**DOCUMENTATION AND SUMMARY**

**Table of Contents**

[1.Objective: 1](#_Toc155701961)

[2.Data Loading: 1](#_Toc155701962)

[3.Data Aggregation and Averaging: 2](#_Toc155701963)

[4.Saving Results: 2](#_Toc155701964)

[5.Aggregating Results Across Subjects: 2](#_Toc155701965)

[6.Plotting: 2](#_Toc155701966)

[7.Folder Management: 2](#_Toc155701967)

[Section 1: Data Loading 3](#_Toc155701968)

[Section 2: Data Aggregation and Averaging 3](#_Toc155701969)

[Section 3: Saving Results 3](#_Toc155701970)

[Section 4: Aggregating Results Across Subjects 3](#_Toc155701971)

[Section 5: Plotting 3](#_Toc155701972)

[Section 6: Folder Management 4](#_Toc155701973)

**Dataset Description:**

Recordings, grouped into 23 cases, were collected from 22 subjects (5 males, ages 3–22; and 17 females, ages 1.5–19). (Case chb21 was obtained 1.5 years after case chb01, from the same female subject.) The file SUBJECT-INFO contains the gender and age of each subject. (Case chb24 was added to this collection in December 2010, and is not currently included in SUBJECT-INFO.)

Each case (chb01, chb02, etc.) contains between 9 and 42 continuous .edf files from a single subject. Hardware limitations resulted in gaps between consecutively-numbered .edf files, during which the signals were not recorded; in most cases, the gaps are 10 seconds or less, but occasionally there are much longer gaps. In order to protect the privacy of the subjects, all protected health information (PHI) in the original .edf files has been replaced with surrogate information in the files provided here. Dates in the original .edf files have been replaced by surrogate dates, but the time relationships between the individual files belonging to each case have been preserved. In most cases, the .edf files contain exactly one hour of digitized EEG signals, although those belonging to case chb10 are two hours long, and those belonging to cases chb04, chb06, chb07, chb09, and chb23 are four hours long; occasionally, files in which seizures are recorded are shorter.

All signals were sampled at 256 samples per second with 16-bit resolution. Most files contain 23 EEG signals (24 or 26 in a few cases). The International 10-20 system of EEG electrode positions and nomenclature was used for these recordings. In a few records, other signals are also recorded, such as an ECG signal in the last 36 files belonging to case chb04 and a vagal nerve stimulus (VNS) signal in the last 18 files belonging to case chb09. In some cases, up to 5 “dummy” signals (named "-") were interspersed among the EEG signals to obtain an easy-to-read display format; these dummy signals can be ignored.

The file RECORDS contains a list of all 664 .edf files included in this collection, and the file RECORDS-WITH-SEIZURES lists the 129 of those files that contain one or more seizures. In all, these records include 198 seizures (182 in the original set of 23 cases); the beginning ([) and end (]) of each seizure is annotated in the .seizure annotation files that accompany each of the files listed in RECORDS-WITH-SEIZURES. In addition, the files named chbnn-summary.txt contain information about the montage used for each recording, and the elapsed time in seconds from the beginning of each .edf file to the beginning and end of each seizure contained in it.

**Link:**[**https://colab.research.google.com/drive/1a0egjTe9c0VTGscSHEqaHL1zTAQebbqT?usp=sharing**](https://colab.research.google.com/drive/1a0egjTe9c0VTGscSHEqaHL1zTAQebbqT?usp=sharing)

**Model summary:**

# Multiclass

numSubClass\_Seiz = len(numAddedVecPerClassMulti\_Seiz)

numSubClass\_NonSeiz = len(numAddedVecPerClassMulti\_NonSeiz)

maxLen = np.max([numSubClass\_Seiz, numSubClass\_NonSeiz])

dataToSave = np.ones((2, maxLen)) \* np.nan

dataToSave[0, 0:numSubClass\_Seiz] = numAddedVecPerClassMulti\_Seiz[0:numSubClass\_Seiz]

dataToSave[1, 0:numSubClass\_NonSeiz] = numAddedVecPerClassNonSeiz[0:numSubClass\_NonSeiz]

outputName = folderOut\_ML + '/' + fileName2 + '\_MultiClass\_AddedToEachSubClass.csv'

np.savetxt(outputName, dataToSave.transpose(), delimiter=",")

outputName = folderOut\_ML + '/' + fileName2 + '\_MultiClass\_SeizModelVecs.csv'

np.savetxt(outputName, ModelVectorsMultiNorm\_Seiz.transpose(), delimiter=",")

outputName = folderOut\_ML + '/' + fileName2 + '\_MultiClass\_NonSeizModelVecs.csv'

np.savetxt(outputName, ModelVectorsMultiNorm\_NonSeiz.transpose(), delimiter=",")

# Multiclass reduced

numSubClassMultiRed\_Seiz = len(ModelVectorsMulti\_Seiz\_red[:, 0])

numSubClassMultiRed\_NonSeiz = len(ModelVectorsMulti\_NonSeiz\_red[:, 0])

dataToSave = np.ones((2, maxLen)) \* np.nan

dataToSave[0, 0:numSubClassMultiRed\_Seiz] = numAddedVecPerClass\_Seiz\_red[0:numSubClassMultiRed\_Seiz]

dataToSave[1, 0:numSubClassMultiRed\_NonSeiz] = numAddedVecPerClassNonSeiz\_red[0:numSubClassMultiRed\_NonSeiz]

outputName = folderOut\_ML + '/' + fileName2 + '\_MultiClassReduced\_AddedToEachSubClass.csv'

np.savetxt(outputName, dataToSave.transpose(), delimiter=",")

outputName = folderOut\_ML + '/' + fileName2 + '\_MultiClassReduced\_SeizModelVecs.csv'

np.savetxt(outputName, ModelVectorsMultiNorm\_Seiz\_red.transpose(), delimiter=",")

outputName = folderOut\_ML + '/' + fileName2 + '\_MultiClassReduced\_NonSeizModelVecs.csv'

np.savetxt(outputName, ModelVectorsMultiNorm\_NonSeiz\_red.transpose(), delimiter=",")

# Multiclass clustered

numSubClassMultiClust\_Seiz = len(ModelVectorsMulti\_Seiz\_clust[:, 0])

numSubClassMultiClust\_NonSeiz = len(ModelVectorsMulti\_NonSeiz\_clust[:, 0])

dataToSave = np.ones((2, maxLen)) \* np.nan

dataToSave[0, 0:numSubClassMultiClust\_Seiz] = numAddedVecPerClass\_Seiz\_clust[0:numSubClassMultiClust\_Seiz]

dataToSave[1, 0:numSubClassMultiClust\_NonSeiz] = numAddedVecPerClassNonSeiz\_clust[0:numSubClassMultiClust\_NonSeiz]

outputName = folderOut\_ML + '/' + fileName2 + '\_MultiClassClustered\_AddedToEachSubClass.csv'

np.savetxt(outputName, dataToSave.transpose(), delimiter=",")

outputName = folderOut\_ML + '/' + fileName2 + '\_MultiClassClustered\_SeizModelVecs.csv'

np.savetxt(outputName, ModelVectorsMultiNorm\_Seiz\_clust.transpose(), delimiter=",")

outputName = folderOut\_ML + '/' + fileName2 + '\_MultiClassClustered\_NonSeizModelVecs.csv'

np.savetxt(outputName, ModelVectorsMultiNorm\_NonSeiz\_clust.transpose(), delimiter=",")

# Saving performance per subject

# (Code for saving optimal values and results for various approaches is provided)

# Plot performances for each subject for each approach and all CV

performancessAll\_train = np.dstack((AllRes\_train, AllResMulti\_train, AllResMultiRed\_train, AllResMultiClust\_train))

func\_plotPerformancesOfDiffApproaches\_thisSubj\_multiClassPaper(pat, 'TrainRes', performancessAll\_train, folderOutPredictionsPlot)

performancessAll\_test = np.dstack((AllRes\_test, AllResMulti\_test, AllResMultiRed\_test, AllResMultiClust\_test))

func\_plotPerformancesOfDiffApproaches\_thisSubj\_multiClassPaper(pat, 'TestRes', performancessAll\_test, folderOutPredictionsPlot)

# Save average for this subject

# (Code for saving average results and standard deviations for various approaches is provided)

# Saving performance for all subjects

# (Code for saving optimal values and results for various approaches is provided)

# Mean of all subjects

# (Code for calculating and saving mean results and standard deviations for various approaches is provided)

**Dataset link:** [**https://www.kaggle.com/datasets/abhishekinnvonix/seizure-epilepcy-chb-mit-eeg-dataset-pediatric**](https://www.kaggle.com/datasets/abhishekinnvonix/seizure-epilepcy-chb-mit-eeg-dataset-pediatric)

# 1.Objective:

The code appears to be focused on loading, aggregating, and analyzing performance results from multiple models and datasets, particularly in the context of multiclass classification. The script performs data processing, aggregation, saving results to CSV files, and generates various plots for performance comparison.

