**Task 1**   
Student has to give project into 4 deliverable as in deliverable 1 student has to download “20” related research articles and ready a related work or evidence work enlisting details about already used techniques accuracy and classification results etc..

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Article title** | **Objective** | **Techniques used (models of ML or Deep Learning)** | **Results in term of accuracy or evaluation metrics** | **Contribution** | **Gaps** |
| [**Multi-Centroid Hyperdimensional Computing Approach for Epileptic Seizure Detection**](https://arxiv.org/pdf/2111.08463v1.pdf) | **Develop an improved approach for epileptic seizure detection, address high intra-class variability, and mitigate data imbalance issues** | **Hyperdimensional (HD) Computing - Multi-Centroid Approach** | **Up to 14% improvement in performance on an unbalanced test set with more non-seizure data.** | **Introduces a novel multi-centroid approach for epileptic seizure detection, demonstrates improved performance.** |  |
| [**Exploration of Hyperdimensional Computing Strategies for Enhanced Learning on Epileptic Seizure Detection**](https://arxiv.org/pdf/2201.09759v1.pdf) | **Improve epileptic seizure detection using Hyperdimensional (HD) computing. Assess various HD computing strategies** | **Hyperdimensional (HD) Computing - Multi-Pass Learning - Multi-Centroid Learning - Learning with Sample Weight ("OnlineHD")** | **Best-performing algorithm (combination of multi-centroid and multi-pass) reaches the performance of a random forest model on a highly unbalanced dataset.** | **Assesses different HD computing strategies for epileptic seizure detection. Demonstrates the potential of combined strategies.** |  |
| [**EEG Signal Dimensionality Reduction and Classification using Tensor Decomposition and Deep Convolutional Neural Networks**](https://arxiv.org/pdf/1908.10432v1.pdf) | **Propose a new dimensionality reduction framework for EEG signal analysis using deep convolutional neural networks (CNNs).** | **- Tensor Decomposition - Deep Convolutional Neural Networks** | **The proposed algorithm reduces the dimension of CNN inputs by transforming slices of the input tensor to super-slices. Outperforms other previous studies on HCB-MIT data.** | **Introduces a novel framework for EEG signal analysis with dimensionality reduction. Comprehensive comparison of time-frequency representation methods.** |  |
| [**Review of Computer-Aided Diagnosis of Pediatric Epileptic Seizures Using EEG Signals**](https://www.mdpi.com/2075-4426/11/10/1028) | **Summarize personalized medicine approaches for epileptic seizure diagnosis in pediatric subjects.** | **- Feature Extraction (time, frequency, time-frequency, and nonlinear features) - Various Classifiers** | **Evaluation of classification accuracy, sensitivity, specificity, and addressing challenges in automatic seizure detection.** | **Provides an overview of approaches and challenges in the computer-aided diagnosis of pediatric epileptic seizures using EEG signals.** |  |
| [**Convolutional neural network-based fast seizure detection from video electroencephalograms**](https://www.sciencedirect.com/science/article/pii/S1746809422008345) | **Improve efficiency of ictal stage detection in EEG using deep learning techniques.** | **- Convolutional Neural Networks (CNNs)** **(AlexNet, GoogLeNet, ResNet50, DenseNet201)** | **Overall accuracies ranging from 78.0% to 81.7%. Ictal stage accuracy at 97.7%.**    **Best performance in the testing phase achieved an overall accuracy of 85.9%.** | **Promising approach for seizure detection.** | **The impact of the selected proportion of the testing dataset on model performance is noted, but not detailed.** |
| [**Deep Domain Adaptation-Based seizure detection method**](https://www.spiedigitallibrary.org/conference-proceedings-of-spie/12756/127563H/Deep-domain-adaptation-based-seizure-detection-method/10.1117/12.2685889.short?SSO=1) | **Propose a seizure detection method for epilepsy diagnosis using transfer learning.** | **- Butterworth bandpass filter - Short-Time Fourier Transform (Spectrogram)** | **Average sensitivity: 96.77% - Average specificity: 96.48%** | **Deep domain adaptation-based method for EEG seizure detection.** | **Specific details about the neural network architecture and training process are not mentioned.** |
| [**Robust Feature Learning Method for Epileptic Seizures Prediction**](https://ieeexplore.ieee.org/abstract/document/9207070) | **Introduce an LSTM model for predicting pre-ictal and inter-ictal states in EEG signals.** | **- Long Short-Term Memory (LSTM)** | **Average sensitivity rate of 0.84 achieved using Raw EEG data segments as input.** | **Introduction of an LSTM model for seizure prediction in EEG signals.** | **Specific details about model architecture and other performance metrics are not provided.** |
