors).

La fase di training del *LocalOutlierFactor* model necessita esclusivamente dei dati genuine, in modo che sia in grado di riconoscere quando l'inserimento PIN effettuato da un'altra persona si discosti dai dati del proprietario effettivo su cui ha eseguito il training.

Dopo aver effettuato il training avviene la fase di test, dove vengono passati al modello ML sia dati genuine che impostor, e per ciascun dato viene controllata la correttezza della decisione confrontando il risultato della *prediction* con il *groundtruth*.

Per ciascun sample, il *LocalOutlierFactor* restituisce un valore che rappresenta la distanza tra il sample e la distribuzione dei dati genuine: più questa è piccola, più ci sono probabilità che il sample sia genuine.

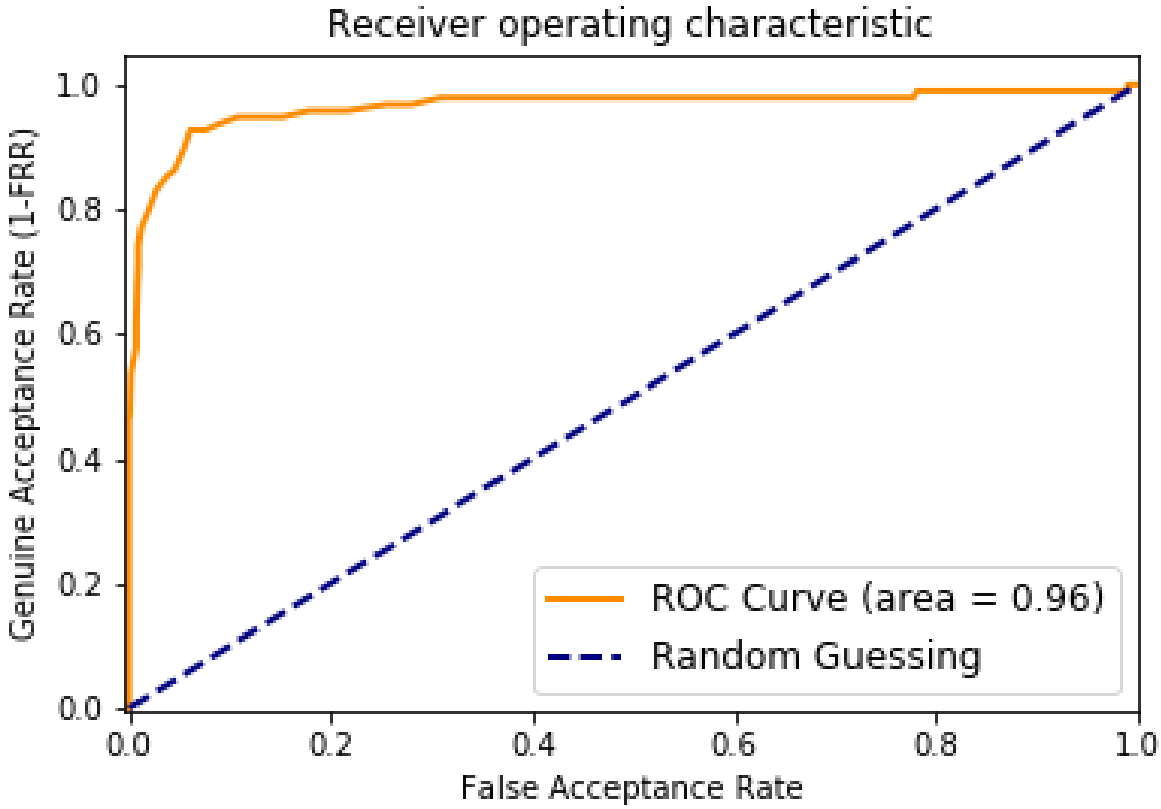
La fase di valutazione delle prestazioni è quindi basata su singole analisi usando vari *distance thresholds*, che rappresentano la distanza limite per cui un sample appartiene ad un utente genuine oppure no.

