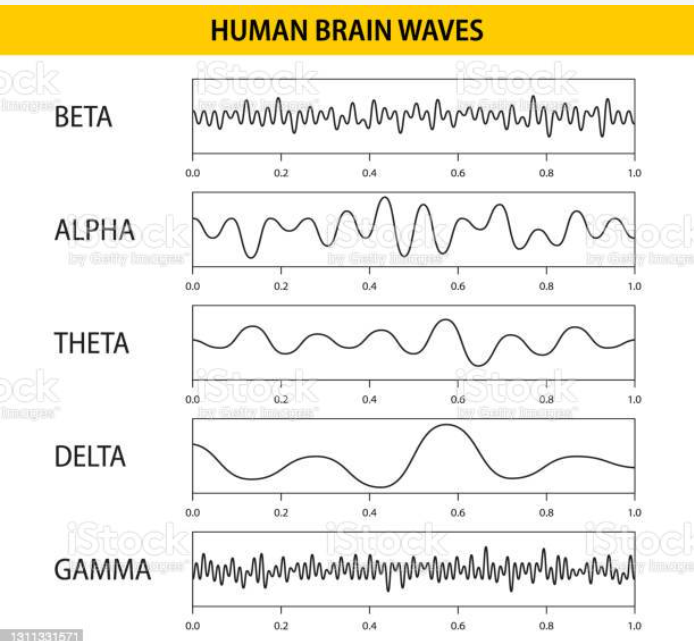
**IL PERCHE’**

Nel dataset sono presenti i valori riguardanti le sensazioni che i partecipanti all’esperimento hanno riportato, uniti ai parametri registrati dalle varie apparecchiature, in particolare le onde alpha e theta. I partecipanti dovevano assegnare un valore a: un’ etichetta emotiva, arousal, valence, dominance, predictability. Le etichette emotive includevano: neutro, ansia, divertimento, tristezza, gioia, disgusto, rabbia, sorpresa e paura. Le onde cerebrali sono suddivise in 5 categorie, a seconda dello spettro di frequenza: theta (4 Hz <f< 8 Hz), slow alpha (8 Hz <f< 10 Hz), alpha (8 Hz <f< 12 Hz), beta (12 Hz <f< 30 Hz), and gamma (30 Hz < f).   
Vogliamo effettuare una regressione sul parametro “valence” il quale indica se un’emozione è piacevole o sgradevole, e successivamente “arousal”, il quale indica se un’emozione è forte o debole. Ciò, in modo da poter capire a quali frequenze di onde cerebrali sono associati determinati punti sul piano PAD. La regressione corrisponde al valore medio atteso in base ai dati di input che noi immettiamo. Grazie alla regressione si può parlare di **previsione** sui valori attesi. Intuitivamente la regressione consente di trovare la relazione tra input e output, in modo da predire output futuri avendo a disposizione solamente l'input.



**IL COME**

Partiamo importando il dataset su python Immagine che contiene testo

Descrizione generata automaticamente

E' buona pratica nel machine learning eseguire l'addestramento su un set di dati per poi verificare i risultati delle sue predizioni con un altro set contente dati non visti durante l'addestramento.  
Questo garantisce che il nostro modello sia in grado di generalizzare su dati sconosciuti e quindi ha realmente "imparato" dai dati, piuttosto che limitarsi a memorizzare il set di addestramento, condizione conosciuta come **overfitting**. Mettiamo, allora, una parte del nostro dataset nella variabile “data\_unseen”, che useremo in seguito. Quello che facciamo allora è dividere il nostro unico dataset in due insiemi che chiamiamo:

* **training se**t, sarà usato nella fase di *training* per consentire al modello di apprendere la relazione nascosta tra i dati. Presente nella variabile “data” del codice;
* **testing set**, sarà usato nella fase di *testing* per valutare le prestazioni di generalizzazione del modello, calcolando l’errore tra i risultati predetti e quelli reali. Presente nella variabile “data\_unseen”;

