**Il perché:**

Proviamo a verificare se l’R2 nella regressione del primo task viene più alto in test piuttosto che in training per un qualche problema di dimensionalità. Una possibile causa potrebbe essere che la stessa informazione venga ripetuta più volte nei campioni misurati all’interno dell’esperimento dal quale vengono presi i dati per effettuare la regressione. Ciò potrebbe essere causato, per esempio, da prossimità spaziali o da prossimità di banda, ovvero:  
- prossimità di due elettrodi, troppo vicini, che quindi raccolgono informazioni non indipendenti fra loro  
- prossimità della banda frequenziale delle onde celebrali, in quanto le onde che vengono misurate durante l’esperimento rientrano in un intervallo di frequenze molto vicine fra loro.

**Il come:**

Torniamo ad utilizzare Pycaret, ancora con il classificatore extraTree. Proviamo, però, ad effettuare una riduzione della dimensionalità, per vedere se in qualche modo possa influire sui risultati delle elaborazioni fatte.

from pycaret.classification import \*

import pandas as pd

import io                                                                   import numpy as np

from sklearn.preprocessing import StandardScaler

# importo il dataset

dataset = pd.read\_csv(io.BytesIO(uploaded['EEG\_MANHOB\_w8\_s2.csv']))

# elimino le colonne che non mi servono

dataset = dataset.drop('video', axis=1)

dataset = dataset.drop('window', axis=1)

dataset = dataset.drop('subject', axis=1)

# split of dataset, in order to locate the features

aux = "activity\_theta\_"

f\_theta\_1 = []

f\_theta\_2 = []

f\_theta\_3 = []

f\_theta\_4 = []

f\_theta\_5 = []

for i in range(0,5):

  for j in range(1,7):

    globals()[f"f\_theta\_{i+1}"] += [aux + str((i \* 5 + j) + i)]

f\_theta\_5 += [aux + str(31)]

f\_theta\_5 += [aux + str(32)]

aux = "activity\_alpha\_"

f\_alpha\_1 = []

f\_alpha\_2 = []

f\_alpha\_3 = []

f\_alpha\_4 = []

f\_alpha\_5 = []

for i in range(0,5):

  for j in range(1,6):

    globals()[f"f\_alpha\_{i+1}"] += [aux + str(i \* 5 + j)]

f\_target = ['arousal']

# Separating out the features

a\_theta\_1 = dataset.loc[:, f\_theta\_1].values

a\_theta\_2 = dataset.loc[:, f\_theta\_2].values

a\_theta\_3 = dataset.loc[:, f\_theta\_3].values

a\_theta\_4 = dataset.loc[:, f\_theta\_4].values

a\_theta\_5 = dataset.loc[:, f\_theta\_5].values

a\_alpha\_1 = dataset.loc[:, f\_alpha\_1].values

a\_alpha\_2 = dataset.loc[:, f\_alpha\_2].values

a\_alpha\_3 = dataset.loc[:, f\_alpha\_3].values

a\_alpha\_4 = dataset.loc[:, f\_alpha\_4].values

a\_alpha\_5 = dataset.loc[:, f\_alpha\_5].values

# Separating out the target

a\_target = dataset.loc[:, f\_target].values

# Standardizing the features

n\_theta\_1 = StandardScaler().fit\_transform(a\_theta\_1)

n\_theta\_2 = StandardScaler().fit\_transform(a\_theta\_2)

n\_theta\_3 = StandardScaler().fit\_transform(a\_theta\_3)

n\_theta\_4 = StandardScaler().fit\_transform(a\_theta\_4)

n\_theta\_5 = StandardScaler().fit\_transform(a\_theta\_5)

n\_alpha\_1 = StandardScaler().fit\_transform(a\_alpha\_1)

n\_alpha\_2 = StandardScaler().fit\_transform(a\_alpha\_2)

n\_alpha\_3 = StandardScaler().fit\_transform(a\_alpha\_3)

n\_alpha\_4 = StandardScaler().fit\_transform(a\_alpha\_4)

n\_alpha\_5 = StandardScaler().fit\_transform(a\_alpha\_5)

# Standardizing the target

n\_target = StandardScaler().fit\_transform(a\_target)

# after standardizing, start with the PCA

from sklearn.decomposition import PCA

pca = PCA(n\_components=1)

theta\_1 = pca.fit\_transform(n\_theta\_1)

theta\_2 = pca.fit\_transform(n\_theta\_2)

theta\_3 = pca.fit\_transform(n\_theta\_3)

theta\_4 = pca.fit\_transform(n\_theta\_4)

theta\_5 = pca.fit\_transform(n\_theta\_5)

alpha\_1 = pca.fit\_transform(n\_alpha\_1)

alpha\_2 = pca.fit\_transform(n\_alpha\_2)

alpha\_3 = pca.fit\_transform(n\_alpha\_3)

alpha\_4 = pca.fit\_transform(n\_alpha\_4)

alpha\_5 = pca.fit\_transform(n\_alpha\_5)

target = pca.fit\_transform(n\_target)