**Key Sections:**

# 2.Data Loading:

Reads results from CSV files for different models and datasets.

Converts data into NumPy arrays for further analysis.

# 3.Data Aggregation and Averaging:

Calculates mean and standard deviation of performance metrics across models and datasets for each subject.

Aggregates results for different models, datasets, and subclasses.

# 4.Saving Results:

Saves aggregated results (mean and standard deviation) to CSV files.

# 5.Aggregating Results Across Subjects:

Calculates mean and standard deviation of performance metrics across all subjects for each model and dataset.

Saves aggregated values to CSV files.

# 6.Plotting:

Generates various plots to compare models, visualize subclass distributions, and analyze performance during subclass removal or clustering.

# 7.Folder Management:

Ensures required folders for saving plots are created.

# Section 1: Data Loading

**reader = csv.reader(open(outputName, "r"))**

**OptimalValuesClust\_test = np.array(list(reader)).astype("float")**

This section loads data from various CSV files, representing training and testing results for different models and datasets. The csv.reader is used to read the contents of the CSV files, and the data is converted to NumPy arrays (OptimalValuesClust\_test, AllRes\_train, etc.) for further processing.

# Section 2: Data Aggregation and Averaging

AllSubjRes\_train[patIndx,:,0] = np.nanmean(AllRes\_train, 0)

Here, the code calculates the mean and standard deviation of performance metrics across different models and datasets for each subject. The results are stored in arrays (AllSubjRes\_train, AllSubjRes\_test, etc.).

# Section 3: Saving Results

np.savetxt(outputName, AllSubj\_OptimalResultsRed\_train[:,:,ni], delimiter=",")

This part of the code saves the aggregated results, including mean and standard deviation values, to CSV files for each subject and model.

# Section 4: Aggregating Results Across Subjects

TotalMean\_2class\_train[0,:] = np.nanmean(AllSubjRes\_train[:,:,0],0)

Here, the script calculates the mean and standard deviation of performance metrics across all subjects for each model and dataset. The aggregated values are saved to CSV files.

# Section 5: Plotting

funct\_plotAllPerformancesForManyApproaches(dataToPlotMean\_train, dataToPlotMean\_test, xLabNames, folderOut\_ML)

# Repeat similar function calls for different plots

This section includes various functions for generating plots, comparing models, and visualizing subclass distributions. It also covers plotting performance during iterative removal or clustering of subclasses.

# Section 6: Folder Management

createFolderIfNotExists(folderOutPredictionsPlots)

# Repeat similar folder creation for different folders

The script ensures that the required folders for saving plots are created. It uses the createFolderIfNotExists function.