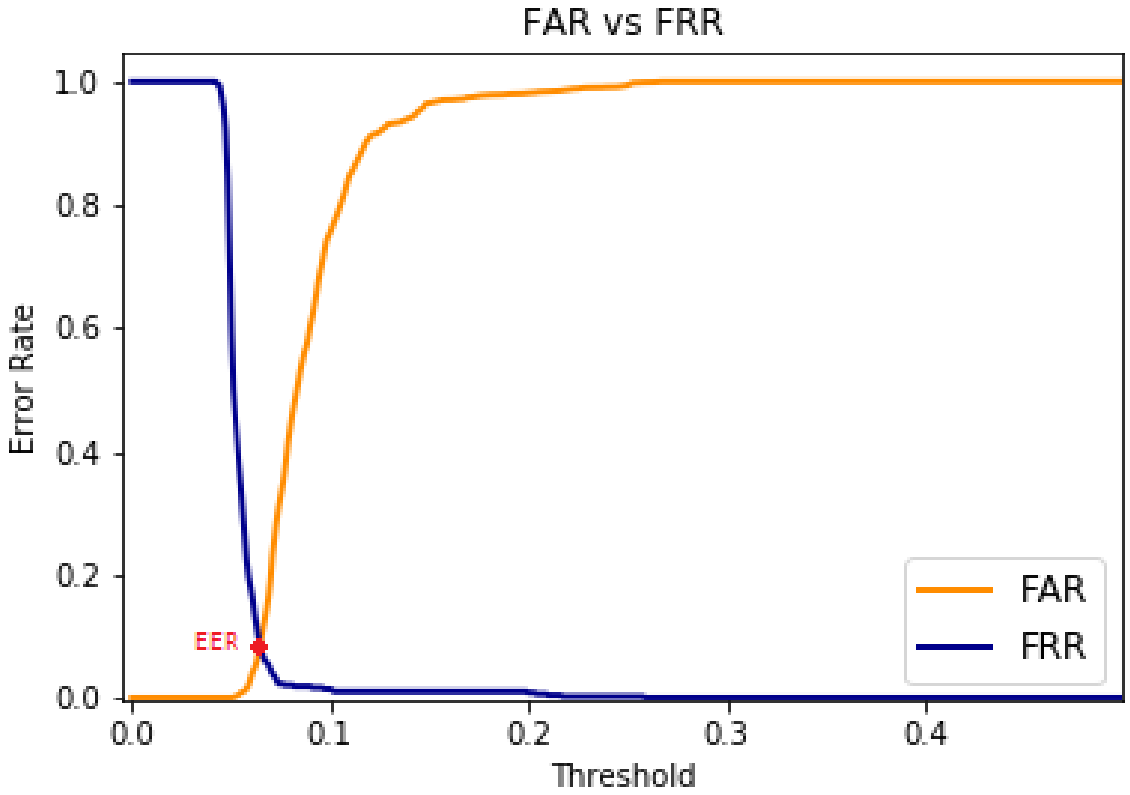
Per generalizzare la valutazione delle prestazioni abbiamo usato la tecnica di *K-Fold Cross Validation* (facente parte della libreria *sklearn*), che divide il dataset in K sottoinsiemi e ne utilizza uno come test set e gli altri K-1 come training set (il training set deve contenere solo dati genuine).