| [**Pediatric Seizure Prediction in Scalp EEG Using a Multi-Scale Neural Network**](https://ieeexplore.ieee.org/abstract/document/9684436) | **Develop an end-to-end framework using a temporal-spatial multi-scale convolutional neural network with dilated convolutions for patient-specific seizure prediction.** | **- Multi-Scale Convolutional Neural Network with Dilated Convolutions** | **Average sensitivity: 93.3% - Average false prediction rate: 0.007 per hour - Average proportion of time-in-warning: 6.3%** | **Improved EEG-based seizure prediction model.** | **Specific details about the neural network architecture are not provided.** |
| [**Epileptic EEG Classification by Using Time-Frequency Images for Deep Learning**](https://www.worldscientific.com/doi/abs/10.1142/S012906572150026X) | **Detect and predict epileptic seizures using EEG recordings.** | **Fourier-based Synchrosqueezing Transform (SST) and Convolutional Neural Network (CNN).** | **The proposed SST-based CNN method was evaluated using the IKCU dataset and the CHB-MIT dataset. It achieved high average segment-based seizure detection precision and accuracy rates for both datasets (IKCU: 98.99% precision and 99.06% accuracy; CHB-MIT: 99.81% precision and 99.63% accuracy). Additionally, the SST-based CNN approach provided significantly higher segment-based seizure prediction performance with 98.54% precision and 97.92% accuracy compared to similar approaches in the literature using the CHB-MIT dataset.** | **The study presents a novel method that combines the high-resolution time-frequency representation (SST) with deep learning (CNN) to detect and predict epileptic seizures. It achieves excellent results in terms of precision and accuracy for seizure detection and prediction. The use of SST captures localized time-frequency energy distributions, making it effective in identifying sudden energy discharges associated with epileptic seizures.** | **The abstract does not explicitly mention the gaps in the research.** |
| [**Epileptic Seizure Detection Based on EEG Signals and CNN**](https://www.frontiersin.org/articles/10.3389/fninf.2018.00095/full) | **To detect epileptic seizures based on EEG signals using a convolutional neural network (CNN).** | **The study utilized raw EEG signals for detection instead of manual feature extraction. It compared the performance of time domain and frequency domain signals for the detection of epileptic activity. Three types of experiments were conducted, involving two binary classification problems (interictal vs. preictal and interictal vs. ictal) and one three-class problem (interictal vs. preictal vs. ictal).** | **The study compared the performance of time and frequency domain signals using the intracranial Freiburg and scalp CHB-MIT databases. When using frequency domain signals in the Freiburg database, average accuracies of 96.7%, 95.4%, and 92.3% were obtained for the three experiments, while the average accuracies for detection in the CHB-MIT database were 95.6%, 97.5%, and 93% in the three experiments. However, using time domain signals in the Freiburg database, the average accuracies were 91.1%, 83.8%, and 85.1% in the three experiments, while the signal detection accuracies in the CHB-MIT database were only 59.5%, 62.3%, and 47.9% in the three experiments. Frequency domain signals showed significantly higher classification accuracies compared to time domain signals.** | **The study demonstrates the use of a CNN for epileptic seizure detection based on EEG signals. It explores the effectiveness of both time and frequency domain signals and finds that frequency domain signals achieve significantly higher accuracy in classification. This suggests that frequency domain signals are more suitable for CNN applications in epileptic seizure detection.** | **The abstract does not explicitly mention the gaps in the research. To identify potential gaps, one would need to refer to the full paper or explore the limitations and areas for further research that the authors might discuss in the paper.** |