|  |
| --- |
| from HDfunctionsLib import \* |
|  |  |  | from parametersSetup import \* |
|  |  |  | from scipy import interpolate |
|  |  |  |  |
|  |  |  | ################################################################################# |
|  |  |  | #SETUPS |
|  |  |  | GeneralParams.plottingON=0 |
|  |  |  | GeneralParams.PersGenApproach='personalized' |
|  |  |  | datasetPreparationType='MoreNonSeizure\_Fact5' # 'MoreNonSeizure\_Fact5' , 'MoreNonSeizure\_Fact10' |
|  |  |  | torch.cuda.set\_device(HDParams.CUDAdevice) |
|  |  |  | HDParams.D=10000 |
|  |  |  |  |
|  |  |  | optType = 'F1DEgmean' # 'simpleAcc', 'F1DEgmean' |
|  |  |  | #MULTI CLASS PARAMS |
|  |  |  | numSteps = 10 |
|  |  |  | groupingThresh = 0.95 |
|  |  |  | subClassReductApproachType = 'clustering' # 'removing', 'clustering' |
|  |  |  | perfDropThr=0.03 #0.01, 0.02, 0.03 |
|  |  |  | #ITTERATIVE LEARNING |
|  |  |  | ItterType='AddAndSubtract' #'AddAndSubtract', 'AddOnly' |
|  |  |  | ItterFact=1 |
|  |  |  | ItterImprovThresh=0.01 #if in threec consecutive runs not bigger improvement then this then stop |
|  |  |  | savingStepData=1 #whether to save improvements per each itteration |
|  |  |  |  |
|  |  |  | #DATASET |
|  |  |  | Dataset='01\_CHBMIT' |
|  |  |  | GeneralParams.patients =['01','02','03','04','05','06','07','08','09','10','11','12','13','14','15','16','17','18','19','20','21','22','23','24'] |
|  |  |  | GeneralParams.patients =['01','02','03'] |
|  |  |  |  |
|  |  |  |  |
|  |  |  | # DEFINING INPUT/OUTPUT FOLDERS |
|  |  |  | folderIn = '01\_datasetProcessed\_'+datasetPreparationType+'/' |
|  |  |  | folderOut0= '03\_predictions\_' +datasetPreparationType |
|  |  |  | createFolderIfNotExists(folderOut0) |
|  |  |  | # folderOut0=folderOut0 +'/'+ str(GeneralParams.PersGenApproach)+'/' |
|  |  |  | # createFolderIfNotExists(folderOut0) |
|  |  |  | folderFeaturesOut='02\_features\_'+datasetPreparationType |
|  |  |  | createFolderIfNotExists(folderFeaturesOut) |
|  |  |  | # folderFeaturesOut0=folderFeaturesOut0 +'/'+ str(GeneralParams.PersGenApproach)+'/' |
|  |  |  | # createFolderIfNotExists(folderFeaturesOut0) |
|  |  |  |  |
|  |  |  | # FEATURS USED - STANDARD ML FEATURES - 45 FEAT |
|  |  |  | HDParams.HDapproachON=1 |
|  |  |  | SegSymbParams.symbolType ='StandardMLFeatures' |
|  |  |  | HDParams.numFeat=45 |
|  |  |  | SegSymbParams.numSegLevels=20 #num dicretized windows for feature values |
|  |  |  | SegSymbParams.segLenSec = 8 #8 # length of EEG sements in sec |
|  |  |  | SegSymbParams.slidWindStepSec = 1 #1 # step of sliding window to extract segments in sec |
|  |  |  | HDParams.vectorTypeLevel = 'scaleNoRand1' # 'random','sandwich','scaleNoRand1','scaleNoRand2','scaleRand1', ,'scaleRand2' |
|  |  |  | HDParams.vectorTypeCh = 'random' # 'random','sandwich','scaleNoRand1','scaleNoRand2','scaleRand1', ,'scaleRand2' |
|  |  |  | HDParams.vectorTypeFeat='random' |
|  |  |  | HDParams.roundingTypeForHDVectors='inSteps' #'inSteps','onlyOne','noRounding' |
|  |  |  | HDParams.bindingFeatures='FeatxVal' #'FeatxVal', 'ChxFeatxVal', 'FeatxChxVal', 'ChFeatCombxVal', 'FeatAppend1000' |
|  |  |  | HDParams.D=10000 |
|  |  |  | #HDParams.ItterativeRelearning='on' |
|  |  |  |  |
|  |  |  | #calculating various parameters |
|  |  |  | seizureStableLenToTestIndx = int(GeneralParams.seizureStableLenToTest / SegSymbParams.slidWindStepSec) |
|  |  |  | seizureStablePercToTest = GeneralParams.seizureStablePercToTest |
|  |  |  | distanceBetweenSeizuresIndx = int(GeneralParams.distanceBetween2Seizures / SegSymbParams.slidWindStepSec) |
|  |  |  | numLabelsPerHour = 60 \* 60 / SegSymbParams.slidWindStepSec |
|  |  |  | toleranceFP\_bef = int(GeneralParams.toleranceFP\_befSeiz / SegSymbParams.slidWindStepSec) |
|  |  |  | toleranceFP\_aft = int(GeneralParams.toleranceFP\_aftSeiz / SegSymbParams.slidWindStepSec) |
|  |  |  |  |
|  |  |  |  |
|  |  |  | # #saving parameters to folder name |
|  |  |  | # folderOutName = SegSymbParams.symbolType +'\_'+ str(HDParams.numFeat)+ '\_' + str(SegSymbParams.numSegLevels) + '\_numFeat' + str( |
|  |  |  | # HDParams.numFeat) + '\_' + HDParams.bindingFeatures + '\_FEATvec' + HDParams.vectorTypeFeat |
|  |  |  | # folderOutNameFeat = SegSymbParams.symbolType + '\_'+ str(HDParams.numFeat) |
|  |  |  | # folderOutName = folderOutName + '\_' + str(SegSymbParams.segLenSec) + '\_' + str( |
|  |  |  | # SegSymbParams.slidWindStepSec) + 's' + '\_' + HDParams.similarityType + '\_RND' + HDParams.roundingTypeForHDVectors + '\_CHVect' + HDParams.vectorTypeCh + '\_LVLVect' + HDParams.vectorTypeLevel+'\_D'+str(HDParams.D) |
|  |  |  | # folderOutNameFeat =folderOutNameFeat+ '\_' + str(SegSymbParams.segLenSec) + '\_' + str(SegSymbParams.slidWindStepSec) + 's' |
|  |  |  | # folderOutName=folderOutName+'\_MultiClassPaper' |
|  |  |  | # folderFeaturesOut = folderFeaturesOut0 + folderOutNameFeat |
|  |  |  | # folderOut\_ML = folderOut0 + folderOutName |
|  |  |  | # createFolderIfNotExists(folderOut\_ML) |
|  |  |  |  |
|  |  |  | #final folder to store data to |
|  |  |  | folderOut\_ML =folderOut0 +'/'+optType+'\_'+ str(perfDropThr) +'\_'+ str(numSteps) |
|  |  |  | createFolderIfNotExists(folderOut\_ML) |
|  |  |  | print('FOLDER OUT:', folderOut\_ML) |
|  |  |  | print('FOLDER OUT FEATURES:', folderFeaturesOut) |
|  |  |  | folderOutPredictionsPlot = folderOut\_ML+'/Plots\_predictions' |
|  |  |  | createFolderIfNotExists(folderOutPredictionsPlot) |
|  |  |  |  |
|  |  |  |  |
|  |  |  | ################################################################################# |
|  |  |  | ## CALCULATING FEATURES FOR EACH FILE |
|  |  |  | numFiles = len(np.sort(glob.glob(folderFeaturesOut + '/\*chb' + '\*.csv'))) |
|  |  |  | if (numFiles==0): |
|  |  |  | print('EXTRACTING FEATURES!!!') |
|  |  |  | func\_calculateFeaturesForInputFiles(SigInfoParams, SegSymbParams, GeneralParams, HDParams, folderIn, folderFeaturesOut) |
|  |  |  |  |
|  |  |  |  |
|  |  |  | ################################################################################# |
|  |  |  | ## TRAINING |
|  |  |  | AllSubjRes\_train = np.zeros((len(GeneralParams.patients), 33, 2)) |
|  |  |  | AllSubjRes\_test = np.zeros((len(GeneralParams.patients), 33, 2)) |
|  |  |  | AllSubjResMulti\_train = np.zeros((len(GeneralParams.patients), 33, 2)) |
|  |  |  | AllSubjResMulti\_test = np.zeros((len(GeneralParams.patients),33, 2)) |
|  |  |  | AllSubjResMultiRed\_train = np.zeros((len(GeneralParams.patients), 33, 2)) |
|  |  |  | AllSubjResMultiRed\_test = np.zeros((len(GeneralParams.patients),33, 2)) |
|  |  |  | AllSubjResMultiClust\_train = np.zeros((len(GeneralParams.patients), 33, 2)) |
|  |  |  | AllSubjResMultiClust\_test = np.zeros((len(GeneralParams.patients),33, 2)) |
|  |  |  | AllSubj\_OptimalResultsReduced\_train= np.zeros((len(GeneralParams.patients),34, 2)) #3+3+1+9+9+9 |
|  |  |  | AllSubj\_OptimalResultsReduced\_test= np.zeros((len(GeneralParams.patients),34, 2)) |
|  |  |  | AllSubj\_OptimalResultsClustered\_train= np.zeros((len(GeneralParams.patients),34, 2)) #3+3+1+9+9+9 |
|  |  |  | AllSubj\_OptimalResultsClustered\_test= np.zeros((len(GeneralParams.patients),34, 2)) |
|  |  |  |  |
|  |  |  | # go through each subject for personalized approach |
|  |  |  | for patIndx, pat in enumerate(GeneralParams.patients): |
|  |  |  | numFiles = len(np.sort(glob.glob(folderFeaturesOut + '/\*chb' + pat + '\*.csv'))) |
|  |  |  | print('-- Patient:', pat, 'NumSeizures:', numFiles) |
|  |  |  |  |
|  |  |  | # go through leave-one-out cross-validations for this subject |
|  |  |  | AllRes\_train=np.zeros((numFiles,33)) |
|  |  |  | AllRes\_test = np.zeros((numFiles, 33)) |
|  |  |  | AllResMulti\_train = np.zeros((numFiles, 33)) |
|  |  |  | AllResMulti\_test = np.zeros((numFiles, 33)) |
|  |  |  | AllResMultiRed\_train = np.zeros((numFiles, 33)) |
|  |  |  | AllResMultiRed\_test = np.zeros((numFiles, 33)) |
|  |  |  | AllResMultiClust\_train = np.zeros((numFiles, 33)) |
|  |  |  | AllResMultiClust\_test = np.zeros((numFiles, 33)) |
|  |  |  | OptimalValues\_train\_red= np.zeros((numFiles, 34)) |
|  |  |  | OptimalValues\_test\_red = np.zeros((numFiles, 34)) |
|  |  |  | OptimalValues\_train\_clust= np.zeros((numFiles, 34)) |
|  |  |  | OptimalValues\_test\_clust = np.zeros((numFiles, 34)) |
|  |  |  | OptimalValuesClustered\_train= np.zeros((numFiles, 34)) |
|  |  |  | OptimalValuesClustered\_test = np.zeros((numFiles, 34)) |
|  |  |  | for cv in range(numFiles): |
