UNIVERSITÀ DI PISA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Triennale in Ingegneria Informatica

Riconoscimento spiegabile di emozioni mediante analisi di dati fisiologici con ensemble di alberi decisionali

Relatori:

Antonio Luca Alfeo

Mario G. C. A. Cimino

Candidato:

Luca Landi

ANNO ACCADEMICO 2023/2024

INDICE

ABSTRACT 4

CAPITOLO 1: INTRODUZIONE 6

1.1 eXPLAINABLE ARTIFICIAL INTELLIGENCE (XAI) 6

1.1.1 Tecniche in ambito locale e globale 7

1.1.2 Tecniche Perturbation-Based e Backpropagation-Based 9

1.1.3 Tecniche model-intrinsic e post-hoc 10

1.2 MODELLI MACHINE LEARNING UTILIZZATI 10

1.2.1 Multilayer Perceptron (MLP) 10

1.2.2 Random Forest 11

1.2.3 Gradient Boosting 11

CAPITOLO 2: CASO DI STUDIO 12

2.1 K-EMOCON DATASET 12

2.2 ESPERIMENTO E RACCOLTA DEI DATI 13

2.3 MODALITÀ RACCOLTA DEI DATI 14

2.4 STRUTTURA DEL DATASET 14

2.5 ANALISI DEI DATI 15

CAPITOLO 3: DESIGN E IMPLEMENTAZIONE 18

3.1 APPROCCIO GENERALE E SCELTA DEI MODELLI 18

3.2 APPROCCIO E IMPLEMENTAZIONE SOFTWARE 18

3.2.1 Approccio nello sviluppo del software 19

3.2.2 Implementazione del software 19

CAPITOLO 4: RISULTATI SPERIMENTALI 21

4.1 METRICHE DI VALUTAZIONE 21

4.1.1 Accuracy score 21

4.1.2 Tempo di eseuzione 22

4.1.3 Feature Importance 22

4.2 REPORT DELLA SPERIMENTAZIONE 24

4.2.1 Report 1: Soggetto n. 25 e Multilayer Perceptron 24
4.2.2 Report 2: variabile target Arousals sul dataset totale utilizzando MLP 26
4.2.3 Report 3: Un primo confrontro tra MLP, Random Forest e Gradient Boosting 28
4.2.4 Report 4: Analisi features importances del Random Forest e Gradient Boosting.. 31
4.2.5 Report 5: Analisi statistiche su subset di caratteristiche dei dispositivi E4 e H7 34
4.2.6 Report 6: Analisi approfondita sulle features più importanti del dataset 37
4.2.7 Report 7: Analisi approfondita sulle features meno importanti del dataset 42

CONCLUSIONI **46**

APPENDICE A: MANUALE D'USO E CODICE DEL SOFTWARE FINALE **48**

A.1 MANUALE D'USO 48
A.2 CODICE 50

APPENDICE B: TABELLE E IMMAGINI SUPPLEMENTARI **61**

SITOGRAFIA **68**

Abstract

L'intelligenza artificiale è sempre più presente nell'attualità della società mondiale, partendo dagli ambiti più leggeri, ad esempio videogiochi e app di intrattenimento, fino ad ambiti più delicati, ad esempio la sanità e la pubblica sicurezza. Sicuramente una sfida attuale è quella di poter sfruttare ed applicare questo nuovo ambito dell'informatica in qualsiasi tipo di attività umana. Un ruolo importante lo ricopre l'*emotion recognition*, una delle tante sfide riguardanti l'intelligenza artificiale, la sua importanza deriva dalle tante possibili applicazioni che si possono trovare nella società, tra cui in quelli in cui sono richieste competenze specifiche e approfondite (come settore medico e automobilistico). L'idea è che la possibilità, da parte di un sistema intelligente, di riconoscere le emozioni degli utenti e quindi di regolare il proprio comportamento sulla base di esse, possa migliorare l'efficacia delle interazioni uomo-macchina e portare benefici sotto molti altri aspetti, ad esempio la sicurezza degli utenti stessi.

Con questo elaborato ci si concentra ad analizzare dei modelli di Machine Learning conosciuti in letteratura, introdotti nel primo capitolo, per capire i punti di forza e di debolezza sulla loro applicazione nell'*emotion recognition*. Si utilizza e analizza nello specifico il dataset *K-EmoCon*, un dataset multimodale introdotto nel testo scientifico "*K-EmoCon, a multimodal sensor dataset for continuous emotion recognition in naturalistic conversations*" pubblicata nel 2020 da *C. Y. Park, N. Cha, S. Kang, A. Kim, A. H. Khandoker, L. Hadjileontiadis, A. Oh, Y. Jeong & U. Lee*. L'obiettivo prefissato è quello di capire se tali modelli possono essere soddisfacenti nel prevedere gli stati d'animo utilizzando segnali fisiologici campionati da dispositivi economici e indossabili. Dopo aver analizzato, nel secondo capitolo, le caratteristiche del contesto e delle modalità della raccolta dei dati, nonché dello stesso dataset, si effettuano una serie di sperimentazioni sequenziali utilizzando tali modelli di apprendimento automatico andando di volta in volta ad analizzare una configurazione nuova di tali modelli che possa fornire risultati sempre più efficaci ed efficienti. Si sviluppa infine un software che consente di riprodurre l'analisi dei modelli esaminati nelle sperimentazioni e analizzare le stesse performance su dispositivi diversi, vedi Appendice A. Una sperimentazione del genere può essere utile perché si va ad analizzare oltre ai modelli anche la struttura del dataset K-EmoCon. Infatti, come sarà approfondito nel quarto capitolo, utilizzando i segnali fisiologici come caratteristiche e le valutazioni sulle emozioni come valori target, cioè da prevedere, si riesce ad avere una visione di insieme su quali siano i segnali e i valori del dataset più importanti nel processo decisionale dei modelli di Machine Learning.

Si elabora quindi una valutazione sia sui modelli di apprendimento automatico, andando a valutare indici di accuratezza, tempi di esecuzione ecc. sia un'analisi sul dataset, andando a

capire quali valori del dataset siano più importanti per tali modelli e se le caratteristiche intrinseche del dataset stesso abbiano un impatto su tali modelli. Pertanto, questo elaborato produrrà risultati che saranno utili per l'applicazione del dataset *K-EmoCon*, nel campo dell'emotion recognition.

CAPITOLO 1: Introduzione

Prima di presentare il dataset, i modelli e i test effettuati è necessario dare una panoramica dei modelli di Machine Learning utilizzati, più in generale dell'Explainable Artificial Intelligence (XAI). Si approfondiscono i modelli del Multi-Layer Perceptron (MLP), Random Forest e Gradient Boosting, con le loro caratteristiche, pregi e difetti.

1.1 eXplainable Artificial Intelligence (XAI)

Il seguente paragrafo si sofferma sul concetto di XAI e da una classificazione dei modelli e tecniche che vengono utilizzate.

L'Explainable Artificail Intelligence, ovvero Intelligenza Artificiale spiegabile, è un campo dell'Intelligenza Artificiale sempre più in evidenza. Nasce con l'intento di rendere più trasparente il processo decisionale degli algoritmi di intelligenza artificiale, i quali sempre più presenti in qualsiasi ambito della società moderna stanno evolvendo e diventando sempre più complessi da comprendere e decifrare. La XAI è un punto di incontro per diverse tipologie di esigenze, tra cui la fiducia di chi utilizza tali modelli capendo le motivazioni dietro i risultati forniti, i regolamenti delle varie giurisdizioni che richiedono un certo livello di trasparenza per applicazioni in determinati ambiti delicati ma anche la maggiore semplicità in fase di debugging e comprensione del modello stesso.

Si introduce in seguito la tassonomia e le tecniche XAI introdotte dallo studio sopracitato. Si propone una classificazione in categorie ben definite con la quale si aiuta a capire gli algoritmi aumentando la chiarezza e l'accessibilità degli approcci. Lo studio definisce queste tre categorie:

1. **Scope**: l'ambito delle spiegazioni, può essere *locale* o *globale*, con alcune eccezioni che prevedono entrambe le tipologie. I metodi localmente spiegabili sono prestabiliti per fornire spiegazioni su come vengono fatte delle previsioni su singoli input, utili per capire delle decisioni effettuate dal modello in casi specifici. I metodi globalmente spiegabili forniscono invece una spiegazione del processo decisionale del modello nel suo complesso, quindi su più serie di dati in ingresso, utili per avere una panoramica generale su come le caratteristiche più importanti interagiscono per influenzare le previsioni globali.
2. **Metodologia**: consiste nel classificare l'algoritmo, del modello spiegabile, in base alla sua metodologia implementativa. Si distinguono quindi in algoritmi *backpropagation-based* e *perturbation-based.* Gli algoritmi che si basano sulla perturbazione mirano a capire come gli input contribuiscono nelle decisioni alterando gli stessi ingressi attraverso molteplici tecniche, tra cui l'oscuramento, sostituendo caratteristiche, usando

il mascheramento ecc. Questo metodo risulta particolarmente utile per interpretare i modelli più complessi. Gli algoritmi che si basano sulla backpropagation effettuano ad ogni passo in avanti nella rete delle valutazioni, utilizzando derivate parziali, che mirano a propagare informazioni sull'importanza dei nodi verso quelli di input per determinare come questi contribuiscono agli output. Delle tecniche utilizzate da questi algoritmi sono le *counterfactual explanations*, esse, attraverso una variazione degli ingressi, hanno lo scopo di spiegare come le uscite potrebbero cambiare se vengono forniti valori diversi di input.

3. **Utilizzo**. Un modello spiegabile può essere incorporato nei modelli di reti neurali oppure può essere un algoritmo esterno utilizzato per la spiegazione. Gli algoritmi *model-intrinsic* sono per la maggioranza specifici per il modello sui quali sono applicati e sfruttano la chiarezza intrinseca del modello per fornire le spiegazioni. Gli algoritmi *post-hoc* sono destinati a modelli già addestrati e sono facilmente applicabili indipendentemente dalle caratteristiche e architettura del modello stesso, sono largamente utilizzati per studiare i modelli complessi con applicazioni su diversi input come immagini testi e dati tabulari.

1.1.1 Tecniche in ambito locale e globale

Le tecniche spiegabili in ambito locale, come già introdotto, hanno il focus su una certa istanza di input e forniscono informazioni utilizzando le caratteristiche di tali input. Forniscono dati utili nel capire il perché di una previsione per un certo ingresso. Di seguito si elencano le tecniche con scope locale.

- *Activation Maximization*, l'obiettivo del metodo è minimizzare la perdita di massimizzazione dell'attivazioni dei singoli neuroni o strati all'interno della rete, andando ad analizzare strato per strato l'importanza delle caratteristiche per una certa istanza di input.
- *Saliency Map Visualization*, utilizzato largamente con le immagini, si evidenzia quali parti di un'immagine hanno avuto maggiore impatto sulle decisioni di un modello. Si utilizzano i gradienti per avere un riassunto equo dell'importanza dei pixel studiando quelli che hanno avuto maggiore impatto sugli output.
- *Layer-wise Relevance BackPropagation (LRP)*, utilizzato per calcolare la rilevanza per le singole caratteristiche dei dai in input scomponendo le previsioni di output. I punteggi di classe di un nodo in output vengono propagati e distribuiti verso gli ingressi andando, in questo modo, a calcolare i singoli punteggi di rilevanza per ogni input.

- *Local Interpretable Model-Agnostic Explanations (LIME)*, questo modello, il cui utilizzo si può trovare anche nella classificazione delle immagini, si applica per le singole istanze di input. Consiste nel trovare l'importanza di superpixel (ovvero patch di pixel) contigui in un'immagine sorgente rispetto alla classe di output e quindi spiega quali siano i percorsi continui che hanno un'alta importanza nella classe di output. Nel caso in cui sia applicato a testi o dati tabulari fornisce rispettivamente parti del testo o feature importance.
- *Shapley Additive Explanations (SHAP)*, modello che studia le previsioni di determinati input mostrando i singoli contributi delle caratteristiche verso le previsioni in output. Si basa sull'utilizzo della teoria dei giochi nello specifico in giochi di coalizione dove le caratteristiche di input hanno il ruolo di giocatori, i valori di Shapley sono calcolati per una equa distribuzione dei pagamenti, cioè l'importanza della previsione.

Le tecniche che forniscono spiegazioni in ambito globale intendono fornire spiegazione sul comportamento del modello nella sua interezza e quindi non più per un certo ingresso ma per una serie di ingressi. Le informazioni prodotte sono relative alle caratteristiche degli input, la correlazione degli output e sui modelli in generale, contribuendo a interpretare più facilmente modelli complessi come, ad esempio, i modelli basati su alberi decisionali. Alcune delle principali tecniche che forniscono spiegazioni globali sono le seguenti.

- *Global Surrogate Models*, questi modelli hanno lo scopo di approssimare un modello complesso e non lineare con altri modelli più semplici da interpretare come un albero decisionale. In questo caso si parla di ambito globale perché questi modelli spiegano il comportamento in modo generale del modello complesso, e non solo tale modello per una certa istanza di input.
- *Class Model Visualization*, l'obiettivo di questi modelli è generare per ogni classe di output delle immagini rappresentano maggiormente tale classe. In questo modo si può analizzare ciò che il modello ha appreso riguardo al processo decisionale delle previsioni.
- *LIME for Global Explanations*, è un'estensione del modello *LIME* citato precedentemente si basa sull'interpretare nel complesso le spiegazioni per singole istanze di input, ricavando tendenze pattern e informazioni importanti riguardo al processo decisionale del modello.
- *Concept Activation Vectors (CAVs)*, è un metodo globale che consiste rendere comprensibili agli umani, non singole caratteristiche ma concetti di livello più alto e in che modo questi pesino nel processo decisionale del modello.

- *Spectral Relevance Analysis (SpRAy)*, modello che si basa sulla tecnica *LRP*, analizza la struttura spaziale delle associazioni che si verificano con frequenza su quest'ultima, andando a trovare il comportamento normale e anormale del modello in esame attraverso un'analisi delle mappe di rilevanza delle attribuzioni LRP.

1.1.2 Tecniche Perturbation-Based e Backpropagation-Based

Le tecniche *perturbation-based* degli ingressi consistono nell'utilizzare metodi con i quali gli ingressi vengono perturbati, ad esempio sostituendo le caratteristiche degli input con degli zeri o valori casuali. Si elencano di seguito alcuni dei principali metodi che attuano tale metodo.

- *DeConvolution nets for Convolution Visualizations*, consistono nell'oscuramento di parti delle immagini, operano in modo inverso alle *Convolutional Neural Network* (CNN), quindi invece che ridurre lo spazio delle caratteristiche generano un'*activation map* che aiuta a capire e studiare il comportamento delle reti complesse.
- *Prediction Difference Analysis*, è un metodo che si basa sullo studio della rilevanza di ciascuna caratteristica degli ingressi. Questa tecnica analizza le variazioni delle previsioni degli output andando a oscurare o togliere una caratteristica dall'insieme totale.
- *Randomized Input Sampling for Explanation (RISE)*, è un metodo che utilizza maschere casuali dalle quali si ricavano immagini mascherate che vengono fornite a sua volta in input ai modelli. Dalla generazione delle *saliency map* da tali previsioni se ne ricava una finale che evidenzia le parti più rilevanti nel processo decisivo.

Le tecniche *Backpropagation-Based* consistono dell'utilizzare metodi dove si ha un flusso di informazioni inverso, dagli output verso gli input. Con tali dati è possibile poi studiare la rilevanza e l'influenza dei nodi della rete. Si elencano di seguito alcuni delle principali tecniche.

- *Saliency Maps*, queste mappe vengono generate usando il metodo del gradiente a partire dagli output fino agli input. Con i valori retro propagati del gradiente si generano tali mappe che definiscono quali caratteristiche sono più incisive nel processo decisionale.
- *Gradient class activation mapping (CAM)*, tecnica che trova maggiore applicazione nei classificatori di immagini. Attraverso query su classi specifiche e tecniche di *counterfactual explanations* mostra le regioni di un'immagine che contribuiscono positivamente o negativamente alla classe stessa.
- *Salient Relevance (SR) Maps*, utilizzano le mappe di rilevanza generate da LRP per una certa immagine per generare una mappa di rilevanza di ogni singolo pixel.

1.1.3 Tecniche model-intrinsic e post-hoc

Le tecniche *model-intrinsic* sono quindi presenti e fornite dal modello di intelligenza artificiale stesso. Quindi per loro natura sono *model-specific* e non posso essere riutilizzati per altri modelli se non prevedono una loro implementazione. Forniscono una spiegazione chiara e articolata dell'architetture e del processo decisionale del modello. Si elencano alcune delle tecniche più rilevanti.

- *Trees and Rule-based Models*, sono modelli basati su regole e alberi decisionali che per loro natura sono facilmente interpretabili. La presenza di regole precise (if, else, elseif) rendono il modello accurato e trasparente. Troviamo l'utilizzo di alberi decisionali anche nella generalizzazione delle tecniche *LIME* e *SHAP* al caso di ambito globale.
- *Generalized additive models (GAMs)*, sono tecniche lineari o non lineari che si affiancano ai modelli lineari che mettono in luce diversi tipologie di relazione tra gli output e gli input. Diverse implementazioni utilizzano però una quantità di alberi decisionali elevata per fornire dati accurati.

Le tecniche *post-hoc* permettono di spiegare modelli già addestrati permettendo una maggiore interpretabilità dei modelli stessi. La maggioranza di queste tecniche è model-agnostic, cioè sono tecniche sviluppate al di fuori del modello stesso ed applicabili ad altri modelli. In base alla conoscenza o meno dell'architettura del modello l'approccio si dice rispettivamente *white box* oppure black box.

1.2 Modelli Machine Learning Utilizzati

Si vanno adesso a presentare i modelli di machine learning utilizzati mostrando le caratteristiche essenziali, tra cui pregi e difetti.

1.2.1 Multilayer Perceptron (MLP)

Il Multilayer Perceptron è una tipologia di rete neurale artificiale, composta da più strati di neuroni. Il primo strato, detto di input, è composto dall'insieme di caratteristiche in ingresso alle quali è associato un nodo di input. Lo strato finale, detto di output, contenente il nodo in uscita dal quale si ha l'esito del processo decisionale per una certa variabile detta target. Infine, si hanno uno o più strati intermedi, detti nascosti, i quali neuroni sono completamente connessi con lo strato successivo e precedente della rete.

Tra le caratteristiche di questo modello di apprendimento automatico abbiamo che ha la capacità di manipolare dati non lineari ma per ottenere risultati ottimali richiede una corretta regolazione degli iperparametri, infine risulta particolarmente sensibile in presenza di dataset con

sbilanciamento delle classi, cioè quando nel dataset il divario tra classi più e meno rappresentate diventa eccessivo.

1.2.2 Random Forest

Il Random Forest è modello che si basa sugli alberi decisionali. Per *albero decisionale* si intende un processo decisionale strutturato gerarchicamente ad albero dove ad ogni nodo è associata una decisione, in questo caso relativa ad una caratteristica e le cui foglie rappresentano i diversi risultati della previsione. Le modalità attraverso le quali un albero viene generato dipendono dalla tipologia dell'albero stesso, un esempio è il *Classification and Regression Trees (CART)* che utilizza l'impurità di Gini per stabilire quali attributi andranno a far parte dei nodi. L'algoritmo del Random Forest genera un insieme di alberi decisionali, con un basso livello di correlazione, relativi ad un sottoinsieme casuale di caratteristiche. La previsione finale è data scegliendo quella che è generata dalla maggioranza degli alberi decisionali.

Tra le caratteristiche principali di questo modello abbiamo l'intrinseca accuratezza e trasparenza derivante dalla struttura interna del modello stesso, offre funzionalità per la valutazione dell'importanza delle caratteristiche, risulta efficiente per dataset sbilanciati ma risulta particolarmente costoso in termini di tempo di esecuzione.

1.2.3 Gradient Boosting

Il modello Gradient Boosting è anch'esso un modello che si basa sugli alberi decisionali. A differenza del random forest genera in modo additivo e sequenziale alberi decisionali che minimizzano la funzione di perdita dell'albero precedente.

Presenta caratteristiche tipiche al precedente modello anche se la sequenzialità nel generare alberi decisionali implica inevitabilmente un maggior tempo di esecuzione per un livello di accuratezza migliore.