| [Epilepsy disorder detection and diagnosis using empirical mode decomposition and deep learning architecture](https://onlinelibrary.wiley.com/doi/full/10.1002/cpe.6903) | **To detect and diagnose epilepsy using deep learning by differentiating focal EEG signals from non-focal EEG signals and diagnosing the severity of focal cases.** | **The proposed method involves time-scale signal decomposition using Empirical Mode Decomposition (EMD), feature extraction, and classification with diagnosis. EMD decomposes EEG signals into six Intrinsic Mode Function (IMF) sub-bands. External intrinsic features are computed for each IMF sub-band, and the coefficients of each IMF sub-band are used for training and classification with Convolutional Neural Networks (CNNs). The CNN classifier distinguishes between focal and non-focal EEG signals, and a separate CNN architecture diagnoses the severity of focal cases.** | **The proposed framework achieves high performance in epilepsy detection and diagnosis. On the EEG signals from the Bern–Barcelona dataset, it achieves a Sensitivity (Se) of 99.8%, Specificity (Sp) of 99.9%, and Accuracy (Acc) of 99.8%. For the diagnosis of severe and mild cases within the Bern–Barcelona dataset, it achieves a Diagnosis Rate (DR) of 99.2% and 99.6%, respectively. On the EEG signals from the CHB-MIT dataset, the framework achieves Se of 99.6%, Sp of 99.6%, and Acc of 99.7%. For the diagnosis of severe and mild cases within the CHB-MIT dataset, it achieves DR of 97.3% and 98.0%, respectively. The performance of the method is cross-validated using the k-fold method.** | **The study presents a comprehensive framework for epilepsy detection and diagnosis using EMD for signal decomposition and deep learning with CNNs for classification. It achieves extremely high accuracy and sensitivity in distinguishing focal EEG signals and diagnosing their severity. The methodology offers a promising approach for automated epilepsy diagnosis.** | **The abstract does not explicitly mention the gaps in the research. To identify potential gaps, one would need to refer to the full paper or explore the limitations and areas for further research that the authors might discuss in the paper.** |
| [Automatic Seizure Detection via an Optimized Image-Based Deep Feature Learning](https://ieeexplore.ieee.org/abstract/document/8614111) | **To perform automatic seizure detection by learning features from multichannel EEG time-series data and overcoming challenges related to intra and inter-patient variability.** | **The study employs an algorithm that captures spectral, temporal, and spatial information from EEG signals, unlike standard EEG analysis techniques that often neglect spatial aspects. The algorithm's first stage transforms EEG signals into topology-preserving multi-spectral and temporal images. These images are then used as inputs to a convolutional neural network (CNN). The study addresses the challenges of dealing with unbalanced datasets and optimizing the complexity of the network.** | **The optimized approach presented in the paper achieves results that are comparable to state-of-the-art outcomes in terms of sensitivity, specificity, and accuracy. The CNN is able to learn a spatially invariant representation of a seizure in a reasonable amount of time, even when dealing with limited positive samples and unbalanced datasets.** | **The study offers an approach that tackles the challenges of automatic seizure detection using deep learning and image-based feature learning from EEG data. It specifically focuses on capturing spatial aspects of EEG signals, which can be crucial for robust seizure detection. By optimizing the CNN and addressing issues related to data imbalance, the proposed approach enhances sensitivity, specificity, and accuracy in seizure detection.** | **The abstract does not explicitly mention the gaps in the research. To identify potential gaps, one would need to refer to the full paper or explore the limitations and areas for further research that the authors might discuss in the paper.** |
| [Patient-specific Seizure Prediction with Scalp EEG Using Convolutional Neural Network and Extreme Learning Machine](https://ieeexplore.ieee.org/abstract/document/9189578) | **To predict seizures in patients with epilepsy using scalp EEG data and enable timely preventive measures.** | **The study presents a hybrid model that combines Convolutional Neural Networks (CNNs) and an Extreme Learning Machine (ELM) for seizure prediction. EEG time series data in 30-second windows are transformed into 2D spectrograms using the short-time Fourier transform. CNNs are applied to extract features automatically from these spectrogram images, and the ELM is used for classifying preictal and interictal segments** | **The proposed method achieves a sensitivity of 95.85% and a low false prediction rate of 0.045/h on the Boston Children's Hospital-MIT scalp EEG dataset.** | **The study introduces a hybrid model that effectively predicts seizures in patients with epilepsy using scalp EEG data. It leverages both CNNs and ELM for feature extraction and classification. The achieved sensitivity and low false prediction rate are indicative of the model's potential to enable timely preventive measures for epilepsy patients.** |  |