Dopo aver ottenuto i valori di FAR, FRR e GAR per ciascuna suddivisione del dataset, di questi viene calcolata la media rispetto a ciascun threshold testato in modo da ricavare il miglior valore discriminante, ovvero quello che corrisponde all'EER.

```python
def get_predictions(distance_scores, threshold):
    return list(map(lambda x: -1 if x > threshold else 1, distance_scores))
```

```python
def train_test_model(genuine_data, impostor_data, thresholds):

    y_trueP=[]
    for i in range(0, genuine_data.shape[0]):
        yP.append(1)

    y_trueN = []
    for i in range (0, impostor_data.shape[0]):
        yN.append(-1)

    y_true = np.concatenate((y_trueP, y_trueN), axis=0) #groundtruth

    kf = RepeatedKFold(n_splits=5, n_repeats=5, random_state=None)

    lof = LocalOutlierFactor(contamination=0.01, novelty=True, n_neighbors = 2)

    #LocalOutlierFactor(algorithm='auto', contamination='auto', leaf_size=30,
    #                   metric='minkowski', metric_params=None, n_jobs=None,
    #

    fpr_kfold = []
    tpr_kfold = []
    fnr_kfold = [] #values for each split

    for train_index, test_index in kf.split(genuine_data):
        X_train, X_test = genuine_data.iloc[train_index], genuine_data.iloc[test_index]
        X_test = pd.concat([X_test, impostor_data], axis=0)
        y_train, y_test = np.asarray(y_trueP)[train_index], np.asarray(y_trueP)[test_index]
        y_test = np.concatenate((y_test, y_trueN), axis=0)
        lof.fit(X_train)
        fpr_threshold = []
        tpr_threshold = []
        fnr_threshold = []
        for ths in thresholds:
            distance_scores = -lof.score_samples(X_test) #dissimilarity scores
            normalized_scores = distance_scores / np.linalg.norm(distance_scores)
            t = get_predictions(normalized_scores, ths)
            tn, fp, fn, tp = confusion_matrix(y_test, t, labels=[-1,1]).ravel()
            fpr_threshold.append(fp / (fp + tn + .000001))
            tpr_threshold.append(tp / (tp + fn + .000001))
            fnr_threshold.append(fn / (tp + fn + .000001))
        fpr_kfold.append(fpr_threshold)
        tpr_kfold.append(tpr_threshold)
        fnr_kfold.append(fnr_threshold) #values for each threshold in a split
    fpr_kfold = list(map(list, zip(*fpr_kfold)))
    tpr_kfold = list(map(list, zip(*tpr_kfold)))
    fnr_kfold = list(map(list, zip(*fnr_kfold)))

    fpr = []
    tpr = []
    fnr = [] #mean values over the splits for each threshold

    for v in fpr_kfold:
        fpr.append(np.mean(v))
    for v in tpr_kfold:
        tpr.append(np.mean(v))
    for v in fnr_kfold:
        fnr.append(np.mean(v)) #compute mean values for each threshold in all the splits
```


Receiver operating characteristic
Genuine Acceptance Rate (1-FRR)
False Acceptance Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC Curve (area = 0.96)
Random Guessing


FAR vs FRR
Error Rate
Threshold
0.0
0.1
0.2
0.3
0.4
0.6
0.8
1.0
EER
FAR
FRR

Una volta trovato il punto di Equal Error Rate, usiamo il relativo threshold per calcolare le performance del nostro modello tramite la funzione di sklearn *classification_report.*

FOR THRESHOLD 0.064000 :

	precision	recall	f1-score	support
-1	0.98	0.83	0.89	103
1	0.84	0.98	0.90	97
accuracy			0.90	200
macro avg	0.91	0.90	0.90	200
weighted avg	0.91	0.90	0.90	200

Conclusioni e sviluppi futuri

Il progetto analizzato in questo report presenta una base di partenza per uno studio più approfondito, in quanto il contesto che abbiamo analizzato risulta essere solo uno dei molti esistenti.

Infatti, uno dei limiti principali di quanto proposto è che il diverso utilizzo dello smartphone in diversi contesti può variare il risultato. Parliamo per esempio dei seguenti casi:

- Inserimento del pin con due mani
- Inserimento del pin con la sola mano destra
- Inserimento del pin con la sola mano sinistra
- Inserimento del pin in contesti di movimento
- Inserimento del pin in diverse posizioni come seduto o in piedi
- Inserimento del pin con diverse posizioni dello smartphone come verticale o orizzontale

Tutti questi particolari contesti descritti possono far variare il risultato.