CAPITOLO 2: Caso di studio

Nel seguente paragrafo si introduce il contesto delle sperimentazioni e il dataset utilizzato. L'intento di questo capitolo è di chiarire il contesto, la modalità e gli strumenti utilizzati per la raccolta delle informazioni e la struttura del dataset di tali informazioni. Si fa riferimento nella spiegazione di tali argomenti alla pubblicazione scientifica "*K-EmoCon, a multimodal sensor dataset for continuous emotion recognition in naturalistic conversations*" pubblicata nel 2020 da *C. Y. Park, N. Cha, S. Kang, A. Kim, A. H. Khandoker, L. Hadjileontiadis, A. Oh, Y. Jeong & U. Lee*.

2.1 K-EmoCon Dataset

Comunemente le emozioni sono viste come stati psicologici espressi attraverso i volti. La rilevazione degli stati emotivi attraverso la sola espressione facciale non è sufficiente per studiare le emozioni spontanee. Questo perché ricerche sostengono che le espressioni facciali relative a stati psicologici diversi, sono correlate e irregolari e quindi non sono in grado di distinguersi completamente. Inoltre, i dati ottenuti sono ricavati da emozioni indotte attraverso stimoli selezionati in un ambiente statico e controllato; quindi, tali dati sono difficilmente generalizzabili in scenari più realistici e istintivi. Per questo i dataset che si basano su questo approccio non sono adatti per lo studio degli stati emozionali che si verificano in modo spontaneo.

Approcci alternativi utilizzano contenuti multimediali e il crowdsourcing proprio per sopperire le carenze dell'approccio comune. L'abbondanza di materiale multimediale disponibile online e la partecipazione di una vasta rete di persone nell'annotazione dei dati costituisce un metodo efficiente e poco costoso, in particolare fornisce un grande campione di dati e una grande diversificazione dei soggetti. Di contro la maggior parte dei contenuti disponibili presenta situazioni in cui le espressioni facciali sono eseguite da attori che stanno recitando una parte e quindi anche una situazione psicologica, inoltre non è possibile accedere ad altre tipologie di dati fisiologici se non la sola espressione facciale. A tale proposito viene introdotto il *K-EmoCon*, un dataset multimodale ottenuto da un serie di dibattiti su questioni sociali utilizzando diverse tipologie di segnale fisiologico e contenuti multimediali introducendo valutazioni da tre prospettive diverse: il *soggetto stesso*, il *partner* del dibattito e un *osservatore esterno*.

Avere prospettive diverse nella valutazione di uno stato emozionale aumenta la precisione del riconoscimento delle emozioni. Attraverso il K-EmoCon viene introdotto un nuovo modello di valutazione che comprende tre differenti prospettive.

1. Il **soggetto**, il cui stato emotivo è in esame e produce un'*autovalutazione* su ciò che ha provato

2. Il **partner del dibattito**, colui che produce una valutazione sul soggetto, tenendo conto che interagisce direttamente con quest'ultimo e che è a piena conoscenza degli argomenti del dibattito che hanno indotto al soggetto stesso delle emozioni precise.

3. Un **osservatore esterno**, colui che produce una valutazione del soggetto senza essere a completa conoscenza degli argomenti del dibattito.

Quindi, troviamo un altro fattore nelle diverse prospettive, cioè il ruolo della *conoscenza contestuale* nella percezione e nel riconoscimento delle emozioni. Questo si contrappone ad altri studi, le cui valutazioni tengono in considerazione soggetti che hanno solo un tipo di conoscenza contestuale, ad esempio solo soggetti esterni. Da notare che si può verificare che le autovalutazioni e le valutazioni da terzi possono non combaciare, questo può essere dovuto dal fatto che una persona tende a nascondere oppure non è in grado di interpretare correttamente le proprie emozioni

2.2 Esperimento e raccolta dei dati

Ai 32 partecipanti dell'esperimento è stato chiesto di sostenere un dibattito a coppie su un argomento di interesse sociale, cioè l'accoglienza dei rifugiati dello Yemen sull'isola sudcoreana Jeju. A tutti i candidati è stato fornito in anticipo, per e-mail, due articoli, uno a favore e uno contro la questione per familiarizzare con l'argomento stesso. Le coppie sono state formate in modo casuale e in base alla disponibilità dei soggetti stessi.

Il contesto dell'esperimento è stato posto per rappresentare al meglio un dibattito che può nascere spontaneamente tra due individui, in modo da raccogliere le emozioni che possono sorgere naturalmente nella quotidianità. Inoltre, permette di studiare l'errata percezione delle emozioni. Essendo un contesto formale ci si aspetta che i candidati regolino le proprie emozioni in un modo socialmente appropriato, potendo studiare in questo modo quelle meno pronunciate che hanno maggiori probabilità di essere fraintese. A questo proposito si affianca alla valutazione espressiva del volto, dei segnali fisiologici ottenuti da dei dispositivi mobili e indossabili. Il *K-EmoCon* fornisce materiale per poter esaminare le emozioni "impercettibili", ad esempio come l'intensità di un'emozione possa influenzare l'accuratezza della decodifica di tale stato d'animo.

Sono stati utilizzati degli smartphone per catturare le espressioni facciali e i movimenti della parte superiore del corpo di entrambi i partecipanti al dibattito. Inoltre, vengono indossati dei dispositivi mobili:

- *Empatica E4 Wristband*, che misura l'accelerazione nei 3 assi, temperatura corporea, l'attività elettrodermica (EDA), il fotopletismogramma (PPG), la frequenza cadiaca e l'interbeat interval (IBI).

- *Polar H7 Bluetooth Heart Rate Sensor*, rileva il battito cardiaco attraverso l'elettrocardiogramma (ECG), affiancandolo al PPG il quale è sensibile ai movimenti.
- *NeuroSky MindWave Headset*, che ricava l'elettroencefalogramma (EEG).
- *LookNTell Head-Mounted Camera*, con una telecamera che cattura i video in prima persona.

2.3 Modalità raccolta dei dati

I dati vengono raccolti da degli esaminatori che gestiscono la sessione di dibattito. Possiamo distinguere questa procedura in quattro fasi:

1. *Onboarding*. Dopo aver compilato i moduli di consenso per la raccolta e la divulgazione dei dati, i candidati scelgono in modo volontario o casuale la parte da prendere in merito alla questione dell'ammissione dei rifugiati. Dopodiché viene fornito ulteriore tempo per preparare le proprie argomentazioni attraverso l'uso di materiale fornitogli precedentemente. Infine, vengono fatti indossare correttamente i dispositivi mobili.
2. *Baseline measurement*. Lo scopo di questa fase consiste nell'indurre uno stato emotivo neutro nei soggetti in modo da stabilire una linea base per ciascun partecipante. In questo esperimento i partecipanti hanno guardato *Color Bars*, favorendo uno stato neutro come riportato da studi recenti.
3. *Dibattito*. La sessione di dibattito dura 10 minuti diviso in turni da 2 minuti per candidato, con la possibilità di intervento in qualsiasi momento anche dal partner in modo da replicare una conversazione naturale.
4. *Annotazione dei risultati*. Ai partecipanti al dibattito e ai valutatori esterni è stato chiesto di annotare le emozioni ad intervalli di cinque secondi. Nello specifico, ai partecipanti al dibattito sono state fornite solo le registrazioni audiovisive catturate attraverso gli smartphone (POV in seconda persona). Nonostante questo possa portare a una prospettiva di un valutatore esterno, questa scelta è stata presa perché i contenuti in prima persona (catturati dal dispositivo *LookNTell Head-Mounted Camera*) rispetto a quelli in seconda persona trascurano alcune informazioni rilevanti come gli sguardi dei candidati.

2.4 Struttura del dataset

Si riporta, in accordo con sopracitato, la struttura del dataset, il quale è disponibile alla consultazione previa richiesta. Si elencano le principali directories.

- *metadata.tar.gz*, contiene tabelle ausiliare, tra cui l'identificativo dei soggetti, il timestamp di inizio e fine dibattito e la disponibilità dei file per ogni soggetto.
- *data_quality_tables.tar.gz*, contiene tabelle per ogni dispositivo che riportano informazioni riguardanti la qualità dei segnali fisiologici.
- *debate_audios.tar.gz* e *debate_recordings.tar.gz*, contenenti le registrazioni audio e video dei dibattiti
- *naurosky_polar_data.tar.gz* e *e4_data.tar.gz*, cartelle riguardanti i due dispositivi che rilevano segnali fisiologici. Ogni cartella ha una tabella per ogni tipologia di segnale fisiologico, ad esempio temperatura, ECG, frequenza cardiaca ecc. Ogni tabella relativa ad un segnale, contiene per ogni soggetto tutta la campionatura del segnale stesso nel tempo.
- *emotion_annotations.tar.gz*, contenente delle sottocartelle riguardanti le valutazioni, una per le autovalutazioni, una per le annotazioni del partner e una per le annotazioni esterne. Dove ognuna contiene le annotazioni per ogni soggetto intervallate da cinque secondi.

2.5 Analisi dei dati

Come mostrato dalla figura 1, le annotazioni delle emozioni risultano sbilanciate prevalentemente verso stati emotivi neutri, in accordo con quanto previsto dato che generalmente le persone sono più neutre che arrabbiate o tristi.

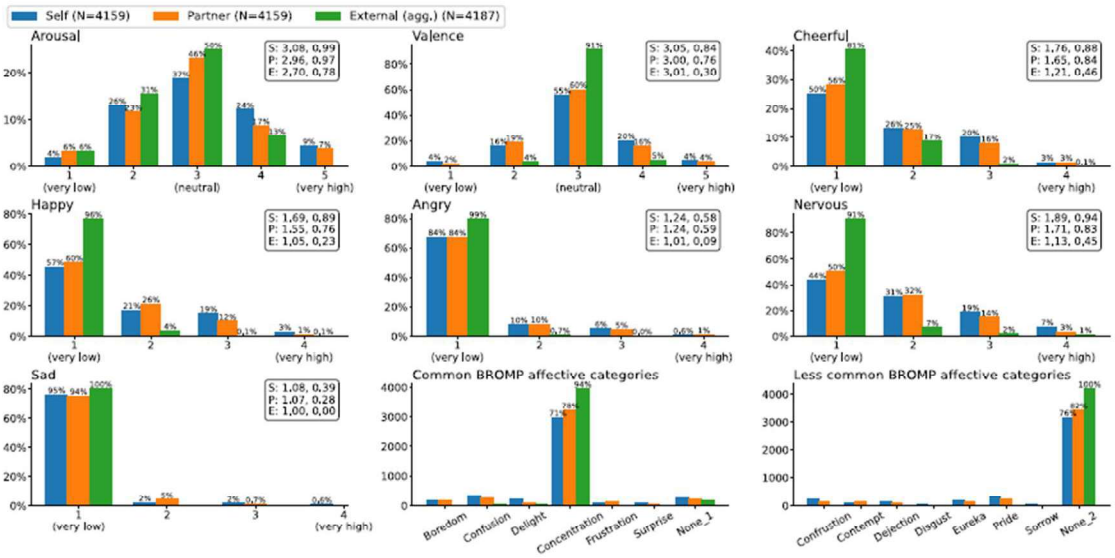

Figura 1.Presa dallo studio sopracitato, illustra la frequenza e la distribuzione delle valutazioni sull'emozioni.

L'Inter-rater reliability (IRR), ovvero l'affidabilità tra i valutatori, utilizzata l'*alpha di Krippendorff*, cioè una statistica generalizzata che misura l'accordo tra valutatori. In figura 2

riportate le mappe di calore dei coefficienti alfa calcolate per sette emozioni. Prima del calcolo i valori sono stati scalati rispetto ad un valore neutro, questo considerando la moda statistica per ogni sensazione (colonna) come neutro e sottraendola ai singoli valori. Tutto questo per evitare la sottostima del IRR dovuta a una interpretazione individuale dei valutatori eterogenea. Mentre per alcune sensazioni abbiamo una scala contenente un valore neutro (valenza e arousal), per altre questo non lo abbiamo (felicità, tristezza, allegria…), quindi per tali emozioni l'assegnazione di un valore dipende dall'interpretazione individuale e quindi è un fatto molto più soggettivo. A tale proposito viene utilizzato il metodo di sottrazione della moda in modo tale da valutare i coefficienti alfa non tanto su un accordo assoluto ma sulle variazioni relative delle emozioni nel tempo. Si nota che i coefficienti alfa sono in generale bassi, in particolare quando si confrontano le annotazioni SP (self vs. partner) e SE (self vs esterni) dove il valor medio è molto vicino allo zero. Questo schema evidenzia che molto probabilmente abbiamo una differenza di percezione delle emozioni per prospettive differenti, aprendo l'argomento a studi più approfonditi.

Tuttavia, i dispositivi per la raccolta dei dati presentano delle limitazioni, ad esempio il sensore EEG che è suscettibile ai rumori come l'espressione accigliata o al movimento degli occhi. Inoltre, la scelta di instaurare un dibattito a turno può aver soppresso le espressioni emotive, poiché una manifestazione eccessiva di emozione è considerata indesiderabile durante un dibattito. Quindi potrebbe essere per questo motivo che il livello di agreement tra le autovalutazioni e le valutazioni del partner/esterni risulti basso.

Altro fatto da sottolineare è l'aver preso in considerazione solo una precisa parte della società: giovani universitari di etnia asiatica sia come candidati che come valutatori. Inoltre, non sono stati presi in considerazioni altri dati come il rapporto tra le coppie del dibattito, il livello dell'inglese parlato e la familiarità dei singoli candidati con l'argomento specifico. Tutti variabili potrebbero aver generato maggiore discrepanza nelle valutazioni.

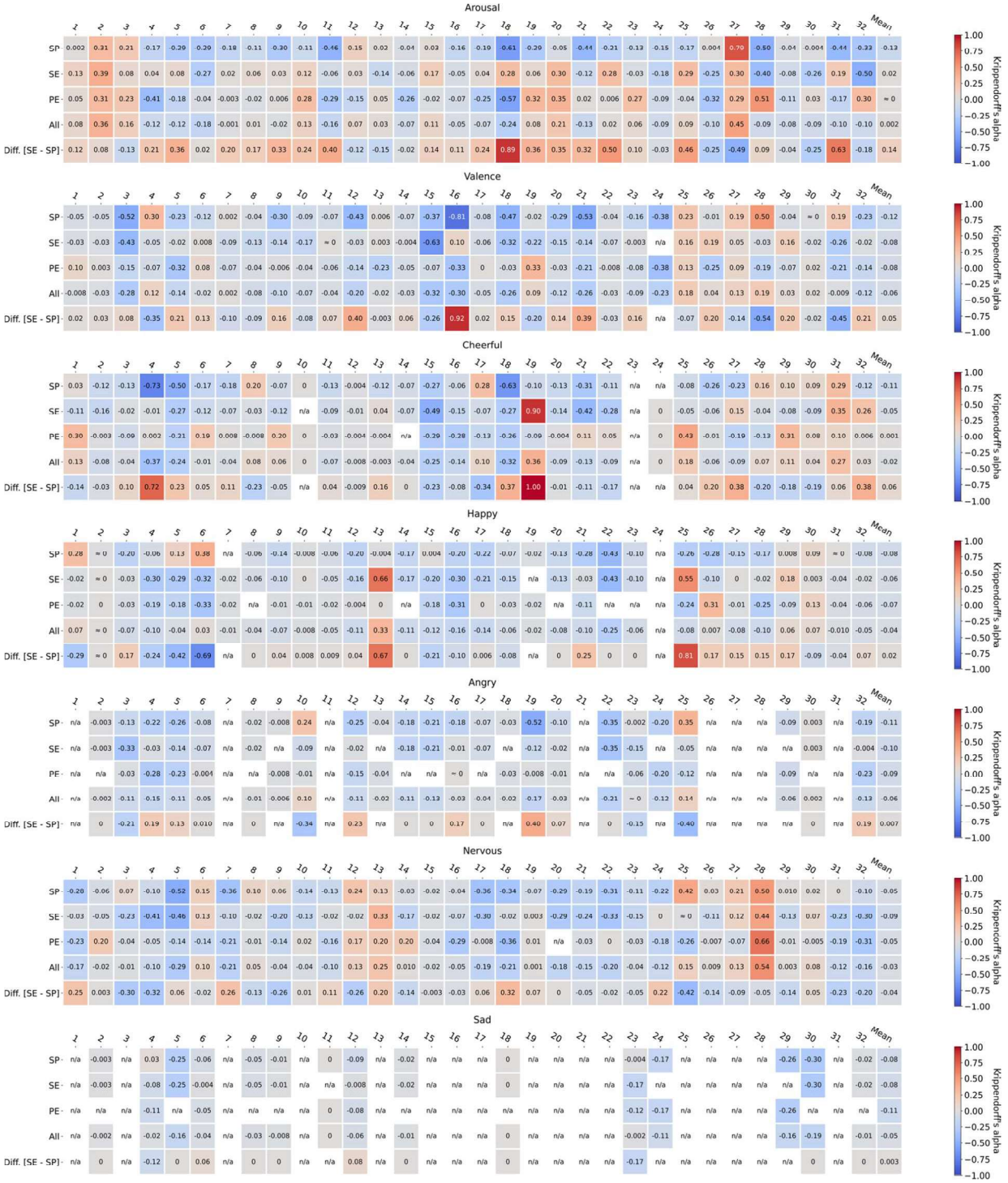

Figura 2. Presa dallo studio sopracitato, si mostrano le mappe di calore relative all'IRR calcolate utilizzando l'alpha di Krippendorff.

CAPITOLO 3: Design e implementazione

In questo capitolo si andrà ad esporre quale approccio è stato utilizzato in questo studio andando ad illustrare le motivazioni dietro la scelta dei modelli di machine learning utilizzati. Si illustrerà, poi, il software finale basato sui risultati di questo elaborato, le scelte implementative e la sua generica articolazione. Inoltre, si andranno ad illustrare casi d'uso e le varie classi del software stesso.

3.1 Approccio generale e scelta dei modelli

Prima di iniziare ad esporre i motivi principali della scelta dei modelli utilizzati, è necessario fare delle osservazioni sulle caratteristiche del dataset K-EmoCon. Come presentato dalla pubblicazione scientifica sopracitata, per quanto riguarda le variabili che possiamo utilizzare come oggetto delle previsioni, ovvero quelle relative alla parte delle valutazioni, sono caratterizzate dalla presenza generale di uno sbilanciamento delle classi, in particolare verso quelle classi relative agli stati d'animo neutri e pacati.

Dall'esamina attraverso i principali modelli di machine learning con lo scopo di prevedere stati d'animo attraverso segnali fisiologici basando il loro addestramento su questo tipo di dataset ci aspettiamo che algoritmi, come il *Multilayer Perceptron*, presentino criticità nell'addestramento e nei risultati finali dovuti alla loro alta sensibilità nello sbilanciamento nelle classi. Mentre per algoritmi basati su alberi decisionali, come *Random Forest* e *Gradient Boosting*, ci si aspetta che tale sbilanciamento delle classi abbia un peso minore per quanto riguarda l'efficienza nell'addestramento e nella correttezza delle previsioni finali.

L'approccio operativo generale di questo elaborato sarà di verificare le criticità per *il Multilayer Perceptron*, confrontare le prestazioni di quest'ultimo con i due modelli basati su alberi decisionali, fare un confronto tra il *Random Forest* e *Gradient Boosting* usando vari parametri di confronto ed infine trovare il modello, tra questi, che più si adatta a questo specifico dataset e una sua configurazione ottimale. Come verrà specificato ulteriormente nel quarto capitolo si è impostato la fase sperimentale in più report sequenziali, dove ogni report è stato impostato tenendo conto dei risultati del precedente, andando adottare strategie diverse oppure analizzando più specificatamente tali risultati.

3.2 Approccio e implementazione software

Nel seguente paragrafo si illustra il software finale generato sulla base dell'analisi effettuata in questo elaborato, nonché le scelte implementative e la struttura generale. Si procede, poi, andando a illustrare i casi d'uso e i diagrammi di classe. Il software è stato implementato nel

linguaggio *Python* utilizzando come ambiente *Google Colab* maggiori informazioni sul codice si trovano all'Appendice A.

3.2.1 Approccio nello sviluppo del software

Sulla base dei risultati dell'analisi dei vari modelli, vedi quarto capitolo, il software è stato pensato allo scopo di analizzare i due modelli basati su alberi decisionali, Random Forest e Gradient Boosting. In particolare, permette la visualizzazione delle feature importance e i grafici relativi al tempo di addestramento dei modelli e del grado di accuratezza delle previsioni. Attraverso essi permette la scelta del modello e dei parametri migliori, con i quali è possibile visualizzare le previsioni con dati nuovi non appartenenti al subset di addestramento e testing. Come sarà poi specificato meglio nel prossimo sottoparagrafo, avremo due tipologie di utenti, un Machine Learning Engineer che, visualizzate le feature importance e i grafici implementati, proporrà un modello da validare, e un Decision Maker che validerà tale modello e avrà la possibilità di fare nuove previsioni.