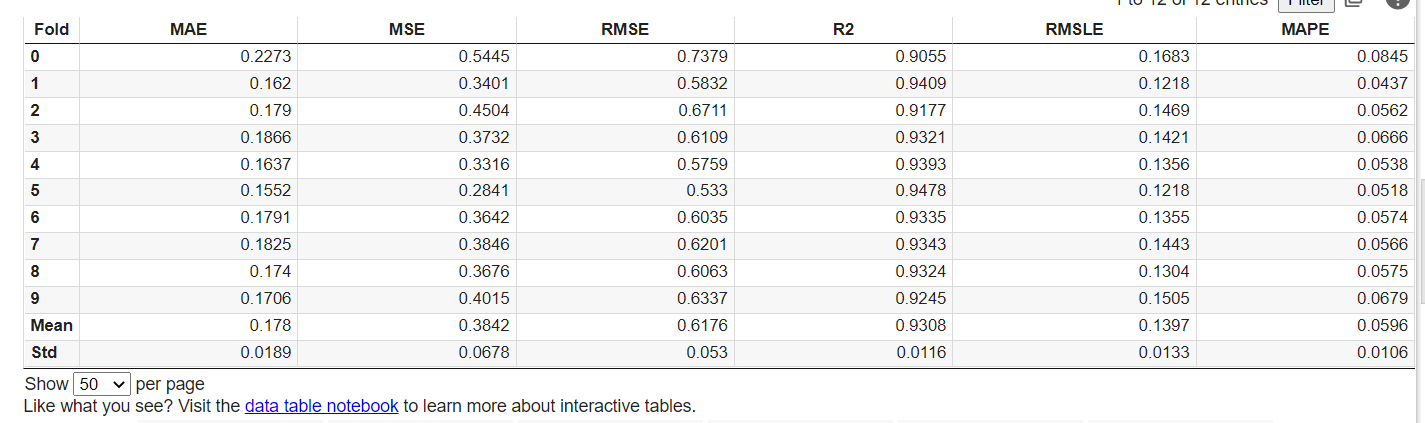
Usiamo il 90% del dataset come insieme di *training*. La parte restante è chiamata *hold out* e, per completezza d’informazione, costituisce il 20%.

**Caso ‘valence’**

Prima di tutto abbiamo bisogno di trovare il modello più adatto al nostro scopo, e per far ciò utiliziamo la “compare\_models()”. Il modello che restituisce le migliori prestazioni è il decision tree, come vediamo nella tabella dal valore di R2.Immagine che contiene tavolo

Descrizione generata automaticamente

Dunque andiamo a creare il nostro modello, il quale restituisce i seguenti valori:



Contestualizziamo i seguenti valori: y valore originale, y cappello valore predetto, y barra valore medio di y

Mean absolute error, rappresenta la differenza fra il valore originale e quello predetto, facendo la media fra la somma delle differenze fra i due valori in valore assoluto.

Immagine che contiene testo

Descrizione generata automaticamente

Means quare error, rappresenta la differenza fra il valore originale e quello predetto, facendo la media fra la somma delle differenze fra i due valori elevata al quadrato.

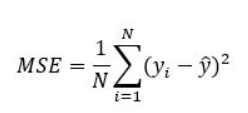


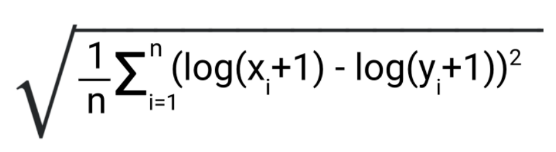
Immagine che contiene testo

Descrizione generata automaticamente La radice quadrata dell’indice sopra, è il tasso di errore

Immagine che contiene testo

Descrizione generata automaticamente

Rappresenta il coefficiente di adattamento dei valori predetti rispetto ai valori originali. Il valore è compreso fra 0 e 1 e va interpretato come una percentuale. Più alto è il valore, migliore è il modello.

RMSLE = 

Root Mean Squared Log Error: stavolta la differenza è fra il logaritmo del valore originale rispetto a quello del valore predetto

Immagine che contiene testo

Descrizione generata automaticamente

Mean absolute percentage error: F è il valore predetto, A quello attuale

Di tutti questi valori, nel momento in cui andiamo a comparare i modelli, viene mostrata la media. Adesso guardiamo i risultati di predizione del modello:

Immagine che contiene testo

Descrizione generata automaticamente

L'R2 sul test/hold-out è 0,9366 rispetto allo 0,9308 ottenuto sui risultati della create\_model(). Questa non è una differenza significativa. Se c'è una grande variazione tra i risultati del test/hold-out e quelli della create\_model(), questo è normalmente dovuto all’ over-fitting, come potrebbe essere dovuto a diversi altri fattori e richiederebbe ulteriori indagini. In questo caso, andremo avanti con la finalizzazione del modello e la previsione su dati invisibili (la parte del dataset inserita nella variabile “data\_unseen” che avevamo separato all'inizio e non abbiamo utilizzato durante la scelta del modello). Andiamo dunque a finalizzare il modello e vedere le prestazioni sui dati non ancora visti:

Immagine che contiene testo

Descrizione generata automaticamente

**Caso ‘arousal’**

Ripetiamo gli stessi passaggi cambiando il target in “arousal”. Anche questa volta il modello scelto è il decision tree regressor, per via delle migliori prestazioni sul parametro R2:

Immagine che contiene tavolo

Descrizione generata automaticamente

Ottenendo risultati leggermente peggiori rispetto al caso precedente. I valori delle altre colonne si riferiscono al valore medio misurato rispetto a quei parametri. I risultati di predizione del modello:

Immagine che contiene testo

Descrizione generata automaticamente

Mentre quelli ottenuti dopo aver finalizzato il modello, sui dati non ancora visti, sono:

Immagine che contiene testo

Descrizione generata automaticamente

Il codice è:

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

**USO DI SKLEARN**

Guardando i risultati ottenuti nelle pagine precedenti, si nota che R2 del modello sui dati non visti è superiore a quello sui dati di training. Per verificare se il modello dà buoni risultati anche utilizzando altre librerie, testiamolo con sklearn, in quanto Pycaret opera ad un livello di astrazione troppo alto per riuscire a capire se sia stata eseguita una qualche azione di processing dei dati che non risulta dall’esposizione dei risultati sopra elencati. Allora andiamo ad importare di nuovo il nostro dataset, insieme a diverse librerie:

Immagine che contiene testo

Descrizione generata automaticamente

Adesso sfruttiamo le funzioni della libreria sklearn per effetuare una regressione su arousal:

Immagine che contiene testo

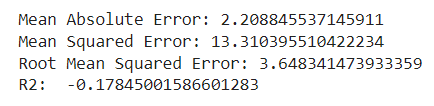
Descrizione generata automaticamente

Per il momento, come fatto in precedenza con pycaret, non consideriamo le colonne “video”, “window” e “subject”. Dunque, ‘arousal’ diventa il parametro della prima colonna, e per selezionarlo utilizziamo “iloc[:,0:1]”. Nel caso in questione, stiamo eseguendo il fit sull’intero dataset, dunque quando andiamo ad eseguire la predizione sui dati di test, questi sono tutti dati già visti ed il modello sa come rispondere. Perciò, i risultati ottenuti sono molto positivi:

Immagine che contiene testo

Descrizione generata automaticamente

La situazione cambia nel momento in cui andiamo ad utilizzare solamente i dati di training per effettuare il fit, dunque “regr.fit(X\_train, y\_train)” al posto di “regr.fit(X,y)”. In questo caso, quando poi andiamo ad eseguire la predizone sui dati di test ( stavolta non sono dati che il modello ha già visto, quindi deve dimostrare di saper apprendere anche su dati non ancora visti), i risultati sono molto negativi:



Stessi risultati si ottengono effettuando la regressione su “valence”.

**ALTRI MODELLI SU SKLEARN**

Se andiamo ad effettuare, con sklearn, una regressione usando altri modelli, i risultati ottenuti sono molto simili a quelli indicati da Pycaret, riporto due esempi:

* Con KNN: “regr = KNeighborsRegressor(n\_neighbors=2)” importando “from sklearn.neighbors import KNeighborsRegressor” otteniamo:

Immagine che contiene testo

Descrizione generata automaticamente

* Con lasso regression: “regr = linear\_model.Lasso(alpha= 0.1)” importando “from sklearn import linear\_model” otteniamo:

Immagine che contiene testo

Descrizione generata automaticamente

**USO DI ExtraTreesRegressor**

Vediamo col modello potenziato dei decision tree, sempre effettuando una regressione:  
**Immagine che contiene testo

Descrizione generata automaticamente**

Nel caso di “arousal”, del codice in figura, il risultato che otteniamo per lo score, che nel caso del metodo in questione rappresenta il valore di R2, è:



Otteniamo quindi prestazioni nettamente superiori a prima, ma che rimangono comunque inferiori a quelle utilizzando Pycaret. Anche effettuando la regressione su “valence”, dunque cambiando la riga “y=dataset.iloc[:,0]” in y=dataset.iloc[:,1], il risultato non è molto diverso:  
 

**USO DI DecisionTreeClassifier**

Proviamo allora a spostare il problema su di una classificazione: non prevediamo più numeri reali, ma un numero intero. Il codice è:  
Immagine che contiene testo

Descrizione generata automaticamente

In cui prendiamo la colonna “arousal” e la trasformiamo in valori interi tramite “round(0)”, visto che per il classificatore ci servono valori in quella forma. Il risultato finale è:  
 

Ovvero il modello classifica correttamente circa il 48% dei campioni che si trova davanti. Similmente, nel caso di “valence” i risultati ottenuti di predizione si aggirano intorno al 50%:



**USO DI ExtraTreesClassifier**  
Utilizzando il modello potenziato del classificatore con decision tree, abbiamo:

Immagine che contiene testo

Descrizione generata automaticamente

E stavolta otteniamo risultati migliori, sia nel caso di “arousal” che nel caso di “valence” la predizione si aggira intorno al 90%.