|  |  |  | # creates list of files to train and test on |
|  |  |  | filesToTrainOn = [] |
|  |  |  | for fIndx, fileName in enumerate(np.sort(glob.glob(folderFeaturesOut + '/\*chb' + pat + '\*.csv'))): |
|  |  |  | if (fIndx != cv): |
|  |  |  | filesToTrainOn.append(fileName) |
|  |  |  | else: |
|  |  |  | filesToTestOn = list(fileName.split(" ")) |
|  |  |  | pom, fileName1 = os.path.split(filesToTestOn[0]) |
|  |  |  | fileName2 = os.path.splitext(fileName1)[0] |
|  |  |  |  |
|  |  |  | # concatenating data from more files |
|  |  |  | (dataTrain, label\_train)= concatenateDataFromFiles(filesToTrainOn) |
|  |  |  | (dataTest, label\_test) = concatenateDataFromFiles(filesToTestOn) |
|  |  |  |  |
|  |  |  | # normalizing data and discretizing |
|  |  |  | (data\_train\_Norm, data\_test\_Norm, data\_train\_Discr, data\_test\_Discr)=normalizeAndDiscretizeTrainAndTestData(dataTrain, dataTest, SegSymbParams.numSegLevels) |
|  |  |  | data\_train\_Discr=data\_train\_Discr.astype(int) |
|  |  |  | data\_test\_Discr = data\_test\_Discr.astype(int) |
|  |  |  |  |
|  |  |  | # INITIALIZING HD VECTORS |
|  |  |  | model = HD\_classifier\_GeneralAndNoCh(SigInfoParams, SegSymbParams, HDParams, HDParams.numFeat\*len(SigInfoParams.chToKeep)) |
|  |  |  | #model = HD\_classifier\_GeneralWithChCombinations(SigInfoParams, SegSymbParams, HDParams, len(SigInfoParams.chToKeep)) |
|  |  |  |  |
|  |  |  | ################# |
|  |  |  | #STANDARD SINGLE PASS 2 CLASS LEARNING |
|  |  |  | #learn on trainin set |
|  |  |  | (ModelVectors, ModelVectorsNorm, numAddedVecPerClass, numLabels) = trainModelVecOnData(data\_train\_Discr, label\_train, model, HDParams) |
|  |  |  | #measure performance on test set |
|  |  |  | (AllRes\_train[cv,:], AllRes\_test[cv,:], predLabelsTrain\_2class, predLabelsTest\_2class)= testModelsAndReturnAllPerformances\_2class(data\_train\_Discr, label\_train, data\_test\_Discr, label\_test, model, |
|  |  |  | ModelVectorsNorm, HDParams, GeneralParams, SegSymbParams) |
|  |  |  | print('2 CLASS acc\_train: ', AllRes\_train[cv,2], 'acc\_test: ', AllRes\_test[cv,2]) |
|  |  |  |  |
|  |  |  | ################# |
|  |  |  | #MULTICLASS LEARNING |
|  |  |  | # learn on trainin set |
|  |  |  | (ModelVectorsMulti\_Seiz, ModelVectorsMultiNorm\_Seiz, ModelVectorsMulti\_NonSeiz, ModelVectorsMultiNorm\_NonSeiz, |
|  |  |  | numAddedVecPerClassMulti\_Seiz, numAddedVecPerClassMulti\_NonSeiz) =trainModelVecOnData\_Multiclass(data\_train\_Discr, label\_train, model, HDParams) |
|  |  |  | #measure performance on test set |
|  |  |  | (AllResMulti\_train[cv,:], AllResMulti\_test[cv,:], predLabelsTrain\_Multi, predLabelsTest\_Multi)=testModelsAndReturnAllPerformances\_MoreClass(data\_train\_Discr, label\_train, data\_test\_Discr, label\_test, model, |
|  |  |  | ModelVectorsMultiNorm\_Seiz, ModelVectorsMultiNorm\_NonSeiz, HDParams, GeneralParams, SegSymbParams) |
|  |  |  | print('MULTI CLASS acc\_train: ', AllResMulti\_train[cv,2], 'acc\_test: ', AllResMulti\_test[cv,2], 'numSubClass\_Seiz', len(numAddedVecPerClassMulti\_Seiz), 'numSubClass\_NonSeiz', len(numAddedVecPerClassMulti\_NonSeiz)) |
|  |  |  |  |
|  |  |  |  |
|  |  |  | ################# |
|  |  |  | #ANALYSE REMOVING LESS CROWDED SUBCLASSES |
|  |  |  | #REMOVING |
|  |  |  | subClassReductApproachType = 'removing' |
|  |  |  | (OptimalValues\_train\_red[cv,:], OptimalValues\_test\_red[cv,:], ModelVectorsMulti\_Seiz\_red, ModelVectorsMulti\_NonSeiz\_red, ModelVectorsMultiNorm\_Seiz\_red, ModelVectorsMultiNorm\_NonSeiz\_red, numAddedVecPerClass\_Seiz\_red, |
|  |  |  | numAddedVecPerClass\_NonSeiz\_red)=reduceNumSubclasses\_removingApproach(data\_train\_Discr, label\_train,data\_test\_Discr, label\_test, model, HDParams, ModelVectorsMulti\_Seiz, ModelVectorsMulti\_NonSeiz, |
|  |  |  | ModelVectorsMultiNorm\_Seiz, ModelVectorsMultiNorm\_NonSeiz, numAddedVecPerClassMulti\_Seiz, numAddedVecPerClassMulti\_NonSeiz, |
|  |  |  | numSteps, optType, perfDropThr, GeneralParams, SegSymbParams, folderOut\_ML, fileName2) |
|  |  |  | # performance on training and test dataset |
|  |  |  | (AllResMultiRed\_train[cv, :], AllResMultiRed\_test[cv, :], predLabelsTrain\_MultiRed, predLabelsTest\_MultiRed) = testModelsAndReturnAllPerformances\_MoreClass(data\_train\_Discr, label\_train, data\_test\_Discr, label\_test, model, |
|  |  |  | ModelVectorsMultiNorm\_Seiz\_red, ModelVectorsMultiNorm\_NonSeiz\_red, HDParams, GeneralParams, SegSymbParams) |
|  |  |  | print('MULTI CLASS REDUCED acc\_train: ', AllResMultiRed\_train[cv, 2], 'acc\_test: ', AllResMultiRed\_test[cv, 2], 'numSubClass\_Seiz', len(ModelVectorsMulti\_Seiz\_red[:,0]), 'numSubClass\_NonSeiz', len(ModelVectorsMulti\_NonSeiz\_red[:,0])) |
|  |  |  |  |
|  |  |  | #CLUSTERING |
|  |  |  | subClassReductApproachType = 'clustering' |
|  |  |  | (OptimalValues\_train\_clust[cv,:], OptimalValues\_test\_clust[cv,:], ModelVectorsMulti\_Seiz\_clust, ModelVectorsMulti\_NonSeiz\_clust, ModelVectorsMultiNorm\_Seiz\_clust,ModelVectorsMultiNorm\_NonSeiz\_clust, numAddedVecPerClass\_Seiz\_clust, |
|  |  |  | numAddedVecPerClass\_NonSeiz\_clust) = reduceNumSubclasses\_clusteringApproach(data\_train\_Discr, label\_train,data\_test\_Discr,label\_test, model, HDParams, ModelVectorsMulti\_Seiz, ModelVectorsMulti\_NonSeiz, |
|  |  |  | ModelVectorsMultiNorm\_Seiz, ModelVectorsMultiNorm\_NonSeiz, numAddedVecPerClassMulti\_Seiz, numAddedVecPerClassMulti\_NonSeiz, |
|  |  |  | numSteps, optType, perfDropThr, groupingThresh, GeneralParams, SegSymbParams, folderOut\_ML, fileName2) |
|  |  |  | # performance on training and test dataset |
|  |  |  | (AllResMultiClust\_train[cv, :], AllResMultiClust\_test[cv, :], predLabelsTrain\_MultiClust, predLabelsTest\_MultiClust) = testModelsAndReturnAllPerformances\_MoreClass(data\_train\_Discr, label\_train, data\_test\_Discr, label\_test, model, |
|  |  |  | ModelVectorsMultiNorm\_Seiz\_clust, ModelVectorsMultiNorm\_NonSeiz\_clust, HDParams, GeneralParams, SegSymbParams) |
|  |  |  | print('MULTI CLASS CLUSTER acc\_train: ', AllResMultiClust\_train[cv, 2], 'acc\_test: ', AllResMultiClust\_test[cv, 2], 'numSubClass\_Seiz', len(ModelVectorsMulti\_Seiz\_clust[:,0]), 'numSubClass\_NonSeiz', len(ModelVectorsMulti\_NonSeiz\_clust[:,0])) |
|  |  |  |  |
|  |  |  |  |
|  |  |  | #SAVE PREDICTIONS FOR ALL APPROACHES |
|  |  |  | dataToSave\_train=np.vstack((label\_train, predLabelsTrain\_2class, predLabelsTrain\_Multi, predLabelsTrain\_MultiRed, predLabelsTrain\_MultiClust )).transpose() |
|  |  |  | outputName = folderOut\_ML + '/' + fileName2 + '\_AllApproaches\_TrainPredictions.csv' |
|  |  |  | np.savetxt(outputName, dataToSave\_train, delimiter=",") |
|  |  |  | dataToSave\_test=np.vstack((label\_test, predLabelsTest\_2class, predLabelsTest\_Multi, predLabelsTest\_MultiRed, predLabelsTest\_MultiClust )).transpose() |
|  |  |  | outputName = folderOut\_ML + '/' + fileName2 + '\_AllApproaches\_TestPredictions.csv' |
|  |  |  | np.savetxt(outputName, dataToSave\_test, delimiter=",") |
|  |  |  | #plot predictions for test |
|  |  |  | approachNames = ['2C', 'MC', 'MCred', 'MCclust'] |
|  |  |  | approachIndx = [1, 2, 4, 6] |
|  |  |  | func\_plotRawSignalAndPredictionsOfDiffApproaches\_thisFile(fileName2, dataToSave\_test,dataToSave\_train, approachNames, approachIndx, folderIn, folderOutPredictionsPlot, SigInfoParams, GeneralParams, SegSymbParams) |
|  |  |  |  |
|  |  |  | #SAVE MODEL VECTORS |
|  |  |  | #standard learning |
|  |  |  | outputName = folderOut\_ML + '/' + fileName2 + '\_StandardLearning\_ModelVecsNorm.csv' #first nonSeiz, then Seiz |
|  |  |  | np.savetxt(outputName, ModelVectorsNorm.transpose(), delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/' + fileName2 + '\_StandardLearning\_ModelVecs.csv' #first nonSeiz, then Seiz |
|  |  |  | np.savetxt(outputName, ModelVectors.transpose(), delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/' + fileName2 + '\_StandardLearning\_AddedToEachSubClass.csv' |
|  |  |  | np.savetxt(outputName, numAddedVecPerClass, delimiter=",") |
|  |  |  | #multiclass |
|  |  |  | numSubClass\_Seiz= len(numAddedVecPerClassMulti\_Seiz) |
|  |  |  | numSubClass\_NonSeiz = len(numAddedVecPerClassMulti\_NonSeiz) |
|  |  |  | maxLen=np.max([numSubClass\_Seiz,numSubClass\_NonSeiz ] ) |
|  |  |  | dataToSave=np.ones((2,maxLen))\*np.nan |
|  |  |  | dataToSave[0,0:numSubClass\_Seiz]=numAddedVecPerClassMulti\_Seiz[0:numSubClass\_Seiz] |