3.2.2 Implementazione del software

Per quanto riguarda il manuale d'uso e il codice del software e le parti salienti si rimanda la lettura all'Appendice A. Per una maggiore chiarezza implementativa si illustrano in figura 3 i casi d'uso del software.

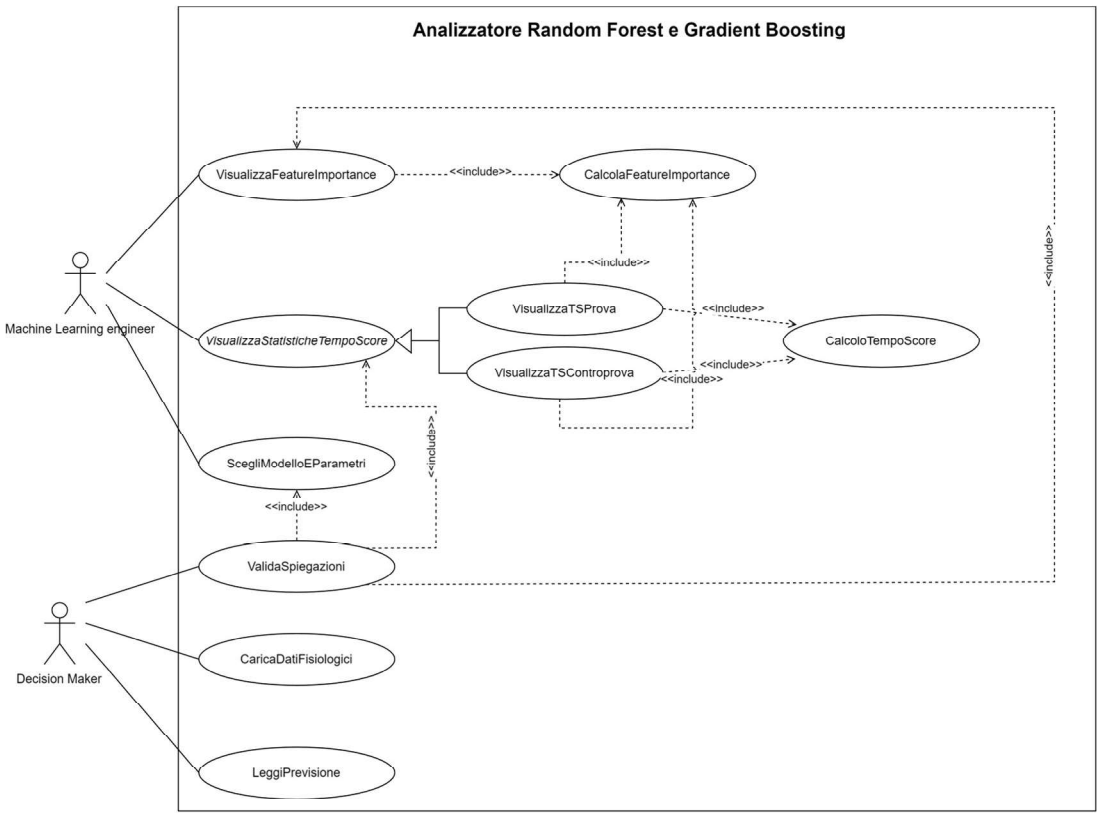

Figura 3. Casi d'uso del software finale specificati tramite Unified Modeling Language (UML)

Come anticipato abbiamo un Machine Learning Engineer, il quale visualizza le feature importance, visualizza il grafico *"tempo di esecuzione – score"* in relazione alle feature utilizzate dopodiché seleziona il modello e i parametri migliori. Abbiamo, inoltre, un Decision Maker, ovvero un soggetto non competente nell'utilizzo di tali modelli e che si affida, per una loro valutazione, al Machine Learning Engineer. Esso valida il modello proposto e con tale modello ha la possibilità di inserire nuovi dati ed effettuare nuove previsioni.

Per un'analisi dettagliata dei casi d'uso si rimanda alle tabelle B.1, B.2, B.3, B.4, B.5, B.6, B.7, B.8, B.9, B.10 dei casi d'uso specifici in Appendice B.

In figura 4 si descrivono, tramite i diagrammi di classe, le classi utilizzate nel software.

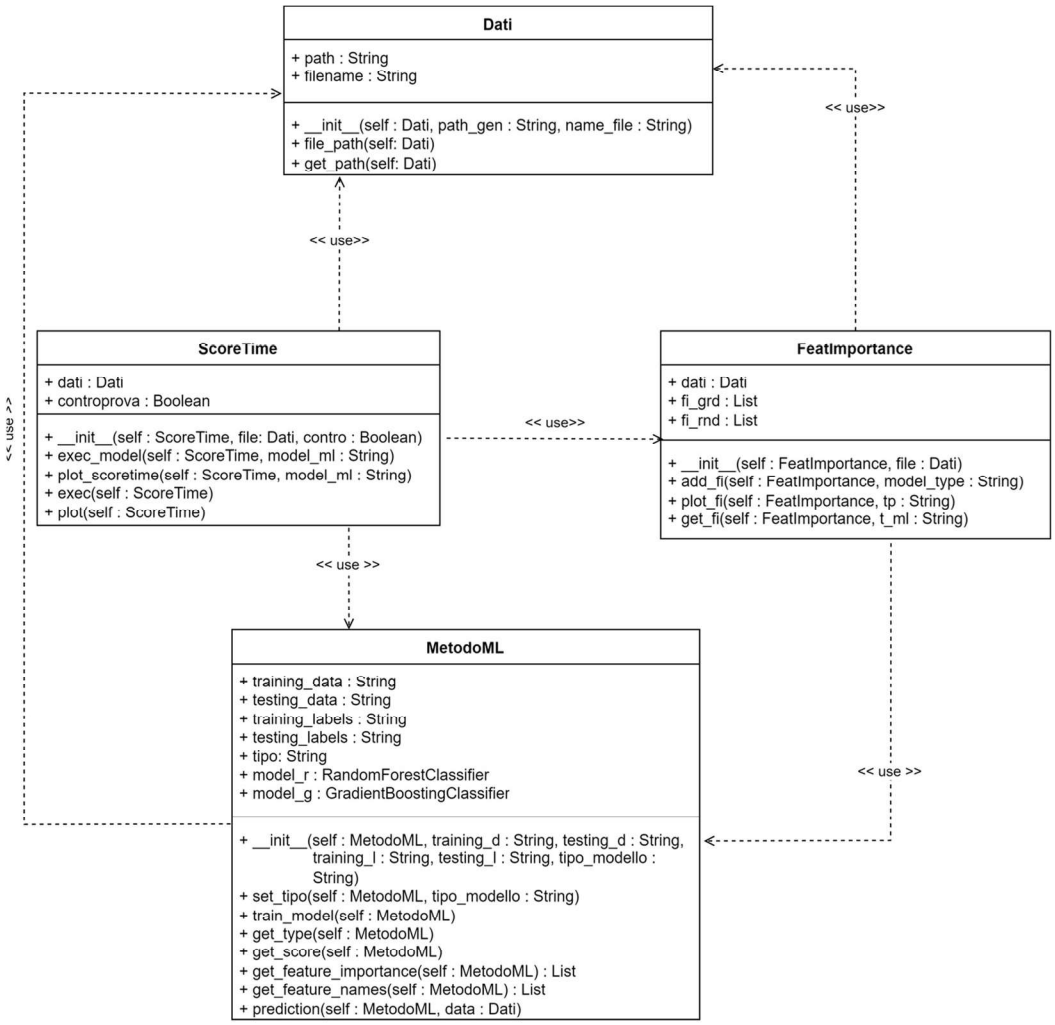

Figura 4. Si mostrano i diagrammi di ogni classe presente nel software, per informazioni dettagliate sulle funzioni e sugli attributi si rimanda all'Appendice A

I casi d'uso e i diagrammi di classe sopra descritti congiuntamente al manuale d'uso e al codice del software presente nel paragrafo A.2 dell'Appendice A forniscono una descrizione dettagliata sull'implementazione del software finale.

CAPITOLO 4: Risultati sperimentali

Nel seguente capitolo si illustrano nel dettaglio la parte sperimentale dell'elaborato, si introducono le metriche di valutazione utilizzate nei report e per ogni report si illustrano i risultati attesi lo svolgimento e i risultati finali. Come già accennato in precedenza, questo studio comprende un insieme di report, dove per ogni elaborato presi i risultati del precedente li analizza e dettaglia oppure utilizza un approccio diverso per avere esiti migliori.

Si introduce analizzando il Multilayer Perceptron fino a quando non si avranno risultati non soddisfacenti, si introdurranno quindi i modelli del Random Forest e Gradient Boosting andando ad evidenziare le differenze di prestazione col modello precedente. Si andrà poi a confrontare i due modelli basati su alberi decisionali trovando la configurazione ottima attenendosi a dei parametri descritti nel prossimo paragrafo.

4.1 Metriche di valutazione

Come accennato nell'esamina dei report e quindi dei modelli applicati si utilizzano parametri di varia natura. Si elencano fornendo una descrizione dei parametri utilizzati.

4.1.1 Accuracy score

L'*accuracy score* è uno strumento utile per valutare la correttezza delle previsioni di un modello di machine learning addestrato. Consente di quantificare le previsioni corrette sul totale delle previsioni che ha effettuato un algoritmo.

$$\boldsymbol{Accuracy\_score} = \frac{\boldsymbol{numero\ previsioni\ corrette}}{\boldsymbol{numero\ previsioni\ totali}} \qquad (5.1)$$

Nel nostro caso viene utilizzato in combinazione alla tecnica *train-test split,* nella quale le rows del dataset vengono partizionate in una prima parte per il training del modello ed una seconda per il testing. La parte relativa al testing viene utilizzata anche per calcolare l'*accuracy score*. Infatti, le previsioni effettuate con questa porzione di dati vengono confrontate con il valore effettivo di tali variabili target, contando quindi le corrette previsioni.

Ovviamente non fornisce informazioni aggiuntive sulle previsioni scorrette e si tratta solo di uno strumento che non sempre fornisce un alto grado di affidabilità delle previsioni. Ad esempio, in un dataset con un alto sbilanciamento delle classi non sempre fornisce informazioni su come le classi minoritarie sono rappresentate correttamente questo perché un modello può tendere a prevedere sempre la classe maggioritaria e potenzialmente ottenere un buono score.

4.1.2 Tempo di eseuzione

Un altro parametro utilizzato è il tempo di esecuzione, cioè il tempo necessario affinché il processo di addestramento del modello si porti a termine. Data la grandezza del dataset, fin da subito ma in linea con le aspettative, l'esecuzione di ogni report ha richiesto un tempo di esecuzione totale elevato dovuto essenzialmente alla fase di training dei modelli di Machine Leaning. In questa fase, infatti, il modello viene addestrato in più iterazioni fino a quando non vengono soddisfatti dei parametri di performance interni al modello, oltre alle tecniche e alla complessità del modello stesso dipende inevitabilmente dalla grandezza del dataset in esame. Come sarà illustrato nella sperimentazione, si adotteranno tecniche di selezione delle caratteristiche in modo tale da ridurre, per quanto possibile, la durata di questa fase. I valori di questo parametro si riferiscono in numero di secondi trascorsi, per maggiore leggibilità nei grafici delle figure 22, 23, 25 e 26 esso è riportato nel formato *'HH-MM-SS'*.

4.1.3 Feature Importance

Un altro parametro che ha avuto importante impatto nella sperimentazione sono le *feature importance,* ovvero l'incidenza di ogni singola feature nel processo decisionale che determina la previsione di una certa variabile target. Esse sono delle tecniche di XAI, vedi primo capitolo, che l'implementazione, tramite *Scikit-learn* in *Python*, del *Grandient boosting* e del *Random forest*, forniscono e indicano il valore dell'importanza delle features per un certo modello generato per particolare set di dati (o set di training).

Dal punto di vista matematico la *feature importance* si trova valutando ogni nodo di un albero decisionale usando una misura di impurità, che essendo un problema di classificazione e trovandosi ad utilizzare *Scikit-learn,* corrisponde all'*impurità di Gini*.
Possiamo mostrare il calcolo matematico applicato sia dal *Random forest* che dal *Gradient boosting* procedendo per step:

- Step 1) *Calcolo dell'importanza di un nodo.*

$$\boldsymbol{ni_j = w_j C_j - w_{left(j)} C_{left(j)} - w_{right(j)} C_{right(j)}} \qquad (5.2)$$

  con ni_j importanza del nodo j, w_j il numero pesato di campioni che raggiungono il nodo j, C_j il valore di impurità del nodo j, $left(j)/right(j)$ figlio destro e sinistro del nodo j.
- Step 2) *Calcolo dell'importanza di ogni feature di un certo albero decisionale*

$$\boldsymbol{fi_i = \frac{\sum_{w:\ nodo\ w\ si\ divide\ sulla\ feature\ i} ni_w}{\sum_{k \in\ ogni\ nodo\ dell'albero} ni_k}} \quad (5.3)$$

con fi_i importanza della feature i, ni_w importanza del nodo w. Dopodiché è possibile normalizzare tale valore tra 0 e 1 dividendolo per la somma di tutte le *features importances* di un certo albero nel seguente modo:

$$\boldsymbol{norm(fi_j) = \frac{fi_i}{\sum_{k \in ogni\ feature} fi_k}} \quad (5.4)$$

- Step 3) *Calcolo valore finale dell'importanza delle features*
  Il valore finale per una certa feature è dato dalla media di tutte i valori dell'importanza tra tutti gli alberi generati dal modello per tale feature.

$$\boldsymbol{RFfi_i = \frac{\sum_{k \in ogni\ albero} norm(fi_i)_k}{T}} \quad (5.5)$$

con $RFfi_i$ valore finale della *feature importance*, $norm(fi_i)_k$ valore normalizzato della *feature importance* di un certo albero k, T numero totale di alberi generati dal modello.

Prima di introdurre il lavoro svolto negli elaborati è necessario chiarire come l'impurità di Gini viene calcolata e come viene utilizzati dai modelli che si basano sugli alberi decisionali.

L'impurità di Gini viene usata nei problemi di classificazione per definire quali features devono essere associate ai nodi degli alberi decisionali generati da un modello di classificazione. In particolare, corrisponde alla probabilità che un nuovo dato casuale venga classificato erroneamente quando viene assegnata un'etichetta di classe in base alla distribuzione delle classi nel dataset.

Quindi l'impurità di Gini di un certo nodo viene calcolata nel seguente modo:

$$\boldsymbol{G = \sum_{i=1}^{C} p(i) * (1 - p(i))} \quad (5.6)$$

con C numero delle classi, $p(i)$ la probabilità che un campione casuale appartenga alla classe i. Da notare che il valore minimo si ottiene quando, per un certo nodo, la variabile target presenta solo un valore di una certa classe e quindi l'impurità di Gini sarà pari a 0. Il valore massimo si ottiene invece quando, per quel nodo, si ha una distribuzione uniforme delle classi, quindi sarà pari ad $\left(1 - \frac{1}{C}\right)$, con C il numero totale delle classi.

Quindi la convenienza ad aggiungere un nodo ad un albero decisionale si ha quando il valore dell'impurità di un dataset è maggiore di quello che si otterrebbe aggiungendo quel nodo e quindi dividendo il dataset originale in due subset, 1 per ramo. Il valore dell'impurità totale dei rami si ottiene pesando i valori dei rami per il numero degli elementi di ciascun subset, quindi:

$$\boldsymbol{G_{total_{branches}}} = \frac{\boldsymbol{N_{right}}}{\boldsymbol{N_{tot}}} \boldsymbol{G_{right_{branch}}} + \frac{\boldsymbol{N_{left}}}{\boldsymbol{N_{tot}}} \boldsymbol{G_{left_{branch}}} \qquad (5.7)$$

con N_{tot} numero di elementi del set iniziale, N_{right} e N_{left} numero di elementi del ramo sinistro e destro. Quindi quando $G_{total\_branches}$ è minore dell'impurità del dataset iniziale si ha convenienza ad immetterlo.

4.2 Report della sperimentazione

Nel seguente paragrafo si andrà a proporre i report effettuati allo scopo di capire le conclusioni di questo elaborato. Si andranno ad analizzare i report soffermandoci sul significato del report, su come è stato svolto operativamente e sui risultati ottenuti.

4.2.1 Report 1: Soggetto n. 25 e Multilayer Perceptron

Il seguente report è stato effettuato sui dati del soggetto numero 25, nello specifico analizzando il *self_valence*, ovvero l'autovalutazione sulla valenza e cioè che esprime *attrazione intrinseca o repulsione*. Attraverso l'algoritmo di classificazione, usando MLP, si istruisce lo strumento a fare delle previsioni della variabile target (*self_valence*) utilizzando come features i dati provenienti dai dispositivi elettronici. È inoltre possibile misurare il grado di accuratezza di tali previsioni, il che ci porta a concludere se tale metodo è idoneo o meno per la variabile in esame.

	self_valence
count	57503.000000
mean	3.771386
std	0.419944
min	3.000000
25%	4.000000
50%	4.000000
75%	4.000000
max	4.000000

Tabella 1. Si riportano in tabella le statistiche della variabile in valutazione (self_valence)

Innanzitutto, il dataset completo relativo al soggetto 25 è stato filtrato prendendo in considerazione solo la colonna *self_valence* oltre a tutti gli altri dati dei dispositivi. Attraverso la procedura "*Train test split*", usando le percentuali di default, si ricavano i dati per il training (75%) e per il testing (25%) del modello. Per ottimizzare lo strumento viene effettuata, mediante la procedura "*GridSearch*", la ricerca dei valori migliori per tale modello parametrizzando il numero massimo di iterazioni. Dopodiché viene effettuato il training e il testing dei dati e ottenuta la configurazione dei parametri ottimale relativo all'*accuracy_score*. Per maggiore chiarezza il procedimento viene schematizzato in figura 5.

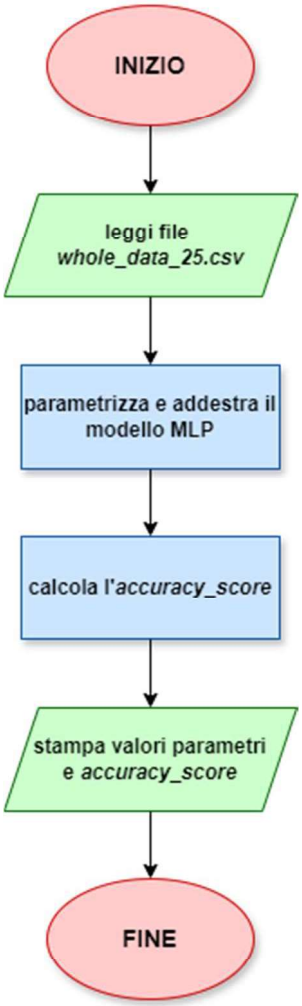

Figura 5. Diagramma di flusso relativo al primo report.

I risultati ottenuti sono:

- La *configurazione ottimale*, che risulta pari a *max_iter=250*, con dei parametri forniti pari a (100, 150, 200, 250, 300).
- Lo *score*, quindi livello di accuratezza della previsione, superiore a 0,90.

Esaminando questi risultati si ottiene che è possibile usare MLP con parametro *max_iter* pari a 250 per ottenere delle previsioni che hanno livello di accuratezza elevato (superiore al 90%).

4.2.2 Report 2: variabile target Arousals sul dataset totale utilizzando MLP

Il seguente report è stato effettuato sull'intero set dei dati, nello specifico analizzando la valutazione sull'*Arousal*, ovvero l'intensità dell'attivazione psicofisiologica di un organismo. Tale valore può variare tra un livello basso (disattenzione), medio (attenzione) e alto (panico, stress, rabbia...). Attraverso l'algoritmo di classificazione, usando MLP, si istruisce lo strumento

a fare delle previsioni della variabile target (*Arousal*) utilizzando come features i dati provenienti dai dispositivi elettronici. È inoltre possibile misurare il grado di accuratezza di tali previsioni, il che ci porta a concludere se tale metodo è idoneo o meno per la variabile in esame.

	AROUSAL
count	1.007154e+06
mean	1.498938e+00
std	4.999991e-01
min	1.000000e+00
25%	1.000000e+00
50%	1.000000e+00
75%	2.000000e+00
max	2.000000e+00

Tabella 2. Statistiche della variabile Arousal.
Da notare che il livello dell'arousal rimane sempre basso (range da 1 a 5).