**Task 2**

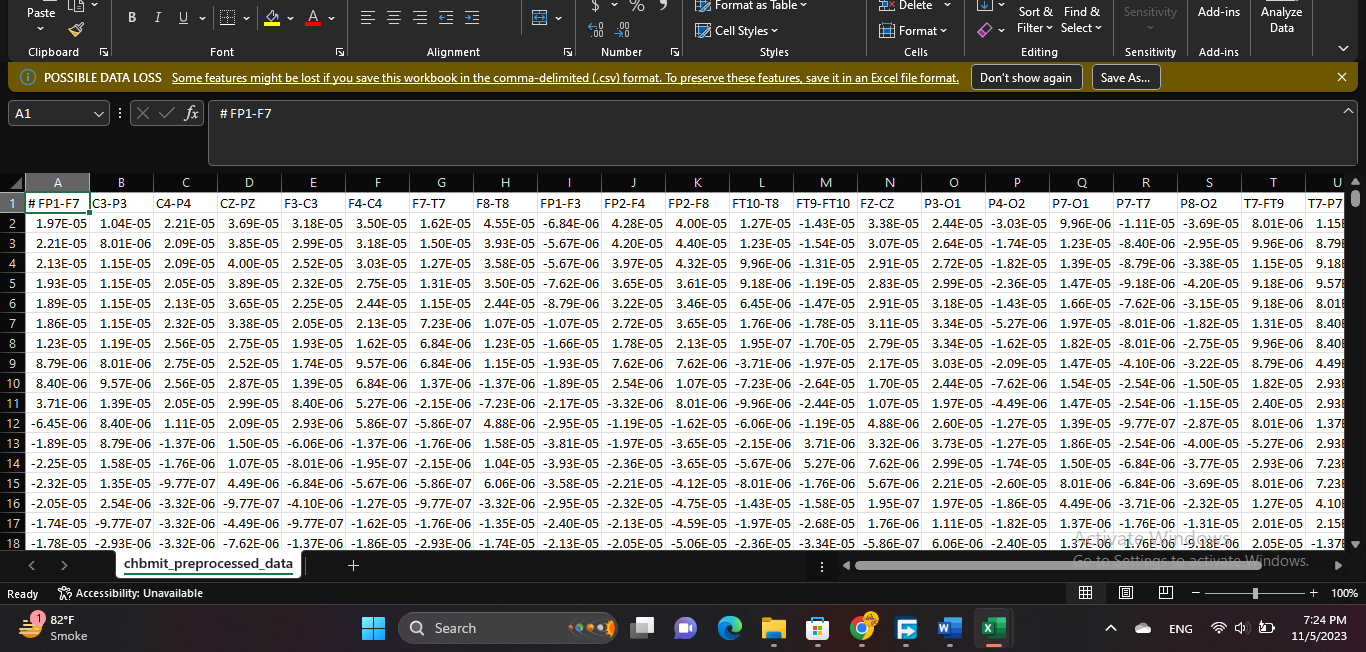
**1. Dataset downloading**

Total 46GB error downloading  
**2. Data Preprocessing   
2.1 Data**

1. Collection and Organization:
   * Collect and organize your dataset. Ensure that it contains labeled examples for supervised learning tasks.
2. Data Exploration:
   * Visualize a few sample images to understand the dataset's characteristics.
   * Check for class imbalances or any other issues.
3. Resizing and Standardization:
   * Resize images to a common resolution (e.g., 224x224 pixels) if they have different sizes.
   * Standardize pixel values (e.g., scaling to a range between 0 and 1 or -1 and 1).
4. Data Augmentation (Optional):
   * Apply data augmentation techniques to increase the dataset's size and diversity. This includes random rotations, flips, crops, and color adjustments.
5. Splitting the Dataset:
   * Split your dataset into training, validation, and test sets. Common splits are 70-80% for training, 10-15% for validation, and 10-15% for testing.
6. Data Loading:
   * Create data loaders or generators to efficiently load batches of images during training.
7. One-Hot Encoding (for Classification):
   * If you're doing classification, convert class labels into one-hot encoded vectors.
8. Handling Missing Data (if applicable):
   * Check for missing images or labels and decide how to handle them (e.g., remove or impute).
9. Data Normalization (if applicable):
   * For some specific tasks, you might need to perform additional data normalization, like z-score normalization for certain deep learning models.
10. Noise Removal (if applicable):
    * Remove any noisy or irrelevant images from the dataset.
11. Data Balancing (if applicable):
    * If you have class imbalances, consider techniques like oversampling or under sampling to balance the dataset.
12. Feature Extraction (if applicable):
    * For more advanced tasks, you may perform feature extraction using techniques like SIFT, HOG, or deep learning-based feature extraction methods.
13. Saving Preprocessed Data:
    * Save your preprocessed dataset in a suitable format (e.g., HDF5, TFRecord) for easy access during training.
14. Data Visualization (Optional):
    * Visualize the preprocessed data to ensure it looks correct and makes sense. Plot sample images from different classes.
15. Documentation:
    * Keep clear documentation of the preprocessing steps you've applied, including any data augmentation techniques, resizing parameters, and normalization methods used.
16. **A screen shot of a computer