|  |  |  | dataToSave[1, 0:numSubClass\_NonSeiz] = numAddedVecPerClassMulti\_NonSeiz[0:numSubClass\_NonSeiz] |
|  |  |  | outputName = folderOut\_ML + '/' + fileName2 + '\_MultiClass\_AddedToEachSubClass.csv' |
|  |  |  | np.savetxt(outputName, dataToSave.transpose(), delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/' + fileName2 + '\_MultiClass\_SeizModelVecs.csv' |
|  |  |  | np.savetxt(outputName, ModelVectorsMultiNorm\_Seiz.transpose(), delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/' + fileName2 + '\_MultiClass\_NonSeizModelVecs.csv' |
|  |  |  | np.savetxt(outputName, ModelVectorsMultiNorm\_NonSeiz.transpose(), delimiter=",") |
|  |  |  | #multiclass reduced |
|  |  |  | numSubClassMultiRed\_Seiz=len(ModelVectorsMulti\_Seiz\_red[:,0]) |
|  |  |  | numSubClassMultiRed\_NonSeiz = len(ModelVectorsMulti\_NonSeiz\_red[:, 0]) |
|  |  |  | dataToSave=np.ones((2,maxLen))\*np.nan |
|  |  |  | dataToSave[0,0:numSubClassMultiRed\_Seiz]=numAddedVecPerClass\_Seiz\_red[0:numSubClassMultiRed\_Seiz] |
|  |  |  | dataToSave[1, 0:numSubClassMultiRed\_NonSeiz] = numAddedVecPerClass\_NonSeiz\_red[0:numSubClassMultiRed\_NonSeiz] |
|  |  |  | outputName = folderOut\_ML + '/' + fileName2 + '\_MultiClassReduced\_AddedToEachSubClass.csv' |
|  |  |  | np.savetxt(outputName, dataToSave.transpose(), delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/' + fileName2 + '\_MultiClassReduced\_SeizModelVecs.csv' |
|  |  |  | np.savetxt(outputName, ModelVectorsMultiNorm\_Seiz\_red.transpose(), delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/' + fileName2 + '\_MultiClassReduced\_NonSeizModelVecs.csv' |
|  |  |  | np.savetxt(outputName, ModelVectorsMultiNorm\_NonSeiz\_red.transpose(), delimiter=",") |
|  |  |  | #multiclass clustered |
|  |  |  | numSubClassMultiClust\_Seiz=len(ModelVectorsMulti\_Seiz\_clust[:,0]) |
|  |  |  | numSubClassMultiClust\_NonSeiz = len(ModelVectorsMulti\_NonSeiz\_clust[:, 0]) |
|  |  |  | dataToSave=np.ones((2,maxLen))\*np.nan |
|  |  |  | dataToSave[0,0:numSubClassMultiClust\_Seiz]=numAddedVecPerClass\_Seiz\_clust[0:numSubClassMultiClust\_Seiz] |
|  |  |  | dataToSave[1, 0:numSubClassMultiClust\_NonSeiz] = numAddedVecPerClass\_NonSeiz\_clust[0:numSubClassMultiClust\_NonSeiz] |
|  |  |  | outputName = folderOut\_ML + '/' + fileName2 + '\_MultiClassClustered\_AddedToEachSubClass.csv' |
|  |  |  | np.savetxt(outputName, dataToSave.transpose(), delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/' + fileName2 + '\_MultiClassClustered\_SeizModelVecs.csv' |
|  |  |  | np.savetxt(outputName, ModelVectorsMultiNorm\_Seiz\_clust.transpose(), delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/' + fileName2 + '\_MultiClassClustered\_NonSeizModelVecs.csv' |
|  |  |  | np.savetxt(outputName, ModelVectorsMultiNorm\_NonSeiz\_clust.transpose(), delimiter=",") |
|  |  |  |  |
|  |  |  |  |
|  |  |  | #saving performance per subj |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_MultiClassLearning\_OptimalResultsReduced\_Train.csv' |
|  |  |  | np.savetxt(outputName, OptimalValues\_train\_red, delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_MultiClassLearning\_OptimalResultsReduced\_Test.csv' |
|  |  |  | np.savetxt(outputName, OptimalValues\_test\_red, delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_MultiClassLearning\_OptimalResultsClustered\_Train.csv' |
|  |  |  | np.savetxt(outputName, OptimalValues\_train\_clust, delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_MultiClassLearning\_OptimalResultsClustered\_Test.csv' |
|  |  |  | np.savetxt(outputName, OptimalValues\_test\_clust, delimiter=",") |
|  |  |  |  |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_StandardLearning\_TrainRes.csv' |
|  |  |  | np.savetxt(outputName, AllRes\_train, delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_StandardLearning\_TestRes.csv' |
|  |  |  | np.savetxt(outputName, AllRes\_test, delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_MultiClassLearning\_TrainRes.csv' |
|  |  |  | np.savetxt(outputName, AllResMulti\_train, delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_MultiClassLearning\_TestRes.csv' |
|  |  |  | np.savetxt(outputName, AllResMulti\_test, delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_MultiClassReduced\_TrainRes.csv' |
|  |  |  | np.savetxt(outputName, AllResMultiRed\_train, delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_MultiClassReduced\_TestRes.csv' |
|  |  |  | np.savetxt(outputName, AllResMultiRed\_test, delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_MultiClassClustered\_TrainRes.csv' |
|  |  |  | np.savetxt(outputName, AllResMultiClust\_train, delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_MultiClassClustered\_TestRes.csv' |
|  |  |  | np.savetxt(outputName, AllResMultiClust\_test, delimiter=",") |
|  |  |  |  |
|  |  |  |  |
|  |  |  | #plot performances for this subj for each approach and all cv |
|  |  |  | performancessAll=np.dstack((AllRes\_train,AllResMulti\_train, AllResMultiRed\_train,AllResMultiClust\_train )) |
|  |  |  | func\_plotPerformancesOfDiffApproaches\_thisSubj\_multiClassPaper(pat, 'TrainRes', performancessAll, folderOutPredictionsPlot) |
|  |  |  | performancessAll = np.dstack((AllRes\_test, AllResMulti\_test, AllResMultiRed\_test, AllResMultiClust\_test)) |
|  |  |  | func\_plotPerformancesOfDiffApproaches\_thisSubj\_multiClassPaper(pat, 'TestRes', performancessAll, folderOutPredictionsPlot) |
|  |  |  |  |
|  |  |  | #save avrg for this subj |
|  |  |  | AllSubjRes\_train[patIndx,:,0] = np.nanmean(AllRes\_train,0) |
|  |  |  | AllSubjRes\_test[patIndx,:,0] = np.nanmean(AllRes\_test,0) |
|  |  |  | AllSubjResMulti\_train[patIndx,:,0] = np.nanmean(AllResMulti\_train,0) |
|  |  |  | AllSubjResMulti\_test[patIndx,:,0] = np.nanmean(AllResMulti\_test,0) |
|  |  |  | AllSubjResMultiRed\_train[patIndx,:,0] = np.nanmean(AllResMultiRed\_train,0) |
|  |  |  | AllSubjResMultiRed\_test[patIndx,:,0] = np.nanmean(AllResMultiRed\_test,0) |
|  |  |  | AllSubjResMultiClust\_train[patIndx,:,0] = np.nanmean(AllResMultiClust\_train,0) |
|  |  |  | AllSubjResMultiClust\_test[patIndx,:,0] = np.nanmean(AllResMultiClust\_test,0) |
|  |  |  | AllSubj\_OptimalResultsReduced\_train[patIndx,:,0] = np.nanmean(OptimalValues\_train\_red,0) |
|  |  |  | AllSubj\_OptimalResultsReduced\_test[patIndx, :,0] = np.nanmean(OptimalValues\_test\_red, 0) |
|  |  |  | AllSubj\_OptimalResultsClustered\_train[patIndx,:,0] = np.nanmean(OptimalValues\_train\_clust,0) |
|  |  |  | AllSubj\_OptimalResultsClustered\_test[patIndx, :,0] = np.nanmean(OptimalValues\_test\_clust, 0) |
|  |  |  | AllSubjRes\_train[patIndx,:,1] = np.nanstd(AllRes\_train,0) |
|  |  |  | AllSubjRes\_test[patIndx,:,1] = np.nanstd(AllRes\_test,0) |
|  |  |  | AllSubjResMulti\_train[patIndx,:,1] = np.nanstd(AllResMulti\_train,0) |
|  |  |  | AllSubjResMulti\_test[patIndx,:,1] = np.nanstd(AllResMulti\_test,0) |
|  |  |  | AllSubjResMultiRed\_train[patIndx,:,1] = np.nanstd(AllResMultiRed\_train,0) |
|  |  |  | AllSubjResMultiRed\_test[patIndx,:,1] = np.nanstd(AllResMultiRed\_test,0) |
|  |  |  | AllSubjResMultiClust\_train[patIndx,:,1] = np.nanstd(AllResMultiClust\_train,0) |
|  |  |  | AllSubjResMultiClust\_test[patIndx,:,1] = np.nanstd(AllResMultiClust\_test,0) |
|  |  |  | AllSubj\_OptimalResultsReduced\_train[patIndx,:,1] = np.nanstd(OptimalValues\_train\_red,0) |
|  |  |  | AllSubj\_OptimalResultsReduced\_test[patIndx, :,1] = np.nanstd(OptimalValues\_test\_red, 0) |
|  |  |  | AllSubj\_OptimalResultsClustered\_train[patIndx,:,1] = np.nanstd(OptimalValues\_train\_clust,0) |
|  |  |  | AllSubj\_OptimalResultsClustered\_test[patIndx, :,1] = np.nanstd(OptimalValues\_test\_clust, 0) |
|  |  |  |  |
|  |  |  | #saving perofmance for all subj |
|  |  |  | meanStd=['\_mean', '\_std'] |