Attraverso la procedura "*Train test split*", usando come il precedente test le percentuali di default, si ricavano i dati per il training e per il testing del modello. Dopodiché viene effettuato il training e il testing dei dati. I risultati ottenuti sono uno *score*, quindi livello di accuratezza della previsione, di circa 0,62 ottenuto con mediamente 70 iterazioni (*n_iter)*. In figura 6 si riporta per maggiore chiarezza il diagramma di flusso di questo report.

Esaminando questi risultati si ottiene che le previsioni hanno livello di accuratezza non ideale (circa 60%) rispetto al report precedente, il quale aveva una accuratezza molto più elevata (circa 90%), quindi il modello generato può essere considerato non idonea per tale variabile e per tale set di dati.

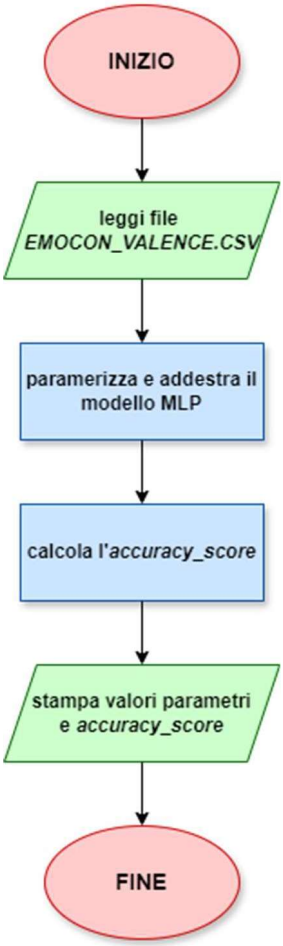

Figura 6. Diagramma di flusso relativo al Report 2.

4.2.3 Report 3: Un primo confrontro tra MLP, Random Forest e Gradient Boosting

L'uso dell'MLP *Classifier* per questo tipo di dataset non produce risultati ottimali come visto precedentemente. Un altro tentativo è quello usare altre tipologie di modelli, ad esempio quelle che si basano su alberi decisionali come il *Gradient boosting classifier* oppure *Random forest classifier*. Questo dataset come visto precedentemente risulta sbilanciato verso emozioni neutre, modelli che si basano su alberi decisionali risultano meno sensibili allo sbilanciamento del dataset rispetto al *Multi-layer Perceptron,* come già accennato nel primo paragrafo. A conclusione di ciò ci si aspetta una maggiore accuratezza delle previsioni utilizzando il *Gradient boosting classifier* oppure *Random forest classifier*.

Come nell'ultimo report si utilizza come variabile features la valutazione sull'*Arousal*. Dopodiché si utilizza il metodo "*train test split"* per ricavare, come al solito, i dati della fase di training e della fase di testing. Si effettuano quindi due prove una con il *Random Forest* e l'altra con il *Gradient Boosting*, facendo ovviamente prima il training poi il testing. Quindi si ottiene

l'accuratezza delle previsioni calcolate. In figura 7 si riporta uno schema generale dell'organizzazione di questo report.

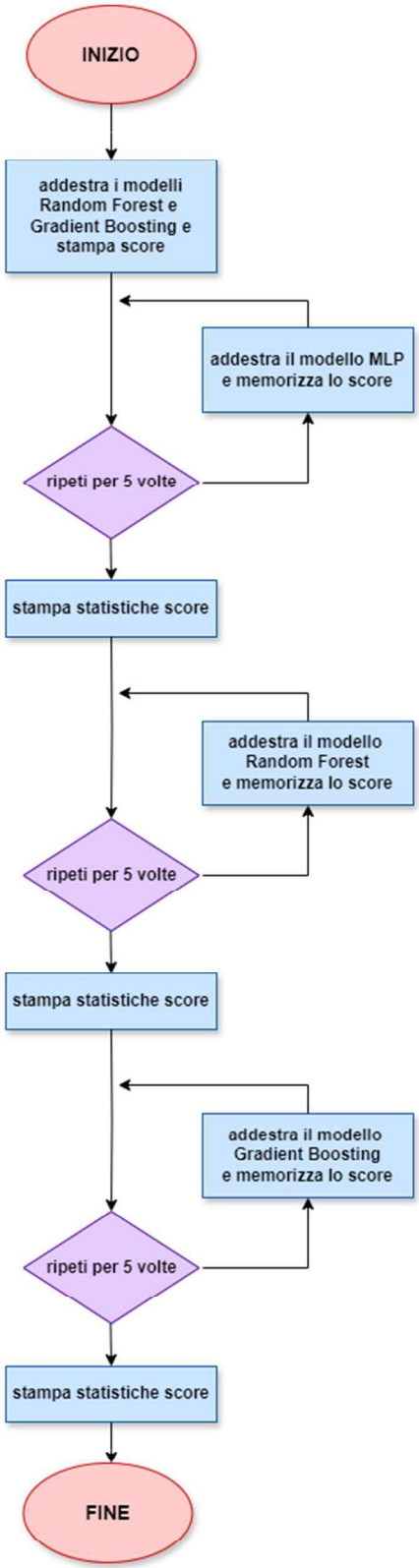

Figura 7. Schema generale operativo del Report 3.

Per la parte relativa al codice si rimanda al secondo paragrafo dell'Appendice A.

I risultati ottenuti sono in linea con le previsioni, per entrambi i metodi si ottiene un valore di score elevato (come riportato in figura 8). Questo, quindi, conferma il fatto che usando un metodo basato su alberi binari per questo dataset, che risulta sbilanciato, si ha un'accuratezza maggiore sulle previsioni rispetto al *MLP,* vedi secondo report.

Un ulteriore considerazione che si può fare è l'analisi statistica dello score dei tre metodi finora utilizzati. Si può pensare di effettuare un numero prefissato di iterazioni per ogni metodo, calcolare l'accuratezza delle previsioni memorizzarla e quindi calcolarne le statistiche, tra cui le media e deviazione standard. Questo test viene quindi integrato andando ulteriormente ad analizzare quindi gli algoritmi di classificazione *Multilayer Perceptron Classifier*, *Random Forest* e *Gradient Boosting*. Per ogni algoritmo si effettuano *cinque iterazioni* memorizzando gli score e calcolando a termine delle iterazioni le statistiche.

Vengono riportati nelle figure 9, 10 e 11 i risultati dell'esecuzione del codice. Come si può notare i due metodi basati su alberi decisionali hanno mediamente uno score più elevato rispetto al *MLP*. Per quanto riguarda la deviazione standard, tutti gli algoritmi hanno un valore basso (dallo 0,49% del *MLP* allo 0,0013% del *Random forest*).

In linea con le aspettative possiamo concludere che i metodi basati su alberi decisionali hanno risultati, in termini di accuratezza delle previsioni, migliori e quindi sono preferibili al *Multi-layer Perceptron.* Attraverso questa prova in particolare, lo score del *Random forest* oltre ad avere una deviazione standard di due ordini di grandezza inferiore alle altre, ha anche un valor medio molto elevato e vicino al valore massimo; quindi, quest'ultimo risulta, per ora, preferibile rispetto agli altri modelli.

Figura 8. Risultati di un'esecuzione del random forest e gradient boosting per il dataset totale.

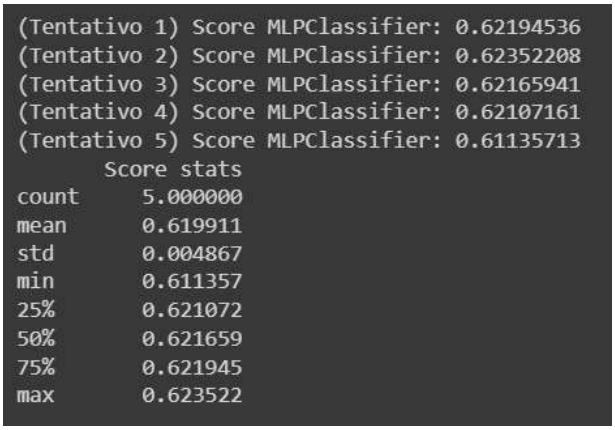

Figura 9. Statistiche dello score di MLP calcolate su cinque iterazioni.

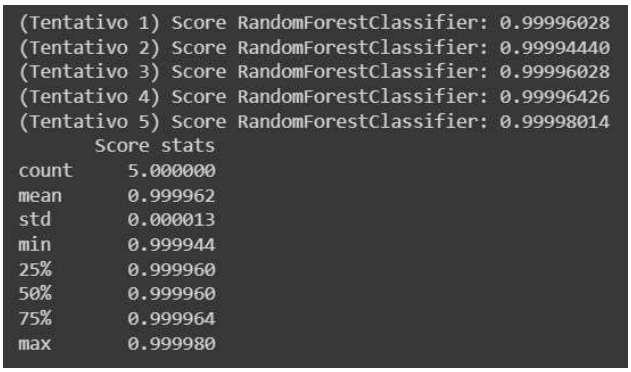

Figura 10. Statistiche dello score di Random Forest calcolate su cinque iterazioni.

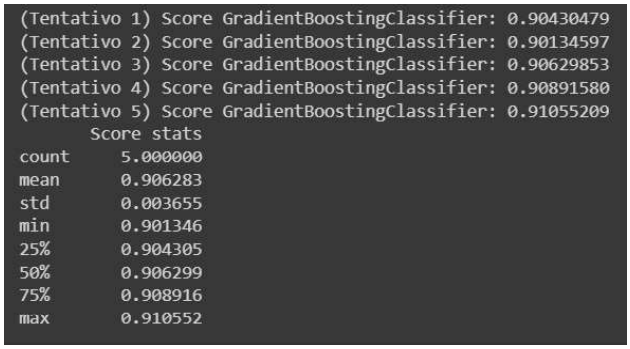

Figura 11. Statistiche dello score di Gradient Boosting calcolate su cinque iterazioni.

4.2.4 Report 4: Analisi features importances del Random Forest e Gradient Boosting.

Nel seguente report si visualizzano e analizzano le feature importances dei modelli *Random Forest* e *Gradient Boosting* per il totale dataset considerado come sempre variabile target l'*Arousal*. In figura 12 si riporta uno schema generale di questo test.

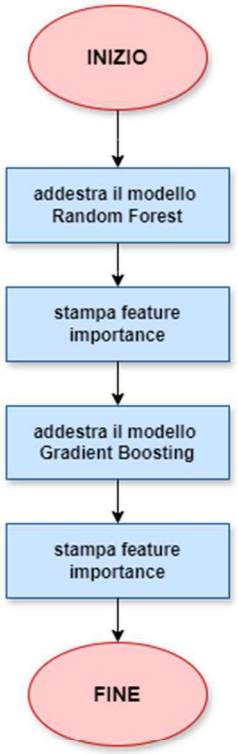

Figura 12. Schema generale operativo del Report 4.

Andando a ricavare le *features importance* del dataset in esame, sia per il Random forest che per il *Gradient boosting*, e mostrandole graficamente otteniamo i risultati figure 13 e 14.

Analizzando tali risultati si conclude che le *features importance* hanno lo stesso andamento sia che per il *Gradient boosting* sia che per il *Random forest*. I valori *dell'Empatica E4 Wristband* sono i valori che hanno avuto più importanza per entrambi i modelli nel prevedere la variabile target, cioè l'*Arousal,* nello specifico i valori con più alta importanza delle features sono quelli relativi alla temperatura, all'attività elettrodermica (EDA) e all'*Interbeat interval* (IBI). I valori del *Polar H7 Bluetooth Heart Rate Sensor* e del *NeuroSky MindWave Headsets* sono tutti prossimi allo 0, il che ci suggerisce che l'uso e i dati prodotti da tali dispositivi per la variabile target non porta ad alcun miglioramento del punto di vista di questi modelli di classificazione. Un'altra osservazione su quest'ultimi dispositivi la possiamo fare tenendo conto che, come riportato nell'articolo sul dataset K-EmoCon, parte dei dati relativi alle variabili *Attention* e *Meditation* sono mancanti dovuto ad un equipaggiamento in modo errato del dispositivo *NeuroSky MindWave Headsets*.

In figura 15 si riportano le statistiche di tali variabili e la quantità di valori pari a zero. Osserviamo che circa un 19% dei dati, sia per *Attention* che *Meditation* ha valore nullo e quindi questo potrebbe essere un motivo per cui tali variabili hanno un'importanza molto minore rispetto a quelle dell'*Empatica E4 Wristband*

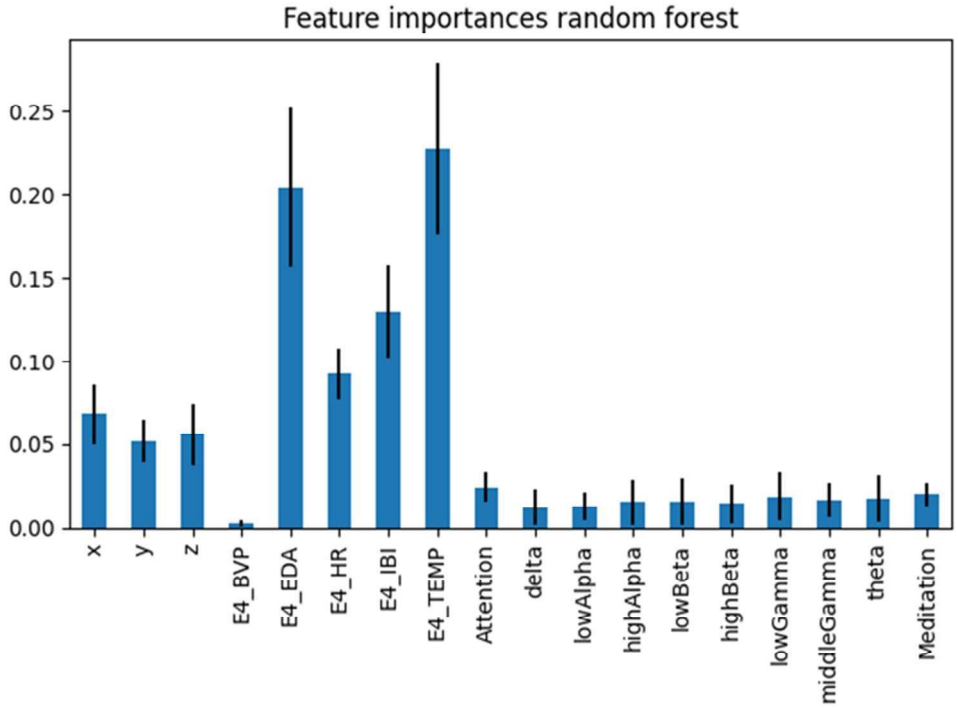

Figura 13. Feature importances del random forest per la variabile Arousal.

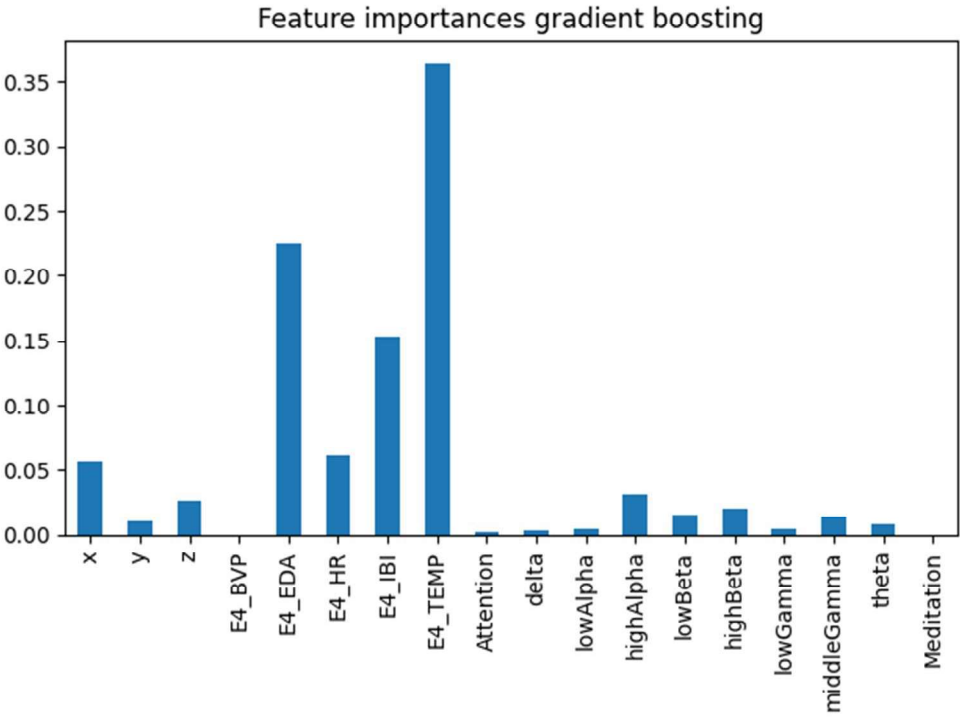

Figura 14. Feature importances del gradient boosting per la variabile Arousal.

```
          Attention    Meditation
count  1.007154e+06  1.007154e+06
mean   3.836415e+01  4.290262e+01
std    3.840947e+01  3.506953e+01
min    0.000000e+00  0.000000e+00
25%    2.059713e+01  2.697832e+01
50%    4.024273e+01  4.670365e+01
75%    5.357894e+01  5.967240e+01
max    9.698960e+02  7.888080e+02
Valori pari a 0 per la variabile Meditation: 187800 su 1007154 cioè 0.18647
Valori pari a 0 per la variabile Attention: 187800 su 1007154 cioè 0.18647
```

Figura 15. Si riportano le statistiche sui valori pari a zero delle caratteristiche Attention e Meditation.

4.2.5 Report 5: Analisi statistiche su subset di caratteristiche dei dispositivi E4 e H7

Considerato i risultati delle *features importance* dei vari modelli è interessante analizzare quale sia lo l'accuratezza delle previsioni di entrambi i modelli, *Random forest* e *Gradient boosting*, nella casistica in cui si utilizzi prima i dati proveniente dal dispositivo *Empatica E4 Wristband* poi quelli provenienti dal *Polar H7 Bluetooth Heart Rate Sensor* e del *NeuroSky MindWave Headsets*. Procederemo quindi prima ad analizzare il modello Random Forest, poi il modello Gradient Boosting.

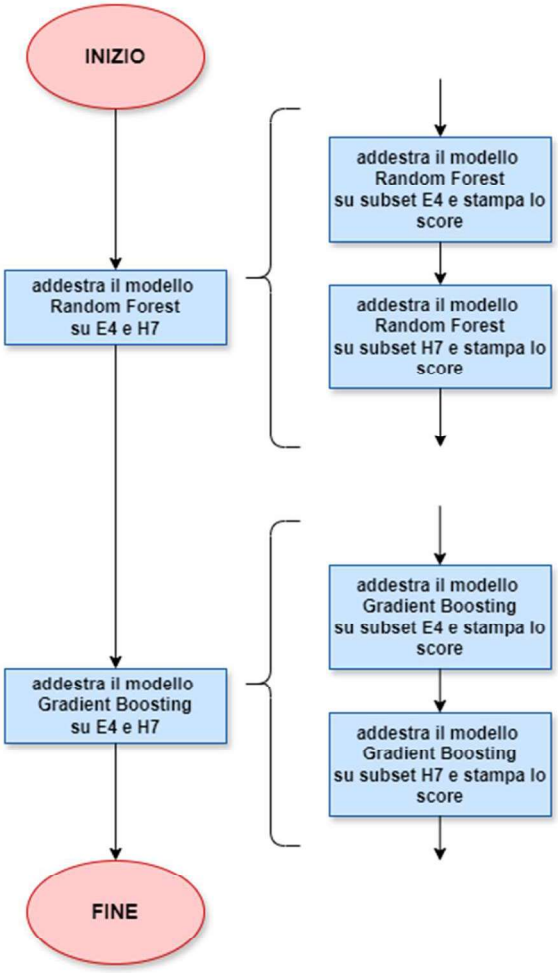

Figura 16. Schema Riassuntivo del Report 5.

Nell'analisi relativa al Random forest si effettuano due prove, una prima tenendo conto la parte del dataset che contiene i dati provenienti dall'E4, vale a dire *accelerazione x, y, z, E4_BVP, E4_EDA, E4_HR, E4_IBI, E4_TEMP*, un'altra tenendo conto dei dati provenienti dagli altri due dispositivi, cioè *Attention, delta, lowAlpha, highAlpha, lowBeta, highBeta, lowGamma, middleGamma, theta, Meditation*. Per entrambi i subset si effettuano cinque iterazioni addestrando il modello e calcolandone lo score, dopodiché si mostrano le statistiche dello score ottenuto, come svolto nel precedente report di analisi dello score. In figura 16 si riporta uno schema generale per chiarire meglio come è stato impostato questo test.