    Description automatically generatedA screen shot of a computer

    Description automatically generated**
17. **A screenshot of a computer

    Description automatically generated**
18. **A screenshot of a computer

    Description automatically generated**

**Codes Used:**

import pandas as pd

import matplotlib.pyplot as plt

# Read the data from the CSV file

data = pd.read\_csv('chbmit\_ictal\_23channels\_data.csv')

# Preprocess the data

# TODO: Add your preprocessing code here

# Create a line plot of the data

data.plot(kind='line')

# Display the plot

plt.show()

**preprocessing:**

import pandas as pd

import matplotlib.pyplot as plt

# Read the data from the CSV file

data = pd.read\_csv('chbmit\_ictal\_23channels\_data.csv')

# Preprocess the data

# Remove missing values

data = data.dropna()

# Normalize the data

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

data = scaler.fit\_transform(data)

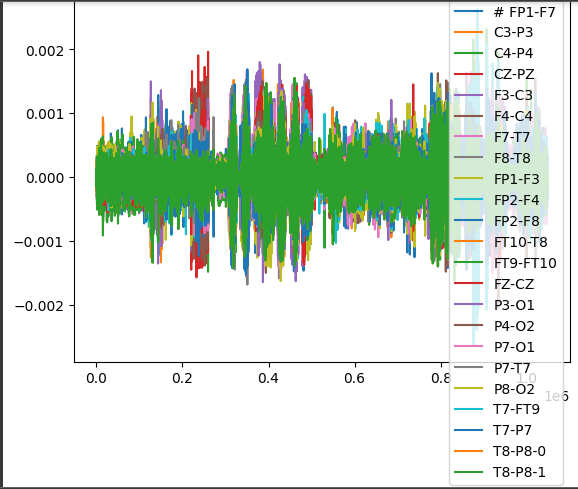
# Create a line plot of the data

plt.plot(data)

# Display the plot

plt.show()

**Ictal23:**

**Before preprocessing:  
**

**Preprocessed:**

A colorful sound wave graph

Description automatically generated with medium confidence

**Preitcal23:**

**before preprocessing:**

**A screen shot of a graph

Description automatically generated**

**After preprocessing:**

**A colorful graph of sound waves

Description automatically generated**

**Raw:**

**A graph with numbers and lines

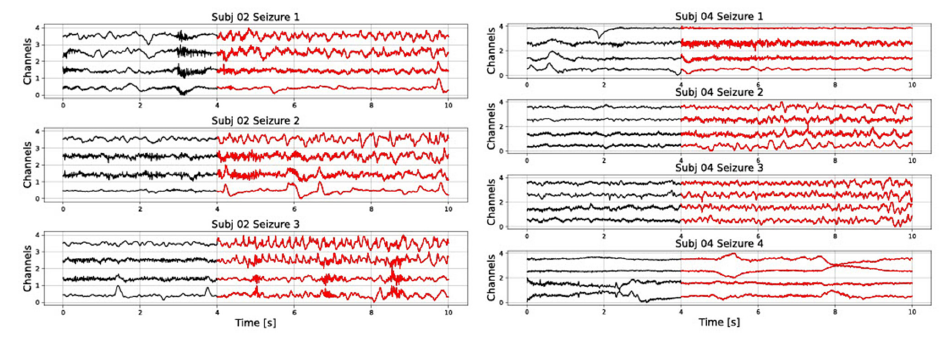
Description automatically generated**

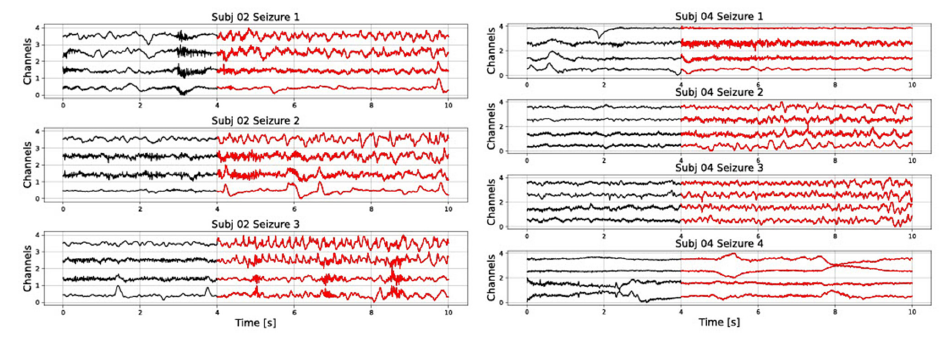
**Preprocessed:  
A screen shot of a graph