|  |  |  | for ni, meanStdVal in enumerate(meanStd): |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_MultiClassLearning\_OptimalResultsReduced\_Train'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubj\_OptimalResultsReduced\_train[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_MultiClassLearning\_OptimalResultsReduced\_Test'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubj\_OptimalResultsReduced\_test[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_MultiClassLearning\_OptimalResultsClustered\_Train'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubj\_OptimalResultsClustered\_train[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_MultiClassLearning\_OptimalResultsClustered\_Test'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubj\_OptimalResultsClustered\_test[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_StandardLearning\_TrainRes'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubjRes\_train[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_StandardLearning\_TestRes'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubjRes\_test[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_MultiClassLearning\_TrainRes'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubjResMulti\_train[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_MultiClassLearning\_TestRes'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubjResMulti\_test[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_MultiClassReduced\_TrainRes'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubjResMultiRed\_train[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_MultiClassReduced\_TestRes'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubjResMultiRed\_test[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_MultiClassClustered\_TrainRes'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubjResMultiClust\_train[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_MultiClassClustered\_TestRes'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubjResMultiClust\_test[:,:,ni] , delimiter=",") |
|  |  |  |  |
|  |  |  | ###################################################################################### |
|  |  |  |  |
|  |  |  | #CALCUALTING AVRG FOR ALL SUBJ (USEFUL IF THINGS RESTARTED FOR ONLY SOME SUBJECTS) |
|  |  |  | AllSubjRes\_train = np.zeros((len(GeneralParams.patients), 33, 2)) |
|  |  |  | AllSubjRes\_test = np.zeros((len(GeneralParams.patients), 33, 2)) |
|  |  |  | AllSubjResMulti\_train = np.zeros((len(GeneralParams.patients), 33, 2)) |
|  |  |  | AllSubjResMulti\_test = np.zeros((len(GeneralParams.patients),33, 2)) |
|  |  |  | AllSubjResMultiRed\_train = np.zeros((len(GeneralParams.patients), 33, 2)) |
|  |  |  | AllSubjResMultiRed\_test = np.zeros((len(GeneralParams.patients),33, 2)) |
|  |  |  | AllSubjResMultiClust\_train = np.zeros((len(GeneralParams.patients), 33, 2)) |
|  |  |  | AllSubjResMultiClust\_test = np.zeros((len(GeneralParams.patients),33, 2)) |
|  |  |  | AllSubj\_OptimalResultsRed\_train= np.zeros((len(GeneralParams.patients),34, 2)) #3+3+1+9+9+9 |
|  |  |  | AllSubj\_OptimalResultsRed\_test= np.zeros((len(GeneralParams.patients),34, 2)) |
|  |  |  | AllSubj\_OptimalResultsClust\_train= np.zeros((len(GeneralParams.patients),34, 2)) #3+3+1+9+9+9 |
|  |  |  | AllSubj\_OptimalResultsClust\_test= np.zeros((len(GeneralParams.patients),34, 2)) |
|  |  |  | for patIndx, pat in enumerate(GeneralParams.patients): |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_MultiClassLearning\_OptimalResultsReduced\_Train.csv' |
|  |  |  | reader = csv.reader(open(outputName, "r")) |
|  |  |  | OptimalValuesRed\_train = np.array(list(reader)).astype("float") |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_MultiClassLearning\_OptimalResultsReduced\_Test.csv' |
|  |  |  | reader = csv.reader(open(outputName, "r")) |
|  |  |  | OptimalValuesRed\_test = np.array(list(reader)).astype("float") |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_MultiClassLearning\_OptimalResultsClustered\_Train.csv' |
|  |  |  | reader = csv.reader(open(outputName, "r")) |
|  |  |  | OptimalValuesClust\_train = np.array(list(reader)).astype("float") |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_MultiClassLearning\_OptimalResultsClustered\_Test.csv' |
|  |  |  | reader = csv.reader(open(outputName, "r")) |
|  |  |  | OptimalValuesClust\_test = np.array(list(reader)).astype("float") |
|  |  |  |  |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_StandardLearning\_TrainRes.csv' |
|  |  |  | reader = csv.reader(open(outputName, "r")) |
|  |  |  | AllRes\_train = np.array(list(reader)).astype("float") |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_StandardLearning\_TestRes.csv' |
|  |  |  | reader = csv.reader(open(outputName, "r")) |
|  |  |  | AllRes\_test = np.array(list(reader)).astype("float") |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_MultiClassLearning\_TrainRes.csv' |
|  |  |  | reader = csv.reader(open(outputName, "r")) |
|  |  |  | AllResMulti\_train = np.array(list(reader)).astype("float") |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_MultiClassLearning\_TestRes.csv' |
|  |  |  | reader = csv.reader(open(outputName, "r")) |
|  |  |  | AllResMulti\_test = np.array(list(reader)).astype("float") |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_MultiClassReduced\_TrainRes.csv' |
|  |  |  | reader = csv.reader(open(outputName, "r")) |
|  |  |  | AllResMultiRed\_train = np.array(list(reader)).astype("float") |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_MultiClassReduced\_TestRes.csv' |
|  |  |  | reader = csv.reader(open(outputName, "r")) |
|  |  |  | AllResMultiRed\_test = np.array(list(reader)).astype("float") |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_MultiClassClustered\_TrainRes.csv' |
|  |  |  | reader = csv.reader(open(outputName, "r")) |
|  |  |  | AllResMultiClust\_train = np.array(list(reader)).astype("float") |
|  |  |  | outputName = folderOut\_ML + '/Subj' + pat + '\_MultiClassClustered\_TestRes.csv' |
|  |  |  | reader = csv.reader(open(outputName, "r")) |
|  |  |  | AllResMultiClust\_test = np.array(list(reader)).astype("float") |
|  |  |  |  |
|  |  |  |  |
|  |  |  | #save avrg for this subj |
|  |  |  | AllSubjRes\_train[patIndx,:,0] = np.nanmean(AllRes\_train,0) |
|  |  |  | AllSubjRes\_test[patIndx,:,0] = np.nanmean(AllRes\_test,0) |
|  |  |  | AllSubjResMulti\_train[patIndx,:,0] = np.nanmean(AllResMulti\_train,0) |
|  |  |  | AllSubjResMulti\_test[patIndx,:,0] = np.nanmean(AllResMulti\_test,0) |
|  |  |  | AllSubjResMultiRed\_train[patIndx,:,0] = np.nanmean(AllResMultiRed\_train,0) |
|  |  |  | AllSubjResMultiRed\_test[patIndx,:,0] = np.nanmean(AllResMultiRed\_test,0) |
|  |  |  | AllSubjResMultiClust\_train[patIndx,:,0] = np.nanmean(AllResMultiClust\_train,0) |
|  |  |  | AllSubjResMultiClust\_test[patIndx,:,0] = np.nanmean(AllResMultiClust\_test,0) |
|  |  |  | AllSubj\_OptimalResultsRed\_train[patIndx,:,0] = np.nanmean(OptimalValuesRed\_train,0) |
|  |  |  | AllSubj\_OptimalResultsRed\_test[patIndx, :,0] = np.nanmean(OptimalValuesRed\_test, 0) |
|  |  |  | AllSubj\_OptimalResultsClust\_train[patIndx,:,0] = np.nanmean(OptimalValuesClust\_train,0) |
|  |  |  | AllSubj\_OptimalResultsClust\_test[patIndx, :,0] = np.nanmean(OptimalValuesClust\_test, 0) |
|  |  |  | AllSubjRes\_train[patIndx,:,1] = np.nanstd(AllRes\_train,0) |
|  |  |  | AllSubjRes\_test[patIndx,:,1] = np.nanstd(AllRes\_test,0) |
|  |  |  | AllSubjResMulti\_train[patIndx,:,1] = np.nanstd(AllResMulti\_train,0) |
|  |  |  | AllSubjResMulti\_test[patIndx,:,1] = np.nanstd(AllResMulti\_test,0) |
|  |  |  | AllSubjResMultiRed\_train[patIndx,:,1] = np.nanstd(AllResMultiRed\_train,0) |
|  |  |  | AllSubjResMultiRed\_test[patIndx,:,1] = np.nanstd(AllResMultiRed\_test,0) |
|  |  |  | AllSubjResMultiClust\_train[patIndx,:,1] = np.nanstd(AllResMultiClust\_train,0) |
|  |  |  | AllSubjResMultiClust\_test[patIndx,:,1] = np.nanstd(AllResMultiClust\_test,0) |
|  |  |  | AllSubj\_OptimalResultsRed\_train[patIndx,:,1] = np.nanstd(OptimalValuesRed\_train,0) |
|  |  |  | AllSubj\_OptimalResultsRed\_test[patIndx, :,1] = np.nanstd(OptimalValuesRed\_test, 0) |
|  |  |  | AllSubj\_OptimalResultsClust\_train[patIndx,:,1] = np.nanstd(OptimalValuesClust\_train,0) |
|  |  |  | AllSubj\_OptimalResultsClust\_test[patIndx, :,1] = np.nanstd(OptimalValuesClust\_test, 0) |
|  |  |  |  |
|  |  |  | #saving perofmance for all subj |
|  |  |  | meanStd=['\_mean', '\_std'] |