Analizzando tali risultati, riportati nelle figure 17 e 18, si nota che per entrambi i set si ha un'accuratezza dello score simile (un *99,99%* contro un *99, 47%*), seppure quella relativa al dispositivo *E4* risulta più elevata. La maggiore differenza riscontrata è relativa al tempo di esecuzione; infatti, nel primo esempio il tempo di esecuzione totale del programma risulta di *16 minuti* mentre nel secondo caso è pari ad *un'ora e 6 minuti*. Andando a considerare i tempi di esecuzione di ogni singola iterazione, calcolati approssimativamente dividendo il tempo totale per il numero di iterazioni, otteniamo che nel primo caso è di circa *3 minuti* ad iterazione e nel secondo di circa *13 minuti* per iterazione, circa quattro volte più elevato. Quindi si conclude che l'alto grado di accuratezza sia ottenuto "pagando" in termini di tempo di esecuzione, che si suppone più elevato nel secondo caso dovuto al fatto che il modello trova difficoltà ad addestrarsi sui dati del *Neurosky* e del *H7*. Questo risulta coerente con i risultati ottenuti in precedenza sull'importanza delle features, i quali mostrano che per i dati dell'*E4* si ha un'importanza maggiore e che quindi il modello li utilizza maggiormente per prevedere in modo più efficiente la variabile target.

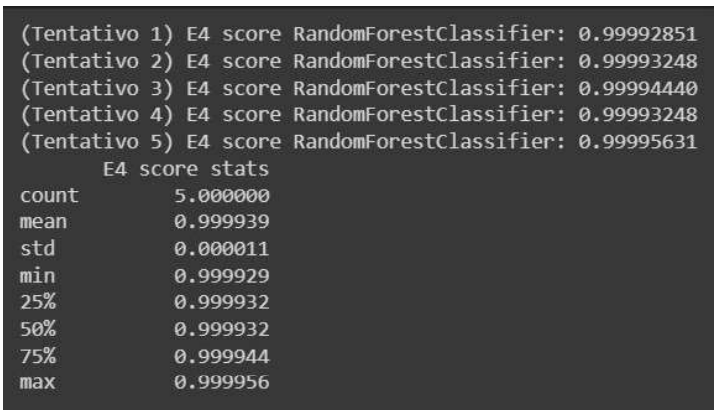

Figura 17. Risultati statistici modello random forest su subset relativo al dispositivo E4.

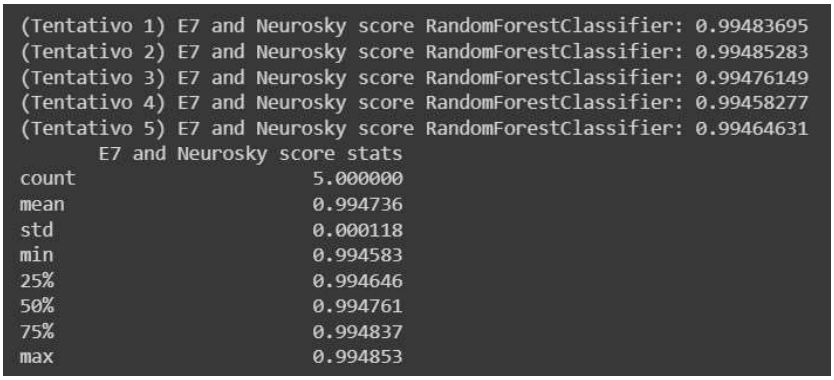

Figura 18. Risultati statistici modello random forest su subset relativo al dispositivo H7.

Come nella precedente analisi si applica adesso il modello *Gradient boosting* prima sul set relativo al *E4*, poi sul set relativo al *Neurolink e H7*. Si analizzano poi i dati statistici dell'accuratezza della previsione ottenuti iterando per cinque volte il modello e calcolandone lo score.

Analizzando tali risultati, riportati nelle figure 19 e 20, si osserva subito che, a differenza del *Random forest*, si ha un'importante degradazione dello score tra i due subset; infatti, si passa da circa un *89%* ad un *62%*. Inoltre, in termini di esecuzione, nel primo caso abbiamo un tempo di esecuzione totale pari a *22 minuti* mentre nel secondo pari a *40 minuti*. Calcolando in modo approssimato come fatto in precedenza di ottiene un tempo di esecuzione per iterazione di circa *quattro minuti* nel primo caso e circa *otto minuti* nel secondo, ovvero il doppio. Quindi utilizzando il *Gradient boosting* utilizzando il set di dati dei dispositivi *H7* e del *NeuroSky* oltre ad avere una sostanziale perdita in accuratezza delle previsioni si ha anche un tempo di esecuzione maggiore rispetto a quando si utilizza i dati dell'*E4*. Anche in questo caso i risultati sono coerenti con l'analisi delle *features importance* del *Gradient boosting*, le quali, nonostante la loro tendenza rimanga simile alle *features importance* del *Random forest*, presentano un maggiore divario tra l'importanza delle caratteristiche del dispositivo *E4* e quella degli altri due dispositivi.

```
(Tentativo 1) E4 score GradientBoostingClassifier: 0.89272764
(Tentativo 2) E4 score GradientBoostingClassifier: 0.89815282
(Tentativo 3) E4 score GradientBoostingClassifier: 0.89088086
(Tentativo 4) E4 score GradientBoostingClassifier: 0.89277530
(Tentativo 5) E4 score GradientBoostingClassifier: 0.89860955
       E4 score stats
count        5.000000
mean         0.894629
std          0.003513
min          0.890881
25%          0.892728
50%          0.892775
75%          0.898153
max          0.898610
```

Figura 19. Risultati statistici modello gadient boosting su subset relativo al dispositivo E4.

```
(Tentativo 1) E7 and Neurosky score GradientBoostingClassifier: 0.62282705
(Tentativo 2) E7 and Neurosky score GradientBoostingClassifier: 0.62315669
(Tentativo 3) E7 and Neurosky score GradientBoostingClassifier: 0.62634587
(Tentativo 4) E7 and Neurosky score GradientBoostingClassifier: 0.62461823
(Tentativo 5) E7 and Neurosky score GradientBoostingClassifier: 0.62557538
       E7 and Neurosky score stats
count                    5.000000
mean                     0.624505
std                      0.001515
min                      0.622827
25%                      0.623157
50%                      0.624618
75%                      0.625575
max                      0.626346
```

Figura 20. Risultati statistici modello gadient boosting su subset relativo al dispositivo E7.

Entrambi i modelli presentano vantaggi dall'esecuzione dei modelli stessi sui dati relativi all'*Empatica E4 Wristband* rispetto i dati provenienti dal *Polar H7 Bluetooth Heart Rate Sensor* e del *NeuroSky MindWave Headsets.* Per il *Random forest* il guadagno principale si ha sul tempo di addestramento del modello, mentre per il *Gradient boosting* si ha un beneficio sia in termini di score che in termini di tempo. Inoltre, possiamo analizzare la differenza tra lo score utilizzando il dataset totale, vedi terzo report, e lo score utilizzando il subset relativo all'*E4*. In entrambi i casi lo score rimane molto simile, solo nel caso del *Gradient boosting* varia negativamente di un punto percentuale. In conclusione, possiamo dire che addestrando i modelli con i dati dell'*E4* rispetto a tutto il dataset non si hanno rilevanti perdite di accuratezza.

4.2.6 Report 6: Analisi approfondita sulle features più importanti del dataset

Dall'ultima analisi effettuata sullo score di entrambi i modelli usando due dataset di features rispettivamente più importanti e meno importanti, si evidenzia che uno studio più dettagliato sullo score, sulle *feature importance* e sui tempi di esecuzione possa aiutare a scegliere il modello più adatto con le caratteristiche più adatte per il dataset *K-EmoCon*. Una possibilità è analizzare gli

score e il tempo di esecuzione, quest'ultimo dipende principalmente dalla fase di training del modello, usando dataset diversi considerano in ogni prova un insieme di features sempre più grande prendendole in esame in base alla importanza.

Nel seguente report si calcola prima l'importanza delle features, basandosi su ciò si parte prima da un piccolo subset composto dalle *tre caratteristiche più importanti* e si calcola lo score e i tempi di esecuzione del modello, dopodiché si ripetono i calcoli considerando un subset sempre più grande fino al dataset totale. I risultati ottenuti si mostrano graficamente, ottenendo così delle curve che indicano lo score e i tempi di esecuzione al variare delle *n* caratteristiche più importanti. Tutte le misurazioni di score e tempi di esecuzione si ottengono facendo una media in modo da aumentare l'affidabilità dei risultati ottenuti. È possibile riassumere questo report in tre fasi:

1. *Calcolo e ordinamento delle feature importance*. In questa fase si effettuano cinque iterazioni dove si addestra il modello scelto e si ricavano i valori delle feature importance e a termine delle iterazioni si calcolano i valori medi per ogni caratteristica. Dopodiché per questioni di praticità di ordinano per importanza decrescente. Nella figura B.11 dell'Appendice B si riporta il risultato dell'esecuzione di questa fase.
2. *Calcolo delle statistiche del modello*. Si parte considerando un subset delle tre features più importanti per un certo modello, si effettua quindi il training e il testing e si memorizzano il tempo di esecuzione e lo score, si ripete questo ciclo aumentando ogni volta il subset considerando la feature più importante che non è già parte del set stesso. Il tempo di esecuzione e lo score è ottenuto facendo, all'interno dell'iterazione stessa, altre cinque iterazioni. Ottenendo quindi cinque tempi di esecuzione e cinque score, come valori di riferimento si prendono la media di tali misurazioni per lo score e per il tempo di esecuzione.
3. *Visualizzazione dei risultati*. Si ottiene quindi una serie di score e tempi che sono relativi al numero di features utilizzate. Si mostrano tali risultati graficamente in modo da analizzare contemporaneamente sia i tempi di esecuzione sia gli score ottenuti sia l'andamento di tali valori in funzione del numero delle caratteristiche prese in considerazione.

Si riporta in figura 21 uno schema generico che riassume ogni passo di questo test, è stato schematizzato solo la parte del Random Forest, quella relativa al Gradient Boosting è duale.

Il codice relativo a questo report è stato anch'esso suddiviso nelle fasi appena descritte, sia perché l'esecuzione del codice totale richiede un tempo molto elevato che risulta sconveniente in fase di debugging sia perché il codice del report è stato eseguito con *Google Colab* il quale ha delle limitazioni tra cui il tempo di esecuzione.

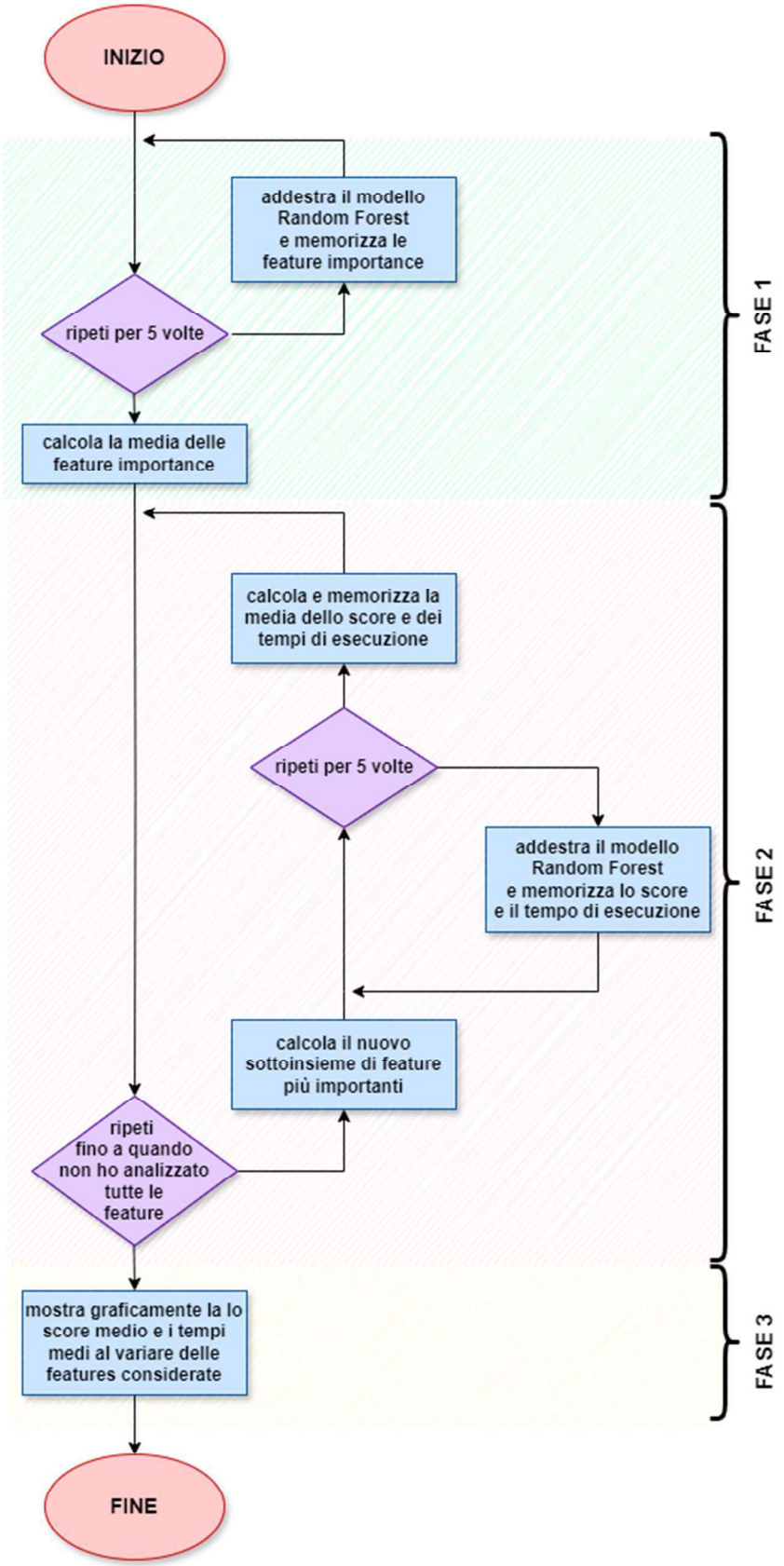

Figura 21. Schema riassuntivo relativo al solo modello random forest del Report 6. L'implementazione del gradient boosting è duale.

Si mostrano adesso, con le figure 22 e 23, i risultati ottenuti graficamente dall'esecuzione del codice di questo report.

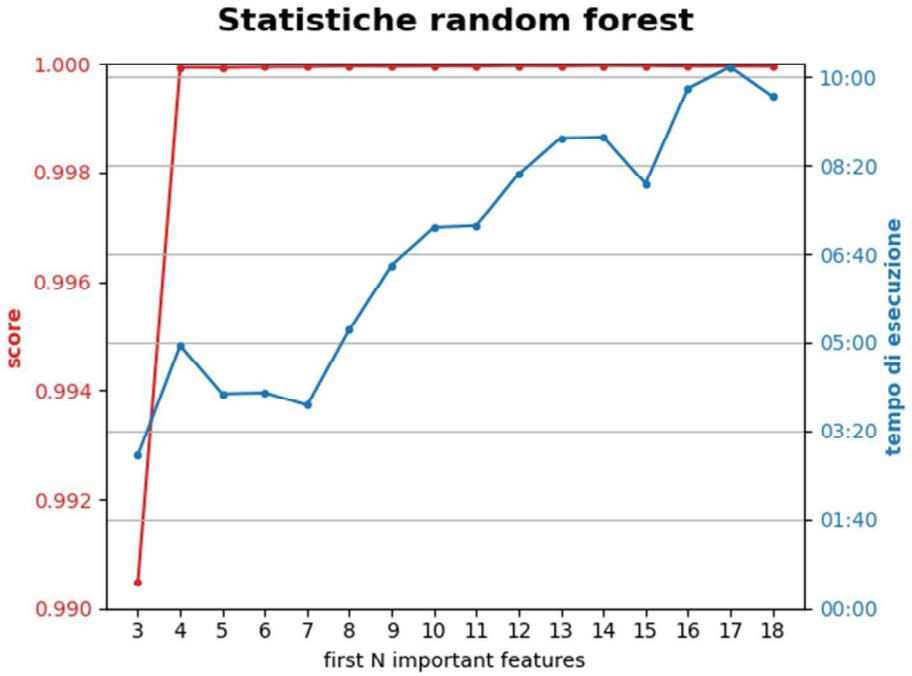

Figura 22. Grafico dei risultati dello score e del tempo di esecuzione in relazione al numero delle features utilizzate col modello random forest.

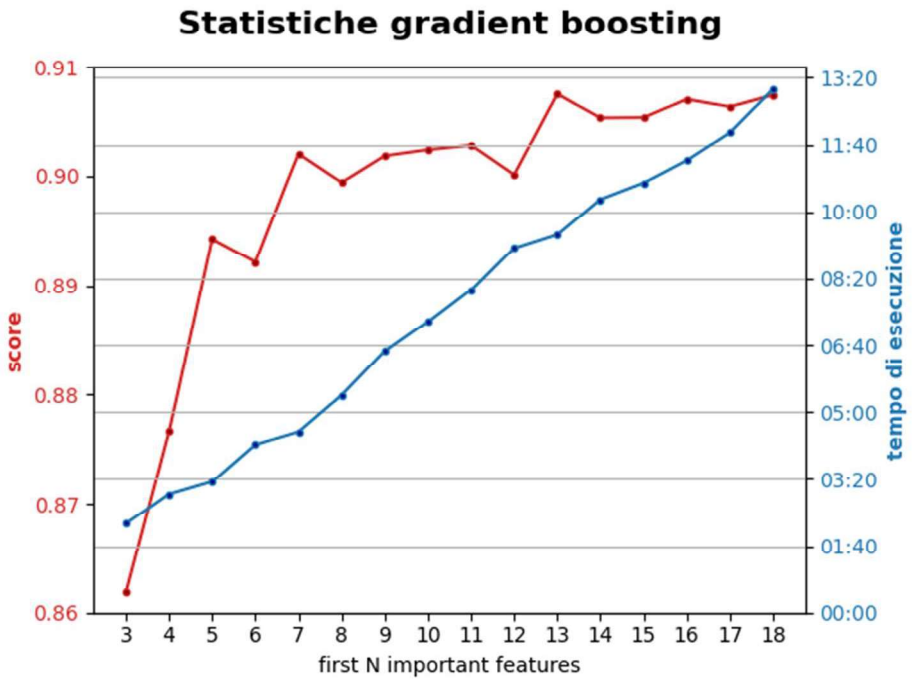

Figura 23. Grafico dei risultati dello score e del tempo di esecuzione in relazione al numero delle features utilizzate col modello gradient boosting.

Analizzando le *statistiche del random forest* osserviamo che lo score resta molto elevato, superiore al *99%*, qualsiasi sia il numero di features che abbiamo valutato (anche con le sole tre feature più importanti). Il tempo di esecuzione presenta una crescita sommariamente lineare partendo da circa *tre minuti*, considerando tre features, arrivando a circa *dieci minuti*, considerando tutto il dataset.

Analizzando le *statistiche del gradient boosting* osserviamo che, lo score parte da un minimo di poco più dell'*86%*, considerando tre features, arrivando a poco meno del *91%*, considerando il dataset totale. Per quanto riguarda i tempi di esecuzione si ha una crescita lineare da circa *2 minuti*, quando si considerano 3 caratteristiche, a circa *13 minuti*, quando si considera tutto il dataset.

Le figure B.1 e B.2 dell'Appendice B mostrano i dati tabellari dei risultati di questo report.

Analizzando lo score dei due modelli abbiamo che entrambi hanno valori elevati ma è evidente che il *random forest* rispetto al *gradient boosting* risulta di circa dieci punti percentuali maggiore.

Analizzando i tempi di esecuzione, vale a dire i tempi di training e testing, per entrambi i modelli si ha una crescita lineare all'aumentare delle features considerate. Il *gradient boosting* ha mediamente un tempo di esecuzione maggiore e presenta inoltre una maggiore deviazione standard rispetto al *random forest*.