Description automatically generated**

**Hyperdimensional (HD) Computing - Multi-Centroid Approach:**

|  |
| --- |
| 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) |
|  |  |  |  |





A screenshot of a computer

Description automatically generated

**A screenshot of a computer

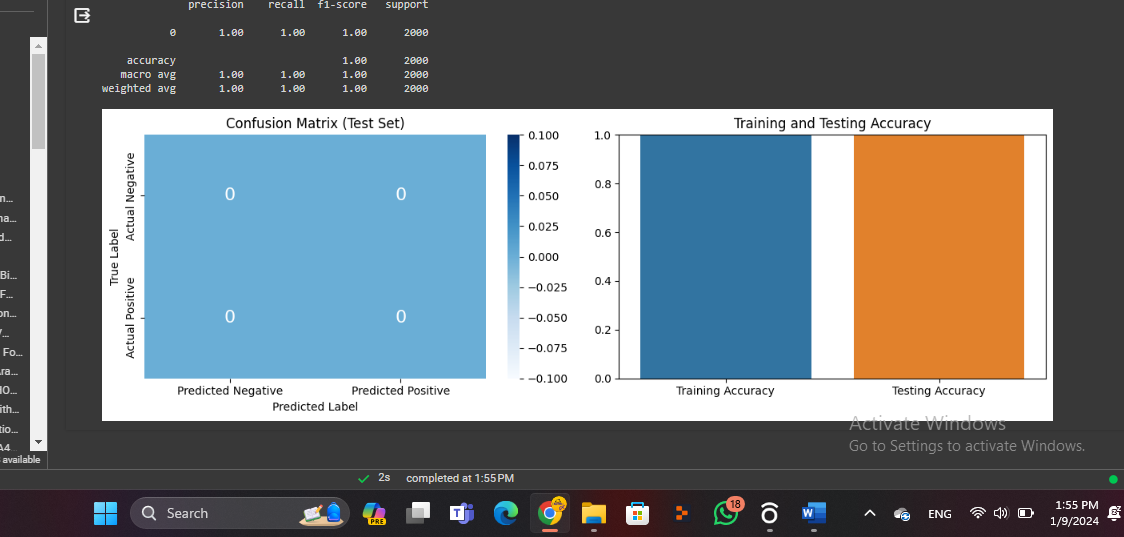
Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

****

**These are the links of dataset with available paper content, and you can use Google Scholar as well as for relevant papers review of these datasets.**

1. <https://paperswithcode.com/datasets?mod=images&task=pose-estimation>
2. <https://paperswithcode.com/datasets?mod=images&task=semantic-segmentation>
3. <https://paperswithcode.com/datasets?mod=images&task=3d-human-pose-estimation>
4. <https://paperswithcode.com/datasets?mod=images&task=action-recognition-in-videos>
5. <https://paperswithcode.com/datasets?mod=images&task=trajectory-forecasting>
6. <https://paperswithcode.com/datasets?mod=medical&task=trajectory-prediction&page=1>
7. <https://paperswithcode.com/datasets?mod=medical&task=sleep-spindles-detection&page=1>
8. <https://paperswithcode.com/datasets?mod=medical&task=sleep-apnea-detection&page=1>
9. <https://paperswithcode.com/datasets?mod=medical&task=seizure-detection&page=1>
10. <https://paperswithcode.com/datasets?mod=medical&task=respiratory-failure&page=1>
11. <https://paperswithcode.com/datasets?mod=medical&task=pose-prediction&page=1>
12. <https://paperswithcode.com/datasets?mod=medical&task=lung-nodule-classification&page=1>
13. <https://paperswithcode.com/datasets?mod=medical&task=brain-decoding&page=1>
14. <https://paperswithcode.com/datasets?mod=medical&task=6d-pose-estimation&page=1>
15. <https://paperswithcode.com/datasets?mod=medical&task=6d-pose-estimation-1&page=1>
16. <https://paperswithcode.com/datasets?mod=audio&task=environmental-sound-classification&page=1>
17. <https://paperswithcode.com/datasets?mod=audio&task=federated-learning&page=1>