1. Read and Process Data

from glob import glob

import os

import mne

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

#read all file

all\_files\_path=glob('C:/Users/Axioo/.conda/envs/chb-mit-scalp-eeg-database-1.0.0/chb01/\*.edf')

print(len(all\_files\_path))

all\_files\_path[0]

healthy\_file\_path=[i for i in all\_files\_path if  'n' in i.split('\\')[1]]

patient\_file\_path=[i for i in all\_files\_path if  'e' in i.split('\\')[1]]

def read\_data(file\_path):

    datax=mne.io.read\_raw\_edf(file\_path,preload=True)

    datax.set\_eeg\_reference()

    datax.filter(l\_freq=1,h\_freq=45)

    epochs=mne.make\_fixed\_length\_epochs(datax,duration=25,overlap=0)

    epochs=epochs.get\_data()

    return epochs #trials,channel,length

data=read\_data(healthy\_file\_path[0])

data.shape

control\_epochs\_array=[read\_data(subject) for subject in healthy\_file\_path]

patients\_epochs\_array=[read\_data(subject) for subject in patient\_file\_path]

control\_epochs\_labels=[len(i)\*[0] for i in control\_epochs\_array]

patients\_epochs\_labels=[len(i)\*[1] for i in patients\_epochs\_array]

print(len(control\_epochs\_labels),len(patients\_epochs\_labels))

data\_list=control\_epochs\_array+patients\_epochs\_array

label\_list=control\_epochs\_labels+patients\_epochs\_labels

print(len(data\_list),len(label\_list))

epochs\_array=control\_epochs\_array+patients\_epochs\_array

epochs\_labels=control\_epochs\_labels+patients\_epochs\_labels

print(len(epochs\_array),len(epochs\_labels))

groups=[[i]\*len(j) for i, j in enumerate(epochs\_array)]

len(groups)

1. Machine learning
2. from scipy import stats
3. def mean(data):
4. return np.mean(data,axis=-1)
6. def std(data):
7. return np.std(data,axis=-1)
8. def ptp(data):
9. return np.ptp(data,axis=-1)
10. def var(data):
11. return np.var(data,axis=-1)
12. def minim(data):
13. return np.min(data,axis=-1)
14. def maxim(data):
15. return np.max(data,axis=-1)
16. def argminim(data):
17. return np.argmin(data,axis=-1)
18. def argmaxim(data):
19. return np.argmax(data,axis=-1)
20. def mean\_square(data):
21. return np.mean(data\*\*2,axis=-1)
22. def rms(data): #root mean square
23. return  np.sqrt(np.mean(data\*\*2,axis=-1))
24. def abs\_diffs\_signal(data):
25. return np.sum(np.abs(np.diff(data,axis=-1)),axis=-1)
26. def skewness(data):
27. return stats.skew(data,axis=-1)
28. def kurtosis(data):
29. return stats.kurtosis(data,axis=-1)

mean(data).shape

def concatenate\_features(data):

    return np.concatenate((mean(data),std(data),ptp(data),var(data),minim(data),maxim(data),argminim(data),argmaxim(data),

                          mean\_square(data),rms(data),abs\_diffs\_signal(data),

                          skewness(data),kurtosis(data)),axis=-1)

from tqdm import tqdm\_notebook

features=[]

for data in tqdm\_notebook(epochs\_array):

    features.append(concatenate\_features(data))

features=np.concatenate(features)

labels=np.concatenate(epochs\_labels)

groups=np.concatenate(groups)

print('features shape', features.shape)

print('labels shape', labels.shape)

print('groups shape', groups.shape)

Classification

from sklearn.linear\_model import LogisticRegression

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import GroupKFold, GridSearchCV, cross\_val\_score, cross\_validate

clf = LogisticRegression()

gkf = GroupKFold(n\_splits=5)

param\_grid = {'classifier\_\_C' : [0.01,0.05,0.1,0.5, 1,2,3,4,5,8, 10,12,15]}

pipe = Pipeline([('scaler', StandardScaler()), ('classifier' , clf)])

gscv = GridSearchCV(pipe, param\_grid, cv=gkf, n\_jobs=16)

gscv.fit(features,labels, groups=groups)

gscv.best\_score\_

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

clf=GradientBoostingClassifier(random\_state=21)

param\_grid = {

    'classifier\_\_n\_estimators' : [25, 50, 75, 100, 200],

    'classifier\_\_learning\_rate' : [0.005, 0.05, 0.5, 1.5],

    'classifier\_\_max\_depth' : [2, 4, 6, 8],

    'classifier\_\_max\_features' : [10, 12, 17]

    }

gkf=GroupKFold(n\_splits=5)

pipe=Pipeline([('scaler', StandardScaler()), ('classifier', clf)])

gscv=GridSearchCV(pipe, param\_grid, cv=gkf, n\_jobs=16)

gscv.fit(features, labels, groups=groups)

gscv.best\_score\_

3. Deep Learning CNN

epochs\_array=np.vstack(epochs\_array)

epochs\_labels=np.hstack(epochs\_labels)

groups\_array=np.hstack(groups)

epochs\_array.shape, epochs\_labels.shape, groups\_array.shape

epochs\_array = np.moveaxis(epochs\_array, 1, 2)

epochs\_array.shape

from tensorflow.keras.layers import Conv1D,BatchNormalization,LeakyReLU,MaxPool1D,\

GlobalAveragePooling1D,Dense,Dropout,AveragePooling1D

from tensorflow.keras.models import Sequential

from tensorflow.keras.backend import clear\_session

def cnnmodel():

    clear\_session()

    model=Sequential()

    model.add(Conv1D(filters=5,kernel\_size=3,strides=1,input\_shape=(6400,23)))#1

    model.add(BatchNormalization())

    model.add(LeakyReLU())

    model.add(MaxPool1D(pool\_size=2,strides=2))#2

    model.add(Conv1D(filters=5,kernel\_size=3,strides=1))#3

    model.add(LeakyReLU())

    model.add(MaxPool1D(pool\_size=2,strides=2))#4

    model.add(Dropout(0.5))

    model.add(Conv1D(filters=5,kernel\_size=3,strides=1))#5

    model.add(LeakyReLU())

    model.add(AveragePooling1D(pool\_size=2,strides=2))#6

    model.add(Dropout(0.5))

    model.add(Conv1D(filters=5,kernel\_size=3,strides=1))#7

    model.add(LeakyReLU())

    model.add(AveragePooling1D(pool\_size=2,strides=2))#8

    model.add(Conv1D(filters=5,kernel\_size=3,strides=1))#9

    model.add(LeakyReLU())

    model.add(GlobalAveragePooling1D())#10

    model.add(Dense(1,activation='sigmoid'))#11

    model.compile('adam',loss='binary\_crossentropy',metrics=['accuracy'])

    return model

model=cnnmodel()

model.summary()