Analizzando quale sia la configurazione ottimale per entrambi i modelli si osserva che all'aumentare delle features entrambi hanno delle variazioni in score poco significative, il *random forest* resta molto vicino allo *0.99* mentre il *gradient boosting* passa dallo *0.861* allo *0.907*. Quindi a fronte di variazioni poco significative nello score conviene scegliere il numero delle features con tempo di esecuzione minore, infatti le variazioni di quest'ultimo sono molto più accentuate. Per il *random forest* si può scegliere *N pari a 3* con tempo di esecuzione pari a *2 minuti e 52 secondi*, mentre per il *gradient boosting* se si vuole un valore di accuratezza vicino alla media si può scegliere *N pari a 5* con score pari a *0.894230* e tempo di esecuzione pari a *3 minuti e 16 secondi*.
Scegliendo come algoritmo il *random forest* per questo tipo di dataset, oltre ad avere dei tempi di esecuzione leggermente minori e una minore deviazione standard, abbiamo un significante guadagno dal punto di vista dell'accuratezza delle previsioni; infatti, se si usano le configurazioni appena determinate si ha che a parità di tempo di esecuzione lo score è maggiore di dieci punti percentuali rispetto al *gradient boosting*.

4.2.7 Report 7: Analisi approfondita sulle features meno importanti del dataset

Con questo report di vuole applicare la controprova del report precedente ovvero consiste nel mostrare l'andamento dello score e del tempo di esecuzione dei due modelli esaminati finora andando a togliere dal dataset totale una feature per volta, nello specifico la feature più importante appartenente al dataset in esame. Per quanto riguarda lo score, ci si aspetta che esso diminuisca in quanto si escludono ogni volta feature più importanti, per quanto riguarda i tempi di esecuzione, ci si aspetta che anch'essi diminuiscono perché diminuisce il numero di features con le quali si addestra il modello. Si inizia analizzando il dataset totale senza la feature più importante, che per entrambi i modelli corrisponde a *E4_TEMP* (temperatura corporea rilevata del dispositivo *E4 Wristband*).

Il test si svolge in modo duale al precedente e si divide nelle seguenti fasi:

1. *Calcolo e ordinamento delle feature importance*. Per il valore delle feature importance si utilizzano le due tabelle (una per il *random forest* e una per il *gradient boosting*) utilizzate nel test precedente, vedi fase 1 del precedente report.
2. *Calcolo delle statistiche del modello*. Si parte considerando il dataset totale senza la feature *E4_TEMP*, si calcolano la media dello score e la media del tempo di esecuzione, cioè la media del tempo necessario ad addestrare e testare il modello. Questo processo si ripete andando a togliere ad ogni iterazione la feature più rilevante appartenente al sottoinsieme che si sta valutando fino a quando non si raggiunge un insieme dato dalle *tre* caratteristiche meno importanti.
3. *Visualizzazione dei risultati*. Come nel precedente test si mostrano graficamente i tempi e gli score trovati evidenziando l'andamento al variare del numero delle caratteristiche meno importanti del dataset prese in considerazione.

La *fase 1* consiste nel leggere i file sull'importanza delle feature prodotti nel report precedente. La *fase 2* è similare alla stessa fase del report precedente, si ha una variazione iniziale dove, adesso, si eliminano le caratteristiche dal dataset totale in ordine di importanza. La *fase 3* consiste nel mostrare graficamente i risultati usando lo stesso procedimento del report precedente, andando a modificare l'*asse x*, al quale adesso si associano le *n* caratteristiche meno importanti.

In figura 24 si riporta in uno schema riassuntivo, come nel precedente report, l'implementazione di questo test della parte relativa al *random forest*.

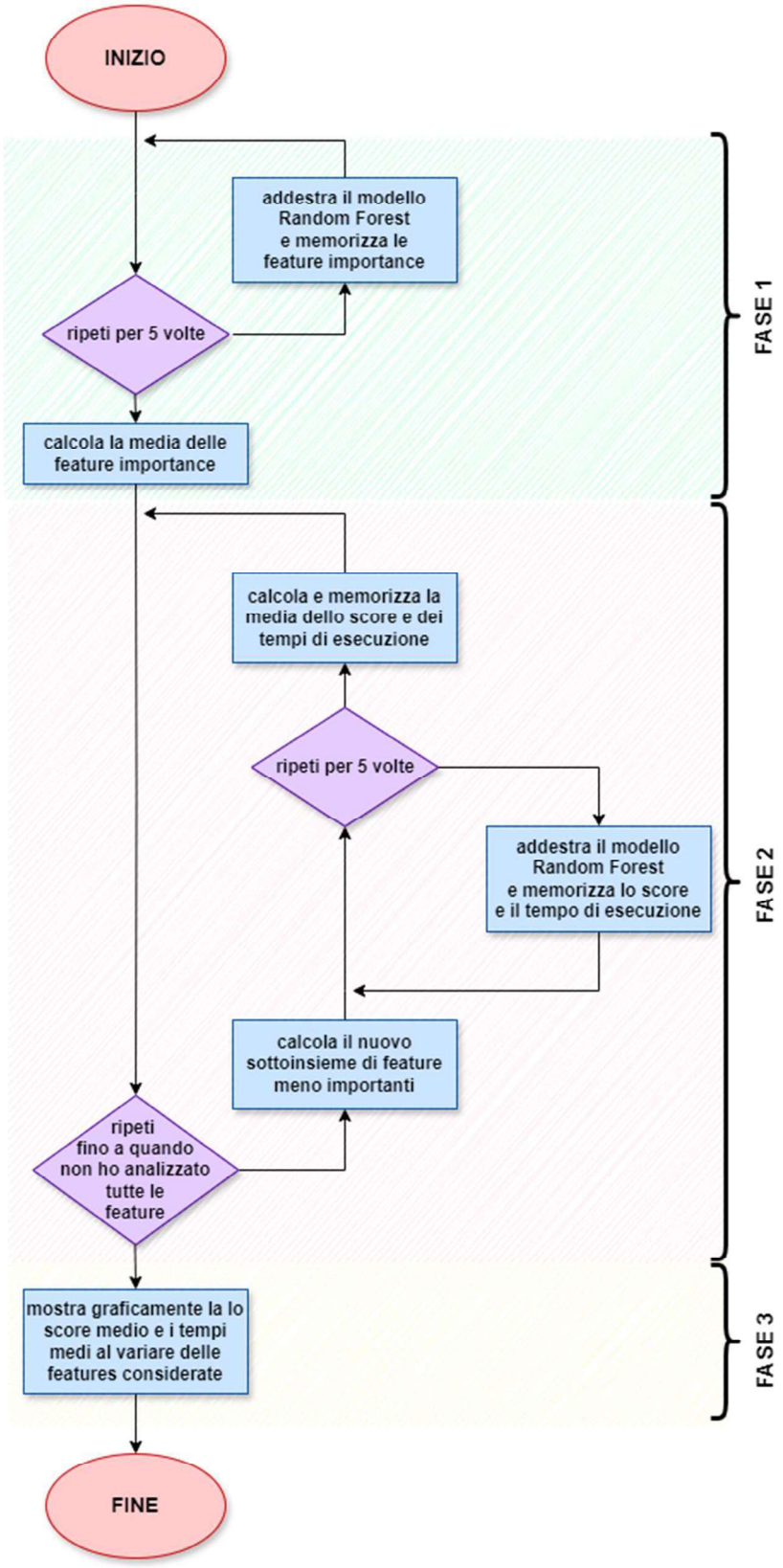

Figura 24. Schema riassuntivo dell'organizzazione del report 7 relativo al random forest. L'implementazione del gradient boosting è duale.

Si mostrano adesso, nelle figure 25 e 26, i risultati ottenuti graficamente dall'esecuzione del codice di questo report.

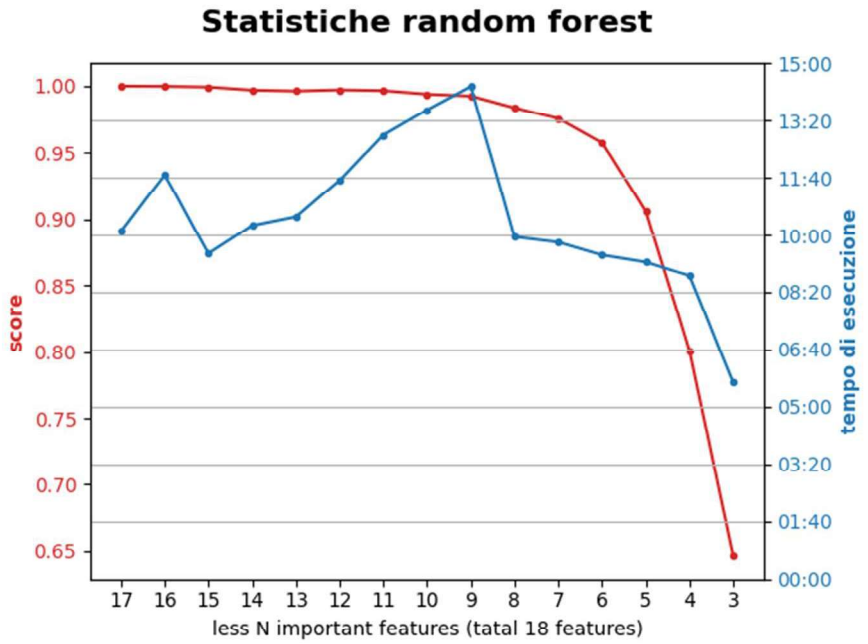

Figura 25. Grafico dei risultati dello score e del tempo di esecuzione in relazione al numero delle features utilizzate col modello random forest.

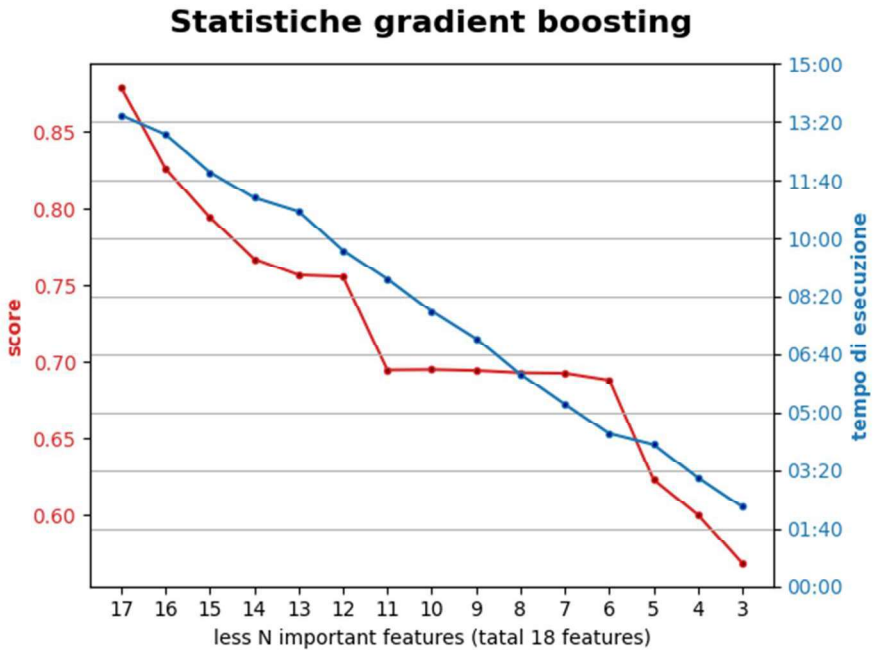

Figura 26. Grafico dei risultati dello score e del tempo di esecuzione in relazione al numero delle features utilizzate col modello gradient boosting.

Analizzando le statistiche del *random forest* si osserva che lo score rimane stabile e superiore a *0,99* per un numero di caratteristiche maggiore di *nove*, dopodiché si ha un calo sull'accuratezza

delle previsioni sostanziale. Per quanto riguarda i tempi di esecuzione, si nota due andamenti diversi; una crescita lineare, per un numero di caratteristiche maggiore di *nove*, seguita da una decrescita in linea con il calo dell'accuratezza delle previsioni. Si può dire che questo modello fino a nove caratteristiche meno importanti riesce a mantenere un livello eccellente di score contro un aumento del tempo medio di esecuzione che cresce al di sopra dei *dieci minuti*.

Analizzando i risultati del *gradient boosting*, si nota che, in linea con le previsioni, si ha una decrescita lineare dello score e dei tempi medi di esecuzione in relazione al numero delle features considerate.

Le figure B.3 e B.4 dell'Appendice B mostrano i dati tabellari dei risultati di questo report.

Con questo test si mette in evidenzia il degrado delle prestazioni di entrambi i modelli, più precisamente dello score. Mentre per il *gradient boosting* il peggioramento dell'accuratezza è più chiaro ed evidente, per il *random forest* lo si ha a partire da un sottoinsieme che considera un certo numero di caratteristiche meno importanti (*nove*). Per quest'ultimo modello il mantenimento dello score elevato, come evidenziato dai risultati del test, lo si ha aumentando il tempo di esecuzione per ogni step, dopodiché si ha un drastico peggioramento in termini di score ma ottenendo tempi di addestramento minori. Confrontando questi risultati con quelli del test precedente (figure 22 e 23) si evidenzia che, considerando circa lo stesso valore di score si hanno tempi di addestramento minori andando ad utilizzare dataset contenenti solo le features più rilevanti, dualmente a parità di tempi di esecuzione, se si utilizzano dataset del precedente report si ha un accuratezza delle previsioni per entrambi i modelli superiore.

CONCLUSIONI

Attraverso le sperimentazioni siamo partiti applicando il Multilayer Perceptron prima ad un sottoinsieme, relativo al soggetto 25, poi alla totalità del dataset *K-EmoCon*, concludendo che un approccio basato su tale modello di Machine Learning per questo preciso dataset non fosse propriamente adatto. Questa conclusione è in linea con quanto ci si aspettava essendo il MLP per sua natura molto sensibile allo sbilanciamento delle classi. Quindi le sperimentazioni sono continuate con la famiglia degli algoritmi basati su alberi decisionali, ottenendo fin da subito risultati migliori, vedi risultati del terzo report, se si considera il dataset nella sua interezza.

Si sono analizzati approfonditamente gli algoritmi del Random Forest e Gradient Boosting, andando a valutare le feature importance. Dalle quali è risultato che i segnali catturati dal dispositivo *Empatica E4 Wristband*, rispetto a quelli catturati dai dispositivi *Polar H7 Bluetooth Heart Rate Sensor* e *NeuroSky MindWave Headsets*, abbiano una maggiore rilevanza nel processo decisionale di entrambi i modelli; infatti, dai risultati ottenuti dal quinto report si notano differenze sostanziali nell'accuratezza delle previsioni effettuate.

Si è elaborata un'analisi ancora più approfondita, con il sesto e il settimo report, non ci si concentra più sui dispositivi ma sulle singole caratteristiche, andando ad analizzare contemporaneamente l'andamento dello score medio e del tempo di addestramento medio di sottoinsiemi di dataset di feature più importanti in modo crescente, e come controprova sottoinsiemi di dataset sempre meno rilevanti, ovvero composti da sole feature meno rilevanti. Andando a concludere che per avere risultati ottimi in tempi di addestramento ragionevoli basta un piccolo sottoinsieme di caratteristiche più rilevanti.

Da questi risultati possiamo concludere che i modelli *Gradient Boosting* e *Random Forest* risultano entrambi adatti con buone performance. Per quanto riguarda questo dataset e per la variabile target *Arousal* il *Random Forest* risulta leggermente più performante rispetto a *Gradient Boosting*. Inoltre, basta avere un piccolo numero, *tre*, di features più importanti, vedi sesto report, per avere sostanziali miglioramenti in tempo di addestramento e sempre un ottimo grado di accuratezza delle previsioni.

Possiamo trarre altre conclusioni, ovvero che l'*Empatica E4 Wristband* può essere, da solo, un buon dispositivo per il campionamento di segnali fisiologici da poter usare in nuovi dataset della stessa tipologia e metodologia di raccolta di informazioni. Infatti, è pensabile che possa essere utilizzato singolarmente, sotto un aspetto economico, consentendo di contenere le spese per più dispositivi e sotto un aspetto psicologico mettendo in uno stato più naturale i soggetti stessi.

Si potrebbe estendere e dettagliare questo elaborato addestrando e analizzando gli stessi modelli di Machine Learning non solo per la variabile *Arousal*, ma per tutte le variabili relative alle altre tipologie di valutazioni del dataset. Ci si può aspettare ragionevolmente che da tale analisi si raggiungano grossomodo gli stessi risultati principalmente perché tutte le variabili target risultano con lo stesso elevato sbilanciamento delle classi. Tuttavia, può essere un modo interessante per confermare l'importanza dei segnali fisiologici del dispositivo *Empatica E4 Wristband*, ma anche se le caratteristiche derivanti dagli strumenti assumono una rilevanza diversa. Un'altra interessante analisi che può nascere da questa ricerca è valutare se le caratteristiche dei segnali dei tre dispositivi utilizzati dallo studio *K-EmoCon* assumano la stessa importanza anche in dataset senza sbilanciamento delle classi, anche se questo richiederebbe la raccolta di nuovi dati, utilizzando gli stessi dispositivi o simili, e la stesura di una nuova tipologia dataset.

APPENDICE A: manuale d'uso e codice del software finale

Il software finale di questo elaborato offre la possibilità di effettuare l'analisi dei due modelli Random Forest e Gradient Boosting per il dataset in esame. Come già anticipato nel terzo capitolo, permette ad un generico Machine Learning Engineer di esaminare le feature importance e i grafici del tempo di esecuzione e dello score, come nei *"Report 6"* e *"Report 7"* del quarto capitolo, dopodiché consente allo stesso di scegliere il modello e i parametri migliori. Consente, inoltre, ad un altro soggetto, il Decision Maker, di approvare le spiegazioni e il modello proposto e fare ulteriori previsioni per nuovi dati con il modello appena validato. Si riporta successivamente il manuale d'uso e il codice finale.

A.1 Manuale d'uso

Innanzitutto, sia il codice e che il dataset (*EMOCON_VALENCE.CSV*) sono reperibili su *GitHub* (*www.github.com/luca7351/software_finale*). Il codice è stato scritto in *Python* ed è stato realizzato ed eseguito con *Google Colab*, ovvero una piattaforma che permette l'esecuzione di codice *Python* direttamente dal browser, ma è possibile eseguirlo in qualsiasi ambiente di sviluppo che supporta il linguaggio *Python.*

Nella fase iniziale il software, vedi la figura A.1, chiede di indicare il path della directory contenente il dataset. All'interno della directory verranno salvati anche tutti i file intermedi.

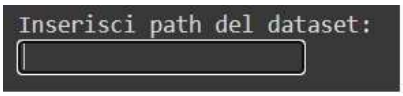

Figura A.1. Screenshot della fase iniziale del software.

Nella fase successiva viene mostrato un menù con il quale è necessario indicare il ruolo dell'utente (*Machine Learning Engineer* oppure *Decision Maker*). In Figura A.2 si riporta lo screenshot di questa fase.

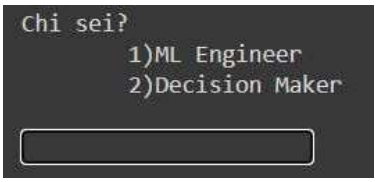

Figura A.2 Screenshot della fase di identificazione del ruolo dell'utente.

Inviando *"1"* o *"ML Engineer"* si accede al menu di tale utente, altrimenti, inviando *"2"* o *"Decision Maker"* si accede all'altro menu.

In figura A.3 si riporta le scelte associate al Machine Learning Engineer, per indicare un comando è necessario inserire il numero associato all'operazione stessa.

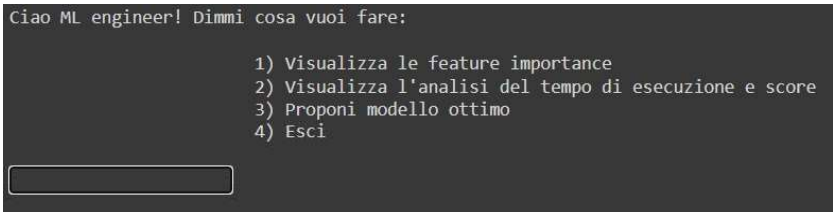

Figura A.3 Screenshot del menu associato al ML engineer.

La scelta *"1"* permette di visualizzare le feature importance del dataset, nello specifico viene mostrata una media dell'importanza calcolata per cinque iterazioni per ciascun modello. Quindi tali valori vengono riportati in due istogrammi, uno per il Random Forest e uno per il Gradient Boosting.

La scelta *"2"* permette di visualizzare l'andamento del tempo di esecuzione e dello score in base al numero di caratteristiche utilizzate, consentendo di specificare se si vogliono analizzare i sottoinsiemi di feature più importanti oppure meno importanti. Ciò equivale all' analisi del sesto report, per subset di feature più importanti, e del settimo report, per subset di feature, meno importanti, vedi quarto capitolo per maggiori chiarimenti. Quindi prima di mostrare i grafici viene chiesta la tipologia di analisi; "prova" per l'analisi per subset più importanti (sesto report), "controprova" per l'analisi per subset meno importanti (settimo report).