La nostra idea è stata quella di studiare il caso di inserimento con due mani, cercando di generare un modello che possa essere quanto più preciso possibile.

Per questo alcuni sviluppi futuri prevedono:

- Lo sviluppo di un sistema di calibrazione per l'inserimento con la mano destra, sinistra o entrambe
- Studiare come nuovi dati possono essere acquisiti da altri sensori sul dispositivo
- Cercare di far girare il modello sul telefono, così da rendere indipendente la soluzione progettata dalla connessione internet
- Studiare come l'inserimento del pin in movimento o stando fermi, può variare il risultato.

Uno studio interessante, in futuro, potrebbe comparare le soluzioni già esistenti con quella proposta al fine di offrire una comparazione competitiva tra le diverse tecnologie e su come reagiscono su diversi dispositivi di diversa fascia di mercato.

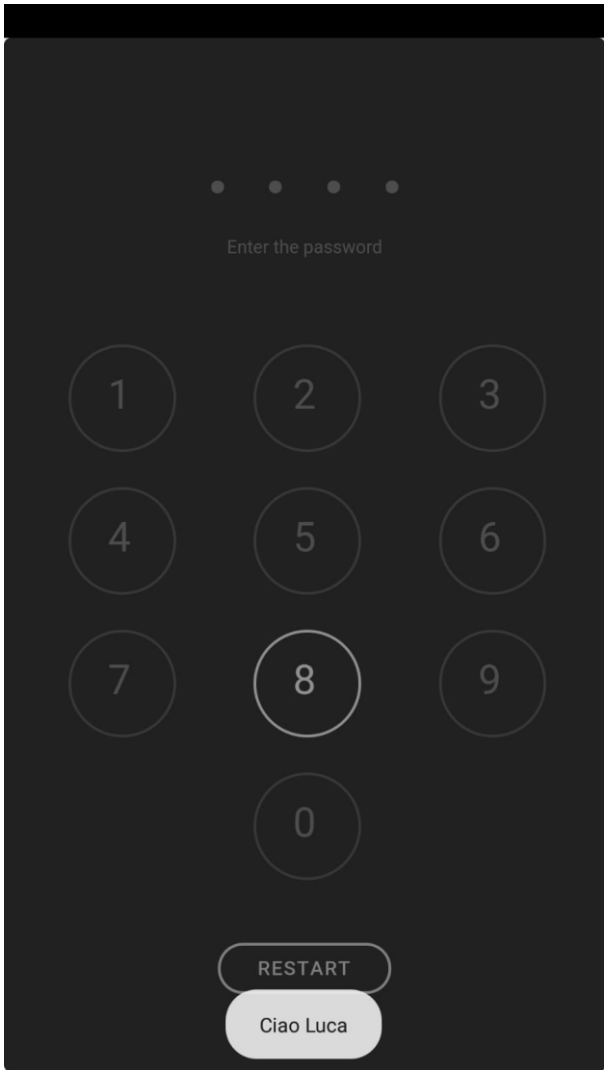
Il progetto proposto risulta essere in ogni sua parte sviluppato e pensato da noi, con tutti i suoi limiti e vantaggi, facendo affidamento solo a risorse esterne necessarie per poter eseguire le operazioni di base per il progetto.

In particolare, le risorse esterne utilizzate sono state:

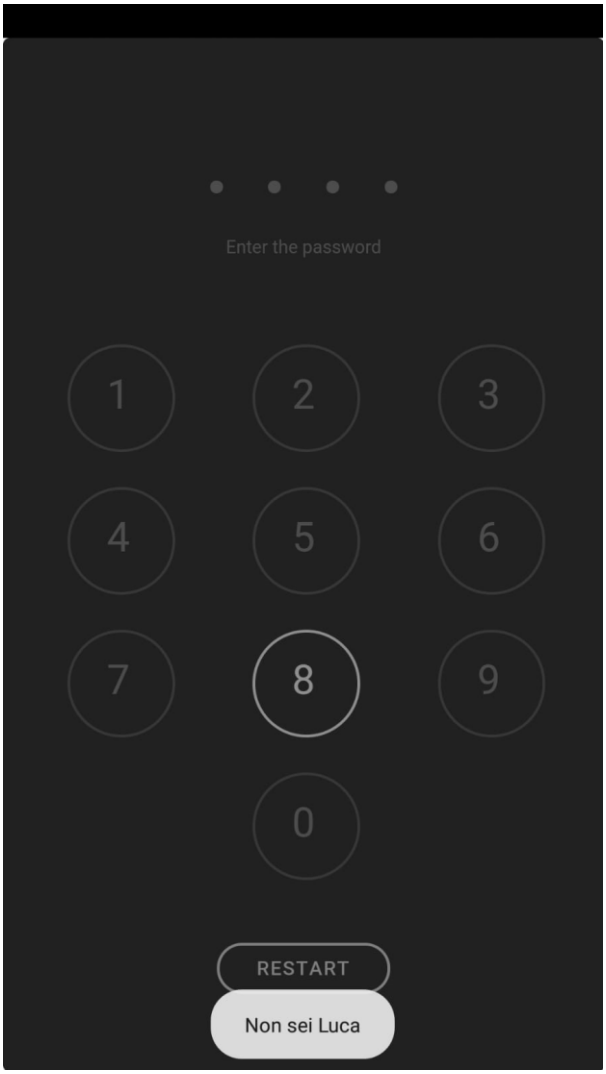
- Heroku cloud service
- Ionic per lo sviluppo dell'applicativo
- Librerie Python per gli script, la gestione dei dati e del modello:
    - Sklearn
    - Tsfresh

Screenshots

Genuine attempt



Impostor attempt



Codice Script Python back-end

```python
from flask import Flask, request, jsonify
from flask_restful import reqparse, abort, Api, Resource
import pickle
import numpy as np
from sklearn import svm
import requests
import json
from flask_cors import CORS, cross_origin
import pandas as pd
from tsfresh import extract_features
from tsfresh.utilities.dataframe_functions import impute

app = Flask(__name__)
api = Api(app) #import funzionalità per le rest

def get_predictions(dscores, threshold):
        return list(map(lambda x: -1 if x > threshold else 1, dscores))


from joblib import dump, load
clf = load('Model.joblib')

# class PredictSentiment(Resource):
@app.route('/api/', methods=['POST'])
@cross_origin(origin='*')
def post(): #definiamo una chiamata come post
    json_data = request.json # lettura del json in arrivo da passare al feature extraction
    if not json_data:
           return {'message': 'No input data provided'}, 400
    print(json_data)
    numpy_2d_arrays = np.array(json_data["prediction"])
    df = pd.DataFrame(json_data["prediction"])
    print(df)

    acc_features = df[["X","Y","Z","Sessione"]]
    time_features = df[["Time1","Time2","Time3","Sessione"]]
    extracted_features = extract_features(acc_features, column_id="Sessione")
    impute(extracted_features)
    time_features = time_features.groupby(['Sessione']).mean()
    data_predict = pd.concat([extracted_features, time_features], axis=1)

    y_predict = clf.predict(data_predict)


    print("PREDICTIONNNNNNNNNNN")
    print(y_predict)
    #predict
    if y_predict == -1:
        pred_text = 'Negative'
    else:
        pred_text = 'Positive'
    output = {'prediction': pred_text}
    return output

# api.add_resource(PredictSentiment, '/') # stiamo mettendo sulla funzione creata la route che si richiama tramite get

if __name__ == '__main__':
    #app.run(host='0.0.0.0', debug=True, port=environ.get("PORT", 5000)) # 0.0.0.0 fa girare llo script sull indirizzo della
macchina
    app.run(host='0.0.0.0', debug=True, port = 5000) # 0.0.0.0 fa girare llo script sull indirizzo della macchina
```