#ricompongo dataset

df\_theta1 = pd.DataFrame(data = theta\_1, columns = ['theta1'])

df\_theta2 = pd.DataFrame(data = theta\_2, columns = ['theta2'])

df\_theta3 = pd.DataFrame(data = theta\_3, columns = ['theta3'])

df\_theta4 = pd.DataFrame(data = theta\_4, columns = ['theta4'])

df\_theta5 = pd.DataFrame(data = theta\_5, columns = ['theta5'])

df\_alpha1 = pd.DataFrame(data = alpha\_1, columns = ['alpha1'])

df\_alpha2 = pd.DataFrame(data = alpha\_2, columns = ['alpha2'])

df\_alpha3 = pd.DataFrame(data = alpha\_3, columns = ['alpha3'])

df\_alpha4 = pd.DataFrame(data = alpha\_4, columns = ['alpha4'])

df\_alpha5 = pd.DataFrame(data = alpha\_5, columns = ['alpha5'])

# trasformo i valori da continui ad interi, altrimenti dà errore

df\_target = pd.DataFrame(data = target, columns = ['target']).round(0)

data = pd.concat([df\_theta1, df\_theta2, df\_theta3, df\_theta4, df\_theta5, df\_alpha1, df\_alpha2, df\_alpha3, df\_alpha4, df\_alpha5, df\_target], axis = 1)

# eseguo su "valence", oppure "arousal"

exp\_reg101 = setup(data = data, target = 'target', session\_id=123)

# uso extraTreeClassifier

et = create\_model('et')

# valuto il modello

evaluate\_model(et)

predict\_model(et);

final\_et = finalize\_model(et)

print(final\_et)

Ottenendo i seguenti risultati per “valence”:

Immagine che contiene testo

Descrizione generata automaticamente

Mentre i seguenti per “arousal”:

Immagine che contiene testo

Descrizione generata automaticamente

**Interpretazione dei risultati:**

Abbiamo dunque riscontrato un calo della accuratezza del modello rispetto al precedente uso con sklearn, in quanto risulta attorno al 70% contro il 90% ottenuto nelle altre prove. L'accuratezza è il numero di elementi classificati correttamente diviso per il numero totale di elementi nel set di test. È compresa tra 0 (meno accurato) e 1 (più accurato). Per quanto riguarda gli altri parametri ottenuti, anch’essi appartenenti all’intervallo [0,1] :  
- La precisione è il rapporto tra il numero delle previsioni corrette di un evento sul totale delle volte che il modello lo prevede, quindi il modello prevede in modo leggermente peggiore dopo aver ridimensionato il dataset. Quando un modello è preciso per una classe, ogni volta che prevede l'evento sbaglia raramente. Potrebbe però non prevedere tutti gli eventi ossia non essere selettivo/sensibile. Nel nostro caso la precisione si aggira intorno al 71-72%.  
- Il richiamo (recall) misura la sensibilità del modello. E' il rapporto tra le previsioni corrette per una classe sul totale dei casi in cui si verifica effettivamente. Quando un modello è sensibile per una classe, lo prevede ogni volta che si verifica. Potrebbe però prevederlo anche quando non si verifica. La precisione e la sensibilità sono spesso correlate inversamente. Quando miglioro la precisione, peggiora la sensibilità del modello. E viceversa, ed F1 consente di bilanciare la [precisione](https://docs.microsoft.com/it-it/dotnet/machine-learning/resources/glossary#precision) e il [richiamo](https://docs.microsoft.com/it-it/dotnet/machine-learning/resources/glossary#recall). Nel nostro caso recall si aggira intorno al 63-64%.  
- AUC, area under the curve, ovvero l'area sotto la curva che traccia il tasso di veri positivi rispetto al tasso di falsi positivi. Per spiegare questo concetto, i veri positivi sono le previsioni positive corrette, mentre i veri negativi sono le previsioni negative corrette. Nel caso, per esempio, di classificazione di email in “spam” oppure “no spam”, se il modello classifica un’email come “spam”, ed effettivamente lo è, siamo nel caso di vero positivo, così come se classificasse un’email come “no spam” e questa si rivelasse effettivamente non di spam. Viceversa quando parliamo di falsi positivi e falsi negativi. Nel caso in questione i valori di AUC risultano molto elevati, intorno al 90%.

Abbiamo dunque riscontrato che, effettuando una riduzione della dimensionalità, dividendo le 32 colonne delle onde theta in 5 nuovi gruppi, così come le 25 delle onde alpha in ulteriori 5 gruppi, la accuratezza diminuisce leggermente, ottenendo prestazioni meno precise col modello.

from pycaret.classification import \*

import pandas as pd

import io

import numpy as np

from sklearn.preprocessing import StandardScaler

# importo il dataset

dataset = pd.read\_csv(io.BytesIO(uploaded['EEG\_MANHOB\_w8\_s2.csv']))

# elimino le colonne che non mi servono

dataset = dataset.drop('video', axis=1)

dataset = dataset.drop('window', axis=1)

dataset = dataset.drop('subject', axis=1)

# ------------------------- split of dataset, in order to locate the features

aux1 = "activity\_theta\_"

aux2 = "activity\_alpha\_"

aux3 = "activity\_beta\_"

aux4 = "activity\_gamma\_"