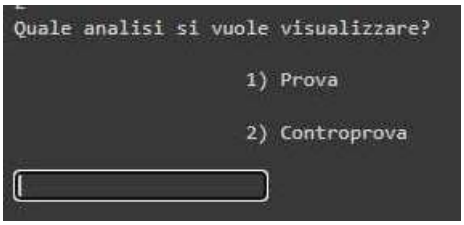

Figura A.4. Menu "secondario" della seconda operazione. Viene chiesto se eseguire l'analisi della prova o della controprova.

La scelta *"3"* permette di indicare, dopo la visualizzazione delle feature importance e dei grafici, il modello scelto e il numero di feature, più importanti, da utilizzare. Quindi prima di questa scelta è necessario avviare le due operazioni precedenti, vedi il paragrafo A.2 per l'implementazione di questi controlli.

In figura A.5 si riporta le scelte associate al Decision Maker, per indicare un comando, come nel caso precedente, è necessario inserire il numero associato all'operazione stessa.

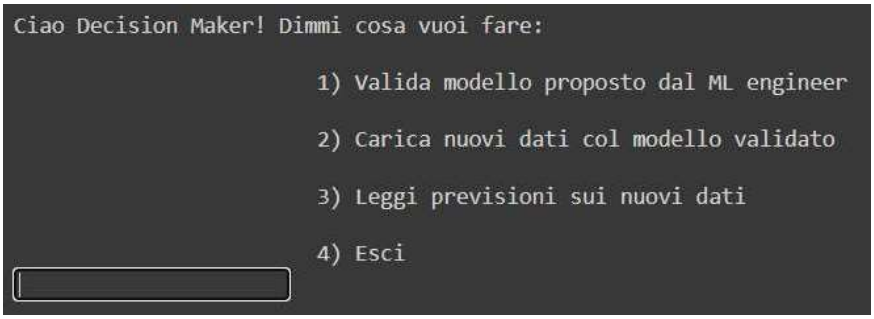

Figura A.5 Screenshot del menu associato al Decision Maker.

La scelta *"1"* consente di validare, se presente, il modello proposto dal ML engineer. Per approfondimenti sull'implementazione di questi controlli si rimanda al paragrafo A.2 contenente il codice totale del software. Nello specifico mostra il modello scelto con il numero di features più importanti scelte, dopodiché è necessario inserire *"y/n"* per validare o rifiutare tale proposta. Nel caso sia validata mostra il nome delle features del modello, in modo da predisporre correttamente gli eventuali nuovi dati attraverso i quali se desidera fare nuove previsioni.

La scelta *"2"* consente di inserire il nome del file (*.csv*) contenente i nuovi dati per fare la previsione. È quindi necessario prima inserire il file nel path del dataset, poi chiamare l'operazione.

La scelta *"3"* consente di utilizzare il modello validato, se esiste, per effettuare la previsione utilizzando i dati contenuti nel file il cui *filename* è stato inserito nell'operazione precedente. È quindi necessario validare un modello e inserire il nome del file prima di iniziare questa operazione. Per informazioni maggiori sull'implementazione di questi controlli si rimanda al paragrafo A.2 contenente il codice del software.

A.2 Codice

Si riporta il codice del software finale che racchiude tutti gli esperimenti di questo elaborato. Si ricorda che lo stesso codice è presente al link sopracitato su GitHub.

```
from numpy import ravel
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
from glob import glob
from pathlib import Path
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
import statistics


# ##############################################################################
#                                 Dati
```

```
# ##############################################################################
# Imposta il path generale da dove ricavare il dataset totale EMOCON_VALENCE.CSV
# e dove riportare i file risultati
# -> file_path()    : ritorna il path di dove si trova il file
# -> get_path()     : ritorna il path del file comprensivo del filename
class Dati:
    def __init__(self, path_gen, name_file):
        if name_file == "":
            self.filename = "EMOCON_VALENCE.CSV"
        else:
            self.filename = name_file
        self.path = path_gen

    def file_path(self):
        return self.path + self.filename

    def get_path(self):
        return self.path


# ##############################################################################
#                                   MetodoML
# ##############################################################################
# Classe utilizzata per addestrare i modelli di ML (Gradient Boosting e Random
# Forest), viene inizializzata fornendogli il dataset già diviso attraverso la
# tecnica 'train test split', viene fornito inoltre il modello che si vuole
# implementare.
# -> set_tipo()         : imposta il tipo del modello di ML (Random Forest o
#                         Gradient Boosting.
# -> train_model()      : addestra il modello con i parametri inseriti
# -> get_type()         : ritorna il modello di ML usato
# -> get_score()        : ritorna lo score del modello addestrato
# -> get_feature_importance()   : ritorna le feature importance del modello
#                                 addestrato.
# -> get_feature_names(): ritorna il nome delle feature del dataset
# -> prediction()       : presi dei dati di ingresso (classe Dati()) mostra le
#                         le previsioni del modello addestrato

class MetodoML:
    def __init__(self, training_d, testing_d, training_l, testing_l, tipo_modello):
        self.training_data = training_d
        self.training_labels = training_l
        self.testing_data = testing_d
        self.testing_labels = testing_l
        self.tipo = tipo_modello
        self.model_g = GradientBoostingClassifier()
        self.model_r = RandomForestClassifier()

    def set_tipo(self, tipo_modello):
        self.tipo = tipo_modello

    def train_model(self):
        if self.tipo == "GradientBoosting":
            self.model_g.fit(self.training_data, ravel(self.training_labels))
        elif self.tipo == "RandomForest":
            self.model_r.fit(self.training_data, ravel(self.training_labels))
        else:
            print("Modello ML inserito non valido.")

    def get_type(self):
        return self.tipo

    def get_score(self):
        if self.tipo == "GradientBoosting":
            model_ml = self.model_g
        elif self.tipo == "RandomForest":
            model_ml = self.model_r
        else:
            print("Modello ML non valido.")
            return
        predicted_l = model_ml.predict(self.testing_data)
        a_s = accuracy_score(ravel(self.testing_labels), predicted_l)
        return a_s
```

```
    def get_feature_importance(self):
        if self.tipo == "GradientBoosting":
            model_ml = self.model_g
        elif self.tipo == "RandomForest":
            model_ml = self.model_r
        else:
            print("Modello ML non valido.")
            return

        importances = model_ml.feature_importances_
        return importances

    def get_feature_names(self):
        features_names = list(self.training_data.columns)
        return features_names

    def prediction(self, data):
        path_file = data.file_path()
        feat_data = read_csv(path_file)
        if self.tipo == "RandomForest":
            pred_labels = self.model_r.predict(feat_data)
        elif self.tipo == "GradientBoosting":
            pred_labels = self.model_g.predict(feat_data)
        else:
            print("Modello ML non previsto.")
            return
        return pred_labels


# ##############################################################################
#                                  FeatImportance
# ##############################################################################
# Elabora le feature importance calcolandone la media per 5 iterazioni e
# mostrandole graficamente, tutto questo per entrambi i modelli di ML.
# -> add_fi ()  : recupera e calcola la media per cinque iterazioni delle
#                 feature importance.
# -> plot_fi()  : mostra graficamente i valori precedentemente calcolati.
# -> get_fi()   : ritorna i valori delle feature importance precedentemente
#                 calcolati.
class FeatImportance:
    def __init__(self, file):
        self.dati = file
        self.fi_grd = None
        self.fi_rnd = None
        self.fi_grd_np = None
        self.fi_rnd_np = None

    def add_fi(self, model_type):
        path_file = self.dati.file_path()
        total_data = read_csv(path_file)
        features_names = list(total_data.iloc[:, 1:len(total_data.columns)])
        # dataframe che conterrà tutti i valori delle cinque iterazioni
        df = pd.DataFrame(columns=features_names)

        path_file = self.dati.file_path()
        total_data = read_csv(path_file)

        # ottengo i nomi delle features
        features_names = list(total_data.iloc[:, 1:len(total_data.columns)].columns)

        i = 0
        while i < 5:
            train_d, test_d, train_l, test_l = train_test_split(
                total_data.iloc[:, 1:len(total_data.columns)],
                total_data.iloc[:, 0])
            model_app = MetodoML(train_d, test_d, train_l, test_l, model_type)
            model_app.train_model()
            feature_importances = model_app.get_feature_importance()

            # creo un DataFrame contenente le caratteristiche dell'iterazione
            model_importances = pd.DataFrame([feature_importances],
                                             columns=features_names)

            # concateno le caratteristiche dell'iterazione al dataframe generale
```

```
            df = pd.concat([df, model_importances], axis=0, ignore_index=True)
            i = i + 1

        # Calcolo media
        df_stat = df.mean()

        df_stat_np = df_stat.to_numpy()
        if model_type == "RandomForest":
            self.fi_rnd_np = df_stat_np
        elif model_type == "GradientBoosting":
            self.fi_grd_np = df_stat_np
        else:
            print("Modello ML non previsto.")
            return

        path_app = self.dati.get_path()
        path_app = path_app + model_type + '_sorted_fi.csv'

        df_stat.to_csv(path_app, index=False)
        df_stat = read_csv(path_app)

        # ##### Ordinamento statistiche
        # traslo il dataframe e gli associo il nome delle features
        df_stat = df_stat.T
        df_stat = df_stat.iloc[[0]]
        df_stat = df_stat.set_axis(features_names, axis=1)

        # ordino in modo decrescente le colonne in base al valore delle caratteristiche
        df_stat = df_stat.sort_values(by='0', axis=1, ascending=False)

        # memorizzo le statistiche sulle feature importance
        if model_type == "RandomForest":
            self.fi_rnd = df_stat
        elif model_type == "GradientBoosting":
            self.fi_grd = df_stat
        else:
            print("Modello ML non previsto.")
            return

        path_dati = self.dati.get_path()
        path_car = path_dati + 'sorted_stat_importance_rnd.csv'
        df_stat.to_csv(path_car, index=False)

    def plot_fi(self, tp):

        fig, ax = plt.subplots()

        path_app = self.dati.file_path()
        dataset = read_csv(path_app)
        dataset = dataset.iloc[:, 1:len(dataset.columns)]

        feat_names = list(dataset.columns)

        if tp == "RandomForest":
            series_importance = pd.Series(self.fi_rnd_np, index=feat_names)
            ax.set_title("Feature importances random forest")
        elif tp == "GradientBoosting":
            series_importance = pd.Series(self.fi_grd_np, index=feat_names)
            ax.set_title("Feature importances gradient boosting")
        else:
            print("Modello ML non previsto.")
            return
        series_importance.plot.bar(ax=ax)
        fig.tight_layout()
        plt.show()

    def get_fi(self, t_ml):
        if t_ml == "RandomForest":
            return self.fi_rnd
        elif t_ml == "GradientBoosting":
            return self.fi_grd


# ##############################################################################
```

```
#                                 ScoreTime
# ##############################################################################
# Si occupa di analizzare un modello di ML (Random Forest o Gradient Boosting),
# andando a prendere in considerazione il tempo di addestramento necessario e
# l'accuratezza delle previsioni per sottoinsiemi del dataset generati in base
# alle feature importance.
# L'analisi si distingue in :
# 1) "Prova"       : i sottoinsiemi generati partono dalle 3 caratteristiche più
#                    importanti andando ad includere ad ogni iterazione la
#                    caratteristica più importante non presente nel sottoinsieme
# 2) "Controprova" : i sottoinsieme generati partono da tutto il dataset andando
#                    a togliere, ad ogni iterazione la caratteristica più
#                    importante fino ad avere un sottoinsieme di 3 caratteristiche
#                    meno importanti
# -> exec_model()        : esegue la fase di rilevazione del tempo di addestramento
#                          e dello score per insiemi crescenti o decrescenti, in
#                          base all'inizializzazione dell'instanza (prova o
#                          controprova), tutto questo per un certo modello
# -> plot_scoretime()    : mostra graficamente l'andamento dei tempi e dello score
#                          al variare dei sottoinsiemi per un certo modello
# -> exec()          : esegue exec_model() chiamandola per entrambi i modelli
#                      (Random Forest e Gradient Boosting)
# -> plot()          : esegue plot_scoretime() chiamandola per entrambi i modelli
class ScoreTime:
    def __init__(self, file, contro):
        self.dati = file
        self.controprova = contro

    def exec_model(self, model_ml):
        # ottengo la media delle feature importance ordinate (!)
        model_feature_importance = FeatImportance(self.dati)
        model_feature_importance.add_fi(model_ml)
        sorted_feature_importance = model_feature_importance.get_fi(model_ml)

        # recupero il dataset
        total_data = read_csv(self.dati.file_path)

        feat_data = total_data.iloc[:, 1:len(total_data.columns)]
        trget_data = total_data.iloc[:, 0]

        # imposto indice n per la valutazione delle feature (più avanti)
        if not controprova:
            n_index = 3
        else:
            n_index = 1

        # imposto il path dove verranno scritti i risultati dell'esecuzione
        start_path = self.dati.get_path()
        if model_ml == "RandomForest":
            start_path = start_path + 'interm_results_rnd_'
        elif model_ml == "GradientBoosting":
            start_path = start_path + 'interm_results_grd_'

        if controprova:
            start_path = start_path + 'c_'

        end_path = '.csv'

        # setto la condizione del ciclo while
        if not controprova:
            while_condition = len(feat_data.columns)
        else:
            while_condition = len(feat_data.columns) - 3

        while n_index <= while_condition:

            if not controprova:
                # trovo le N-len features meno rilevanti (PROVA)
                n_proc_features = sorted_feature_importance.iloc[:,
n_index:len(feat_data.columns)]
            else:
                # trovo le N features più rilevanti (CONTROPROVA)
                n_proc_features = sorted_feature_importance.iloc[:, 0:n_index]
```

```
            # per ogni feature tolgo dal dataset totale
            n_dataset = features_data.copy()
            for feat in n_proc_features.columns:
                n_dataset = n_dataset.drop(feat, axis=1)

            # addestro per 5 volte il modello e ottengo la media degli score e tempi
            i = 0
            times = np.float64([0, 0, 0, 0, 0])
            scores = np.float64([0, 0, 0, 0, 0])
            while i < 5:
                start = time.time()

                train_data, test_data, train_labels, test_labels = train_test_split(
                    n_dataset, trget_data)

                model_app = MetodoML(train_data, test_data, train_labels, test_labels,
model_ml)

                # addestro il modello
                model_app.train_model()

                # trovo lo score del modello
                score_app = model_app.get_score()

                end = time.time()
                # mi salvo dati dello score e tempi
                times[i] = end - start
                scores[i] = score_app
                i = i + 1

            # calcolo statistiche
            avg_times = statistics.mean(times)
            avg_scores = statistics.mean(scores)

            # salvo valore delle statistiche
            data = [{'score': avg_scores, 'times': avg_times}]
            df_elem = pd.DataFrame(data)

            if not controprova:
                file_index = n_index - 3
            else:
                file_index = n_index

            path_n = start_path + str(file_index) + end_path
            df_elem.to_csv(path_n, index=False)

            n_index = n_index + 1

    def plot_scoretime(self, model_ml):
        path_file = self.dati.file_path()
        total_data = read_csv(path_file)
        feat_data = total_data.iloc[:, 1:len(total_data.columns)]
        i = 0
        if not controprova:
            n_index = 0
        else:
            n_index = 1

        # imposto il path dei file dei risultati
        start_path = self.dati.get_path()
        if model_ml == "RandomForest":
            start_path = start_path + 'interm_results_rnd_'
        elif model_ml == "GradientBoosting":
            start_path = start_path + 'interm_results_grd_'

        if controprova:
            start_path = start_path + 'c_'

        end_path = '.csv'

        labels = []
        scores = []
        times = []
        times_s = []
```

```
        # setto la condizione del ciclo while
        if not controprova:
            while_condition = 16
        else:
            while_condition = len(feat_data.columns) - 3

        while n_index < while_condition:
            # imposto il path n-esimo e leggo i risultati
            n_path = start_path + str(n_index) + end_path
            n_data = pd.read_csv(n_path)

            n_data = n_data.iloc[0]

            # imposto n. di feature utilizzate
            if not controprova:
                n_label = str(n_index + 3)
            else:
                n_label = str(17 - i)

            # memorizzo i risultati
            labels.append(n_label)
            scores.append(n_data['score'])
            times.append(n_data['times'])
            times_s.append(time.strftime("%M:%S", time.gmtime(n_data['times'])))
            n_index = n_index + 1
            if controprova:
                i = i + 1

        ###############################################################################
        df_date = pd.DataFrame({'score': scores, 'time': times_s}, index=labels)
        print(df_date)
        print(" ")

        fig, ax1 = plt.subplots()

        # imposto il titolo della x label
        if not controprova:
            x_label_title = 'first N important features'
        else:
            x_label_title = 'less N important features (tatal 18 features)'
        ax1.set_xlabel(x_label_title)

        ax1.set_ylabel('score', color="tab:red", weight='bold')
        ax1.plot(scores, color="tab:red", marker='.', markerfacecolor='maroon')
        ax1.tick_params(axis='y', labelcolor="tab:red")

        # si mostrano le due curve sullo stesso grafico
        ax2 = ax1.twinx()
        ax2.set_ylabel('tempo di esecuzione', color="tab:blue", weight='bold')
        ax2.plot(times, color="tab:blue", marker='.', markerfacecolor='darkblue')
        ax2.tick_params(axis='y', labelcolor="tab:blue")

        # imposto il range di etichette dell'asse del tempo
        if controprova:
            right_edge = 1000
        elif model_ml == 'RandomForest':
            right_edge = 700
        elif model_ml == 'GradientBoosting':
            right_edge = 900
        else:
            print("Modello ml non previsto.")
            return

        y_minutes = [time.strftime("%M:%S", time.gmtime(x)) for x in np.arange(0,
right_edge, 100)]
        y_t = [x for x in np.arange(0, right_edge, 100)]

        plt.yticks(y_t, y_minutes)

        # imposto i valori dell'asse x
        if not controprova:
            plt.xticks(np.arange(0, 16, step=1), np.arange(3, 19, step=1))
        else:
```

```
            x_label = np.arange(3, 18, step=1)
            x_label = sorted(x_label, reverse=True)
            plt.xticks(np.arange(0, 15, step=1), x_label)

        # imposto titolo del grafico
        if model_ml == 'RandomForest':
            text_title = "Statistiche random forest"
        elif model_ml == 'GradientBoosting':
            text_title = "Statistiche gradient boosting"
        else:
            print("Modello ml non previsto.")
            return

        if controprova:
            text_title = text_title + " controprova"
        else:
            text_title = text_title + " prova"

        plt.title(text_title, weight='bold', fontsize=16, pad=16)
        plt.grid()
        fig.tight_layout()
        plt.show()

    def exec(self):
        self.exec_model("RandomForest")
        self.exec_model("GradientBoosting")

    def plot(self):
        self.plot_scoretime("RandomForest")
        self.plot_scoretime("GradientBoosting")


if __name__ == "__main__":
    print("Inserisci path del dataset:")
    path_dataset = input()
    filename = "EMOCON_VALENCE.CSV"
    dati_emocon = Dati(path_dataset, filename)
    print("path memorizzato:\n\t")
    print(dati_emocon.file_path())
    print("")
    while 1:
        print("Chi sei?\n\t1)ML Engineer\n\t2)Decision Maker\n")
        chi_sono = input()
        if chi_sono == '1' or chi_sono == 'ML Engineer':
            ml = True
            break
        elif chi_sono == '2' or chi_sono == 'Decision Maker':
            ml = False
            break
        print("Formato sbagliato o ruolo errato!")

    if ml:
        # MENU PER IL MACHINE LEARNING ENGINEER
        check_fi = False
        check_st_p = False
        check_st_c = False
        while 1:
            print("""Ciao ML engineer! Dimmi cosa vuoi fare:\n
                \t1) Visualizza le feature importance
                \t2) Visualizza l'analisi del tempo di esecuzione e score
                \t3) Proponi modello ottimo
                \t4) Esci
                """)
            risposta = input()
            if risposta == '1':
                fi_class = FeatImportance(dati_emocon)
                fi_class.add_fi("RandomForest")
                fi_class.add_fi("GradientBoosting")
                fi_class.plot_fi("RandomForest")
                fi_class.plot_fi("GradientBoosting")
                check_fi = True
            elif risposta == '2':
                print("""Quale analisi si vuole visualizzare?\n\t
                    1) Prova\n\t
```

```
                    2) Controprova\n""")
                while 1:
                    risposta = input()
                    if risposta == '1':
                        controprova = False
                        check_st_p = True
                    if risposta == '2':
                        controprova = True
                        check_st_c = True
                    else:
                        print("Formato errato, inserisci 1 o 2.")
                        continue
                    st_class = ScoreTime(dati_emocon, controprova)
                    st_class.exec()
                    st_class.plot()
            elif risposta == '3':
                if check_fi and check_st_p and check_st_c:
                    print("""Quale modello si propone?\n
                        \t1) RandomForest\n
                        \t2) Gradient Boosting""")
                    while 1:
                        model = input()
                        if model == '1' or model == 'RandomForest':
                            model = 'RandomForest'
                            break
                        elif model == '2' or model == 'GradientBoosting':
                            model = 'Gradient boosting'
                            break
                        else:
                            print("Formato o modello non previsto.")

                    print("Quante feature più importanti si propongono?")
                    while 1:
                        num = input()
                        num = int(num)
                        if 3 <= num <= 18:
                            break
                        else:
                            print("Inserire un numero nel range [3,18]")
                    df_modello_proposto = pd.DataFrame(data={'modello': model, 'n':
num})
                    path = dati_emocon.get_path() + 'prop_model.csv'
                    df_modello_proposto.to_csv(path, index=False)
                    print("Modello proposto:")
                    print(df_modello_proposto)
                else:
                    if not check_fi:
                        print("Manca ancora la visualizzazione delle feature
importance.")
                    if not check_st_p:
                        print("Manca ancora la visualizzazione dell'analisi score-
time.")
                    if not check_st_c:
                        print("Manca ancora la visualizzazione della controprova score-
time.")
            elif risposta == '4':
                break
            else:
                print("Comando inserito non valito!\n")

    if not ml:
        # MENU PER IL DECISION MAKER
        new_dati = None
        validated_model = None
        while 1:
            print("""Ciao Decision Maker! Dimmi cosa vuoi fare:\n
                \t1) Valida modello proposto dal ML engineer\n
                \t2) Carica nuovi dati col modello validato\n
                \t3) Leggi previsioni sui nuovi dati\n
                \t4) Esci""")
            risposta = input()
            if risposta == '1':
                path = dati_emocon.get_path()
                peth_emocon = dati_emocon.file_path()
```

```
tot_dataset = read_csv(path)

# Verifico la presenza di risultati da mostrare
file_check_path_1 = path + 'interm_results_rnd_*'
file_check_path_2 = path + 'interm_results_rnd_c_*'
if glob(file_check_path_1):
    if glob(file_check_path_2):
        # display prova - controprova
        prova = ScoreTime(dati_emocon, False)
        controprova = ScoreTime(dati_emocon, True)
        prova.plot()
        controprova.plot()
    else:
        print("Dati della controprova non disponibili.")
        continue
else:
    print("Dati dell'analisi score-time non disponibili.")

print("Recupero le feature importance...")
fi_class = FeatImportance(dati_emocon)
fi_class.add_fi("RandomForest")
fi_class.add_fi("GradientBoosting")
fi_class.plot_fi("RandomForest")
fi_class.plot_fi("GradientBoosting")
fi_list_rnd = fi_class.get_fi("RandomForest")
fi_list_grd = fi_class.get_fi("GradientBoosting")

file_check_path = path + 'prop_model.csv'
# verifico se esiste un modello da validare
if glob(file_check_path):
    prop_model = read_csv(file_check_path)
else:
    print("Ancora nessun modello è stato proposto dal ML engineer.\n")
    continue
model = prop_model["modello"]
model = model.iloc[[0]]
n = prop_model["n"]
n = int(n.iloc[[0]])

print(f"""Modello proposto dal ML engineer:\n
    \t-> Modello ML : {model}\n
    \t-> Numero di caratteristiche più importanti : {n}\n
    \n Validare? (y\n)""")
while 1:
    risposta = input()
    if risposta == 'y':
        # Genera modello e addestra modello
        if model == "RandomForest":
            fi = fi_list_rnd
        elif model == "GradientBoosting":
            fi = fi_list_grd
        else:
            print("Modello ML non previsto.")
            continue
        print("Modello validato.")
        # filtra dataset
        features_data = tot_dataset.iloc[:, 1:len(tot_dataset.columns)]
        target_data = tot_dataset.iloc[:, 0]

        feature_to_drop = fi.iloc[:, n:len(features_data.columns)]
        good_feat = fi.iloc[:, 0:n - 1]
        filtered_data = features_data.copy()
        for feature in feature_to_drop.columns:
            filtered_data = filtered_data.drop(feature, axis=1)

        print("(!!) Caratteristiche del modello: (!!)")
        for g_f in good_feat.columns:
            print(g_f)

        # train test split
        training_data, testing_data, training_labels, \
            testing_labels = train_test_split(filtered_data,
                                              target_data)
```

```
                # inizializza modello

                if model == "RandomForest":
                    validated_model = RandomForestClassifier()
                elif model == "GradientBoosting":
                    validated_model = GradientBoostingClassifier()
                else:
                    print("Modello ML non previsto.")

                print(f"Addestramento modello ({model}) in corso...")

                # train modello
                validated_model.fit(training_data, ravel(training_labels))

                # test modello
                predicted_labels = validated_model.predict(testing_data)
                score = accuracy_score(ravel(testing_labels),
                                       predicted_labels)

                print(f"Modello validato, score: {score}")
            if risposta == 'n':
                print("""Modello NON validato, in attesa di un altro
                        modello dal ML engineer.""")
            # elimina file csv
            file_to_delete = Path(file_check_path)
            file_to_delete.unlink()
    elif risposta == '2':
        print("Inserisci il filename: (.csv)")
        risposta = input()
        new_flnm = risposta
        print("Nome del file inserito:\n\t\t")
        print(new_flnm)

        print("Inserisci il path del file name:")
        risposta = input()
        path_flnm = risposta
        print("Path inserito:\n\t\t")
        print(path_flnm)

        new_dati = Dati(path_flnm, new_flnm)
    elif risposta == '3':
        try:
            results = validated_model.predict(new_dati)
            print("Risultati della previsione:")
            print(results)
        except NameError:
            print("Modello non ancora validato.")
        else:
            print("Modello non ancora validato.")
    elif risposta == '4':
        break
    else:
        print("Comando inserito non valito!\n")
```