|  |  |  | for ni, meanStdVal in enumerate(meanStd): |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_MultiClassLearning\_OptimalResultsReduced\_Train'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubj\_OptimalResultsRed\_train[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_MultiClassLearning\_OptimalResultsReduced\_Test'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubj\_OptimalResultsRed\_test[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_MultiClassLearning\_OptimalResultsClustered\_Train'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubj\_OptimalResultsClust\_train[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_MultiClassLearning\_OptimalResultsClustered\_Test'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubj\_OptimalResultsClust\_test[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_StandardLearning\_TrainRes'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubjRes\_train[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_StandardLearning\_TestRes'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubjRes\_test[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_MultiClassLearning\_TrainRes'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubjResMulti\_train[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_MultiClassLearning\_TestRes'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubjResMulti\_test[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_MultiClassReduced\_TrainRes'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubjResMultiRed\_train[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_MultiClassReduced\_TestRes'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubjResMultiRed\_test[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_MultiClassClustered\_TrainRes'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubjResMultiClust\_train[:,:,ni] , delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubj\_MultiClassClustered\_TestRes'+meanStdVal+'.csv' |
|  |  |  | np.savetxt(outputName, AllSubjResMultiClust\_test[:,:,ni] , delimiter=",") |
|  |  |  |  |
|  |  |  |  |
|  |  |  | #mean of all subj |
|  |  |  | TotalMean\_2class\_train=np.zeros((2,33)) |
|  |  |  | TotalMean\_2class\_test=np.zeros((2,33)) |
|  |  |  | TotalMean\_Multi\_train=np.zeros((2,33)) |
|  |  |  | TotalMean\_Multi\_test=np.zeros((2,33)) |
|  |  |  | TotalMean\_MultiRed\_train=np.zeros((2,33)) |
|  |  |  | TotalMean\_MultiRed\_test=np.zeros((2,33)) |
|  |  |  | TotalMean\_MultiClust\_train=np.zeros((2,33)) |
|  |  |  | TotalMean\_MultiClust\_test=np.zeros((2,33)) |
|  |  |  | TotalMean\_2class\_train[0,:] = np.nanmean(AllSubjRes\_train[:,:,0],0) |
|  |  |  | TotalMean\_2class\_test[0,:] = np.nanmean(AllSubjRes\_test[:,:,0],0) |
|  |  |  | TotalMean\_Multi\_train[0,:] = np.nanmean(AllSubjResMulti\_train[:,:,0],0) |
|  |  |  | TotalMean\_Multi\_test[0,:] = np.nanmean(AllSubjResMulti\_test[:,:,0],0) |
|  |  |  | TotalMean\_MultiRed\_train[0,:] = np.nanmean(AllSubjResMultiRed\_train[:,:,0],0) |
|  |  |  | TotalMean\_MultiRed\_test[0,:] = np.nanmean(AllSubjResMultiRed\_test[:,:,0],0) |
|  |  |  | TotalMean\_MultiClust\_train[0,:] = np.nanmean(AllSubjResMultiClust\_train[:,:,0],0) |
|  |  |  | TotalMean\_MultiClust\_test[0,:] = np.nanmean(AllSubjResMultiClust\_test[:,:,0],0) |
|  |  |  |  |
|  |  |  | TotalMean\_2class\_train[1,:] = np.nanstd(AllSubjRes\_train[:,:,0],0) |
|  |  |  | TotalMean\_2class\_test[1,:] = np.nanstd(AllSubjRes\_test[:,:,0],0) |
|  |  |  | TotalMean\_Multi\_train[1,:] = np.nanstd(AllSubjResMulti\_train[:,:,0],0) |
|  |  |  | TotalMean\_Multi\_test[1,:] = np.nanstd(AllSubjResMulti\_test[:,:,0],0) |
|  |  |  | TotalMean\_MultiRed\_train[1,:] = np.nanstd(AllSubjResMultiRed\_train[:,:,0],0) |
|  |  |  | TotalMean\_MultiRed\_test[1,:] = np.nanstd(AllSubjResMultiRed\_test[:,:,0],0) |
|  |  |  | TotalMean\_MultiClust\_test[1,:] = np.nanstd(AllSubjResMultiClust\_test[:,:,0],0) |
|  |  |  | TotalMean\_MultiClust\_train[1,:] = np.nanstd(AllSubjResMultiClust\_train[:,:,0],0) |
|  |  |  |  |
|  |  |  | outputName = folderOut\_ML + '/AllSubjAvrg\_StandardLearning\_TrainRes.csv' |
|  |  |  | np.savetxt(outputName, TotalMean\_2class\_train, delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubjAvrg\_StandardLearning\_TestRes.csv' |
|  |  |  | np.savetxt(outputName, TotalMean\_2class\_test, delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubjAvrg\_MultiClassLearning\_TrainRes.csv' |
|  |  |  | np.savetxt(outputName, TotalMean\_Multi\_train, delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubjAvrg\_MultiClassLearning\_TestRes.csv' |
|  |  |  | np.savetxt(outputName, TotalMean\_Multi\_test, delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubjAvrg\_MultiClassReduced\_TrainRes.csv' |
|  |  |  | np.savetxt(outputName, TotalMean\_MultiRed\_train, delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubjAvrg\_MultiClassReduced\_TestRes.csv' |
|  |  |  | np.savetxt(outputName, TotalMean\_MultiRed\_test, delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubjAvrg\_MultiClassClustered\_TrainRes.csv' |
|  |  |  | np.savetxt(outputName, TotalMean\_MultiClust\_train, delimiter=",") |
|  |  |  | outputName = folderOut\_ML + '/AllSubjAvrg\_MultiClassClustered\_TestRes.csv' |
|  |  |  | np.savetxt(outputName, TotalMean\_MultiClust\_test, delimiter=",") |
|  |  |  |  |
|  |  |  |  |
|  |  |  | ###################################################################################### |
|  |  |  | ###################################################################################### |
|  |  |  | ###################################################################################### |
|  |  |  | #PLOT PREDICTIONS, PERFORMANCE PER SUBJ AND MODEL |
|  |  |  | folderOutPredictionsPlots=folderOut\_ML+'/Plots\_predictions' |
|  |  |  | createFolderIfNotExists(folderOutPredictionsPlots) |
|  |  |  |  |
|  |  |  | ###################################################################################### |
|  |  |  | #PLOTS FOR THE MULTICLASS PAPER - ONE SET OF PARAMETERS (that are set at the beginnign of file) |
|  |  |  |  |
|  |  |  | # plot comparison between 2C, MC, MCred, MCclust performance for this setup |
|  |  |  | #funct\_plotPerformancesForMultiClassPaper\_SingleParamsSetup(folderOut\_ML) |
|  |  |  | dataToPlotMean\_train=np.dstack((TotalMean\_2class\_train,TotalMean\_Multi\_train, TotalMean\_MultiRed\_train, TotalMean\_MultiClust\_train)) |
|  |  |  | dataToPlotMean\_test=np.dstack((TotalMean\_2class\_test,TotalMean\_Multi\_test, TotalMean\_MultiRed\_test, TotalMean\_MultiClust\_test)) |
|  |  |  | xLabNames = ['2C', 'MC', 'MCred', 'MCclust'] |
|  |  |  | func\_plotAllPerformancesForManyApproaches(dataToPlotMean\_train, dataToPlotMean\_test, xLabNames, folderOut\_ML) |
|  |  |  |  |
|  |  |  | #plot percentage of data per subclasses |
|  |  |  | GeneralParams.patients =['01','02','03','06', '07'] #plot only for some subjects |
|  |  |  | func\_plotNumDataPerSubclasses\_forMultiClassPaper( folderOut\_ML, folderOutPredictionsPlot, GeneralParams) |
|  |  |  |  |
|  |  |  | # plotting numsbclasses and performances when itteratively removing or clustering subclasses |
|  |  |  | folderInRemov=folderOut0 +'/F1DEgmean\_0.03\_10/ItterativeRemovingSubclasses\_numSteps10' |
|  |  |  | folderInClust=folderOut0 +'/F1DEgmean\_0.03\_10/ItterativeClusteringSubclasses\_numSteps10\_PercThr0.95' |
|  |  |  | func\_plotWhenItterativelyRemovingSubclasses\_forMultiClassPaper(folderInRemov, folderInClust, folderOut\_ML, GeneralParams, numSteps) |
|  |  |  |  |
|  |  |  | ###################################################################################### |
|  |  |  | # PLOTTING COMPARISONS BETWEEN DIFFERENT FACTORS |
|  |  |  | folderPlots = '04\_PlotsForPaper/' |
|  |  |  | createFolderIfNotExists(folderPlots) |
|  |  |  |  |
|  |  |  | datasetPreparationTypeArray=['MoreNonSeizure\_Fact1', 'MoreNonSeizure\_Fact5', 'MoreNonSeizure\_Fact10'] |
|  |  |  | factNames=['Fact1', 'Fact5','Fact10'] |
|  |  |  | folderOutList= [] |
|  |  |  | for foldI, foldN in enumerate(datasetPreparationTypeArray): |
|  |  |  | folderOutList.append('03\_predictions\_' +foldN +'/'+optType+'\_'+ str(perfDropThr) +'\_'+ str(numSteps) ) |
|  |  |  | # #plot errorbars |
|  |  |  | # funct\_plotPerformancesForMultiClassPaper\_ComparisonSeveralParamsSetup(folderOutList, folderPlots) |
|  |  |  | #plot boxplot only for test smooth |
|  |  |  | funct\_plotPerformancesForMultiClassPaper\_ComparisonSeveralParamsSetup\_boxplot(folderOutList, folderPlots) |
|  |  |  |  |
|  |  |  | # plotting 6 performances of Fac1, 5, 10 for 2c, MC, MCred and MCclust |
|  |  |  | funct\_plotPerformancesForMultiClassPaper\_ComparisonSeveralParamsSetup\_graph2(folderOutList, folderPlots, factNames) |
|  |  |  |  |
|  |  |  | #plotting perf imrov and num subclasses after MCred for Fact1, 5, 10 |
|  |  |  | funct\_plotPerformancesForMultiClassPaper\_ComparisonSeveralParamsSetup\_graph3(folderOutList, folderPlots, factNames) |
|  |  |  | funct\_plotPerformancesForMultiClassPaper\_ComparisonSeveralParamsSetup\_graph3\_boxplot(folderOutList, folderPlots, factNames) |
|  |  |  |  |