# dichiaro gli array

for i in range(1,33):

  globals()[f"f\_{i}"] = []

for i in range(1,33):

  for j in range(1,5):

    globals()[f"f\_{i}"] += [globals()[f"aux{j}"] + str(i)]

f\_target = ['arousal']

# --------------------------- separating out the features

# dichiaro gli array

for i in range(1,33):

  globals()[f"a\_{i}"] = []

for i in range(1,33):

  globals()[f"a\_{i}"] = dataset.loc[:, globals()[f"f\_{i}"]].values

# Separating out the target

a\_target = dataset.loc[:, f\_target].values

# --------------------------- standardizing the features

# dichiaro gli array

for i in range(1,33):

  globals()[f"n\_{i}"] = []

for i in range(1,33):

  globals()[f"n\_{i}"] = StandardScaler().fit\_transform( globals()[f"a\_{i}"] )

# Standardizing the target

n\_target = StandardScaler().fit\_transform(a\_target)

# ----------------------------- iniziamo la PCA effettiva

from sklearn.decomposition import PCA

pca = PCA(n\_components=1)

# dichiaro gli array

for i in range(1,33):

  globals()[f"p\_{i}"] = []

for i in range(1,33):

    globals()[f"p\_{i}"] = pca.fit\_transform( globals()[f"n\_{i}"] )

target = pca.fit\_transform(n\_target)

#ricomponiamo il dataset

df\_1 = pd.DataFrame(data = p\_1, columns = ['p\_1'])

df\_2 = pd.DataFrame(data = p\_2, columns = ['p\_2'])

df\_3 = pd.DataFrame(data = p\_3, columns = ['p\_3'])

df\_4 = pd.DataFrame(data = p\_4, columns = ['p\_4'])

df\_5 = pd.DataFrame(data = p\_5, columns = ['p\_5'])

df\_6 = pd.DataFrame(data = p\_6, columns = ['p\_6'])

df\_7 = pd.DataFrame(data = p\_7, columns = ['p\_7'])

df\_8 = pd.DataFrame(data = p\_8, columns = ['p\_8'])

df\_9 = pd.DataFrame(data = p\_9, columns = ['p\_9'])

df\_10 = pd.DataFrame(data = p\_10, columns = ['p\_10'])

df\_11 = pd.DataFrame(data = p\_11, columns = ['p\_11'])

df\_12 = pd.DataFrame(data = p\_12, columns = ['p\_12'])

df\_13 = pd.DataFrame(data = p\_13, columns = ['p\_13'])

df\_14 = pd.DataFrame(data = p\_14, columns = ['p\_14'])

df\_15 = pd.DataFrame(data = p\_15, columns = ['p\_15'])

df\_16 = pd.DataFrame(data = p\_16, columns = ['p\_16'])

df\_17 = pd.DataFrame(data = p\_17, columns = ['p\_17'])

df\_18 = pd.DataFrame(data = p\_18, columns = ['p\_18'])

df\_19 = pd.DataFrame(data = p\_19, columns = ['p\_19'])

df\_20 = pd.DataFrame(data = p\_20, columns = ['p\_20'])

df\_21 = pd.DataFrame(data = p\_21, columns = ['p\_21'])

df\_22 = pd.DataFrame(data = p\_22, columns = ['p\_22'])

df\_23 = pd.DataFrame(data = p\_23, columns = ['p\_23'])

df\_24 = pd.DataFrame(data = p\_24, columns = ['p\_24'])

df\_25 = pd.DataFrame(data = p\_25, columns = ['p\_25'])

df\_26 = pd.DataFrame(data = p\_26, columns = ['p\_26'])

df\_27 = pd.DataFrame(data = p\_27, columns = ['p\_27'])

df\_28 = pd.DataFrame(data = p\_28, columns = ['p\_28'])

df\_29 = pd.DataFrame(data = p\_29, columns = ['p\_29'])

df\_30 = pd.DataFrame(data = p\_30, columns = ['p\_30'])

df\_31 = pd.DataFrame(data = p\_31, columns = ['p\_31'])

df\_32 = pd.DataFrame(data = p\_32, columns = ['p\_32'])

# trasformo i valori da continui ad interi, altrimenti dà errore

df\_target = pd.DataFrame(data = target, columns = ['target']).round(0)

data = pd.concat([df\_1, df\_2, df\_3, df\_4, df\_5,df\_6, df\_7, df\_8, df\_9, df\_10,

                  df\_11, df\_12, df\_13, df\_14, df\_15,df\_16, df\_17, df\_18, df\_19, df\_20,

                  df\_21, df\_22, df\_23, df\_24, df\_25,df\_26, df\_27, df\_28, df\_29, df\_30,

                  df\_31, df\_32, df\_target], axis = 1)

# eseguo la classificazione su "valence", oppure "arousal"

exp\_reg101 = setup(data = data, target = 'target', session\_id=123)

# uso extraTreeClassifier

et = create\_model('et')

# valuto il modello

evaluate\_model(et)

predict\_model(et);

final\_et = finalize\_model(et)

print(final\_et)