APPENDICE B: Tabelle e immagini supplementari

Tabella B.1 Tabella caso d'uso "VisualizzaFeatureImportance"

Use case: VisualizzaFeatureImportance
ID: 1
Brief descprition: il ML engineer visualizza il valore delle feature importance, ottenuto calcolandole per cinque iterazioni e facendo la media di tali valori
Primary actors: Machine Learning engineer
Secondary actors: Decision Maker
Preconditions: Il ML engineer ha inserito il path del dataset
Main flow: 1. Il caso d'uso inizia quando il Machine Learning engineer immette "VisualizzaFI" 2. **WHILE** il sistema non ha calcolato per cinque volte le feature importance 2.1. include(CalcolaFeatureImportance) 3. il sistema calcola la media delle feature importance per il Random Forest 4. il sistema calcola la media delle feature importance per il Gradient Boosting 5. il sistema memorizza la media delle feature importance su file 6. il sistema mostra all'utente i valori medi delle feature importance del Random Fores e del Gradient Boosting
Postconditions: Nessuno
Alternative flows: Nessuno

Tabella B.2 Tabella caso d'uso "CalcolaFeatureImportance"

Use case: CalcolaFeatureImportance
ID: 2
Brief descprition: il sistema calcola il valore delle feature importance dei modelli Random Forest e Gradient Boosting
Primary actors: Machine Learning engineer
Secondary actors: Nessuno
Preconditions: Nessuna
Main flow:

1. Il caso d'uso inizia quando il Machine Learning engineer chiede il calcolo delle features importance 2. Il sistema addestra il modello Random Forest 3. Il sistema ritorna le feature importance per tale modello 4. Il sistema addestra il modello Gradient Boosting 5. Il sistema ritorna le feature importance per tale modello
Postconditions: Nessuno
Alternative flows: Nessuno

Tabella B.3 Tabella caso d'uso relativa a "VisualizzaStatisticheTempoScore"

Use case: *VisualizzaStatisticheTempoScore*
ID: 3
Brief descprition: Il sistema mostra al ML engineer e al Decision Maker graficamente le statistiche sul tempo di esecuzione medio e valore di accuratezza delle previsioni medio in relazione al numero delle caratteristiche prese in considerazione.
Primary actors: Machine Learning engineer
Secondary actors: Decision Maker
Preconditions: Il sistema ha calcolato le feature importance
Main flow: 1. Il caso d'uso inizia quando il cliente immette "CalcoloFI" 2. **WHILE** il sistema non ha valutato tutte le feature importance del Random Forest 2.1 il sistema calcola il nuovo sottoinsieme del dataset 2.2 include(CalcolaTempoScore) 3. il sistema memorizza i dati calcolati su file 4. il sistema mostra nello stresso grafico la curva degli score medi e la curva del tempo di esecuzione medio in relazione alle feature utilizzate per il Random Forest 5. **WHILE** il sistema non ha valutato tutte le feature importance del Gradient Boosting 5.1 il sistema calcola il nuovo sottoinsieme del dataset 5.2 include(CalcolaTempoScore) 6. il sistema memorizza i dati calcolati su file 7. il sistema mostra nello stresso grafico la curva degli score medi e la curva del tempo di esecuzione medio in relazione alle feature utilizzate per il Gradient Boosting
Postconditions: Nessuno
Alternative flows: Nessuno

Tabella B.4. Tabella caso d'uso relativa a "VisualizzaTSProva"

Use case: VisualizzaTSProva
ID: 4
Parent ID: 3
Brief descprition: Il sistema mostra al ML engineer e al Decision Maker graficamente le statistiche sul tempo di esecuzione medio e valore di accuratezza delle previsioni medio per un numero di feature in ordine crescente in base all'importanza.

Primary actors: Machine Learning engineer
Secondary actors: Decision Maker
Preconditions: Il sistema ha calcolato le feature importance
Main flow: 1. (o1.) Il caso d'uso inizia quando il cliente immette "CalcoloFIProva" 2. include(CalcolaFeatureImportance) 3. (2.) **WHILE** il sistema non ha valutato tutte le feature importance del Random Forest 3.1 (o2.1) il sistema calcola il nuovo sottoinsieme del dataset aggiungendo la caratteristica più importante che non fa parte di tale sottoinsieme 3.2 (2.2) include(CalcolaTempoScore) 4. (3.) il sistema memorizza i dati calcolati su file 5. (4.) il sistema mostra nello stresso grafico la curva degli score medi e la curva del tempo di esecuzione medio in relazione alle feature utilizzate per il Random Forest 6. (5.) **WHILE** il sistema non ha valutato tutte le feature importance del Gradient Boosting 6.1 (o5.1) il sistema calcola il nuovo sottoinsieme del dataset aggiungendo la caratteristica più importante che non fa parte di tale sottoinsieme 6.2 (5.2) include(CalcolaTempoScore) 7. (5.) il sistema memorizza i dati calcolati su file 8. (7.) il sistema mostra nello stresso grafico la curva degli score medi e la curva del tempo di esecuzione medio in relazione alle feature utilizzate per il Gradient Boosting
Postconditions: Nessuno
Alternative flows: Nessuno

Tabella B.5. Tabella caso d'uso relativa a "VisualizzaTSControprova"

Use case: VisualizzaTSControprova
ID: 5
Parent ID: 3
Brief descprition: Il sistema mostra al ML engineer e al Decision Maker graficamente le statistiche sul tempo di esecuzione medio e valore di accuratezza delle previsioni medio per un numero di feature in ordine decrescente in base all'importanza.
Primary actors: Machine Learning engineer
Secondary actors: Decision Maker
Preconditions: Il sistema ha calcolato le feature importance
Main flow: 1. (o1.) Il caso d'uso inizia quando il cliente immette "CalcoloFIControprova" 2. include(CalcolaFeatureImportance) 3. (2.) **WHILE** il sistema non ha valutato tutte le feature importance del Random Forest 3.1 (o2.1) il sistema calcola il nuovo sottoinsieme del dataset togliendo la caratteristica più importante che fa parte di tale sottoinsieme 3.2 (2.2) include(CalcolaTempoScore) 4. (3.) il sistema memorizza i dati calcolati su file 5. (4.) il sistema mostra nello stresso grafico la curva degli score medi e la curva del tempo di esecuzione medio in relazione alle feature utilizzate per il Random Forest 6. (5.) **WHILE** il sistema non ha valutato tutte le feature importance del Gradient Boosting 6.1 (o5.1) il sistema calcola il nuovo sottoinsieme del dataset togliendo la caratteristica più importante che fa parte di tale sottoinsieme 6.2 (5.2) include(CalcolaTempoScore) 7. (5.) il sistema memorizza i dati calcolati su file

8. (7.) il sistema mostra nello stresso grafico la curva degli score medi e la curva del tempo di esecuzione medio in relazione alle feature utilizzate per il Gradient Boosting
Postconditions: Nessuno
Alternative flows: Nessuno

Tabella B.6. Tabella caso d'uso relativa a "CalcolaTempoScore"

Use case: CalcolaTempoScore
ID: 6
Brief descprition: Il sistema calcola il tempo medio di esecuzione e lo score medio per cinque iterazioni per un sottoinsieme del dataset e lo fornisce al Machine Learning engineer.
Primary actors: Machine Learning engineer
Secondary actors: Nessuno
Preconditions: Nessuno
Main flow: 1. Il caso d'uso inizia quando il Machine Learning engineer chiede il calcolo del tempo medio di esecuzione e del valore medio dello score 2. **WHILE** il sistema non ho fatto cinque iterazioni per il modello Random Forest 2.1 Il sistema addestra il modello Random Forest per il sottoinsieme in esame 2.2 Il sistema registra il tempo di esecuzione e calcola l'accuratezza della previsione 3. Il sistema ritorna al ML engineer il valore medio del tempo e dello score registrati per il Random Forest 4. **WHILE** il sistema non ho fatto cinque iterazioni per il modello Gradient Boosting 4.1 Il sistema addestra il modello Random Forest per il sottoinsieme in esame 4.2 Il sistema registra il tempo di esecuzione e calcola l'accuratezza della previsione 5. Il sistema ritorna al ML engineer il valore medio del tempo e dello score registrati per il Gradient Boosting
Postconditions: Nessuno
Alternative flows: Nessuno

Tabella B.7 Tabella caso d'uso relativa a "ScegliModelloEParametri"

Use case: ScegliModelloEParametri
ID: 7
Brief descprition: Il Machine Learning engineer viste le spiegazioni sceglie il modello e i parametri più adatti
Primary actors: Machine Learning engineer
Secondary actors: DecisionMaker
Preconditions: Machine Learning engineer ha letto le spiegazioni sulle feature importance e le graficazioni tempo - score
Main flow: 1. Il caso d'uso inizia quando il Machine Learning engineer digita "ScegliModelloEParametri" 2. Il ML engineer digita il modello da utilizzare 3. Il ML engineer digita il numero di caratteristiche da utilizzare 4. Il sistema invia il modello e i parametri scelti al Decision Maker quando lo richiede

Postconditions: Nessuno
Alternative flows: Nessuno

Tabella B.8 Tabella caso d'uso relativa a "ValidaSpiegazioni"

Use case: ValidaSpiegazioni
ID: 8
Brief descprition: Il Decision Maker valida le spiegazioni e il modello scelto
Primary actors: Decision Maker
Secondary actors: Nessuno
Preconditions: Machine Learning engineer ha generato le spiegazioni e scelto il modello con i parametri
Main flow: 1. include(VisualizzaFeatureImportance) 2. include(VisualizzaStatisticheTempoScore) 3. include(ScegliModelloEParametri) 4. Il Decision Maker valida le spiegazione e il modello scelto dal ML engineer
Postconditions: Nessuno
Alternative flows: Nessuno

Tabella B.9 Tabella caso d'uso relativa a "CaricaDatiFisiologici"

Use case: CaricaDatiFisiologici
ID: 9
Brief descprition: Il Decision Maker inserisce nuovi dati di segnali fisiologiciper poi utilizzare il modello validato
Primary actors: Decision Maker
Secondary actors: Nessuno
Preconditions: Il Decision Maker ha validato le spiegazioni e il modello scelto dal ML engineer
Main flow: 1. Il caso d'uso inizia quando il Decision Maker digita "InserisciNuoviDati" 2. Il Decision Maker inserisce i nuovi dati fisiologici
Postconditions: Nessuno
Alternative flows: Nessuno

Tabella B.10 Tabella caso d'uso relativa a "LeggiPrevisione"

Use case: LeggiPrevisione
ID: 10
Brief descprition: Il Decision Maker utilizzando il modello validato legge le previsioni per il set di dati inseriti
Primary actors: Decision Maker
Secondary actors: Nessuno
Preconditions: Il Decision Maker inserito i nuovi dati su cui calcolare la predizione
Main flow: 1. Il caso d'uso inizia quando il Decision Maker digita "LeggiPrevisione" 2. Il sistema applica il modello approvato e i parametri ai dati di ingresso 3. Il sistema mostra al Decision Maker la previsione
Postconditions: Nessuno
Alternative flows: Nessuno

Tabella B.11 Risultato dell'esecuzione della fase 1 del Report 6, features importance ordinate in modo decrescente (file sorted_stat_importance_rnd.csv)

E4_TEMP	E4_EDA	E4_IBI	E4_HR	x	z	y	Attention	highAlpha
0.227491566	0.2054628577	0.1264146761	0.09426653991	0.07052704319	0.05689949064	0.0525427345	0.02307403264	0.02007922275

Meditation	lowGamma	theta	middleGamma	highBeta	lowAlpha	lowBeta	delta	E4_BVP
0.01967144635	0.01923147048	0.01695542096	0.01441109067	0.01350694214	0.01296858341	0.01200138224	0.01181111759	0.002684382682

Figura B1. Dati tabellari dei risultati del Random Forest del Report 6.

	score	time
3	0.990468	02:52
4	0.999932	04:56
5	0.999928	04:01
6	0.999939	04:03
7	0.999942	03:49
8	0.999954	05:14
9	0.999948	06:27
10	0.999953	07:10
11	0.999950	07:12
12	0.999961	08:09
13	0.999954	08:50
14	0.999963	08:51
15	0.999959	07:59
16	0.999950	09:47
17	0.999952	10:11
18	0.999949	09:38

Figura B2. Dati tabellari dei risultati del Gradient Boosting del Report 6.

	score	time
3	0.861920	02:13
4	0.876592	02:57
5	0.894230	03:16
6	0.892170	04:10
7	0.902044	04:30
8	0.899417	05:25
9	0.901866	06:30
10	0.902425	07:14
11	0.902838	08:02
12	0.900131	09:05
13	0.907565	09:25
14	0.905360	10:18
15	0.905391	10:42
16	0.907087	11:16
17	0.906403	11:57
18	0.907469	13:03

Figura B3. Dati tabellari dei risultati del Random Forest del Report 7.

	score	time
17	0.999929	10:07
16	0.999715	11:44
15	0.999201	09:27
14	0.996912	10:16
13	0.996223	10:31
12	0.997108	11:35
11	0.996565	12:54
10	0.993944	13:38
9	0.992591	14:18
8	0.983937	09:56
7	0.975903	09:47
6	0.957690	09:25
5	0.905933	09:12
4	0.801037	08:48
3	0.646788	05:43

Figura B4. Dati tabellari dei risultati del Gradient Boosting del Report 7.

	score	time
17	0.878941	13:30
16	0.826144	12:57
15	0.794499	11:53
14	0.767334	11:09
13	0.756733	10:46
12	0.755676	09:39
11	0.694920	08:49
10	0.695177	07:54
9	0.694547	07:07
8	0.693102	06:06
7	0.692774	05:14
6	0.688264	04:23
5	0.623119	04:04
4	0.600214	03:07
3	0.568756	02:17

SITOGRAFIA

[1] www.scikit-learn.org/stable/modules/neural_networks_supervised.html

[2] www.towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141

[3] www.ibm.com/topics/decision-trees

[4] www.ibm.com/topics/random-forest

[5] https://www.ibm.com/topics/boosting

[6] www.nature.com/articles/s41597-020-00630-y

[7] https://colab.google/

[8] *https://victorzhou.com/blog/gini-impurity/*

[9] *https://www.learndatasci.com/glossary/gini-impurity*

[10] *https://www.psicologopadova-robertogava.it/relazione-arousal-prestazione-sportiva.htm#:~:text=In%20psicologia%20fisiologica%20il%20termine,dal%20sonno%20all'eccitazione%20diffusa*.