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

# Load your preprocessed CSV dataset

# Replace 'your\_dataset.csv' with the actual filename

dataset\_path = '/content/drive/MyDrive/dataset/Book2.csv'

df = pd.read\_csv(dataset\_path)

# Assuming your dataset has features in columns and labels in the 'Outcome' column

label\_column = 'Outcome'

X = df.drop(label\_column, axis=1)

y = df[label\_column]

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train a RandomForestClassifier as an example model

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

# Make predictions on the training set

y\_train\_pred = model.predict(X\_train)

# Make predictions on the test set

y\_test\_pred = model.predict(X\_test)

# Evaluate training accuracy

train\_accuracy = accuracy\_score(y\_train, y\_train\_pred)

print(f'Training Accuracy: {train\_accuracy:.2f}')

# Evaluate testing accuracy

test\_accuracy = accuracy\_score(y\_test, y\_test\_pred)

print(f'Testing Accuracy: {test\_accuracy:.2f}')

# Display classification report for the test set

print('Classification Report (Test Set):')

print(classification\_report(y\_test, y\_test\_pred))

# Plot confusion matrix for the test set

cm = confusion\_matrix(y\_test, y\_test\_pred)

# Ensure cm is a 2x2 matrix

if cm.shape == (1, 1):

    cm = np.array([[0, 0], [0, 0]])

cm\_df = pd.DataFrame(cm, index=['Actual Negative', 'Actual Positive'], columns=['Predicted Negative', 'Predicted Positive'])

plt.figure(figsize=(12, 4))

# Plot confusion matrix

plt.subplot(1, 2, 1)

sns.heatmap(cm\_df, annot=True, fmt='d', cmap='Blues', annot\_kws={'size': 16})

plt.title('Confusion Matrix (Test Set)')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Plot training and testing accuracy

plt.subplot(1, 2, 2)

accuracy\_values = [train\_accuracy, test\_accuracy]

labels = ['Training Accuracy', 'Testing Accuracy']

sns.barplot(x=labels, y=accuracy\_values)

plt.ylim([0, 1])

plt.title('Training and Testing Accuracy')

plt.tight\_layout()

plt.show()